MULTI-RESOLUTION ANALYSIS USING WAVELET BASIS CONDITIONED ON HOMOGENIZATION

by

Abibat Adebisi Lasisi

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

DOCTOR OF PHILOSOPHY

in

Mathematical Sciences

Approved:

Joseph V. Koebbe, Ph.D. Major Professor	James S. Cangelosi, Ph.D. Committee Member
Nghiem Nguyen, Ph.D. Committee Member	Luis Gordillo, Ph.D. Committee Member
Todd Moon, Ph.D. Committee Member	Laurens H. Smith, Ph.D. Interim Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2018

ABSTRACT

Multi-Resolution Analysis Using Wavelet Basis Conditioned on Homogenization

by

Abibat Adebisi Lasisi, Doctor of Philosophy

Utah State University, 2018

Major Professor: Joseph V. Koebbe, Ph.D.
Department: Mathematics and Statistics

Wavelets and homogenization methods have been used in the development of algorithms for the approximate solution of differential equations. In this dissertation, we propose a homogenization wavelet reconstruction algorithm for computing the solution of elliptic partial differential equations. The proposed algorithm is based on a wavelet characterization of homogenization methods in multi-resolution analysis. We employ orthogonal decomposition of the problem into two scale problems on nested dyadic grids using wavelet multi-resolution analysis. The unique aspect of this dissertation is the combination of homogenization theory with wavelet multi-resolution analysis to provide solutions to elliptic differential equations. To illustrate the proposed methodology, we first deal with the problem of the one dimensional case. The problem of fluid flow in a porous medium with conductivity/permeability depending on the spatial variable provides an example in the one dimensional situation. It is well known in one dimension that if an average value of the conductivity/permeability is computed, it must be equal to the harmonic average of the function representing the fine scale parameter values. Our fast transform algorithm also preserves the harmonic average of the conductivity/permeability. Furthermore, we extend the proposed methodology to the two dimensional case using homogenization theory. We developed a fast transform algorithm in two dimensions that gives exactly the same results

as the solutions to the local problems in two dimensions with diagonal tensors. We also developed Java codes that compute the solution of the elliptic problems in two dimensions, the results are the same as those computed by hand. Finally, we implement Java codes for our fast transform and the inverse transform which also give results that are consistent with the analytical solutions.

(210 pages)

PUBLIC ABSTRACT

Multi-Resolution Analysis Using Wavelet Basis Conditioned on Homogenization

Abibat Adebisi Lasisi

This dissertation considers an approximation strategy using a wavelet reconstruction scheme for solving elliptic problems. The foci of the work are on (1) the approximate solution of differential equations using multiresolution analysis based on wavelet transforms and (2) the homogenization process for solving one and two-dimensional problems, to understand the solutions of second order elliptic problems. We employed homogenization to compute the average formula for permeability in a porous medium. The structure of the associated multiresolution analysis allows for the reconstruction of the approximate solution of the primary variable in the elliptic equation. Using a one-dimensional wavelet reconstruction algorithm proposed in this work, we are able to numerically compute the approximations of the pressure variables. This algorithm can directly be applied to elliptic problems with discontinuous coefficients.We also implemented Java codes to solve the two dimensional elliptic problems using our methods of solutions. Furthermore, we propose homogenization wavelet reconstruction algorithm, fast transform and the inverse transform algorithms that use the results from the solutions of the local problems and the partial derivatives of the pressure variables to reconstruct the solutions.

ACKNOWLEDGMENTS

My gratitude goes to my major professor, Dr. Joseph V. Koebbe. Thank you for your advice, feedback, comments, guidance, and for leading me through how to do research in multiscale analysis using wavelets. I also want to thank other members of my PhD dissertation committee, Dr. James S. Cangelosi, Dr. Nghiem Nguyen, Dr. Luis Gordillo, and Dr. Todd Moon, for your comments, feedback, suggestions, and questions.

I would like to thank Dr. James Cangelosi for your invaluable support since the beginning of my master's program up till now. Thanks for all your support, advice, and assistance for me and my family.

My profound gratitude also goes to my sweetheart, Dr. Ramoni Lasisi, and my kids for their support and assistance throughout the course of my studies. Finally, I want to express my gratitude to my mom, siblings, and in-laws. I pray to Allah to continue to shower his blessings on my late dad, mother-in-law and father-in-law (Ameen).

Abibat.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 Overview

A vast number of methods have been proposed for the approximate solution of elliptic *partial differential equations* (PDEs). Such methods as finite difference methods, finite volume methods, finite element methods, [1–7] have been applied to determine approximate solutions at discrete points in domains under consideration. In recent years, researchers have become interested in the development of multiscale methods [1, 2, 8–10]. These methods are used to determine a way to approximate solutions of a given PDE on a coarse scale, while retaining small scale behaviors in the approximation. Methods like homogenization and multi-resolution analysis have been employed in a number of ways to compute coarse scale parameter values from fine scale measurements or realizations for given parameters. It would certainly be better to approximately solve a problem on the finest mesh possible. However, computer resources become an obstacle in most complicated problems. The desire is to use upscaling methods like those mentioned above to produce coarse scale parameters on a discrete mesh or grid that can be managed in computer simulations.

For this work, we consider elliptic *differential equations* (DEs) with coefficients that depend on the spatial variables in the problem. To illustrate the methodology we start with the one dimensional case. The problem of flow in a porous medium with conductivity depending on the spatial variable provides an example of this situation. It is well known that if an average value of the conductivity is computed it must be equal to the harmonic average of the function representing the fine scale parameter values. Note that homogenization methods described in the next section in fact, preserve the harmonic average. The method introduced in this work was built on the idea of creating a fast transform method that preserves the harmonic average of the conductivity in one dimension. This represents the

forward transform algorithm. An algorithm for computing the harmonic average has been developed that uses a recursive formula on a dyadic mesh in the spatial domain.

Additionally we are interested in approximating the solutions of DEs arising from many applications in sciences and engineering that exhibit a number of different scales. This can be done in a fashion similar to the inverse transform algorithm. There are many applications involving solutions that vary over different scales. Some examples are composite materials and flows in porous media. The solutions to these problems may be impossible to find since fine scale grids are needed to resolve important behaviors in the solutions [5, 11, 12]. The results in the DEs which describe the physical phenomenon that occur at different length and time scales might be difficult or complicated to analyze. Thus, the need to represent the problem using simpler models. One such simple modeling tool that can be employed is *homogenization*. The main goal of homogenization is to represent complex, rapidly varying media with slowly varying media in which the fine scale structures are averaged out appropriately.

Homogenization problems are grouped into *periodic,* where $K(x+p) = K(x)$ for all x, and *aperiodic* problems. Periodicity means that the coefficients of a DE that model a physical phenomenon are repeated at regular intervals. In the case of aperiodic (for instance, a random permeability or porosity) the coefficients are not periodic [13]. We are concerned in this dissertation with problems with either aperiodic or periodic structures which are more realistic and find usefulness in several real-life applications, e.g., composite materials or porous media.

We consider in this work, elliptic problems of the form shown in Equation 1.1, in one dimension to illustrate the usefulness of our proposed methods. Let Ω be a periodic bounded domain and $\partial\Omega$ smooth on the boundary.

$$\frac{d}{dx}K(x)\frac{dh}{dx} = f(x) \qquad x \in \Omega \tag{1.1}$$

$$h = g \qquad on \quad \partial\Omega$$

This DE can be used to model fluid flow in porous media, where $K(x)$ is the permeability or conductivity tensor, h is the pressure variable, and $f(x)$ represents a forcing function. In practice, we need to solve this type of equation where $K(x)$ exhibits rapidly changing behavior on multiple scales for a small but fixed parameter, ϵ. In homogenization $K(x)$ is assumed to vary on two disparate scales, a microscopic scale, ϵ, $0 < \epsilon << 1$, and a macroscopic scale that captures behavior on a scale that is $O(1)$. Instead of solving the original problem, we may have to solve the homogenized version of the problem which is computationally simpler to solve. Some techniques for solving these problems are based on the assumption that the coefficients are periodic on the fine scale. However, that may not be the case in many applications. In fact we will assume that the elliptic coefficient tensor values are obtained by sampling $K(x)$ (e.g. permeability) at some given number of equally spaced locations in the domain of interest. This will be useful in the development of fast transform methods.

1.2 Literature Review

Analytical and numerical methods for solving partial differential equations are widely studied [1–6], and have been applied to solving various problems in science and engineering. We review the following notable works relevant to the problem considered in this dissertation. Babuska and Osborn [12] consider a generalized finite element method (FEM) for problems with rough coefficients in a simple one dimensional case. The concept of a generalized FEM includes practical FEM using different test and trial functions - methods in which the shape function is governed by differential equations. Their method offers a larger freedom in the computational procedures than standard FEMs. The method also offers the possibility of significant improvement in accuracy when used in conjunction with adaptive procedures. Furthermore, they show that their method reacts well when the roughness of a coefficient is reduced and that changing from measurable coefficients to coefficients with bounded variation improves the rate of convergence.

New methods have recently been developed to solve second order elliptic problems with heterogeneous and highly varying coefficients. These methods were developed to over-

come performance issues of classical FEMs when the diffusion coefficient has discontinuities and/or high variation. One such method, referred to as the *discontinuous Galerkin multiscale method* (DGMsM) for solving second order elliptic problem is the focus of the work of Elfverson et al. [7] and Beatrice [14]. Discontinuous Galerkin (DG) methods approximate solutions of partial differential equations in finite dimensional spaces spanned by piecewise polynomial basis functions. DG methods resemble classical FEMs with the exception of explicitly imposing continuity constraints at inter element interfaces, i.e., they impose weak continuity on numerical solutions without explicit constraints on the approximation space. This results in the inclusion of jump terms across interfaces. Some penalty terms must also be added to control jump terms in the weak formulation of a problem.

Babuska, Caloz, and Osborn [11] consider a class of second order, two dimensional elliptic problems with rough or highly oscillating coefficients. They present several methods called special FEMs that were applied to unidirectional composite materials. Their methods use special shape functions that are chosen to accurately model unknown solutions for this class of problems. They also show that their methods have the same accuracy as the usual FEMs for problems with smooth coefficients.

In the work of Hou [8], a multiscale finite element method (MsFEM) for solving a class of elliptic problems generated from composite materials and flows in porous media which contain many spatial scales were considered. The method efficiently captures the large scale behavior of the solution without resolving all the small scale behaviors. They were able to accomplish this with the construction of multiscale finite element basis functions that adapt to the local properties of the differential operator. The formation of the basis functions is fully decoupled from element to element. This makes the method perfectly parallel and adapted to massively parallel computers, thus having the ability to handle large degrees of freedom found in highly heterogeneous media.

Arbogast et al. [1] and Arbogast, Tao, and Xiao [2] developed multiscale mortar mixed FEMs for second order elliptic problems. Their methods impose continuity of flux via a mortar finite element space on a coarse grid scale, with the equations in coarse elements

discretized on a fine grid scale. Their methods achieve approximations that are comparable to the fine scale on their coarse grids by using higher order polynomials. Furthermore, they derived apriori error estimates, obtained optimal order convergence, and some superconvergence [1] on the fine scale for both the solution and its flux. Finally, they derived a posteriori error estimators that were used in an adaptive mesh refinement algorithm to obtain appropriate subdomain and mortar grids.

Karakashian [9] and Larson [10] propose a variational multiscale method and a discontinuous Galerkin formulation based on a posteriori error estimates which relate the error in an energy norm to the discretization errors. Becker et al. [15] presented a residual based on a posteriori error estimate of a natural mesh dependence on an energy norm of the error in a family as DG approximations of elliptic problems. Dryja [16] analyzed the error bound of DGMs, and as well designed and analyzed a multilevel additive Schwarz preconditioner for one of the discrete problems. Dryja found that the preconditioner is not optimal but more suited for parallel computations. Finally, the author concluded that the rate of convergence of this method is also independent of the jump in the coefficients.

Wavelet-based numerical homogenization is well studied [13, 17–19]. Mihai and Bjorn [13] considered a numerical homogenization process for elliptic differential equations that are based on wavelet decompositions of discrete operators in fine and coarse scale components followed by the removal of the fine scale contributions. Per-Olof and Olof [19] applied wavelet-based numerical homogenization to the simulation of an optical waveguide filter. They derived a one dimensional model and subgrid models of the filter, and then presented numerical examples and computational payoffs of their techniques.

In the work of Alina and Doron [18], a wavelet-based method was applied for the systematic derivation of subgrid scale models in the solution of PDEs. They represented the discrete operator in a wavelet space and projected the fine scales onto a coarser subspace. They did some modifications to improve the efficiency of the numerical homogenization method by choosing a different compact representation of the homogenized operator. Furthermore,

[1] Superconvergence involves finding points where a finite element solution in more accurate than at location where basis functions are defined. Superconvergence occurs due to the fact that FEM approximation oscillate around the exact solution.

they proposed a natural fine scale correction which they implemented at the final step in their homogenization process. Brewster and Beylkin [17] presented a multi-resolution analysis based on homogenization of differential equations. They consider a system of linear *ordinary differential equations* (ODEs) with variable coefficients and forcing terms. They also developed an efficient algorithm from the method of lines discretization of PDEs and perform homogenization over the variable time scales.

1.3 Main Contributions

Until now, a careful combination of the homogenization theory with wavelet multiresolution analysis to provide solutions to elliptic DE of the form introduced in equations 1.1 and 4.1 is yet to be researched. We provide a summary of our main results as follow:

- We perform homogenization process on two-cell problems in a multiscale analysis. Instead of using a perturbation parameter, $0 < \epsilon << 1$ where ϵ tends to zero, we set $\epsilon = \frac{1}{2}$ and use more terms in the perturbation series.

- We propose Homogenization Wavelet Reconstruction (HWR) algorithm for the approximate solution of elliptic differential equations. The approximations are applied to problems where coefficients vary rapidly.

- We employ orthogonal decomposition using a Haar multiresolution analysis conditioned on homogenization. An analogy between homogenization and the MRA is described in one and two spatial dimensions.

- We develop a fast transform method in one and two dimensions for computing the homogenized value of elliptic coefficients at all scales on a given dyadic mesh. The transform method honors the "correct" average value predicted by homogenization theory.

- We develop the associated inverse fast transform method using simple algebraic steps. The inverse transform is used to aid in the reconstruction algorithm central to this dissertation.

- A novel aspect of the research in this dissertation involves optimizing the performance of the fast transform algorithm by investigating differences in the solution of local problems at successive scales. The resulting algorithm avoids the computation of derivatives in the original version of the HWR algorithm.

- We show that the two dimensional extension of the HWR algorithm where the elliptic coefficient is a diagonal tensor amounts to one dimensional problems on dyadic meshes. The full tensor case is also resolved. The full tensor case requires a more complicated solution process, but can be done.

- We implement Java codes that compute the pressure variables and came up with a close form generalization of the differencing formula in our solution in one dimension.

- Finally, we implement Java codes that compute the fast and inverse transform algorithms to verify our results.

1.4 Organization of the Dissertation

The remainder of this dissertation is organized as follows. Chapter 2 presents preliminaries to provide necessary background in multi-scale methods, including homogenization and wavelets analysis. In Chapter 3, we develop fast transforms for homogenization, and give an analogy between the method of homogenization and wavelets in one dimension and consider reconstruction of solutions of elliptic differential equations using multi-resolution analysis conditioned on homogenization. In chapter 4 we consider two-dimensional homogenization wavelet reconstruction algorithm with diagonal tensors. Chapter 5 discusses two-dimensional homogenization wavelet reconstruction algorithm with full tensors and presents numerical results. We conclude in Chapter 6 and as well gives directions for future research.

CHAPTER 2

MULTI-SCALE METHODS

2.1 Description of Multi-scale Methods

Multi-scale methods are employed in solving problems that have important features at multiple time and/or space scales. There are many fundamental and practical problems involving a wide range of length scales. Examples include highly heterogeneous porous media and composite materials with fine micro-structures [20–26]. Systems evolving on widely separated time scales present significant challenges both theoretically and numerically. Standard computational schemes fail due to the wide separation between the rapidly varying time/spatial scale in the system one must compute with and the slowest time scales one is typically interested in analyzing. Thus, it is important to treat such problems effectively. Multi-scale methods carry fine-scale information throughout simulation, and the coarse scale equations are generally not expressed analytically, but rather formed and solved numerically. The discussion in this chapter provides descriptions of two concepts: (a) *Multiresolution Analysis (MRA)* and (b) *Homogenization* for solving multi-scale problems that we consider in this dissertation.

2.2 Multiresolution Analysis

A wavelet is a wave-like feature that travels for one or more periods and is nonzero only over a finite interval. Wavelets are building blocks that are designed to model sound signals with isolated noisy pops that need to be filtered. A wavelet can be translated forward or backwards in time, and stretched or compressed by scaling to represent low- and high frequency, this concept is called *translation invariance*. Wavelets are useful in analyzing data, and are used to remove noise without smoothing out the main features of the data which

makes it more effective for data cleaning [27]. Wavelets were first applied in Geophysics to analyze data from seismic surveys [28].

As a comparison, the *Fourier transform* is a well-known and useful tool for analyzing components of signals. However, the main disadvantage of the Fourier expansion is that it has only frequency resolution with no time resolution. To overcome this problem in the past decades, several solutions have been developed that were able to represent a signal in the time and frequency domain at the same time. *Wavelet transforms* are one such solution to the problem which can keep track of both the time and frequency information [28, 29]. Wavelets have advantages over traditional Fourier transform in analyzing physical situations where signals contain discontinuities, sharp spikes, and signals with compact support.

The *scaling function*, ϕ, and the *wavelet* function, ψ, are two basic functions that play important roles in wavelet analysis. ϕ and ψ are used to generate orthogonal basis functions that are needed to *decompose* (i.e., break up) or *reconstruct* (i.e., combine) signals. The Haar scaling function is defined as

$$\phi(x) = \begin{cases} 1 & if \quad 0 \leq x < 1, \\ 0 & elsewhere. \end{cases}$$

The *Haar* wavelet can be defined as a linear combination of scaling functions as follows.

$$\psi(x) = \phi(2x) - \phi(2x-1)$$

It is considered the simplest of all the wavelets. The Haar wavelet scheme depends on the Haar scaling function and the Haar wavelet which are simple to describe and lead to easy decomposition algorithms. Haar wavelets are well localized in the time/space domains but have the disadvantage of not being continuous. Thus, they do not efficiently approximate continuous signals. Note that in this dissertation, the elliptic coefficient tensor is assumed to be discontinuous.

To provide solutions for this disadvantage of *Haar* wavelets, Stephane Mallat in 1998 proposed a theory called *multiresolution analysis*, MRA. MRA is the process of analyzing signals at different scales with different resolutions. Let the approximation spaces, V_j for $j = \ldots -2, -1, 0, 1, 2, \ldots$ define a sequence of subspaces of functions in $L^2(R)$. The collection of spaces $V_j, j \in Z$ is called a multiresolution analysis with scaling function ϕ if the following conditions hold.

- Nested: The approximation space, V_j is a subset of V_{j+1}
- Density: $\overline{\cup V_j} = L^2(R)$. That is the closure of the union of V_j, $\overline{\cup V_j}$ is defined as $f \in \overline{\cup V_j}$ if and only of for every $\epsilon > 0$ one can find j such that there is an $f_j \in V_j$ for which $\| f - f_j \| < \epsilon$,. This property means that every $f \in L^2$ can be approximated as closely as one likes by a function in a V_j, provided that j is large enough.
- Separation: $\cap V_j = 0$. The intersection of the approximation spaces is contains only zero function.
- Scaling: The function $f(x)$ belongs to V_j if and only if the function $f(2^{-j}x)$ belongs to V_0.
- Orthonormal Basis: The function ϕ belongs to V_0 and the set $\{\phi(x-k), k \in Z\}$ is an orthonormal basis (using the L^2 inner product) for V_0.

We note some examples of MRA in this section, see [28] for more detail. *Linear Splines* are continuous and piecewise linear functions which have infinite support. They decay rapidly at infinity. *Shannon wavelets* are very smooth, they extend throughout the whole real line and they decay slowly at infinity. *Daubechies wavelets* are continuous but they are not differentiable. These wavelets are usually characterized by the number of vanishing moments. The smoothness of the scaling and wavelet function increases with the number of vanishing moments.

2.3 A Brief Review of Homogenization

It is common in science and engineering to deal with problems formed from multiple components. Solving a mathematical problem with rapidly varying coefficients in the structure can be difficult, even numerically. It is therefore ideal to find simpler equations that will effectively smooth the coarse structures of the constituents that may arise with spatially heterogeneous materials [30, 31].

Homogenization deals with derivation of equations for averaging of solutions of equations with rapidly varying coefficients. In a homogenization process, we begin with a problem that includes structural variations, and derive a simpler problem that serves as a first-order approximation of the original problem. The method of homogenization is a type of multiscale analysis. In a multidimensional case, it might be very difficult to find the correct secularity condition. Thus many researchers assume that the substructure is periodic. In this dissertation the substructure need not be periodic.

2.3.1 Standard Homogenization Applied to Elliptic Differential Equation

We consider the following elliptic differential equation (DE) (i.e., Equation 2.1) in one dimension to model the flow of fluid in porous media, where $K(x)$ is the permeability or conductivity tensor that is periodic on the domain, h is the pressure variable, and $f(x)$ is some forcing function:

$$\frac{d}{dx}K(x)\frac{dh}{dx} = f(x). \tag{2.1}$$

We employ a perturbation analysis that produces a system of equations at various scales based on power series representation in terms of a small parameter,$0 < \epsilon << 1$. Using homogenization methods, we assume that the primary variables can be expanded in a perturbation series in this parameter. The homogenized solution converges to a weak solution [32] of the original problem as the parameter ϵ tends to zero.

First we define the associated first order system:

$$v(x) = -K(x)\frac{dh}{dx} \tag{2.2}$$

$$-\frac{dv}{dx} = f \tag{2.3}$$

where $v(x)$ can be thought of as the fluid velocity or Darcy velocity.
In a standard two-scale homogenization, we define

$$y = \frac{x}{\epsilon}$$

and expand h and v as

$$h(x,y) = h_0(x,y) + \epsilon h_1(x,y) + \epsilon^2 h_2(x,y) + \cdots \tag{2.4}$$

$$v(x,y) = v_0(x,y) + \epsilon v_1(x,y) + \epsilon^2 v_2(x,y) + \cdots \tag{2.5}$$

The dependence of h and v on x and y will be suppressed to simplify notation in the sequel. The differential operator $\frac{d}{dx}$ becomes

$$\frac{d}{dx} = \frac{d}{dx} + \frac{1}{\epsilon}\frac{d}{dy} + \frac{1}{\epsilon^2}\frac{d}{dz} + \cdots \tag{2.6}$$

For a standard presentation of mathematical homogenization, the differential operator is restricted to

$$\frac{d}{dx} = \frac{d}{dy_0} + \frac{1}{\epsilon}\frac{d}{dy_1}. \tag{2.7}$$

Next, substitute these definitions into the first order system of DEs for v and h. The result is,

$$\begin{aligned} v &= v_0 + \epsilon v_1 + \epsilon^2 v_2 + \cdots = -K\left(\frac{d}{dy_0} + \frac{1}{\epsilon}\frac{d}{dy_1}\right)(h_0 + \epsilon h_1 + \epsilon^2 h_2 + \cdots) \\ &- \left(\frac{d}{dy_0} + \frac{1}{\epsilon}\frac{d}{dy_1}\right)(v_0 + \epsilon v_1 + \epsilon^2 v_2 + \cdots) = f. \end{aligned}$$

The next step is to compare terms involving like powers of ϵ. In this process, we will neglect all the terms that are multiplied by a power of ϵ greater than zero. We are interested in the first two sets of equations for ϵ^n, $n = -1, 0$. This is because the functions h_i and $v_i, i = 0, 1, 2, \cdots$ are assumed to be bounded. Also, note that in standard homogenization, the limit as ϵ tends to zero is used. Finally, We make an assumption that allows the equation to be balanced on both sides.

For ϵ^{-1}

$$0 = -K\frac{dh_0}{dy_1} \tag{2.8}$$

$$0 = -\frac{dv_0}{dy_1}. \tag{2.9}$$

For ϵ^0

$$v_0 = -K\left(\frac{dh_0}{dy_0} + \frac{dh_1}{dy_1}\right) \tag{2.10}$$

$$f_0 = -\left(\frac{dv_0}{dy_0} + \frac{dv_1}{dy_1}\right). \tag{2.11}$$

For ϵ^1

$$v_1 = -K\left(\frac{dh_1}{dy_0} + \frac{dh_2}{dy_1}\right) \tag{2.12}$$

$$f_1 = -\left(\frac{dv_1}{dy_0} + \frac{dv_2}{dy_1}\right). \tag{2.13}$$

Note that the last equation 2.12 is not needed in a standard application of homogenization. From equation 2.8 and 2.9, we found out that since K is not zero, then $\frac{dh_0}{dy_1} = 0$ and as a result, h_0 is a function of y_0. Also, $-\frac{dv_0}{dy_1} = 0$ implies that v_0 is a function of y_0. To finish the analysis, it is assumed that

$$h_1 = w_0(y_1)\frac{dh_0}{dy_0}.$$

From equation 2.10, we have:

$$
\begin{aligned}
v_0 &= -K\left(\frac{dh_0}{dy_0}+\frac{dh_1}{dy_1}\right) && (2.14)\\
&= -K\left(\frac{dh_0}{dy_0}+\frac{d}{dy_1}(w_0(y_1)\frac{dh_0}{dy_0})\right) && (2.15)\\
&= -K\left(\frac{dh_0}{dy_0}+\frac{dw_0(y_1)}{dy_1}\frac{dh_0}{dy_0}\right) && (2.16)\\
&= -K\left(1+\frac{dw_0(y_1)}{dy_1}\right)\frac{dh_0}{dy_0}. && (2.17)
\end{aligned}
$$

Differentiating both sides of equation 2.17 with respect to y_1, we obtain:

$$
\frac{dv_0}{dy_1} = -\frac{d}{dy_1}K\left(1+\frac{dw_0(y_1)}{dy_1}\right)\frac{dh_0}{dy_0} \tag{2.18}
$$

or

$$
0 = -\frac{d}{dy_1}K\left(1+\frac{dw_0(y_1)}{dy_1}\right)\frac{dh_0}{dy_0}. \tag{2.19}
$$

Since v_0 is a function of y_0, equation 2.19 defines the local problem for $w_0(y_1)$.

2.3.2 Direct Computation of the Average Coefficient

Using equation 2.10, we can compute the average velocity v_0, by integrating both sides of the equation. So, integrating the left hand side

$$
\begin{aligned}
\int_0^1 v_0(y_0)dy_1 &= v_0(y_0)\int_0^1 dy_1 && (2.20)\\
&= v_0(y_0)(1) && (2.21)\\
&= \bar{v_0}. && (2.22)
\end{aligned}
$$

The right side of 2.10 can be integrated as follows

$$\begin{aligned}\int_0^1 K(y_1)\left(\frac{dh_0}{dy_0}+\frac{dh_1}{dy_1}\right) &= -\int_0^1 K(y_1)\left(\frac{dh_0}{dy_0}+\frac{d}{dy_1}\left(w_0(y_1)\frac{dh_0}{dy_0}\right)\right)dy_1\\ &= -\int_0^1 K(y_1)\left(1+\frac{d}{dy_1}\left(w_0(y_1)\right)\right)\frac{dh_0}{dy_0}dy_1\\ &= -\left(\int_0^1 K(y_1)\left(1+\frac{d}{dy_1}\left(w_0(y_1)\right)\right)dy_1\right)\frac{dh_0}{dy_0}.\end{aligned}$$

Since $h_0 = h_0(y_0)$ and $\frac{d}{dy_1}h_0 = 0$, so h_0 is a constant with respect to y_1.

So, if we set

$$K^\sharp = \int_0^1 K(y_1)\left(1+\frac{dw_0}{dy_1}\right)dy_1$$

Then the averaged equation is

$$\bar{v_0} = -K^\sharp\frac{dh_0}{dy_0}$$

where $\bar{v_0}$ = average velocity. This works provided the integral can be computed on a microscopic cell.

Note: For the work in this dissertation, the averaging will be performed recursively over multiple scales. In one-dimension, the theory behind homogenization guarantees $K^\sharp$ is the harmonic average.

2.3.3 The Homogenized Average Coefficient in a Two-Cell Problem

A formula for computing the homogenized parameter value, $K^\sharp$, which agrees with the harmonic average is given below. Suppose that the coefficient, K is defined as follows:

$$K = \begin{cases} K_0 & y_1 \in \left[0, \frac{1}{2}\right) \\ K_1 & y_1 \in \left[\frac{1}{2}, 1\right] \end{cases}$$

then

$$K^{\sharp} = \int_0^1 K\left(1+\frac{dw_0}{dy_1}\right)dy_1 \tag{2.23}$$

$$= \int_0^1 K dy_1 + \int_0^1 K\left(\frac{dw_0}{dy_1}\right)dy_1 \tag{2.24}$$

$$= \frac{2K_0K_1}{K_0+K_1} \tag{2.25}$$

where

$$\bar{K} = \int_0^1 K dy_1$$

is the arithmetic average, and

$$\Delta K = \int_0^1 K\left(\frac{dw_0}{dy_1}\right)dy_1$$

is a perturbation needed to produce the harmonic average. The homogenization method produces the harmonic average of the signals by summing these two values.

The function $w_0(y_1)$ is the solution of the associated local problem with appropriate boundary conditions, and it satisfies the local problem in a weak sense [32] since we have assumed that the permeability is discontinuous. The local problem is defined by the equation

$$\frac{d}{dy_1}K(y_1)\frac{d}{dy_1}w_0(y_1) = -\frac{d}{dy_1}K(y_1) \quad for \quad 0 < y_1 < 1 \tag{2.26}$$

with

$$w_0(1) = w_0(0) = 0$$

2.4 Equivalence of Wavelet Analysis and Two-Cell Homogenization

The analogy between the method of homogenization and wavelets can be seen in the computation of the harmonic average i.e., the averaging formula for computing the permeability coefficients 2.25. Homogenization methods give the harmonic average of the permeability field and the wavelet characterization lets us compute the harmonic average using

a *wavelet transform*. The wavelet transform decomposes a function into a weighted sum of its various space/frequency components.

In the one dimensional case, the homogenized coefficient is the harmonic average defined by

$$K^{\sharp} = \frac{2K_0K_1}{K_0 + K_1}$$

The average formula from the homogenization process is given by 2.23 and 2.24:

From equation 2.23 w_0 satisfies the ordinary differential equation as stated in 2.26. Note that $\int_0^1 K dy_1$ is the arithmetic average of K and $\int_0^1 K \frac{dw_0}{dy_1} dy_1$ is the perturbation needed to produce harmonic average. The detail in the wavelet transform is exactly the perturbation in the homogenization formula as seen below. To determine w_0, we define the scaled local problem:

$$\frac{d}{dy_1} K \frac{d}{dy_1} w_0 = -\frac{d}{dy_1} K$$

with

$$K = \begin{cases} K_0 & y_1 \in \left[0, \frac{1}{2}\right) \\ K_1 & y_1 \in \left[\frac{1}{2}, 1\right] \end{cases}$$

where K is the elliptic coefficient. With this definition of the permeability, we find a weak solution of $w_0(y_1)$ as:

$$w_0(y_1) = \frac{K_1 - K_0}{K_1 + K_0} \begin{cases} 2y_1, & y_1 \in \left[0, \frac{1}{2}\right) \\ 2 - 2y_1, & y_1 \in \left[\frac{1}{2}, 1\right] \end{cases} \tag{2.27}$$

In the development of the fast transform for computing the average and reconstruction of the solution, the coefficient of $w_0(y_1)$ in 2.27 is the same as the wavelet coefficient needed in the reconstruction.

With this solution of the average formula, we have the following:

$$\begin{aligned}
K^\sharp &= \int_0^1 K\left(1+\frac{dw_0}{dy_1}\right)dy_1 \\
&= \int_0^1 K(1+w_{0,0}\psi(y_1))dy_1 \\
&= \int_0^1 K dy_1 + w_{0,0}\int_0^1 K\psi(y_1))dy_1 \\
&= \int_0^{\frac{1}{2}} K_0 dy_1 + \int_{\frac{1}{2}}^1 K_1 dy_1 + w_{0,0}\left(\int_0^{\frac{1}{2}} K_0 dy_1 + \int_{\frac{1}{2}}^1 K_1(-1)dy_1\right) \\
&= \frac{1}{2}K_0 + \frac{1}{2}K_1 + w_{0,0}\left(\frac{1}{2}K_0 - \frac{1}{2}K_1\right) \\
&= \frac{1}{2}(K_0+K_1) + \frac{w_{0,0}}{2}(K_0-K_1) \\
&= \frac{2K_0K_1}{K_0+K_1}.
\end{aligned}$$

Thus, we can conclude that there is a clear analogy between wavelet transform and the homogenization process for computing the permeability coefficient.

CHAPTER 3

HOMOGENIZATION WAVELET RECONSTRUCTION IN ONE DIMENSION

3.1 A One Dimensional Fast Transform for Homogenized Coefficient Values

There are large number of methods that have been developed for the approximate solution of elliptic DEs [8, 13, 33, 34]. In this section, the first step in the Homogenization-Wavelet-Reconstruction (HWR) algorithm is presented. The first step in the HWR method defines a fast transform method that preserves the harmonic average of the coefficient, $K(x)$.

3.1.1 A Fast Transform Method that Preserves the Harmonic Average

The correct averaged/upscaled value of $K(x)$ in one dimension is the harmonic average. That is, if two values are given, say K_0 and K_1, then

$$K^\sharp = \frac{2K_0\,K_1}{K_0 + K_1}$$

or if $K(x)$ is a continuous function of x

$$K^\sharp = \left(\int \frac{1}{K(x)} d(x) \right)^{-1}.$$

A more typical setting is to be given a sequence of values, $\left\{K_{1,0}, K_{1,1}, K_{1,2}, \ldots, K_{1,2^{m-1}}\right\}$ with the idea of computing an average value of all samples written as:

$$K^\sharp = \left(\sum_{j=0}^{2^m} \frac{1}{K_{i,j}} \right)^{-1}.$$

If these methods are used, all intermediate scale information will be lost. The result of these computations is a single homogenized value, $K^\sharp$, for the entire microscopic scale.

Yet, another method for computing this average is to define a transform method that will define averages at all intermediate scales. Suppose we are given the discrete sequence as above. We consider using pairs of values to compute local averages. For example, using two values from a sequence of samples, $K_{m,0}$ and $K_{m,1}$, we can define

$$K^\sharp = \frac{2K_{m,0}K_{m,1}}{K_{m,0} + K_{m,1}}.$$

To recast the averaging process in a form that can be used in a fast transform algorithm, the simple computation could be rewritten in the following three steps.

$$\begin{aligned} \bar{K} &= \frac{(K_{m,0} + K_{m,1})}{2}; && (3.1)\\ \Delta K &= -\frac{(K_{m,1} - K_{m,0})^2}{2(K_{m,1} + K_{m,0})}; && (3.2)\\ K^\sharp &= \overline{K} + \triangle K = \frac{2K_{m,0}\,K_{m,1}}{K_{m,0} + K_{m,1}}. && (3.3) \end{aligned}$$

Now, we consider the sequence

$$K_m = \left\{K_{m,0}, K_{m,1}, K_{m,2}, \ldots, K_{m,2^{m-1}}\right\}$$

and apply the process above to pairs of the sequence elements as follows:

$$\begin{aligned} \overline{K}_{m-1,j} &= \frac{(K_{m,2j+1} + K_{m,2j})}{2}; && (3.4)\\ \Delta K_{m-1,j} &= -\frac{(K_{m,2j+1} - K_{m,2j})^2}{2(K_{m,2j+1} + K_{m,2j})}; && (3.5)\\ K^\sharp &= \overline{K}_{m-1,j} + \Delta K_{m-1,j} = \frac{2K_{m,2j}\,K_{m,2j+1}}{K_{m,2j} + K_{m,2j+1}}. && (3.6) \end{aligned}$$

The algorithm for the fast transform is given as follows. (See also Figures 3.1 and 3.2 for illustrations of visual descriptions of the procedure).

Fast Transform Algorithm

- *Assume that the permeability tensor $K_{m,j}$ represents samples of $K(x)$ at 2^m equally spaced points in each spatial dimension.*
- *Compute $\overline{K}$, the arithmetic average of the two neighboring values of permeability tensor as in 3.4.*
- *Compute $\triangle K$, the detail involving the difference of the two neighboring values of the permeability tensor as in 3.5.*
- *Then, add the arithmetic average and detail to obtain $K^\sharp$, the harmonic average of the original signals 3.6.*

The output from this algorithm is a sequence arithmetic averages and the details needed to compute $K^\sharp$. Now, suppose we apply this idea to a sequence as defined above. We can use pairs of samples to define two cell problems where the harmonic average is computed and the detail is also retained. In this way, the details are available for later use.

The pairwise average values form a sequence of averages with 2^{m-1} values and a sequence of details, also of length 2^{m-1}. For example, if we have a sequence of 2^m values and their details, we use the pairwise samples to compute the harmonic average and their details, this will reduce the sequence to a sequence of 2^{m-1}. Recursively applying the algorithm will produce 2^2, then 2^1 averages and the details. The end result is the harmonic average of the original values and the details associated with all the dyadic scales.

The transform method not only computes the correct values at coarse scales but also stores the details at all scales. Computationally, the details can be stored using the original array by overwriting half the values at each level. The details in the signals allow a fast method for the computation of the average of the elliptic coefficient at all scales. The details can be used to reconstruct the original signal or to reconstruct the approximate pressure variable for the original DE. Keeping the details along with the average values at all scales provides the data necessary for a fast inverse transform. The inverse transform will also be

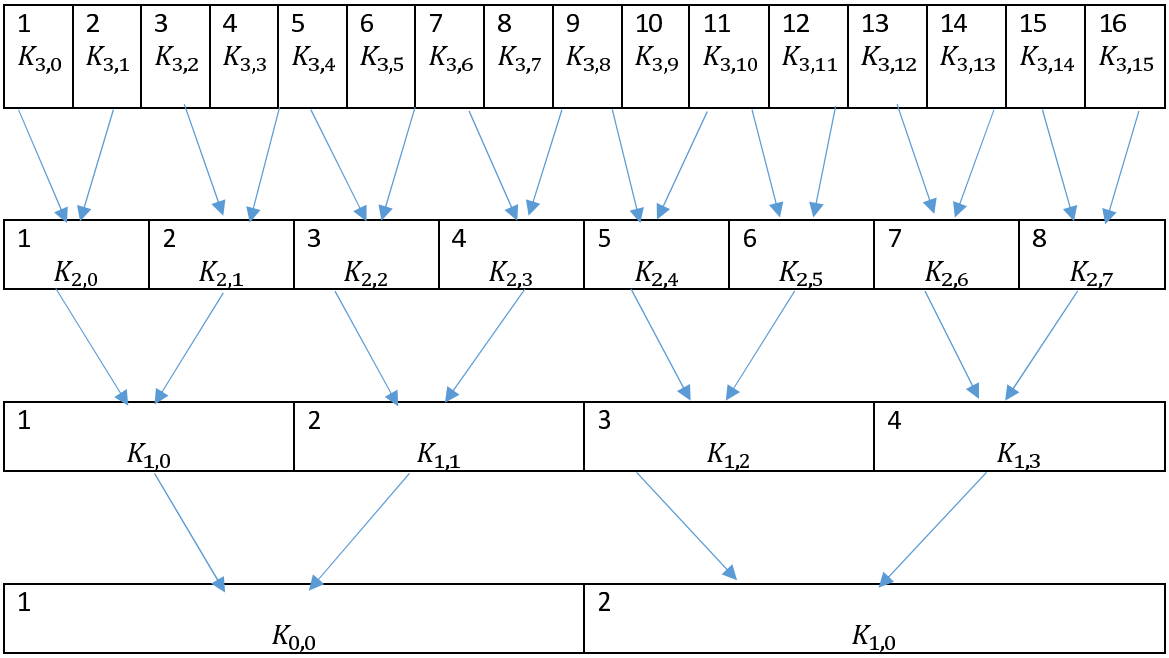

Figure 3.1. Illustration of visual description of the procedure

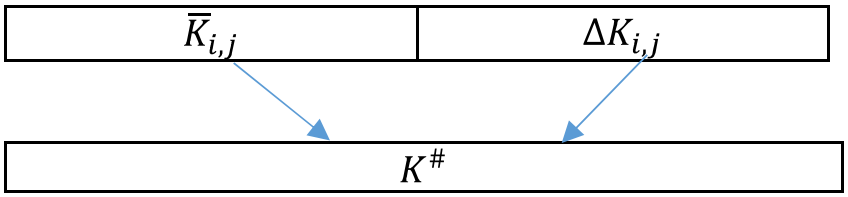

Figure 3.2. Illustration of description of computation of the harmonic average

used to aid in the reconstruction of the primary variable, h, in the original elliptic differential equation.

3.1.2 The Analogy between Homogenization and the Fast Transform Method

The analogy between the method of homogenization and wavelets can be seen in the fast transform method presented in the last section that preserves the harmonic average, i.e., the averaging formula for computing the permeability coefficients. Homogenization methods give the harmonic average of the permeability and a wavelet characterization that can be used to compute the harmonic average using a wavelet transform.

The average formula from homogenization given by:

$$K^{\sharp} = \int_0^1 K\left(1+\frac{dw_0}{dy_1}\right)dy_1 \tag{3.7}$$

$$= \int_0^1 K dy_1 + \int_0^1 K\frac{dw_0}{dy_1}dy_1 \tag{3.8}$$

will be used.

From Equation 3.8,

$$\overline{K} = \int_0^1 K dy_1$$

is the arithmetic average, and

$$\triangle K = \int_0^1 K\frac{dw_0}{dy_1}dy_1$$

is the perturbation needed to produce harmonic average.

3.1.3 Generation of Wavelets Conditioned on Homogenization

The fast transform method motivates the development of a multi-resolution analysis using homogenization to condition the wavelet basis.

For the local problem

$$\frac{d}{dy_1}K(y_1)\frac{d}{dy_1}w_0(y_1) = -\frac{d}{dy_1}K(y_1),$$

with

$$K = \begin{cases} K_0 & y_1 \in \left[0,\frac{1}{2}\right) \\ K_1 & y_1 \in \left[\frac{1}{2},1\right], \end{cases}$$

$$w_0(y_1) = \left(\frac{K_1-K_0}{K_0+K_1}\right)\begin{cases} y_1, & y_1 \in \left[0,\frac{1}{2}\right) \\ 1-y_1, & y_1 \in \left[\frac{1}{2},1\right]. \end{cases}$$

The weak derivative of $w_0(y_1)$ is given by

$$\frac{dw_0}{dy_1} = \left(\frac{K_1 - K_0}{K_0 + K_1}\right) \begin{cases} 1, & y_1 \in \left[0, \frac{1}{2}\right) \\ -1, & y_1 \in \left[\frac{1}{2}, 1\right] \end{cases}.$$

In addition, if we define

$$w_{0,0} = \frac{K_1 - K_0}{K_1 + K_0}$$

and

$$\psi(y_1) = \begin{cases} 1, & y_1 \in \left[0, \frac{1}{2}\right) \\ -1, & y_1 \in \left[\frac{1}{2}, 1\right] \end{cases}$$

then

$$\frac{dw_0}{dy_1} = w_{0,0}\psi(y_1)$$

The weak derivative of $w_0(y_1)$ is a constant multiple $w_{0,0}$ of a Haar wavelet, $\psi(y_1)$. Alternatively, one can think of the constant multiplier, $w_{0,0}$, as being conditioned by the solution of the local problem obtained in the homogenization process.

We can rewrite Equation 3.7 as follow

$$\begin{aligned} K^\sharp &= \int_0^1 K\left(1 + \frac{dw_0}{dy_1}\right) dy_1 \\ &= \int_0^1 K(1 + w_{0,0}\psi(y_1))dy_1 \\ &= \int_0^1 K dy_1 + w_{0,0}\int_0^1 K\psi(y_1)dy_1 \end{aligned}$$

where, $K = K\chi_{[0,1]}$ = scaling function and ψ = wavelet function.

With this solution the second integral in the average formula becomes:

$$\int_0^1 K\frac{dw_0}{dy_1}dy_1 = \Delta K = -\frac{(K_{i,j+1} - k_{i,j})^2}{2(K_{i,j+1} + k_{i,j})}. \tag{3.9}$$

The detail in the wavelet transform integrates to exactly the perturbation in the homogenization formula as seen in Equation 3.9. Thus, it is clear that there is a relationship between the wavelet transform that preserves the harmonic average and the homogenization process for computing the permeability coefficient.

3.1.4 The Inverse Transform

The inverse transform can be obtained by reversing the order of the steps given above as long as the details are kept at all scales. The algorithms are as follows:

The Inverse Transform Algorithm

- *Given* $K^\sharp = \bar{K} + \Delta K$*, where* $K^\sharp$ *is the harmonic average of the original signals,* $\bar{K}$ *is the arithmetic average of* K*, and* $\Delta K = -\frac{(K_{m,1}-K_{m,0})^2}{2(K_{m,1}+K_{m,0})} = -\frac{(K_{m,1}-\bar{K})^2}{\bar{K}}$ *is the details needed to compute the average.*
- *Compute* $\bar{K} = K^\sharp - \Delta K$
- *Since* $\bar{K} = \frac{(K_{m,0}+K_{m,1})}{2}$*, then we have* $2\bar{K} = K_{m,0} + K_{m,1}$.

$$
\begin{aligned}
K^\sharp &= \bar{K} + \Delta K, \\
\Delta K &= -\frac{(K_{m,1} - K_{m,0})^2}{2(K_{m,1} + K_{m,0})} = -\frac{(K_{m,1} - \bar{K})^2}{\bar{K}}, \\
\bar{K} &= K^\sharp - \Delta K, \\
K_{m,1} &= (-\Delta K \cdot \bar{K})^{\frac{1}{2}} + \bar{K}, \\
2\bar{K} &= K_{m,0} + K_{m,1}, \\
K_{m,0} &= 2\bar{K} - K_{m,1}.
\end{aligned}
$$

The inverse transform results can be recursively apply to homogenization wavelet reconstruction algorithm to compute successively finer scale values from the coarse scale values.

3.1.5 Reconstruction of Solutions of Elliptic Differential Equations Using Multiresolution Analysis Conditioned on Homogenization

We propose the one-dimensional *Homogenization-Wavelet Reconstruction* (HWR) algorithm in this section. Knowing the signal at successive fine scales, we can actually compute the wavelet coefficients needed to move from coarser scales to finer scales using the inverse of the wavelet transform. Thus, a reconstruction can be done at desired scale as long as the fine scales information are available.

The reconstruction formula is based on the fundamental assumption that must be made in the homogenization process. The homogenization process assumes that the primary variables h and v from the original problem can be expanded in a perturbation series of the form:

$$h = h_0 + \epsilon h_1 + \epsilon^2 h_2 + \cdots + \epsilon^m h_m + \ldots$$

where ϵ is a small parameter, $0<\epsilon<<1$. In this process, we make the assumption

$$h_1 = w_0 \frac{dh_0}{dy_0}.$$

The next step is to assume that the same type of relationship occurs between successive dyadic scales in the equation 3.12. Using this assumption we can write:

$$\begin{aligned}
h &\approx h_0, \\
h &\approx h_0 + \epsilon h_1 = h_0 + \epsilon w_0 \frac{dh_0}{dy_0}, \\
&\vdots \\
h &\approx h_0 + \epsilon h_1 + \epsilon^2 h_2 + \cdots + \epsilon^m h_m = h_0 + \epsilon w_0 \frac{dh_0}{dy_0} + \epsilon^2 w_1 \frac{dh_1}{dy_1} + \cdots + \epsilon^m w_{m-1} \frac{dh_{m-1}}{dy_{m-1}}.
\end{aligned}$$

Once we have determined the value of h_i, then we can use this to compute the next term in the perturbation expansion by computing the derivative of h_i with respect to y_1. So, to reconstruct the pressure variable, h, we consider the following algorithm referred to

as Homogenization-Wavelet Reconstruction algorithm. The algorithm is as follow:

Homogenization Wavelet Reconstruction Algorithm

- *Given the finest scale permeability samples $K_{m,j}$, on a finest scale,where m refers to the finite levels and the number of samples is 2^m, and j refers to the jth sample at the given level. Compute the details at all scales.*

- *Compute the solution of the DE for the coarsest level homogenized problem on the entire domain, that is compute the solution of*

$$v = -K\nabla h, \tag{3.10}$$

$$\frac{d}{dx}K_{0,0}\frac{dh}{dx} = f, \tag{3.11}$$

where $K_{0,0} = K^{\sharp}$ is the harmonic average of the permeability.

- *Then use*

$$h = h_0 + \epsilon h_1 + \epsilon^2 h_2 + \cdots + \epsilon^m h_m, \tag{3.12}$$

$$v = v_0 + \epsilon v_1 + \epsilon^2 v_2 + \cdots + \epsilon^m v_m, \tag{3.13}$$

and the ansatz

$$h_1 = w_0 \frac{dh_0}{dy_0}.$$

In our work we use an extension of this ansatz of the form

$$h_{l+1} = w_l(y_{l+1})\frac{dh_l}{dy_l}.$$

This means

$$h = h_0 + \epsilon w_0 \frac{dh_0}{dy_0} + \epsilon^2 w_1 \frac{dh_1}{dy_1} + \cdots + \epsilon^m w_{m-1}\frac{dh_{m-1}}{dy_{m-1}}$$

with $\epsilon = \frac{1}{2}$.

3.2 Recursive Differencing of Analytic Solutions

One novel aspect of the work in this dissertation is described in this section. In the development of the Fast Fourier Transform (FFT) over the past decades, many research papers have been written about improving the speed/efficiency of the FFT. The work in this section is analogous in the following sense. The brute force transform first proposed [35] was not efficient relative to the wavelet transform. By investigating the difference in analytic solution between dyadic scales, a more efficient algorithm for the fast wavelet transform has been created. We use another approach to compute the solution of the problem, using the information about the boundary conditions and the wavelet coefficients (α's and β's) where $\beta = \frac{K_{i,j}}{K_{i,j-1}+K_{i,j}}$ and $\alpha = \frac{K_{i,j-1}}{K_{i,j-1}+K_{i,j}}$. This process is a sort of a brute force verification of the HWR method. The process is used to derive solutions to the zero-scale, two-cell, and four-cell local elliptic problems. We compute the differences of the solutions and provide a Java implementation of the algorithm that can generate recursive formulas of the differences for values of $n = 1, 2, \cdots$.

Given the elliptic equation, we are required to compute the solution to the problem. We derived the solutions to one-scale, two-scale, and three-scale elliptic problems in terms of α and β using the properties of wavelets multi-resolution analysis (MRA).

3.3 Computing Differences of Solutions of n-Cell Problem

In this section, we compute the differences in the analytical solution of the local $2-$ cell problems. In addition, simpler formula are obtained for the HWR algorithm.

3.3.1 Zeroth Scale Analytic Solution

We provide a step-by-step derivation of the solution of the coarsest scale problem

$$-\frac{d}{dx}K(x)\frac{dh}{dx} = 0, \tag{3.14}$$

with the boundary conditions:

$$\begin{aligned} h(0) &= B_L, \\ h\left(\frac{1}{2}\right) &= B_R. \end{aligned}$$

Integrate equation (3.14) with $K(x) = K_{0,0}$. We obtain

$$\begin{aligned} -K_{0,0}\frac{dh}{dx} &= c_{0,0}, \\ \frac{dh}{dx} &= -\frac{c_{0,0}}{K_{0,0}}. \end{aligned}$$

Integrating the above equation, we have

$$h(x) = -\frac{c_{0,0}}{K_{0,0}}x + b_{0,0}, \tag{3.15}$$

$$h(x) = -\frac{c_{0,0}}{K_{0,0}}x + B_L. \tag{3.16}$$

Next, we compute the value of $c_{0,0}$ given that,

$$\begin{aligned} h(0) &= B_L = b_{0,0}, \\ h\left(\frac{1}{2}\right) &= B_R = -\frac{c_{0,0}}{K_{0,0}}(\frac{1}{2}) + B_L, \end{aligned}$$

we obtain

$$c_{0,0} = 2(B_R - B_L)K_{0,0}.$$

Putting the value of $c_{0,0}$ back into equation (3.16) , we have

$$\begin{aligned} h(x) &= \frac{2K_{0,0}(B_R - B_L)}{K_{0,0}}x + B_L, \\ &= 2(B_R - B_L)x + B_L. \\ h_0(y_0) &= 2(B_R - B_L)y_0 + B_L. \end{aligned}$$

3.3.2 Two-Cell Analytic Solution

Given an elliptic equation, we are required to compute the solution to the problem:

$$-\frac{d}{dx}K(x)\frac{dh}{dx} = 0, \tag{3.17}$$

with the boundary conditions:

$$\begin{aligned} h(0) &= B_L, \\ h(1) &= B_R, \end{aligned}$$

where,

$$K(x) = \begin{cases} K_{1,0}, & x \in \left[0, \frac{1}{2}\right), \\ K_{1,1}, & x \in \left[\frac{1}{2}, 1\right]. \end{cases}$$

Integrating the given equation, we obtain

$$-K(x)\frac{dh}{dx} = \begin{cases} c_{1,0}, & x \in \left[0, \frac{1}{2}\right), \\ c_{1,1}, & x \in \left[\frac{1}{2}, 1\right]. \end{cases}$$

Dividing both sides by $-K(x)$, we have

$$\frac{dh}{dx} = -\begin{cases} \frac{c_{1,0}}{K_{1,0}}, & x \in \left[0, \frac{1}{2}\right), \\ \frac{c_{1,1}}{K_{1,1}}, & x \in \left[\frac{1}{2}, 1\right]. \end{cases}$$

Integrate this with respect to x, we have

$$h(x) = -\begin{cases} \frac{c_{1,0}}{K_{1,0}}x + b_{1,0}, & x \in \left[0, \frac{1}{2}\right), \\ \frac{c_{1,1}}{K_{1,1}}(x-1) + b_{1,1}, & x \in \left[\frac{1}{2}, 1\right]. \end{cases}$$

Again, the boundary conditions are:

$$h(0) = B_L = b_{1,0},$$
$$h(1) = B_R = b_{1,1}.$$

Assuming continuity of $K\nabla h$ at $x = \frac{1}{2}$, so $c_{1,0} = c_{0,0}$,

$$\begin{aligned} \frac{c_{1,0}}{K_{1,0}}x + B_L &= \frac{c_{1,0}}{K_{1,1}}(x-1) + B_R, \\ \frac{c_{1,0}}{K_{1,0}}\left(\frac{1}{2}\right) + B_L &= \frac{c_{1,0}}{K_{1,1}}\left(-\frac{1}{2}\right) + B_R. \end{aligned}$$

We next solve for the value of $c_{1,0}$ as follows:

$$\begin{aligned} \frac{c_{1,0}}{2K_{1,0}} + \frac{c_{1,0}}{2K_{1,1}} &= B_R - B_L, \\ c_{1,0}\left(\frac{K_{1,1}+K_{1,0}}{2K_{1,0}K_{1,1}}\right) &= B_R - B_L, \\ c_{1,0} &= \left(\frac{2K_{1,0}K_{1,1}}{K_{1,1}+K_{1,0}}\right)(B_R - B_L), \\ c_{1,0} &= K_{0,0}^{\sharp}(B_R - B_L). \end{aligned}$$

Substituting the value of $c_{1,0}$ into h we obtain:

$$h_1(y_1) = \begin{cases} \dfrac{K_{0,0}^{\sharp}}{K_{1,0}}(B_R - B_L)(y_1) + B_L, & y_1 \in \left[0, \dfrac{1}{2}\right) \\ \dfrac{K_{0,0}^{\sharp}}{K_{1,1}}(B_R - B_L)(y_1 - 1) + B_R, & y_1 \in \left[\dfrac{1}{2}, 1\right]. \end{cases}$$

Simplifying the above equation results in:

$$\begin{aligned} \frac{K_{0,0}^{\sharp}}{K_{1,0}} &= \frac{\frac{2K_{1,0}K_{1,1}}{K_{1,1}+K_{1,0}}}{K_{1,0}} \\ &= \frac{2K_{1,1}}{K_{1,0}+K_{1,1}} \\ &= 2\beta_{0,0}, \end{aligned}$$

$$\begin{aligned}\frac{K_{0,0}^{\sharp}}{K_{1,1}} &= \frac{\frac{2K_{1,0}K_{1,1}}{K_{1,1}+K_{1,0}}}{K_{1,1}} \\ &= \frac{2K_{1,0}}{K_{1,0}+K_{1,1}} \\ &= 2\alpha_{0,0},\end{aligned}$$

where $\alpha_{0,0} = \frac{K_{1,0}}{K_{1,0}+K_{1,1}}$ and $\beta_{0,0} = \frac{K_{1,1}}{K_{1,0}+K_{1,1}}$. Note that $\alpha_{0,0} + \beta_{0,0} = 1$.
The solution now becomes:

$$h_1(y_1) = \begin{cases} 2\beta_{0,0}(B_R - B_L)(y_1) + B_L, & y_1 \in \left[0, \frac{1}{2}\right) \\ 2\alpha_{0,0}(B_R - B_L)(y_1 - 1) + B_R, & y_1 \in \left[\frac{1}{2}, 1\right]. \end{cases}$$

3.3.3 Four-cell Analytic Solution

Given the same equation as before, we compute the solution to the problem using the same steps as above:

$$-\frac{d}{dx}K(x)\frac{dh}{dx} = 0, \tag{3.18}$$

with the boundary conditions

$$\begin{aligned}h(0) &= B_L, \\ h(2) &= B_R,\end{aligned}$$

where,

$$K(x) = \begin{cases} K_{2,0} & x \in \left[0, \frac{1}{2}\right) \\ K_{2,1} & x \in \left[\frac{1}{2}, 1\right] \\ K_{2,2} & x \in \left[1, \frac{3}{2}\right] \\ K_{2,3} & x \in \left[\frac{3}{2}, 2\right]. \end{cases}$$

Integrating the given equation, we obtain

$$-K(x)\frac{dh}{dx} = \begin{cases} c_{2,0} & x \in \left[0, \frac{1}{2}\right) \\ c_{2,1} & x \in \left[\frac{1}{2}, 1\right] \\ c_{2,2} & x \in \left[1, \frac{3}{2}\right] \\ c_{2,3} & x \in \left[\frac{3}{2}, 2\right]. \end{cases}$$

Dividing both sides by $-K(x)$, we have

$$\frac{dh}{dx} = -\begin{cases} \frac{c_{2,0}}{K_{2,0}} & x \in \left[0, \frac{1}{2}\right) \\ \frac{c_{2,1}}{K_{2,1}} & x \in \left[\frac{1}{2}, 1\right] \\ \frac{c_{2,2}}{K_{2,2}} & x \in \left[1, \frac{3}{2}\right] \\ \frac{c_{2,3}}{K_{2,3}} & x \in \left[\frac{3}{2}, 2\right]. \end{cases}$$

Integrating this with respect to x, we obtain

$$h(x) = -\begin{cases} \dfrac{c_{2,0}}{K_{2,0}}x + b_{2,0} & x \in \left[0, \dfrac{1}{2}\right) \\ \dfrac{c_{2,1}}{K_{2,1}}(x-1) + b_{2,1} & x \in \left[\dfrac{1}{2}, 1\right] \\ \dfrac{c_{2,2}}{K_{2,2}}(x-1) + b_{2,2} & x \in \left[1, \dfrac{3}{2}\right] \\ \dfrac{c_{2,3}}{K_{2,3}}(x-2) + b_{2,3} & x \in \left[\dfrac{3}{2}, 2\right]. \end{cases}$$

Again, the boundary conditions are:

$$\begin{aligned} h(0) &= B_L = b_{2,0}, \\ h\left(2\right) &= B_R = b_{2,3}. \end{aligned}$$

We let

$$b_{2,1} = b_{2,2} = B.$$

Again, assume continuity at $x = \frac{1}{2}, x = \frac{3}{2}$, so $c_{2,0} = c_{2,1} = c_{2,2} = c_{2,3}$, and

$$\begin{aligned} \frac{c_{2,0}}{K_{2,0}}y_2 + B_L &= \frac{c_{2,0}}{K_{2,1}}(y_2 - 1) + B, \\ \frac{c_{2,0}}{K_{2,0}}\left(\frac{1}{2}\right) + B_L &= \frac{c_{2,0}}{K_{2,1}}\left(-\frac{1}{2}\right) + B. \end{aligned}$$

We next solve for the value of $c_{2,0}$

$$\begin{aligned} \frac{c_{2,0}}{2K_{2,0}} + \frac{c_{2,0}}{2K_{2,1}} &= B - B_L, \\ c_{2,0}\left(\frac{K_{2,1} + K_{2,0}}{2K_{2,0}K_{2,1}}\right) &= B - B_L, \\ c_{2,0} &= \left(\frac{2K_{2,0}K_{2,1}}{K_{2,1} + K_{2,0}}\right)(B - B_L), \\ c_{2,0} &= K_{1,0}^{\sharp}(B - B_L), \\ B - B_L &= \frac{c_{2,0}}{K_{1,0}^{\sharp}}. \end{aligned}$$

Also,

$$\begin{aligned}\frac{c_{2,0}}{K_{2,2}}(y_2-1)+B &= \frac{c_{2,0}}{K_{2,3}}(y_2-2)+B_R,\\ \frac{c_{2,0}}{K_{2,2}}\left(\frac{1}{2}\right)+B &= \frac{c_{2,0}}{K_{2,3}}\left(-\frac{1}{2}\right)+B_R.\end{aligned}$$

We again solve for the value of $c_{2,0}$

$$\begin{aligned}\frac{c_{2,0}}{2K_{2,2}}+\frac{c_{2,0}}{2K_{2,3}} &= B_R-B, &(3.19)\\ c_{2,0}\left(\frac{K_{2,2}+K_{2,3}}{2K_{2,2}K_{2,3}}\right) &= B_R-B, &(3.20)\\ c_{2,0} &= \left(\frac{2K_{2,2}K_{2,3}}{K_{2,2}+K_{2,3}}\right)(B_R-B), &(3.21)\\ c_{2,0} &= K^{\sharp}_{1,1}(B_R-B), &(3.22)\\ B_R-B &= \frac{c_{2,0}}{K^{\sharp}_{1,1}}. &(3.23)\end{aligned}$$

Add equations 3.19 and 3.23:

$$\begin{aligned}\frac{c_{2,0}}{K^{\sharp}_{1,0}}+\frac{c_{2,0}}{K^{\sharp}_{1,1}} &= B_R-B_L,\\ \frac{2c_{2,0}}{2K^{\sharp}_{1,0}}+\frac{2c_{2,0}}{2K^{\sharp}_{1,1}} &= B_R-B_L,\\ 2c_{2,0} &= \left(\frac{2K^{\sharp}_{1,0}K^{\sharp}_{1,1}}{K^{\sharp}_{1,1}+K^{\sharp}_{1,0}}\right)(B_R-B_L),\\ 2c_{2,0} &= K^{\sharp}_{0,0}(B_R-B_L),\\ c_{2,0} &= \frac{1}{2}K^{\sharp}_{0,0}(B_R-B_L).\end{aligned}$$

Substituting the value of $c_{2,0}$ into the equation above, we obtain:

$$h_2(y_2) = \begin{cases} \dfrac{\frac{1}{2}K^{\sharp}_{0,0}}{K_{2,0}}(B_R - B_L)(y_2) + B_L, & y_2 \in \left[0, \dfrac{1}{2}\right) \\ \dfrac{\frac{1}{2}K^{\sharp}_{0,0}}{K_{2,1}}(B_R - B_L)(y_2 - 1) + \beta_{0,0}(B_R - B_L) + B_L, & y_2 \in \left[\dfrac{1}{2}, 1\right] \\ \dfrac{\frac{1}{2}K^{\sharp}_{0,0}}{K_{2,2}}(B_R - B_L)(y_2 - 1) - \alpha_{0,0}(B_R - B_L) + B_R, & y_2 \in \left[1, \dfrac{3}{2}\right] \\ \dfrac{\frac{1}{2}K^{\sharp}_{0,0}}{K_{2,3}}(B_R - B_L)(y_2 - 2) + B_R, & y_2 \in \left[\dfrac{3}{2}, 2\right]. \end{cases}$$

Simplifying the equation above as follows:

$$\begin{aligned} \frac{\frac{1}{2}K^{\sharp}_{0,0}}{K_{2,0}} &= \frac{\frac{1}{2}\frac{2K^{\sharp}_{1,0}K^{\sharp}_{1,1}}{K^{\sharp}_{1,1}+K^{\sharp}_{1,0}}}{K_{2,0}}, \\ &= \left(\frac{K^{\sharp}_{1,1}}{K^{\sharp}_{1,0} + K^{\sharp}_{1,1}}\right)\frac{K^{\sharp}_{1,0}}{K2,0}, \\ &= \beta_{0,0}\left(\frac{\frac{2K_{2,0}K_{2,1}}{K_{2,0}+K_{2,1}}}{K_{2,0}}\right), \\ &= 2\beta_{0,0}\beta_{1,0}, \end{aligned}$$

$$\begin{aligned} \frac{\frac{1}{2}K^{\sharp}_{0,0}}{K_{2,1}} &= \frac{\frac{1}{2}\frac{2K^{\sharp}_{1,0}K^{\sharp}_{1,1}}{K^{\sharp}_{1,1}+K^{\sharp}_{1,0}}}{K_{2,1}}, \\ &= \left(\frac{K^{\sharp}_{1,1}}{K^{\sharp}_{1,0} + K^{\sharp}_{1,1}}\right)\frac{K^{\sharp}_{1,0}}{K2,1}, \\ &= \beta_{0,0}\left(\frac{\frac{2K_{2,0}K_{2,1}}{K_{2,0}+K_{2,1}}}{K_{2,1}}\right), \\ &= 2\beta_{0,0}\alpha_{1,0}, \end{aligned}$$

$$\begin{aligned}
\frac{\frac{1}{2}K^{\sharp}_{0,0}}{K_{2,2}} &= \frac{\frac{1}{2}\frac{2K^{\sharp}_{1,0}K^{\sharp}_{1,1}}{K^{\sharp}_{1,1}+K^{\sharp}_{1,0}}}{K_{2,2}}, \\
&= \left(\frac{K^{\sharp}_{1,0}}{K^{\sharp}_{1,0}+K^{\sharp}_{1,1}}\right)\frac{K^{\sharp}_{1,1}}{K2,2}, \\
&= \alpha_{0,0}\left(\frac{\frac{2K_{2,2}K_{2,3}}{K_{2,2}+K_{2,3}}}{K_{2,2}}\right), \\
&= 2\alpha_{0,0}\beta_{1,1},
\end{aligned}$$

and

$$\begin{aligned}
\frac{\frac{1}{2}K^{\sharp}_{0,0}}{K_{2,3}} &= \frac{\frac{1}{2}\frac{2K^{\sharp}_{1,0}K^{\sharp}_{1,1}}{K^{\sharp}_{1,1}+K^{\sharp}_{1,0}}}{K_{2,3}}, \\
&= \left(\frac{K^{\sharp}_{1,0}}{K^{\sharp}_{1,0}+K^{\sharp}_{1,1}}\right)\frac{K^{\sharp}_{1,1}}{K2,3}, \\
&= \alpha_{0,0}\left(\frac{\frac{2K_{2,2}K_{2,3}}{K_{2,2}+K_{2,3}}}{K_{2,3}}\right), \\
&= 2\alpha_{0,0}\alpha_{1,1}.
\end{aligned}$$

where $\alpha_{1,1} = \frac{K_{2,2}}{K_{2,2}+K_{2,3}}$ and $\beta_{1,1} = \frac{K_{2,3}}{K_{2,2}+K_{2,3}}$. Also note that $\alpha_{1,1} + \beta_{1,1} = 1$.
The solution becomes:

$$h_2(y_2) = \begin{cases}
2\beta_{0,0}\beta_{1,0}(B_R - B_L)(y_2) + B_L, & y_2 \in \left[0, \frac{1}{2}\right) \\
2\beta_{0,0}\alpha_{1,0}(B_R - B_L)(y_2 - 1) + \beta_{0,0}B_R + (1 - \beta_{0,0})B_L, & y_2 \in \left[\frac{1}{2}, 1\right] \\
2\alpha_{0,0}\beta_{1,1}(B_R - B_L)(y_2 - 1) + (1 - \alpha_{0,0})B_R + \alpha_{0,0}B_L, & y_2 \in \left[1, \frac{3}{2}\right] \\
2\alpha_{0,0}\alpha_{1,1}(B_R - B_L)(y_2 - 2) + B_R, & y_2 \in \left[\frac{3}{2}, 2\right].
\end{cases}$$

After performing some mathematical analysis on the elliptic equations, we summarize the results as follows. We denote the solutions to zeroth scale, two-cell, and four-cell as $h_0(y_0), h_1(y_1)$, and $h_2(y_2)$, respectively.

$$h_0(y_0) = 2(B_R - B_L)y_0 + B_L. \tag{3.24}$$

$$h_1(y_1) = \begin{cases} 2\beta_{0,0}(B_R - B_L)(y_1) + B_L, & y_1 \in \left[0, \frac{1}{2}\right), \\ 2\alpha_{0,0}(B_R - B_L)(y_1 - 1) + B_R, & y_1 \in \left[\frac{1}{2}, 1\right]. \end{cases} \tag{3.25}$$

$$h_2(y_2) = \begin{cases} 2\beta_{0,0}\beta_{1,0}(B_R - B_L)(y_2) + B_L, & y_2 \in \left[0, \frac{1}{2}\right), \\ 2\beta_{0,0}\alpha_{1,0}(B_R - B_L)(y_2 - 1) + \beta_{0,0}B_R + (1 - \beta_{0,0})B_L, & y_2 \in \left[\frac{1}{2}, 1\right], \\ 2\alpha_{0,0}\beta_{1,1}(B_R - B_L)(y_2 - 1) + (1 - \alpha_{0,0})B_R + \alpha_{0,0}B_L, & y_2 \in \left[1, \frac{3}{2}\right], \\ 2\alpha_{0,0}\alpha_{1,1}(B_R - B_L)(y_2 - 2) + B_R, & y_2 \in \left[\frac{3}{2}, 2\right]. \end{cases} \tag{3.26}$$

3.3.4 Difference in the Homogenization Wavelet Reconstruction Approximation Between Scales

We derived closed form formulae for the solutions and now compute the differences $h_n(y_n) - h_{n-1}(y_{n-1})$, between $h_n(y_n)$ and $h_{n-1}(y_{n-1})$. The results are as follows:

$$h_1(y_1) - h_0(y_0) = (B_R - B_L) \begin{cases} (\beta_{0,0} - \alpha_{0,0})y_1, & y_1 \in \left[0, \frac{1}{2}\right) \\ (\beta_{0,0} - \alpha_{0,0})(1 - y_1), & y_1 \in \left[\frac{1}{2}, 1\right]. \end{cases}$$

$$h_2(y_2) - h_1(y_1) = (B_R - B_L) \begin{cases} \beta_{0,0}(\beta_{1,0} - \alpha_{1,0})y_2, & y_2 \in \left[0, \frac{1}{2}\right) \\ \beta_{0,0}(\beta_{1,0} - \alpha_{1,0})(1 - y_2), & y_2 \in \left[\frac{1}{2}, 1\right] \\ \alpha_{0,0}(\beta_{1,1} - \alpha_{1,1})(y_2 - 1), & y_2 \in \left[1, \frac{3}{2}\right] \\ \alpha_{0,0}(\beta_{1,1} - \alpha_{1,1})(2 - y_2), & y_2 \in \left[\frac{3}{2}, 2\right]. \end{cases}$$

$$h_3(y_3) - h_2(y_2) = (B_R - B_L) \begin{cases} \beta_{0,0}\beta_{1,0}(\beta_{2,0} - \alpha_{2,0})y_3, & y_3 \in \left[0, \frac{1}{2}\right) \\ \beta_{0,0}\beta_{1,0}(\beta_{2,0} - \alpha_{2,0})(1 - y_3), & y_3 \in \left[\frac{1}{2}, 1\right] \\ \beta_{0,0}\alpha_{1,0}(\beta_{2,1} - \alpha_{2,1})(y_3 - 1), & y_3 \in \left[1, \frac{3}{2}\right] \\ \beta_{0,0}\alpha_{1,0}(\beta_{2,1} - \alpha_{2,1})(2 - y_3), & y_3 \in \left[\frac{3}{2}, 2\right] \\ \alpha_{0,0}\beta_{1,1}(\beta_{2,2} - \alpha_{2,2})(y_3 - 2), & y_3 \in \left[2, \frac{5}{2}\right] \\ \alpha_{0,0}\beta_{1,1}(\beta_{2,2} - \alpha_{2,2})(3 - y_3), & y_3 \in \left[\frac{5}{2}, 3\right] \\ \alpha_{0,0}\alpha_{1,1}(\beta_{2,3} - \alpha_{2,3})(y_3 - 3), & y_3 \in \left[3, \frac{7}{2}\right] \\ \alpha_{0,0}\alpha_{1,1}(\beta_{2,3} - \alpha_{2,3})(4 - y_3), & y_3 \in \left[\frac{7}{2}, 4\right]. \end{cases}$$

Simplifying the above differences, we let $\gamma_{i,j} = (\beta_{i,j} - \alpha_{i,j})$ and $\varphi = (B_R - B_L)$, so we obtain the following:

$$h_1(y_1) - h_0(y_0) = \varphi \begin{cases} \gamma_{0,0}y_1 \\ \gamma_{0,0}(1 - y_1) \end{cases} = \varphi \cdot \gamma_{0,0} \begin{cases} y_1 \\ (1 - y_1) \end{cases}$$

$$h_2(y_2) - h_1(y_1) = \varphi \begin{cases} \beta_{0,0}\gamma_{1,0}y_2 \\ \beta_{0,0}\gamma_{1,0}(1 - y_2) \\ \alpha_{0,0}\gamma_{1,1}(y_2 - 1) \\ \alpha_{0,0}\gamma_{1,1}(2 - y_2) \end{cases} = \varphi \begin{cases} \beta_{0,0}\gamma_{1,0} \begin{cases} y_2 \\ (1 - y_2) \end{cases} \\ \alpha_{0,0}\gamma_{1,1} \begin{cases} (y_2 - 1) \\ (2 - y_2) \end{cases} \end{cases}$$

$$h_3(y_3) - h_2(y_2) = \varphi \begin{cases} \beta_{0,0}\beta_{1,0}\gamma_{2,0}y_3 \\ \beta_{0,0}\beta_{1,0}\gamma_{2,0}(1-y_3) \\ \beta_{0,0}\alpha_{1,0}\gamma_{2,1}(y_3-1) \\ \beta_{0,0}\alpha_{1,0}\gamma_{2,1}(2-y_3) \\ \alpha_{0,0}\beta_{1,1}\gamma_{2,2}(y_3-2) \\ \alpha_{0,0}\beta_{1,1}\gamma_{2,2}(3-y_3) \\ \alpha_{0,0}\alpha_{1,1}\gamma_{2,3}(y_3-3) \\ \alpha_{0,0}\alpha_{1,1}\gamma_{2,3}(4-y_3 \end{cases} = \varphi \begin{cases} \beta_{0,0}\beta_{1,0}\gamma_{2,0} \begin{cases} y_3 \\ (1-y_3) \end{cases} \\ \beta_{0,0}\alpha_{1,0}\gamma_{2,1} \begin{cases} (y_3-1) \\ (2-y_3) \end{cases} \\ \alpha_{0,0}\beta_{1,1}\gamma_{2,2} \begin{cases} (y_3-2) \\ (3-y_3) \end{cases} \\ \alpha_{0,0}\alpha_{1,1}\gamma_{2,3} \begin{cases} (y_3-3) \\ (4-y_3) \end{cases} \end{cases}$$

Another method for the computation of the harmonic average is to use the algebraic formula, $K^\sharp = \frac{2K_{i,j}k_{i,j+1}}{K_{i,j}+k_{i,j+1}}$. The algorithm for the procedure is given below.

Steps for Computing Harmonic Average Using Algebraic formula

- *First, we assume that the permeability tensor is sampled on a number that is a power of two in each spatial dimension.*

- *Randomly generate the real permeability tensor from* 0.1 *to* 1.0 *defined at* 2^m *samples from* $0, 1, 2, ..., 2^m - 1$.

- *Compute the harmonic average which generate the next level of the permeability tensors.*

- *Then, we compute the* $\alpha's$ *and* $\beta's$.

3.3.5 Some Numerical Results for One-Dimension

We implement a Java code for the above algorithm and using the results generated, we compute the values of the pressure variables and the results are shown in Table 3.1 below. We also plot the graph of the pressure variable against the length, y. Figures 3.1, 3.2, and 3.3 shows the individual graph of $h_1(y_1), h_2(y_1)$, and $h_3(y_1)$. Figure 3.4 combines the three graphs drawn together in a single graph.

Table 3.1. Example of Recursive Differencing of Analytic Solutions

y	$h_1(y_1)$	$h_2(y_1)$	$h_3(y_1)$
0.00000	0.00000	0.00000	0.00000
0.12500			0.11827
0.25000		0.18753	0.18753
0.37500			0.28649
0.50000	0.37370	0.37370	0.37370
0.62500			0.42591
0.75000		0..57192	0.57192
0.87500			0.840414
1.00000	1.00000	1.00000	1.00000

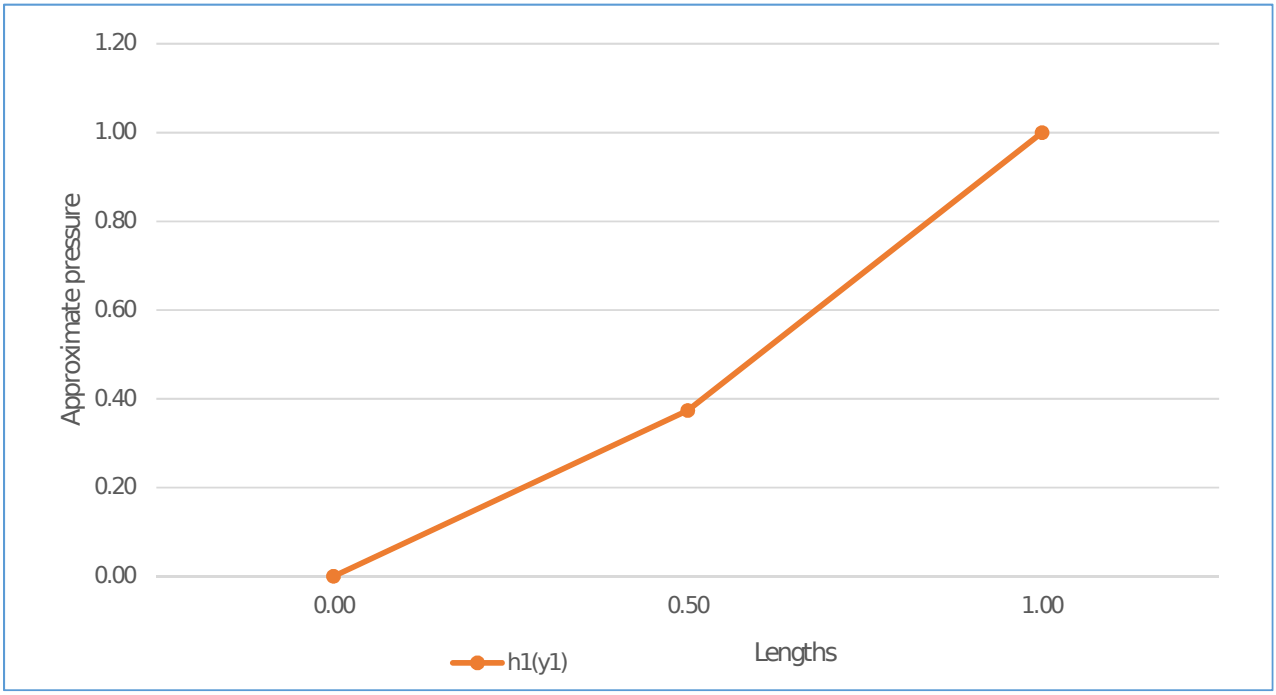

Figure 3.3. The graph of $h_1(y_1)$ against y

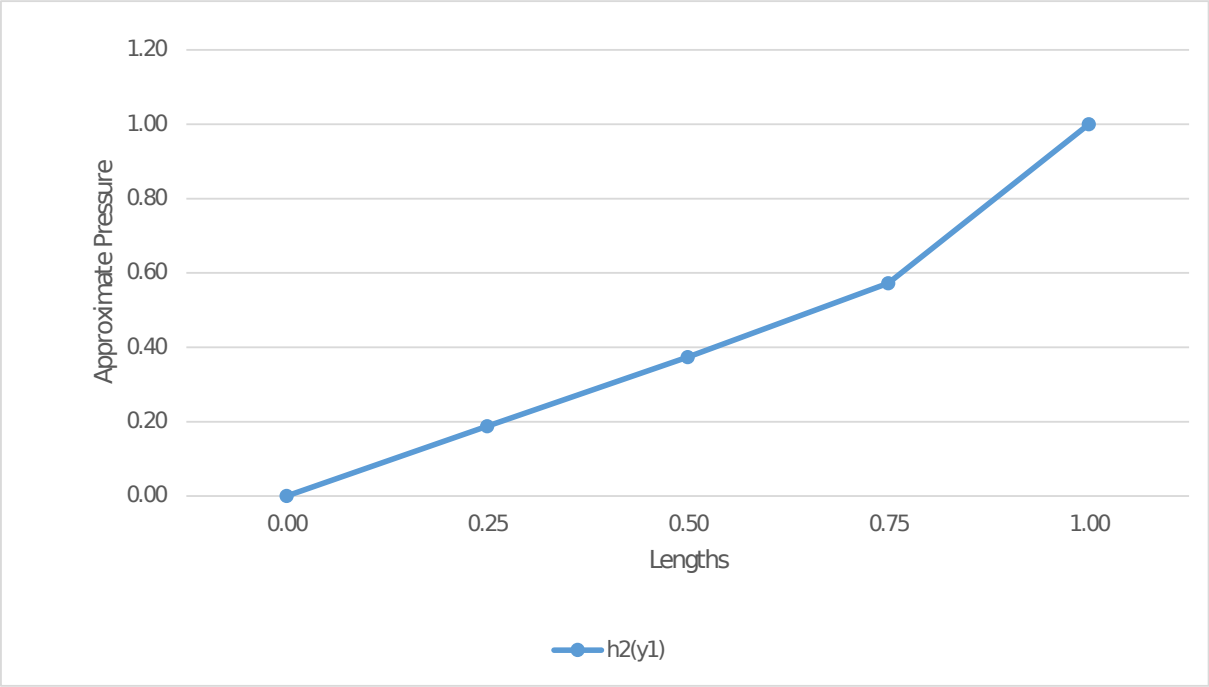

Figure 3.4. The graph of $h_2(y_1)$ against y

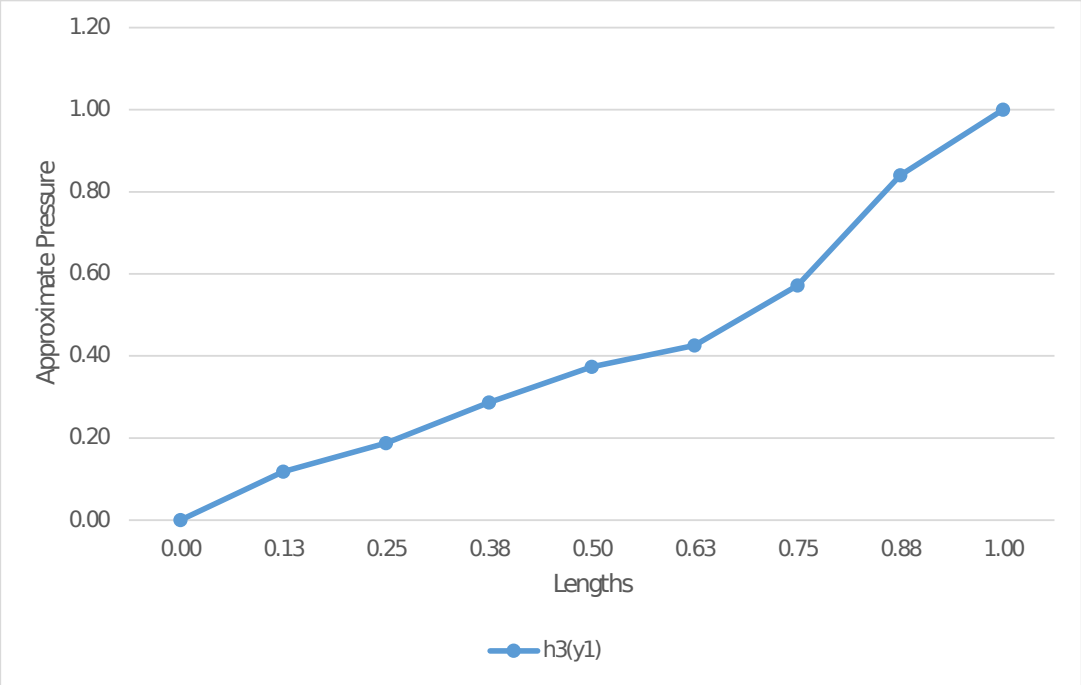

Figure 3.5. The graph of $h_3(y_1)$ against y

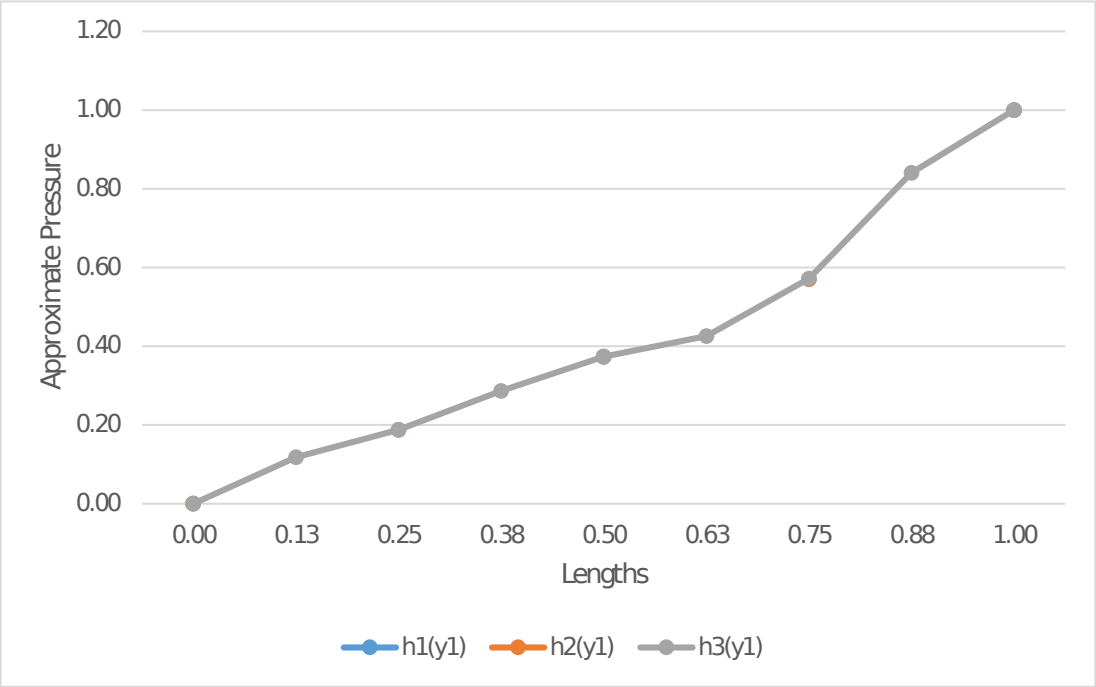

Figure 3.6. The graph of $h_1(y_1), h_2(y_1)$, and $h_3(y_1)$ against y

CHAPTER 4

HOMOGENIZATION WAVELET RECONSTRUCTION IN TWO DIMENSIONS WITH DIAGONAL TENSORS

In this chapter, we consider two dimensional elliptic problem of the form:

$$\begin{cases} \nabla\cdot K\nabla h(x,y) = 0 & (x,y)\in\Omega \\ h = 0 & (x,y)\ on\ \partial\Omega \end{cases} \tag{4.1}$$

where K is a 2×2 permeability or conductivity tensor and h is the pressure head variable in two dimensions. The porous medium is contained in a smooth bounded domain, Ω in $\mathbb{R}^2$ that is covered by a regular mesh of size $n\times n$.

We use the results from the one dimensional case to develop solutions to the local problems in two dimensions. We then develop a fast transform method using homogenization theory over the brute force method. Furthermore, we reverse the fast transform process to obtain the inverse transform. The results of the reverse process are then used in the reconstruction algorithm in the two dimensions.

4.1 The Local Problem in Two Dimensions

We define the local problem in two dimensions by the system of elliptic equations as follows:

$$\nabla\cdot K\nabla w_i = -\nabla\cdot K\vec{e_i} \tag{4.2}$$

for $i = 1, 2$, where w_i is the Jacobian matrix of the functions, and $\vec{e_i}$ is the unit vector in $\mathbb{R}^2$ with the following permeability definitions

$$K(y_1, y_2) = \begin{cases} K^I & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ K^{II} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ K^{III} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ K^{IV} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right], \end{cases}$$

and let K and K^{-1} be 2×2 matrices of the form :

$$K^s = \begin{bmatrix} K^s_{xx} & K^s_{xy} \\ K^s_{yx} & K^s_{yy} \end{bmatrix},$$

$$(K^{-1})^s = \begin{bmatrix} a^s_{11} & a^s_{12} \\ a^s_{21} & a^s_{22} \end{bmatrix},$$

where $s = I, II, III, IV$ and note that $K_{xy} = K_{yx}$ and $a_{12} = a_{21}$.

4.1.1 The Weak Solution of the Elliptic Local Problems

The Jacobian matrix J, contains the solution of the system of elliptic problems. We need to write out appropriate formula to compute the details of the solutions and do the necessary integration. We provide approximate solutions to the systems of elliptic partial differential equations in two cell problem as described below.

Given the local problem in two dimension:

$$\nabla \cdot K \nabla w_i = -\nabla \cdot K \vec{e_i}.$$

Rewriting the equation as a homogeneous equation, we have for i = 1

$$\nabla \cdot K \nabla w_1 + \nabla \cdot K \vec{e_1} = 0. \tag{4.3}$$

Integrating the above equation, we have

$$K(\nabla w_1 + \vec{e_1}) = \begin{cases} c_1 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ c_2 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ c_3 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ c_4 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right] \end{cases}$$

where $c_1, c_2, c_3, c_4 \in \mathbb{R}^2$ and are defined by : $c_1 = \begin{bmatrix} c_{11} \\ c_{12} \end{bmatrix}, c_2 = \begin{bmatrix} c_{21} \\ c_{22} \end{bmatrix}, c_3 = \begin{bmatrix} c_{31} \\ c_{32} \end{bmatrix}$ and $c_4 = \begin{bmatrix} c_{41} \\ c_{42} \end{bmatrix}$.

Then, multiply both sides by K^{-1},

$$\nabla w_1 + \vec{e_1} = \begin{cases} K_I^{-1} c_1 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ K_{II}^{-1} c_2 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ K_{III}^{-1} c_3 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ K_{IV}^{-1} c_4 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right]. \end{cases}$$

Solving for ∇w_1, we see that

$$\nabla w_1 = \begin{cases} K_I^{-1} c_1 - \vec{e_1} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ K_{II}^{-1} c_2 - \vec{e_1} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ K_{III}^{-1} c_3 - \vec{e_1} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ K_{IV}^{-1} c_4 - \vec{e_1} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right] \end{cases}$$

where $\vec{e_1} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $\vec{e_2} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$.

Integrate the above equation with respect to y, we obtain

$$w_1 = \begin{cases} (K_I^{-1}c_1 - \vec{e_1}) \cdot [y_1, y_2] & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ (K_{II}^{-1}c_2 - \vec{e_1}) \cdot [y_1 - 1, y_2] & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ (K_{III}^{-1}c_3 - \vec{e_1}) \cdot [y_1, y_2 - 1] & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ (K_{IV}^{-1}c_4 - \vec{e_1}) \cdot [y_1 - 1, y_2 - 1] & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right] \end{cases}$$

and then we make assumptions on the continuity of $K\nabla w$ on the 2×2 cell boundaries. This allows us to reduce the problem with eight unknowns to four equations with four unknowns. We set the following four conditions on each of the step functions with $c_{11} = c_{21}, c_{12} = c_{32}, c_{22} = c_{42}$, and $c_{31} = c_{41}$:

$$\begin{aligned} (K_I^{-1}c_1 - \vec{e_1})(y_1, y_2) &= (K_{II}^{-1}c_2 - \vec{e_1})(y_1 - 1, y_2), \\ (K_I^{-1}c_1 - \vec{e_1})(y_1, y_2) &= (K_{II}^{-1}c_2 - \vec{e_1})(y_1 - 1, y_2), \\ (K_{III}^{-1}c_3 - \vec{e_1})(y_1, y_2 - 1) &= (K_{IV}^{-1}c_4 - \vec{e_1})(y_1 - 1, y_2 - 1), \\ (K_{III}^{-1}c_3 - \vec{e_1})(y_1, y_2 - 1) &= (K_{IV}^{-1}c_4 - \vec{e_1})(y_1 - 1, y_2 - 1). \end{aligned}$$

Solving this system of equations with the following points : $(y_1, y_2) = (\frac{1}{2}, 0), (y_1, y_2) = (\frac{1}{2}, \frac{1}{2}), (y_1, y_2) = (1, \frac{1}{2})$, we obtain the system of equations:

$$\frac{1}{2}(a_{11}^{I}+a_{11}^{II})c_{11}+\frac{1}{2}a_{12}^{I}c_{12}+\frac{1}{2}a_{12}^{II}c_{22}=1, \tag{4.4}$$

$$\frac{1}{2}(a_{12}^{I}-a_{12}^{II})c_{11}+\frac{1}{2}a_{22}^{I}c_{12}-\frac{1}{2}a_{22}^{II}c_{22}=0, \tag{4.5}$$

$$\frac{1}{2}(a_{12}^{IV}-a_{12}^{III})c_{31}-\frac{1}{2}a_{22}^{III}c_{12}+\frac{1}{2}a_{22}^{IV}c_{22}=0, \tag{4.6}$$

$$\frac{1}{2}(a_{11}^{III}+a_{11}^{IV})c_{31}+\frac{1}{2}a_{12}^{III}c_{12}+\frac{1}{2}a_{12}^{IV}c_{22}=1. \tag{4.7}$$

These equations in matrix form are as follow:

$$\begin{bmatrix} \frac{1}{2}(a_{11}^{I}+a_{11}^{II}) & \frac{1}{2}a_{12}^{I} & \frac{1}{2}a_{12}^{II} & 0 \\ \frac{1}{2}(a_{12}^{I}-a_{12}^{II}) & \frac{1}{2}a_{22}^{I} & -\frac{1}{2}a_{22}^{II} & 0 \\ 0 & -\frac{1}{2}a_{22}^{III} & \frac{1}{2}a_{22}^{IV} & \frac{1}{2}(a_{12}^{IV}-a_{12}^{III}) \\ 0 & \frac{1}{2}a_{12}^{III} & \frac{1}{2}a_{12}^{IV} & \frac{1}{2}(a_{11}^{III}+a_{11}^{IV}) \end{bmatrix} \cdot \begin{bmatrix} c_{11} \\ c_{12} \\ c_{22} \\ c_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Since this is a diagonal tensor, $a_{12}=0$, and solving the system we obtain the following solutions:

$$\begin{bmatrix} \frac{1}{2}(a_{11}^{I}+a_{11}^{II}) & 0 & 0 & 0 \\ 0 & \frac{1}{2}a_{22}^{I} & -\frac{1}{2}a_{22}^{II} & 0 \\ 0 & -\frac{1}{2}a_{22}^{III} & \frac{1}{2}a_{22}^{IV} & 0 \\ 0 & 0 & 0 & \frac{1}{2}(a_{11}^{III}+a_{11}^{IV}) \end{bmatrix} \cdot \begin{bmatrix} c_{11} \\ c_{12} \\ c_{22} \\ c_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

$$\begin{bmatrix} c_{11} \\ c_{12} \\ c_{22} \\ c_{31} \end{bmatrix} = \begin{bmatrix} \frac{2}{a_{11}^{I}+a_{11}^{II}} \\ 0 \\ 0 \\ \frac{2}{a_{11}^{III}+a_{11}^{IV}} \end{bmatrix}.$$

We then put the values of $c_{11} - c_{31}$ into $w_1(y_1, y_2)$

$$w_1 = \begin{cases} (K_I^{-1}c_1 - \vec{e_1})(y_1, y_2) \\ (K_{II}^{-1}c_2 - \vec{e_1})(y_1 - 1, y_2) \\ (K_{III}^{-1}c_3 - \vec{e_1})(y_1, y_2 - 1) \\ (K_{IV}^{-1}c_4 - \vec{e_1})(y_1 - 1, y_2 - 1), \end{cases}$$

which implies that

$$w_1 = \begin{cases} (2a_{11}^{I}(a_{11}^{I} + a_{11}^{II})^{-1} - 1)(y_1) + (2a_{12}^{I}(a_{11}^{I} + a_{11}^{II})^{-1})(y_2) \\ (2a_{11}^{II}(a_{11}^{I} + a_{11}^{II})^{-1} - 1)(y_1 - 1) + (2a_{12}^{II}(a_{11}^{I} + a_{11}^{II})^{-1})(y_2) \\ (2a_{11}^{III}(a_{11}^{III} + a_{11}^{IV})^{-1} - 1)(y_1) + (2a_{12}^{III}(a_{11}^{III} + a_{11}^{IV})^{-1})(y_2 - 1) \\ (2a_{11}^{IV}(a_{11}^{III} + a_{11}^{IV})^{-1} - 1)(y_1 - 1) + (2a_{12}^{IV}(a_{11}^{III} + a_{11}^{IV})^{-1})(y_2 - 1). \end{cases}$$

Since a_{12} is equal to zero then,

$$w_1 = \begin{cases} (2a_{11}^{I}(a_{11}^{I} + a_{11}^{II})^{-1} - 1)(y_1) \\ (2a_{11}^{II}(a_{11}^{I} + a_{11}^{II})^{-1} - 1)(y_1 - 1) \\ (2a_{11}^{III}(a_{11}^{III} + a_{11}^{IV})^{-1} - 1)(y_1) \\ (2a_{11}^{IV}(a_{11}^{III} + a_{11}^{IV})^{-1} - 1)(y_1 - 1) \end{cases}$$

$$= \begin{cases} \left(\frac{a_{11}^{I} - a_{11}^{II}}{a_{11}^{I} + a_{11}^{II}}\right)(y_1) \\ \left(\frac{a_{11}^{II} - a_{11}^{I}}{a_{11}^{I} + a_{11}^{II}}\right)(y_1 - 1) \\ \left(\frac{a_{11}^{III} - a_{11}^{IV}}{a_{11}^{III} + a_{11}^{IV}}\right)(y_1) \\ \left(\frac{a_{11}^{IV} - a_{11}^{III}}{a_{11}^{III} + a_{11}^{IV}}\right)(y_1 - 1) \end{cases}$$

$$= \begin{cases} \left(\dfrac{a_{11}^{I} - a_{11}^{II}}{a_{11}^{I} + a_{11}^{II}}\right)(y_1) \\ \left(\dfrac{a_{11}^{I} - a_{11}^{II}}{a_{11}^{I} + a_{11}^{II}}\right)(1 - y_1) \\ \left(\dfrac{a_{11}^{III} - a_{11}^{IV}}{a_{11}^{III} + a_{11}^{IV}}\right)(y_1) \\ \left(\dfrac{a_{11}^{III} - a_{11}^{IV}}{a_{11}^{III} + a_{11}^{IV}}\right)(1 - y_1). \end{cases}$$

Now for $i = 2$, the homogeneous equation becomes

$$\nabla \cdot K \nabla w_2 + \nabla \cdot K\vec{e_2} = 0. \tag{4.8}$$

Integrating the above equation, we have

$$K(\nabla w_2 + \vec{e_2}) = \begin{cases} c_1 & (y_1, y_2) \in \left[0, \dfrac{1}{2}\right] \times \left[0, \dfrac{1}{2}\right] \\ c_2 & (y_1, y_2) \in \left[\dfrac{1}{2}, 1\right] \times \left[0, \dfrac{1}{2}\right] \\ c_3 & (y_1, y_2) \in \left[0, \dfrac{1}{2}\right] \times \left[\dfrac{1}{2}, 1\right] \\ c_4 & (y_1, y_2) \in \left[\dfrac{1}{2}, 1\right] \times \left[\dfrac{1}{2}, 1\right]. \end{cases}$$

Then, multiply both sides by K^{-1},

$$\nabla w_2 + \vec{e_2} = \begin{cases} K_I^{-1} c_1 & (y_1, y_2) \in \left[0, \dfrac{1}{2}\right] \times \left[0, \dfrac{1}{2}\right] \\ K_{II}^{-1} c_2 & (y_1, y_2) \in \left[\dfrac{1}{2}, 1\right] \times \left[0, \dfrac{1}{2}\right] \\ K_{III}^{-1} c_3 & (y_1, y_2) \in \left[0, \dfrac{1}{2}\right] \times \left[\dfrac{1}{2}, 1\right] \\ K_{IV}^{-1} c_4 & (y_1, y_2) \in \left[\dfrac{1}{2}, 1\right] \times \left[\dfrac{1}{2}, 1\right] \end{cases}$$

$$\nabla w_2 = \begin{cases} K_I^{-1}c_1 - \vec{e_2} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ K_{II}^{-1}c_2 - \vec{e_2} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ K_{III}^{-1}c_3 - \vec{e_2} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ K_{IV}^{-1}c_4 - \vec{e_2} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right]. \end{cases}$$

Integrate the later, we arrive at

$$w_2 = \begin{cases} (K_I^{-1}c_1 - \vec{e_2}) \cdot [y_1, y_2] & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ (K_{II}^{-1}c_2 - \vec{e_2}) \cdot [y_1 - 1, y_2] & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ (K_{III}^{-1}c_3 - \vec{e_2}) \cdot [y_1, y_2 - 1] & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ (K_{IV}^{-1}c_4 - \vec{e_2}) \cdot [y_1 - 1, y_2 - 1] & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right] \end{cases}$$

and again we make assumptions on continuity at some points. This again allows us to reduce the problem with eight unknowns to four equations with four unknowns. We set the following four conditions on each of the step functions with $c_{11} = c_{21}, c_{12} = c_{32}, c_{22} = c_{42}$, and $c_{31} = c_{41}$:

$$\begin{aligned} (K_I^{-1}c_1 - \vec{e_2})(y_1, y_2) &= (K_{II}^{-1}c_2 - \vec{e_2})(y_1 - 1, y_2), \\ (K_I^{-1}c_1 - \vec{e_2})(y_1, y_2) &= (K_{II}^{-1}c_2 - \vec{e_2})(y_1 - 1, y_2), \\ (K_{III}^{-1}c_3 - \vec{e_2})(y_1, y_2 - 1) &= (K_{IV}^{-1}c_4 - \vec{e_2})(y_1 - 1, y_2 - 1), \\ (K_{III}^{-1}c_3 - \vec{e_2})(y_1, y_2 - 1) &= (K_{IV}^{-1}c_4 - \vec{e_2})(y_1 - 1, y_2 - 1). \end{aligned}$$

Solving this system of equations with the following points : $(y_1 = 0, y_2 = \frac{1}{2}), (y_1 = \frac{1}{2}, y_2 = \frac{1}{2}), (y_1 = \frac{1}{2}, y_2 = \frac{1}{2})$, and $(y_1 = 1, y_2 = \frac{1}{2})$, we obtain the system of equations:

$$\frac{1}{2}(a_{22}^{I}+a_{22}^{III})c_{12}+\frac{1}{2}a_{12}^{I}c_{11}+\frac{1}{2}a_{12}^{III}c_{31}=1, \tag{4.9}$$

$$\frac{1}{2}(a_{12}^{I}-a_{12}^{III})c_{12}+\frac{1}{2}a_{11}^{I}c_{11}-\frac{1}{2}a_{11}^{III}c_{31}=0, \tag{4.10}$$

$$\frac{1}{2}(a_{12}^{IV}-a_{12}^{II})c_{22}-\frac{1}{2}a_{11}^{II}c_{11}+\frac{1}{2}a_{11}^{IV}c_{31}=0, \tag{4.11}$$

$$\frac{1}{2}(a_{22}^{II}+a_{22}^{IV})c_{22}+\frac{1}{2}a_{12}^{II}c_{11}+\frac{1}{2}a_{12}^{IV}c_{31}=1. \tag{4.12}$$

These equations in matrix form are as follows:

$$\begin{bmatrix} \frac{1}{2}a_{12}^{I} & \frac{1}{2}(a_{22}^{I}+a_{22}^{III}) & 0 & \frac{1}{2}a_{12}^{III} \\ \frac{1}{2}a_{11}^{I} & \frac{1}{2}(a_{12}^{I}-a_{12}^{III}) & 0 & -\frac{1}{2}a_{11}^{III} \\ -\frac{1}{2}a_{11}^{II} & 0 & \frac{1}{2}(a_{12}^{IV}-a_{12}^{II}) & \frac{1}{2}a_{11}^{IV} \\ \frac{1}{2}a_{12}^{II} & 0 & \frac{1}{2}(a_{22}^{II}+a_{22}^{IV}) & \frac{1}{2}a_{12}^{IV} \end{bmatrix} \cdot \begin{bmatrix} c_{11} \\ c_{12} \\ c_{22} \\ c_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Putting $a_{12}=0$, and solving the system, we obtain the following solutions:

$$\begin{bmatrix} 0 & \frac{1}{2}(a_{22}^{I}+a_{22}^{III}) & 0 & 0 \\ \frac{1}{2}a_{11}^{I} & 0 & 0 & -\frac{1}{2}a_{11}^{III} \\ -\frac{1}{2}a_{11}^{II} & 0 & 0 & \frac{1}{2}a_{11}^{IV} \\ 0 & 0 & \frac{1}{2}(a_{22}^{II}+a_{22}^{IV}) & 0 \end{bmatrix} \cdot \begin{bmatrix} c_{11} \\ c_{12} \\ c_{22} \\ c_{31} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix},$$

$$\begin{bmatrix} c_{11} \\ c_{12} \\ c_{22} \\ c_{31} \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{2}{a_{22}^{I}+a_{22}^{III}} \\ \frac{2}{a_{22}^{II}+a_{22}^{IV}} \\ 0 \end{bmatrix}.$$

We then put the values of $c_{11} - c_{31}$ into $w_2(y_1, y_2)$

$$w_2 = \begin{cases} (K_I^{-1}c_1 - \vec{e_2})(y_1, y_2) \\ (K_{II}^{-1}c_2 - \vec{e_2})(y_1 - 1, y_2) \\ (K_{III}^{-1}c_3 - \vec{e_2})(y_1, y_2 - 1) \\ (K_{IV}^{-1}c_4 - \vec{e_2})(y_1 - 1, y_2 - 1) \end{cases}$$

$$= \begin{cases} (2a_{12}^{I}(a_{22}^{I} + a_{22}^{III})^{-1})(y_1) + (2a_{22}^{I}(a_{22}^{I} + a_{22}^{III})^{-1} - 1)(y_2) \\ (2a_{12}^{II}(a_{22}^{II} + a_{22}^{IV})^{-1})(y_1 - 1) + (2a_{22}^{II}(a_{22}^{II} + a_{22}^{IV})^{-1} - 1)(y_2) \\ (2a_{12}^{III}(a_{22}^{I} + a_{22}^{III})^{-1})(y_1) + (2a_{22}^{III}(a_{22}^{I} + a_{22}^{III})^{-1} - 1)(y_2 - 1) \\ (2a_{12}^{IV}(a_{22}^{II} + a_{22}^{IV})^{-1})(y_1 - 1) + (2a_{22}^{IV}(a_{22}^{II} + a_{22}^{IV})^{-1} - 1)(y_2 - 1). \end{cases}$$

Since a_{12} is equal to zero then,

$$w_2 = \begin{cases} (2a_{22}^{I}(a_{22}^{I} + a_{22}^{III})^{-1} - 1)(y_2) \\ (2a_{22}^{II}(a_{22}^{II} + a_{22}^{IV})^{-1} - 1)(y_2) \\ (2a_{22}^{III}(a_{22}^{I} + a_{22}^{III})^{-1} - 1)(y_{2-1}) \\ (2a_{22}^{IV}(a_{22}^{II} + a_{22}^{IV})^{-1} - 1)(y_2 - 1) \end{cases}$$

$$= \begin{cases} \left(\frac{a_{22}^{I} - a_{22}^{III}}{a_{22}^{I} + a_{22}^{III}}\right)(y_2) \\ \left(\frac{a_{22}^{II} - a_{22}^{IV}}{a_{22}^{II} + a_{22}^{IV}}\right)(y_2) \\ \left(\frac{a_{22}^{III} - a_{22}^{I}}{a_{22}^{I} + a_{22}^{III}}\right)(y_2 - 1) \\ \left(\frac{a_{22}^{IV} - a_{22}^{II}}{a_{22}^{II} + a_{22}^{IV}}\right)(y_2 - 1) \end{cases}$$

$$w_2 = \begin{cases} \left(\dfrac{a_{22}^I - a_{22}^{III}}{a_{22}^I + a_{22}^{III}}\right)(y_2) \\ \left(\dfrac{a_{22}^{II} - a_{22}^{IV}}{a_{22}^{II} + a_{22}^{IV}}\right)(y_2) \\ \left(\dfrac{a_{22}^I - a_{22}^{III}}{a_{22}^I + a_{22}^{III}}\right)(1 - y_2) \\ \left(\dfrac{a_{22}^{II} - a_{22}^{IV}}{a_{22}^{II} + a_{22}^{IV}}\right)(1 - y_2). \end{cases}$$

Putting all the calculations above together, we can compute the averaging formula, $K^\sharp$ as follows:

$$K^\sharp = \int K(I + J^T) d\Omega,$$

where

$$J^T = \nabla w_i = \begin{bmatrix} \frac{\partial w_1}{\partial y_1} & \frac{\partial w_2}{\partial y_1} \\ \frac{\partial w_1}{\partial y_2} & \frac{\partial w_2}{\partial y_2} \end{bmatrix}.$$

4.2 The Solution for Diagonal Tensors

In solving this problem, we apply the method in section 4.1.1 to the diagonal matrix, and solve the problem the same way. The results provide the average of the harmonic mean in both x and y directions.

4.2.1 A Symbolic Example of Computing $K^\sharp$ for Constant Diagonal Tensors

Let

$$K_I = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix}, K_{II} = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix}, K_{III} = \begin{bmatrix} c & 0 \\ 0 & c \end{bmatrix}, K_{IV} = \begin{bmatrix} d & 0 \\ 0 & d \end{bmatrix}$$

,

So that,

$$K_I^{-1} = \begin{bmatrix} \frac{1}{a} & 0 \\ 0 & \frac{1}{a} \end{bmatrix}, K_{II}^{-1} = \begin{bmatrix} \frac{1}{b} & 0 \\ 0 & \frac{1}{b} \end{bmatrix}, K_{III}^{-1} = \begin{bmatrix} \frac{1}{c} & 0 \\ 0 & \frac{1}{c} \end{bmatrix}, K_{IV}^{-1} = \begin{bmatrix} \frac{1}{d} & 0 \\ 0 & \frac{1}{d} \end{bmatrix}$$

. Then, we compute the Jacobian of w_1, ∇w_1, we get

$$\nabla w_1 = \begin{cases} \begin{bmatrix} \frac{a-b}{a+b} \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} \frac{b-a}{a+b} \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} \frac{c-d}{c+d} \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[1, \frac{1}{2}\right] \\ \begin{bmatrix} \frac{d-c}{c+d} \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right], \end{cases}$$

and

$$\nabla w_2 = \begin{cases} \begin{bmatrix} 0 \\ \frac{a-c}{a+c} \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} 0 \\ \frac{b-d}{b+d} \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} 0 \\ \frac{c-a}{a+c} \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[1, \frac{1}{2}\right] \\ \begin{bmatrix} 0 \\ \frac{d-b}{b+d} \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right]. \end{cases}$$

We then compute the product of the tensors and the transpose of the Jacobian,

$$K_I \cdot J^T = \begin{bmatrix} a(\frac{a-b}{b+a}) & 0 \\ 0 & a(\frac{a-c}{a+c}) \end{bmatrix}, \; K_{II} \cdot J^T = \begin{bmatrix} b(\frac{b-a}{b+a}) & 0 \\ 0 & b(\frac{b-d}{b+d}) \end{bmatrix},$$

$$K_{III} \cdot J^T = \begin{bmatrix} c(\frac{c-d}{c+d}) & 0 \\ 0 & c(\frac{c-a}{a+c}) \end{bmatrix}, \; K_{IV} \cdot J^T = \begin{bmatrix} d(\frac{d-c}{c+d}) & 0 \\ 0 & d(\frac{d-b}{b+d}) \end{bmatrix}.$$

The next step is to compute the $K^\sharp$ on the entire domain,

$$\begin{aligned} K^\sharp &= \int K(I + J^T)d\Omega \\ &= \int K d\Omega + \int K J^T d\Omega, \end{aligned}$$

where

$$\int K d\Omega = \frac{1}{4}\begin{bmatrix} a+b+c+d & 0 \\ 0 & a+b+c+d \end{bmatrix},$$

and

$$\int K J^T d\Omega = -\frac{1}{4}\begin{bmatrix} \frac{(a-b)^2}{a+b} + \frac{(c-d)^2}{c+d} & 0 \\ 0 & \frac{(a-c)^2}{a+c} + \frac{(b-d)^2}{b+d} \end{bmatrix}.$$

So,

$$K^\sharp = \frac{1}{4}\begin{bmatrix} a+b+c+d & 0 \\ 0 & a+b+c+d \end{bmatrix} - \frac{1}{4}\begin{bmatrix} \frac{(a-b)^2}{a+b} + \frac{(c-d)^2}{c+d} & 0 \\ 0 & \frac{(a-c)^2}{a+c} + \frac{(b-d)^2}{b+d} \end{bmatrix}$$

$$K^\sharp = \frac{1}{4}\begin{bmatrix} -\left(\frac{(a-b)^2}{a+b} + \frac{(c-d)^2}{c+d}\right) + (a+b+c+d) & 0 \\ 0 & -\left(\frac{(a-c)^2}{a+c} + \frac{(b-d)^2}{b+d}\right) + (a+b+c+d) \end{bmatrix}$$

$$K^\sharp = \begin{bmatrix} \left(\frac{ab}{a+b} + \frac{cd}{c+d}\right) & 0 \\ 0 & \left(\frac{ac}{a+c} + \frac{bd}{b+d}\right) \end{bmatrix}.$$

4.2.2 A Numerical Example of Solution for Stratified Diagonal Tensors

In solving this problem, we apply the method in section 4.1.1 to the diagonal matrix, and solve the problem the same way. We obtain the harmonic average of the tensors in the x direction and the y direction gives the arithmetic average of the tensors.

4.2.3 An Example of Computing $K^\sharp$ for Stratified Diagonal Tensors

Let

$$K_I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ K_{II} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix},\ K_{III} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ K_{IV} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}.$$

So that,

$$K_I^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ K_{II}^{-1} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix},\ K_{III}^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ K_{IV}^{-1} = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}.$$

Then, we compute the Jacobian of w_1, ∇w_1, we get

$$\nabla w_1 = \begin{cases} \begin{bmatrix} 0.82 \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} -0.82 \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} 0.82 \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ \begin{bmatrix} -0.82 \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right], \end{cases}$$

and

$$\nabla w_2 = \begin{cases} \begin{bmatrix} 0 \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ \begin{bmatrix} 0 \\ 0 \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right]. \end{cases}$$

We then compute the product of the tensors and the transpose of the Jacobian,

$$K_I \cdot J^T = \begin{bmatrix} 0.82 & 0 \\ 0 & 0 \end{bmatrix}, \; K_{II} \cdot J^T = \begin{bmatrix} -8.2 & 0 \\ 0 & 0 \end{bmatrix},$$

$$K_{III} \cdot J^T = \begin{bmatrix} 0.82 & 0 \\ 0 & 0 \end{bmatrix}, \; K_{IV} \cdot J^T = \begin{bmatrix} -8.2 & 0 \\ 0 & 0 \end{bmatrix}.$$

The next step is to compute the $K^\sharp$, on the entire domain,

$$\begin{aligned} K^\sharp &= \int K(I + J^T) d\Omega \\ &= \int K d\Omega + \int K J^T d\Omega, \end{aligned}$$

where

$$\int K d\Omega = \begin{bmatrix} 5.5 & 0 \\ 0 & 5.5 \end{bmatrix},$$

and

$$\int K J^T d\Omega = \begin{bmatrix} -3.69 & 0 \\ 0 & 0 \end{bmatrix}.$$

So,

$$K^{\sharp} = \begin{bmatrix} 5.5 & 0 \\ 0 & 5.5 \end{bmatrix} + \begin{bmatrix} -3.69 & 0 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 1.81 & 0 \\ 0 & 5.5 \end{bmatrix}.$$

4.2.4 Analogy between Homogenization and Wavelets in Two Dimensions

In line with the one dimensional case, we present an analogy between homogenization and wavelets in two dimensions. The averaging formula in this case can be written as:

$$K^{\sharp} = \iint_{\Omega} K(I + J^T)\partial\Omega \tag{4.13}$$

$$= \iint_{\Omega} K d\Omega + \iint_{\Omega} K J^T \partial\Omega \tag{4.14}$$

The first term is again the arithmetic average of the permeability tensors and the second term is the perturbation needed to compute an appropriate equivalent permeability tensors, where J^T is the Jacobian matrix of the function w_i with respect to y_1, y_2:

$$J^T = \begin{bmatrix} \frac{\partial w_1}{\partial y_1} & \frac{\partial w_2}{\partial y_1} \\ \frac{\partial w_1}{\partial y_2} & \frac{\partial w_2}{\partial y_2} \end{bmatrix}.$$

From Equation 4.14,

$$\overline{K} = \iint K\partial\Omega$$

is the arithmetic average, and

$$\Delta K = \iint K J^T \partial\Omega$$

is the perturbation needed to produce the mean of harmonic average.

4.3 A Two Dimensional Fast Transform for Homogenized Coefficient Values

The correct averaged/upscaled value of $K(xy)$ in two dimensions is the average of the sum of harmonic averages in both x and y directions. That is, if we are given four 2×2

tensors of the form

$$K_I = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix},\ K_{II} = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix},\ K_{III} = \begin{bmatrix} c & 0 \\ 0 & c \end{bmatrix},\ K_{IV} = \begin{bmatrix} d & 0 \\ 0 & d \end{bmatrix},$$

then,

$$K^\sharp = \begin{bmatrix} \left(\frac{ab}{a+b} + \frac{cd}{c+d}\right) & 0 \\ 0 & \left(\frac{ac}{a+c} + \frac{bd}{b+d}\right). \end{bmatrix}$$

Yet, another method for computing this average is to define a transform method that will define averages at all intermediate scales. This process is similar to the procedure in one dimension. We decompose the problem into two sets, one in the x direction and the other one in the y direction as follows:

$$K_{12}^\sharp = \bar{K_1} + \iint \begin{bmatrix} K_{xx}\frac{\partial w_1}{\partial y_1} & 0 \\ 0 & 0 \end{bmatrix} \partial\Omega,$$

and

$$K_{34}^\sharp = \bar{K_2} + \iint \begin{bmatrix} 0 & 0 \\ 0 & K_{yy}\frac{\partial w_1}{\partial y_1} \end{bmatrix} \partial\Omega.$$

Suppose we are given 2×2 tensors as before. We consider using pairs of tensors to compute local averages. For example, using the pair of 2×2 tensors from a sequence of samples, K_I and K_{II}, together with K_{III} and K_{IV}, we can define

$$\begin{aligned} K^\sharp &= K_{12}^\sharp + K_{34}^\sharp, \\ K^\sharp &= \begin{bmatrix} \left(\frac{ab}{a+b} + \frac{cd}{c+d}\right) & 0 \\ 0 & \left(\frac{ac}{a+c} + \frac{bd}{b+d}\right) \end{bmatrix}, \end{aligned}$$

where

$$K_{12}^{\sharp} = \begin{bmatrix} \left(\frac{ab}{a+b} + \frac{cd}{c+d}\right) & 0 \\ 0 & 0 \end{bmatrix},$$

$$K_{34}^{\sharp} = \begin{bmatrix} 0 & 0 \\ 0 & \left(\frac{ac}{a+c} + \frac{bd}{b+d}\right) \end{bmatrix}.$$

To recast the averaging process in a form that can be used in a fast transform algorithm, the simple computation could be rewritten in the following three steps.

Fast Transform Algorithm in Two Dimensions

Let

$$K_I = \begin{bmatrix} a & 0 \\ 0 & 0 \end{bmatrix},\ K_{II} = \begin{bmatrix} b & 0 \\ 0 & 0 \end{bmatrix},\ K_{III} = \begin{bmatrix} c & 0 \\ 0 & 0 \end{bmatrix},\ K_{IV} = \begin{bmatrix} d & 0 \\ 0 & 0 \end{bmatrix},$$

be 2×2 *tensors then*

- *Step 1: Compute the average of* K_I, *and* K_{II}

$$\bar{K} = K_0 = \frac{1}{2}(K_I + K_{II}) = \begin{bmatrix} \frac{a+b}{2} & 0 \\ 0 & \frac{a+b}{2} \end{bmatrix},$$

so,

$$K_0^{-1} = \begin{bmatrix} \frac{2}{a+b} & 0 \\ 0 & \frac{2}{a+b} \end{bmatrix}.$$

- *Step 2: Compute* $\Delta K = K_1$ *using results from step 1 and* $K_{II} - K_0 = \begin{bmatrix} \frac{b-a}{2} & 0 \\ 0 & \frac{b-a}{2} \end{bmatrix}$

$$K_1 = -\begin{bmatrix} \frac{b-a}{2} & 0 \\ 0 & \frac{b-a}{2} \end{bmatrix} \cdot \begin{bmatrix} \frac{a+b}{2} & 0 \\ 0 & \frac{a+b}{2} \end{bmatrix} \cdot \begin{bmatrix} \frac{b-a}{2} & 0 \\ 0 & \frac{b-a}{2} \end{bmatrix} = -\begin{bmatrix} \frac{(b-a)^2}{2(a+b)} & 0 \\ 0 & \frac{(b-a)^2}{2(a+b)} \end{bmatrix}.$$

- *Step 3: Add the results from step 1 and 2 together to get* $K_1^\sharp$

$$\begin{aligned} K_1^\sharp &= \bar{K} + \Delta K = K_0 + K_1 \\ &= \begin{bmatrix} \frac{a+b}{2} & 0 \\ 0 & \frac{a+b}{2} \end{bmatrix} - \begin{bmatrix} \frac{(b-a)^2}{2(a+b)} & 0 \\ 0 & \frac{(b-a)^2}{2(a+b)} \end{bmatrix} \\ &= \begin{bmatrix} \frac{2ab}{a+b} & 0 \\ 0 & \frac{2ab}{a+b} \end{bmatrix}. \end{aligned}$$

Let $K_1^\sharp$ *be* 2×2 *tensors in the* x *direction, we have*

$$K_1^\sharp = \begin{bmatrix} \frac{2ab}{a+b} & 0 \\ 0 & 0 \end{bmatrix}.$$

Repeat step 1 through 3 for tensors K_{III} *and* K_{IV} *, we get*

$$\begin{aligned} K_2^\sharp &= \bar{K} + \Delta K = K_0 + K_1 \\ &= \begin{bmatrix} \frac{c+d}{2} & 0 \\ 0 & \frac{c+d}{2} \end{bmatrix} - \begin{bmatrix} \frac{(d-c)^2}{2(c+d)} & 0 \\ 0 & \frac{(d-c)^2}{2(c+d)} \end{bmatrix} \\ &= \begin{bmatrix} \frac{2cd}{c+d} & 0 \\ 0 & \frac{2cd}{c+d} \end{bmatrix}. \end{aligned}$$

Again, we let $K_2^\sharp$ *be* 2×2 *tensors in the* x *direction, we have*

$$K_2^\sharp = \begin{bmatrix} \frac{2cd}{c+d} & 0 \\ 0 & 0 \end{bmatrix}.$$

Then, we compute the average of the two results in the x direction to obtain:

$$\begin{aligned} K_{12}^{\sharp} &= \frac{1}{2}(K_1^{\sharp} + K_2^{\sharp}) \\ &= \begin{bmatrix} \frac{ab}{a+b} + \frac{cd}{c+d} & 0 \\ 0 & 0 \end{bmatrix}. \end{aligned}$$

Also using the 2×2 tensors in the y direction, then we perform the same process as in the steps above in the following way:

- *Step 4: Compute the average of K_I, and K_{III}*

$$\bar{K} = K_0 = \frac{1}{2}(K_I + K_{III}) = \begin{bmatrix} \frac{a+c}{2} & 0 \\ 0 & \frac{a+c}{2} \end{bmatrix}$$

so,

$$K_0^{-1} = \begin{bmatrix} \frac{2}{a+c} & 0 \\ 0 & \frac{2}{a+c} \end{bmatrix}.$$

- *Step 5: Compute $\Delta K = K_1$ using results from step 1 and $K_{III} - K_0 = \begin{bmatrix} \frac{c-a}{2} & 0 \\ 0 & \frac{c-a}{2} \end{bmatrix}$*

$$K_1 = -\begin{bmatrix} \frac{c-a}{2} & 0 \\ 0 & \frac{c-a}{2} \end{bmatrix} \cdot \begin{bmatrix} \frac{a+c}{2} & 0 \\ 0 & \frac{a+c}{2} \end{bmatrix} \cdot \begin{bmatrix} \frac{c-a}{2} & 0 \\ 0 & \frac{c-a}{2} \end{bmatrix} = -\begin{bmatrix} \frac{(c-a)^2}{2(a+c)} & 0 \\ 0 & \frac{(c-a)^2}{2(a+c)} \end{bmatrix}.$$

- *Step 6: Add the results from step 4 and 5 together to get* $K_3^\sharp$

$$\begin{aligned} K_3^\sharp &= \bar{K} + \Delta K = K_0 + K_1 \\ &= \begin{bmatrix} \frac{a+c}{2} & 0 \\ 0 & \frac{a+c}{2} \end{bmatrix} - \begin{bmatrix} \frac{(c-a)^2}{2(a+c)} & 0 \\ 0 & \frac{(c-a)^2}{2(a+c)} \end{bmatrix} \\ &= \begin{bmatrix} \frac{2ac}{a+c} & 0 \\ 0 & \frac{2ac}{a+c} \end{bmatrix}, \end{aligned}$$

we let $K_3^\sharp$ *be* 2×2 *tensors in the* y *direction, so we have*

$$K_3^\sharp = \begin{bmatrix} 0 & 0 \\ 0 & \frac{2ac}{a+c} \end{bmatrix}.$$

Repeat step 4 through 6 for tensors K_{II} *and* K_{IV} *, we get*

$$\begin{aligned} K_4^\sharp &= \bar{K} + \Delta K = K_0 + K_1 \\ &= \begin{bmatrix} \frac{2ac}{a+c} & 0 \\ 0 & \frac{b+d}{2} \end{bmatrix} - \begin{bmatrix} \frac{(d-b)^2}{2(b+d)} & 0 \\ 0 & \frac{(d-b)^2}{2(b+d)} \end{bmatrix} \\ &= \begin{bmatrix} \frac{2bd}{b+d} & 0 \\ 0 & \frac{2bd}{b+d} \end{bmatrix}, \end{aligned}$$

again we let $K_4^\sharp$ *be* 2×2 *tensors in the* y *direction, so we have*

$$K_3^\sharp = \begin{bmatrix} 0 & 0 \\ 0 & \frac{2bd}{b+d} \end{bmatrix},$$

then, we compute the average of the two results in the y direction to obtain:

$$\begin{aligned} K_{34}^{\sharp} &= \frac{1}{2}(K_3^{\sharp} + K_4^{\sharp}) \\ &= \begin{bmatrix} 0 & 0 \\ 0 & \frac{ac}{a+c} + \frac{bd}{b+d} \end{bmatrix}, \end{aligned}$$

we then add $K_{12}^{\sharp}$ and $K_{34}^{\sharp}$ to get $K^{\sharp}$,

$$\begin{aligned} K^{\sharp} &= K_{12}^{\sharp} + K_{34}^{\sharp} \\ &= \begin{bmatrix} \frac{ab}{a+b} + \frac{cd}{c+d} & 0 \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \frac{ac}{a+c} + \frac{bd}{b+d} \end{bmatrix} \\ &= \begin{bmatrix} \frac{ab}{a+b} + \frac{cd}{c+d} & 0 \\ 0 & \frac{ac}{a+c} + \frac{bd}{b+d} \end{bmatrix}. \end{aligned}$$

If the given tensors represent a stratified medium, the average $K^{\sharp}$ is the harmonic average in the x direction and the y direction gives the arithmetic average. For instance, suppose we are given four 2×2 tensors of the form

$$K_I = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix}, \; K_{II} = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix}, \; K_{III} = \begin{bmatrix} a & 0 \\ 0 & a \end{bmatrix}, \; K_{IV} = \begin{bmatrix} b & 0 \\ 0 & b \end{bmatrix}.$$

We propose the fast transform for this special case by taking the same steps as in the previous case. The procedure are as follows:

Fast Transform Algorithm

- *Step 1: Compute the average of K_I, K_{II}, K_{III}, and K_{IV}*

$$\bar{K} = K_0 = \frac{1}{4}(K_I + K_{II} + K_{III} + K_{IV}) = \begin{bmatrix} \frac{2a+2b}{4} & 0 \\ 0 & \frac{2a+2b}{4} \end{bmatrix}$$

$$\bar{K} = K_0 = \begin{bmatrix} \frac{a+b}{2} & 0 \\ 0 & \frac{a+b}{2} \end{bmatrix}$$

so,

$$K_0^{-1} = \begin{bmatrix} \frac{2}{a+b} & 0 \\ 0 & \frac{2}{a+b} \end{bmatrix}.$$

- *Step 2: Compute* $\Delta K = K_1$ *using results from step 1 and* $K_I - K_0 = \begin{bmatrix} \frac{b-a}{2} & 0 \\ 0 & \frac{b-a}{2} \end{bmatrix}$

$$K_1 = -\begin{bmatrix} \frac{b-a}{2} & 0 \\ 0 & \frac{b-a}{2} \end{bmatrix} \cdot \begin{bmatrix} \frac{a+b}{2} & 0 \\ 0 & \frac{a+b}{2} \end{bmatrix} \cdot \begin{bmatrix} \frac{b-a}{2} & 0 \\ 0 & \frac{b-a}{2} \end{bmatrix} = -\begin{bmatrix} \frac{(b-a)^2}{2(a+b)} & 0 \\ 0 & \frac{(b-a)^2}{2(a+b)} \end{bmatrix}$$

So, we have

$$K_1 = -\begin{bmatrix} \frac{(b-a)^2}{2(a+b)} & 0 \\ 0 & 0 \end{bmatrix}.$$

- *Step 3: Add the results from step 1 and 2 together to get* $K^\sharp$

$$\begin{aligned} K^\sharp &= \bar{K} + \Delta K = K_0 + K_1 \\ &= \begin{bmatrix} \frac{a+b}{2} & 0 \\ 0 & \frac{a+b}{2} \end{bmatrix} - \begin{bmatrix} \frac{(b-a)^2}{2(a+b)} & 0 \\ 0 & 0 \end{bmatrix} \\ &= \begin{bmatrix} \frac{2ab}{a+b} & 0 \\ 0 & \frac{a+b}{2} \end{bmatrix}. \end{aligned}$$

4.3.1 The Inverse Fast Transform for Homogenized Coefficient Values

We can back calculate the tensors from the results of $K^\sharp$ and the details. With this in mind we can rewrite the fast transform in the following ways.

The Inverse Fast Transform Algorithm in Two Dimensions

- *Step1: Given that* $K^\sharp$ *is the sum of the average of the tensors and the details needed to compute* $K^\sharp$*, i.e.* $K^\sharp = \bar{K} + \Delta K$*, then, we can say that* $\Delta K = K^\sharp - \bar{K}$
- *Step 2: Compute the difference between the average and* K^{II} *by finding the square root of the diagonal entries in the product of details and average, then store the results in* K_1*.*
- *Step 3: From the above results, since the difference* $K_1 = K^{II} - \bar{K}$*, then* $K^{II} = K_1 + \bar{K}$*. So this gives us the second* 2×2 *tensors.*
- *Step 4: Since the tensors are of the form,* $K^I = K^{III}$ *and* $K^{II} = K^{IV}$ *then we can obtain the first tensor by using the average formula,*

$$\begin{aligned} \bar{K} &= \frac{1}{4}(K^I + K^{II} + K^{III} + K^{IV}) \\ &= \frac{1}{4}(K^I + K^{II} + K^I + K^{II}) \\ &= \frac{1}{4}(2K^I + 2K^{II}) \\ &= \frac{1}{2}(K^I + K^{II}). \end{aligned}$$

This implies $2\bar{K} = K^I + K^{II}$ *and from this we get* $K^I = 2\bar{K} - K^{II}$

On the other hand, if the tensors contain constant diagonals, we can also get back the tensors by rewriting the fast transform in the following ways.

- *Step 1: Given that* $K^\sharp$ *is the average of two different* $K^\sharp$ *i.e.* $K^\sharp = K^\sharp_{12} + K^\sharp_{34}$*, then, we can say that* $2K^\sharp - K^\sharp_{34} = K^\sharp_{12}$*.*
- *Step 2: Next we can now decompose* $K^\sharp_{12}$ *as follows,* $K^\sharp_{12} = K^\sharp_1 + K^\sharp_2$*.*
- *Step 3: Given* $K^\sharp_1 = \bar{K} + \Delta K$*, we can now perform the same process as in the previous section to recover two of the four tensors.*

- *Step 4: Compute the difference between the average and K^{II} by finding the square root of the diagonal entries in the product of details and average, then store the results in K_1.*

- *Step 5: From the above results, since the difference $K_1 = K^{II} - \bar{K}$, then $K^{II} = K_1 + \bar{K}$. So this gives us the second 2×2 tensors.*

- *Step 6: Since the tensors are non-stratified, $K^I \neq K^{III}$ and $K^{II} \neq K^{IV}$ then we can obtain the first tensor by using the average formula, $\bar{K} = \frac{1}{2}(K^I + K^{II})$ so, this implies $2\bar{K} = K^I + K^{II}$ and from this we get $K^I = 2\bar{K} - K^{II}$.*

- *Step 7: Repeat step 4 through step 6 for $K_2^\sharp$, from this we get the remaining two tensors, K^{III} and K^{IV} respectively.*

4.4 Homogenization Wavelet Reconstruction in Two Dimensions

We develop the two dimensional Homogenization Wavelet Reconstruction in this section. We follow the same process as in one dimensional case. We therefore base the reconstruction formula on the fundamental assumption that needs to be made in homogenization process. The homogenization process assumes that the primary variables h, from the original problem can be expanded in a perturbation series of the form:

$$h = h_0 + \epsilon h_1 + \epsilon^2 h_2 + \cdots + \epsilon^m h_m + \ldots \tag{4.15}$$

where ϵ is a small parameter, $0<\epsilon<<1$. We assumed that

$$h_1 = w_{0,i}^T \cdot \nabla h_0$$

in two dimensions. Using this assumption we can write:

$$
\begin{aligned}
h &\approx h_0 \\
h &\approx h_0 + \epsilon h_1 = h_0 + \epsilon w_{0,i}^T \cdot \nabla h_0 \\
&\vdots \\
h &\approx h_0 + \epsilon h_1 + \epsilon^2 h_2 + \cdots + \epsilon^m h_m = h_0 + \epsilon w_{0,i}^T \cdot \nabla h_0 + \epsilon^2 w_{1,i}^T \cdot \nabla h_1 + \ldots \\
&+ \epsilon^m w_{m-1,i} \cdot \nabla h_{m-1}
\end{aligned}
$$

where $i = 1, 2$.

Once we have determined the value of h_i, then we can use this to compute the next term in the perturbation expansion by computing the partial derivative of h_i with respect to x and y.

So, to reconstruct the pressure variable, h, we consider the following algorithm referred to as Homogenization-Wavelet Reconstruction algorithm in two dimensions. The algorithm is as follow:

HWR algorithm in two dimensions

- *Compute the solution of the DE for the coarsest level homogenized problem on the entire domain, that is compute the solution of*

$$\nabla \cdot K^\sharp \nabla h = 0$$
$$v = -K^\sharp \nabla h$$

  where $K^\sharp$ is the 2×2 matrix with harmonic average of the permeability in the x direction and the arithmetic average in the y direction.

- *Then use 4.15 and the ansatz*

$$h_1 = w_{0,1} \frac{\partial h_0}{dy_{0,1}} + w_{0,2} \frac{\partial h_0}{dy_{0,2}}.$$

In two dimensional case, we use the extension of this ansatz of the form

$$h_{l+1} = w_l^T(y_{l+1}) \cdot \nabla h_l.$$

So, we have

$h = h_0 + \epsilon \left(w_{0,1} \frac{\partial h_0}{dy_{0,1}} + w_{0,2} \frac{\partial h_0}{dy_{0,2}} \right) + \epsilon^2 \left(w_{1,1} \frac{\partial h_1}{dy_{1,1}} + w_{1,2} \frac{\partial h_1}{dy_{1,2}} \right) + \dots$
$+ \epsilon^m \left(w_{m-1,1} \frac{\partial h_{m-1}}{\partial y_{m-1,1}} + w_{m-1,2} \frac{\partial h_{m-1}}{\partial y_{m-1,2}} \right)$
where $\epsilon = \frac{1}{2}$.

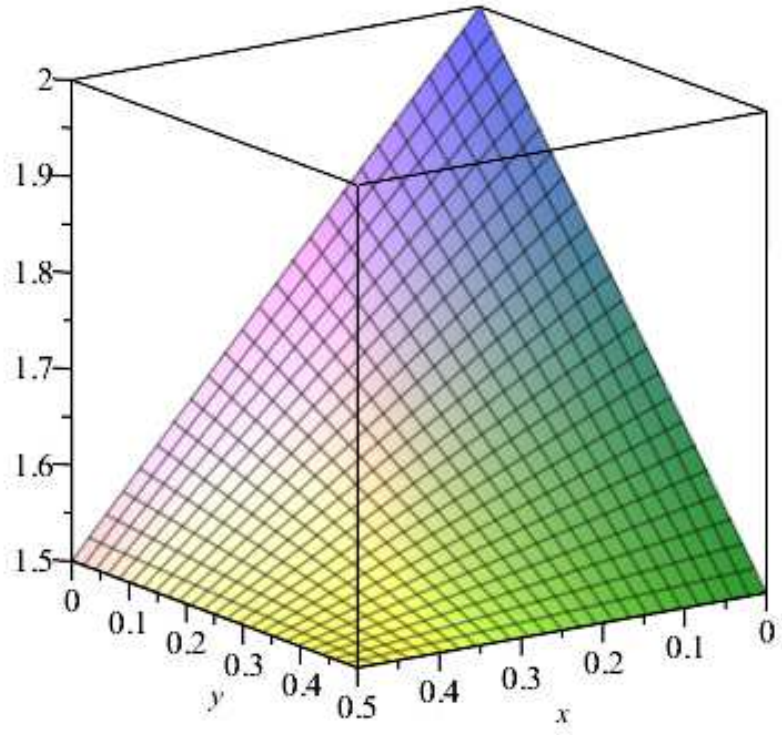

Figure 4.1. The graph of $h_0(x, y)$ on $(0, \frac{1}{2}) \times (0, \frac{1}{2})$

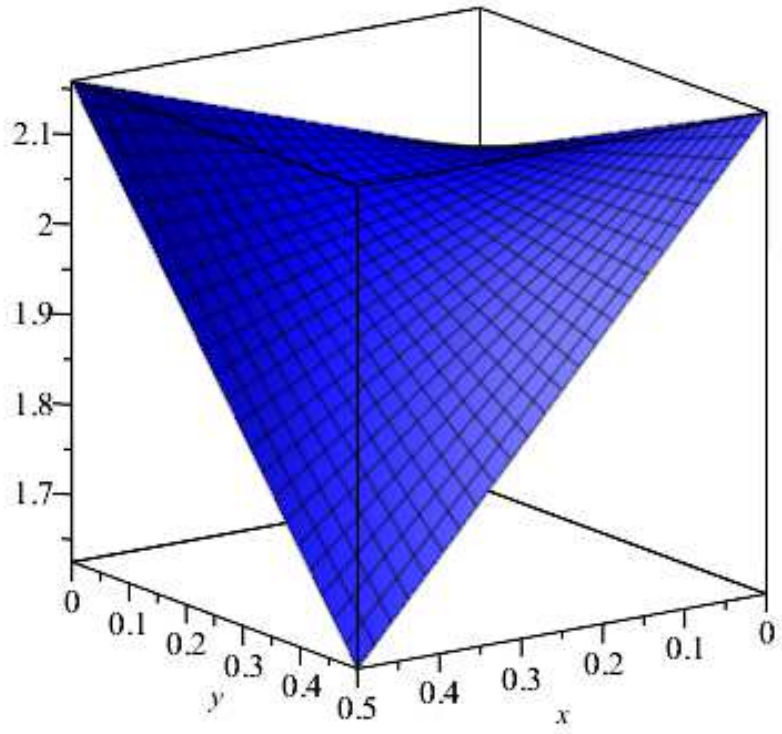

Figure 4.2. The graph of $h_1(x, y)$ on $(0, \frac{1}{2}) \times (0, \frac{1}{2})$

4.4.1 Some Numerical Results for HWR in Two Dimensions

We generate three dimensional figures that show the reconstruction of the pressure variable at various scales. We compute

$$h_0(x, y) = axy + bx + cy + d$$

where $a = 4(B_{11} + B_{00} - B_{10} - B_{01});\ b = 2(B_{10} - B_{00});\ c = 2(B_{01} - B_{00});\ and\ d = B_{00}$. And,

$$h_1 = h_0 + \epsilon h_1$$

for

$$h_1 = w_{0,1}\frac{\partial h_0}{dx} + w_{0,2}\frac{\partial h_0}{dy}.$$

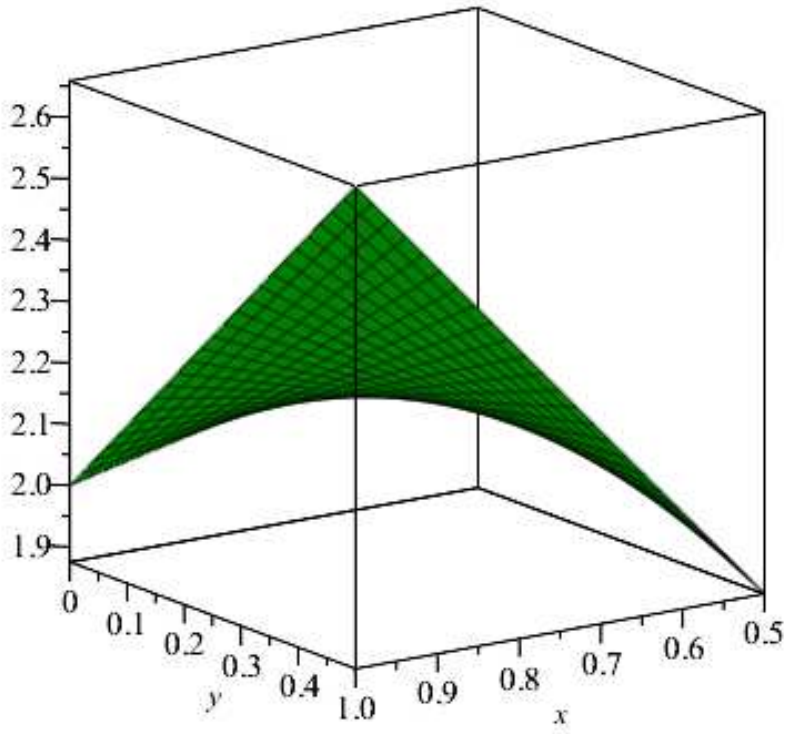

Figure 4.3. The graph of $h_1(x, y)$ on $(\frac{1}{2}, 1) \times (0, \frac{1}{2})$

The values of $w_{0,i}^T$ are obtained from the solution of the local problems. Figure 4.1 shows the reconstruction of the pressure variable, h_0 on interval $(0, \frac{1}{2}) \times (0, \frac{1}{2})$. Also, figures 4.2, 4.3,4.4 and 4.5 show the reconstruction of the pressure variable, h_1 on various intervals $h_1(x, y)$ on $(0, \frac{1}{2}) \times (0, \frac{1}{2}), h_1(x, y)$ on $(0, \frac{1}{2}) \times (\frac{1}{2}, 1)$, $h_1(x, y)$ on $(0, 1) \times (0, \frac{1}{2})$, and $h_1(x, y)$ on $(\frac{1}{2}, 1) \times (\frac{1}{2}, 1)$ respectively. In figure 4.6, we have h_1 that combine all the figures on various intervals together on interval $(0, 1) \times (0, 1)$.

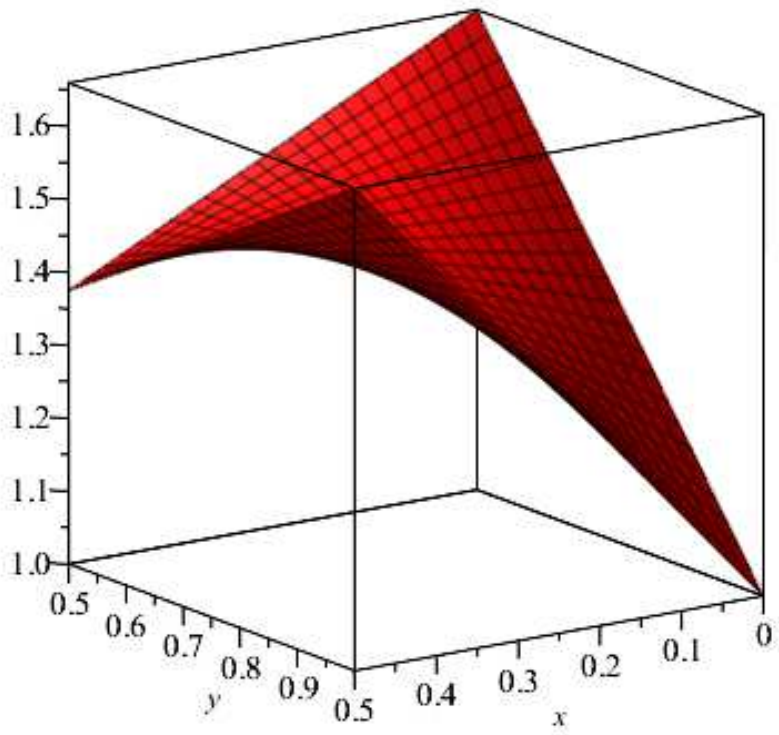

Figure 4.4. The graph of $h_1(x, y)$ on $(0, \frac{1}{2}) \times (\frac{1}{2}, 1)$

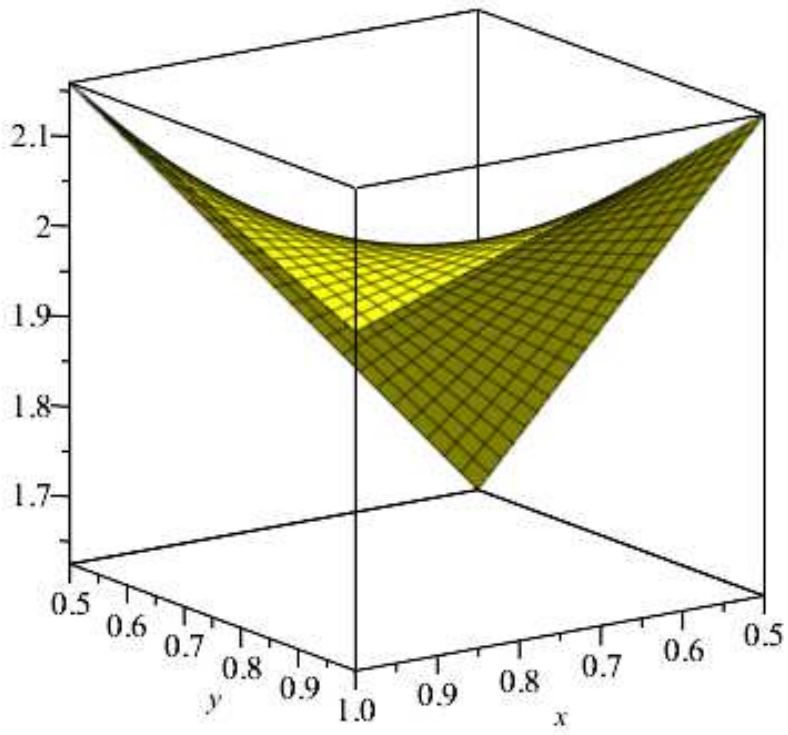

Figure 4.5. The graph of $h_1(x, y)$ on $(\frac{1}{2}, 1) \times (\frac{1}{2}, 1)$

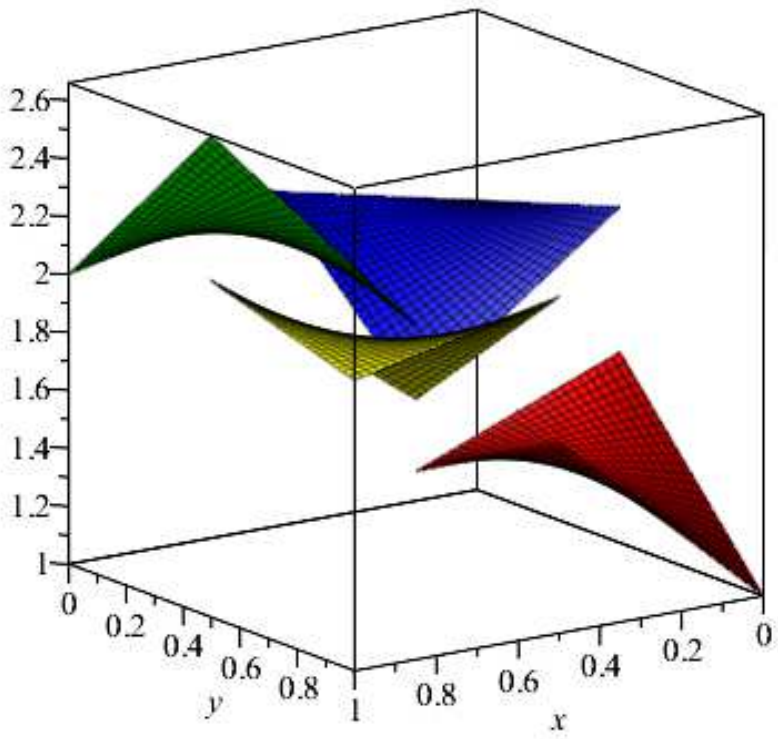

Figure 4.6. The graph of $h_1(x, y)$ on $(0, 1) \times (0, 1)$

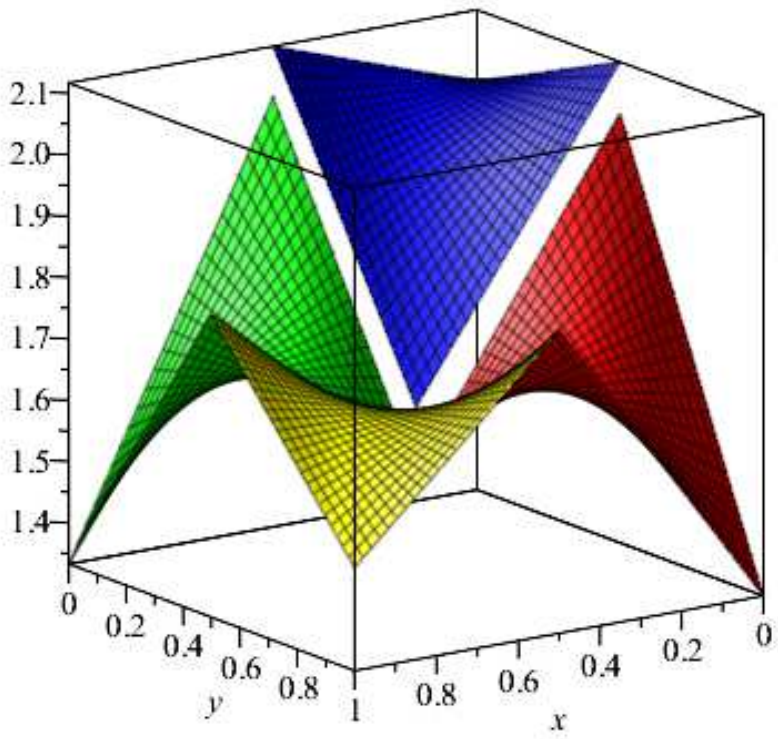

Figure 4.7. The graph of $h_2(x, y)$ on $(0, 1) \times (0, 1)$

CHAPTER 5

HOMOGENIZATION WAVELET RECONSTRUCTION IN TWO DIMENSIONS WITH FULL TENSORS

In this chapter, we return to the general problem of elliptic DE with full tensor. We outline symbolic solution of the local problem in two dimensions using full tensor and compute numerical example to illustrate the solution. Finally, we give numerical results from the implementation of our method of solutions.

5.1 The Local Problem in Two Dimensions with Full Tensors

We define the local problem in two dimensions by the system of elliptic equations as follows:

$$\nabla\cdot K\nabla w_i = -\nabla\cdot K\vec{e_i} \tag{5.1}$$

for $i = 1, 2$, where w_i is the Jacobian matrix of the functions, and $\vec{e_i}$ is the unit vector in $\mathbb{R}^2$ with the following permeability definitions

$$K_{(y_1,y_2)} = \begin{cases} K^I & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ K^{II} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ K^{III} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ K^{IV} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right], \end{cases}$$

and let K and K^{-1} be 2×2 matrices of the form with entries $K_{xy} = K_{yx}$ and $a_{12} = a_{21}$:

$$K = \begin{bmatrix} K_{xx} & K_{xy} \\ K_{xy} & K_{yy} \end{bmatrix},$$

$$K^{-1} = \begin{bmatrix} a_{11} & a_{12} \\ a_{12} & a_{22} \end{bmatrix}.$$

5.1.1 The Solution of the Elliptic Problems with Full Tensors

The Jacobian matrix, J , contains the solution of the system of elliptic problems. We need to write out appropriate formula to compute the details of the solutions and do the necessary integration. We provide approximate solutions to the systems of elliptic partial differential equations in two cell problem as described below.

Given the local problem in two dimension:

$$\nabla \cdot K \nabla w_i = -\nabla \cdot K \vec{e_i}.$$

Rewriting the equation as a homogeneous equation, we have: for i = 1

$$\nabla \cdot K \nabla w_1 + \nabla \cdot K \vec{e_1} = 0. \tag{5.2}$$

Integrating the above equation, we have

$$K(\nabla w_1 + \vec{e_1}) = \begin{cases} c_1 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ c_2 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ c_3 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ c_4 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right]. \end{cases}$$

Then, multiply both sides by K^{-1},

$$\nabla w_1 + \vec{e_1} = \begin{cases} K_I^{-1} c_1 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ K_{II}^{-1} c_2 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ K_{III}^{-1} c_3 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ K_{IV}^{-1} c_4 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right] \end{cases}$$

$$\nabla w_1 = \begin{cases} K_I^{-1} c_1 - \vec{e_1} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ K_{II}^{-1} c_2 - \vec{e_1} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ K_{III}^{-1} c_3 - \vec{e_1} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ K_{IV}^{-1} c_4 - \vec{e_1} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right]. \end{cases} \tag{5.3}$$

Integrating the above equation with respect to y, we obtain

$$w_1 = \begin{cases} (K_I^{-1} c_1 - \vec{e_1}) \cdot [y_1, y_2] & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ (K_{II}^{-1} c_2 - \vec{e_1}) \cdot [y_1 - 1, y_2] & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ (K_{III}^{-1} c_3 - \vec{e_1}) \cdot [y_1, y_2 - 1] & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ (K_{IV}^{-1} c_4 - \vec{e_1}) \cdot [y_1 - 1, y_2 - 1] & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right] \end{cases}$$

and then make assumptions on the continuity at some points. This allows us to reduce the problem with eight unknowns to four equations with four unknowns. We set the following four conditions on each of the step functions with $c_{11} = c_{21}, c_{12} = c_{32}, c_{22} = c_{42}$, and $c_{31} = c_{41}$.

To solve this system of equations, we let $y_1 = \frac{1}{2}$, and $y_2 \in (0, y_2]$,

$$w_1 = \begin{cases} (a_{11}^I c_{11} + a_{12}^I c_{12} - 1)y_1 + (a_{12}^I c_{11} + a_{22}^I c_{12})y_2 \\ (a_{11}^{II} c_{21} + a_{12}^{II} c_{22} - 1)(y_1 - 1) + (a_{12}^{II} c_{21} + a_{22}^{II} c_{22})y_2 \\ (a_{11}^{II} c_{31} + a_{12}^{III} c_{32} - 1)y_1 + (a_{12}^{III} c_{31} + a_{22}^{III} c_{32})(y_2 - 1) \\ (a_{11}^{IV} c_{41} + a_{12}^{IV} c_{42} - 1)(y_1 - 1) + (a_{12}^{IV} c_{41} + a_{22}^{IV} c_{42})(y_2 - 1). \end{cases}$$

From this, we obtain the following system of equations:

$$\begin{aligned} (a_{11}^I c_{11} + a_{12}^I c_{12} - 1)(\frac{1}{2}) &= (a_{11}^{II} c_{21} + a_{12}^{II} c_{22} - 1)(-\frac{1}{2}) \\ (a_{12}^I c_{11} + a_{22}^I c_{12}) &= (a_{12}^{II} c_{21} + a_{22}^{II} c_{22}) \\ (a_{11}^{II} c_{31} + a_{12}^{III} c_{32} - 1)(\frac{1}{2}) &= (a_{11}^{IV} c_{41} + a_{12}^{IV} c_{42} - 1)(-\frac{1}{2}) \\ (a_{12}^{III} c_{31} + a_{22}^{III} c_{32})(y_2 - 1) &= (a_{12}^{IV} c_{41} + a_{22}^{IV} c_{42}). \end{aligned}$$

Then, we solve for c_{11} and c_{31} in terms of c_{12} and c_{22}

$$c_{11} = \frac{2}{a_{11}^I + a_{11}^{II}} - \frac{a_{12}^I c_{12}}{a_{11}^I + a_{11}^{II}} - \frac{a_{12}^{II} c_{22}}{a_{11}^I + a_{11}^{II}} \tag{5.4}$$

$$c_{11} = \frac{a_{22}^{II} c_{22}}{a_{12}^I - a_{12}^{II}} - \frac{a_{22}^I c_{12}}{a_{12}^I - a_{12}^{II}} \tag{5.5}$$

$$c_{31} = \frac{2}{a_{11}^{III} + a_{11}^{IV}} - \frac{a_{12}^{III} c_{12}}{a_{11}^{III} + a_{11}^{IV}} - \frac{a_{12}^{IV} c_{22}}{a_{11}^{III} + a_{11}^{IV}} \tag{5.6}$$

$$c_{31} = \frac{a_{22}^{IV} c_{22}}{a_{12}^{III} - a_{12}^{IV}} - \frac{a_{22}^{III} c_{12}}{a_{12}^{III} - a_{12}^{IV}}. \tag{5.7}$$

Simplifying these equations, we get

$$\left(\frac{a_{22}^{II}}{a_{12}^I - a_{12}^{II}} + \frac{a_{12}^{II}}{a_{11}^I + a_{11}^{II}}\right) c_{22} + \left(\frac{a_{12}^I}{a_{11}^I + a_{11}^{II}} - \frac{a_{22}^I}{a_{12}^I - a_{12}^{II}}\right) c_{12} = \frac{2}{a_{11}^I + a_{11}^{II}}$$

$$\left(\frac{a_{22}^{IV}}{a_{12}^{III} - a_{12}^{IV}} + \frac{a_{12}^{IV}}{a_{11}^{III} + a_{11}^{IV}}\right) c_{22} + \left(\frac{a_{12}^{III}}{a_{11}^{III} + a_{11}^{IV}} - \frac{a_{22}^{III}}{a_{12}^{III} - a_{12}^{IV}}\right) c_{12} = \frac{2}{a_{11}^{III} + a_{11}^{IV}}.$$

These equations in matrix form are as follow:

$$\begin{bmatrix} \left(\frac{a_{12}^{I}}{a_{11}^{I}+a_{11}^{II}} - \frac{a_{22}^{I}}{a_{12}^{I}-a_{12}^{II}}\right) & \left(\frac{a_{12}^{II}}{a_{11}^{I}+a_{11}^{II}} + \frac{a_{22}^{II}}{a_{12}^{I}-a_{12}^{II}}\right) \\ \left(\frac{a_{12}^{III}}{a_{11}^{III}+a_{11}^{IV}} - \frac{a_{22}^{III}}{a_{12}^{III}-a_{12}^{IV}}\right) & \left(\frac{a_{12}^{IV}}{a_{11}^{III}+a_{11}^{IV}} + \frac{a_{22}^{IV}}{a_{12}^{III}-a_{12}^{IV}}\right) \end{bmatrix} \cdot \begin{bmatrix} c_{12} \\ c_{22} \end{bmatrix} = \begin{bmatrix} \frac{2}{a_{11}^{I}+a_{11}^{II}} \\ \frac{2}{a_{11}^{III}+a_{11}^{IV}} \end{bmatrix}.$$

We then solve the 2×2 system of equations to get c_{12} and c_{22}. These results are used to obtain the values for c_{11} and c_{31} using Equations 5.5 and 5.7. So we substitute these values into Equation 5.3, to get the value of $\bigtriangledown w_1$.

For i = 2

$$\nabla \cdot K \nabla w_2 + \nabla \cdot K \vec{e_2} = 0. \tag{5.8}$$

Integrating the above equation, we have

$$K(\nabla w_2 + \vec{e_2}) = \begin{cases} c_1 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ c_2 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ c_3 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ c_4 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right]. \end{cases}$$

Then, we multiply both sides by K^{-1},

$$\nabla w_2 + \vec{e_2} = \begin{cases} K_I^{-1} c_1 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ K_{II}^{-1} c_2 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ K_{III}^{-1} c_3 & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ K_{IV}^{-1} c_4 & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right], \end{cases}$$

$$\nabla w_2 = \begin{cases} K_I^{-1}c_1 - \vec{e_2} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ K_{II}^{-1}c_2 - \vec{e_2} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ K_{III}^{-1}c_3 - \vec{e_2} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ K_{IV}^{-1}c_4 - \vec{e_2} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right]. \end{cases} \tag{5.9}$$

Integrating the above equation with respect to y, we obtain

$$w_2 = \begin{cases} (K_I^{-1}c_1 - \vec{e_2})(y_1, y_2) & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ (K_{II}^{-1}c_2 - \vec{e_2})(y_1 - 1, y_2) & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ (K_{III}^{-1}c_3 - \vec{e_2})(y_1, y_2 - 1) & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ (K_{IV}^{-1}c_4 - \vec{e_2})(y_1 - 1, y_2 - 1) & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right] \end{cases}$$

and then make assumptions on the continuity at some points as we did in computing w_1. This allows us to reduce the problem with eight unknowns to four equations with four unknowns. We set the following four conditions on each of the step functions with $c_{11} = c_{21}, c_{12} = c_{32}, c_{22} = c_{42}$, and $c_{31} = c_{41}$.

To solve this system of equations, we let $y_2 = \frac{1}{2}$, and $y_1 \in (0, y_1]$,

$$w_2 = \begin{cases} (a_{11}^I c_{11} + a_{12}^I c_{12})y_1 + (a_{12}^I c_{11} + a_{22}^I c_{12} - 1)y_2 \\ (a_{11}^{II} c_{21} + a_{12}^{II} c_{22})(y_1 - 1) + (a_{12}^{II} c_{21} + a_{22}^{II} c_{22} - 1)y_2 \\ (a_{11}^{II} c_{31} + a_{12}^{III} c_{32})y_1 + (a_{12}^{III} c_{31} + a_{22}^{III} c_{32} - 1)(y_2 - 1) \\ (a_{11}^{IV} c_{41} + a_{12}^{IV} c_{42})(y_1 - 1) + (a_{12}^{IV} c_{41} + a_{22}^{IV} c_{42} - 1)(y_2 - 1). \end{cases}$$

From this, we obtain the following system of equations:

$$(a_{12}^{I}c_{11} + a_{22}^{I}c_{12} - 1)\left(\frac{1}{2}\right) = (a_{12}^{III}c_{31} + a_{22}^{III}c_{12} - 1)\left(-\frac{1}{2}\right)$$
$$(a_{11}^{I}c_{11} + a_{12}^{I}c_{12}) = (a_{11}^{III}c_{31} + a_{12}^{III}c_{12})$$
$$(a_{12}^{II}c_{11} + a_{22}^{II}c_{212} - 1)\left(\frac{1}{2}\right) = (a_{12}^{IV}c_{31} + a_{22}^{IV}c_{22} - 1)\left(-\frac{1}{2}\right)$$
$$(a_{11}^{II}c_{11} + a_{12}^{II}c_{22}) = (a_{11}^{IV}c_{31} + a_{12}^{IV}c_{22}).$$

Then, we solve for c_{12} and c_{22} in terms of c_{11} and c_{31}

$$c_{12} = \frac{2}{a_{22}^{I} + a_{22}^{III}} - \frac{a_{12}^{I}c_{11}}{a_{22}^{I} + a_{22}^{III}} - \frac{a_{12}^{III}c_{31}}{a_{22}^{I} + a_{22}^{III}}; \tag{5.10}$$
$$c_{12} = \frac{a_{11}^{III}c_{31}}{a_{12}^{I} - a_{12}^{III}} - \frac{a_{11}^{I}c_{11}}{a_{12}^{I} - a_{12}^{III}}; \tag{5.11}$$
$$c_{22} = \frac{2}{a_{22}^{II} + a_{22}^{IV}} - \frac{a_{12}^{IV}c_{31}}{a_{22}^{II} + a_{22}^{IV}} - \frac{a_{12}^{II}c_{11}}{a_{22}^{II} + a_{22}^{IV}}; \tag{5.12}$$
$$c_{22} = \frac{a_{11}^{IV}c_{31}}{a_{12}^{II} - a_{12}^{IV}} - \frac{a_{11}^{II}c_{11}}{a_{12}^{II} - a_{12}^{IV}}. \tag{5.13}$$

Simplifying these equations, we obtain

$$\left(\frac{a_{11}^{III}}{a_{12}^{I} - a_{12}^{III}} + \frac{a_{12}^{III}}{a_{22}^{I} + a_{22}^{III}}\right)c_{31} + \left(\frac{a_{12}^{I}}{a_{22}^{I} + a_{22}^{III}} - \frac{a_{11}^{I}}{a_{12}^{I} - a_{12}^{IV}}\right)c_{11} = \frac{2}{a_{22}^{I} + a_{22}^{III}},$$
$$\left(\frac{a_{11}^{IV}}{a_{12}^{II} - a_{12}^{IV}} + \frac{a_{12}^{IV}}{a_{22}^{II} + a_{22}^{IV}}\right)c_{31} + \left(\frac{a_{12}^{II}}{a_{22}^{II} + a_{22}^{IV}} - \frac{a_{11}^{II}}{a_{12}^{II} - a_{12}^{IV}}\right)c_{11} = \frac{2}{a_{22}^{II} + a_{22}^{IV}}.$$

These equations in matrix form are as follow:

$$\begin{bmatrix} \left(\frac{a_{12}^{I}}{a_{22}^{I}+a_{22}^{III}} - \frac{a_{11}^{I}}{a_{12}^{I}-a_{12}^{IV}}\right) & \left(\frac{a_{12}^{III}}{a_{22}^{I}+a_{22}^{III}} + \frac{a_{11}^{III}}{a_{12}^{I}-a_{12}^{III}}\right) \\ \left(\frac{a_{12}^{II}}{a_{22}^{II}+a_{22}^{IV}} - \frac{a_{11}^{II}}{a_{12}^{II}-a_{12}^{IV}}\right) & \left(\frac{a_{12}^{IV}}{a_{22}^{II}+a_{22}^{IV}} + \frac{a_{11}^{IV}}{a_{12}^{II}-a_{12}^{IV}}\right) \end{bmatrix} \cdot \begin{bmatrix} c_{11} \\ c_{31} \end{bmatrix} = \begin{bmatrix} \frac{2}{a_{22}^{I}+a_{22}^{III}} \\ \frac{2}{a_{22}^{II}+a_{22}^{IV}} \end{bmatrix}.$$

We then solve the 2×2 system of equations to obtain c_{11} and c_{31}. These results are

used to obtain the values of c_{12} and c_{22} using Equations 5.11 and 5.13. So we substitute these values into Equation 5.9, to get the $\bigtriangledown w_2$.

With this, we can compute the averaging formula, $K^\sharp$

$$K^\sharp = \int K(I + J^T) d\Omega,$$

where

$$J^T = \nabla w_i = \begin{bmatrix} \frac{\partial w_1}{\partial y_1} & \frac{\partial w_2}{\partial y_1} \\ \frac{\partial w_1}{\partial y_2} & \frac{\partial w_2}{\partial y_2} \end{bmatrix}.$$

5.1.2 An Example of Computing $K^\sharp$ for Full Tensors

Given that

$$K_I = \begin{bmatrix} 1 & 1 \\ 1 & 10 \end{bmatrix},\ K_{II} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix},\ K_{III} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix},\ K_{IV} = \begin{bmatrix} 10 & 1 \\ 1 & 1 \end{bmatrix}.$$

We compute their inverses and get:

$$K_I^{-1} = \begin{bmatrix} \frac{10}{9} & -\frac{1}{9} \\ -\frac{1}{9} & \frac{1}{9} \end{bmatrix},\ K_{II}^{-1} = \begin{bmatrix} -\frac{1}{3} & \frac{2}{3} \\ \frac{2}{3} & -\frac{1}{3} \end{bmatrix},\ K_{III}^{-1} = \begin{bmatrix} \frac{2}{3} & -\frac{1}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{bmatrix},\ K_{IV}^{-1} = \begin{bmatrix} \frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{9} & \frac{10}{9} \end{bmatrix}.$$

Using the result of the inverse matrix, we compute the values of the constants c_{11} to c_{42} by solving the system of equations:

$$\begin{bmatrix} 0 & \frac{9}{7} \\ \frac{18}{7} & -\frac{36}{7} \end{bmatrix} \cdot \begin{bmatrix} c_{12} \\ c_{22} \end{bmatrix} = \begin{bmatrix} \frac{18}{7} \\ \frac{18}{7} \end{bmatrix}.$$

From this, we obtain

$$c_{12} = c_{32} = 2,$$

$$c_{22} = c_{42} = 5,$$

$$c_{11} = c_{21} = 2.43,$$

$$c_{31} = c_{41} = -19.$$

Then, we compute the Jacobian of w_1, ∇w_1, using the following equation

$$\nabla w_1 = \begin{cases} K_I^{-1}c_1 - \vec{e_1} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ K_{II}^{-1}c_2 - \vec{e_1} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ K_{III}^{-1}c_3 - \vec{e_1} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ K_{IV}^{-1}c_4 - \vec{e_1} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right] \end{cases} \tag{5.14}$$

we get

$$\nabla w_1 = \begin{cases} \begin{bmatrix} 1.48 \\ -0.05 \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} 1.52 \\ -0.05 \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} -14.33 \\ 7.67 \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ \begin{bmatrix} -3.67 \\ 7.67 \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right] \end{cases}$$

and to compute ∇w_2 we compute the values of the constants as in above and we change the unit vector $\vec{e_1}$ to $\vec{e_2}$

$$\begin{bmatrix} \frac{18}{7} & -\frac{36}{7} \\ 0 & \frac{9}{7} \end{bmatrix} \cdot \begin{bmatrix} c_{31} \\ c_{11} \end{bmatrix} = \begin{bmatrix} \frac{18}{7} \\ \frac{18}{7} \end{bmatrix}.$$

Solving this system of equations, we get

$$c_{12} = c_{32} = 5,$$

$$c_{22} = c_{42} = \frac{11}{7},$$

$$c_{11} = c_{21} = 2,$$

$$c_{31} = c_{41} = 5$$

.

We then put these results back into the equation:

$$\nabla w_2 = \begin{cases} (K_I^{-1} c_1 - \vec{e_2}) & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ (K_{II}^{-1} c_2 - \vec{e_2}) & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ (K_{III}^{-1} c_3 - \vec{e_2}) & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ (K_{IV}^{-1} c_4 - \vec{e_2}) & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right], \end{cases}$$

so we have:

$$\nabla w_2 = \begin{cases} \begin{bmatrix} 1.67 \\ -0.67 \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} 0.38 \\ -0.19 \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[0, \frac{1}{2}\right] \\ \begin{bmatrix} 1.67 \\ 0.67 \end{bmatrix} & (y_1, y_2) \in \left[0, \frac{1}{2}\right] \times \left[\frac{1}{2}, 1\right] \\ \begin{bmatrix} 0.38 \\ 0.19 \end{bmatrix} & (y_1, y_2) \in \left[\frac{1}{2}, 1\right] \times \left[\frac{1}{2}, 1\right]. \end{cases}$$

We then compute the product of the tensors and the transpose of the Jacobian,

$$K_I \cdot J^T = \begin{bmatrix} 1.43 & 1 \\ 0.98 & -5.03 \end{bmatrix}, \; K_{II} \cdot J^T = \begin{bmatrix} 1.27 & 0 \\ 2.99 & 0.57 \end{bmatrix},$$

$$K_{III} \cdot J^T = \begin{bmatrix} -20.99 & 4.01 \\ 1.01 & 3.01 \end{bmatrix}, \; K_{IV} \cdot J^T = \begin{bmatrix} -29.03 & 3.99 \\ 4 & 0.57 \end{bmatrix}.$$

The next steep is to compute the $K^\sharp$, on the entire domain,

$$\begin{aligned} K^\sharp &= \int K(I + J^T) d\Omega \\ &= \int K d\Omega + \int K J^T d\Omega, \end{aligned}$$

where

$$\int K d\Omega = \begin{bmatrix} 3.5 & 1.25 \\ 1.25 & 3.5 \end{bmatrix},$$

and

$$\int KJ^T d\Omega = \begin{bmatrix} -11.83 & 2.25 \\ 2.25 & -0.22 \end{bmatrix}.$$

Therefore,

$$K^\sharp = \begin{bmatrix} 3.5 & 1.25 \\ 1.25 & 3.5 \end{bmatrix} + \begin{bmatrix} -11.83 & 2.25 \\ 2.25 & -0.22 \end{bmatrix} = \begin{bmatrix} -8.33 & 3.5 \\ 3.5 & 3.28 \end{bmatrix}.$$

5.1.3 The Local Problem in Two Dimensions with Diagonal Tensors

In order to diagonalize a real symmetric tensor, we begin by building an orthogonal matrix from an orthonormal basis of eigenvectors, as in the procedure below. The symmetric matrix:

$$K = \begin{bmatrix} K_{xx} & K_{xy} \\ K_{xy} & K_{yy} \end{bmatrix}$$

has eigenvalues

$$\lambda_1 = \frac{K_{xx} + K_{yy}}{2} + \frac{\sqrt{(K_{xx} - K_{yy})^2 + 4K_{xy}^2}}{2}$$

and

$$\lambda_2 = \frac{K_{xx} + K_{yy}}{2} - \frac{\sqrt{(K_{xx} - K_{yy})^2 + 4K_{xy}^2}}{2}$$

with eigenvectors

$$u = \begin{bmatrix} \frac{-2K_{xy}}{(K_{xx}-K_{yy})-\sqrt{(K_{xx}-K_{yy})^2+4K_{xy}^2}} \\ 1 \end{bmatrix}$$

and

$$v = \begin{bmatrix} \frac{-2K_{xy}}{(K_{xx}-K_{yy})+\sqrt{(K_{xx}-K_{yy})^2+4K_{xy}^2}} \\ 1 \end{bmatrix}$$

respectively. After normalizing these eigenvectors, we build the orthogonal matrix, P such that $P^TP = I$. So $D = P^TKP$, where D is the diagonalized form of K and P the associated

change-of-basis matrix from the standard basis to the basis of eigenvectors:

$$D = \begin{bmatrix} \frac{K_{xx}+K_{yy}}{2} + \frac{\sqrt{(K_{xx}-K_{yy})^2+4K_{xy}^2}}{2} & 0 \\ 0 & \frac{K_{xx}+K_{yy}}{2} - \frac{\sqrt{(K_{xx}-K_{yy})^2+4K_{xy}^2}}{2} \end{bmatrix}.$$

So

$$D^{-1} = \begin{bmatrix} \frac{2}{(K_{xx}+K_{yy})+\sqrt{(K_{xx}-K_{yy})^2+4K_{xy}^2}} & 0 \\ 0 & \frac{2}{(K_{xx}+K_{yy})-\sqrt{(K_{xx}-K_{yy})^2+4K_{xy}^2}} \end{bmatrix}.$$

5.2 Numerical Computations of the Permeability in Two Dimensions

This section is devoted to numerical examples illustrating the above methods. We simulate a fluid flow in a porous media, such as oil reservoir, and solve the problem using the brute force method. This method approximates the unique solution of partial differential equation. The numerical results provide the same results as the solutions computed with hand. The homogenized results in the x direction is the harmonic average and the results in the y direction is the arithmetic average.

5.2.1 Numerical Example 1

We consider the local problem in two dimensions:

$$\nabla \cdot K \nabla w_i = -\nabla \cdot K \vec{e_i}$$

with the permeability tensors of the form (stratified matrix):

$$K_I = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix},\ K_{II} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ K_{III} = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix},\ K_{IV} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}.$$

We solve the problem numerically using the method we explained in the previous section,the results obtain from the Java code are as follows:

```
run:
```

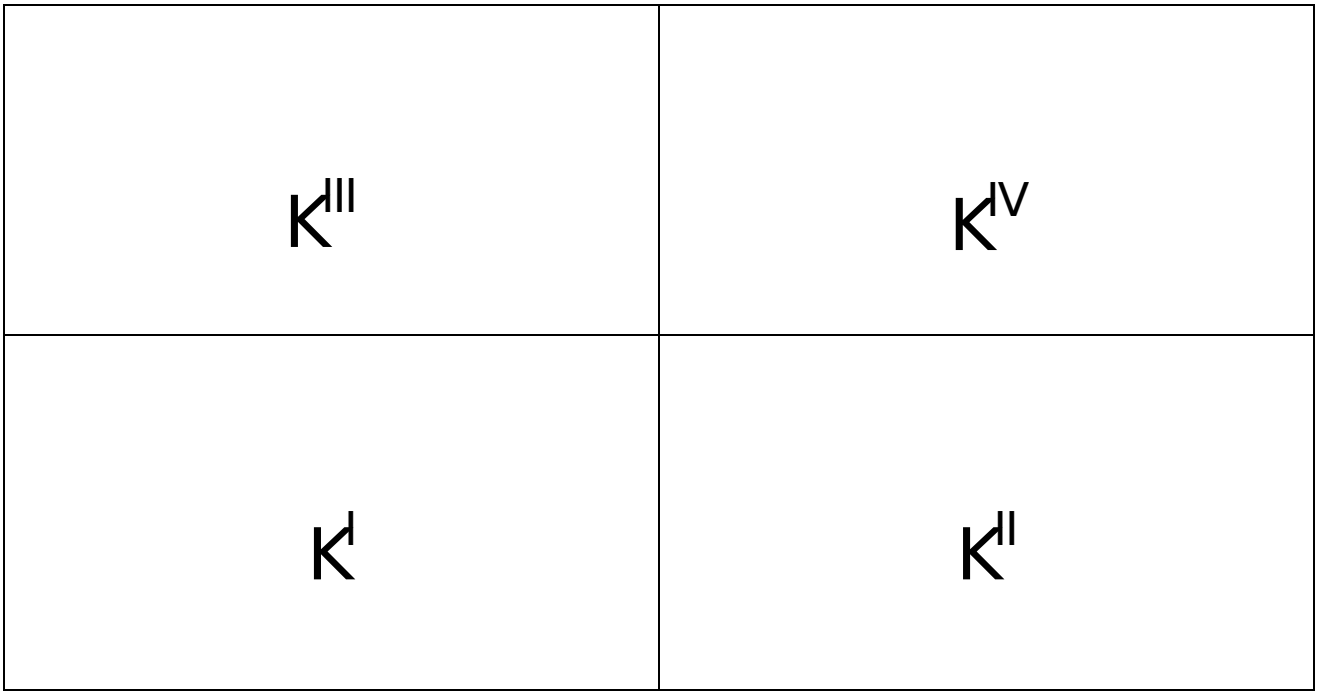

Figure 5.1. The Permeability Tensors (K) values

```
K-Pound Matrix - Diagonal Tensor
1.3333333333333333  0.0
0.0                 1.5
BUILD SUCCESSFUL (total time: 1 second)
```

The result produced is the same with the analytical solution.

5.2.2 Numerical Example 2

We also implement a Java code for a simple example, we as well have the same result as the solution computed with hand. The result is as follow: Given

$$K_I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ K_{II} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix},\ K_{III} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix},\ K_{IV} = \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix}.$$

```
run:


K-Pound Matrix - Diagonal Tensor
1.8181818181818183 0.0
0.0                5.5
BUILD SUCCESSFUL (total time: 1 second)
```

5.2.3 Numerical Example 3

Using a full tensor with

$$K_I = \begin{bmatrix} 1 & 1 \\ 1 & 10 \end{bmatrix},\ K_{II} = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix},\ K_{III} = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix},\ K_{IV} = \begin{bmatrix} 10 & 1 \\ 1 & 1 \end{bmatrix}.$$

We then apply our method of computation, the results still behave well and we have the same results as the one we computed by hand.

```
run:


K-Pound Matrix-Full Tensor
-8.28571428571429 3.5
3.5000000000000004 3.285714285714287
BUILD SUCCESSFUL (total time: 1 second)
```

5.2.4 Numerical Example 4

In the fast transform algorithm presented in 4, we implement a Java code for the procedure solving the examples as those presented in 5.2.1 and 5.2.2. The code produce the same results as in the previous examples. The results are as follows:

```
run:
Input Matrix - Diagonal Tensor
2.0 0.0  1.0 0.0
0.0 2.0  0.0 1.0

2.0 0.0  1.0 0.0
0.0 2.0  0.0 1.0

Result Matrix - Fast Transform Algorithm
1.3333333333333333 0.0
0.0                1.5
BUILD SUCCESSFUL (total time: 0 seconds)



run:
Input Matrix - Diagonal Tensor
1.0 0.0   10.0 0.0
0.0 1.0   0.0  10.0

1.0 0.0   10.0 0.0
0.0 1.0   0.0 10.0
```

```
Result Matrix - Fast Transform Algorithm
1.818181818181818 0.0
0.0                5.5
BUILD SUCCESSFUL (total time: 1 second)
```

5.2.5 Numerical Example 5

This example numerically generates the inverse transform of the fast transform algorithm in two dimensions. We start with $K^{\sharp}$ and regenerate the permeability tensors. We use the same examples to be consistent with our computation, the results generated in this procedure are given below.

```
run:

Result - k0
1.0 0.0
0.0 1.0

Result - k1
2.0 0.0
0.0 2.0

Result - k2
1.0 0.0
0.0 1.0

Result - k3
2.0 0.0
0.0 2.0
```

```
Inverse Transform Result Matrix - Diagonal Tensor
1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0
BUILD SUCCESSFUL (total time: 1 second)


run:

Result - k0
1.0 0.0
0.0 1.0

Result - k1
10.0 0.0
0.0 10.0

Result - k0
1.0 0.0
0.0 1.0

Result - k1
10.0 0.0
0.0 10.0
```

```
Inverse Transform Result Matrix - Diagonal Tensor
1.0 0.0  10.0 0.0
0.0 1.0  0.0 10.0

1.0 0.0  10.0 0.0
0.0 1.0  0.0 10.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

5.3 Weak Solution of the Full Tensor Local Elliptic Problem

In this section, we provide another method for computing the solution of the local elliptic problem with full tensors. The solution obtain in this process is then used in the reconstruction algorithm to compute the pressure variable. Required to compute the solution of the following elliptic problem

$$\nabla\cdot K^{\sharp}\nabla h = 0$$

with some linear boundary conditions.

Using the idea of [36], we will decompose the tensor as follows:

$$K^{\sharp} = \begin{bmatrix} K^{\sharp}_{xx} & 0 \\ 0 & K^{\sharp}_{yy} \end{bmatrix} + \begin{bmatrix} 0 & K^{\sharp}_{xy} \\ K^{\sharp}_{xy} & 0 \end{bmatrix} = K^{\sharp}_{d} + K^{\sharp}_{n}.$$

Also, we need to write the PDE as a system of two first order PDEs using the Darcy velocity. The reduced order system of equations is as follows:

$$v = -K^{\sharp}\nabla h,$$

$$-\nabla v = 0.$$

Using the decomposed tensor, the Darcy velocity can be written as:

$$\begin{aligned} v &= -(K_d^\sharp + K_n^\sharp)\nabla h \\ &= -K_d^\sharp \nabla h - K_n^\sharp \nabla h \\ &= v_d + v_n \end{aligned}$$

where $v_d = -K_d^\sharp \nabla h$ and $v_n = -K_n^\sharp \nabla h$.

Now, let us look at $K_n^\sharp$, we can write

$$-\begin{bmatrix} 0 & K_{xy}^\sharp \\ K_{xy}^\sharp & 0 \end{bmatrix} \cdot \begin{bmatrix} \frac{\partial h}{\partial x} \\ \frac{\partial h}{\partial y} \end{bmatrix} = -\begin{bmatrix} K_{xy}^\sharp \frac{\partial h}{\partial y} \\ K_{xy}^\sharp \frac{\partial h}{\partial x} \end{bmatrix} = -\begin{bmatrix} \frac{K_{xy}^\sharp}{K_{yy}^\sharp} K_{yy}^\sharp \frac{\partial h}{\partial y} \\ \frac{K_{xy}^\sharp}{K_{xx}^\sharp} K_{xx}^\sharp \frac{\partial h}{\partial x} \end{bmatrix}.$$

From this equation, we let

$$v_x = K_{xx}^\sharp \frac{\partial h}{\partial x},$$

and

$$v_y = K_{yy}^\sharp \frac{\partial h}{\partial y}.$$

So, we can rewrite the above equation as follows:

$$-\begin{bmatrix} \frac{K_{xy}^\sharp}{K_{yy}^\sharp} v_y \\ \frac{K_{xy}^\sharp}{K_{xx}^\sharp} v_x \end{bmatrix} = -\begin{bmatrix} 0 & \frac{K_{xy}^\sharp}{K_{yy}^\sharp} \\ \frac{K_{xy}^\sharp}{K_{xx}^\sharp} & 0 \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \end{bmatrix}$$

and write

$$\begin{bmatrix} 0 & \frac{K_{xy}^\sharp}{K_{yy}^\sharp} \\ \frac{K_{xy}^\sharp}{K_{xx}^\sharp} & 0 \end{bmatrix} = T_n.$$

So, the system can be rewritten as

$$v = v_d + v_n = v_d + T_n v_d = (I + T_n) v_d$$

with

$$v_d = -K_d^\sharp \nabla h = -\begin{bmatrix} K_{xx}^\sharp & 0 \\ 0 & K_{yy}^\sharp \end{bmatrix} \nabla h.$$

So, with this definition, we can rewrite the system as

$$\begin{aligned} -\nabla\cdot(I + T_n)v_d &= 0, \\ v_d &= -K_d^\sharp \nabla h, \end{aligned}$$

and have an equivalent system that includes the off diagonal entries in the original tensor. So, we can solve for v_d more easily and then we use the divergence property to complete the computation as follows:

$$v_d = -K_d^\sharp \nabla h,$$

$$(K_d^\sharp)^{-1} v_d = \nabla h,$$

$$\begin{bmatrix} \frac{1}{K_{xx}^\sharp} & 0 \\ 0 & \frac{1}{K_{yy}^\sharp} \end{bmatrix} \cdot \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \frac{v_x}{K_{xx}^\sharp} \\ \frac{v_y}{K_{yy}^\sharp} \end{bmatrix} = \begin{bmatrix} \frac{\partial h}{\partial x} \\ \frac{\partial h}{\partial y} \end{bmatrix}.$$

This allows us to get a relationship between K, v_x, and v_y. So,

$$\begin{aligned} \frac{\partial h}{\partial x} &= -\frac{v_x}{K_{xx}}, \\ h(x,y) &= -\frac{v_x}{K_{xx}}x + f(y), \\ \frac{\partial h}{\partial y} &= 0 + f'(y) = -\frac{v_y}{K_{yy}}, \\ f(y) &= -\frac{v_y}{K_{yy}}y + C_1, \end{aligned}$$

and we can write

$$h(x,y) = -\frac{v_x}{K_{xx}}x - \frac{v_y}{K_{yy}}y + C_1.$$

This approach produces a representation with only three constants. So we try another approach. If we consider the diagonal velocity, we can consider a projection idea, so if we

assume the solution:

$$h(x,y) = axy + bx + cy + d$$

. Note that,

$$\begin{aligned}
\nabla \cdot K_d^\sharp \cdot \nabla h &= \nabla \cdot K_d^\sharp \cdot \nabla(axy + bx + cy + d) \\
&= \nabla \cdot \begin{bmatrix} K_{xx}^\sharp & 0 \\ 0 & K_{yy}^\sharp \end{bmatrix} \cdot \begin{bmatrix} ay + b \\ ax + c \end{bmatrix} = \nabla \cdot \begin{bmatrix} K_{xx}^\sharp(ay + b) \\ K_{yy}^\sharp(ax + c) \end{bmatrix} \\
&= [\frac{\partial}{\partial x}, \frac{\partial}{\partial y}] \cdot \begin{bmatrix} K_{xx}^\sharp(ay + b) \\ K_{yy}^\sharp(ax + c) \end{bmatrix} = 0.
\end{aligned}$$

The end result is that

$$h(x,y) = axy + bx + cy + d$$

is in the null space of our operator. The next step for this is to fit the corner values to the polynomial. That is,

$$h(x,y) = axy + bx + cy + d$$

$$\begin{aligned}
h(0,0) &= a(0)(0) + b(0) + c(0) + d = d \\
\Rightarrow h(0,0) &= d \\
h(1,0) &= a(1)(0) + b(1) + c(0) + d = b + d \\
\Rightarrow b &= h(1,0) - d = h(1,0) - h(0,0) \\
h(0,1) &= a(0)(1) + b(0) + c(1) + d \\
\Rightarrow c &= h(0,1) - d = h(0,1) - h(0,0) \\
h(1,1) &= a(1)(1) + b(1) + c(1) + d \\
\Rightarrow h(1,1) - h(0,0) &= a + (h(1,0) - h(0,0)) + (h(0,1) - h(0,0)) \\
\Rightarrow a &= (h(1,1) - h(0,0)) - (h(1,0) - h(0,0)) - (h(0,1) - h(0,0))
\end{aligned}$$

So, we can write the formula out for this set of polynomial:

$$\begin{aligned} d &= h(0,0), \\ b &= h(1,0) - d, \\ c &= h(0,1) - d, \\ a &= h(1,1) - d - b - c. \end{aligned}$$

Then, we can evaluate this at any given (x, y)

$$h(x,y) = axy + bx + cy + d.$$

Now that we have a solution, we need to incorporate this into the diagonal matrix problem. Again, the system is

$$v_d = -K_d^\sharp \nabla h, \tag{5.15}$$

$$-\nabla \cdot (I + T_n) v_d = 0. \tag{5.16}$$

So,

$$\begin{aligned} \nabla h &= \nabla(axy + bx + cy + d) \\ &= \begin{bmatrix} ay + b \\ ax + c \end{bmatrix} \\ v_d &= -K_d^\sharp \cdot \begin{bmatrix} ay + b \\ ax + c \end{bmatrix} \\ -\begin{bmatrix} K_{xx}^\sharp & 0 \\ 0 & K_{yy}^\sharp \end{bmatrix} \cdot \begin{bmatrix} ay + b \\ ax + c \end{bmatrix} &= -\nabla \cdot \begin{bmatrix} K_{xx}^\sharp (ay + b) \\ K_{yy}^\sharp (ax + c) \end{bmatrix} \end{aligned}$$

and then, equation 5.3 gives:

$$-\nabla \cdot \begin{bmatrix} v_x + (\frac{K^\sharp_{xy}}{K^\sharp_{yy}})v_y \\ v_y + (\frac{K^\sharp_{xy}}{K^\sharp_{xx}})v_x \end{bmatrix} = 0.$$

Integrating the above equation gives:

$$\begin{bmatrix} v_x + (\frac{K^\sharp_{xy}}{K^\sharp_{yy}})v_y \\ v_y + (\frac{K^\sharp_{xy}}{K^\sharp_{xx}})v_x \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \vec{v}.$$

This is the original Darcy velocity.

CHAPTER 6
SUMMARY OF WORK, CONCLUSIONS AND FUTURE RESEARCH

6.1 Summary and Conclusions

In this work, we propose a fast transform algorithm in one dimension for computing harmonic average of functions representing the fine scale parameter values that uses a dyadic mesh in the spatial domain. The fast transform algorithm introduced is built on the idea of wavelet multi-resolution that preserves the harmonic average of the permeability. We also provide a process that computes the solution of the pressure variable using wavelet multi-resolution analysis. Furthermore, we implemented Java codes that compute the pressure variables and came up with a close form generalization of the formula in our solution.

We also extend the proposed methodology to the two dimensional case. We presented a methodology for the construction of solutions of elliptic differential equations in two dimensions with piecewise coefficients using diagonal tensors. This methods are effective in computing the approximate solutions for the elliptic problems in two dimensions. We developed fast transform algorithm in two dimensions that computes the correct average using homogenization theory. Furthermore, we develop Java codes that compute the fast transform and the inverse transform algorithm. The results obtained are consistent with the result of the local problems solutions. Lastly, we provide the reconstruction algorithm in two dimensions using the results of the local problems, this methods allow the reconstruction of the solution to any desired scale.

6.2 Future Research Direction

I plan to investigate the convergence of our proposed homogenization wavelet reconstruction method using ideas employed in [37]. The approach involves showing the equivalency of their proposed block-centered finite differences to a mixed finite element method whose convergence is known to be of second-order accuracy. The authors then conclude that their method also converges to the same accuracy as the mixed finite element method since both methods behave the same way.

I plan to continue my research in multi-resolution analysis with applications in higher dimensions of the current elliptical problems. Current results of the extension of the elliptical problems from the one-dimensional to the two-dimensional case will form the basis of the generalization of the solutions to the multi-dimensional case. The case of a regular full tensor will also be looked into in the near future. I also plan to continue the analysis of solution differences to two and three dimensional cases as well as look into the is efficiency of my codes. Work is underway for the application to real life problems like Cahn Hilliard Equation.

I also plan at investigating construction of wavelet bases conditioned on differential equations for modeling nonlinear conservation laws. In this problem, a combination of the lifting method of Sweldons and a polynomial framework for defining finite difference approximations for nonlinear hyperbolic conservation laws may be used to define shape functions, and thus wavelet basis functions conditioned on the discrete formula. The polynomial framework can then be used as an interpolation operator in the lifting method. In applying lifting methods, the polynomial framework defines shape functions that are conditioned on a discrete version of the hyperbolic differential operator. Examples of bases conditioned on upwind, Lax-Wendroff, TVD, and other discrete operators can be computed. One main application of interest involves the definition of Entropy Satisfying Multi Resolution Analyses.

REFERENCES

[1] T. Arbogast, G. Pencheva, M. Wheeler, and I. Yotov, "A multiscale mortar mixed finite element method," *Multiscale Model. Simul.,*, vol. 6, pp. 319–346, 2007.

[2] T. Arbogast, Z. Tao, and H. Xiao, "Multiscale mortar mixed methods for heterogeneous elliptic problems," *Contemporary Mathematics*, vol. 586, pp. 9–21, 2013.

[3] D. N. Arnold, "An interior penalty finite element method with discontinuous elements," *SIAM Journal on Numerical Analysis*, vol. 19, pp. 742 – 760, 1982.

[4] I. Babuska and R. Lipton, "Optimal local approximation spaces for generalized finite element methods with application to multiscale problems," *Multiscale Model. Simul.*, vol. 9, pp. 373–406, 2011.

[5] L. Berlyand and H. Owhadi, "Flux norm approach to finite dimensional homogenization approximations with non-separated scales and high contrast," *Arch. Ration. Mech. Anal.*, vol. 198, pp. 677–721, 2010.

[6] Y. Efendiev, J. Galvis, and T. Y. Hou, "Generalized multiscale finite element methods (gmsfem)," *Journal of Computational Physics*, vol. 251, pp. 116 – 135, 2013.

[7] D. Elfverson, E. H. Geogoulis, A. Malqvist, and D. Peterseim, "Convergence of a discontinuous galerkin multiscale method," *SIAM Journal on Numerical Analysis*, vol. 51, pp. 3351–3372, 2013.

[8] T. Y. Hou and X.-H. Wu, "A multiscale finite element method for elliptic problems in composite materials and porous media," *Journal of Computational Physics*, vol. 134, pp. 169 – 189, 1997.

[9] O. A. Karakashian and F. Pascal, "A posteriori error estimates for a discontinuous galerkin approximation of second-order elliptic problems," *SIAM Journal on Numerical Analysis*, vol. 41, pp. 2374 – 2399, 2003.

[10] M. G. Larson and A. Malqvist, "Adaptive variational multiscale methods based on a posteriori error estimatimation," *Computational Methods in Applied Sciences and Engineering*, vol. 196, pp. 2313 – 2324, 2007.

[11] I. Babuska, G. Caloz, and J. E. Osborn, "Special finite element methods for a class of second order elliptic problems with rough coefficients," *SIAM Journal on Numerical Analysis*, vol. 31, pp. 945 – 981, 1994.

[12] I. Babuska and J. E. Osborn, "Generalized finite element methods: Their performance and their relation to mixed methods," *SIAM Journal on Numerical Analysis*, vol. 20, pp. 510– 536, 1983.

[13] M. Dorobantu and B. Engquist, "Wavelet-based numerical homogenization," *SIAM J. NUMER. ANAL.*, vol. 35, pp. 540–559, 1998.

[14] B. Riviere, *Discontinuous Galerkin Methods for Solving Elliptic and Parabolic Equatons-Theory and Implementation*, 2008.

[15] R. Becker, P. Hansbo, and M. G. Larson, "Energy norm a posteriori error estimation for discontinuous galerkin methods," *Computer Methods in Applied Mechanics and Engineering*, vol. 192, pp. 723–733, 2003.

[16] M. Dryja, "On discontinuous galerkin methods for elliptic problems with discontinuous coefficients," *Computational Methods in Applied Mathematics*, vol. 3, pp. 76– 85, 2003.

[17] M. Brewster and G. Beylkin, "A multiresolution strategy for numerical homogenization," *Applied and Computational Harmonic Analysis*, vol. 2, pp. 327–349, 1995.

[18] A. Chertock and D. Levy, "On wavelet-based numerical homogenization," *Multiscale Model Simulation*, vol. 3, pp. 65–88, 2004.

[19] P.-O. Persson and O. Runborg, "Simulation of a waveguide filter using wavelet-based numerical homogenization," *Journal of Computational Physics*, vol. 166, pp. 361–382, 2001.

[20] Y. Efendiev, T. Hou, and V. Ginting, "Multiscale finite element methods for nonlinear problems and their applications," *Comm. Math. Sci.*, vol. 2, pp. 553–589, 2004.

[21] L. Jiang, D. Copeland, and J. Moultons, "Mixed multiscale finite element methods and their applications for flows in porous media," *Multiscale Analysis*, vol. 2, pp. 1–33, 2012.

[22] A. C. Gilbert, "Multiscale analysis and data networks," *Applied and Computational Harmonic Analysis*, vol. 10, pp. 185–202, 2001.

[23] L. Zhang, L. Cao, and J. Luo, "Multiscale analysis and computation for a stationary schrodinger-poisson system in heterogeneous nanostructures," *Multiscale Model Simulations*, vol. 4, pp. 1561–1591, 2014.

[24] D. Elfverson and A. Malqvist, "Discontinuous galerkin multiscale methods for convection dominated problems," *Tech. report 2013 -011, Department of Information Technology, Uppsala University, Sweden*, 2013.

[25] D. Elfverson, E. H. Geogoulis, and A. Malqvist, "An adaptive discontinuous galerkin multiscale method for elliptic problems," *Multiscale Model. Simul.*, vol. 11, pp. 747–765, 2013.

[26] Y. Efendiev, J. Galvis, R. Lazarov, M. Moon, and M. Sarkis, "Generalized multiscale finite element method. symmetric interior penalty coupling," *Journal of Computational Physics*, vol. 255, pp. 1 – 15, 2013.

[27] J. Han, M. Kamber, and J. Pei, *Data Mining Concepts and Techniques*, 2012.

[28] A. Boggess and F. J. Narcowich, *A First Course in Wavelets with Fourier Analysis*, 2009.

[29] C. M. Leavey, M. N. James, J. Summerscales, and R. Sutton, "An introduction to wavelets transforms: A tutorial approach," *Insight*, vol. 45, pp. 344–353, 2003.

[30] K. Dalyand and T. Roose, "Homogenization of two fluid flow in porous media," *rspa.royalsocietypublishing.org*, vol. A, pp. 1–20, 2015.

[31] H. Douanla and J. L. Woukeng, "Homogenization of reaction-diffusion equations in fractured porous media," *Math.AP*, vol. 1, pp. 1–19, 2015.

[32] A. Braides and A. Defranceschi, *Homogenization of Multiple Integrals (Oxford Lecture Series in Mathematics and its Applications)*, 1999.

[33] P. Henning, A. Malqvist, and D. Peterseim, "A localized orthogonal decomposition method for semi-linear elliptic problems," *ESAIM: Mathematical Modelling and Numerical Analysis*, vol. 48, pp. 1331–1349, 2014.

[34] P. Henning and D. Peterseim, "Oversampling for the multiscale finite element method," *Multiscale Model. Simul.*, vol. 11, pp. 1149–1175, 2013.

[35] J. V. Koebbe, "Homogenization-wavelet reconstruction methods for elliptic differential equations," *Manuscript for Review Purpose*, pp. 1–12, 2017.

[36] ——, "A computationally efficient modification of mixed finite element methods for flow problems with full transmissivity tensors," *Numerical Methods for Partial Differential Equations*, vol. 9, pp. 339–355, 1993.

[37] A. Weiser and M. F. Wheeler, "On convergence of block-centered finite differences for elliptic problems," *SIAM Journal on Numerical Analysis*, vol. 25, pp. 351 – 375, 1988.

[38] D. Cherney, T. Denton, R. Thomas, and A. Waldron, *Linear Algebra*, 2013.

[39] R. J. LeVeque, *Finite Difference Methods for Ordinary and Partial Differential Equations*, 2007.

APPENDICES

Appendix A

Linear Algebra Basics

We outline some important properties of matrices and theorems that support our methods of computation in this dissertation. The following theorems and definitions can be found in [38] and [39].

Theorem 1. *(The Fundamental Theorem of Algebra) Any polynomial can be factored into a product of first order polynomials.*

Let A be an $m \times m$ matrix with m eigenvalues, $\lambda_1, \lambda_2, \ldots, \lambda_m$, that are the roots of the characteristic polynomial, $P_A(\lambda) = det(A - \lambda I)$. Theorem 1 implies that there exists a collection of n complex numbers λ_i (possibly with repetition) such that

$$P_A(\lambda) = (x - \lambda_1), (x\lambda_2), \ldots, (x - \lambda_m)$$

which implies that

$$P_A(\lambda) = det(A - \lambda I).$$

The polynomial of degree m will always have m roots, but there may be multiple roots. If there are no repeated roots, then we have distinct roots. The set of m eigenvalues is known as the spectrum of the matrix. The spectral radius of A is the maximum magnitude of any eigenvalue ($\rho(A) = \max \mid \lambda_p \mid$). If the characteristic polynomial has a factor $(x-\lambda)^s$, the eigenvalue has algebraic multiplicity of s, ($m_a(\lambda) = s$). The space of all vectors with eigenvalue λ is called an eigenspace.

Also, any vector u in the eigenspace satisfies the equation $Au = \lambda u$. The dimension of this eigenspace is known as the geometric multiplicity $m_g(\lambda)$, of the eigenvalue, λ. If $m_g(\lambda) = m_a(\lambda)$, then A has a complete set of eigenvectors for this eigenvalue, otherwise this eigenvalue is defective. If A has one or more defective eigenvalues, then A is called a

defective matrix. If the eigenvalues of A are all distinct, then $m_g(\lambda) = m_a(\lambda) = 1$ for every eigenvalue and the matrix is not defective. A diagonal matrix cannot be defective. The eigenvalues are simply the diagonal elements, and the unit vectors e_j form a complete set of eigenvectors.

Similarity Transformation

Definition 1. *A matrix A is diagonalizable if there exists an invertible matrix P and a diagonal matrix D such that*

$$D = P^{-1}AP$$

.

This can be summarized as follows:

- Change of basis rearranges the components of a vector by the change of basis matrix P, to give components in the new basis.
- To get the matrix of a linear transformation in the new basis, we conjugate the matrix of A by the change of basis matrix:

$$A = P^{-1}AP.$$

Corollary 1. *A square matrix A is diagonalizable if and only if there exists a basis of eigenvectors for A. Moreover, these eigenvectors are the columns of a change of basis matrix P which diagonalizes A.*

Diagonalizing Symmetric Matrices

Definition 2. *A matrix A is symmetric if $A^T = A$.*

One nice property of symmetric matrices is that they always have real eigenvalues.

Theorem 2. *Eigenvectors of a symmetric matrix with distinct eigenvalues are orthogonal.*

This means that

$$P^{-1} = P^T,$$

or

$$PP^T = I = P^T P.$$

Theorem 3. *Every symmetric matrix is similar to a diagonal matrix of its eigenvalues.*

In other words,

$$A = A^T \Leftrightarrow A = PDP^T$$

where P is an orthogonal matrix and D is a diagonal matrix whose entries are the eigenvalues of A.

Appendix B

Numerical Solutions in One Dimension for Computing Alphas and Betas

This appendix contains Java codes that randomly generates permeability tensors K in one dimension and computes α's and β's using our method of solutions. The results of α's and β's are then used in the reconstruction of the solutions. The next Java class, Formula compute the close form recursive formula for the differences of solution of the local problem in one dimension.

```
package HomogenizationProject;

/**
 *
 * @author Abibat Lasisi
 */
public class RandomHomo {

    public static int ALPHA = 1;
    public static int BETA = 2;

    // assign value of n
    public static int N = 4;

    //static double [] waveCoeff =
```

```
{0.1, 0.3, 0.2, 0.4, 0.6, 0.5, 0.8, 0.7};
static int numPartitions = (int) Math.pow(2, N);
static double waveCoeff[] = new double[numPartitions];

public static double [] kOfN(int n){

    double [] newWaveCoeff = new double [waveCoeff.length / 2];

    if(n == N ) {
    // only for the first time, randomly generate the indices

        System.out.println("--------------------------------");
        // compute the starting indices randomly
        for(int i = 0; i < waveCoeff.length; i++) {
            double random = 0.1 + Math.random();
            waveCoeff[i] =  random < 1.0 ? random : 1.0;

            System.out.println("k" + N + "," + i +
                     " = " + waveCoeff[i]);
        }

        System.out.println("--------------------------------");

        return waveCoeff;
    }

    //subsequently generate new indices from the previous
```

```
int j = 0;  // starting from the first even position
for(int i = 0; i <  newWaveCoeff.length; i++) {
    newWaveCoeff[i] =
    (2.0*waveCoeff[j]*waveCoeff[j+1])/((waveCoeff[j]+waveCoeff[j+1]));
        j += 2;
    }

    //update the waveCoeff
    waveCoeff  = newWaveCoeff;

    return newWaveCoeff;
}



//compute alpha values using the k's
public static double [] alphaValues(int n) {

    int aLevel = n - 1;

    double [] waveCoeffAve = kOfN(n);
    double [] alphas = new double [waveCoeffAve.length / 2];

    int j = 0;  // starting from the first even position
    for(int i = 0; i <  alphas.length; i++) {
        alphas[i] = waveCoeffAve[j] /
        (waveCoeffAve[j] + waveCoeffAve[j + 1]);
        j += 2;
    }
```

```
  // for display
  //for(int i = 0; i < x.length; i++)
  //  System.out.println("a" + aLevel + "," + i + " = " + x[i]);

    return alphas;
}

// compute the beta values using 1 - alphaValues
public static double [] betaValues(double [] x) {

    int bLevel = N - 1;

    double [] betas = new double [x.length];
    for(int i = 0; i < betas.length; i++)
        betas[i] = 1 - x[i];

 // for display
 // for(int i = 0; i < betas.length; i++)
 //   System.out.println("b" + bLevel + "," + i + " = " + betas[i]);

    return betas;
}


//compute all alphas and betas from  n = N down to n = 0
public static double [] allAphasOrBetas(int n, int type) {
```

```
//get how many alpha or beta values for this n
int arraySize = 0;
for(int i = 0; i < n; i++) {

    int numPartitions = (int) Math.pow(2, i);
    arraySize += numPartitions;
}

double [] alphas = new double[arraySize];
double [] betas = new double[arraySize];
int arrayIndex = 0;

// all beta values for n
for(int i = n; i >= 0; i--) {
    // compute the alpha and beta values for n = i
    double [] x = alphaValues(i);

    for(int j = 0; j < x.length; j++) {
        alphas[arrayIndex] = x[j];
        betas[arrayIndex] = 1 - x[j];

        double result = (int)(alphas[arrayIndex++] * 10000) / 10000.0;

        // for display
System.out.println("a" + (i - 1) + "," + j + " = " + result);
System.out.println("b" + (i - 1) + "," + j + " = " + (1 - result));

    }
```

```
            System.out.println();

        }

        if(type == ALPHA)
            return alphas;
        else
           return betas;
    }

    public static void main(String [] args) {

        allAphasOrBetas(N, ALPHA);
    }
}
```

Recursive Differencing Formula of Analytic Solution

This program computes the close form recursive formula for the differences of solution of the local problem in one dimension.

```
package HomogenizationProject;



/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Abibat Lasisi
 */
public class Formula {

    public static final String ALPHA = "\u03B1";
    public static final String BETA = "\u03B2";
    public static final String DOT = "\u00B7";

    public static final String PHI = "\u03A6";
    public static final String PSI = "\u03A8";

    public static String buildString(int n){

        int power = (int) Math.pow(2, n);
```

```
    int midValue = power - 2;
    StringBuilder string = new StringBuilder();

    String result =  "y" + n + ",";
    string.append(result);
    int counter = 0;
    for(int i = 1; i <= midValue / 2; i++) {
        counter++;
        for(int j = 0; j < 2; j ++) {
            if(j % 2 == 0)
                result =  "(" + counter + " - " +  "y" + n + "),";
            else
                result = "(y" + n + " - " + counter + "),";
            string.append(result);
        }
    }
    if(midValue >= 0) {
        result =  "(" + power / 2 + " - " + "y" + n + ")";
        string.append(result);
    }
    result = ",";
    string.append(result);

    return string.toString();
}

private static String getTruthTable(int n) {
    int rows = (int) Math.pow(2,n);
```

```
    String binary = "";

    for (int i = 0; i < rows; i++) {
        for (int j = n - 1; j > 0; j--)
            binary += i / (int) Math.pow(2, j) % 2 + " ";

        binary += ",";  // comma to seperate the items
    }

    return binary;
}

// generate alpha and beta indices
public static String [] generateIndices(int n) {

    int power = (int) Math.pow(2, n);
    int [][] matrix = new int[power][n - 1];
    int incValue = 10;

    String [] string = new String[power];
    for(int i = 0; i < string.length; i++)
        string[i] = "00,";

    //intialize the first row of matrix to 0, 10, 20, etc
    for(int i = 1; i < matrix[0].length; i++) {
        matrix[0][i] += incValue;
        incValue += 10;
```

```
}

//determine the number of columns
int cols = matrix[0].length;
int divider = matrix.length;

//now compute the rest values for matrix, skipping row 1
for(int j = 1; j < cols; j++) {
    divider /= 2;
    int counter = 1;
    for(int i = 1; i < matrix.length; i++){
        if(counter < divider){
            matrix[i][j] = matrix[i - 1][j];
            counter++;
        }
        else {
            matrix[i][j] = matrix[i - 1][j] + 1;
            counter = 1;
        }
    }
}

//create a string array
for(int i = 0; i < matrix.length; i++) {
    for(int j = 0; j < matrix[i].length; j++) {
        if(matrix[i][j] != 0)
           string[i] += matrix[i][j] + ",";
    }
```

```
    }

    return string;

}

public static String getAlphaBeta(String binary, String indices) {

    String binaryArray [] = binary.split(" ");
    String indicesArray [] = indices.split(",");

    String characters = "";

    for(int i = 0; i < binaryArray.length; i++) {

        if(binaryArray[i].equals("0"))
            characters += BETA + "_" + indicesArray[i];
        else if(binaryArray[i].equals("1"))
            characters += ALPHA + "_" + indicesArray[i];

        characters += " "+ DOT + " ";
    }

    return characters;
}

public static void generateFormula(int n) {
```

```
// get string for the expressions
String string = buildString(n);
String expr [] = string.split(",");
int expCount = 0;

//get truth table to generate the beta and alpha coefficients
String table = getTruthTable(n);
String truthTable [] = table.split(",");

//get indices to generate the beta and alpha indices
String indices [] = generateIndices(n);



int midPower = (int) Math.pow(2, n) / 2;
int index = n - 1;

int j = 0;
int k = 0;

//print upper
for(int i = 0; i < midPower; i++) {
    if(j > 1) {
        k++;
        j = 0;
    }

    if(n > 1) {
        String coeff =
```

```
            getAlphaBeta(truthTable[expCount], indices[expCount]);
            System.out.println(coeff + PSI+ "_" + index + "," + k + " "
             + DOT +" " + PHI + " " + DOT + " " + expr[expCount++]);
        }
        else
            System.out.println(PSI+ "_" + index + "," + k + " "
            + DOT + " " + PHI + " " + DOT + " " + expr[expCount++]);
        j++;
    }

    // print lower
    for(int i = 0; i < midPower; i++) {
        if(j > 1) {
            k++;
            j = 0;
        }
        if(n > 1) {
            String coeff =
            getAlphaBeta(truthTable[expCount], indices[expCount]);
            System.out.println(coeff + PSI+ "_" + index + ","
            + k + " " + DOT + " " + PHI + " " + DOT + " "
             + expr[expCount++]);
        }
        else
            System.out.println(PSI+ "_" + index + "," + k + " "+
             DOT + " " + PHI + " " + DOT + " " + expr[expCount++]);
        j++;
    }
```

```
    }

    public static void main(String [] args) {

        int n = 2;

        System.out.println("n = " + n);
        generateFormula(n);
    }

}
```

Appendix C

Numerical Solution of Local Elliptic Problem in Two Dimensions

In this appendix, we provide Java codes that compute the numerical solution of local elliptic problem in two dimensions. In the first class in this package, we use Gaussian Elimination method to solve system of linear equations generated from the solution of the local problems. We create a method that performs the elimination operation using matrix and vector. We compute the multipliers and then create a method that performs back substitution operation on the final matrix to determine the solutions. The next classes, Matrix and Vector are used to manage the matrices and vectors permeability tensors, for easy accessibility. We create a class called MatrixComputation which computes the inverse of a matrix, matrix coefficients, perform vector and matrix multiplication, computes product of vector and matrix. We then create a method that computes the gradient of w and finally perform the operation that computes $K^{\sharp}$. The last class, Generalization is used to generalize the procedure to manage $n \times n$ matrices.

```
package HomogenizationProject;

/**
 *
 * @author Abibat Lasisi
 */

import java.util.Scanner;
public class GaussElimination {
```

```
//this is the matrix size, row and column
private static final int MATRIX_SIZE = 2;

private boolean rowInterchanged = false;
private int interchangedRow = 0;

// This method performs the elimination operation using
// a matrix and vector
public void eliminationProcedure(double matrix[][], double b[]){

    //interchange row if matrix[0][0] is zero
    if(Math.abs(0 - matrix[0][0]) < 0.00000000001 ){

        int row = 1;
        while(matrix[row][0] == 0 && row < matrix.length)
            row++;

        double temp [] = new double[matrix[0].length];
        //matrix
        for(int i = 0; i < matrix[0].length; i++) {
            temp[i] = matrix[0][i];
            matrix[0][i] = matrix[row][i];
            matrix[row][i] = temp[i];
        }

        // b vector
        double t = b[0];
        b[0] = b[row];
```

```
            b[row] = t;

            rowInterchanged = true;
            interchangedRow = row;
        }

         //double multiplier
        for(int k = 0; k < MATRIX_SIZE - 1; k++){
            for(int i = k + 1; i < MATRIX_SIZE; i++) {

                double multiplier = matrix[i][k] / matrix[k][k];

                b[i] = b[i] - multiplier * b[k];
                for(int j = k; j < MATRIX_SIZE; j++){
                    matrix[i][j] = matrix[i][j] - (multiplier * matrix[k][j]);
                }
            }

        }
    }

    //this method performs back substitution operation on the final matrix to
    //determine the solutions
    public double [] backSubstitution(double matrix[][], double b[]) {
        //int i, j;
        double x[] = new double[MATRIX_SIZE];

        x[MATRIX_SIZE - 1] = b[MATRIX_SIZE - 1] /
```

```
            matrix[MATRIX_SIZE - 1][MATRIX_SIZE - 1];

    for(int i = MATRIX_SIZE - 2; i >= 0; i--){
        x[i] = b[i];
        for(int j = MATRIX_SIZE - 1; j >= i + 1; j--){
            x[i] = x[i] - matrix[i][j] * x[j];
        }
        x[i] = x[i] / matrix[i][i];
    }

    if(rowInterchanged) {
         // b vector
        double t = x[0];
        x[0] = x[interchangedRow];
        x[interchangedRow] = t;

        rowInterchanged = false;
    }

    return x;
}

//method to print the content of a matrix
public void printMatrix(double a[][]) {
    //int i, j;

    for(int i = 0; i < MATRIX_SIZE; i++) {
        for(int j = 0; j < MATRIX_SIZE; j++) {
```

```
            System.out.print(a[i][j] + "   ");
        }
        System.out.println();
    }
    System.out.println();
}

//method to print the content of vectors
public void printVector(double a[]) {
    //int i;

    for(int i = 0; i < MATRIX_SIZE; i++) {
         System.out.print(a[i] + "   ");
    }

    System.out.println("\n");
}

public static void main(String[] args) {
    // TODO code application logic here
   int i, j;
    //read matrices from the console
   Scanner in = new Scanner(System.in);

    //create the object of the class GaussElimination
    GaussElimination gauss = new GaussElimination();

    System.out.println("Input the matrix size");//a from console\n");
```

```
  // gauss.MATRIX_SIZE = in.nextInt();


  // double a[][] = new double[MATRIX_SIZE][MATRIX_SIZE];
  // double b[] = new double[MATRIX_SIZE];

   /*
   System.out.println("Input matrix a from console\n");

   for(i = 0; i < gauss.MATRIX_SIZE; i++){
      for(j = 0; j < gauss.MATRIX_SIZE; j++){
          System.out.print((i+1)+","+(j+1) +":");
          a[i][j] = in.nextDouble(); //in.nextInt();
      }
      //System.out.println();
  }

  System.out.println();
  System.out.println("Input vector b from console\n");
  for(i = 0; i < gauss.MATRIX_SIZE; i++){
     System.out.print((i+1)+":");
     b[i] = in.nextInt();
  }

 in.close();

System.out.println();
*/
```

```
 double a [][] = {
                  {0, 9 / 7.0},
                  {18 / 7.0, -36 / 7.0}

                };

 double b [] = {18 / 7.0, 18 / 7.0};

 //double a [][] = null;
 //double b [] = null;

 System.out.println("Original Coefficient Matrix a");
 gauss.printMatrix(a);  // call the printMatrix method to print matrx a

 System.out.println("Original Vector b");
 gauss.printVector(b);  //// call the printVector method to print vector

 System.out.println("Solutions");

//call elimination procedure
 gauss.eliminationProcedure(a, b);

//call backSubstitution
gauss.backSubstitution(a, b);

 System.out.println("Modified Coefficient Matrix a");
 gauss.printMatrix(a);  // call the printMatrix method to print matrx a
```

```
      System.out.println("Modified Vector b");
      gauss.printVector(b);  //// call the printVector method to print vector
   }
}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package HomogenizationProject;

/**
 *
 * @author Abibat Lasisi
 */

//Matrix class to manage matrices
public class Matrix {

    private double [][] matrix;

    public Matrix(){

    }

    public Matrix(double [] v1, double [] v2){

        this.matrix = new double[v1.length][v1.length];

        for(int i = 0; i < matrix.length; i++) {
            this.matrix[i][0] = v1[i];
            this.matrix[i][1] = v2[i];
```

```
    }
}

public Matrix(double [][] matrix){

    this.matrix = new double[matrix.length][matrix[0].length];

    for(int i = 0; i < matrix.length; i++)
        for(int j = 0; j < matrix[i].length; j++)
            this.matrix[i][j] = matrix[i][j];
}

// 4 x 4 matrix from 4 2 x 2 matrices
public Matrix(double [][] m1, double [][] m2,
            double [][] m3, double [][] m4){

    int matrixSize = m1.length * m1.length;
    this.matrix = new double[matrixSize][matrixSize];

    for(int i = 0; i < m1.length; i++) {
        for(int j = 0; j < m1[i].length; j++) {
            this.matrix[i][j] = m1[i][j];
            this.matrix[i][j + 2] = m2[i][j];
            this.matrix[i + 2][j] = m3[i][j];
            this.matrix[i + 2][j + 2] = m4[i][j];
        }
    }
}
```

```
public void setMatrix(double [][] matrix) {
    this.matrix = new double[matrix.length][matrix[0].length];

    for(int i = 0; i < matrix.length; i++)
       for(int j = 0; j < matrix[i].length; j++)
           this.matrix[i][j] = matrix[i][j];
}

public double [][] getMatrix(){
    return this.matrix;
}

public void printMatrix() {

    for(int i = 0; i < matrix.length; i++) {
        for(int j = 0; j < matrix[i].length; j++) {
            System.out.print(matrix[i][j] + " ");
        }
        System.out.println();
    }
}

public Matrix matrixInverse() {

    double [][] matrix_ = {
                            {matrix[1][1], -1.0 * matrix[0][1]},
                            {-1.0 * matrix[1][0], matrix[0][0]}
```

```
                        };


        double determinant = matrix[0][0] * matrix[1][1] -
                             matrix[0][1] * matrix[1][0];

        for(int i = 0; i < matrix.length; i++)
            for(int j = 0; j < matrix[i].length; j++) {
                 if(matrix[i][j] == 0)
                     matrix_[i][j] = 0;
                 matrix_[i][j] = matrix_[i][j] / determinant;
            }

        return new Matrix(matrix_);
    }
}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package HomogenizationProject;

/**
 *
 * @author Abibat Lasisi
 */

// Vector class to manage vectors
public class Vector {

    private double [] vector;

    public Vector(){

    }

    public Vector(double [] vector){

        this.vector = new double[vector.length];

        for(int i = 0; i < vector.length; i++)
            this.vector[i] = vector[i];
    }
```

```
    public void setVector(double [] vector) {

        this.vector = new double[vector.length];

        for(int i = 0; i < vector.length; i++)
            this.vector[i] = vector[i];
    }

    public double [] getVector(){
        return this.vector;
    }

    public void printVector() {

        for(int i = 0; i < vector.length; i++)
          System.out.println(vector[i]);
    }

}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package HomogenizationProject;

/**
 *
 * @author Abibat Lasisi
 */

public class MatrixComputation {

    public static final int M = 2;
    public static final int W_1 = 1;
    public static final int W_2 = 2;

    // compute inverses of the list of the 2 x 2 matrices
    public static Matrix [] computeInverses (Matrix [] matrixList) {

         Matrix [] inverseList = new Matrix[matrixList.length];

         for(int i = 0; i < matrixList.length; i++) {
             inverseList[i] = new Matrix();
             inverseList[i] = matrixList[i].matrixInverse();
         }
```

```
    return inverseList;
}

// compute other coefficient
public static double [] computeOtherCoefficients(Matrix [] matrixList,
                  double [] cVector, int type) {

    double [] vector = new double[M];
    double [] result = new double[M];

    int i = 0;
    int j = 0;

    if(type == W_1) {
        double [][] m = matrixList[i].getMatrix();
        double [][] m_ = matrixList[i + 1].getMatrix();

        vector[j] = (-1 * m[1][1]) / (m[0][1] - m_[0][1]);
        vector[j + 1] =       m_[1][1] / (m[0][1] - m_[0][1]);

        //compute product of cVector and vector
        for(int k = 0; k < vector.length; k++)
            result[0] += vector[k] * cVector[k];

        //
        i += 2;
        m = matrixList[i].getMatrix();
        m_ = matrixList[i + 1].getMatrix();
```

```
    j = 0;
    vector[j] =      -1 * m[1][1] / (m[0][1] - m_[0][1]);
    vector[j + 1] =       m_[1][1] / (m[0][1] - m_[0][1]);

    for(int k = 0; k < vector.length; k++)
        result[1] += vector[k] * cVector[k];
}
else if(type == W_2) {

    double [][] m = matrixList[i].getMatrix();
    double [][] m_ = matrixList[i + 2].getMatrix();

    vector[j] =  m_[0][0] / (m[0][1] - m_[0][1]);
    vector[j + 1] =   (-1 * m[0][0]) / (m[0][1] - m_[0][1]);

    //compute product of cVector and vector
    for(int k = 0; k < vector.length; k++)
        result[0] += vector[k] * cVector[k];

    //
    i++;
    m = matrixList[i].getMatrix();
    m_ = matrixList[i + 2].getMatrix();

    j = 0;
    vector[j] =       m_[0][0] / (m[0][1] - m_[0][1]);
    vector[j + 1] =   (-1 *   m[0][0]) / (m[0][1] - m_[0][1]);
```

```
        for(int k = 0; k < vector.length; k++)
            result[1] += vector[k] * cVector[k];
    }

    return result;
}

public static double [] computeBVector(Matrix [] matrixList, int type) {
    double [] vector = new double[M];
    int j = 0;

     if(type == W_1) {

        for(int i = 0; i < matrixList.length; i += 2){

            double [][] m = matrixList[i].getMatrix();
            double [][] m_ = matrixList[i + 1].getMatrix();

            vector[j++] = 2 / (m[0][0] + m_[0][0]);
        }
     }
     else if(type == W_2) {

        for(int i = 0; i < matrixList.length - 2; i++){

            double [][] m = matrixList[i].getMatrix();
            double [][] m_ = matrixList[i + 2].getMatrix();
```

```
            vector[j++] = 2 / (m[1][1] + m_[1][1]);

        }
     }

    return vector;
}

public static double [][] computeCoefficientMatrix
         (Matrix [] matrixList, int type) {

    double [][] matrix = new double[M][M];

    if(type == W_1) {
        for(int i = 0; i < matrixList.length; i += 2){

            int row = i / 2;
            int col = i % 2;
            double [][] m = matrixList[i].getMatrix();
            double [][] m_ = matrixList[i + 1].getMatrix();

            matrix[row][col] = m[0][1] / (m[0][0] + m_[0][0]) -
             (m[1][1] / (m[0][1] - m_[0][1]));

            col = (i + 1) % 2;
            matrix[row][col] = m_[0][1] / (m[0][0] + m_[0][0]) +
            (m_[1][1] / (m[0][1] - m_[0][1]));
```

```
        }
    }
    else if(type == W_2) {
        for(int i = 0; i < matrixList.length - 2; i++){

            int row = i;
            int col = 0;
            double [][] m = matrixList[i].getMatrix();
            double [][] m_ = matrixList[i + 2].getMatrix();

            matrix[row][col] = m_[0][0] / (m[0][1] - m_[0][1]) +
            (m_[0][1] / (m[1][1] + m_[1][1]));

            col = 1; //(i + 1) % 2;
            matrix[row][col] = m[0][1] / (m[1][1] + m_[1][1]) -
            (m[0][0] / (m[0][1] - m_[0][1]));
        }
    }

    return matrix;
}

 //compute the product of a matrix and a matrix
public static double [][] matrixByMatrix(double a[][],
            double [][] b) {

    double sum = 0;
    double product[][] = new double [a.length][a.length];
```

```
    //compute matrix product
    for(int i = 0; i < product.length; i++) {
        for(int k = 0; k < product.length; k++) {
            sum = 0;
            for(int j = 0; j < product.length; j++) {
               sum = sum + (a[i][j] * b[j][k]);
            }
            product[i][k] = sum;
        }
   }

   return product; //return the product matrix
}

 //compute the product of a matrix and a vector
public static double [] matrixByVector(double a[][],
 double [] b, int type) {

    double [] product = new double [M];
    double [] unitVector = new double [M];

    if(type == W_1)
        unitVector[0] = 1.0;
    else
        unitVector[1] = 1.0;

    //compute matrix product
```

```
    for(int i = 0; i < a.length; i++) {
        double sum = 0;
        for(int j = 0; j < a.length; j++) {
           sum = sum + (a[i][j] * b[j]);
        }
        product[i] = sum;
    }

   //subtract unit vector from the product
   for(int i = 0; i < product.length; i++)
       product[i] = product[i] - unitVector[i];

   return product;
}

// compute gradient of W
public static Vector [] gradientOfW(Matrix [] matricesInverse,
           Vector [] v, int type) {

   Vector [] vectorsList = new Vector[v.length];

   int counter = 0;
   for(int i = 0; i < matricesInverse.length; i++) {
        double [] a = matrixByVector(matricesInverse[i].getMatrix(),
                       v[i].getVector(), type);
        vectorsList[i] = new Vector(a);
   }
```

```
   return vectorsList;
 }

public static Vector [] createDiagCVectors(double [] cVector, int type){

    // construct array of c vectors
   Vector [] cVectors = new Vector[4];
    if(type == W_1) {
        double [] c = {cVector[0], cVector[1]};

        // construct the individual c vector
        double [] v1 = {c[0], 0};
        double [] v2 = {c[0], 0};
        double [] v3 = {c[1], 0};
        double [] v4 = {c[1], 0};

        cVectors[0] = new Vector(v1);
        cVectors[1] = new Vector(v2);
        cVectors[2] = new Vector(v3);
        cVectors[3] = new Vector(v4);
    }
    else {
        double [] c = {cVector[0], cVector[1]};

        // construct the individual c vector
        double [] v1 = {0, c[0]};
        double [] v2 = {0, c[1]};
        double [] v3 = {0, c[0]};
```

```
        double [] v4 = {0, c[1]};

        cVectors[0] = new Vector(v1);
        cVectors[1] = new Vector(v2);
        cVectors[2] = new Vector(v3);
        cVectors[3] = new Vector(v4);
    }

    return cVectors;
}

public static Vector [] createCVectors(double [] cVector,
              double [] cVector1, int type){

    // construct array of c vectors
   Vector [] cVectors = new Vector[4];
    if(type == W_1) {
        double [] c = {cVector1[0], cVector[0],
         cVector[1],  cVector1[1]};

        // construct the individual c vector
        double [] v1 = {c[0], c[1]};
        double [] v2 = {c[0], c[2]};
        double [] v3 = {c[3], c[1]};
        double [] v4 = {c[3], c[2]};

        cVectors[0] = new Vector(v1);
        cVectors[1] = new Vector(v2);
```

```
            cVectors[2] = new Vector(v3);
            cVectors[3] = new Vector(v4);
        }
        else {
            double [] c = {cVector[0], cVector[1],
             cVector1[0], cVector1[1]};

            // construct the individual c vector
            double [] v1 = {c[1], c[0]};
            double [] v2 = {c[1], c[3]};
            double [] v3 = {c[0], c[2]};
            double [] v4 = {c[0], c[3]};

            cVectors[0] = new Vector(v1);
            cVectors[1] = new Vector(v2);
            cVectors[2] = new Vector(v3);
            cVectors[3] = new Vector(v4);
        }

       return cVectors;
    }

    public static double [][] sumOfKs(Matrix [] matricesList) {

        double [][] sumOfK = new double[M][M];

        for(Matrix m : matricesList) {
            double [][] matrix = m.getMatrix();
```

```
        for(int i = 0; i < sumOfK.length; i++) {
            for(int j = 0; j < sumOfK.length; j++) {
                sumOfK[i][j] += matrix[i][j];
            }
        }
    }

    //now multiply the result by 0.25
    for(int i = 0; i < sumOfK.length; i++) {
        for(int j = 0; j < sumOfK.length; j++) {
            sumOfK[i][j] *= 0.25;
        }
    }

    return sumOfK;
}

public static double [][] computeJacobianProduct(Matrix [] matricesList,
       Vector [] w1Gradient, Vector [] w2Gradient) {

    double [][] jacobianProduct = new double[M][M];

     int count = 0;
     for(Matrix m : matricesList) {
     // from matricesList
        double [][] matrix1 = m.getMatrix();
        Matrix m_ = new Matrix(w1Gradient[count].getVector(),
         // from w1 and w2 gradients
```

```
            w2Gradient[count].getVector());
        double [][] matrix2 = m_.getMatrix();
        double [][] product = matrixByMatrix(matrix1, matrix2);

        // sum the resulting matrices
        for(int i = 0; i < jacobianProduct.length; i++) {
            for(int j = 0; j < jacobianProduct.length; j++) {
                jacobianProduct[i][j] += product[i][j];
            }
        }

        count++;

     }

      //now multiply the result by 0.25
    for(int i = 0; i < jacobianProduct.length; i++) {
        for(int j = 0; j < jacobianProduct.length; j++) {
            jacobianProduct[i][j] *= 0.25;
        }
    }

    return jacobianProduct;

}

public static double [][]
computeKPoundsDiagTensor(double [][] matrix) {
```

```
 // reduce the original matrix to list of 2 x 2 matrices
Matrix [] matricesList = reduceMatrixTo2By2(matrix, M);

// compute a list of the inverses of the 2 x 2 matrices
Matrix [] matricesInverse = computeInverses(matricesList);

double [] cVector = computeBVector(matricesInverse, W_1);

 // construct array of c vectors
Vector [] cVectors = createDiagCVectors(cVector, W_1);

Vector [] w1Gradient = gradientOfW(matricesInverse,  cVectors, W_1);

cVector = computeBVector(matricesInverse, W_2);

 // construct array of c vectors
cVectors = createDiagCVectors(cVector, W_2);

Vector [] w2Gradient = gradientOfW(matricesInverse,  cVectors, W_2);

 // compute sum of K's i.e., sum of the list of matrices
 double [][] sumOfK = sumOfKs(matricesList);

 // compute Jacobian of K products
 double [][] jacobianProduct = computeJacobianProduct(matricesList,
                               w1Gradient, w2Gradient);
```

```
    // now compute kPounds
    double [][] kPounds = new double[M][M];

    for(int i = 0; i < kPounds.length; i++) {
        for(int j = 0; j < kPounds.length; j++) {
            kPounds[i][j] = sumOfK[i][j] + jacobianProduct[i][j];
        }
    }

    return kPounds;
}


public static double [][] computeKPounds(double [][] matrix) {
   //create instance of Gauss Elimination
   GaussElimination gauss = new GaussElimination();

   // reduce the original matrix to list of 2 x 2 matrices
   Matrix [] matricesList = reduceMatrixTo2By2(matrix, M);

   // compute a list of the inverses of the 2 x 2 matrices
   Matrix [] matricesInverse = computeInverses(matricesList);

   // compute coefficient of w1 and w1 gradient
   double [][] coeff = computeCoefficientMatrix(matricesInverse, W_1);

   double [] bVector = computeBVector(matricesInverse, W_1);
```

```
//call elimination procedure
gauss.eliminationProcedure(coeff, bVector);

//call backSubstitution
double [] cVector = gauss.backSubstitution(coeff, bVector);
double [] cVector1 = computeOtherCoefficients(matricesInverse, cVector, W_1);

// construct array of c vectors
Vector [] cVectors = createCVectors(cVector, cVector1, W_1);
Vector [] w1Gradient = gradientOfW(matricesInverse,  cVectors, W_1);

 // coefficient of w2 and w2 gradient
coeff = computeCoefficientMatrix(matricesInverse, W_2);
bVector = computeBVector(matricesInverse, W_2);

 //call elimination procedure
gauss.eliminationProcedure(coeff, bVector);

//call backSubstitution
cVector = gauss.backSubstitution(coeff, bVector);
cVector1 = computeOtherCoefficients(matricesInverse, cVector, W_2);

// construct array of c vectors
cVectors = createCVectors(cVector, cVector1, W_2);
Vector [] w2Gradient = gradientOfW(matricesInverse,  cVectors, W_2);

// compute sum of K's i.e., sum of the list of matrices
double [][] sumOfK = sumOfKs(matricesList);
```

```
   // compute Jacobian of K products
   double [][] jacobianProduct = computeJacobianProduct(matricesList,
                                 w1Gradient, w2Gradient);

   // now compute kPounds
   double [][] kPounds = new double[M][M];

   for(int i = 0; i < kPounds.length; i++) {
        for(int j = 0; j < kPounds.length; j++) {
            kPounds[i][j] = sumOfK[i][j] + jacobianProduct[i][j];
        }
    }

    return kPounds;
}

// reduce the original large matrix to list of 2 x 2 matrices
public static Matrix [] reduceMatrixTo2By2(double [][] matrix,
 int reductionSize) {

    Matrix [] matricesList = new Matrix[matrix.length * matrix.length /
                  (reductionSize * reductionSize)];

    int listCount = 0;
    for(int i = 0; i < matrix.length; i += reductionSize) {

        // make new matrix
```

```
        double [][] m = new double[reductionSize][reductionSize];

        // break down matrix into sub matrices of size reductionSize
        for(int j = 0; j < matrix[i].length; j += reductionSize) {

            for(int x = i; x < i + reductionSize; x++) {
                for(int y = j; y < j + reductionSize; y++) {
                     m[x % reductionSize][y % reductionSize] = matrix[x][y];
                }
            }

            //System.out.println("index = " + index);
            matricesList[listCount++] = new Matrix(m);
        }
    }

    return matricesList;
}

public static void printMatrices(Matrix [] matricesList) {

    for(Matrix m : matricesList) {

        printMatrix(m.getMatrix());
        System.out.println();
    }
}
```

```
// print a matrix
public static void printMatrix(double[][] matrix) {

    for(int i = 0; i < matrix.length; i++) {
        for(int j = 0; j < matrix[i].length; j++) {
            System.out.print(matrix[i][j] + "\t");
        }
        System.out.println();
    }
}

// print a vector
public static void printVector(double [] vector) {

    for(int i = 0; i < vector.length; i++)
      System.out.println(vector[i]);
}

public static void main(String [] args) {


    // tensor computation
    double [][] K = {
                     {1, 1, 1, 2},
                     {1, 10, 2, 1},
                     {2, 1, 10, 1},
                     {1, 2, 1, 1},
                    };
```

```
        System.out.println("Input Matrix");
        printMatrix(K);

        System.out.println();

        double [][] kPounds = computeKPounds(K);

        System.out.println("\nResult Matrix");
        printMatrix(kPounds);

/*

        // diagonal tensor computation
       double [][] K_DiagTensor = {
                       {1, 0, 10, 0},
                       {0, 1, 0, 10},
                       {1, 0, 10, 0},
                       {0, 1, 0, 10},
                      };
        System.out.println("Input Matrix - Diagonal Tensor");
        printMatrix(K_DiagTensor);

        System.out.println();
        double [][] kPounds =  computeKPoundsDiagTensor(K_DiagTensor);
        System.out.println("\nResult Matrix - Diagonal Tensor");
        printMatrix(kPounds);
```

```
*/

    }

}
```

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package HomogenizationProject;

import static HomogenizationProject.MatrixComputation.*;

/**
 *
 * @author Abibat Lasisi
 */

public class Generalization {

     public static final int M = 2;
      public static final int REGULAR_TENSOR = 1;
     public static final int DIAGONAL_TENSOR = 2;



     // print a matrix
    public static void printMatrix(double[][] matrix) {

        for(int i = 0; i < matrix.length; i++) {
            for(int j = 0; j < matrix[i].length; j++) {
                System.out.print(matrix[i][j] + "\t");
            }
```

```
            System.out.println();
        }
    }

    public static void printMatrices(Matrix [] matricesList) {

        for(Matrix m : matricesList) {

            printMatrix(m.getMatrix());
            System.out.println();
        }
    }

    // generate matrix
    public static double [][] generateMatrix(int n) {

   // change this matrix using random number generation based on some pattern
       /* double [][] K = {
                        {1, 1, 1, 2},
                        {1, 10, 2, 1},
                        {2, 1, 10, 1},
                        {1, 2, 1, 1}
                       };
*/

        double [][] K = {
                        {1, 0, 10, 0},
                        {0, 1, 0, 10},
```

```
                {1, 0, 10, 0},
                {0, 1, 0, 10},
               };

int matrixSize = (int) Math.pow(2, n);

if(matrixSize <= K.length)
    return K;

double [][] matrix = new double[matrixSize][matrixSize];

int row = -1;
for(int i = 0; i < matrix.length; i++) {

    row++;
    int col =  -1;

    for(int j = 0; j < matrix.length; j++) {

        col++;
        matrix[i][j] = K[row][col];
        col = col == 3 ? -1 : col;
    }

   row = row == 3 ? -1 : row;
}

return matrix;
```

```
}

// split matrix and store all the possible 2 x 2
// sub matrices generated from matrix
public static Matrix [] splitMatrix(double [][] matrix) {

    Matrix [] matricesList = new Matrix[matrix.length * matrix.length / 4];

    int matrixCount = 0;
    for(int i = 0; i < matrix.length; i += M) {

        // make new matrix
        double [][] m = new double[M][M];

        // break down matrix into sub matrices of size reductionSize
        for(int j = 0; j < matrix[i].length; j += M) {

            for(int x = i; x < i + M; x++) {
                for(int y = j; y < j + M; y++) {
                     m[x % M][y % M] = matrix[x][y];
                }
            }

            matricesList[matrixCount++] = new Matrix(m);
        }
    }

    return matricesList;
```

```
}

public static double [][] mergeMatrices
        (Matrix [] reducedMatricesList,int matrixSize) {

    if(reducedMatricesList == null)
        return null;

    int reductionSize = M;

    double [][] matrix = new double[matrixSize][matrixSize];

    int listCount = 0;
    for(int i = 0; i < matrix.length; i += reductionSize) {

        // make new matrix
        double [][] m = reducedMatricesList[listCount].getMatrix();

        // break down matrix into sub matrices of size reductionSize
        for(int j = 0; j < matrix[i].length; j += reductionSize) {

            for(int x = i; x < i + reductionSize; x++) {
                for(int y = j; y < j + reductionSize; y++) {
                     matrix[x][y] = m[x % reductionSize][y % reductionSize];
                }
            }
        }
```

```
            listCount++;
        }

        return matrix;
    }

// reduce a matrix of reductionSize x reductionSize to a matrix of
// reductionSize / 2 x reductionSize / 2
    public static double [][] reduceMatrix(double [][] matrix, int type) {

        int reductionSize = matrix.length / 2;

        // store all the possible 2 x 2 sub matrices generated from matrix
        Matrix [] matricesList = splitMatrix(matrix);

        // reduce the sub matrices in matricesList from
        // matricesList.length to matricesList.length / 4
        Matrix [] reducedMatricesList = new Matrix[matricesList.length / 4];
        int matrixCount = 0;

        int i = 0;
        int j = 1;
        int deadline = matrix.length;
        for(int count = 0; count < matricesList.length / 2;  count += 2) {

            double [][] m1 = matricesList[i].getMatrix();
            double [][] m2 = matricesList[j].getMatrix();
            double [][] m3 = matricesList[i + reductionSize].getMatrix();
```

```
        double [][] m4 = matricesList[j + reductionSize].getMatrix();

        Matrix m = new Matrix(m1, m2, m3, m4); // make a 4 x 4 Matrix object

        double [][] m_  = null;

        if(type == REGULAR_TENSOR)
         // reduce the matrix of the object to 2 x 2
            m_ = computeKPounds(m.getMatrix());
        else
            m_ = computeKPoundsDiagTensor(m.getMatrix());

        // make a 2 x 2 matrix object
        reducedMatricesList[matrixCount++] = new Matrix(m_);

        if(j + reductionSize + 1 < deadline) {
            i += 2;
            j += 2;
        }
        else{
            i = j + reductionSize + 1;
            j = i + 1;
            deadline += matrix.length;
        }
    }

    return mergeMatrices(reducedMatricesList, reductionSize);
}
```

```
    public static void main(String [] args) {

         double [][] matrix = generateMatrix(6);
         System.out.println("Matrix size = " + matrix.length);
         printMatrix(matrix);

        while(matrix.length > 2) {
            matrix = reduceMatrix(matrix, DIAGONAL_TENSOR);

            System.out.println("Matrix size = " + matrix.length);
            printMatrix(matrix);
        }

    }

}
```

Appendix D

Fast Transform in Two Dimensions

This appendix illustrate the Fast Transform algorithm in two dimensions. We provide Java codes that implement the Fast Transform Algorithm presented in Chapter 4. We compute the arithmetic average, the details and the inverse of the average to generate $K^{\sharp}$, the harmonic average and the arithmetic average of the permeability tensors.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package HomogenizationProject;

/**
 * @author Abibat Lasisi
 *
 */

import static HomogenizationProject.MatrixComputation.*;

public class FastTransform {

    public static final int X_DIRECTION = 0;
    public static final int Y_DIRECTION = 1;
```

```
       // diagonal tensor computation
   /* public static double [][] K_DiagTensor = {
                        {1, 0, 10, 0},
                        {0, 1, 0, 10},
                        {1, 0, 10, 0},
                        {0, 1, 0, 10},
                       };

*/

     public static double [][] K_DiagTensor = {
                        {2, 0, 1, 0},
                        {0, 2, 0, 1},
                        {2, 0, 1, 0},
                        {0, 2, 0, 1},
                       };




         // diagonal tensor computation
    /* public static double [][] K_DiagTensor = {
                    {1, 0, 2, 0},
                    {0, 1, 0, 2},
                    {3, 0, 4, 0},
                    {0, 3, 0, 4},
                   };
   */
```

```
public static double [][] kPound = null;
//public static double [][] matrixAverage = null;
public static double [][] k1Pound = null;
public static double [][] k2Pound = null;

public static double [][] computeAverage(double [][] m, double [][] m_) {

    double [][] matrix = new double[m.length][m.length];

    for(int i = 0; i < m_.length; i++){
     for(int j = 0; j < m_.length; j++){
         matrix[i][j] = (m[i][j] + m_[i][j]) / 2.0;
     }
 }

 return matrix;
}

public static double [][] computeAverage(Matrix [] matricesList) {

    double [][] matrix = new double[M][M];

     for(Matrix m : matricesList) {

         double m_[][] = m.getMatrix();
         for(int i = 0; i < m_.length; i++){
             for(int j = 0; j < m_.length; j++){
```

```
                matrix[i][j] += m_[i][j];
            }
        }
    }


    for(int i = 0; i < matrix.length; i++){
        for(int j = 0; j < matrix.length; j++){
            matrix[i][j] = matrix[i][j] / 4.0;
        }
     }

     return matrix;
}

// k1 - avg
public static double [][] computeDifference(double [][] k1, double [][] avg) {

    double [][] matrix = new double[k1.length][k1.length];

    for(int i = 0; i < matrix.length; i++) {
        for(int j = 0; j < matrix.length; j++) {
            matrix[i][j] = k1[i][j] - avg[i][j];
        }
    }


    return matrix;
```

```
}

// m + m_
public static double [][] computeSum(double [][] m, double [][] m_) {

    double [][] matrix = new double[m.length][m.length];

    for(int i = 0; i < matrix.length; i++) {
        for(int j = 0; j < matrix.length; j++) {
            matrix[i][j] = m[i][j] + m_[i][j];
        }
    }

    return matrix;
}

//multiply product by -1
 public static double [][] multiplyProdByNegOne(double [][] prod) {

    for(int i = 0; i < prod.length; i++) {

        for(int j = 0; j < prod.length; j++) {
            prod[i][j] = prod[i][j] == 0 ? 0 : prod[i][j] * -1;
        }
    }

    return prod;
 }
```

```
// computeDetail
public static double [][] computeDetail(double [][] diff,
               double [][] inv, int direction) {

    double [][] prod = matrixByMatrix(diff, inv);

    prod = matrixByMatrix(prod, diff);

    prod = multiplyProdByNegOne(prod);

    //set prod[0][0] or prod[1][1] to 0 depending on direction
    if(direction == X_DIRECTION) prod[1][1] = 0;
    else prod[0][0] = 0;

    return prod;
}

//
public static double [][] computeKPoundStratified(double [][] k1,
               double [][] avg) {

    double [][] matrix = new double[k1.length][k1.length];

    for(int i = 0; i < matrix.length; i++) {
        for(int j = 0; j < matrix.length; j++) {
            matrix[i][j] = k1[i][j] + avg[i][j];
```

```
        }
    }

    return matrix;
}

public static double [][] stratifiedProcedure(double [][] matrix,
              double [][] matrixAverage, int direction) {

    double [][] diff = computeDifference(matrix, matrixAverage);

   // compute inverse matrix of the average
    Matrix m = new Matrix(matrixAverage);
    double matrixInverse [][] = m.matrixInverse().getMatrix();

    // compute detail
    double [][] detail = computeDetail(diff, matrixInverse, direction);

    // compute kPounds
    return computeKPoundStratified(matrixAverage, detail);

}

public static double [][] fastTransStratifiedCase(double [][] K_DiagTensor) {

      // reduce the original matrix to list of 2 x 2 matrices
   Matrix [] matricesList = reduceMatrixTo2By2(K_DiagTensor, M);
```

```
   // compute average
   double [][] matrixAverage = computeAverage(matricesList);

   double [][] kPound =  stratifiedProcedure(matricesList[1].getMatrix(),
                         matrixAverage, X_DIRECTION);

   return kPound;
}

public static double [][] fastTransNonStratifiedCase
(double [][] K_DiagTensor) {

   // reduce the original matrix to list of 2 x 2 matrices
   Matrix [] matricesList = reduceMatrixTo2By2(K_DiagTensor, M);

   // first part
   double [][] k0 = matricesList[0].getMatrix();
   double [][] k1 = matricesList[1].getMatrix();

   /******************/
   double [][] matrixAverage = computeAverage(k0, k1);
   k1Pound =  stratifiedProcedure(k1, matrixAverage, X_DIRECTION);


   double [][] k2 = matricesList[2].getMatrix();
   double [][] k3 = matricesList[3].getMatrix();

   matrixAverage = computeAverage(k2, k3);
```

```
    k2Pound =  stratifiedProcedure(k3, matrixAverage, X_DIRECTION);

    double [][] k_12_Pound = computeAverage(k1Pound, k2Pound);

    //set k_12_Pound[1][1] to 0  for X_DIRECTION
    k_12_Pound[1][1] = 0;

    // second part

    matrixAverage = computeAverage(k0, k2);
    double [][] k3Pound =  stratifiedProcedure(k2, matrixAverage, Y_DIRECTION);

    matrixAverage = computeAverage(k1, k3);
    double [][] k4Pound =  stratifiedProcedure(k3, matrixAverage, Y_DIRECTION);

    double [][] k_34_Pound = computeAverage(k3Pound, k4Pound);

    //set k_34_Pound[1][1] to 0  for Y_DIRECTION
    k_34_Pound[0][0] = 0;

    return computeSum(k_12_Pound, k_34_Pound);
}


public static void main(String [] args) {

    System.out.println("Input Matrix - Diagonal Tensor");
    printMatrix(K_DiagTensor);
```

```
        // stratifiedCase
        kPound = fastTransStratifiedCase(K_DiagTensor);

        // non stratifiedCase
        kPound = fastTransNonStratifiedCase(K_DiagTensor);

      System.out.println("\nResult Matrix - Fast Transform Algorithm");
      printMatrix(kPound);

    }
}
```

Appendix E

Inverse Transform in Two Dimensions

We provide the implementation of the inverse transform algorithm presented in Chapter 4. The program generates the permeability tensors, $K's$ values from the average formula, $K^\sharp$. This procedure takes as input the $K^\sharp$ and the details, ΔK, to produce the K values.

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package HomogenizationProject;

/**
 * @author Abibat Lasisi
 *
 *
 */

import static HomogenizationProject.FastTransform.*;
import static HomogenizationProject.MatrixComputation.*;
import static HomogenizationProject.Generalization.mergeMatrices;

public class InverseTransform {

    public static final int STRATIFIED_CASE = 0;
```

```
public static final int NON_STRATIFIED_CASE = 1;

//multiply product by -1
 public static double [][] computeSquareRoot(double [][] matrix) {

    for(int i = 0; i < matrix.length; i++) {

        for(int j = 0; j < matrix.length; j++) {
            matrix[i][j] = Math.sqrt(matrix[i][j]);
        }
    }

    return matrix;
 }


 //multiply product by -1
 public static double [][] setEqual(double [][] m) {

    double [][] m_ = new double[m.length][m.length];

    for(int i = 0; i < m.length; i++) {

        for(int j = 0; j < m.length; j++) {
            m_[i][j] = m[i][j];
        }
    }
```

```
    return m_;
 }

 // do inverse transform for both cases
public static Matrix [] inverseTransform(double [][] matrixAverage,
 int caseType) {

    double [][] detail = new double[kPound.length][kPound.length];

    for(int i = 0; i < kPound.length; i++) {
        for(int j = 0; j < kPound.length; j++) {
            detail[i][j] = kPound[i][j] - matrixAverage[i][j];
        }
    }

    // multiply matrices
    double [][] diff_squared = matrixByMatrix(detail, matrixAverage);

    //multiply result by negative one
    diff_squared = multiplyProdByNegOne(diff_squared);

    //compute square root of the result
    double [][] diff = computeSquareRoot(diff_squared);

    //set diff[1][1] to diff[0][0]
    diff[1][1] =  diff[0][0];

    double [][] k1 = computeSum(diff, matrixAverage);
```

```
double [][] k0 = matrixByScalar(matrixAverage, 2);

k0 = computeDifference(k0, k1);

Matrix [] matricesList;

if(caseType == STRATIFIED_CASE) {
// set matrices equal to one another
    double [][] k2 = setEqual(k0);
    double [][] k3 = setEqual(k1);

    System.out.println("\nResult - k0");
    printMatrix(k0);

    System.out.println("\nResult - k1");
    printMatrix(k1);

    System.out.println("\nResult - k2");
    printMatrix(k2);

    System.out.println("\nResult - k3");
    printMatrix(k3);

    matricesList = new Matrix[4];

    matricesList[0] = new Matrix(k0);
    matricesList[1] = new Matrix(k1);
```

```
        matricesList[2] = new Matrix(k2);
        matricesList[3] = new Matrix(k3);

       return matricesList; //mergeMatrices(matricesList, 4);
    }
    else{

        matricesList = new Matrix[2];

        System.out.println("\nResult - k0");
        printMatrix(k0);

        System.out.println("\nResult - k1");
        printMatrix(k1);

        matricesList[0] = new Matrix(k0);
        matricesList[1] = new Matrix(k1);

        return matricesList;
    }
}



public static double [][] inverseTransStratifiedCase() {

    // stratifiedCase - called to obtain kpound
    kPound = fastTransStratifiedCase(K_DiagTensor);
```

```
    // recover the arithmetic average
    double [][] matrixAverage = { {kPound[1][1], 0},

                                  {0, kPound[1][1]}
                                };

    Matrix [] matricesList = inverseTransform(matrixAverage,
            STRATIFIED_CASE);

    Matrix m = new Matrix(matricesList[0].getMatrix(),
                          matricesList[1].getMatrix(),
                          matricesList[2].getMatrix(),
                          matricesList[3].getMatrix());

    return  m.getMatrix();
}


 // recover average
public static double [][] recoverAverage() {

    return null;
}

public static void recoverKpounds(double [][] k_12_Pound,
      double [][] k_34_Pound) {
```

```
        k_12_Pound[0][0] = kPound[0][0];
        k_34_Pound[1][1] = kPound[1][1];

    }

    public static void updateKPound(double [][] pound) {

        for(int i = 0; i < kPound.length; i++) {
            for(int j = 0; j < kPound.length; j++) {
                kPound[i][j] = pound[i][j];
            }
        }
    }

    public static double [][] inverseTransNonStratifiedCase() {

        // non stratifiedCase - called to obtain kpound
        kPound = fastTransNonStratifiedCase(K_DiagTensor);

        // recover k_12_Pound and k_34_Pound from kPound
////        double [][] k_12_Pound = new double[kPound.length][kPound.length];
////        double [][] k_34_Pound = new double[kPound.length][kPound.length];
////
////        recoverKpounds(k_12_Pound, k_34_Pound);
////
////        System.out.println("\nResult Matrix - k_12_Pound");
////        printMatrix(k_12_Pound);
////
```

```
////        System.out.println("\nResult Matrix - k_34_Pound");
////        printMatrix(k_34_Pound);


         // recover the arithmetic average **********
        double [][] matrixAverage = { {k1Pound[1][1], 0},

                                      {0, k1Pound[1][1]}
                                    };

        //update kPound
        kPound = k1Pound;
        //updateKPound(k1Pound);
        Matrix [] matricesList1 = inverseTransform(matrixAverage,
         NON_STRATIFIED_CASE);

        // recover the arithmetic average **********
        double [][] matrixAverage1 = { {k2Pound[1][1], 0},

                                       {0, k2Pound[1][1]}
                                     };
         //update kPound
        kPound = k2Pound;
        //updateKPound(k2Pound);
        Matrix [] matricesList2 = inverseTransform(matrixAverage1,
         NON_STRATIFIED_CASE);

        Matrix m = new Matrix(matricesList1[0].getMatrix(),
```

```
                                matricesList1[1].getMatrix(),
                                matricesList2[0].getMatrix(),
                                matricesList2[1].getMatrix());

        return  m.getMatrix();
    }


    public static void main(String [] args) {

       // inverse Transform for Stratified Case
       double [][] K_DiagTensor = inverseTransStratifiedCase();

       // inverse Transform for NonStratified Case
       //double [][] K_DiagTensor = inverseTransNonStratifiedCase();

      System.out.println("\nInverse Transform Result Matrix - Diagonal Tensor");
      printMatrix(K_DiagTensor);



    }

}
```

Appendix F

Additional Numerical Examples

Here, we provide more examples of the numerical results obtained from the implementation of the codes.

Example 1

```
run:
Matrix size = 4
1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0
1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0
Matrix size = 2
1.8181818181818183 0.0
0.0 5.5
BUILD SUCCESSFUL (total time: 1 second)
```

```
run:
Matrix size = 8
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
Matrix size = 4
1.8181818181818183 0.0 1.8181818181818183 0.0
0.0 5.5 0.0 5.5
1.8181818181818183 0.0 1.8181818181818183 0.0
0.0 5.5 0.0 5.5
Matrix size = 2
1.8181818181818183 0.0
0.0 5.5
BUILD SUCCESSFUL (total time: 1 second)
```

```
run:
Matrix size = 16
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0
0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0 0.0 1.0 0.0 10.0
Matrix size = 8
1.8181818181818183 0.0 1.8181818181818183 0.0 1.8181818181818183
0.0 1.8181818181818183 0.0
0.0 5.5 0.0 5.5 0.0 5.5 0.0 5.5
1.8181818181818183 0.0 1.8181818181818183 0.0 1.8181818181818183
0.0 1.8181818181818183 0.0
0.0 5.5 0.0 5.5 0.0 5.5 0.0 5.5
1.8181818181818183 0.0 1.8181818181818183 0.0 1.8181818181818183
0.0 1.8181818181818183 0.0
0.0 5.5 0.0 5.5 0.0 5.5 0.0 5.5
```

```
1.8181818181818183 0.0 1.8181818181818183 0.0 1.8181818181818183
0.0 1.8181818181818183 0.0
0.0 5.5 0.0 5.5 0.0 5.5 0.0 5.5
Matrix size = 4
1.8181818181818183 0.0 1.8181818181818183 0.0
0.0 5.5 0.0 5.5
1.8181818181818183 0.0 1.8181818181818183 0.0
0.0 5.5 0.0 5.5
Matrix size = 2
1.8181818181818183 0.0
0.0 5.5
BUILD SUCCESSFUL (total time: 1 second)
```

Example 2

```
run:
Matrix size = 4
1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0
Matrix size = 2
1.3333333333333333 0.0
0.0 1.5
BUILD SUCCESSFUL (total time: 1 second)
```

```
run:
Matrix size = 8
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
Matrix size = 4
1.3333333333333333 0.0 1.3333333333333333 0.0
0.0 1.5 0.0 1.5
1.3333333333333333 0.0 1.3333333333333333 0.0
0.0 1.5 0.0 1.5
Matrix size = 2
1.3333333333333333 0.0
0.0 1.5
BUILD SUCCESSFUL (total time: 1 second)
```

```
run:
Matrix size = 16
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0
0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0 0.0 1.0 0.0 2.0
Matrix size = 8
1.3333333333333333 0.0 1.3333333333333333 0.0
1.3333333333333333 0.0 1.3333333333333333 0.0
0.0 1.5 0.0 1.5 0.0 1.5 0.0 1.5
1.3333333333333333 0.0 1.3333333333333333 0.0
1.3333333333333333 0.0 1.3333333333333333 0.0
0.0 1.5 0.0 1.5 0.0 1.5 0.0 1.5
1.3333333333333333 0.0 1.3333333333333333 0.0
1.3333333333333333 0.0 1.3333333333333333 0.0
0.0 1.5 0.0 1.5 0.0 1.5 0.0 1.5
```

```
1.3333333333333333 0.0 1.3333333333333333 0.0
1.3333333333333333 0.0 1.3333333333333333 0.0
0.0 1.5 0.0 1.5 0.0 1.5 0.0 1.5
Matrix size = 4
1.3333333333333333 0.0 1.3333333333333333 0.0
0.0 1.5 0.0 1.5
1.3333333333333333 0.0 1.3333333333333333 0.0
0.0 1.5 0.0 1.5
Matrix size = 2
1.3333333333333333 0.0
0.0 1.5
BUILD SUCCESSFUL (total time: 1 second)
```

Abibat A. Lasisi

Department of Mathematics and Statistics
Utah State University
Logan, UT 84341, USA

435-890-6569
abibat.lasisi@aggiemail.usu.edu

Employment

- **Graduate Instructor** Aug. 2013 – June 2018
  Department of Mathematics and Statistics, Utah State University
- **Teaching Assistant** Aug. 2010 – May 2012
  Department of Mathematics and Statistics, Utah State University

Education

- **Ph.D., Applied Mathematics**, Utah State University, USA Aug. 2013 –July 2018
  GPA: 4.0/4.0
  Dissertation: Multi-resolution Analysis Using Wavelet Basis Conditioned on Homogenization
  Advisor: Dr. Joseph Koebbie
- **MMath, Mathematics**, Utah State University, USA Aug. 2010 – Aug. 2012
  GPA: 3.88/4.0
  Report: Validation Study : A Case Study of Calculus 1 (MATH 1210)
- **M.Sc., Computer Science**, University of Lagos, Nigeria Sept. 2006 – Jan. 2008
  Thesis: Implicit Numerical Methods for Solving Initial Value Problem of Ordinary Differential Equations
- **B.Sc., Mathematical Sciences**, Federal University of Agric., Abeokuta, Nigeria Apr. 1999 - May 2003
  Project: Numerical Solutions of Rational Functions Via Pade and Msehly's Methods of Approximation

Research Interests
Computational and applied mathematics with emphasis on:

- multi-resolution analysis (MRA) using wavelet basis conditioned on homogenization
- wavelet construction/reconstruction and analysis of numerical schemes

Teaching Interests

- Undergraduate and graduate algebra, calculus, trigonometry, geometry, numerical analysis, ODE, PDE, and numerical optimization courses

Technical Skills

- SAS (proficient), R (good), MATLAB (good), and LaTeX (proficient)

Awards and Honors

- 2013–Present : Graduate Tuition Award, Utah State University
- 2012–Present : Honoree, Golden Key International Honor Society
- 2011– 2012 : Joseph Reuel Harris Scholarship, College of Science, Utah State University
- 2011 : Mathematics and Statistics department's Scholarship, Utah State University

Professional Affiliation

- Member, Golden Key International Honor Society 2012 - present
- Member, Society for Industrial and Applied Mathematics (SIAM) 2014 - present
- Member, American Mathematical Society (AMS) 2014 - present

Abibat A. Lasisi

Refereed Publications

- Joseph V. Koebbe and **Abibat A. Lasisi**. Homogenization-Wavelet Reconstruction Methods for Elliptic Differential Equations [In Preparation].
- Joseph V. Koebbe, **Abibat A. Lasisi**, and Ju Yi. Construction of Wavelet Bases Conditioned on Elliptic PDE and Hyperbolic Conservation Laws. Rocky Mountain Partial Differential Equations Conference, Brigham Young University, May 18 – 19, 2017 [Abstract].
- Ramoni O. Lasisi and **Abibat A. Lasisi**. Improved Heuristic for Manipulation of Second-order Copeland Elections. In proceedings of the 3rd Global Conference on Artificial Intelligence (GCAI 2017), Miami, Florida, USA, 18–22 October 2017, pp. 162 –174.
- Ramoni O. Lasisi and **Abibat A. Lasisi**. Bounds on Manipulation by Merging in Weighted Voting Games. In the 6th International Workshop on Computational Social Choice, Toulouse, France, June 22 - 24, 2016.
- Ramoni O. Lasisi and **Abibat A. Lasisi**. The Shapley Value in Voting Games: Computing Single Large Party's Power and Bounds for Manipulation by Merging. In proceedings of the 28th International Florida Artificial Intelligence Research Society Conference, Florida, USA, May 18 - 20, 2015, pp. 55 – 60.
- Ramoni O. Lasisi and **Abibat A. Lasisi**. Manipulation of Second-Order Copeland Elections Using Branch-and-Bound Heuristic. In proceedings of the 28th International Florida Artificial Intelligence Research Society Conference, Hollywood, Florida, USA, May 18 - 20, 2015. [Poster abstract].

Teaching Experience

- Summer 2018 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching a section of Pre-Calculus course, **MATH 1060 - Trigonometry**.
- Spring 2018 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching two sections of Pre-Calculus course, **MATH 1060 - Trigonometry**.
- Fall 2017 : (1) **Teaching Assistant**, Department of Mathematics and Statistics, Utah State University. Teaching assistant for **MATH 2250 - Differential Equations and Linear Algebra**. My duties include teaching one section of recitation class, proctoring tests, grading, providing valuable feedbacks on quizzes and holding office hours. (2) **Grader:** Grader for **STAT 5200 - Design of Experiments**.
- Summer 2017 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching a section of Pre-Calculus course, **MATH 1050 - College Algebra**.
- Spring 2017 : **Teaching Assistant**, Department of Mathematics and Statistics, Utah State University. Teaching assistant for **MATH 0995 - College Mathematics Preparation**. My duties include teaching three sections of recitation classes, proctoring tests, grading, providing valuable feedbacks on quizzes and holding office hours.
- Summer 2016 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching a section of Pre-Calculus course, **MATH 1050 - College Algebra**.
- Spring 2016 : **Teaching Assistant**, Department of Mathematics and Statistics, Utah State University. Teaching assistant for **MATH 1210 - Calculus I**. My duties include teaching two sections of recitation classes, proctoring tests, grading, providing valuable feedbacks on quizzes and holding office hours.
- Fall 2015 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching two sections of Pre-Calculus course, **MATH 1060 - Trigonometry**.

Abibat A. Lasisi

- Summer 2015 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching two sections of Pre-Calculus course, **MATH 1060 - Trigonometry**.
- Spring 2015 : **Teaching Assistant**, Department of Mathematics and Statistics, Utah State University. Teaching Assistant for **Math 1050 - College Algebra**. My duties include **teaching three recitation classes**, proctoring tests, grading, providing valuable feedbacks on quizzes, and holding tutor hours.
- Fall 2014 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching a lower level mathematics course, **MATH 1010 - Intermediate Algebra**.
- Summer 2014 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching a lower level mathematics course, **MATH 0990 - Beginning Algebra**.
- Spring 2014 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching a lower level mathematics course, **MATH 0990 - Beginning Algebra**.
- Fall 2013 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching a lower level mathematics course, **MATH 0990 - Beginning Algebra**.
- Spring 2012 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching a lower level mathematics course, **MATH 0990 - Beginning Algebra**.
- Fall 2011 : **Graduate Instructor**, Department of Mathematics and Statistics, Utah State University. Teaching a lower level mathematics course, **MATH 0990 - Beginning Algebra**.

  My duties as a **Graduate Instructor from Fall 2011 to Falll 2014** included teaching the assigned courses, grading, providing valuable feedbacks (on tests, homework, exams), and holding office hours.
- Spring 2011 : **Teaching Assistant**, Department of Mathematics and Statistics, Utah State University. Teaching Assistant for **MATH 2210 - Multivariate Calculus**. My duties included grading, providing valuable feedbacks on homework, and holding office hours.
- Fall 2010 : **Teaching Assistant**, Department of Mathematics and Statistics, Utah State University. Teaching Assistant for **MATH 1100 - Calculus Techniques**. My duties included **teaching three recitation classes**, grading, providing valuable feedbacks on homeworks, and holding office hours.
- August 2010 : Completed and passed the **International Teaching Assistants' Workshop** Pre-Fall 2010. Evaluation of the workshop is based on participant's comprehensibility in a teaching role using the following factors: pronunciation, fluency, organization, and classroom interaction.

References and Contact Information

1. **Dr. Joseph V. Koebbe**
   Associate Professor and Ph.D. Advisor
   Department of Mathematics & Statistics
   Utah State University
   Phone: 435-797-2825
   Email : joe.koebbe@usu.edu
2. **Dr. James S. Cangelosi**
   Professor
   Department of Mathematics & Statistics
   Utah State University
   Phone: 435-797-1415
   Email : jim.cangelosi@usu.edu

Abibat A. Lasisi

3. **Dr. Nghiem Nguyen**
Associate Professor
Department of Mathematics & Statistics
Utah State University
Phone: 435-797-2819
Email : nghiem.nguyen@usu.edu