University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/Specialization: Petroleum Engineering (5 Year Program) / Reservoir Engineering	Spring semester, 2017 Open access
Writer: Abdul Razzaq A. Aqrawi	…… (Writer's signature)
Faculty supervisor: Steinar Evje External supervisor(s): Helmer André Friis	
Thesis title: Two-Dimensional Fluid Flow in Heterogeneous Porous Media Using Finite Analytic Method	
Credits (ECTS): 30	
Key words: Reservoir, Engineering, Porous Media, Fluid Flow, Simulation, Discretization, Numerical Method, Scheme, Heterogeneous, Permeability, Isotropy, Anisotropy,	Pages: ……75…… + enclosure: ……62…… Stavanger, …June/2017….. Date/year

Two-Dimensional Fluid Flow in Heterogeneous Porous Media Using Finite Analytic Method

Abdul-Razzaq A. Aqrawi

Master Thesis

Simulation and Reservoir Engineering

Faculty of Science and Technology

Department of Petroleum Engineering

June, 2017

Table of Contents

List of Figures

Acknowledgements

This work is the result of a Master thesis project assigned by the Department of Petroleum Engineering at the University of Stavanger. I would like to first and foremost thank my supervisor Steinar Evje for his support throughout the project, and for taking a chance on me.

I would also like to give special thanks to my supervisor from the International Research Institute of Stavanger (IRIS), Helmer André Friis, for believing in me and giving me a chance to prove myself. Without his steadfast encouragement, feedback, patience, and support, this project would have never come to fruition. With his great understanding and dedication; the initiation, testing, and conclusions were made possible.

Through the support and sponsoring of IRIS, all the time and resources needed were made available to both myself and everyone involved in the project.

I would like to give special thanks to my family, especially to my brother, Ahmed A. Aqrawi, and my father, Adnan A. M. Aqrawi, for their incredible support and guidance throughout the entirety of the project. Their invaluable help and knowledge made the scope and quality of the project possible.

Thank you all.

Stavanger, 2017

Abstract

This study started by exploring the different discretization methods that have been explored and developed throughout the years for reservoir simulators. Understanding simulators in general and how they are used in the oil and gas industry was essential to fully grasp the extent that this topic has been investigated. An interesting and new development in this field comes from a couple of individuals from a research group at the University of China in the department of Thermal Science and Energy Engineering. They utilized analytical techniques as a bases for solving for heterogeneous reservoirs. The most common method used in the industry today is the harmonic mean method for discretizing/representing the permeability at the grid interfaces. This has been proven to highly under estimate the effective permeability of heterogeneous reservoirs. Especially as the permeability ratio between the grid cells increase. The results would have a low resolution, and where there is low resolution in the effective permeability calculation, there are inaccuracies in the pressure field. And so, these researchers developed a method, the finite analytic method, and it proved to be highly accurate on both synthetic checkerboard data, and real data. Even more surprising, very little grid refinement was needed to achieve results with minimal error margins (Liu & Wang, 2013). Thus, an extension to this method was decided to be the main focus of this study. More specifically, enhancing the method to be able to solve for anisotropic permeabilities.

Starting by first implementing the method described by the article. This proved to be a very challenging task as insufficient information was supplied by the article. However, what most puzzling was the fact the pressure equations presented proved to be incorrect and the results erroneous. Therefore, a great amount of time was dedicated to first understanding the method on a fundamental and mathematical level, and then actually correctly derive and express the pressure equations. After the corrections were done, the results become directly comparable with that which was presented in the article. This was compared to other industry standard methods, mainly geometric mean and harmonic mean, and the finite analytic method proved to be much more reliable and much more accurate. The test were done on a mirrored checkerboard, and it was tested with varying grid refinements (4x4, 16x16, 64x64) and varying permeability ratios (1:2, 1:10, 1:100, 1:1000, 1:10000). The permeability tested here was isotropic.

Lastly, a novel technique was developed for solving anisotropic permeabilities. Basing the method on the core concepts of the finite analytic method for isotropic permeability discussed in the article, an anisotropic extension was derived and implemented. However, due to the time required to correct for the pressure equations earlier, little time was left to fully implement the anisotropic approach. As such, there was not enough time to adapt the necessary calculations that are needed to solve for the boundaries for anisotropic permeabilities. Therefore, this method could only be tested on isotropic data. Nevertheless, the method gave identical results to that of the isotropic approach, validating the methodology applied to it. Therefore, the first component that should be implemented in the future should be the MPFA method to solve for anisotropic permeabilities at the boundary (Aavatsmark, Reiso, Reme, & Teiland, 2001). Other than that, so that the method is comparable with current industry standard software, multi phase flow and three dimensional solutions should be derived as well. Unstructured grids could be explored once these other, more vital, parts are applied.

1. Introduction

Reservoir simulation have always been a vital part of geological modelling. It can yield important information about the behavior of the reservoir and the fluids-and-gases inside. Understanding fluid flow in a porous media can help determine the ideal parameters and circumstances for extraction. Additionally, indication of the pressure-and-temperature distribution as well as the heterogeneity of the reservoir can lead to dramatic changes in both the method and execution of the fluid withdrawal process. Therefore, accurately and precisely simulating the reservoir is of great importance, and has stark consequences for drilling.

Numerical methods has been an important tool in reservoir simulations. They are the mathematical building blocks to which simulators are built on. Numerical schemes are crucial for solving the fluid flow and pressure distribution for large and complex reservoirs. One of the major challenges facing the industry today is the fact that accuracy and high level of detail, especially in intricate heterogeneous reservoirs, is inherently difficult to incorporate in reservoir models. The commonly used numerical methods have a hard time resolving such situations, and if to be used, would require heavy compute power and time.

1.1. Motivation

Resolving highly detailed, intricate, and heterogeneous reservoirs is one of the biggest challenges for reservoir simulation. Thanks to recent improvements in geoscience techniques, it is now possible to attain very accurate reservoir models with precise portrayal of all the geological substances present. Unfortunately, most of the current discretization methods that are used rely heavily on principles that were established and applied when reservoir models were much less exhaustive and comprehensive. As such, it is becoming increasingly more difficult to accurately simulate such models without the need of extensive compute power and/or compute time, through the use of particularly fine grids. This study will therefore try to determine a method and approach that will be able to resolve and delineate solutions that are greatly accurate for vastly heterogeneous reservoirs, without the need to use fine grids.

1.2. Objective

The main focus for this thesis is to apply a new and enhanced discretization method for fluid flow in a porous media. Through the use of analytical methods, the approach should be able to solve discontinuities and inconsistent structures in the reservoir model without having very fine grids or ad-hoc methods. This can be used to not only save time and compute power, but also give an overall greater understanding of the model, as well as result in a more accurate simulation, specifically for heterogeneous reservoirs.

1.3. Hypothesis

Numerical methods and solutions are a vital part of reservoir simulation. They allow for issues due to complex behavior to be solved, such as; multiphase flow, nonlinearity, and heterogeneity to name a

few. Analytical solutions, although can lead to exact solutions to be calculated, cannot be used to solve such difficult and complicated problems. Analytical solutions are usually achieved by making assumptions that simplify such aspects as boundary conditions, properties, and geometry. These assumptions however, can often make it impossible to solve the fore mentioned problems.

Geological modelling data is regularly too vast and/or too detailed for numerical methods to solve directly. As such, computations are usually performed with upscaled transport properties. There are various upscaling methods used per today, the most common of which will be mentioned below (Background Research). These upscaling methods are often derived from both numerical-and-analytical solutions to integrate micro scale properties in to the macro scale.

The hypothesis of this study is that by using the power law behavior, that is the most prevalent behavior present in heterogeneous reservoirs, an analytical based local solution can be derived. After which, a numerical scheme can be constructed based on this solution, and used to examine heterogeneous reservoirs.

2. General Background

This section focuses on the importance of reservoir simulation. The relevance of simulators in the industry will be examined briefly. In addition, a little information about the concepts of numerical and analytical methods will be given. Lastly, a comparison between the two methods, and a quick look at the benefits and limitations of both will be considered.

2.1. Reservoir Simulation

Reservoir simulation is referred to as "the process of inferring the behavior of a real reservoir from the performance of a mathematical model of that physical system" (Soleng & Holden, 1998). It has become ingrained in the oil and gas industry. So much so, that most aspects of reservoir engineering problems are and can be solved using simulators. There is a simulator for everything from well testing to EOR (Enhanced Oil Recovery) predictions (Islam, Hossain, Mousavizadegan, Mustafiz, & Abou-Kassam, 2016). Simulation is a combination of physics, mathematics, and computer programming, coming together to develop a tool for estimating and predicting hydrocarbon behavior under various situations and operations. Figure 2.1 details the necessary steps that are involved when developing a reservoir simulator (Odeh, 1982). The purpose of simulators are to take an environment and all the necessary forces and characteristics, and then simulate/imitate the reaction and feedback of the environment and all the elements involved given a set boundary.

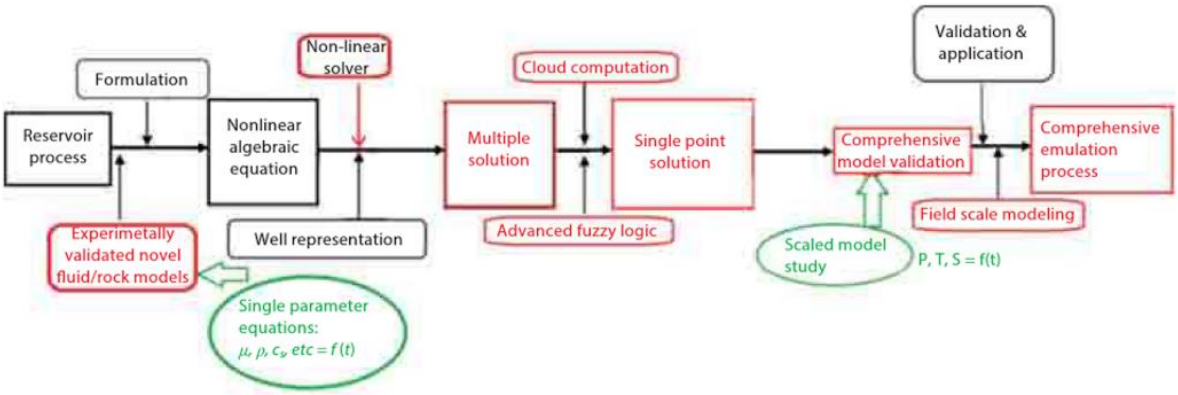

Figure 2.1: The major steps involved in reservoir simulation development and procedure (Odeh, 1982)

A very similar objective is present for the reservoir; to simulate the behavior of all the components involved (fluids, geo-mechanics, etc.) without the cost or effort of testing it in real life. However, what makes reservoir simulators so different from most others is largely by the fact that the portrayal and model of the reservoir, coupled with the boundary conditions and flow calculations of porous media, have a great deal of uncertainty. The pore systems and the flow patterns through them occurs on a level of detail that is near impossible to model or even characterize (Pettersen, 2006).

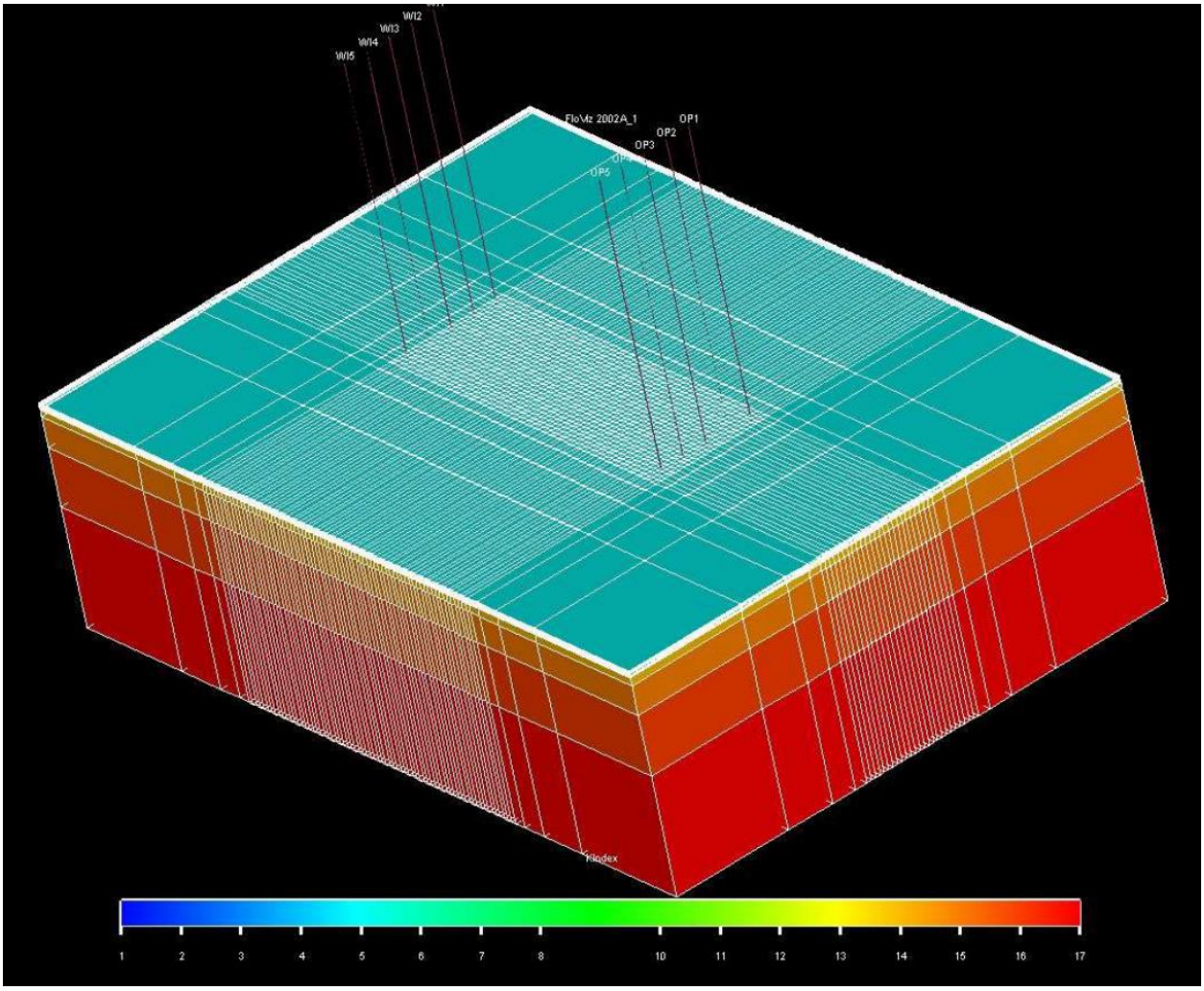

Figure 2.2: A regular Cartesian grid pattern (Pettersen, 2006)

The structure of the pore system is so complex and unsystematic, that without the possibility to scan or extract the reservoir on the nano scale, the system will remain largely unknown. On the other hand, even if it were possible to achieve such a level of detail in the model, it would result in a problem that is too large and intensive to solve for computers presently. The uncertainty that is incorporated in these calculations is what make them so increasingly difficult to simulate accurately (Peaceman, 1977).

These problems occur essentially due to; the generalization and upscaling from micro scale to macro scale from subsurface data such as seismic and well logs, and the simplification and/or uncertainty in the model and calculations themselves. However, despite these hindrances, reservoir simulation is commonly used with great success. It is still one of the fundamental parts used for decision making in the industry. Not only can yield vital information about the reservoir and the flow patterns, but also highlight areas that need to be investigated further (Carlson, 2006).

2.2. Gridding in Reservoir Simulation

A reservoir description is a model that maps the geology of a region. The geological data is often obtained from well logs, seismic, or other similar techniques. This is then used to create a geological model, which is often of fine scale reflecting the input information, such as core samples. Though, through upscaling and over generalization of the data, coupled with the necessity of having manageable computations, leads to simulators having to have a coarser scale. As such, simulators grid the information, where some statistical method is applied, and makes it more feasible to perform calculations on (Soleng & Holden, 1998). Gridding is an essential part of any numerical reservoir simulation.

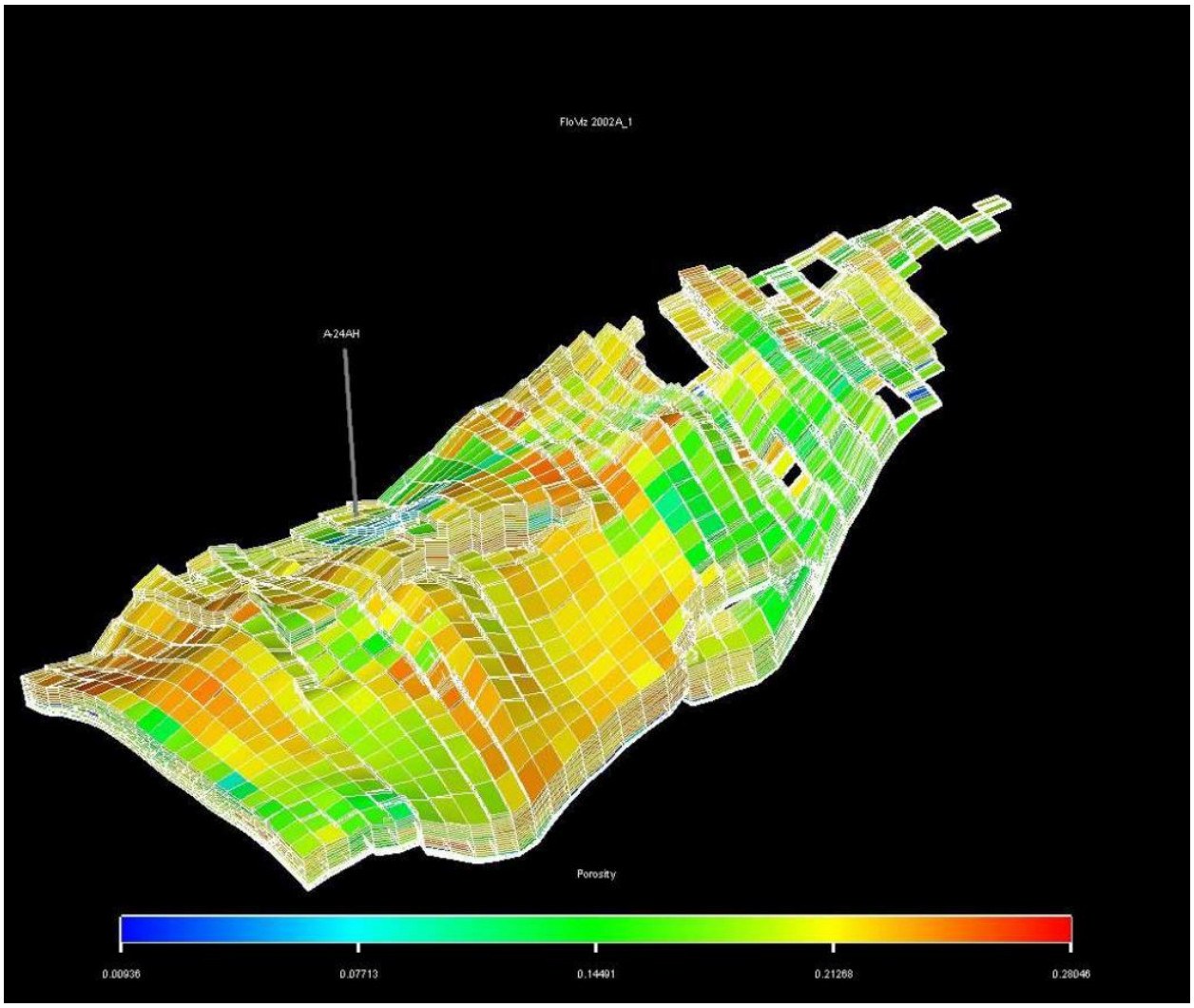

Figure 2.3: A regular Cartesian grid pattern as found in a reservoir model. Varying grid lengths and dipping structures are taken into account when creating such a model (Pettersen, 2006)

When reservoir simulators were first being introduced into the industry, Cartesian grids (rectangular/cuboidal) were what was most commonly used (Cao, 2002). Radial grids were then later developed to simulate flow near the well bore (Pedrosa & Aziz, 1985), and then local grid refinement

was established to attain higher accuracy in regions of either high flow or where more information is available (Nacul, 1991). Not too long thereafter, a technique referred to as corner point gridding was developed and introduced to the industry (Ponting, 1989). This ushered a new and radical way of approaching the subject matter of gridding. Corner point gridding made it possible to design grid blocks that are non-rectangular (Peaceman, 1996), making it possible to model faults and other intricate geological features more accurately and with more precise geometry.

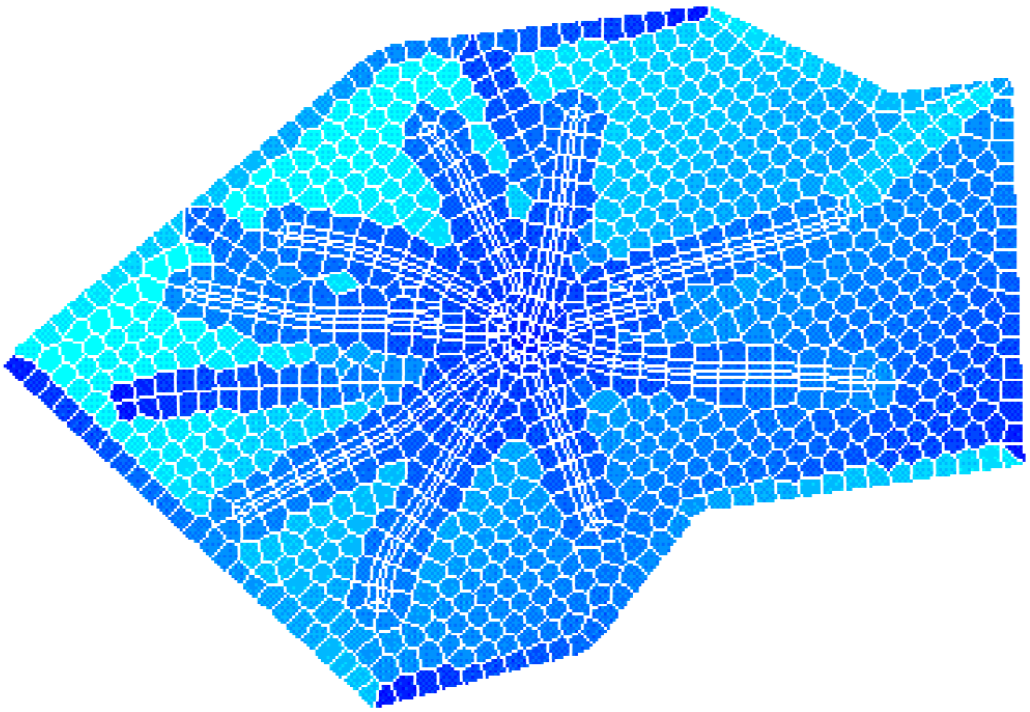

Figure 2.4: An example of an unstructured grid, where faults and multilateral wells can be seen as well (Cao, 2002)

In the last few decades, there has been a large focus on unstructured grids (Aavatsmark, Barkve, & Mannseth, 1998). Similar to the concept of corner point gridding, unstructured grids can adapt to geological features. The way in which it can achieve that is by allowing the grids to be flexible in nature, non-orthogonal, and can contain multiple points. This allows for not only being able to model complex geological structures, but also be used to varying sizes for the grid blocks, performing similarly if not better than local grid refinement and corner point gridding (Prévost, Lepage, Durlofsky, & Mallet, 2005).

3. Background Research

This section looks closer at the different discretization methods typically used in the industry. The technologies that will be described in this section are considered relevant to this study. There will also be a discussion on the methods and their use. More specifically, the importance of applying the correct method in the correct circumstance, and a few new advances within this field.

3.1. Discretization

Analytical methods result in continuous solutions with regards to space and time, whereas numerical methods result in discrete solutions at specific points in time and space (Cheng, 2012). The gridding represents the space domain, which is then iterated/discretized over the time domain, also referred to as time steps.

This chapter will look at the basics of; finite difference and finite element, geometric averaging, and harmonic averaging. A quick overview of these techniques will be considered here. If more details and in depth analysis of the methods are desired, please refer to the various references provided both in this chapter and in the References section.

3.2. Finite Difference & Finite Element

As mentioned earlier, to obtain analytical solutions one must make several simplifying assumptions. However, is many cases, these assumptions are not applicable. As such, approximation methods using numerical techniques must be used. Two such methods are the finite difference and the finite element methods (Wang & Anderson, 1982). These methods make it possible to operate and solve the differential equations that make up the reservoir model.

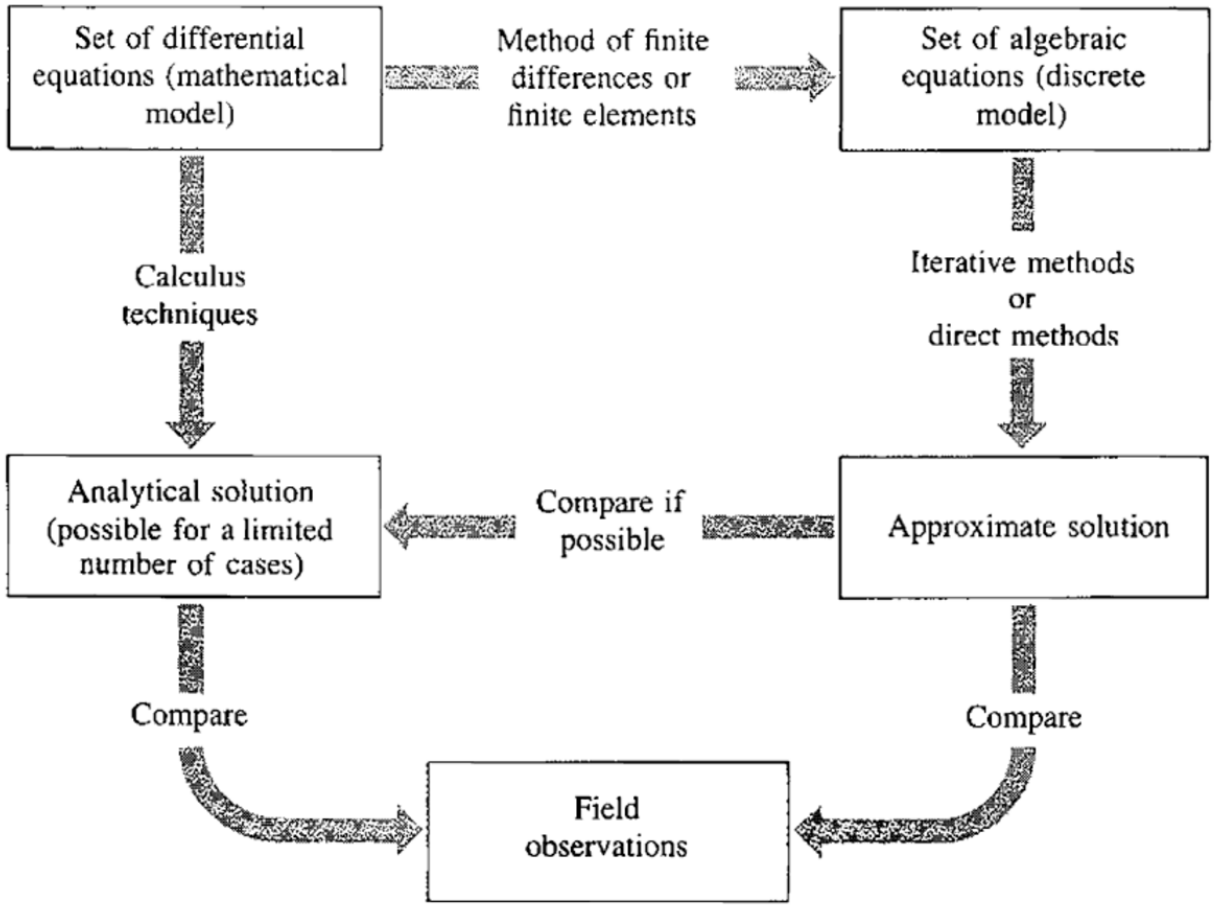

Figure 3.1: An overview of the relationship between the different components that are involved in models and solutions (Wang & Anderson, 1982)

Finite difference is an approximation based on the Taylor's series expansion (Smith, 1985). This is used in most commercial reservoir simulators for solving fluid flow equations. The core of the method involved is to replace the differential equations with difference equations between nodal points. Finite element differs slightly from the finite difference method. The idea is rather than simplifying the differential equations into difference equations, the solution is acquired by continuously interpolating the between the nodal points (Rao, 2011).

Finite difference is the method that is most commonly used in the industry and is the method that most approximation methods are based on.

3.3. Pressure Equation for One Phase Flow

Reservoir simulators are built up of mathematical models, consisting of a set of equations that describe fluid flow and the boundary conditions in a reservoir. The fluid flow is governed by the conservation of mass, momentum, and energy. This is most often described by Darcy's law which defines the linear relationship between the viscosity and the pressure head (Darcy, 1856). (Chen, Huan, & Ma, 2006). In

this study, the calculations are performed on single phase flow, to simplifying the approach, but can then be generalized and expanded for multi phase flow.

$$\nabla \cdot (k\nabla P) = 0$$

Equation 1

The general single phase pressure equation, which is described in Equation 1, can be integrated by using the divergence theorem (also known as Gauss's theorem). This can be defined as the outcome that relates the flux of a vector field through a surface to that of the vector field inside the surface (Katz, 1979). In other words, the volume integral of the divergence over the region inside the surface is equal to the outward flux of a vector field through a closed surface, as illustrated in Equation 2.

$$\iiint_V (\nabla \cdot F) dV = \oiint_S (F \cdot n) dS$$

Equation 2

Applying the divergence theorem to the pressure equation from Equation 1, the result in a two dimensional environment would be as follows:

$$\int_S [(k\nabla P) \cdot \vec{n}] dS = 0$$

Equation 3

In order to manage Equation 3 numerically, a discretization is introduced such that the domain for which the equation is valid is completely covered by a set of non-overlapping control volumes. The control volumes examined and used in this study are standard structured quadrilaterals. When relating Equation 3 to a control volume, the surface, which is referred to as S in Equation 3, would represent the boundary of the control volume. It is also useful to define a flux along a given edge of a control volume. This flux is denoted as Q_i, and reads as follows:

$$Q_i = \int_{S_i} [(k\nabla P) \cdot \vec{n}] dS$$

Equation 4

When using discretization standard approaches such as geometric averaging or, the industry standard, harmonic averaging, the fluxes are usually computed along the entire edge of the grid block, and in the direction facing out of the control volume. This can be seen illustrated in Figure 3.2 below. Thus, when calculating the flux in a direction, it follows, for example, in the form of:

$$Q_i = k_i \frac{P_{j+1} - P_j}{\Delta x} \Delta S$$

Equation 5

Where j is a control volume index, $j+1$ is a direct neighboring control volume, and the permeability is expressed using a discretization method such as the two mentioned above, namely geometric mean or harmonic mean. These are expressed in Equation 6 below. There are, of course, several other discretization methods. However, these are the two most commonly used in reservoir simulators, and are explained more generally and in a little more detail in the following sub sections.

$$k_{HM} = \frac{2k_j k_{j+1}}{k_j + k_{j+1}}$$

$$k_{GM} = \sqrt{k_j k_{j+1}}$$

Equation 6

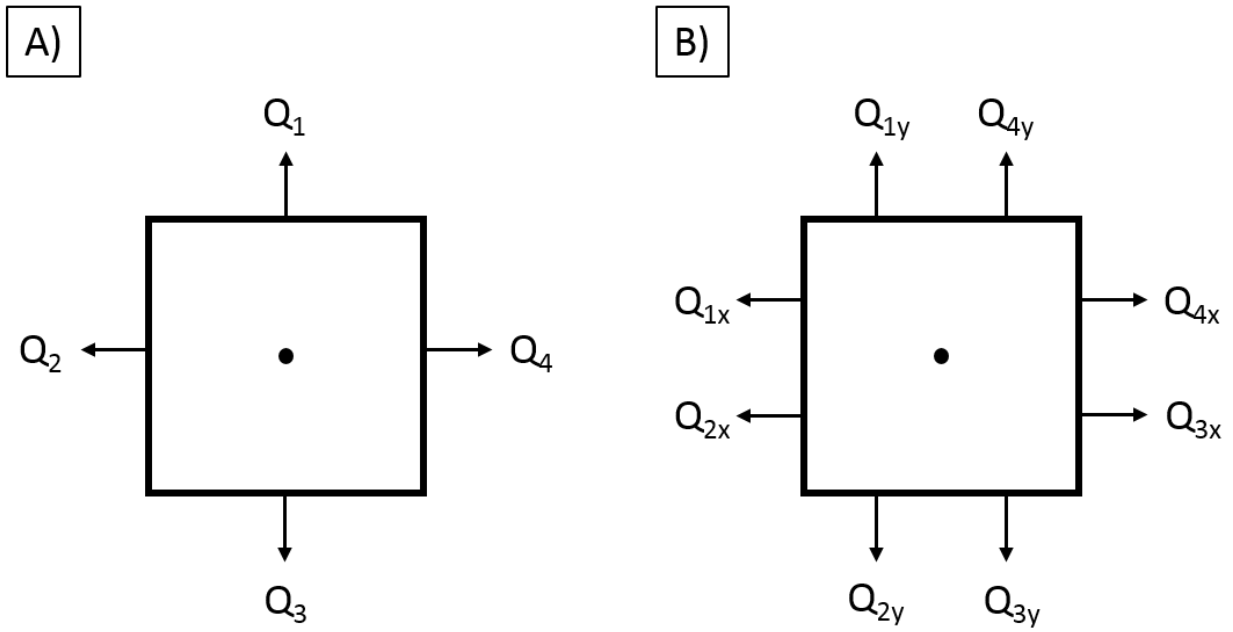

Figure 3.2: Illustrating the standard whole edge flux approach (a), and the refined half flux approach used in this study (b) on a control volume

A discretized flux calculation, of Equation 3 can now be expressed as a sum of the four fluxes that are calculated from the control volume edges, for example:

$$\sum_{i=1}^{4} Q_i = Q_1 + Q_2 + Q_3 + Q_4 = 0$$

Equation 7

The method that is explored in this study, however, applies a certain level of refinement when calculating the fluxes. Rather than computing the flux on the whole edge of the control volume, a half edge approach is used where each of the four fluxes are separated into x and y components. A demonstration of this can be seen in Figure 3.2 above. This allows for a more distinguished and precise

result, as more refined information is being calculated. The sum of the fluxes can then be expressed as follows:

$$\sum_{i=1}^{4} Q_{i_x} + Q_{i_y} = \left[Q_{1_x} + Q_{1_y}\right] + \left[Q_{2_x} + Q_{2_y}\right] + \left[Q_{3_x} + Q_{3_y}\right] + \left[Q_{4_x} + Q_{4_y}\right] = 0$$

Equation 8

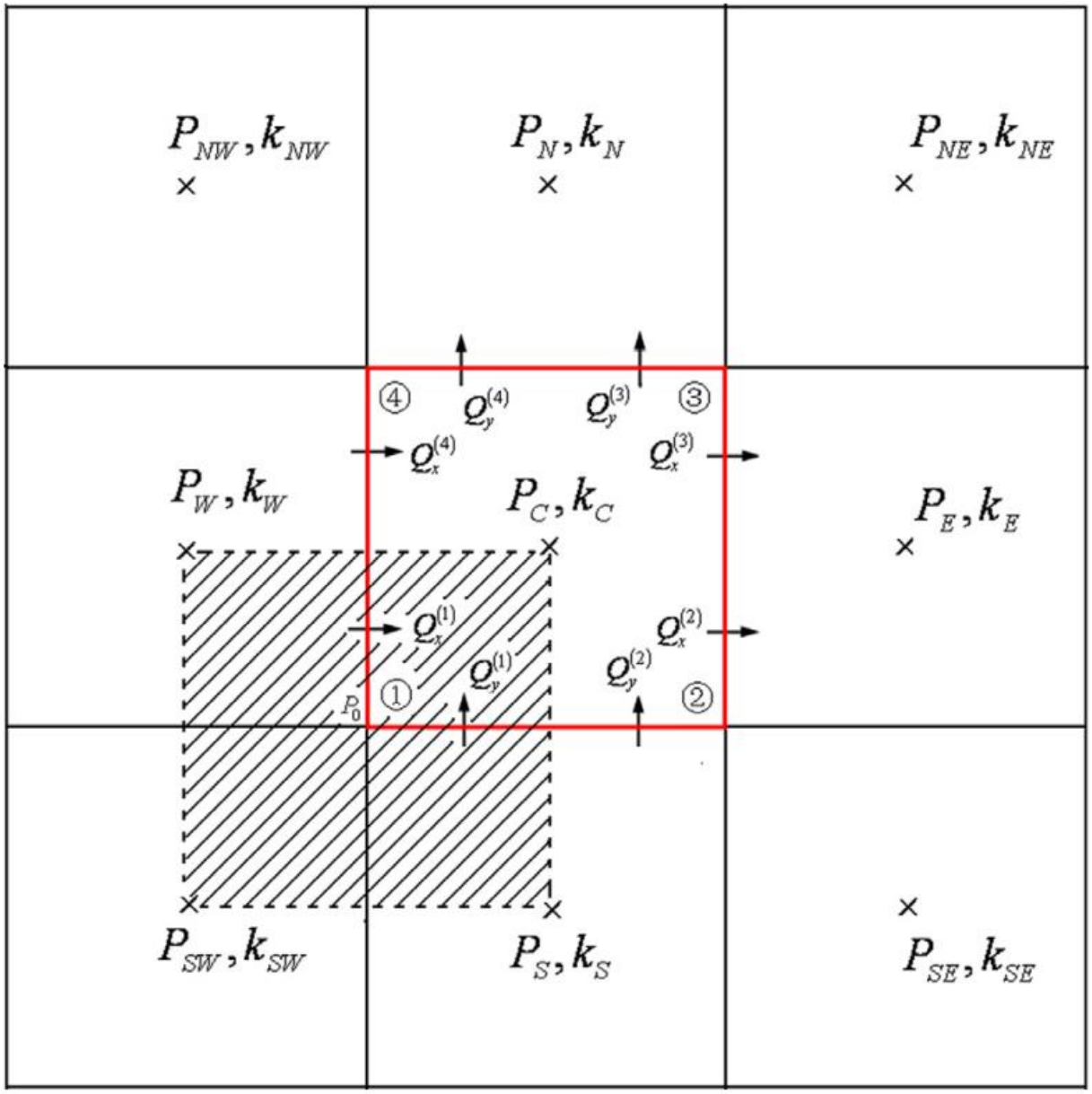

Figure 3.3: A sketch of the method described in this study, where half edge fluxes are illustrated, and the shaded region represents the influenced area of the grid node (Liu & Wang, 2013)

The fluxes that are utilized are the ones that can be seen highlighted in Figure 3.3. However, in actual implementation, the fluxes are calculated around the grid node and use the grid node as the reference point. The derivation and use of these fluxes are explored towards the end of the

Finite Analytic Method section, after the analytical nodal solution, which this is based on, is presented.

3.4. Effective Permeability

Usually, when permeability is mentioned, it is the absolute permeability that is being referenced. Absolute permeability is "The measurement of the permeability, or ability to flow or transmit fluids through a rock, conducted when a single fluid, or phase, is present in the rock" (Oilfield-Glossary, 2017). This can be determined by taking a bases in Darcy's law, which can be seen in Equation 9.

$$\vec{v} = -k\nabla P \rightarrow Q = -A\frac{k\Delta P}{\mu L} = -A\frac{k}{\mu}\nabla P$$

Equation 9

The effective permeability k_{eff}, also known as the equivalent permeability, however, is defined as "The ability to preferentially flow or transmit a particular fluid when other immiscible fluids are present in the reservoir (e.g., effective permeability of gas in a gas-water reservoir). The relative saturations of the fluids as well as the nature of the reservoir affect the effective permeability" (Oilfield-Glossary, 2017). However, the interest here lies in the effective permeability in the case of one phase flow. Technically, this is the same as upscaling. The permeability field is represented at the grid scale, meaning a single permeability per control volume. In effect, the following upscaling procedure results in a single effective permeability at the domain scale (coarse scale).

Considering, for example, that there is a single phase flow in the positive x direction, where the upper and lower boundaries are closed. The flow rate though the right boundary can be written as such:

$$Q = \int_S \vec{V} \cdot \vec{n} dS = -\int_S k\nabla P \cdot \vec{n} dS$$

Equation 10

Which follows from the definition of the Darcy flow rate defined in Equation 9. If the pressure and the permeability are known, the flux can be computed. When exploring the expression for the Darcy flow rate on the domain scale, the flux can then be expressed as follows when taking into account the same example presented earlier, flow in the positive x direction, where the upper and lower boundaries are closed:

$$Q = \int_S \vec{V}_{eff} \cdot \vec{n} dS = V_{eff} \Delta y$$

$$V_{eff} = -k_{eff} \frac{\partial P}{\partial x} = -k_{eff} \frac{\Delta P}{\Delta x}$$

$$k_{eff} = -\frac{Q}{\Delta P} \frac{\Delta x}{\Delta y}$$

Equation 11

Where it is noted that the flux is computed from the grid scale pressure field.

3.5. Geometric Averaging

The geometric mean is a form of averaging that indicate the dominant tendency of a set of numbers/values. It achieves this by using the product of these values. It is defined as follows:

$$\left(\prod_{i=1}^{n} x_i \right)^{\frac{1}{n}} = \sqrt[n]{x_1 x_2 \cdots x_n}$$

Equation 12

The geometric mean is often used when comparing different items where each item has multiple properties with different numeric ranges (Crawley, 2005). One thing to note, however, is that this method of averaging cannot accept values of differing signs, as it would lead to complex numbers with imaginary parts. Additionally, series of numbers that involve the number zero cannot produce viable results, as it is a product sum and will lead to results that are simply zero. If the geometric mean is desired in such situations, the number zero needs to be excluded from the series.

This is used in reservoir simulation and can give rather accurate results in regions where there is high permeability contrast. However, it is highly unstable and requires rather specific input data (Peaceman, 1983). Therefore, this is not used as often as harmonic mean in industry standard reservoir simulators.

3.6. Harmonic Averaging

The harmonic mean is form of averaging that is appropriate for indicating the average rates of a set of numbers/values. It achieves this by using the reciprocal of the arithmetic mean of these values. It is defined as follows:

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} = \frac{n}{\frac{1}{x_1} + \frac{1}{x_2} + \cdots + \frac{1}{x_n}}$$

Equation 13

The harmonic mean is often used when the series of numbers contain several outliers. It tends to alleviate the influence of larger outliers, but also intensify the influence of smaller ones (Chou, 1963).

This is the most commonly used form of averaging in industry standard reservoir simulators. It accepts all ranges of numbers (besides zero) and is stable in most relevant situations. The drawback of using this method, however, is that it can greatly underestimate the effective permeability, especially in regions where there is a high permeability ratio. As such, the results in heterogeneous reservoirs is often misleading and incorrect.

4. Finite Analytic Method

Although harmonic mean is the most commonly used method for approximating intermodal transmissivity, it very often underestimates effective permeability, especially when there is a strong contrast in permeability present. This is most evident when using the theoretical example of a checkerboard geometry. This consists of alternating low and high permeability, which is a similar structure as can be seen on a chess board. Harmonic averaging yields numerical error that increases substantially as the ratio between the permeabilities grows (Yeo & Zimmerman, 2001). Thus, as the use of harmonic averaging is frequent in most commercial simulators, the results of heterogeneous reservoirs can often be erroneous.

The finite analytic method was developed as an improvement for heterogeneous porous media (Liu & Wang, 2013). It assumes power law behavior of the pressure as the node is approached, (see Figure 4.1). This is supported by both numerical (Yeo & Zimmerman, 2001) and experimental observations (Dawe & Grattoni, 2008).

Note: the pressure equations presented in the article are not entirely correct. They do not abide by the criteria and results in erroneous solutions. As such, a great deal of time was required and was essential to correct the equations.

4.1. Power Law Behavior

Spatial discontinuities in permeability causes large spikes in the pressure field, where the pressure gradient would be discontinuous across the interfaces. Solving for such conditions numerically can be rather difficult, a condition that is common in heterogeneous reservoirs. The power law behavior has been explored recently, as is discussed above, and it was discovered that the pressure and its normal derivative exhibit a behavior similar to the power law toward the grid node.

Power law is a polynomial based function that relates two quantities, where relative changes in one leads to proportional changes in the other. In other words, “one quantity varies as a power of another” (Yaneer, 2017). As an example, comparing the length of a square to its area, if the length is doubled then the area is quadrupled. The general power law relation with respect to pressure is as follows (Liu & Wang, 2013):

$$P - P_0 \propto x^{\alpha}\,, \qquad \frac{\partial P}{\partial L} \propto x^{\beta}$$

Equation 14

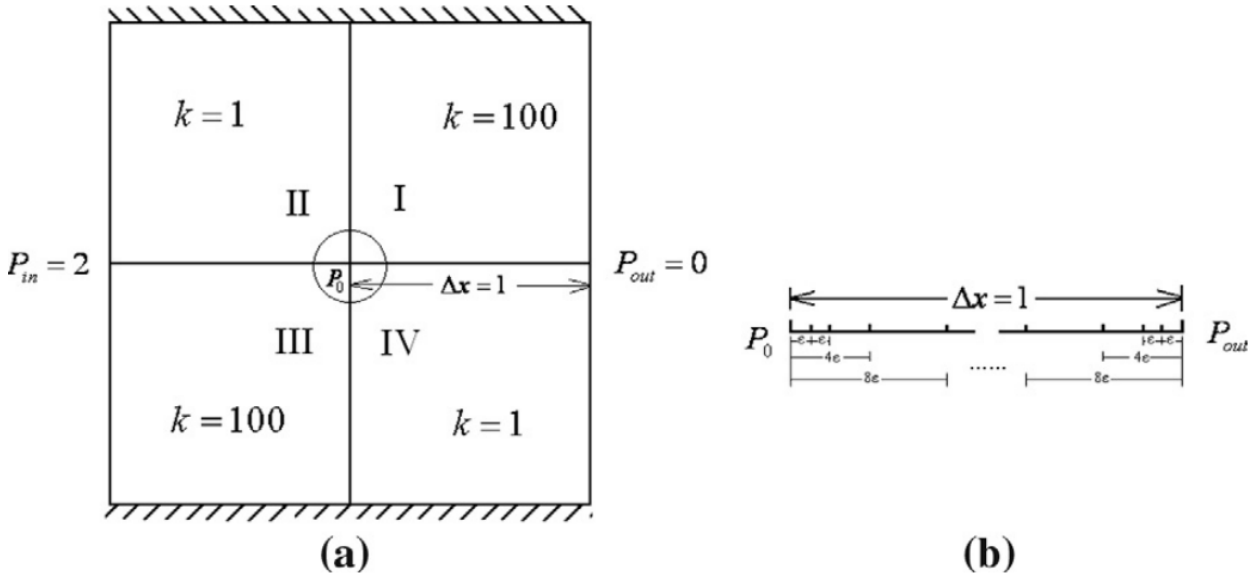

Figure 4.1: A 2x2 grid block showcasing the permeability distribution (a), and an illustration of the discretization of the boundary element (b) (Liu & Wang, 2013)

A test was conducted on a two dimensional 2 by 2 grid, as can be seen in Figure 4.1, where the permeability was set to constant values in each of the four quadrants. To accurately perform the test, several control volumes were used in order to achieve more precise results. The boundary conditions was a simple test case, two parallel sides that are impermeable and two parallel sides with a constant pressure difference. This type of test is often used to test for the accuracy of algorithms since the effective permeability can be determined analytically with an exact value. The test concluded that the power law behavior is in fact present and is prominent around the grid node.

After having identified that the power law behavior is prominent in heterogeneous reservoirs, especially near the grid node, it can be assumed that the pressure and the pressure gradient will exhibit power law behavior. The assumption for the solutions of the pressure equation can be seen in Equation 15, where both the pressure and the pressure gradient display protuberant power law structure in their formulation.

As a result, the assumptions for the finite analytic method, as presented in the article (Liu & Wang, 2013), is as follows for a grid node in $x = 0, y = 0$.

1. Along the positive x-axis:

$$P|_{x>0,y=0} = A_1 x^{1-\alpha_1}, \qquad \left.\frac{\partial P}{\partial y}\right|_{x>0,y\to 0^+} = B_1(1-\beta_1)x^{-\beta_1}$$

2. Along the positive y-axis:

$$P|_{x=0,y>0} = A_2 y^{1-\alpha_2}, \qquad \left.\frac{\partial P}{\partial y}\right|_{x\to 0^-,y>0} = B_2(1-\beta_2)y^{-\beta_2}$$

3. Along the negative x-axis:

$$P|_{x<0,y=0} = A_3(-x)^{1-\alpha_3}, \qquad \left.\frac{\partial P}{\partial y}\right|_{x<0,y\to 0^-} = B_3(1-\beta_3)(-x)^{-\beta_3}$$

4. Along the negative y-axis:

$$P|_{x=0,y<0} = A_4(-y)^{1-\alpha_4}, \qquad \left.\frac{\partial P}{\partial y}\right|_{x\to 0^+,y<0} = B_4(1-\beta_4)(-y)^{-\beta_4}$$

Equation 15

4.2. Analytical Nodal Solution

The analytical solution for the finite analytic method is detailed below. The equations and calculations that follow are in large part replicated and derived again based on the solutions presented by (Liu & Wang, 2013). The article goes through the procedure, but avoided to mention some very important steps, some of which took a significant amount of time and effort to understand and derive.

The analytical solution is set in an infinite domain with four heterogeneous permeabilities connected to the nodal point.

$$\nabla \cdot [(k(x,y)\nabla P)] = 0$$

Equation 16

The solution is derived by applying Equation 16 in an infinite coordinate plane comprised of four quadrants, each with their select permeability. Starting by first assuming the power law behavior, as was determined earlier, the solution of Equation 16 should fulfill the important continuity criteria, namely the pressure continuity and flux continuity. Equation 17 highlights the continuity criteria for both the pressure and the flux between the interfaces at the grid node.

1. Along the positive x-axis:

$$P_1(x,y)|_{x>0,y\to 0^+} = P_4(x,y)|_{x>0,y\to 0^-}$$
$$k_1 \left.\frac{\partial P_1}{\partial y}\right|_{x>0,y\to 0^+} = k_4 \left.\frac{\partial P_4}{\partial y}\right|_{x>0,y\to 0^-}$$

2. Along the positive y-axis:

$$P_2(x,y)|_{x\to 0^-,y>0} = P_1(x,y)|_{x\to 0^+,y>0}$$
$$k_2 \left.\frac{\partial P_2}{\partial x}\right|_{x\to 0^-,y>0} = k_1 \left.\frac{\partial P_1}{\partial x}\right|_{x\to 0^+,y>0}$$

3. Along the negative x-axis:

$$P_3(x,y)|_{x>0,y\to 0^-} = P_2(x,y)|_{x>0,y\to 0^+}$$
$$k_3 \left.\frac{\partial P_3}{\partial y}\right|_{x>0,y\to 0^-} = k_2 \left.\frac{\partial P_2}{\partial y}\right|_{x>0,y\to 0^+}$$

4. Along the negative y-axis:

$$P_4(x,y)|_{x\to 0^+,y>0} = P_3(x,y)|_{x\to 0^-,y>0}$$
$$k_4 \left.\frac{\partial P_4}{\partial x}\right|_{x\to 0^+,y>0} = k_3 \left.\frac{\partial P_3}{\partial x}\right|_{x\to 0^-,y>0}$$

Equation 17

Where $A_i, B_i, \alpha_i, \beta_i$ are unknowns relative to the quadrant that they represent. One can also say the $\alpha_i < 1$ and $\beta_i \geq 0$ due to the fact that the pressure is continuous and the flux is divergent when approaching the origin.

Starting with applying complex function theory (Lang, 1985), assigning $z = x + iy$ to be a complex number, and $f(z) = u(x,y) + iv(x,y)$ to be a complex analytic function. It is known that the Cauchy-Riemann relation can be written as follows:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y}, \qquad \frac{\partial u}{\partial y} = -\frac{\partial v}{\partial x}$$

Equation 18

By using Equation 18 it can be observed that:

$$\frac{\partial^2 u}{\partial x^2} = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right) = \frac{\partial}{\partial x}\left(\frac{\partial v}{\partial y}\right) = \frac{\partial^2 v}{\partial x \partial y} = \frac{\partial}{\partial x}\left(\frac{\partial v}{\partial x}\right) = \frac{\partial}{\partial x}\left(-\frac{\partial u}{\partial y}\right) = -\frac{\partial^2 u}{\partial y^2}$$

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

Equation 19

Relating the same logic for the imaginary component v, the result is as follows:

$$\frac{\partial^2 v}{\partial y^2} = \frac{\partial}{\partial y}\left(\frac{\partial u}{\partial x}\right) = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial y}\right) = \frac{\partial}{\partial x}\left(-\frac{\partial v}{\partial x}\right) = -\frac{\partial^2 v}{\partial x^2}$$

$$\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} = 0$$

Equation 20

Therefore, it can be concluded that both the imaginary part and the real part of any complex analytic function satisfies the Laplace equation. However, it can be useful to express the solution more generally, following the fore mentioned article, (Liu & Wang, 2013). The solution of Laplace's equation in a given quadrant is then written in the form:

$$P_i(x, y) = f_i(x + iy) + g_i(x - iy) = f_i(z) + g_i(\bar{z})$$

Equation 21

Where $\bar{z} = x - iy$ is the complex conjugant of $z = x + iy$, and f_i and g_i are complex analytic functions. It is then also understood that the sum of the two functions should yield no imaginary part. Now, using Equation 15, is can be noted that the solution for the first quadrant can be presented as:

$$f_1(x) + g_1(x) = A_1 x^{1-\alpha_1}$$
$$if'_1(x) - ig'_1(x) = (1 - \beta_1) B_1 x^{-\beta_1}$$

Equation 22

Which leads to:

$$f_1(x) = \frac{1}{2}\left(A_1 x^{1-\alpha_1} - iB_1 x^{-\beta_1} + C\right)$$
$$g_1(x) = \frac{1}{2}\left(A_1 x^{1-\alpha_1} + iB_1 x^{-\beta_1} - C\right)$$

Equation 23

Then, referring back to Equation 21, the pressure at the first quadrant can be written as:

$$P_1(x, y) = \frac{1}{2}\left\{A_1\left[z^{1-\alpha_1} + \bar{z}^{1-\alpha_1}\right] - iB_1\left[z^{1-\beta_1} - \bar{z}^{1-\beta_1}\right]\right\}$$

Equation 24

Using a similar approach, the pressure in all the quadrants are found to be:

$$P_1(x,y) = \frac{1}{2}\left\{A_1[(z)^{1-\alpha_1} + (\bar{z})^{1-\alpha_1}] - iB_1\left[(z)^{1-\beta_1} - (\bar{z})^{1-\beta_1}\right]\right\}$$
$$P_2(x,y) = \frac{1}{2}\left\{A_2[(-iz)^{1-\alpha_2} + (i\bar{z})^{1-\alpha_2}] + iB_2\left[(-iz)^{1-\beta_2} - (i\bar{z})^{1-\beta_2}\right]\right\}$$
$$P_3(x,y) = \frac{1}{2}\left\{A_3[(-z)^{1-\alpha_3} + (-\bar{z})^{1-\alpha_3}] + iB_3\left[(-z)^{1-\beta_3} - (-\bar{z})^{1-\beta_3}\right]\right\}$$
$$P_4(x,y) = \frac{1}{2}\left\{A_4[(iz)^{1-\alpha_4} + (-i\bar{z})^{1-\alpha_4}] - iB_4\left[(iz)^{1-\beta_4} - (-i\bar{z})^{1-\beta_4}\right]\right\}$$

Equation 25

By applying the criteria established in Equation 17, regarding the continuity of pressure and flux, on the general pressure solutions in Equation 25, the A_i, B_i unknowns can be determined by algebraic computations. Taking the $P_1 = P_2$ and the $k_1 \frac{\partial P_1}{\partial y} = k_2 \frac{\partial P_2}{\partial y}$ criteria set as an initial example (along positive y-axis), from Equation 17, and utilize the pressure solutions from Equation 25, the following can be inferred:

$$\frac{1}{2}\left\{A_1[(iy)^{1-\alpha_1} + (-iy)^{1-\alpha_1}] - iB_1\left[(iy)^{1-\beta_1} - (-iy)^{1-\beta_1}\right]\right\} = A_2(y)^{1-\alpha_2}$$
$$\frac{1}{2}\left\{A_1(1-\alpha_1)[(iy)^{-\alpha_1} + (-iy)^{-\alpha_1}] - iB_1(1-\beta_1)\left[(iy)^{-\beta_1} - (-iy)^{-\beta_1}\right]\right\} = \frac{k_2}{k_1}B_2(1-\beta_2)(y)^{-\beta_2}$$

Equation 26

The same can be done for the other pressure equations to achieve comparable equations.

Firstly, since Equation 26 should be valid for all $y > 0$ in the domain, and in order to also have unique solutions, it must be concluded that $\alpha_1 = \beta_1 = \alpha_2 = \beta_2 = \alpha$, and as such, for the general equations for all four quadrants, the exponents can be simplified as $\alpha_i = \beta_i = \alpha$. After some algebraic manipulations, the various equations can be solved as follows:

$$A_1 = A_4 \sin\left(\frac{\pi}{2}\alpha\right) + B_4 \cos\left(\frac{\pi}{2}\alpha\right)$$
$$B_1 = \frac{k_4}{k_1}\left[-A_4 \cos\left(\frac{\pi}{2}\alpha\right) + B_4 \sin\left(\frac{\pi}{2}\alpha\right)\right]$$

$$A_2 = A_1 \sin\left(\frac{\pi}{2}\alpha\right) + B_1 \cos\left(\frac{\pi}{2}\alpha\right)$$
$$B_2 = \frac{k_1}{k_2}\left[A_1 \cos\left(\frac{\pi}{2}\alpha\right) - B_1 \sin\left(\frac{\pi}{2}\alpha\right)\right]$$

$$A_3 = A_2 \sin\left(\frac{\pi}{2}\alpha\right) - B_2 \cos\left(\frac{\pi}{2}\alpha\right)$$
$$B_3 = \frac{k_2}{k_3}\left[A_2 \cos\left(\frac{\pi}{2}\alpha\right) + B_2 \sin\left(\frac{\pi}{2}\alpha\right)\right]$$

$$A_4 = A_3 \sin\left(\frac{\pi}{2}\alpha\right) - B_3 \cos\left(\frac{\pi}{2}\alpha\right)$$
$$B_4 = \frac{k_3}{k_4}\left[-A_3 \cos\left(\frac{\pi}{2}\alpha\right) - B_3 \sin\left(\frac{\pi}{2}\alpha\right)\right]$$

Equation 27

Looking at Equation 27, it can be observed that the individual equations are interconnected in such a manner that is possible to reduce the number of unknowns to only two. This can done using the backward substitution method on the entire cycle from the equations. The step by step expansion is significantly comprehensive and is therefore difficult to include in its entirety here. However, scanned copies of the entire calculation can be found in the appendix.

By performing the substitution method on the entire cycle, Equation 27 can be simplified to a set of two linear equations consisting of only A_1, B_1, α as unknowns, seen in Equation 28.

$$0 = A_1\left\{\sin^4\left(\frac{\pi}{2}\alpha\right) + \left[\frac{k_1k_3}{k_2k_4}\right]\cos^4\left(\frac{\pi}{2}\alpha\right) - \left[\frac{k_1}{k_2} + \frac{k_1}{k_3} + \frac{k_1}{k_4} + \frac{k_2}{k_3} + \frac{k_2}{k_4} + \frac{k_3}{k_4}\right]\frac{1}{4}\sin^2(\pi\alpha) - 1\right\}$$
$$+B_1\left\{\frac{1}{8}\left[1 + \frac{k_1}{k_2} + \frac{k_1}{k_3} + \frac{k_1}{k_4}\right][2\sin(\pi\alpha) - \sin(2\pi\alpha)] - \frac{1}{8}\left[\frac{k_1k_3}{k_2k_4} + \frac{k_2}{k_3} + \frac{k_2}{k_4} + \frac{k_3}{k_4}\right][2\sin(\pi\alpha) + \sin(2\pi\alpha)]\right\}$$

$$0 = A_1\left\{\frac{1}{8}\left[\frac{k_2k_4}{k_1k_3} + \frac{k_3}{k_2} + \frac{k_4}{k_2} + \frac{k_4}{k_3}\right][2\sin(\pi\alpha) + \sin(2\pi\alpha)]\right\} - \frac{1}{8}\left[1 + \frac{k_2}{k_1} + \frac{k_3}{k_1} + \frac{k_4}{k_1}\right][2\sin(\pi\alpha) - \sin(2\pi\alpha)]$$
$$+B_1\left\{\sin^4\left(\frac{\pi}{2}\alpha\right) + \left[\frac{k_2k_4}{k_1k_3}\right]\cos^4\left(\frac{\pi}{2}\alpha\right) - \left[\frac{k_4}{k_1} + \frac{k_4}{k_2} + \frac{k_4}{k_3} + \frac{k_3}{k_1} + \frac{k_3}{k_2} + \frac{k_2}{k_1}\right]\frac{1}{4}\sin^2(\pi\alpha) - 1\right\}$$

Equation 28

To find non-zero solutions for the unknowns, the corresponding determinant of the linear equations must equal zero. This is a well-known result from linear algebra. Thus, after considerable manipulation, the determinant of Equation 28 can be expressed as:

$$\sin^8\left(\frac{\pi}{2}\alpha\right)+\cos^8\left(\frac{\pi}{2}\alpha\right)+4\sin^6\left(\frac{\pi}{2}\alpha\right)\cos^2\left(\frac{\pi}{2}\alpha\right)+4\sin^2\left(\frac{\pi}{2}\alpha\right)\cos^6\left(\frac{\pi}{2}\alpha\right)+6\sin^4\left(\frac{\pi}{2}\alpha\right)\cos^4\left(\frac{\pi}{2}\alpha\right)$$
$$-2\sin^4\left(\frac{\pi}{2}\alpha\right)-\left[\frac{k_1k_3}{k_2k_4}+\frac{k_2k_4}{k_1k_3}\right]\cos^4\left(\frac{\pi}{2}\alpha\right)$$
$$+\left[\frac{k_1}{k_2}+\frac{k_1}{k_3}+\frac{k_1}{k_4}+\frac{k_2}{k_3}+\frac{k_2}{k_4}+\frac{k_3}{k_4}\right]\sin^2\left(\frac{\pi}{2}\alpha\right)\cos^2\left(\frac{\pi}{2}\alpha\right)+1=0$$

Equation 29

It is noted that:

$$\sin^8\left(\frac{\pi}{2}\alpha\right)+\cos^8\left(\frac{\pi}{2}\alpha\right)+4\sin^6\left(\frac{\pi}{2}\alpha\right)\cos^2\left(\frac{\pi}{2}\alpha\right)+4\sin^2\left(\frac{\pi}{2}\alpha\right)\cos^6\left(\frac{\pi}{2}\alpha\right)+6\sin^4\left(\frac{\pi}{2}\alpha\right)\cos^4\left(\frac{\pi}{2}\alpha\right)=1$$

And thus:

$$2-2\sin^4\left(\frac{\pi}{2}\alpha\right)-\left[\frac{k_1k_3}{k_2k_4}+\frac{k_2k_4}{k_1k_3}\right]\cos^4\left(\frac{\pi}{2}\alpha\right)+\left[\frac{k_1}{k_2}+\frac{k_1}{k_3}+\frac{k_1}{k_4}+\frac{k_2}{k_3}+\frac{k_2}{k_4}+\frac{k_3}{k_4}\right]\sin^2\left(\frac{\pi}{2}\alpha\right)\cos^2\left(\frac{\pi}{2}\alpha\right)=0$$

Equation 30

Finally, the expression for α can be given as:

$$\alpha=\frac{\pm\cos^{-1}\left(6-\left[\frac{k_1k_3}{k_2k_4}+\frac{k_2k_4}{k_1k_3}\right]+\left[\frac{k_1}{k_2}+\frac{k_1}{k_3}+\frac{k_1}{k_4}+\frac{k_2}{k_3}+\frac{k_2}{k_4}+\frac{k_3}{k_4}\right]\right)}{\pi}$$

Equation 31

In fact, this can further be simplified to:

$$\alpha=\left|\frac{2}{\pi}\tan^{-1}\left[\frac{k_1k_3-k_2k_4}{\sqrt{(k_1+k_2+k_3+k_4)(k_1k_2k_3+k_1k_3k_4+k_1k_2k_4+k_2k_3k_4)}}\right]\right|$$

Equation 32

Which is the equation that is represented in the article (Liu & Wang, 2013).

Note that if $k_1k_3=k_2k_4$, then α would be zero, and there would be no singular behavior from the pressure gradient. This represents a situation with much less severe heterogeneity, and as such can be treated by simply using the harmonic averaging scheme.

The pressure equations can then be simplified even further by assigning a constant for the value of $\frac{B_1}{A_1}$. Employing the solution for α, coupled with one of the expressions from Equation 28, then $\frac{B_1}{A_1}$ can be expressed as a constant, as follows:

$$C = \frac{B_1}{A_1} = sgn(k_1k_3 - k_2k_4)\sqrt{\frac{k_4^2(k_1 + k_2 + k_3 + k_4)}{(k_1k_2k_3 + k_1k_3k_4 + k_1k_2k_4 + k_2k_3k_4)}}$$

Equation 33

After having an expression for both α and C, the pressure equations can finally be expressed in a simplified form where the only unknown is A_1. First, using Equation 27 for the substitution to express all the equations with only A_1, B_1, α as unknowns. Then via Equation 32 and Equation 33, α and B_1 can be eliminated. Note that A_1 can be arbitrary.

Polar coordinates, which was in fact also used in the former manipulations, are a form of coordinate system in mathematics, where each point is represented by a distance and angle from a reference plane (Brown, Gleason, & Brown, 1992). In comparison to a Cartesian coordinates, which is the standard x, y coordinate system, polar coordinates use trigonometric functions to convert Cartesian coordinates, where x and y are related to the length/radius r and angle θ.

$$x = r\cos\theta\ , \qquad y = r\sin\theta$$
$$r = \sqrt{x^2 + y^2}\ , \qquad \theta = \tan^{-1}\left(\frac{y}{x}\right)$$

Equation 34

The same conversion, seen in Equation 34, can be done to complex numbers, where $x = r\cos\theta$ and $iy = ri\sin\theta$. Thus, $z = x + iy = r[\cos\theta + i\sin\theta]$ in polar coordinates. Moreover, Euler's formula can be applied to express the complex numbers in an exponential form, where $r[\cos\theta + i\sin\theta] = re^{i\theta}$ is the expression for any given complex number (Lang, 1985).

The following are the pressure equations expressed with polar coordinates by using the method described above. These differ slightly from what was suggested in (Liu & Wang, 2013). This is due to the fact that the equations in the article do not abide all the criteria in Equation 17 that were set by the authors, which in turn results in incorrect solutions. This is probably due to some strange misprints in the article. However, by using the following equations that were derived from the same methodology suggested in the article, both the criteria are upheld and the results gave the correct solution. The numerical solutions are directly comparable with that of the article, which will be briefly shown and discussed in the Results section. Correcting these misprints caused a considerable amount of time for both troubleshooting the issues, and once the problem was pin pointed, a lot of time was required to actually derive the correct corresponding equations.

$$
\begin{aligned}
P_1 &= A_1 r^{1-\alpha}\{\cos[\theta(1-\alpha)] + C\sin[\theta(1-\alpha)]\} = A\lambda_1 \\
P_2 &= A_1 r^{1-\alpha}\left\{\left[\sin\left(\frac{\pi}{2}\alpha\right) + C\cos\left(\frac{\pi}{2}\alpha\right)\right]\cos\left[\left(\theta - \frac{\pi}{2}\right)(1-\alpha)\right]\right. \\
&\qquad \left. - \left[\frac{k_1}{k_2}\right]\left[\cos\left(\frac{\pi}{2}\alpha\right) - C\sin\left(\frac{\pi}{2}\alpha\right)\right]\sin\left[\left(\theta - \frac{\pi}{2}\right)(1-\alpha)\right]\right\} = A\lambda_2 \\
P_3 &= A_1 r^{1-\alpha}\left\{\left[\sin^2\left(\frac{\pi}{2}\alpha\right) - \left[\frac{k_1}{k_2}\right]\cos^2\left(\frac{\pi}{2}\alpha\right) + \left[1 + \frac{k_1}{k_2}\right]C\sin\left(\frac{\pi}{2}\alpha\right)\cos\left(\frac{\pi}{2}\alpha\right)\right]\cos[(\theta-\pi)(1-\alpha)]\right. \\
&\qquad - \left[\left[\frac{k_1}{k_3} + \frac{k_2}{k_3}\right]\sin\left(\frac{\pi}{2}\alpha\right)\cos\left(\frac{\pi}{2}\alpha\right) - C\sin\left(\frac{\pi}{2}\alpha\right) + \left[\frac{k_2}{k_3}\right]C\cos^2\left(\frac{\pi}{2}\alpha\right)\right. \\
&\qquad \left.\left. - \left[\frac{k_1}{k_2}\right]C\sin^2\left(\frac{\pi}{2}\alpha\right)\right]\sin[(\theta-\pi)(1-\alpha)]\right\} = A\lambda_3 \\
P_4 &= A_1 r^{1-\alpha}\left\{\left[\sin^3\left(\frac{\pi}{2}\alpha\right) - \left[\frac{k_2}{k_3}\right]C\cos^3\left(\frac{\pi}{2}\alpha\right) + \left[1 + \frac{k_1}{k_2} + \frac{k_1}{k_3}\right]C\sin^2\left(\frac{\pi}{2}\alpha\right)\cos\left(\frac{\pi}{2}\alpha\right)\right.\right. \\
&\qquad \left. - \left[\frac{k_1}{k_2} + \frac{k_1}{k_3} + \frac{k_2}{k_3}\right]C\sin\left(\frac{\pi}{2}\alpha\right)\cos^2\left(\frac{\pi}{2}\alpha\right)\right]\cos\left[\left(\theta + \frac{\pi}{2}\right)(1-\alpha)\right] \\
&\qquad + \left[\left[\frac{k_1}{k_4}\right]C\sin^3\left(\frac{\pi}{2}\alpha\right) + \left[\frac{k_1 k_3}{k_2 k_4}\right]\cos^3\left(\frac{\pi}{2}\alpha\right) - \left[\frac{k_1}{k_4} + \frac{k_2}{k_4} + \frac{k_3}{k_4}\right]\sin^2\left(\frac{\pi}{2}\alpha\right)\cos\left(\frac{\pi}{2}\alpha\right)\right. \\
&\qquad \left.\left. - \left[\frac{k_1 k_3}{k_2 k_4} + \frac{k_2}{k_4} + \frac{k_3}{k_4}\right]C\sin\left(\frac{\pi}{2}\alpha\right)\cos^2\left(\frac{\pi}{2}\alpha\right)\right]\sin\left[\left(\theta + \frac{\pi}{2}\right)(1-\alpha)\right]\right\} = A\lambda_4
\end{aligned}
$$

Equation 35

Where λ_i notation is introduced and A_1 is set to A for simplicity.

With this, the analytical nodal solution is derived and expressed. Now, the numerical fragment, which was discussed at the end of section 3.3, will be tackled. Below is the necessary approach to achieve the correct numerical expressions for the fluxes needed in the implementation of the finite analytic method.

By assuming a continuity in the flux, which is denoted as Q, the flux solution at quadrant 1 and 3 need to be calculated and included in the final solution. This is required to fulfill both criteria set above, as well as the continuity functions, as seen in Figure 3.3.

Note: in following equations; $\Delta x = \frac{1}{2}\Delta x$ and $\Delta y = \frac{1}{2}\Delta y$. This was done to make the transition into the program easier as the foundation of the code that was used in this study applied this notation.

$$Q_{x_1} = -\int_0^{\Delta y} \left(k_1 \frac{\partial P_1}{\partial x}\right) \partial y$$

$$\frac{\partial P_1}{\partial x} = A(1-\alpha) r^{-\alpha} \frac{\partial r}{\partial x} \{\cos[\theta(1-\alpha)] + C \sin[\theta(1-\alpha)]\}$$
$$+A(1-\alpha) r^{1-\alpha} \frac{\partial \theta}{\partial x} \{-\sin[\theta(1-\alpha)] + C \cos[\theta(1-\alpha)]\}$$

$$r = \sqrt{x^2 + y^2} \rightarrow \frac{\partial r}{\partial x} = \frac{x}{r}, \theta = \tan^{-1}\left(\frac{y}{x}\right) \rightarrow \frac{\partial \theta}{\partial x} = -\frac{y}{r^2}$$

$$as\ x \rightarrow 0, \theta \rightarrow \frac{\pi}{2}, calculations\ are\ done\ along\ positive\ y-axis$$

$$\frac{\partial P_1}{\partial x} = A(1-\alpha) y^{1-\alpha} \frac{-y}{r^2} \left\{-\sin\left[\frac{\pi}{2}(1-\alpha)\right] + C \cos\left[\frac{\pi}{2}(1-\alpha)\right]\right\}$$
$$= A(1-\alpha) y^{-\alpha} \left\{\sin\left(\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}\alpha\right) - \sin\left(\frac{\pi}{2}\alpha\right)\cos\left(\frac{\pi}{2}\right) - C\cos\left(\frac{\pi}{2}\right)\cos\left(\frac{\pi}{2}\alpha\right) - C\sin\left(\frac{\pi}{2}\right)\sin\left(\frac{\pi}{2}\alpha\right)\right\}$$
$$= A(1-\alpha) y^{-\alpha} \left\{\cos\left(\frac{\pi}{2}\alpha\right) - C\sin\left(\frac{\pi}{2}\alpha\right)\right\}$$

$$Q_{x_1} = -\int_0^{\Delta y} \left(k_1 \frac{\partial P_1}{\partial x}\right) \partial y = -k_1 A y^{1-\alpha} \left\{\cos\left(\frac{\pi}{2}\alpha\right) - C\sin\left(\frac{\pi}{2}\alpha\right)\right\} = \lambda_{x_1} A$$

Equation 36

Similarly for $Q_{y_1}, Q_{x_3}, Q_{y_3}$:

$$Q_{x_1} = -\int_0^{\Delta y} \left(k_1 \frac{\partial P_1}{\partial x}\right) \partial y = -k_1 A y^{1-\alpha} \left\{\cos\left(\frac{\pi}{2}\alpha\right) - C\sin\left(\frac{\pi}{2}\alpha\right)\right\} = \lambda_{x_1} A$$

$$Q_{y_1} = -\int_0^{\Delta x} \left(k_1 \frac{\partial P_1}{\partial y}\right) \partial x = -k_1 A x^{1-\alpha} C = \lambda_{y_1} A$$

$$Q_{x_3} = -\int_0^{\Delta y} \left(k_3 \frac{\partial P_3}{\partial x}\right) \partial y$$
$$= k_3 A y^{1-\alpha} \left\{\left[\sin^2\left(\frac{\pi}{2}\alpha\right) - \left[\frac{k_1}{k_2}\right]\cos^2\left(\frac{\pi}{2}\alpha\right) + \left[1 + \frac{k_1}{k_2}\right]\sin\left(\frac{\pi}{2}\alpha\right)\cos\left(\frac{\pi}{2}\alpha\right)\right]\cos\left(\frac{\pi}{2}\alpha\right)\right.$$
$$\left. + \left[-\left[\frac{k_1}{k_3}\right]\sin^2\left(\frac{\pi}{2}\alpha\right) + \left[\frac{k_2}{k_3}\right] C\cos^2\left(\frac{\pi}{2}\alpha\right) + \left[\frac{k_1}{k_3} + \frac{k_2}{k_3}\right]\sin\left(\frac{\pi}{2}\alpha\right)\cos\left(\frac{\pi}{2}\alpha\right)\right]\sin\left(\frac{\pi}{2}\alpha\right)\right\}$$
$$= \lambda_{x_3} A$$

$$Q_{y_3} = -\int_0^{\Delta x} \left(k_3 \frac{\partial P_3}{\partial y}\right) \partial x$$
$$= -k_3 A x^{1-\alpha} \left\{-\left[\frac{k_1}{k_3}\right] C\sin^2\left(\frac{\pi}{2}\alpha\right) + \left[\frac{k_2}{k_3}\right] C\cos^2\left(\frac{\pi}{2}\alpha\right)\right.$$
$$\left. + \left[\frac{k_1}{k_3} + \frac{k_2}{k_3}\right]\sin\left(\frac{\pi}{2}\alpha\right)\cos\left(\frac{\pi}{2}\alpha\right)\right\} = \lambda_{y_3} A$$

Equation 37

Define the inter nodal transmissivity γ as follows:

$$\gamma = \frac{Q}{P_i - P_j} = \frac{A\lambda_{x,y}}{A(\lambda_i - \lambda_j)} = \frac{\lambda_{x,y}}{\lambda_i - \lambda_j}$$

Equation 38

And thus, the final solution for the inter nodal transmissivity can be expressed as such:

$$\gamma_{x_1} = \frac{\lambda_{x_1}}{\lambda_2 - \lambda_1}, \qquad \gamma_{y_1} = \frac{\lambda_{y_1}}{\lambda_4 - \lambda_1}$$
$$\gamma_{x_2} = \frac{\lambda_{x_1}}{\lambda_2 - \lambda_1}, \qquad \gamma_{y_2} = \frac{\lambda_{y_3}}{\lambda_3 - \lambda_2}$$
$$\gamma_{x_3} = \frac{\lambda_{x_3}}{\lambda_3 - \lambda_4}, \qquad \gamma_{y_3} = \frac{\lambda_{y_3}}{\lambda_3 - \lambda_2}$$
$$\gamma_{x_4} = \frac{\lambda_{x_3}}{\lambda_3 - \lambda_4}, \qquad \gamma_{y_4} = \frac{\lambda_{y_1}}{\lambda_4 - \lambda_1}$$

Equation 39

Equation 8 can be now be expressed as follows (see also Figure 3.3):

$$P_1\{(\gamma_{x_1} + \gamma_{x_2}) + (\gamma_{y_1} + \gamma_{y_4})\} + P_2\{(\gamma_{x_1} + \gamma_{x_2}) + (\gamma_{y_2} + \gamma_{y_3})\} + P_3\{(\gamma_{x_3} + \gamma_{x_4}) + (\gamma_{y_2} + \gamma_{y_3})\} + P_4\{(\gamma_{x_3} + \gamma_{x_4}) + (\gamma_{y_1} + \gamma_{y_4})\} = 0$$

Equation 40

Where $k_1 k_3 = k_2 k_4$, a traditional harmonic scheme is used. This is due to the fact that the flux relations are reduced to it when the numerical scheme's sub cell are not directly joined to the grid node. Same goes for the inter nodal transmissivity, when $k_1 k_3 = k_2 k_4$, $\gamma_{x_i}$and γ_{y_i} from the finite analytic method are reduced to the results from the traditional harmonic average (Liu & Wang, 2013).This shows that the power law behavior is only dominant around the grid node where the joined cells have different permeabilities.

At the boundary, two kinds of boundary conditions are typically used. It is either assumed that the pressure is constant at the boundary (Dirchlet). Otherwise, it is assumed the boundary is closed, which means that there is no flux (Neumann). Regular harmonic averaging scheme can be used at the boundary. The practical implementation and reasons for its use are detailed in the article (Liu & Wang, 2013). The situation at a boundary is illustrated in Figure 4.2.

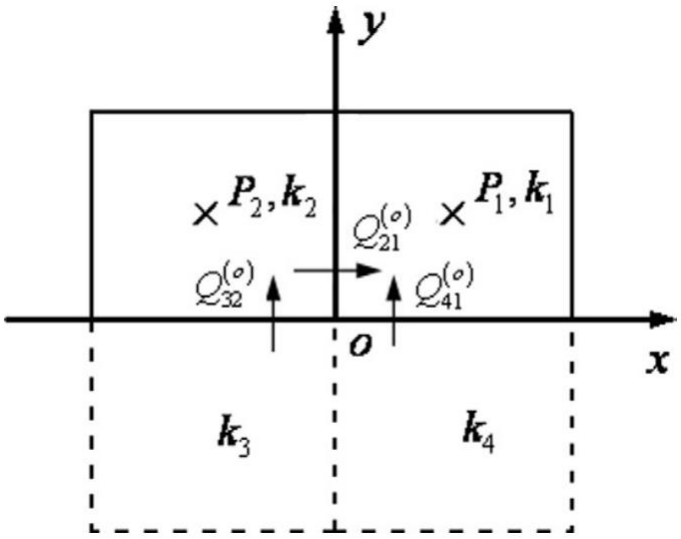

Figure 4.2: An illustration of the fluxes on the surface of the boundary cells (Liu & Wang, 2013)

5. Finite Analytic Method Extension: Anisotropic Solution

This section will examine the new method that was developed during this study. The objective of this method is to take the concepts of the previously described method, and improve on one of its limitation. This method is tries to use the core concepts of the isotropic finite analytic method as a bases, and then modify it to include the ability to solve anisotropic problems.

5.1. Anisotropy

According to Webster's Dictionary, the word isotropic means "exhibiting properties (as velocity of light transmission) with the same values when measured along axis in all directions". Anisotropic however, is when the values are difference in the different directions (Lake, 1988). Petroleum reservoirs are often described as being both anisotropic and heterogeneous. This is especially true for regions such as fractured beds, where the permeability should be treated as a full tensor (Liu & Wang, 2016).

Researchers have developed several numerical schemes where the permeability is a full tensor with a discontinuous distribution. The manner in which this study approached anisotropy is by assigning every permeability within a cell as a 2×2 matrix. Additionally, the coordinate system is attributed an angle to the deviation from the standard space. These are then combined to develop a new coordinate system that can easily transform back and forth to the standard x, y space (Nelson, 2001).

5.2. Finite Analytic Method with Anisotropy

The permeabilities are expressed in matrix form, as can be seen in Figure 5.1 where the permeabilities are shown with the tensor components.

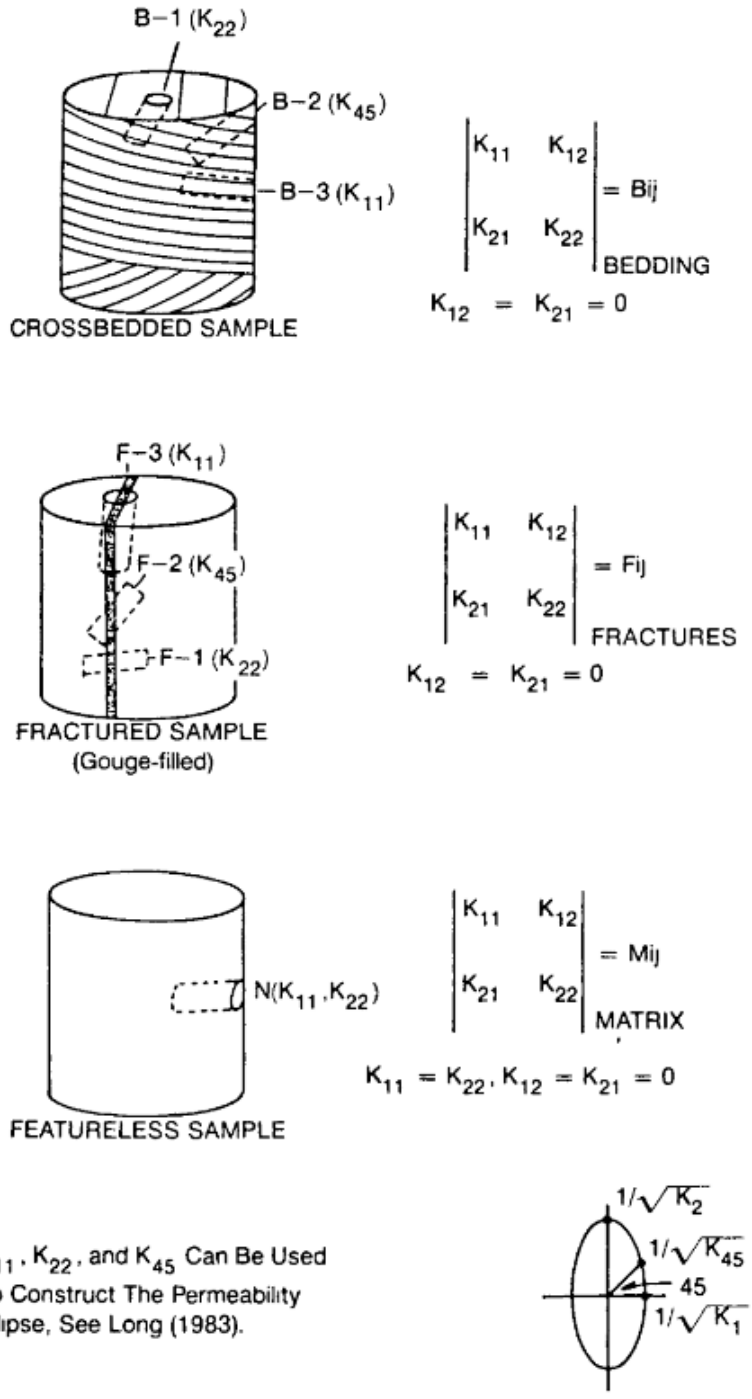

Figure 5.1: Permeability tensor components shown alongside their respective permeability plug and the features they represent (Nelson, 2001)

Starting with the full symmetric tensor representation of the permeability

$$k_i = \begin{pmatrix} k_{i_{11}} & k_{i_{12}} \\ k_{i_{21}} & k_{i_{22}} \end{pmatrix}$$

Equation 41

Where i refers to the quadrants, and $k_{i_{12}}$ and $k_{i_{21}}$ can be set equal to each other due to orientation, as is seen in Figure 5.1.

The eigenvalues of the permeability tensor k_i can then be calculated from the following expression

$$det(\lambda I - k_i) = \begin{vmatrix} \lambda - k_{i_{11}} & k_{i_{12}} \\ k_{i_{12}} & \lambda - k_{i_{22}} \end{vmatrix} = (\lambda - k_{i_{11}})(\lambda - k_{i_{22}}) - {k_{i_{12}}}^2 = 0$$
$$\lambda^2 - (k_{i_{11}} + k_{i_{22}})\lambda + k_{i_{11}}k_{i_{22}} - {k_{i_{12}}}^2 = 0$$

$$\lambda = \frac{1}{2}\left[(k_{i_{11}} + k_{i_{22}}) \pm \sqrt{(k_{i_{11}} + k_{i_{22}})^2 - 4(k_{i_{11}}k_{i_{22}} - {k_{i_{12}}}^2)}\right]$$
$$\lambda = \frac{1}{2}\left[(k_{i_{11}} + k_{i_{22}}) \pm \sqrt{{k_{i_{11}}}^2 + {k_{i_{22}}}^2 - 2k_{i_{11}}k_{i_{22}} + 4{k_{i_{12}}}^2}\right]$$

Equation 42

For a given eigenvector, the following applies:

$$\begin{pmatrix} \lambda - k_{i_{11}} & k_{i_{12}} \\ k_{i_{12}} & \lambda - k_{i_{22}} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$x(\lambda - k_{i_{11}}) - yk_{i_{12}} = 0$$
$$y(\lambda - k_{i_{22}}) - xk_{i_{12}} = 0$$

Equation 43

Therefore, using $(\lambda - k_{i_{11}})(\lambda - k_{i_{22}}) = {k_{i_{12}}}^2$ from Equation 42, the two eigen vectors can then be shown to be:

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \lambda_1 - k_{i_{22}} \\ k_{i_{12}} \end{pmatrix} \& \begin{pmatrix} \lambda_2 - k_{i_{22}} \\ k_{i_{12}} \end{pmatrix}$$

Equation 44

This is well known from linear algebra (Anton, 2010).

A similarity transformation can now be performed on the matrix k_i to obtain a diagonal matrix $\tilde{k}_i$, where $\tilde{k}_i = Q^{-1}k_iQ$, Q^τ and Q is a matrix consisting of the eigenvectors for k_i.

$$\tilde{k}_i = \begin{pmatrix} \tilde{k}_{i_1} & 0 \\ 0 & \tilde{k}_{i_2} \end{pmatrix}$$

Equation 45

Furthermore, it is well known that this similarity transformation corresponds to a rotation of the original standard Cartesian coordinate system to a coordinate system where the new axis are aligned with the eigenvectors of k_i. The rotation, which is denoted as $\tilde{\theta}_i$, is given by the expression:

$$\tilde{\theta}_i = \cos^{-1}\frac{\vec{v}\cdot\vec{y}}{\|\vec{v}\|}$$
$$where\ \vec{y} = \begin{pmatrix}0\\1\end{pmatrix},\ and\ \vec{v}\begin{pmatrix}\lambda_1 - k_{i_{22}}\\ k_{i_{12}}\end{pmatrix} \rightarrow the\ first\ eigenvector\ of\ k_i$$

Equation 46

Returning to the pressure equation with the full tensor permeability, it can now be written as:

$$\nabla\cdot[(k_i(x,y)\nabla P_i)] = k_{i_{11}}\frac{\partial^2 P_i}{\partial x^2} + 2k_{i_{12}}\frac{\partial^2 P_i}{\partial x\partial y} + k_{i_{22}}\frac{\partial^2 P_i}{\partial y^2} = 0$$

Equation 47

Furthermore, it is also noted that:

$$\nabla\cdot[(k_i(x,y)\nabla P_i)]\cdot\vec{n} = \left[\begin{pmatrix}k_{i_{11}} & k_{i_{12}}\\ k_{i_{12}} & k_{i_{22}}\end{pmatrix}\begin{pmatrix}\frac{\partial P_i}{\partial x}\\ \frac{\partial P_i}{\partial y}\end{pmatrix}\right]\begin{pmatrix}n_x\\ n_y\end{pmatrix} = \begin{bmatrix}\frac{\partial P_i}{\partial x}k_{i_{11}} & \frac{\partial P_i}{\partial y}k_{i_{12}}\\ \frac{\partial P_i}{\partial x}k_{i_{12}} & \frac{\partial P_i}{\partial y}k_{i_{22}}\end{bmatrix}\begin{pmatrix}n_x\\ n_y\end{pmatrix}$$
$$= \begin{cases}\frac{\partial P_i}{\partial x}k_{i_{11}} + \frac{\partial P_i}{\partial y}k_{i_{12}} & \rightarrow x-axis\\ \frac{\partial P_i}{\partial x}k_{i_{12}} + \frac{\partial P_i}{\partial y}k_{i_{22}} & \rightarrow y-axis\end{cases}$$

Equation 48

$\tilde{k}_i = Q^{-1}k_iQ$ shows how with a repositioned coordinate system, one can eliminate the off diagonal elements. In the rotated coordinate system, denoted by $\tilde{x},\tilde{y}$, Equation 47 can now be written as:

$$\tilde{k}_{i_1}\frac{\partial^2\tilde{P}_i}{\partial\tilde{x}^2} + \tilde{k}_{i_2}\frac{\partial^2\tilde{P}_2}{\partial\tilde{y}^2} = 0$$
$$\begin{pmatrix}\tilde{x}\\ \tilde{y}\end{pmatrix} = \begin{pmatrix}\cos\tilde{\theta}_i & \sin\tilde{\theta}_i\\ -\sin\tilde{\theta}_i & \cos\tilde{\theta}_i\end{pmatrix}\begin{pmatrix}x\\ y\end{pmatrix} = \begin{pmatrix}x\cos\tilde{\theta}_i & y\sin\tilde{\theta}_i\\ -x\sin\tilde{\theta}_i & y\cos\tilde{\theta}_i\end{pmatrix}$$

Equation 49

Such that there is a rotation $\tilde{\theta}_i$ between the x,y coordinate system and the $\tilde{x},\tilde{y}$ coordinate system.

Moreover, an additional transformation is introduced (stretching transformation), such that:

$$\tilde{x} = \hat{x}\sqrt{\tilde{k}_{i_1}} = x\cos\tilde{\theta}_i + y\sin\tilde{\theta}_i \;\rightarrow\; \frac{\partial\hat{x}}{\partial x} = \frac{\cos\tilde{\theta}_i}{\sqrt{\tilde{k}_{i_1}}}$$
$$\tilde{y} = \hat{y}\sqrt{\tilde{k}_{i_2}} = y\cos\tilde{\theta}_i - x\sin\tilde{\theta}_i \;\rightarrow\; \frac{\partial\hat{y}}{\partial y} = \frac{\cos\tilde{\theta}_i}{\sqrt{\tilde{k}_{i_2}}}$$

Equation 50

In the new $\hat{x},\hat{y}$ coordinate system, the pressure equation can finally be written as:

$$\frac{\partial^2 P_i}{\partial\hat{x}^2} + \frac{\partial^2 P_2}{\partial\hat{y}^2} = 0$$

Equation 51

In other words, the Laplace equation for the pressure can is obtained in the $\hat{x},\hat{y}$ coordinate system.

Introducing the variable τ_i, which represents the rotation of the axis and is described in complex form as $\tau_i = e^{i\tilde{\theta}}$

$$z = x + iy\,, \qquad \bar{z} = x - iy$$
$$\tau z = \hat{z}\,, \qquad \overline{\tau z} = \bar{\hat{z}}$$
$$\hat{z} = \hat{x} + i\hat{y}\,, \qquad \bar{\hat{z}} = \hat{x} - i\hat{y}$$
$$\hat{z} = \frac{x\cos\tilde{\theta}_i + y\sin\tilde{\theta}_i}{\sqrt{\tilde{k}_{i_1}}} + i\frac{y\cos\tilde{\theta}_i - x\sin\tilde{\theta}_i}{\sqrt{\tilde{k}_{i_2}}}$$
$$\bar{\hat{z}} = \frac{x\cos\tilde{\theta}_i + y\sin\tilde{\theta}_i}{\sqrt{\tilde{k}_{i_1}}} - i\frac{y\cos\tilde{\theta}_i - x\sin\tilde{\theta}_i}{\sqrt{\tilde{k}_{i_2}}}$$

Equation 52

By following to a certain degree the same methodology as with the isotropic case, the analytical nodal solution for the anisotropic case can be determined. However, the computations in the following will naturally be much more involved. The solutions for the individual pressure equation will be using the coordinate system described above $(\hat{x},\hat{y})$.

Assuming the same power law behavior as described in Equation 17, (but in the $\hat{x},\hat{y}$ coordinate system) the pressure equations can then be expressed as follows:

$$P_i = f_i(\tau z) + g_i(\overline{\tau z})\,, \qquad or$$
$$P_i(\hat{x},\hat{y}) = f_i(\hat{z}) + g_i\left(\bar{\hat{z}}\right)$$

Equation 53

As before, f_i and g_i are complex analytic functions. Thus, for the first quadrant, the following is valid:

$$f_1(\hat{x}) + g_1(\hat{x}) = A_1\hat{x}^{1-\alpha_1}$$
$$if'_1(\hat{x}) - ig'_1(\hat{x}) = (1-\beta_1)B_1\hat{x}^{-\beta_1}$$

Equation 54

The details of the solution is presented in the following:

$$f'_1(\hat{x}) + g'_1(\hat{x}) = (1-\alpha_1)A_1\hat{x}^{-\alpha_1}, \qquad |\times i \; and \; sum$$
$$2if'_1(\hat{x}) = (1-\alpha_1)iA_1\hat{x}^{-\alpha_1} + (1-\beta_1)B_1\hat{x}^{-\beta_1}, \qquad \left|\times \frac{1}{2i}\right.$$
$$f'_1(\hat{x}) = \frac{1}{2}\{(1-\alpha_1)A_1\hat{x}^{-\alpha_1} - (1-\beta_1)iB_1\hat{x}^{-\beta_1}\}$$
$$thus: f_1(\hat{x}) = \frac{1}{2}\left(A_1\hat{x}^{1-\alpha_1} - iB_1\hat{x}^{-\beta_1} + C\right)$$
$$g_1(\hat{x}) = \frac{1}{2}\left(A_1\hat{x}^{1-\alpha_1} + iB_1\hat{x}^{-\beta_1} - C\right)$$
$$P_1(\hat{x},\hat{y}) = \frac{1}{2}\{A_1[\hat{z}^{1-\alpha_1} + \bar{\hat{z}}^{1-\alpha_1}] - iB_1[\hat{z}^{1-\beta_1} - \bar{\hat{z}}^{1-\beta_1}]\}$$

Equation 55

Performing the same procedure on all four quadrants, the pressure equations now read:

$$P_1(\hat{x},\hat{y}) = \frac{1}{2}\left\{A_1\left[(\hat{z})^{1-\alpha_1} + \left(\bar{\hat{z}}\right)^{1-\alpha_1}\right] - iB_1\left[(\hat{z})^{1-\beta_1} - \left(\bar{\hat{z}}\right)^{1-\beta_1}\right]\right\}$$
$$P_2(\hat{x},\hat{y}) = \frac{1}{2}\left\{A_2\left[(-i\hat{z})^{1-\alpha_2} + \left(i\bar{\hat{z}}\right)^{1-\alpha_2}\right] + iB_2\left[(-i\hat{z})^{1-\beta_2} - \left(i\bar{\hat{z}}\right)^{1-\beta_2}\right]\right\}$$
$$P_3(\hat{x},\hat{y}) = \frac{1}{2}\left\{A_3\left[(-\hat{z})^{1-\alpha_3} + \left(-\bar{\hat{z}}\right)^{1-\alpha_3}\right] + iB_3\left[(-\hat{z})^{1-\beta_3} - \left(-\bar{\hat{z}}\right)^{1-\beta_3}\right]\right\}$$
$$P_4(\hat{x},\hat{y}) = \frac{1}{2}\left\{A_4\left[(i\hat{z})^{1-\alpha_4} + \left(-i\bar{\hat{z}}\right)^{1-\alpha_4}\right] - iB_4\left[(i\hat{z})^{1-\beta_4} - \left(-i\bar{\hat{z}}\right)^{1-\beta_4}\right]\right\}$$

Equation 56

Their respective derivatives are then deduced to be:

$$\frac{\partial P_1}{\partial \hat{x}} = \frac{1}{2}\left\{(1-\alpha_1)A_1\left[(\hat{z})^{-\alpha_1} + \left(\bar{\hat{z}}\right)^{-\alpha_1}\right] - (1-\beta_1)iB_1\left[(\hat{z})^{-\beta_1} - \left(\bar{\hat{z}}\right)^{-\beta_1}\right]\right\}$$
$$\frac{\partial P_1}{\partial \hat{y}} = \frac{1}{2}i\left\{(1-\alpha_1)A_1\left[(\hat{z})^{-\alpha_1} - \left(\bar{\hat{z}}\right)^{-\alpha_1}\right] - (1-\beta_1)iB_1\left[(\hat{z})^{-\beta_1} + \left(\bar{\hat{z}}\right)^{-\beta_1}\right]\right\}$$

$$\frac{\partial P_2}{\partial \hat{x}} = \frac{1}{2}\left\{(1-\alpha_2)A_2\left[(-i\hat{z})^{-\alpha_2} + \left(i\bar{\hat{z}}\right)^{-\alpha_2}\right] + (1-\beta_2)iB_2\left[(-i\hat{z})^{-\beta_2} - \left(i\bar{\hat{z}}\right)^{-\beta_2}\right]\right\}$$
$$\frac{\partial P_2}{\partial \hat{y}} = \frac{1}{2}i\left\{(1-\alpha_2)A_2\left[(-i\hat{z})^{-\alpha_2} - \left(i\bar{\hat{z}}\right)^{-\alpha_2}\right] + (1-\beta_2)iB_2\left[(-i\hat{z})^{-\beta_2} + \left(i\bar{\hat{z}}\right)^{-\beta_2}\right]\right\}$$

$$\frac{\partial P_3}{\partial \hat{x}} = \frac{1}{2}\left\{(1-\alpha_3)A_3\left[(-\hat{z})^{-\alpha_3} + \left(-\bar{\hat{z}}\right)^{-\alpha_3}\right] + (1-\beta_3)iB_3\left[(-\hat{z})^{-\beta_3} - \left(-\bar{\hat{z}}\right)^{-\beta_3}\right]\right\}$$
$$\frac{\partial P_3}{\partial \hat{y}} = \frac{1}{2}i\left\{(1-\alpha_3)A_3\left[(-\hat{z})^{-\alpha_3} - \left(-\bar{\hat{z}}\right)^{-\alpha_3}\right] + (1-\beta_3)iB_3\left[(-\hat{z})^{-\beta_3} + \left(-\bar{\hat{z}}\right)^{-\beta_3}\right]\right\}$$

$$\frac{\partial P_4}{\partial \hat{x}} = \frac{1}{2}\left\{(1-\alpha_4)A_4\left[(i\hat{z})^{-\alpha_4} + \left(-i\bar{\hat{z}}\right)^{-\alpha_4}\right] - (1-\beta_4)iB_4\left[(i\hat{z})^{-\beta_4} - \left(-i\bar{\hat{z}}\right)^{-\beta_4}\right]\right\}$$
$$\frac{\partial P_4}{\partial \hat{y}} = \frac{1}{2}i\left\{(1-\alpha_4)A_4\left[(i\hat{z})^{-\alpha_4} - \left(-i\bar{\hat{z}}\right)^{-\alpha_4}\right] - (1-\beta_4)iB_4\left[(i\hat{z})^{-\beta_4} + \left(-i\bar{\hat{z}}\right)^{-\beta_4}\right]\right\}$$

Equation 57

For the same reason as mentioned in the

Finite Analytic Method section; $\alpha_i = \beta_i = \alpha$. This is due to the fact that if unique solutions are to be possible, this has to occur.

From Equation 48, the various flux expressions for all the quadrants are as follows:

$$Q_{x_1} = -\int_0^{\frac{1}{2}\Delta y} \left(k_{1_{11}} \frac{\partial P_1}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{1_{12}} \frac{\partial P_1}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y}\right) \partial y$$

$$Q_{y_1} = -\int_0^{\frac{1}{2}\Delta x} \left(k_{1_{12}} \frac{\partial P_1}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{1_{22}} \frac{\partial P_1}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y}\right) \partial x$$

$$Q_{x_2} = -\int_0^{\frac{1}{2}\Delta y} \left(k_{2_{11}} \frac{\partial P_2}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{2_{12}} \frac{\partial P_2}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y}\right) \partial y$$

$$Q_{y_2} = -\int_0^{\frac{1}{2}\Delta x} \left(k_{2_{12}} \frac{\partial P_2}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{2_{22}} \frac{\partial P_2}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y}\right) \partial x$$

$$Q_{x_3} = -\int_0^{\frac{1}{2}\Delta y} \left(k_{3_{11}} \frac{\partial P_3}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{3_{12}} \frac{\partial P_3}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y}\right) \partial y$$

$$Q_{y_3} = -\int_0^{\frac{1}{2}\Delta x} \left(k_{3_{12}} \frac{\partial P_3}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{3_{22}} \frac{\partial P_3}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y}\right) \partial y$$

$$Q_{x_4} = -\int_0^{\frac{1}{2}\Delta y} \left(k_{4_{11}} \frac{\partial P_4}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{4_{12}} \frac{\partial P_4}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y}\right) \partial y$$

$$Q_{y_4} = -\int_0^{\frac{1}{2}\Delta x} \left(k_{4_{12}} \frac{\partial P_4}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{4_{22}} \frac{\partial P_4}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y}\right) \partial x$$

Equation 58

The final pressure equations must abide by the same criteria as was set in Equation 17, however, using the new coordinate system. Thus, the criteria can be expressed as follows:

1. Along the positive x-axis:

$$P_1(\hat{x},\hat{y}) = P_4(\hat{x},\hat{y})$$
$$k_{1_{12}}\frac{\partial P_1}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial x} + k_{1_{22}}\frac{\partial P_1}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial y} = k_{4_{12}}\frac{\partial P_4}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial x} + k_{4_{22}}\frac{\partial P_4}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial y}$$

2. Along the positive y-axis:

$$P_2(\hat{x},\hat{y}) = P_1(\hat{x},\hat{y})$$
$$k_{2_{11}}\frac{\partial P_2}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial x} + k_{2_{12}}\frac{\partial P_2}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial y} = k_{1_{11}}\frac{\partial P_1}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial x} + k_{1_{12}}\frac{\partial P_1}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial y}$$

3. Along the negative x-axis:

$$P_3(\hat{x},\hat{y}) = P_2(\hat{x},\hat{y})$$
$$k_{3_{12}}\frac{\partial P_3}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial x} + k_{3_{22}}\frac{\partial P_3}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial y} = k_{2_{12}}\frac{\partial P_2}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial x} + k_{2_{22}}\frac{\partial P_2}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial y}$$

4. Along the negative y-axis:

$$P_4(\hat{x},\hat{y}) = P_3(\hat{x},\hat{y})$$
$$k_{4_{11}}\frac{\partial P_4}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial x} + k_{4_{12}}\frac{\partial P_4}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial y} = k_{3_{11}}\frac{\partial P_3}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial x} + k_{3_{12}}\frac{\partial P_3}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial y}$$

Equation 59

Using the polar form of complex numbers, the pressure equations set up in Equation 56 can be written as:

$$P_1(\hat{x},\hat{y}) = A_1\hat{r}_1^{\,1-\alpha}\cos\left[\hat{\theta}_1(1-\alpha)\right] + B_1\hat{r}_1^{\,1-\alpha}\sin\left[\hat{\theta}_1(1-\alpha)\right] = A_1\hat{\lambda}_1$$
$$P_2(\hat{x},\hat{y}) = A_2\hat{r}_2^{\,1-\alpha}\cos\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right] - B_2\hat{r}_2^{\,1-\alpha}\sin\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right] = A_2\hat{\lambda}_2$$
$$P_3(\hat{x},\hat{y}) = A_3\hat{r}_3^{\,1-\alpha}\cos\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] - B_3\hat{r}_3^{\,1-\alpha}\sin\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] = A_3\hat{\lambda}_3$$
$$P_4(\hat{x},\hat{y}) = A_4\hat{r}_4^{\,1-\alpha}\cos\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right] - B_4\hat{r}_4^{\,1-\alpha}\sin\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right] = A_4\hat{\lambda}_4$$

Equation 60

Note: $\hat{r}_i = \sqrt{\hat{x}^2 + \hat{y}^2}$

With this, the analytical nodal solution for the anisotropic case is expressed. However, the unknowns A_i, B_i, α are yet to be determined. This is delayed till after the numerical fragment is examined (see end of section 3.3). Below is the necessary approach to achieve the correct numerical expressions for the fluxes needed in the implementation of the anisotropic extension for the finite analytic method.

Similar to before, the fluxes need to be expressed at the first and third quadrant, and to be included in the final solution for the inter nodal transmissivity. This is done to uphold the continuity critera described above. Taking basis in Equation 58, the fluxes can be denoted as such:

Note: in following equations; $\Delta x = \frac{1}{2}\Delta x$ and $\Delta y = \frac{1}{2}\Delta y$. This was done again to make the transition into the program easier as the foundation of the code that was used in this study applied this notation.

$$Q_{x_1} = -\int_0^{\Delta y} \left(k_{1_{11}} \frac{\partial P_1}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{1_{12}} \frac{\partial P_1}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y} \right) \partial y = A_1 \hat{\lambda}_{x_1}$$

$$Q_{y_1} = -\int_0^{\Delta x} \left(k_{1_{12}} \frac{\partial P_1}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{1_{22}} \frac{\partial P_1}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y} \right) \partial x = A_1 \hat{\lambda}_{y_1}$$

$$Q_{x_3} = -\int_0^{\Delta y} \left(k_{3_{11}} \frac{\partial P_3}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{3_{12}} \frac{\partial P_3}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y} \right) \partial y = A_3 \hat{\lambda}_{x_3}$$

$$Q_{y_3} = -\int_0^{\Delta x} \left(k_{3_{12}} \frac{\partial P_3}{\partial \hat{x}} \frac{\partial \hat{x}}{\partial x} + k_{3_{22}} \frac{\partial P_3}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial y} \right) \partial y = A_3 \hat{\lambda}_{y_3}$$

Equation 61

Using the following facts:

$$\frac{\partial \hat{x}}{\partial x} = \frac{\cos \tilde{\theta}_i}{\sqrt{\tilde{k}_{i_1}}}, \qquad \frac{\partial \hat{y}}{\partial y} = \frac{\cos \tilde{\theta}_i}{\sqrt{\tilde{k}_{i_2}}}$$

$$\frac{\partial \hat{r}}{\partial \hat{x}} = \frac{\hat{x}}{\hat{r}}, \qquad \frac{\partial \hat{r}}{\partial \hat{y}} = \frac{\hat{y}}{\hat{r}}$$

$$\frac{\partial \hat{\theta}}{\partial \hat{x}} = -\frac{\hat{y}}{\hat{r}^2}, \qquad \frac{\partial \hat{\theta}}{\partial \hat{y}} = \frac{\hat{x}}{\hat{r}^2}$$

$\hat{\lambda}_{x,y_i}$ can be expressed as follows:

$$\hat{\lambda}_{x_1} = k_{1_{11}} \frac{y \cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_1}}} \left\{ \frac{\hat{x}}{\hat{r}_1} \hat{r}_1^{\,-\alpha} \left(A_1 \cos\left[\hat{\theta}_1(1-\alpha)\right] + B_1 \sin\left[\hat{\theta}_1(1-\alpha)\right] \right) \right.$$
$$\left. + \frac{-\hat{y}}{\hat{r}_1^{\,2}} \hat{r}_1^{\,1-\alpha} \left(-A_1 \cos\left[\hat{\theta}_1(1-\alpha)\right] + B_1 \sin\left[\hat{\theta}_1(1-\alpha)\right] \right) \right\}$$
$$+ k_{1_{12}} \frac{y \cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_2}}} \left\{ \frac{\hat{y}}{\hat{r}_1} \hat{r}_1^{\,-\alpha} \left(A_1 \cos\left[\hat{\theta}_1(1-\alpha)\right] + B_1 \sin\left[\hat{\theta}_1(1-\alpha)\right] \right) \right.$$
$$\left. + \frac{\hat{x}}{\hat{r}_1^{\,2}} \hat{r}_1^{\,1-\alpha} \left(-A_1 \cos\left[\hat{\theta}_1(1-\alpha)\right] + B_1 \sin\left[\hat{\theta}_1(1-\alpha)\right] \right) \right\}$$

$$\hat{\lambda}_{y_1} = k_{1_{12}} \frac{x \cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_1}}} \left\{ \frac{\hat{x}}{\hat{r}_1} \hat{r}_1^{\,-\alpha} \left(A_1 \cos\left[\hat{\theta}_1(1-\alpha)\right] + B_1 \sin\left[\hat{\theta}_1(1-\alpha)\right] \right) \right.$$
$$\left. + \frac{-\hat{y}}{\hat{r}_1^{\,2}} \hat{r}_1^{\,1-\alpha} \left(-A_1 \cos\left[\hat{\theta}_1(1-\alpha)\right] + B_1 \sin\left[\hat{\theta}_1(1-\alpha)\right] \right) \right\}$$
$$+ k_{1_{22}} \frac{x \cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_2}}} \left\{ \frac{\hat{y}}{\hat{r}_1} \hat{r}_1^{\,-\alpha} \left(A_1 \cos\left[\hat{\theta}_1(1-\alpha)\right] + B_1 \sin\left[\hat{\theta}_1(1-\alpha)\right] \right) \right.$$
$$\left. + \frac{\hat{x}}{\hat{r}_1^{\,2}} \hat{r}_1^{\,1-\alpha} \left(-A_1 \cos\left[\hat{\theta}_1(1-\alpha)\right] + B_1 \sin\left[\hat{\theta}_1(1-\alpha)\right] \right) \right\}$$

$$\hat{\lambda}_{x_3} = k_{3_{11}} \frac{y \cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_1}}} \left\{ \frac{\hat{x}}{\hat{r}_3} \hat{r}_3^{\,-\alpha} \left(A_3 \cos\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] - B_3 \sin\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] \right) \right.$$
$$\left. + \frac{-\hat{y}}{\hat{r}_3^{\,2}} \hat{r}_3^{\,1-\alpha} \left(-A_3 \cos\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] - B_3 \sin\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] \right) \right\}$$
$$+ k_{3_{12}} \frac{y \cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_2}}} \left\{ \frac{\hat{y}}{\hat{r}_3} \hat{r}_3^{\,-\alpha} \left(A_3 \cos\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] - B_3 \sin\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] \right) \right.$$
$$\left. + \frac{\hat{x}}{\hat{r}_3^{\,2}} \hat{r}_3^{\,1-\alpha} \left(-A_3 \cos\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] - B_3 \sin\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] \right) \right\}$$

$$\hat{\lambda}_{y_3} = k_{3_{12}} \frac{x \cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_1}}} \left\{ \frac{\hat{x}}{\hat{r}_3} \hat{r}_3^{\,-\alpha} \left(A_3 \cos\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] - B_3 \sin\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] \right) \right.$$
$$\left. + \frac{-\hat{y}}{\hat{r}_3^{\,2}} \hat{r}_3^{\,1-\alpha} \left(-A_3 \cos\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] - B_3 \sin\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] \right) \right\}$$
$$+ k_{3_{22}} \frac{x \cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_2}}} \left\{ \frac{\hat{y}}{\hat{r}_3} \hat{r}_3^{\,-\alpha} \left(A_3 \cos\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] - B_3 \sin\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] \right) \right.$$
$$\left. + \frac{\hat{x}}{\hat{r}_3^{\,2}} \hat{r}_3^{\,1-\alpha} \left(-A_3 \cos\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] - B_3 \sin\left[(\hat{\theta}_3 - \pi)(1-\alpha)\right] \right) \right\}$$

Equation 62

And thus, the final solution for the inter nodal transmissivity can be expressed as such:

$$\gamma_{x_1} = \frac{\hat{\lambda}_{x_1}}{\hat{\lambda}_2 - \hat{\lambda}_1}, \qquad \gamma_{y_1} = \frac{\hat{\lambda}_{y_1}}{\hat{\lambda}_4 - \hat{\lambda}_1}$$
$$\gamma_{x_2} = \frac{\hat{\lambda}_{x_1}}{\hat{\lambda}_2 - \hat{\lambda}_1}, \qquad \gamma_{y_2} = \frac{\hat{\lambda}_{y_3}}{\hat{\lambda}_3 - \hat{\lambda}_2}$$
$$\gamma_{x_3} = \frac{\hat{\lambda}_{x_3}}{\hat{\lambda}_3 - \hat{\lambda}_4}, \qquad \gamma_{y_3} = \frac{\hat{\lambda}_{y_3}}{\hat{\lambda}_3 - \hat{\lambda}_2}$$
$$\gamma_{x_4} = \frac{\hat{\lambda}_{x_3}}{\hat{\lambda}_3 - \hat{\lambda}_4}, \qquad \gamma_{y_4} = \frac{\hat{\lambda}_{y_1}}{\hat{\lambda}_4 - \hat{\lambda}_1}$$

Equation 63

Lastly, to solve for the unknowns A_i and B_i, which are functions of α, a backward substitution needs to be applied similar to that described in the previous section. A_i and B_i form a cyclical behavior (see Equation 66), where the solution of one is determined by the other. As such, by expressing equations for these unknowns, it is not only possible to set up a solution for each, but also solve for α.

Seeing as how these sets of equations are much more involved than before, several temporary variables and changes in notation need to be established so as to keep the expressions more contained.

Starting by first expressing the following quantities with these notations:

$$\hat{x}_i = \frac{x\cos\tilde{\theta}_i + y\sin\tilde{\theta}_i}{\sqrt{\tilde{k}_{i_1}}}, \qquad \hat{y}_i = \frac{y\cos\tilde{\theta}_i - x\sin\tilde{\theta}_i}{\sqrt{\tilde{k}_{i_2}}}$$

$$\frac{\partial\hat{x}_i}{\partial x} = \frac{\cos\tilde{\theta}_i}{\sqrt{\tilde{k}_{i_1}}}, \qquad \frac{\partial\hat{y}_i}{\partial y} = \frac{\cos\tilde{\theta}_i}{\sqrt{\tilde{k}_{i_2}}}$$

$$\hat{r}_i = \sqrt{\hat{x}_i{}^2 + \hat{y}_i{}^2}, \qquad \hat{\theta}_i = \theta - \tilde{\theta}_i$$

$$\frac{\partial\hat{r}_i}{\partial\hat{x}_i} = \frac{\hat{x}_i}{\hat{r}_i}, \qquad \frac{\partial\hat{r}_i}{\partial\hat{y}_i} = \frac{\hat{y}_i}{\hat{r}_i}, \qquad \frac{\partial\hat{\theta}_i}{\partial\hat{x}_i} = -\frac{\hat{y}_i}{\hat{r}_i{}^2}, \qquad \frac{\partial\hat{\theta}_i}{\partial\hat{y}_i} = \frac{\hat{x}_i}{\hat{r}_i{}^2}$$

Equation 64

Where θ is the angle between the vertex and the grid node in the standard x, y Cartesian coordinate system. Moreover, the following set of definitions are introduced:

1. Along the positive x-axis ($y = 0$):

$P_1 = P_4$:

$$a_{1_1} = \hat{r}_1^{\,1-\alpha} \cos\left[\hat{\theta}_1(1-\alpha)\right]$$
$$b_{1_1} = \hat{r}_1^{\,1-\alpha} \sin\left[\hat{\theta}_1(1-\alpha)\right]$$
$$a_{4_1} = \hat{r}_4^{\,1-\alpha} \cos\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]$$
$$b_{4_1} = \hat{r}_4^{\,1-\alpha} \sin\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]$$

$Q_{y_1} = Q_{y_4}$:

$$a_{1_2} = k_{1_{12}} \frac{\cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_1}}}(1-\alpha)\left\{\frac{\partial \hat{r}_1}{\partial \hat{x}_1}\hat{r}_1^{\,-\alpha}\cos\left[\hat{\theta}_1(1-\alpha)\right] - \frac{\partial \hat{\theta}_1}{\partial \hat{x}_1}\hat{r}_1^{\,1-\alpha}\sin\left[\hat{\theta}_1(1-\alpha)\right]\right\}$$
$$+ k_{1_{22}} \frac{\cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_2}}}(1-\alpha)\left\{\frac{\partial \hat{r}_1}{\partial \hat{y}_1}\hat{r}_1^{\,-\alpha}\cos\left[\hat{\theta}_1(1-\alpha)\right] - \frac{\partial \hat{\theta}_1}{\partial \hat{y}_1}\hat{r}_1^{\,1-\alpha}\sin\left[\hat{\theta}_1(1-\alpha)\right]\right\}$$

$$b_{1_2} = k_{1_{12}} \frac{\cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_1}}}(1-\alpha)\left\{\frac{\partial \hat{r}_1}{\partial \hat{x}_1}\hat{r}_1^{\,-\alpha}\sin\left[\hat{\theta}_1(1-\alpha)\right] + \frac{\partial \hat{\theta}_1}{\partial \hat{x}_1}\hat{r}_1^{\,1-\alpha}\cos\left[\hat{\theta}_1(1-\alpha)\right]\right\}$$
$$+ k_{1_{22}} \frac{\cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_2}}}(1-\alpha)\left\{\frac{\partial \hat{r}_1}{\partial \hat{y}_1}\hat{r}_1^{\,-\alpha}\sin\left[\hat{\theta}_1(1-\alpha)\right] - \frac{\partial \hat{\theta}_1}{\partial \hat{y}_1}\hat{r}_1^{\,1-\alpha}\cos\left[\hat{\theta}_1(1-\alpha)\right]\right\}$$

$$a_{4_2} = k_{4_{12}} \frac{\cos\tilde{\theta}_4}{\sqrt{\tilde{k}_{4_1}}}(1-\alpha)\left\{\frac{\partial \hat{r}_4}{\partial \hat{x}_4}\hat{r}_4^{\,-\alpha}\cos\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right.$$
$$\left. - \frac{\partial \hat{\theta}_4}{\partial \hat{x}_4}\hat{r}_4^{\,1-\alpha}\sin\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$
$$+ k_{4_{22}} \frac{\cos\tilde{\theta}_4}{\sqrt{\tilde{k}_{4_2}}}(1-\alpha)\left\{\frac{\partial \hat{r}_4}{\partial \hat{y}_4}\hat{r}_4^{\,-\alpha}\cos\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right.$$
$$\left. - \frac{\partial \hat{\theta}_4}{\partial \hat{y}_4}\hat{r}_4^{\,1-\alpha}\sin\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$

$$b_{4_2} = k_{4_{12}} \frac{\cos\tilde{\theta}_4}{\sqrt{\tilde{k}_{4_1}}}(1-\alpha)\left\{\frac{\partial \hat{r}_4}{\partial \hat{x}_4}\hat{r}_4^{\,-\alpha}\sin\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right.$$
$$\left. + \frac{\partial \hat{\theta}_4}{\partial \hat{x}_4}\hat{r}_4^{\,1-\alpha}\cos\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$
$$+ k_{4_{22}} \frac{\cos\tilde{\theta}_4}{\sqrt{\tilde{k}_{4_2}}}(1-\alpha)\left\{\frac{\partial \hat{r}_4}{\partial \hat{y}_4}\hat{r}_4^{\,-\alpha}\sin\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right.$$
$$\left. + \frac{\partial \hat{\theta}_4}{\partial \hat{y}_4}\hat{r}_4^{\,1-\alpha}\cos\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$

2. Along the positive y-axis ($x = 0$):

$P_2 = P_1$:

$$a_{2_1} = \hat{r}_2^{\,1-\alpha} \cos\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]$$
$$b_{2_1} = -\hat{r}_2^{\,1-\alpha} \sin\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]$$
$$a_{1_1} = \hat{r}_1^{\,1-\alpha} \cos\left[\hat{\theta}_1(1-\alpha)\right]$$
$$b_{1_1} = \hat{r}_1^{\,1-\alpha} \sin\left[\hat{\theta}_1(1-\alpha)\right]$$

$Q_{x_2} = Q_{x_1}$:

$$a_{2_2} = k_{2_{11}} \frac{\cos\tilde{\theta}_2}{\sqrt{\tilde{k}_{2_1}}}(1-\alpha)\left\{\frac{\partial \hat{r}_2}{\partial \hat{x}_2}\hat{r}_2^{\,-\alpha}\cos\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right] - \frac{\partial \hat{\theta}_2}{\partial \hat{x}_2}\hat{r}_2^{\,1-\alpha}\sin\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$
$$+ k_{2_{12}} \frac{\cos\tilde{\theta}_2}{\sqrt{\tilde{k}_{2_2}}}(1-\alpha)\left\{\frac{\partial \hat{r}_2}{\partial \hat{y}_2}\hat{r}_2^{\,-\alpha}\cos\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right] - \frac{\partial \hat{\theta}_2}{\partial \hat{y}_2}\hat{r}_2^{\,1-\alpha}\sin\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$

$$b_{2_2} = -k_{2_{11}} \frac{\cos\tilde{\theta}_2}{\sqrt{\tilde{k}_{2_1}}}(1-\alpha)\left\{\frac{\partial \hat{r}_2}{\partial \hat{x}_2}\hat{r}_2^{\,-\alpha}\sin\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right] + \frac{\partial \hat{\theta}_2}{\partial \hat{x}_2}\hat{r}_2^{\,1-\alpha}\cos\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$
$$- k_{2_{12}} \frac{\cos\tilde{\theta}_2}{\sqrt{\tilde{k}_{2_2}}}(1-\alpha)\left\{\frac{\partial \hat{r}_2}{\partial \hat{y}_2}\hat{r}_2^{\,-\alpha}\sin\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right] + \frac{\partial \hat{\theta}_2}{\partial \hat{y}_2}\hat{r}_2^{\,1-\alpha}\cos\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$

$$a_{1_2} = k_{1_{12}} \frac{\cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_1}}}(1-\alpha)\left\{\frac{\partial \hat{r}_1}{\partial \hat{x}_1}\hat{r}_1^{\,-\alpha}\cos\left[\hat{\theta}_1(1-\alpha)\right] - \frac{\partial \hat{\theta}_1}{\partial \hat{x}_1}\hat{r}_1^{\,1-\alpha}\sin\left[\hat{\theta}_1(1-\alpha)\right]\right\}$$
$$+ k_{1_{22}} \frac{\cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_2}}}(1-\alpha)\left\{\frac{\partial \hat{r}_1}{\partial \hat{y}_1}\hat{r}_1^{\,-\alpha}\cos\left[\hat{\theta}_1(1-\alpha)\right] - \frac{\partial \hat{\theta}_1}{\partial \hat{y}_1}\hat{r}_1^{\,1-\alpha}\sin\left[\hat{\theta}_1(1-\alpha)\right]\right\}$$

$$b_{1_2} = k_{1_{12}} \frac{\cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_1}}}(1-\alpha)\left\{\frac{\partial \hat{r}_1}{\partial \hat{x}_1}\hat{r}_1^{\,-\alpha}\sin\left[\hat{\theta}_1(1-\alpha)\right] + \frac{\partial \hat{\theta}_1}{\partial \hat{x}_1}\hat{r}_1^{\,1-\alpha}\cos\left[\hat{\theta}_1(1-\alpha)\right]\right\}$$
$$+ k_{1_{22}} \frac{\cos\tilde{\theta}_1}{\sqrt{\tilde{k}_{1_2}}}(1-\alpha)\left\{\frac{\partial \hat{r}_1}{\partial \hat{y}_1}\hat{r}_1^{\,-\alpha}\sin\left[\hat{\theta}_1(1-\alpha)\right] - \frac{\partial \hat{\theta}_1}{\partial \hat{y}_1}\hat{r}_1^{\,1-\alpha}\cos\left[\hat{\theta}_1(1-\alpha)\right]\right\}$$

3. Along the negative x-axis ($y = 0$):

$P_3 = P_2$:

$$a_{3_1} = \hat{r}_3{}^{1-\alpha} \cos\left[\left(\hat{\theta}_3 + \pi\right)(1-\alpha)\right]$$
$$b_{3_1} = -\hat{r}_3{}^{1-\alpha} \sin\left[\left(\hat{\theta}_3 + \pi\right)(1-\alpha)\right]$$
$$a_{2_1} = \hat{r}_2{}^{1-\alpha} \sin\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]$$
$$b_{2_1} = -\hat{r}_2{}^{1-\alpha} \sin\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]$$

$Q_{y_3} = Q_{y_2}$:

$$a_{3_2} = k_{3_{12}} \frac{\cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_1}}}(1-\alpha)\left\{\frac{\partial\hat{r}_3}{\partial\hat{x}_3}\hat{r}_3{}^{-\alpha}\cos\left[\left(\hat{\theta}_3+\pi\right)(1-\alpha)\right] - \frac{\partial\hat{\theta}_3}{\partial\hat{x}_3}\hat{r}_3{}^{1-\alpha}\sin\left[\left(\hat{\theta}_3+\pi\right)(1-\alpha)\right]\right\} + k_{3_{22}}\frac{\cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_2}}}(1-\alpha)\left\{\frac{\partial\hat{r}_3}{\partial\hat{y}_3}\hat{r}_3{}^{-\alpha}\cos\left[\left(\hat{\theta}_3+\pi\right)(1-\alpha)\right] - \frac{\partial\hat{\theta}_3}{\partial\hat{y}_3}\hat{r}_3{}^{1-\alpha}\sin\left[\left(\hat{\theta}_3+\pi\right)(1-\alpha)\right]\right\}$$

$$b_{3_2} = -k_{3_{12}} \frac{\cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_1}}}(1-\alpha)\left\{\frac{\partial\hat{r}_3}{\partial\hat{x}_3}\hat{r}_3{}^{-\alpha}\sin\left[\left(\hat{\theta}_3+\pi\right)(1-\alpha)\right] + \frac{\partial\hat{\theta}_3}{\partial\hat{x}_3}\hat{r}_3{}^{1-\alpha}\cos\left[\left(\hat{\theta}_3+\pi\right)(1-\alpha)\right]\right\} - k_{3_{22}}\frac{\cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_2}}}(1-\alpha)\left\{\frac{\partial\hat{r}_3}{\partial\hat{y}_3}\hat{r}_3{}^{-\alpha}\sin\left[\left(\hat{\theta}_3+\pi\right)(1-\alpha)\right] + \frac{\partial\hat{\theta}_3}{\partial\hat{y}_3}\hat{r}_3{}^{1-\alpha}\cos\left[\left(\hat{\theta}_3+\pi\right)(1-\alpha)\right]\right\}$$

$$a_{2_2} = k_{2_{11}} \frac{\cos\tilde{\theta}_2}{\sqrt{\tilde{k}_{2_1}}}(1-\alpha)\left\{\frac{\partial\hat{r}_2}{\partial\hat{x}_2}\hat{r}_2{}^{-\alpha}\cos\left[\left(\hat{\theta}_2-\frac{\pi}{2}\right)(1-\alpha)\right] - \frac{\partial\hat{\theta}_2}{\partial\hat{x}_2}\hat{r}_2{}^{1-\alpha}\sin\left[\left(\hat{\theta}_2-\frac{\pi}{2}\right)(1-\alpha)\right]\right\} + k_{2_{12}}\frac{\cos\tilde{\theta}_2}{\sqrt{\tilde{k}_{2_2}}}(1-\alpha)\left\{\frac{\partial\hat{r}_2}{\partial\hat{y}_2}\hat{r}_2{}^{-\alpha}\cos\left[\left(\hat{\theta}_2-\frac{\pi}{2}\right)(1-\alpha)\right] - \frac{\partial\hat{\theta}_2}{\partial\hat{y}_2}\hat{r}_2{}^{1-\alpha}\sin\left[\left(\hat{\theta}_2-\frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$

$$b_{2_2} = -k_{2_{11}} \frac{\cos\tilde{\theta}_2}{\sqrt{\tilde{k}_{2_1}}}(1-\alpha)\left\{\frac{\partial \hat{r}_2}{\partial \hat{x}_2}\hat{r}_2{}^{-\alpha}\sin\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]\right.$$
$$\left. + \frac{\partial \hat{\theta}_2}{\partial \hat{x}_2}\hat{r}_2{}^{1-\alpha}\cos\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$
$$- k_{2_{12}} \frac{\cos\tilde{\theta}_2}{\sqrt{\tilde{k}_{2_2}}}(1-\alpha)\left\{\frac{\partial \hat{r}_2}{\partial \hat{y}_2}\hat{r}_2{}^{-\alpha}\sin\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]\right.$$
$$\left. + \frac{\partial \hat{\theta}_2}{\partial \hat{y}_2}\hat{r}_2{}^{1-\alpha}\cos\left[\left(\hat{\theta}_2 - \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$

4. Along the negative y-axis ($x = 0$):
   $P_4 = P_3$:

$$a_{4_1} = \hat{r}_4{}^{1-\alpha}\cos\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]$$
$$b_{4_1} = \hat{r}_4{}^{1-\alpha}\sin\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]$$
$$a_{3_1} = \hat{r}_3{}^{1-\alpha}\cos[(\hat{\theta}_3 + \pi)(1-\alpha)]$$
$$b_{3_1} = -\hat{r}_3{}^{1-\alpha}\sin[(\hat{\theta}_3 + \pi)(1-\alpha)]$$

   $Q_{x_4} = Q_{x_3}$:

$$a_{4_2} = k_{4_{12}} \frac{\cos\tilde{\theta}_4}{\sqrt{\tilde{k}_{4_1}}}(1-\alpha)\left\{\frac{\partial \hat{r}_4}{\partial \hat{x}_4}\hat{r}_4{}^{-\alpha}\cos\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right.$$
$$\left. - \frac{\partial \hat{\theta}_4}{\partial \hat{x}_4}\hat{r}_4{}^{1-\alpha}\sin\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$
$$+ k_{4_{22}} \frac{\cos\tilde{\theta}_4}{\sqrt{\tilde{k}_{4_2}}}(1-\alpha)\left\{\frac{\partial \hat{r}_4}{\partial \hat{y}_4}\hat{r}_4{}^{-\alpha}\cos\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right.$$
$$\left. - \frac{\partial \hat{\theta}_4}{\partial \hat{y}_4}\hat{r}_4{}^{1-\alpha}\sin\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$

$$b_{4_2} = k_{4_{12}} \frac{\cos\tilde{\theta}_4}{\sqrt{\tilde{k}_{4_1}}}(1-\alpha)\left\{\frac{\partial \hat{r}_4}{\partial \hat{x}_4}\hat{r}_4{}^{-\alpha}\sin\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right.$$
$$\left. + \frac{\partial \hat{\theta}_4}{\partial \hat{x}_4}\hat{r}_4{}^{1-\alpha}\cos\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$
$$+ k_{4_{22}} \frac{\cos\tilde{\theta}_4}{\sqrt{\tilde{k}_{4_2}}}(1-\alpha)\left\{\frac{\partial \hat{r}_4}{\partial \hat{y}_4}\hat{r}_4{}^{-\alpha}\sin\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right.$$
$$\left. + \frac{\partial \hat{\theta}_4}{\partial \hat{y}_4}\hat{r}_4{}^{1-\alpha}\cos\left[\left(\hat{\theta}_4 + \frac{\pi}{2}\right)(1-\alpha)\right]\right\}$$

$$
\begin{aligned}
a_{3_2} = k_{3_{12}} \frac{\cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_1}}}(1-\alpha)\Bigg\{ & \frac{\partial \hat{r}_3}{\partial \hat{x}_3}\hat{r}_3{}^{-\alpha}\cos\big[(\hat{\theta}_3+\pi)(1-\alpha)\big] \\
& - \frac{\partial \hat{\theta}_3}{\partial \hat{x}_3}\hat{r}_3{}^{1-\alpha}\sin\big[(\hat{\theta}_3+\pi)(1-\alpha)\big]\Bigg\} \\
& + k_{3_{22}} \frac{\cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_2}}}(1-\alpha)\Bigg\{ \frac{\partial \hat{r}_3}{\partial \hat{y}_3}\hat{r}_3{}^{-\alpha}\cos\big[(\hat{\theta}_3+\pi)(1-\alpha)\big] \\
& - \frac{\partial \hat{\theta}_3}{\partial \hat{y}_3}\hat{r}_3{}^{1-\alpha}\sin\big[(\hat{\theta}_3+\pi)(1-\alpha)\big]\Bigg\} \\
b_{3_2} = -k_{3_{12}} \frac{\cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_1}}}(1-\alpha)\Bigg\{ & \frac{\partial \hat{r}_3}{\partial \hat{x}_3}\hat{r}_3{}^{-\alpha}\sin\big[(\hat{\theta}_3+\pi)(1-\alpha)\big] \\
& + \frac{\partial \hat{\theta}_3}{\partial \hat{x}_3}\hat{r}_3{}^{1-\alpha}\cos\big[(\hat{\theta}_3+\pi)(1-\alpha)\big]\Bigg\} \\
& - k_{3_{22}} \frac{\cos\tilde{\theta}_3}{\sqrt{\tilde{k}_{3_2}}}(1-\alpha)\Bigg\{ \frac{\partial \hat{r}_3}{\partial \hat{y}_3}\hat{r}_3{}^{-\alpha}\sin\big[(\hat{\theta}_3+\pi)(1-\alpha)\big] \\
& + \frac{\partial \hat{\theta}_3}{\partial \hat{y}_3}\hat{r}_3{}^{1-\alpha}\cos\big[(\hat{\theta}_3+\pi)(1-\alpha)\big]\Bigg\}
\end{aligned}
$$

Equation 65

With this simplification, both the pressure continuity and the flux continuity can be systematically written as:

1. Along the positive x-axis ($y = 0$):
$$A_4 a_{4_1} + B_4 b_{4_1} = A_1 a_{1_1} + B_1 b_{1_1}$$
$$A_4 a_{4_2} + B_4 b_{4_2} = A_1 a_{1_2} + B_1 b_{1_2}$$
2. Along the positive y-axis ($x = 0$):
$$A_1 a_{1_1} + B_1 b_{1_1} = A_2 a_{2_1} + B_2 b_{2_1}$$
$$A_1 a_{1_2} + B_1 b_{1_2} = A_2 a_{2_2} + B_2 b_{2_2}$$
3. Along the negative x-axis ($y = 0$):
$$A_2 a_{2_1} + B_2 b_{2_1} = A_3 a_{3_1} + B_3 b_{3_1}$$
$$A_2 a_{2_2} + B_2 b_{2_2} = A_3 a_{3_2} + B_3 b_{3_2}$$
4. Along the negative y-axis ($x = 0$):
$$A_3 a_{3_1} + B_3 b_{3_1} = A_4 a_{4_1} + B_4 b_{4_1}$$
$$A_3 a_{3_2} + B_3 b_{3_2} = A_4 a_{4_2} + B_4 b_{4_2}$$

Equation 66

By using Cramer's rule, one can acquire solutions for the individual unknowns. These can be expressed as follows:

$$A_1 = \frac{\left(A_4 a_{4_1} + B_4 b_{4_1}\right) b_{1_2} - \left(A_4 a_{4_2} + B_4 b_{4_2}\right) b_{1_1}}{a_{1_1} b_{1_2} - a_{1_2} b_{1_1}}$$
$$B_1 = \frac{-\left(A_4 a_{4_1} + B_4 b_{4_1}\right) a_{1_2} + \left(A_4 a_{4_2} + B_4 b_{4_2}\right) a_{1_1}}{a_{1_1} b_{1_2} - a_{1_2} b_{1_1}}$$

$$A_2 = \frac{\left(A_1 a_{1_1} + B_1 b_{1_1}\right) b_{2_2} - \left(A_1 a_{1_2} + B_1 b_{1_2}\right) b_{2_1}}{a_{2_1} b_{2_2} - a_{2_2} b_{2_1}}$$
$$B_2 = \frac{-\left(A_1 a_{1_1} + B_1 b_{1_1}\right) a_{2_2} + \left(A_1 a_{1_2} + B_1 b_{1_2}\right) a_{2_1}}{a_{2_1} b_{2_2} - a_{2_2} b_{2_1}}$$

$$A_3 = \frac{\left(A_2 a_{2_1} + B_2 b_{2_1}\right) b_{3_2} - \left(A_2 a_{2_2} + B_2 b_{2_2}\right) b_{3_1}}{a_{3_1} b_{3_2} - a_{3_2} b_{3_1}}$$
$$B_3 = \frac{-\left(A_2 a_{2_1} + B_2 b_{2_1}\right) a_{3_2} + \left(A_2 a_{2_2} + B_2 b_{2_2}\right) a_{3_1}}{a_{3_1} b_{3_2} - a_{3_2} b_{3_1}}$$

$$A_4 = \frac{\left(A_3 a_{3_1} + B_3 b_{3_1}\right) b_{4_2} - \left(A_3 a_{3_2} + B_3 b_{3_2}\right) b_{4_1}}{a_{4_1} b_{4_2} - a_{4_2} b_{4_1}}$$
$$B_4 = \frac{-\left(A_3 a_{3_1} + B_3 b_{3_1}\right) a_{4_2} + \left(A_3 a_{3_2} + B_3 b_{3_2}\right) a_{4_1}}{a_{4_1} b_{4_2} - a_{4_2} b_{4_1}}$$

Equation 67

This allows for the unknowns to be solved as long as α is determined, as it is the only unknown in the equations above other than the subjects.

By changing notation, a more compact set of equations can be formed:

$$u_{1_1} = \frac{a_{4_1}b_{1_2} - a_{4_2}b_{1_1}}{a_{1_1}b_{1_2} - a_{1_2}b_{1_1}}, \qquad u_{1_2} = \frac{-a_{4_1}a_{1_2} + a_{4_2}a_{1_1}}{a_{1_1}b_{1_2} - a_{1_2}b_{1_1}}$$
$$v_{1_1} = \frac{b_{4_1}b_{1_2} - b_{4_2}b_{1_1}}{a_{1_1}b_{1_2} - a_{1_2}b_{1_1}}, \qquad v_{1_2} = \frac{-b_{4_1}a_{1_2} + b_{4_2}a_{1_1}}{a_{1_1}b_{1_2} - a_{1_2}b_{1_1}}$$

$$u_{2_1} = \frac{a_{1_1}b_{2_2} - a_{1_2}b_{2_1}}{a_{2_1}b_{2_2} - a_{2_2}b_{2_1}}, \qquad u_{2_2} = \frac{-a_{1_1}a_{2_2} + a_{1_2}a_{2_1}}{a_{2_1}b_{2_2} - a_{2_2}b_{2_1}}$$
$$v_{2_1} = \frac{b_{1_1}b_{2_2} - b_{1_2}b_{2_1}}{a_{2_1}b_{2_2} - a_{2_2}b_{2_1}}, \qquad v_{2_2} = \frac{-b_{1_1}a_{2_2} + b_{1_2}a_{2_1}}{a_{2_1}b_{2_2} - a_{2_2}b_{2_1}}$$

$$u_{3_1} = \frac{a_{2_1}b_{3_2} - a_{2_2}b_{3_1}}{a_{3_1}b_{3_2} - a_{3_2}b_{3_1}}, \qquad u_{3_2} = \frac{-a_{2_1}a_{3_2} + a_{2_2}a_{3_1}}{a_{3_1}b_{3_2} - a_{3_2}b_{3_1}}$$
$$v_{3_1} = \frac{b_{2_1}b_{3_2} - b_{2_2}b_{3_1}}{a_{3_1}b_{3_2} - a_{3_2}b_{3_1}}, \qquad v_{3_2} = \frac{-b_{2_1}a_{3_2} + b_{2_2}a_{3_1}}{a_{3_1}b_{3_2} - a_{3_2}b_{3_1}}$$

$$u_{4_1} = \frac{a_{3_1}b_{4_2} - a_{3_2}b_{4_1}}{a_{4_1}b_{4_2} - a_{4_2}b_{4_1}}, \qquad u_{4_2} = \frac{-a_{3_1}a_{4_2} + a_{3_2}a_{4_1}}{a_{4_1}b_{4_2} - a_{4_2}b_{4_1}}$$
$$v_{4_1} = \frac{b_{3_1}b_{4_2} - b_{3_2}b_{4_1}}{a_{4_1}b_{4_2} - a_{4_2}b_{4_1}}, \qquad v_{4_2} = \frac{-b_{3_1}a_{4_2} + b_{3_2}a_{4_1}}{a_{4_1}b_{4_2} - a_{4_2}b_{4_1}}$$

Equation 68

Thus, the equations for the unknowns A_i and B_i are as follows:

$$A_1 = A_4u_{1_1} + B_4v_{1_1}$$
$$B_1 = A_4u_{1_2} + B_4v_{1_2}$$

$$A_2 = A_1u_{2_1} + B_1v_{2_1}$$
$$B_2 = A_1u_{2_2} + B_1v_{2_2}$$

$$A_3 = A_2u_{3_1} + B_2v_{3_1}$$
$$B_3 = A_2u_{3_2} + B_2v_{3_2}$$

$$A_4 = A_3u_{4_1} + B_3v_{4_1}$$
$$B_4 = A_3u_{4_2} + B_3v_{4_2}$$

Equation 69

Applying the backward substitution method here, similar to that seen in the

Finite Analytic Method section, a general solution for α can be derived. Starting with the expressions deduced from the substitution:

$$
\begin{aligned}
A_1 &= A_3\left[u_{4_1}u_{1_1} + u_{4_2}v_{1_1}\right] + B_3\left[v_{4_1}u_{1_1} + v_{4_2}v_{1_1}\right] \\
&= A_2\left[u_{3_1}u_{4_1}u_{1_1} + u_{3_1}u_{4_2}v_{1_1} + u_{3_2}v_{4_1}u_{1_1} + u_{3_2}v_{4_2}v_{1_1}\right] \\
&\qquad + B_2\left[v_{3_1}u_{4_1}u_{1_1} + v_{3_1}u_{4_2}v_{1_1} + v_{3_2}v_{4_1}u_{1_1} + v_{3_2}v_{4_2}v_{1_1}\right] \\
&= A_1\left[u_{2_1}u_{3_1}u_{4_1}u_{1_1} + u_{2_1}u_{3_1}u_{4_2}v_{1_1} + u_{2_1}u_{3_2}v_{4_1}u_{1_1} + u_{2_1}u_{3_2}v_{4_2}v_{1_1} + u_{2_2}v_{3_1}u_{4_1}u_{1_1} + u_{2_2}v_{3_1}u_{4_2}v_{1_1}\right. \\
&\qquad \left. + u_{2_2}v_{3_2}v_{4_1}u_{1_1} + u_{2_2}v_{3_2}v_{4_2}v_{1_1}\right] \\
&\qquad + B_1\left[v_{2_1}u_{3_1}u_{4_1}u_{1_1} + v_{2_1}u_{3_1}u_{4_2}v_{1_1} + v_{2_1}u_{3_2}v_{4_1}u_{1_1} + v_{2_1}u_{3_2}v_{4_2}v_{1_1} + v_{2_2}v_{3_1}u_{4_1}u_{1_1}\right. \\
&\qquad \left. + v_{2_2}v_{3_1}u_{4_2}v_{1_1} + v_{2_2}v_{3_2}v_{4_1}u_{1_1} + v_{2_2}v_{3_2}v_{4_2}v_{1_1}\right] \\
&= A_1\varphi_{1_1} + B_1\varphi_{2_1}
\end{aligned}
$$

$$
\begin{aligned}
B_1 &= A_3\left[u_{4_1}u_{1_2} + u_{4_2}v_{1_2}\right] + B_3\left[v_{4_1}u_{1_2} + v_{4_2}v_{1_2}\right] \\
&= A_2\left[u_{3_1}u_{4_1}u_{1_2} + u_{3_1}u_{4_2}v_{1_2} + u_{3_2}v_{4_1}u_{1_2} + u_{3_2}v_{4_2}v_{1_2}\right] \\
&\qquad + B_2\left[v_{3_1}u_{4_1}u_{1_2} + v_{3_1}u_{4_2}v_{1_2} + v_{3_2}v_{4_1}u_{1_2} + v_{3_2}v_{4_2}v_{1_2}\right] \\
&= A_1\left[u_{2_1}u_{3_1}u_{4_1}u_{1_2} + u_{2_1}u_{3_1}u_{4_2}v_{1_2} + u_{2_1}u_{3_2}v_{4_1}u_{1_2} + u_{2_1}u_{3_2}v_{4_2}v_{1_2} + u_{2_2}v_{3_1}u_{4_1}u_{1_2} + u_{2_2}v_{3_1}u_{4_2}v_{1_2}\right. \\
&\qquad \left. + u_{2_2}v_{3_2}v_{4_1}u_{1_2} + u_{2_2}v_{3_2}v_{4_2}v_{1_2}\right] \\
&\qquad + B_1\left[v_{2_1}u_{3_1}u_{4_1}u_{1_2} + v_{2_1}u_{3_1}u_{4_2}v_{1_2} + v_{2_1}u_{3_2}v_{4_1}u_{1_2} + v_{2_1}u_{3_2}v_{4_2}v_{1_2} + v_{2_2}v_{3_1}u_{4_1}u_{1_2}\right. \\
&\qquad \left. + v_{2_2}v_{3_1}u_{4_2}v_{1_2} + v_{2_2}v_{3_2}v_{4_1}u_{1_2} + v_{2_2}v_{3_2}v_{4_2}v_{1_2}\right] \\
&= A_1\varphi_{1_2} + B_1\varphi_{2_2}
\end{aligned}
$$

Equation 70

The variables φ_{i_j} are simply a change of notation, used to make the equations shorter and more comprehensible.

Finally, the following can be obtained:

$$
\begin{aligned}
A_1\left(\varphi_{1_1} - 1\right) + B_1\varphi_{2_1} &= 0 \\
A_1\varphi_{1_2} + B_1\left(\varphi_{2_2} - 1\right) &= 0
\end{aligned}
$$

Equation 71

Again, the corresponding determinant is taken is set to equal zero, as follows:

$$
\begin{vmatrix}
\left(\varphi_{1_1} - 1\right) & \varphi_{2_1} \\
\varphi_{1_2} & \left(\varphi_{2_2} - 1\right)
\end{vmatrix} = \varphi_{1_1}\varphi_{2_2} - \varphi_{1_2}\varphi_{2_1} - \varphi_{1_1} - \varphi_{2_2} + 1 = 0
$$

Equation 72

Note: φ_{i_j} are the components derived from the substitution method.

Now, a solution for Equation 72 must be found in order to determine the value of α. The Newton Secant method would be applied to find the best fit for α. This is a numerical approach to the solution of α, as opposed to the analytical solution that was determined in the previous section for the isotropic

permeability. Using the equation established above, the same solution for α was achieved for the checkerboard test case used in this study, despite the two methods and algorithms varying so greatly in the approach. In addition to finding a solution for α, the constant $C = \frac{B_1}{A_1}$ can be solved by taking either $C = -\frac{(\varphi_{1_1}-1)}{\varphi_{2_1}}$, or $C = -\frac{\varphi_{1_2}}{(\varphi_{2_2}-1)}$. This was also determined to give the same value as the theoretical approach described in the

Finite Analytic Method section.

Newton Secant, also referred to as the Newton-Raphson method, is a root seeking algorithm where better approximations for the roots are found in a successive fashion (Papakonstantinou, 2007). The approach starts with a function f and its corresponding derivative f' (approximated numerically in the approach used in this study, see Equation 73), together with an initial guess x_0 for the root of the function. If the guess converges towards a better assumption, then the process is repeated until a sufficiently better approximation is found based on a criteria and/or margin of error. The general formula takes the form of:

$$f'(x_1) = \frac{f(x_1) - f(x_0)}{x_1 - x_0} \quad \rightarrow \quad f'(x_{n+1}) = \frac{f(x_{n+1}) - f(x_n)}{x_{n+1} - x_n}$$

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} \quad \rightarrow \quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Equation 73

In this manner, the finite analytic method is reconfigured to support anisotropic permeability. This novel method differs greatly from what was recently developed by the same authors of the article that was explored in the

Finite Analytic Method section. The approach that they tackled was a more generalized form of the Laplace equations (Liu & Wang, 2016), where an entirely new set of equations are derived. However, the novel method developed during this study is entirely independent.

6. Results

In this section, a comparison study will be performed where both the finite analytic method and the anisotropic extension that was produced during this study will be evaluated in an isotropic condition. The results from the finite analytic method will be performed first to greater understand the behavior of the scheme. Later, after implementing the anisotropic extension, it will be tested on isotropic data. This is done to validate the method developed during this study, and assure that it is in fact possible to derive the solution in such a manner. Both the isotropic and anisotropic versions of the finite analytic method will be compared to the results achieved from geometric averaging and harmonic averaging schemes, as these are industry standard techniques.

6.1. Dataset

The dataset that was used during this study is a checkboard grid with a permeability contrast ratio of 1:2, 1:10, 1:100, 1:1000, and 1:10000. Additionally, the grid is also divided in either 4x4, 16x16, or 64x64 control volumes. An illustration of the grid used can be seen in Figure 6.1 where the expected results for α and C at the different nodes are also present. The flow is expected to flow along the high permeable zones and pass through the edges, where there is a bottleneck.

The checkerboard example is a commonly used problem when evaluating heterogeneous features. It represents the most severe case of heterogeneity, where the permeability contrast around the grid node is at a maximum, given the parameters used. Computational complexity and the inaccuracy of the solution usually increases with increasing permeability ratio.

The test consisted of evaluating the effective permeability that the schemes estimated. It is possible to determine the effective permeability theoretically when evaluating problems such as the checkerboard example. It can expressed as $k_{eff} = \sqrt{k_1 k_2}$ (Keller, 1964). More general details on the effective permeability can be found in the

Background Research section.

To test the methods, a program was developed from the ground up. This program is essentially a prototype simulator where the different discretization methods were written into. Namely, harmonic mean, geometric mean, and the isotropic and anisotropic versions of the finite analytic method. Thanks to the support of IRIS (International Research Institute of Stavanger) and Helmer André Friis, the backend of the program was able to be developed within the timeframe.

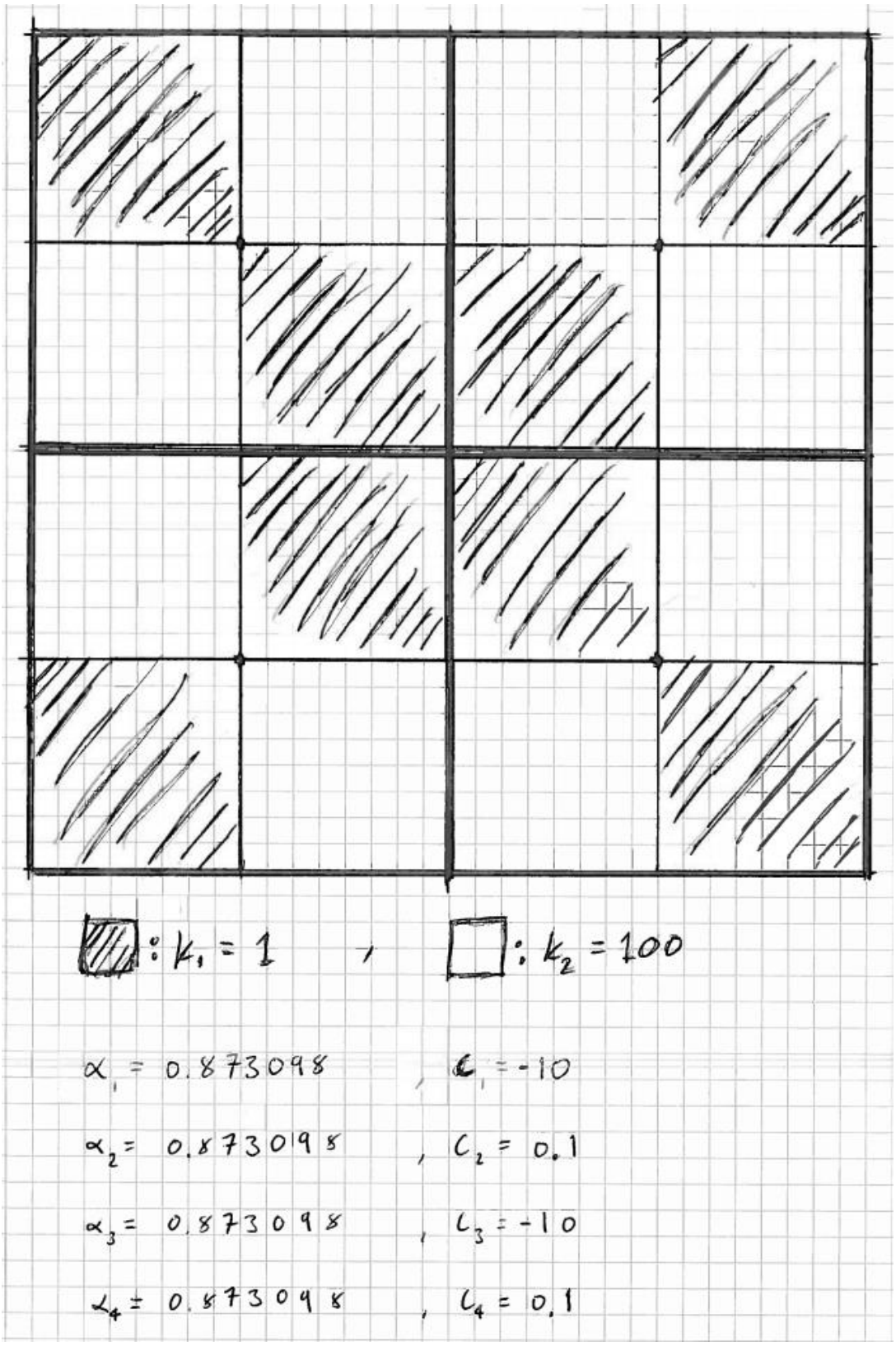

Figure 6.1: A sketch of the grid that was used during this study to test both the isotropic and the anisotropic finite analytic method, with the expected values for the unknowns α and C given the case of permeability ratio of 1:100

6.2. Isotropic Results

Below are three tables showcasing the results for the computed effective permeability for the three methods explored in this study; namely harmonic mean, geometric mean, and the finite analytic method, using the approach discussed in section 3.4 for computing the effective permeability. Additionally, the theoretical result is also presented. The tests were done using varying permeability ratios, as well as different grid refinements, to explore the changes in accuracy and precision for the different methods. The permeability ratios used were 1:2, 1:10, 1:100, 1:1000, and 1:10000, and the three grids refinements used were 4x4, 16x16, and 64x64.

Note: the heterogeneity distribution retained its size and grid node positions, the only changes were made to the permeability ratios and the grid refinement. Additionally, the results from the finite analytic method are directly comparable to that which was discussed in the article (Liu & Wang, 2013).

4x4 Grid				
Permeability Ratio	Harmonic Mean	Geometric Mean	Finite Analytic Method	Theoretical
1 to 2	1.371428571	1.41348468	1.479543924	1.414213562
1 to 10	2.183622829	3.162243364	3.396019342	3.16227766
1 to 100	2.6061994	10.03609011	10.4254933	10
1 to 1000	2.660462465	31.70741447	32.1354455	31.6227766
1 to 10000	2.66604471	100.110472	100.5437339	100

Table 1: Results of the effective permeability for the three methods explored in this study and the theoretical result, for the different permeability ratios, on a 4x4 grid

16x16 Grid				
Permeability Ratio	Harmonic Mean	Geometric Mean	Finite Analytic Method	Theoretical
1 to 2	1.4078486	1.414276021	1.417508858	1.414213562
1 to 10	2.77523138	3.278767573	3.189865668	3.16227766
1 to 100	4.1034331	12.68807143	10.10749708	10
1 to 1000	4.337647026	48.1538745	31.78565715	31.6227766
1 to 10000	4.362904038	165.6021411	100.1853329	100

Table 2: Results of the effective permeability for the three methods explored in this study and the theoretical result, for the different permeability ratios, on a 16x16 grid

64x64 Grid				
Permeability Ratio	Harmonic Mean	Geometric Mean	Finite Analytic Method	Theoretical
1 to 2	1.41344831	1.41282759	1.414458961	1.414213562
1 to 10	3.02514645	3.196671831	3.171159618	3.16227766
1 to 100	5.452266091	11.86982237	10.07480591	10
1 to 1000	6.05053855	46.1550946	31.76760897	31.6227766
1 to 10000	6.119158676	162.8353881	100.1782617	100

Table 3: Results of the effective permeability for the three methods explored in this study and the theoretical result, for the different permeability ratios, on a 64x64 grid

The results from the tables above show that the harmonic mean results give good approximations for low permeability ratios. However, when the ratio becomes large, such as 1 to 100, the methods stars highly under estimates the effective permeability. Rather surprisingly, the results from the geometric mean method gave very accurate approximation for small grid refinement. This is probably due to the fact that the nature of the geometric averaging method solves for the effective permeability directly, similar to that of the theoretical result, when only two sets of permeabilities are used (see section 3.5). Though, for a more general heterogeneous reservoir that consists of more variations, the results will most likely be poorer (which is explored further below). However, the error increased significantly when the refinement increased. This can be due to the fact the geometric averaging scheme was used across the entire domain, as opposed to only where there is high heterogeneity (where $k_1 k_3 \neq k_2 k_4$. Though, the effective permeability approximation is much more accurate than the harmonic mean approach, especially for higher permeability ratios. The results from the finite analytic method were very consistent throughout the different permeability ratios and different grid refinements. Only ever so slightly over estimating the effective permeability, and the error grew even smaller with increased refinement. This ascertains the method as being superior to the others, and more importantly, superior to the method used most commonly in the industry today, the harmonic averaging method.

The effective permeability approximation from the finite analytic method was remarkably accurate compared to the other methods and compares well with the theoretical results. Even for very small refinements, the method was able to have a very small error margin. This can be seen in more detail in the table below. It demonstrates than the margins are very low, verifying the method established by Liu and Wang in their 2013 article (Liu & Wang, 2013).

Finite Analytic Method Error Margin			
Permeability Ratio	4x4 [%]	16x16 [%]	64x64 [%]
1 to 2	4.415574301	0.232470886	0.017349265
1 to 10	6.882813609	0.864864247	0.280085494
1 to 100	4.081277412	1.06353811	0.742504771
1 to 1000	1.595337753	0.512434112	0.455912084
1 to 10000	0.540793397	0.184990013	0.177944522

Table 4: A table exploring the error margins for the finite analytic method for the different grid refinements and permeability ratios, where all the numbers are presented as percentages

To visually explore the pressure gradient sharpening from the increasing permeability ratios, images of the pressure field were developed. The cases that were considered were at a grid refinement of 64x64 and permeability ratios of 1:10, 1:100, and 1:1000. The grid refinement chosen was due to the fact that a high resolution was desired to present the pressure gradient more clearly.

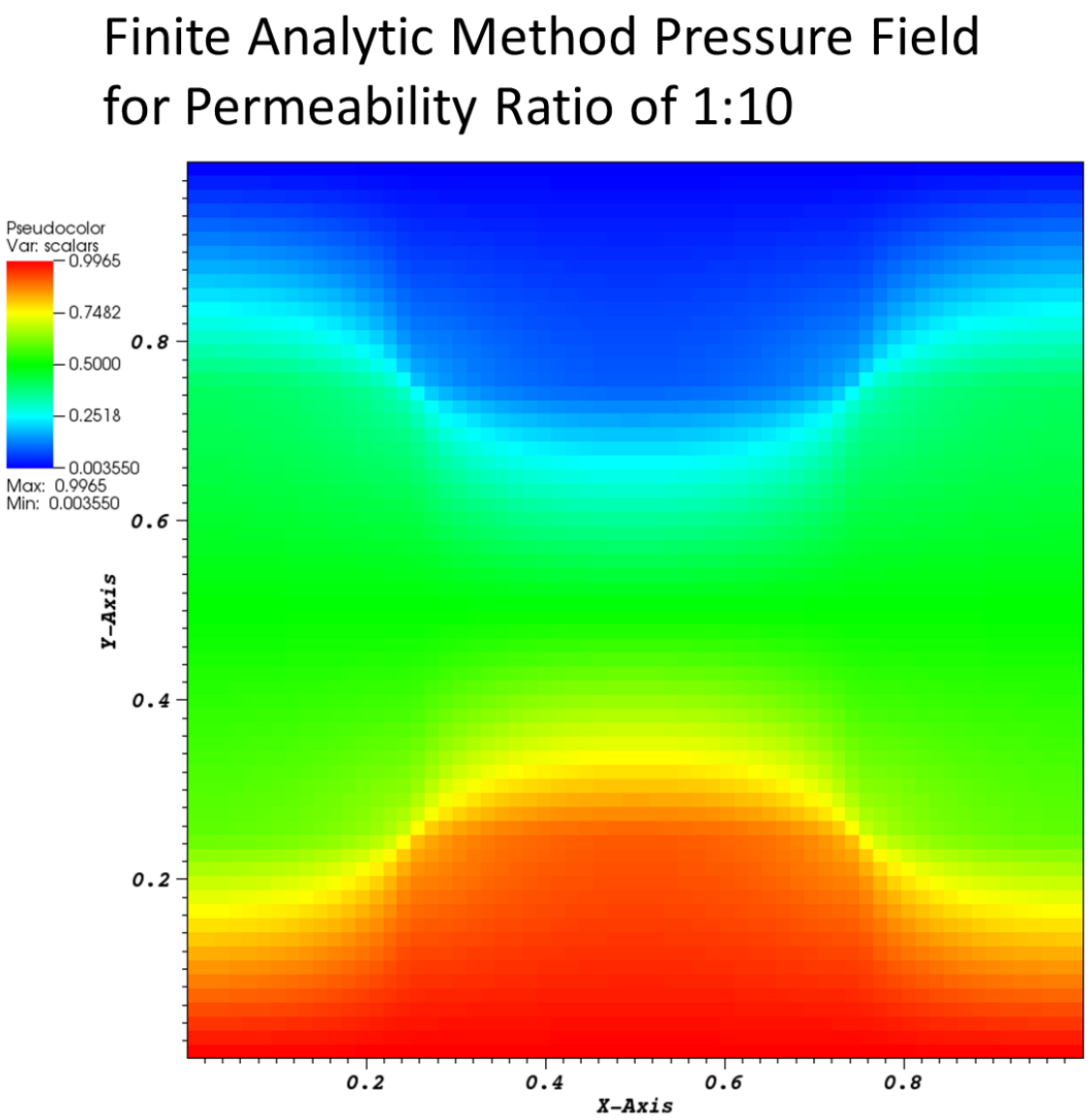

Figure 6.2: An illustration of the pressure field from the finite analytic method for the permeability ratio of 1:10

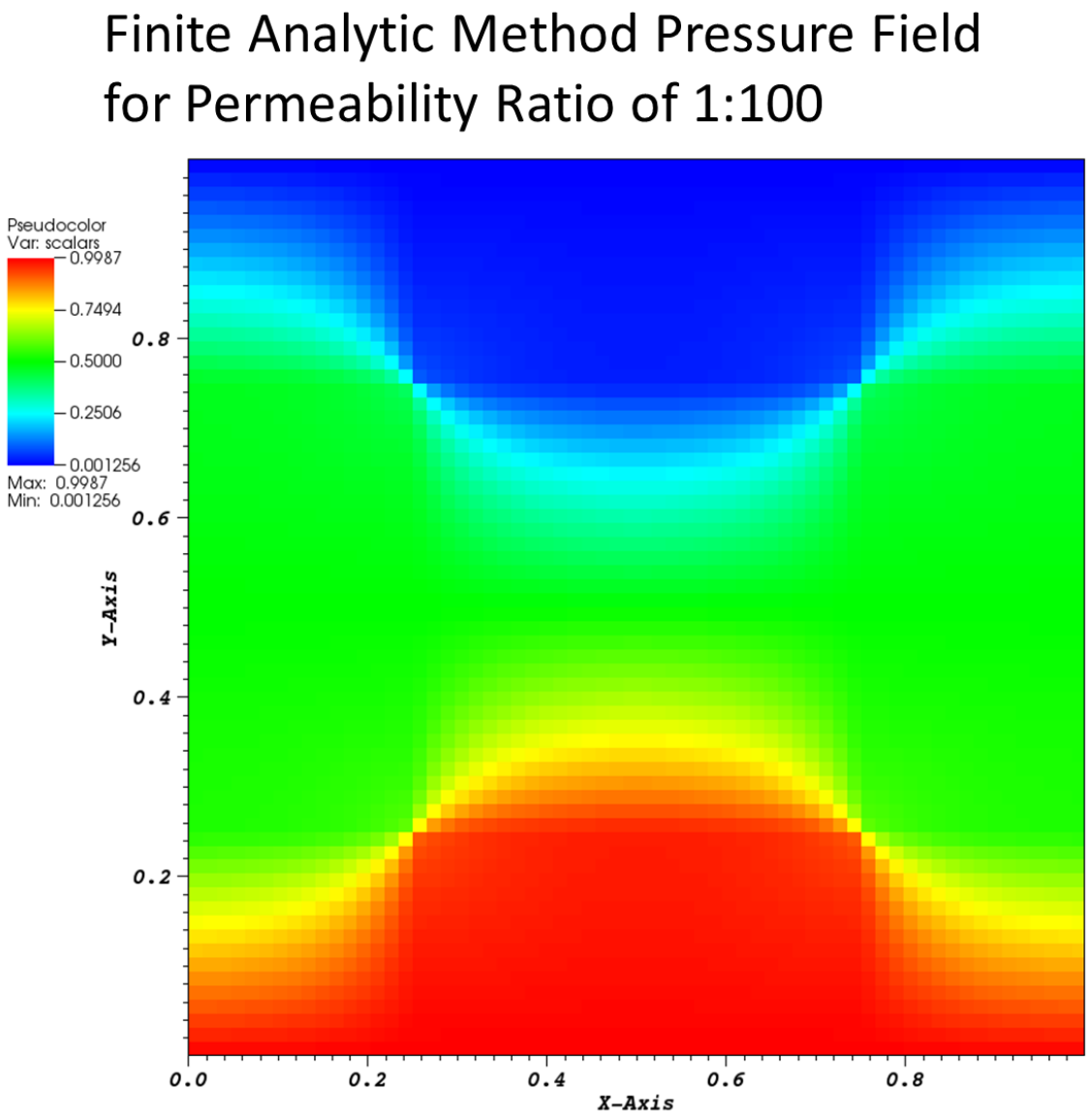

Figure 6.3: An illustration of the pressure field from the finite analytic method for the permeability ratio of 1:100

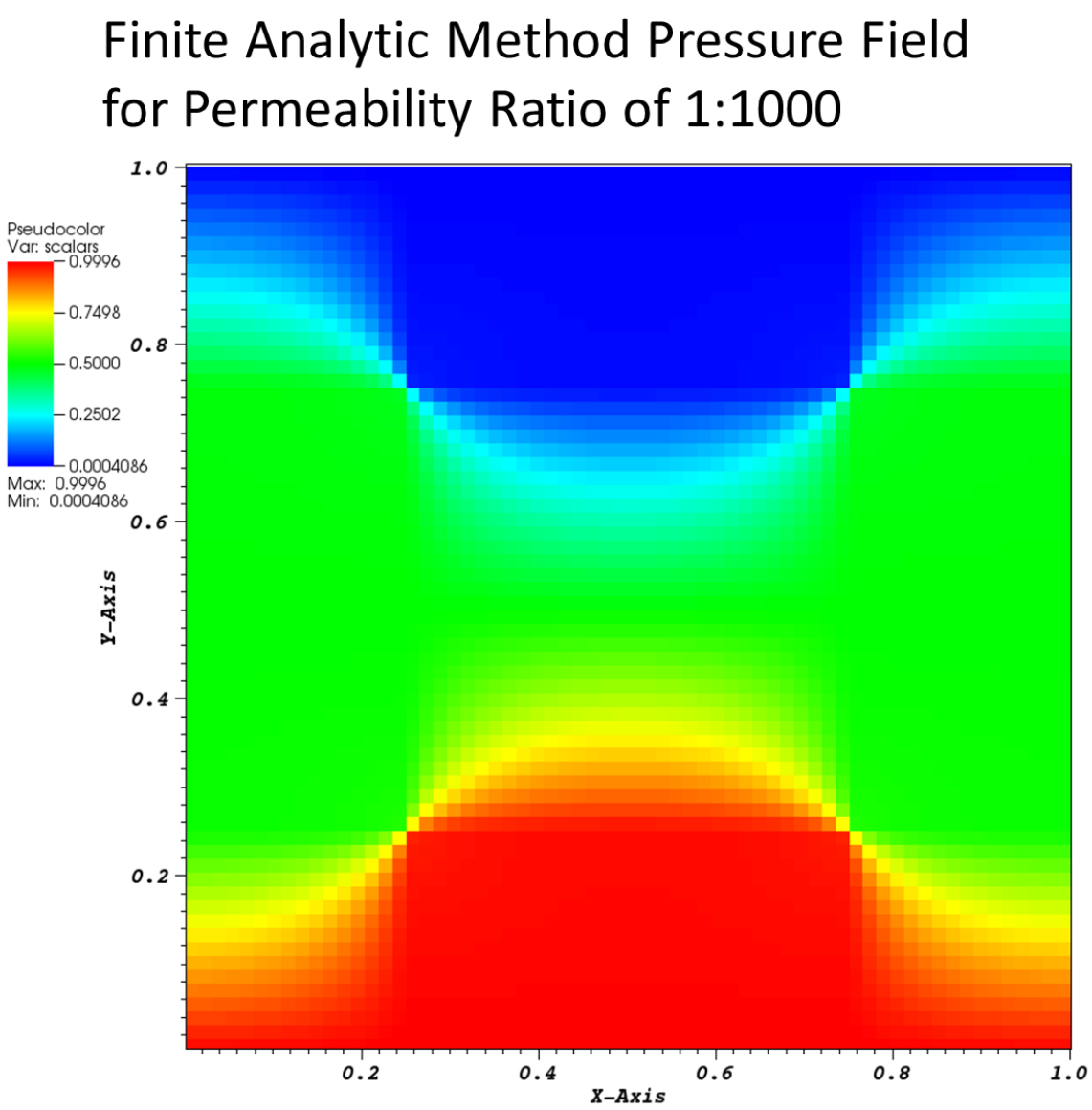

Figure 6.4: An illustration of the pressure field from the finite analytic method for the permeability ratio of 1:1000

The figures above shows how the pressure gradient becomes sharper as the permeability ratio increases. Higher contrast and harsher edges become more prominent, and the pressure field becomes more constrained. This is the expected behavior for these cases, and the finite analytic method reveals this accurately and gratifyingly.

The article (Liu & Wang, 2013) also explored real reservoir data on a 5x4 grid (see Figure 6.5) and compared it to both geometric mean and harmonic mean. There are no theoretical results for that data, however, the finite analytic method proves to not only be more accurate, but also achieve the results with fewer grid refinement, as seen in Figure 6.6. Results could also have been obtained using the program developed during this study, however, due to both time constraints and the fact that the implementations give identical results, a reference to the article seemed sufficient.

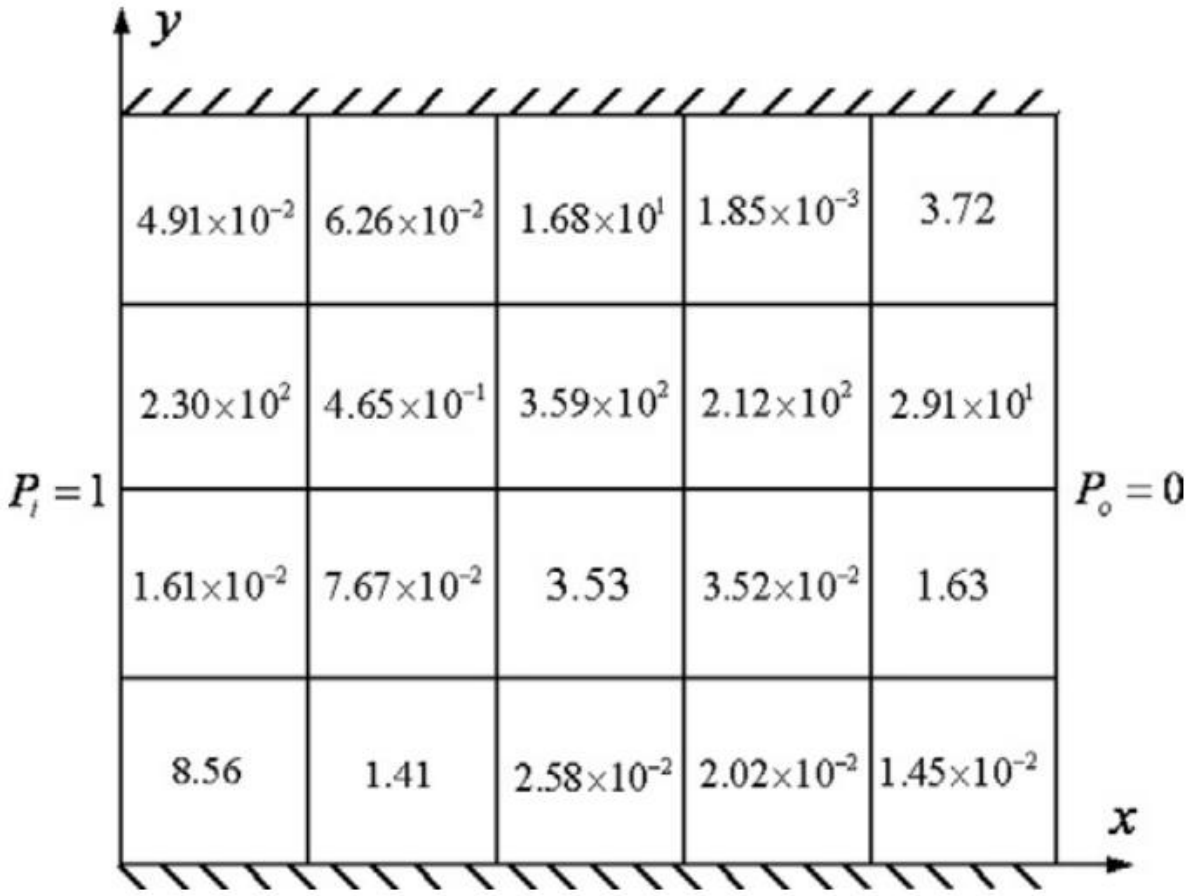

Figure 6.5: The spatial permeability distribution in a 5x4 grid (Liu & Wang, 2013)

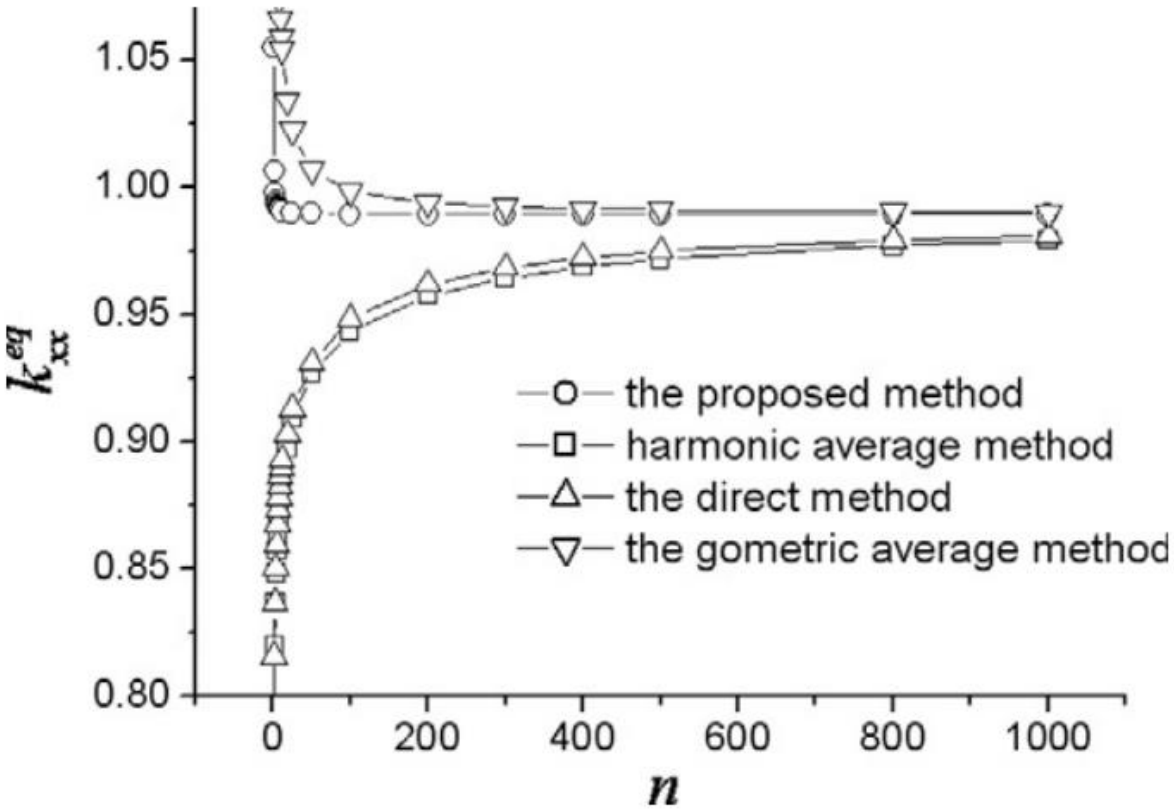

Figure 6.6: A plot of the equivalent permeability as a function of the grid refinement "n", comparing different numerical schemes (Liu & Wang, 2013)

6.3. Anisotropic Results

Due to the complications with the implementation of the finite analytic method, explained in the

Finite Analytic Method section in more detail, little time was left to fully apply and administer the anisotropic implementation. The boundary flux calculations are missing from the method for general anisotropic permeability tensors. The approach that was going to be used was MPFA, which has been research and validated extensively (Aavatsmark, Reiso, Reme, & Teiland, 2001). However, due to the MPFA method not being applied for the boundary calculations, as well as the parts of the domain with homogeneous permeability, in the approach, and time becoming highly constrained, it was decided that at the very least an isotropic test should be done. The MPFA method is somewhat difficult to implement and requires significant time to apply and test. Thus, the novel anisotropic extension was tested on isotropic data to validate it and prove that it can be used and performed in such a manner.

The results from this method proved to be identical to that of the isotropic implementation of the finite analytic method. Different test cases were performed and, on all accounts, the outcomes were indistinguishable. As such, one can conclude that the novel anisotropic extension is a valid approach and can be expanded upon to solve for anisotropic permeability test cases, though obviously more comprehensive testing is required for general anisotropic permeability tensors. Code extracts from both the isotropic and anisotropic implementations are included in the appendix for validation.

7. Conclusion

The results from the finite analytic method proved to be comparable with that was deduced in the article (Liu & Wang, 2013). The power law behavior was very evident and the effective permeability was calculated within a 7% error margin, where the harmonic averaging method gave results with over 75% error margin. Thus, the finite analytic method proved to be much more accurate and precise in its calculation.

The anisotropic form of the finite analytic method that was developed in this study was able to give identical results to that of the standard isotropic method. This showed that the approach was valid and could replicate what was discovered earlier. The novel anisotropic method differs from that which the authors of the finite analytic method recently designed. However, it was both precise enough and robust enough to fully resolve anisotropic permeability.

The ambition of this study was to create a novel approach to anisotropic permeability data, and to possibly publish the results. However, due to time constraints, anisotropic data could not be fully tested, and as such were not included in this study. Having to both redevelop and correct the method described for the finite analytic method (Liu & Wang, 2013), as well as both program it and develop a suitable foundation to test it. Additionally, deriving the novel anisotropic form. All of this required more time than originally anticipated. Though, the only missing part to implement is the MPFA discretization to account for the boundaries (Aavatsmark, Reiso, Reme, & Teiland, 2001). Given that the anisotropic method presented and discussed in this study is novel, an attempt for publication will be made as soon as the above mentioned piece is in place and a test on anisotropic permeability data can be made.

Nevertheless, in conclusion, this study was successful in replicating and verifying the finite analytic method. This study was also able to develop a unique method for solving anisotropic permeability problems using the core concepts of the finite analytic method. The results were positive, proving the power law behavior is prominent around the grid nodes where high permeability contrast is present. And as such was able to show that the use of harmonic averaging, which is an industry standard, greatly underestimates the effective permeability.

8. Future Work

The pressure equations for the isotropic finite analytic approach, explored in (Liu & Wang, 2013), were corrected and validated. It proved to be much more stable, reliable, and accurate when compared to industry standard methods, such as geometric averaging and, more importantly, harmonic averaging. The work on the novel anisotropic extension developed in this study, which is completely independent from the work done in (Liu & Wang, 2016), proved to be valid when tested on isotropic data.

However, due to time constraints, the anisotropic method could not be fully implemented. The calculations that occur at the boundary and in homogeneous zones should use the MPFA method (Aavatsmark, Reiso, Reme, & Teiland, 2001). Yet, this was not and could not be implemented in the allotted time. Therefore this should be the first and most important addition to be made in the future.

Additionally, the approach is only implemented for single phase flow. The approach can easily be expanded to account for multi phase flow. This is a key addition to be made and would bring even greater value to the approach and method discussed in this study.

Lastly, the other vital component that is currently missing is the fact that the present implementation is only applied in a two dimensional bases. Given that today practically all data and reservoir model are in three dimensions, this would have to be implemented to wholly be comparable and to improve current industry standard simulators and methods. This addition can be done in multiple ways, however, the simplest approach would be to apply the same two dimensional approach that is present today, and apply it to the remaining two planes in the three dimensional environment. A sample by sample calculation should be performed, where the calculations are done on the three planes (x, y), (z, x), and (z, y).

These are a few of the accompaniments that should be added to this study in the future. There several other additions that can be made, such as expanding the method to solve for unstructured grids. However, the focus should be on these first as they are essential components for reservoir simulators given today's standards. These, coupled with the accuracy and precision gained from using the method examined and developed in this study, should result in a very robust and enhanced reservoir simulator.

References

Aavatsmark, I., Barkve, T., & Mannseth, T. (1998). Control-Volume Discretization Methods for 3D Quadrilateral Grids in Inhomogeneous Anisotropic Reservoirs. *SPE Journal 3.02*, 146-154.

Aavatsmark, I., Reiso, E., Reme, H., & Teiland, R. (2001). MPFA for Faults and Local Refinements in 3D Quadrilateral Grids with Application to Field Simulations. *SPE Reservoir Simulation Symposium.* Society of Petroleum Engineers.

Anton, H. (2010). *Elementary Linear Algebra.* John Wiley & Sons.

Brown, R. G., Gleason, A. M., & Brown, M. A. (1992). *Advanced Mathematics: Precalculus with Discrete Mathematics and Data Analysis.* Houghton Mifflin Company.

Cao, H. (2002). *Development of Techniques for Genral Purpose Simulators.* PhD, Stanford University.

Carlson, M. R. (2006). *Practical Reservoir Simulation: Using, Assessing, and Developing Results.* PennWell Corporation.

Chen, Z., Huan, G., & Ma, Y. (2006). *Computational Methods for Multiphase Flows in Porous Media.* Society for Industrial and Applied Mathematics.

Cheng, Y. (2012). *Reservoir Simulation.* Dept. Petroleum & Natural Gas Engineering, West Virginia University.

Chou, Y. L. (1963). *Statistical Analysis.* Holt, Rinehart & Winston.

Cordazzo, J., Hurtado, F. S., Maliska, C. R., & Da Silva, A. F. (2003). *Numerical Techniques for Solving Partial Differential Equations in Heterogeneous Media.* Fortaleza, Ceará, Brazil: The 24th Iberian Latin-American Congress on Computational Methods in Engineering.

Crawley, M. J. (2005). *Statistics: An Introduction Using R.* John Wiley & Sons Ltd.

Darcy, H. (1856). *The Public Fountains of the City of Dijon.* Dalmont, Paris 647.

Dawe, R. A., & Grattoni, C. A. (2008). Experimental Displacement Patterns in a 2 x 2 Quadrant Block with Permeability and Wettability Heterogeneities: Problems for Numerical Modelling. *Transport in Porous Media 71. 1*, 5-22.

Gautier, Y., & Noetinger, B. (1997). Preferential Flow-Paths Detection for Hetergeneous Reservoirs Using a New Renormalization Technique. *Transport in Porous Media 26*, 1-23.

Islam, M. R., Hossain, M. E., Mousavizadegan, S. H., Mustafiz, S., & Abou-Kassam, J. H. (2016). *Advanced Petroleum Reservoir Simulation: Towards Developing Reservoir Emulators.* Scrivener Publishing LLC., John Wiley & Sons Inc.

Katz, V. J. (1979). The History of Stoke's Theorem. *Methatics Magazine* (pp. 146-156). Mathematical Association of America 52.

Keller, J. B. (1964). A Theorem on the Conductivity of a Composite Medium. *Journal of Mathematical Physics 4*, 548-549.

Lake, L. W. (1988). The Origins of Anisotropy. *Journal of Petroleum Technology*, 395-396.

Lang, S. (1985). *Complex Analysis.* Springer Science & Business Media.

Liu, Z. F., & Wang, X. H. (2013). Finite Analytic Numerical Method for Two Dimensional Fluid Flow in Heterogeneous Porous Media. *Journal of Computational Physics*, 286-301.

Liu, Z. F., & Wang, X. H. (2016). Finite Analytic Method for 2D Fluid Flow in Porous Media with Permeability in Tensor Form. *Journal of Porous Media*, 539-555.

Nacul, E. C. (1991). *Use of Domain Decomposition and Local Grid Refinement in Reservoir Simulation.* PhD Stanford University.

Nelson, R. A. (2001). *Geologic Analysis of Naturally Fractured Reservoirs.* Gulf Professional Publishing (Elsevier).

Odeh, A. S. (1982). An Overview of Mathematical Modeling of the Behavior of Hydrocarbon Reservoirs. (pp. 263-273). SIAM Review 24.3.

Oilfield-Glossary. (2017, March 19). Retrieved from www.slb.com: http://www.glossary.oilfield.slb.com/en/Terms/a/absolute_permeability.aspx

Oilfield-Glossary. (2017, March 19). Retrieved from www.slb.com: http://www.glossary.oilfield.slb.com/Terms/e/effective_permeability.aspx

Papakonstantinou, J. (2007). The Histrorical Development of the Secant Method in 1-D. *Mathematical Assosiation of America.* San Jose, California.

Peaceman, D. W. (1977). *Fundamentals of Numerical Reservoir Simulation.* Elsevier.

Peaceman, D. W. (1983). Interpretation of Well-Block Pressures in Numerical Reservoir Simulation with Nonsquare Grid Blocks and Anisotropic Permeability. *Society of Petroleum Engineers Journal 23.03*, 531-543.

Peaceman, D. W. (1996). Calcualtion of Transmissibilities of Gridblocks Define by Arbitrary Corner Point Geometry. *SPE*.

Pedrosa, O. A., & Aziz, K. (1985). Use of Hybrid Grid in Reservoir Simulation. *SPE Symposium on Reservoir Simulation 8.* Dallas, TX.

Pettersen, Ø. (2006). *Basics of Reservoir Simulation with the Eclipse Reservoir Simulator.* Dept. of Mathematics, University of Bergen.

References

Ponting, D. K. (1989). Corner Point Geometry in Reservoir Simulation. *European Conference on Mathematics of Oil Recovery 1*, (pp. 45-65). Cambridge, England.

Prévost, M., Lepage, F., Durlofsky, L. J., & Mallet, J. L. (2005). Unstructured 3D Gridding and Upscaling for Coarse Modelling of Geometrically Complex Reservoirs. *Petroleum Geoscience 11(4)*, 339-345.

Rao, S. S. (2011). *The Finite Element Method in Engineering.* Elsevier Inc.

Smith, G. D. (1985). *Numerical Solution of Partial Differential Equations: Finite Difference Methods.* Clarendon Press.

Soleng, H. H., & Holden, L. (1998). *Gridding for Petroleum Reservoir Simulation.* Norwegian Computing Center.

Wang, H. F., & Anderson, P. A. (1982). *Introduction to Groundwater Modeling: Finite Difference and Finite Element Methods.* Acedemic Press Inc.

Yaneer, B. (2017, April 4). *Concepts: Power Law*. Retrieved from New England Complex Systems Institute: http://www.necsi.edu/guide/concepts/powerlaw.html

Yeo, I. W., & Zimmerman, R. W. (2001). Accuracy of the Renormalization Method for Computing Effective Conductivities of Heterogeneous Media. *Transport in Porous Media 45. 1*, 129-138.

Appendix

This section is divided into four separate parts; appendix A, B, C, and D. Appendix A consists of scanned copies of some of the calculations, which were done by hand. These are more in depth mathematically, and contain a few intermediary expressions that were not included in this study, due to size and relevance. Appendix B consists of the code that was programmed for the isotropic finite analytic method. Appendix C consists of the code that was programmed for the anisotropic extension for the finite analytic method. And lastly, appendix D consists of the setup routines that use the classes examined in appendix B and C. These setup routines are used to generate the linear equation system corresponding to the actual finite analytic method discretization.

Appendix A: Scanned Calculations by Hand of Finite Analytic Method

Below are some scanned copies of a more extensive look on the calculations involved in deriving the finite analytic method. These include some of the calculations, specifically the backward substitution process, which was too exhaustive to include in this study. Keep in mind that these were all done by hand and included little description of the processes, and therefore might be difficult to follow without having revised this study thoroughly.

Calculations related to the backward substitution process

$$B_1 = -\frac{k_4}{k_1}\cos\tfrac{\pi}{2}\alpha\left[A_3\sin\tfrac{\pi}{2}\alpha - B_3\cos\tfrac{\pi}{2}\right] + \frac{k_3}{k_1}\sin\tfrac{\pi}{2}\left[A_3\cos\tfrac{\pi}{2}\alpha + B_3\sin\tfrac{\pi}{2}\alpha\right]$$

$$= -\frac{k_4}{k_1}\cos\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha\left[A_2\sin\tfrac{\pi}{2}\alpha - B_2\cos\tfrac{\pi}{2}\alpha\right] + \frac{k_4k_2}{k_1k_3}\cos^2\tfrac{\pi}{2}\alpha\left[A_2\cos\tfrac{\pi}{2}\alpha + B_2\sin\tfrac{\pi}{2}\alpha\right]$$

$$- \frac{k_3}{k_1}\sin\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha\left[A_2\sin\tfrac{\pi}{2}\alpha - B_2\cos\tfrac{\pi}{2}\alpha\right] - \sin^2\tfrac{\pi}{2}\alpha\frac{k_2}{k_1}\left[A_2\cos\tfrac{\pi}{2}\alpha + B_2\sin\tfrac{\pi}{2}\alpha\right]$$

$$= -\frac{k_4}{k_1}\sin^2\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha\left[A_1\sin\tfrac{\pi}{2}\alpha + B_1\cos\tfrac{\pi}{2}\alpha\right] + \frac{k_4}{k_2}\cos^2\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha\left[A_1\cos\tfrac{\pi}{2}\alpha - B_1\sin\tfrac{\pi}{2}\alpha\right]$$

$$+ \frac{k_4k_2}{k_1k_3}\cos^3\tfrac{\pi}{2}\alpha\left[A_1\sin\tfrac{\pi}{2}\alpha + B_1\cos\tfrac{\pi}{2}\alpha\right] + \frac{k_4}{k_3}\cos^2\tfrac{\pi}{2}\alpha\sin\alpha\left[A_1\cos\tfrac{\pi}{2}\alpha - B_1\sin\tfrac{\pi}{2}\alpha\right]$$

$$- \frac{k_3}{k_1}\sin^2\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha\left[A_1\sin\tfrac{\pi}{2}\alpha + B_1\cos\tfrac{\pi}{2}\alpha\right] + \frac{k_3}{k_2}\cos^2\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha\left[A_1\cos\tfrac{\pi}{2}\alpha - B_1\sin\tfrac{\pi}{2}\alpha\right]$$

$$- \frac{k_2}{k_1}\sin^2\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha\left[A_1\sin\tfrac{1}{2}\alpha + B_1\cos\tfrac{1}{2}\alpha\right] - \sin^3\tfrac{\pi}{2}\alpha\left[A_1\cos\tfrac{\pi}{2}\alpha - B_1\sin\tfrac{\pi}{2}\alpha\right]$$

$$0 = A_1\left[-\frac{k_4}{k_1}\sin^3\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha + \frac{k_4}{k_2}\cos^3\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha + \frac{k_4k_2}{k_1k_3}\cos^2\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha + \frac{k_4}{k_3}\cos^3\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha\right.$$

$$\left. - \frac{k_3}{k_1}\sin^3\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha + \frac{k_3}{k_2}\cos^3\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha - \frac{k_2}{k_1}\sin^3\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha - \sin^3\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha\right]$$

$$+ B_1\left[-\frac{k_4}{k_1}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_4}{k_2}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \frac{k_4k_2}{k_1k_3}\cos^4\tfrac{\pi}{2}\alpha + \frac{k_4}{k_3}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - 1\right.$$

$$\left. - \frac{k_3}{k_1}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_3}{k_2}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_2}{k_1}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \sin^4\tfrac{\pi}{2}\alpha\right]$$

~~$$0 = A_1\left[\frac{1}{8}\left(2\sin(\pi\alpha) - \sin(2\pi\alpha)\right)\right]\left[-1 - \frac{k_2}{k_1} - \frac{k_3}{k_1} - \frac{k_4}{k_1}\right]$$~~

$$0 = A_1\left\{-\frac{1}{8}\left(2\sin(\pi\alpha) - \sin(2\pi\alpha)\right)\left[1 + \frac{k_2}{k_1} + \frac{k_3}{k_1} + \frac{k_4}{k_1}\right] + \frac{1}{8}\left(2\sin(\pi\alpha) + \sin(2\pi\alpha)\right)\left[\frac{k_4k_2}{k_1k_3} + \frac{k_4}{k_3} + \frac{k_4}{k_2} + \frac{k_3}{k_2}\right]\right\}$$

$$+ B_1\left\{\sin^4\tfrac{\pi}{2}\alpha + \frac{k_4k_2}{k_1k_3}\cos^4\tfrac{\pi}{2}\alpha - \frac{1}{4}\sin^2(\pi\alpha)\left[\frac{k_2}{k_1} + \frac{k_3}{k_1} + \frac{k_4}{k_1} + \frac{k_3}{k_2} + \frac{k_4}{k_2} + \frac{k_4}{k}\right] - 1\right\}$$

Calculations related to the backward substitution process

$$A_1 = \sin\tfrac{\pi}{2}\alpha\left[A_3\sin\tfrac{\pi}{2}\alpha - B_3\cos\tfrac{\pi}{2}\alpha\right] + \cos\tfrac{\pi}{2}\alpha\,\frac{k_3}{k_4}\left[A_3\cos\tfrac{\pi}{2}\alpha + B_3\sin\tfrac{\pi}{2}\alpha\right]$$

$$= \sin^2\tfrac{\pi}{2}\alpha\left[A_2\sin\tfrac{\pi}{2}\alpha - B_2\cos\tfrac{\pi}{2}\alpha\right] - \sin\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha\,\frac{k_2}{k_3}\left[A_2\cos\tfrac{\pi}{2}\alpha + B_2\sin\tfrac{\pi}{2}\alpha\right]$$

$$- \cos^2\tfrac{\pi}{2}\alpha\,\frac{k_3}{k_4}\left[A_2\sin\tfrac{\pi}{2}\alpha - B_2\cos\tfrac{\pi}{2}\alpha\right] - \cos\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha\,\frac{k_2}{k_4}\left[A_2\cos\tfrac{\pi}{2}\alpha + B_2\sin\tfrac{\pi}{2}\alpha\right]$$

$$= \sin^3\tfrac{\pi}{2}\alpha\left[A_1\sin\tfrac{\pi}{2}\alpha + B_1\cos\tfrac{\pi}{2}\alpha\right] - \sin^2\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha\,\frac{k_1}{k_2}\left[A_1\cos\tfrac{\pi}{2}\alpha - B_1\sin\tfrac{\pi}{2}\alpha\right]$$

$$- \sin\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha\,\frac{k_2}{k_3}\left[A_1\sin\tfrac{\pi}{2}\alpha + B_1\cos\tfrac{\pi}{2}\alpha\right] + \sin^2\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha\,\frac{k_1}{k_3}\left[A_1\cos\tfrac{\pi}{2}\alpha - B_1\sin\tfrac{\pi}{2}\alpha\right]$$

$$- \cos^2\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha\,\frac{k_3}{k_4}\left[A_1\sin\tfrac{\pi}{2}\alpha + B_1\cos\tfrac{\pi}{2}\alpha\right] + \cos^3\tfrac{\pi}{2}\alpha\,\frac{k_3k_1}{k_4k_2}\left[A_1\cos\tfrac{\pi}{2}\alpha - B_1\sin\tfrac{\pi}{2}\alpha\right]$$

$$- \cos^2\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha\,\frac{k_2}{k_4}\left[A_1\sin\tfrac{\pi}{2}\alpha + B_1\cos\tfrac{\pi}{2}\alpha\right] + \cos\tfrac{\pi}{2}\alpha\sin^2\tfrac{\pi}{2}\alpha\,\frac{k_1}{k_4}\left[A_1\cos\tfrac{\pi}{2}\alpha - B_1\sin\tfrac{\pi}{2}\alpha\right]$$

$$0 = A_1\left[\sin^4\tfrac{\pi}{2}\alpha - \sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha\,\frac{k_1}{k_2} - \sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha\,\frac{k_2}{k_3} - \sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha\,\frac{k_1}{k_3} - 1\right.$$
$$\left. - \sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha\,\frac{k_3}{k_4} + \cos^4\tfrac{\pi}{2}\alpha\,\frac{k_3k_1}{k_4k_2} - \sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha\,\frac{k_2}{k_4} - \sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha\,\frac{k_1}{k_4}\right]$$
$$+ B_1\left[\sin^3\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha + \sin^3\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha\,\frac{k_1}{k_2} - \cos^3\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha\,\frac{k_2}{k_3} + \sin^3\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha\,\frac{k_1}{k_3}\right.$$
$$\left. - \cos^3\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha\,\frac{k_3}{k_4} - \cos^3\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha\,\frac{k_1k_3}{k_2k_4} - \cos^3\tfrac{\pi}{2}\alpha\sin\tfrac{\pi}{2}\alpha\,\frac{k_2}{k_4} + \sin^3\tfrac{\pi}{2}\alpha\cos\tfrac{\pi}{2}\alpha\,\frac{k_1}{k_4}\right]$$

$$0 = A_1\left\{\sin^4\tfrac{\pi}{2}\alpha + \cos^4\tfrac{\pi}{2}\alpha\,\frac{k_1k_3}{k_2k_4} - \frac{1}{4}\sin^2(\pi\alpha)\left[\frac{k_1}{k_2} + \frac{k_1}{k_3} + \frac{k_1}{k_4} + \frac{k_2}{k_3} + \frac{k_2}{k_4} + \frac{k_3}{k_4}\right] - 1\right\}$$
$$+ B_1\left\{\left[\frac{1}{8}\left(2\sin(\pi\alpha) - \sin(2\pi\alpha)\right)\left[1 + \frac{k_1}{k_2} + \frac{k_1}{k_3} + \frac{k_1}{k_4}\right] + \frac{1}{8}\left(2\sin(\pi\alpha) + \sin(2\pi\alpha)\right)\left[\frac{k_1k_3}{k_2k_4} + \frac{k_2}{k_3} + \frac{k_2}{k_4} + \frac{k_3}{k_4}\right]\right.\right.$$

Ligning for A_1 $\longrightarrow$ $\left(\dfrac{k_1k_3}{k_2k_4} + \dfrac{k_2}{k_3} + \dfrac{k_2}{k_4} + \dfrac{k_3}{k_4}\right)$

Computing the determinant

$$
\begin{aligned}
0 = \Big\{ & \sin^8\tfrac{\pi}{2}\alpha - \frac{k_2}{k_1}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_3}{k_2}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_3}{k_1}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \sin^4\tfrac{\pi}{2}\alpha \\
& - \frac{k_4}{k_3}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_4}{k_2}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_4}{k_1}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \frac{k_4k_2}{k_3k_1}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha \\
& - \frac{k_1}{k_2}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1k_3}{k_2^2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3}{k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1}{k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha \\
& + \frac{k_1k_4}{k_2k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1k_4}{k_2^2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_4}{k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha - \frac{k_4}{k_3}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha \\
& - \frac{k_2}{k_3}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \frac{k_2^2}{k_1k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2}{k_1}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2}{k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha \\
& + \frac{k_2k_4}{k_3^2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_4}{k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2k_4}{k_1k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha - \frac{k_4k_2^2}{k_1k_3^2}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha \\
& - \frac{k_1}{k_3}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \frac{k_2}{k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1}{k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1}{k_3}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha \\
& + \frac{k_1k_4}{k_3^2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1k_4}{k_2k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_4}{k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha - \frac{k_4k_2}{k_3^2}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha \\
& - \sin^4\tfrac{\pi}{2}\alpha + \frac{k_2}{k_1}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \frac{k_3}{k_2}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \frac{k_3}{k_1}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + 1 \\
& + \frac{k_4}{k_3}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \frac{k_4}{k_2}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \frac{k_4}{k_1}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_2k_4}{k_1k_3}\cos^4\tfrac{\pi}{2}\alpha \\
& - \frac{k_3}{k_4}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \frac{k_2k_3}{k_1k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3^2}{k_2k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3^2}{k_1k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3}{k_4}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha \\
& + \sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3}{k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3}{k_1}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha - \frac{k_2}{k_1}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha \\
& - \frac{k_2}{k_4}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \frac{k_2^2}{k_1k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3}{k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2k_3}{k_1k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2}{k_4}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha \\
& + \frac{k_2}{k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2}{k_1}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha - \frac{k_2^2}{k_1k_3}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha \\
& - \frac{k_1}{k_4}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha + \frac{k_2}{k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1k_3}{k_2k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3}{k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1}{k_4}\sin^2\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha \\
& + \frac{k_1}{k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1}{k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha - \frac{k_2}{k_3}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha \\
& + \frac{k_1k_3}{k_2k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \cos^8\tfrac{\pi}{2}\alpha - \frac{k_1k_3}{k_2k_4}\cos^4\tfrac{\pi}{2}\alpha - \sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha\left[\frac{k_3}{k_4} + \frac{k_1k_3^2}{k_4k_2^2} + \frac{k_3^2}{k_2k_4} + \frac{k_1}{k_2} + \frac{k_1k_3}{k_2^2} + \frac{k_3}{k_2}\right]\Big\}
\end{aligned}
$$

$$
\begin{aligned}
-\Big\{ & -\frac{k_4}{k_1}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_3}{k_1}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_2}{k_1}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha \\
& + \frac{k_4}{k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3k_4}{k_1k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_4}{k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3}{k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha \\
& - \frac{k_4}{k_2}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_3}{k_2}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_1}{k_2}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha \\
& + \frac{k_1k_4}{k_2^2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_4}{k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1k_4}{k_2k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1k_3}{k_2^2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha \\
& - \frac{k_4}{k_3}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_2}{k_3}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_1}{k_3}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha \\
& + \frac{k_1k_4}{k_2k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2k_4}{k_3^2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1k_4}{k_3^2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1}{k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha \\
& - \sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_3}{k_4}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_2}{k_4}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha - \frac{k_1}{k_4}\sin^6\tfrac{\pi}{2}\alpha\cos^2\tfrac{\pi}{2}\alpha \\
& + \frac{k_1}{k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2}{k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1}{k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1k_3}{k_2k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha \\
& + \frac{k_2k_4}{k_1k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2}{k_1}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2^2}{k_1k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2}{k_3}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha \\
& - \frac{k_4}{k_3}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \frac{k_4k_2}{k_1k_3}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \frac{k_2k_4}{k_3^2}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha \quad \left(\frac{k_3k_2}{k_1k_4}\right) \\
& + \frac{k_3}{k_1}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3^2}{k_1k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3}{k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha \\
& - \frac{k_3}{k_2}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \frac{k_2}{k_1}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \frac{k_3^2}{k_2k_4}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha \\
& + \frac{k_3}{k_2}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3^2}{k_2k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_3}{k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_1k_3}{k_2k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha \\
& - \frac{k_1k_3}{k_2^2}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \frac{k_1}{k_2}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \frac{k_1k_3^2}{k_4k_2^2}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha \\
& + \frac{k_2}{k_1}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2k_3}{k_1k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k}{k_1k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4\tfrac{\pi}{2}\alpha + \frac{k_2}{k_4}\sin^4\tfrac{\pi}{2}\alpha\cos^4{}_2\alpha \\
& - \sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \frac{k_2^2}{k_1k_3}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \frac{k_{e2}}{k_3}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha - \frac{k_3}{k_4}\sin^2\tfrac{\pi}{2}\alpha\cos^6\tfrac{\pi}{2}\alpha \Big\}
\end{aligned}
$$

$$= 0$$

$$\left[\sin^8\frac{\pi}{2}\alpha+\cos^8\frac{\pi}{2}\alpha+6\sin^4\frac{\pi}{2}\alpha\cos^4\frac{\pi}{2}\alpha-2\sin^4\frac{\pi}{2}\alpha+1\right.$$
$$\left.-\frac{k_1k_3}{k_2k_4}\cos^4\frac{\pi}{2}\alpha-\frac{k_2k_4}{k_1k_3}\cos^4\frac{\pi}{2}\alpha+\sin^2\frac{\pi}{2}\alpha\cos^2\frac{\pi}{2}\alpha\left(\frac{k_1}{k_2}+\frac{k_1}{k_3}+\frac{k_1}{k_4}+\frac{k_2}{k_3}+\frac{k_2}{k_4}+\frac{k_3}{k_4}\right)\right]$$
$$-\left[-4\sin^6\frac{\pi}{2}\alpha\cos^2\frac{\pi}{2}\alpha-4\sin^2\frac{\pi}{2}\alpha\cos^6\frac{\pi}{2}\alpha\right]=0$$

$$0=2-2\sin^4\left(\frac{\pi}{2}\alpha\right)-\left(\frac{k_1k_3}{k_2k_4}+\frac{k_2k_4}{k_1k_3}\right)\cos^4\left(\frac{\pi}{2}\alpha\right)+\left(\frac{k_1}{k_2}+\frac{k_1}{k_3}+\frac{k_1}{k_4}+\frac{k_2}{k_3}+\frac{k_2}{k_4}+\frac{k_3}{k_4}\right)\sin^2\left(\frac{\pi}{2}\alpha\right)\cos^2\left(\frac{\pi}{2}\alpha\right)$$

Because:

$$\sin^8\frac{\pi}{2}\alpha+\cos^8\frac{\pi}{2}\alpha+6\sin^4\frac{\pi}{2}\alpha\cos^4\frac{\pi}{2}\alpha+4\sin^6\frac{\pi}{2}\alpha\cos^2\frac{\pi}{2}\alpha+4\sin^2\frac{\pi}{2}\alpha\cos^6\frac{\pi}{2}\alpha=1$$

According to Matematica:

$$a=\frac{k_1k_3}{k_2k_4}+\frac{k_2k_4}{k_1k_3},\quad b=\frac{k_1}{k_2}+\frac{k_1}{k_3}+\frac{k_1}{k_4}+\frac{k_2}{k_3}+\frac{k_2}{k_4}+\frac{k_3}{k_4}$$

$$\alpha=\frac{\pm\cos^{-1}\left(\frac{6-a+b}{2+a+b}\right)+2\pi n}{\pi},\quad n\in\mathbb{Z},\quad a+b+2\neq 0$$

"n" should be zero.

Appendix B: Finite Analytic Method Code Implementation (Isotropic)

```
#include "FAM2DSGDiscritization.h"
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

FAM2DSGDiscritization::FAM2DSGDiscritization()
:	m_alpha_FAM(0.0),
	m_C_FAM(0.0),
	m_perm(),
	m_C(),
	m_k1(0.0),
	m_k2(0.0),
	m_k3(0.0),
	m_k4(0.0),
	m_Ftol(0.00000001),
	m_notHarmonicScheme(true)
{
}
FAM2DSGDiscritization::~FAM2DSGDiscritization()
{
}
void FAM2DSGDiscritization::setup(const CSnShape& C, const ArrayS<realS>& perm)
{

	m_C = C;
    m_perm = perm;
}
void FAM2DSGDiscritization::computeAlphaAndC(const Pair<int,int> Pqhat, const int ihat,
const bool atBoundary)
{
	int i1 = 0;
	int i2 = 0;
	int i3 = 0;
	int i4 = 0;

	//Must find control volume neighbores "Pqhat", use a CSnShape 'getNeighbors....'

	if (!atBoundary)
	{
		ArrayS<int> neigh;
		m_C.u_getNeighbours_RF(Pqhat, 2, neigh);// the '2' is hardcoded due to 2D
space, for different problem dimensions, use 'problemDimension'....?
		i1 = neigh[0];
		i2 = neigh[1];
		i3 = neigh[2];
		i4 = neigh[3];

		//Must first find the correct quadrant for ihat:
		double FTOLL = 0.000001;
		Pair<int, int> Pihat(2, ihat);
		RnPoint XI = m_C.u_getCoordinate_0u(Pihat);
```

```
                RnPoint XQ = m_C.u_getCoordinate_0u(Pqhat);
                double XI_x = XI.u_getElement_0u(0);
                double XI_y = XI.u_getElement_0u(1);
                double XQ_x = XQ.u_getElement_0u(0);
                double XQ_y = XQ.u_getElement_0u(1);

                int whichQuad = -1;
                if ((XI_x-XQ_x) > FTOLL && (XI_y-XQ_y) > FTOLL)
                {
                  //1 quadrant
                  whichQuad = 1;
                }
                else if ((XQ_x-XI_x) > FTOLL && (XI_y-XQ_y) > FTOLL)
                {
                  //2 quadrant
                  whichQuad = 2;
                }
                else if ((XQ_x-XI_x) > FTOLL && (XQ_y-XI_y) > FTOLL)
                {
                  //3 quadrant
                  whichQuad = 3;
                }
                else if ((XI_x-XQ_x) > FTOLL && (XQ_y-XI_y) > FTOLL)
                {
                  //4 quadrant
                  whichQuad = 4;
                }
                 else
                {
                  cout << "Something is terribly wrong!!! Program is stopped... " << endl;
exit(0);
                }

                if (whichQuad == 1)//ihat in first quadrant
                {
                       if (i1 == ihat)
                       {
                       i1 = neigh[0];
                       i2 = neigh[1];
                       i3 = neigh[2];
                       i4 = neigh[3];
                       }
                       else if (i2 == ihat)
                       {
                       i1 = neigh[1];
                       i2 = neigh[2];
                       i3 = neigh[3];
                       i4 = neigh[0];
                       }
                       else if (i3 == ihat)
                       {
                       i1 = neigh[2];
                       i2 = neigh[3];
                       i3 = neigh[0];
                       i4 = neigh[1];
                       }
                       else if (i4 == ihat)
                       {
```

```
        i1 = neigh[3];
        i2 = neigh[0];
        i3 = neigh[1];
        i4 = neigh[2];
        }
}
else if (whichQuad == 2)//ihat in second quadrant
{
        if (i1 == ihat)
        {
        i1 = neigh[3];
        i2 = neigh[0];
        i3 = neigh[1];
        i4 = neigh[2];
        }
        else if (i2 == ihat)
        {
        i1 = neigh[0];
        i2 = neigh[1];
        i3 = neigh[2];
        i4 = neigh[3];
        }
        else if (i3 == ihat)
        {
        i1 = neigh[1];
        i2 = neigh[2];
        i3 = neigh[3];
        i4 = neigh[0];
        }
        else if (i4 == ihat)
        {
        i1 = neigh[2];
        i2 = neigh[3];
        i3 = neigh[0];
        i4 = neigh[1];
        }
}
else if (whichQuad == 3)//ihat in third quadrant
{
        if (i1 == ihat)
        {
        i1 = neigh[2];
        i2 = neigh[3];
        i3 = neigh[0];
        i4 = neigh[1];
        }
        else if (i2 == ihat)
        {
        i1 = neigh[3];
        i2 = neigh[0];
        i3 = neigh[1];
        i4 = neigh[2];
        }
        else if (i3 == ihat)
        {
        i1 = neigh[0];
        i2 = neigh[1];
        i3 = neigh[2];
```

```
                i4 = neigh[3];
                }
                else if (i4 == ihat)
                {
                i1 = neigh[1];
                i2 = neigh[2];
                i3 = neigh[3];
                i4 = neigh[0];
                }
        }
        else if (whichQuad == 4)//ihat in fourth quadrant
        {
                if (i1 == ihat)
                {
                i1 = neigh[1];
                i2 = neigh[2];
                i3 = neigh[3];
                i4 = neigh[0];
                }
                else if (i2 == ihat)
                {
                i1 = neigh[2];
                i2 = neigh[3];
                i3 = neigh[0];
                i4 = neigh[1];
                }
                else if (i3 == ihat)
                {
                i1 = neigh[3];
                i2 = neigh[0];
                i3 = neigh[1];
                i4 = neigh[2];
                }
                else if (i4 == ihat)
                {
                i1 = neigh[0];
                i2 = neigh[1];
                i3 = neigh[2];
                i4 = neigh[3];
                }
        }

        m_k1 = m_perm[i1];
        m_k2 = m_perm[i2];
        m_k3 = m_perm[i3];
        m_k4 = m_perm[i4];

        m_notHarmonicScheme = (fabs(m_k1*m_k3 - m_k2*m_k4) > m_Ftol) ? true :
false;


        if (m_notHarmonicScheme)
        {
        double api = atan(1.0)*4;

        m_alpha_FAM = fabs((2/api)*atan(((m_k1*m_k3)-
(m_k2*m_k4))/(sqrt((m_k1+m_k2+m_k3+m_k4)*((m_k1*m_k2*m_k3)+(m_k1*m_k3*m_k4)+(m_k1*m_k2*m_
k4)+(m_k2*m_k3*m_k4))))));
```

```
		if (((m_k1*m_k3)-(m_k2*m_k4)) < 0.0)
			m_C_FAM = -
sqrt(((m_k4*m_k4)*(m_k1+m_k2+m_k3+m_k4))/((m_k1*m_k2*m_k3)+(m_k1*m_k3*m_k4)+(m_k1*m_k2*m_
k4)+(m_k2*m_k3*m_k4)));
		else
			m_C_FAM =
sqrt(((m_k4*m_k4)*(m_k1+m_k2+m_k3+m_k4))/((m_k1*m_k2*m_k3)+(m_k1*m_k3*m_k4)+(m_k1*m_k2*m_
k4)+(m_k2*m_k3*m_k4)));
		}
		else
		{
			m_alpha_FAM  = 0.0;
			m_C_FAM = 1.0;//check for all 'k' = 3
		}
	}
	else
	{
		//NOT NECCESARY!!!!
	}
}
realS FAM2DSGDiscritization::computeRHSContribution(const int ihat, const int khat, const
int qhat, const ArrayS<realS>& g)
{
	double ret = 0.0;
	Pair<int,int> PQ(0, qhat);
	Pair<int,int> PK(1, khat);
	Pair<int,int> PI(2, ihat);
	double delta_x = 0.0;
	double delta_y = 0.0;

	RnPoint XQ = m_C.u_getCoordinate_0u(PQ);
	double XQ_x = XQ.u_getElement_0u(0);
	double XQ_y = XQ.u_getElement_0u(1);
	RnPoint XI = m_C.u_getCoordinate_0u(PI);
	double XI_x = XI.u_getElement_0u(0);
	double XI_y = XI.u_getElement_0u(1);

	delta_x = fabs(XI_x - XQ_x);// Half actual delta_x in the grid!!!!
	delta_y = fabs(XI_y - XQ_y);// Half actual delta_y in the grid!!!!


	RnPoint normalVec = m_C.u_getNormalVector_0u(PI, PK);

	if (m_C.u_boundaryIndex_0u(PK))
	{
		////NOTE: In case of a Neumann (i.e. flux) B.C. we assume that "g" is a
vector of zero's.

		if ((fabs(normalVec.u_getElement_0u(1)) <= m_Ftol) ? (ret =
(delta_y/delta_x)*m_perm[ihat]*g[m_C.u_findBoundaryPosition_0u(PK)]) : (ret =
(delta_x/delta_y)*m_perm[ihat]*g[m_C.u_findBoundaryPosition_0u(PK)]));
	}

	return (-1.0)*ret;
}
```

```
realS FAM2DSGDiscritization::computeMatrixContribution(const int ihat, const int itilde,
const int khat, const int qhat, const ArrayS<realS>& alpha1, const ArrayS<realS>& alpha2,
const bool atBoundary)
{
	realS ret=0.0;
	double api = atan(1.0)*4;
	double delta_x = 0.0;
	double delta_y = 0.0;
	double theta = 0.0;
	double radius = 0.0;
	double gamma_x = 0.0;
	double gamma_y = 0.0;

	Pair<int,int> PQ(0, qhat);
    Pair<int,int> PI(2, ihat);
	Pair<int,int> PK(1, khat);

	RnPoint XQ = m_C.u_getCoordinate_0u(PQ);
	double XQ_x = XQ.u_getElement_0u(0);
	double XQ_y = XQ.u_getElement_0u(1);
	RnPoint XI = m_C.u_getCoordinate_0u(PI);
	double XI_x = XI.u_getElement_0u(0);
	double XI_y = XI.u_getElement_0u(1);
	RnPoint XK = m_C.u_getCoordinate_0u(PK);
	double XK_x = XK.u_getElement_0u(0);
	double XK_y = XK.u_getElement_0u(1);

	RnPoint v1 = XK;
	v1 -= XQ;
	RnPoint v2 = XI;
	v2 -= XQ;

	theta = acos(v1.u_dot_0u(v2)/(v1.u_norm2_0u()*v2.u_norm2_0u()));

	delta_x = fabs(XI_x - XQ_x);// Half actual delta_x in the grid!!!!
	delta_y = fabs(XI_y - XQ_y);// Half actual delta_y in the grid!!!!
	radius = sqrt((delta_x*delta_x)+(delta_y*delta_y));
	bool Geometric = false; //This should be turned true and “m_notHarmonicScheme =
false” if the geometric mean calcualtions are desired

		if (m_notHarmonicScheme)
		{
			double lambda_x = 0.0;
			double lambda_y = 0.0;
			double lambda_c = 0.0;
			double lambda_w = 0.0;
			double lambda_s = 0.0;
			double lambda_e = 0.0;
			double lambda_n = 0.0;
			double lambda_sw = 0.0;


lambda_c = (pow(radius, 1-m_alpha_FAM))*((cos((api/4)*(1-
m_alpha_FAM)))+(m_C_FAM*sin((api/4)*(1-m_alpha_FAM))));

lambda_w = (pow(radius, 1-m_alpha_FAM))*((cos((api/4)*(1-
m_alpha_FAM))*(sin(0.5*api*m_alpha_FAM)+(m_C_FAM*cos(0.5*api*m_alpha_FAM))))-
```

```
((m_k1/m_k2)*sin((api/4)*(1-m_alpha_FAM))*(cos(0.5*api*m_alpha_FAM)-
(m_C_FAM*sin(0.5*api*m_alpha_FAM)))));

lambda_sw = (pow(radius, 1-m_alpha_FAM))*((cos((api/4)*(1-
m_alpha_FAM))*((pow(sin(0.5*api*m_alpha_FAM),2))-
((m_k1/m_k2)*(pow(cos(0.5*api*m_alpha_FAM),2)))+(m_C_FAM*(1+(m_k1/m_k2))*cos(0.5*api*m_al
pha_FAM)*sin(0.5*api*m_alpha_FAM))))-(sin((api/4)*(1-
m_alpha_FAM))*((((m_k1/m_k3)+(m_k2/m_k3))*cos(0.5*api*m_alpha_FAM)*sin(0.5*api*m_alpha_FA
M))+((m_k2/m_k3)*m_C_FAM*(pow(cos(0.5*api*m_alpha_FAM),2)))-
((m_k1/m_k3)*m_C_FAM*(pow(sin(0.5*api*m_alpha_FAM),2))))));

lambda_s = (pow(radius, 1-m_alpha_FAM))*((cos((api/4)*(1-
m_alpha_FAM))*((pow(sin(0.5*api*m_alpha_FAM),3))-
(m_C_FAM*(m_k2/m_k3)*(pow(cos(0.5*api*m_alpha_FAM),3)))+(m_C_FAM*(1+(m_k1/m_k2)+(m_k1/m_k
3))*cos(0.5*api*m_alpha_FAM)*(pow(sin(0.5*api*m_alpha_FAM),2)))-
(((m_k1/m_k2)+(m_k1/m_k3)+(m_k2/m_k3))*sin(0.5*api*m_alpha_FAM)*(pow(cos(0.5*api*m_alpha_
FAM),2)))))+(sin((api/4)*(1-
m_alpha_FAM))*((m_C_FAM*(m_k1/m_k4)*(pow(sin(0.5*api*m_alpha_FAM),3)))+(((m_k1*m_k3)/(m_k
2*m_k4))*(pow(cos(0.5*api*m_alpha_FAM),3)))-
(((m_k1/m_k4)+(m_k2/m_k4)+(m_k3/m_k4))*cos(0.5*api*m_alpha_FAM)*(pow(sin(0.5*api*m_alpha_
FAM),2)))-
(m_C_FAM*(((m_k1*m_k3)/(m_k2*m_k4))+(m_k2/m_k4)+(m_k3/m_k4))*sin(0.5*api*m_alpha_FAM)*(po
w(cos(0.5*api*m_alpha_FAM),2))))));

                    if (XI_x > XQ_x && XI_y > XQ_y)
                    {

                            if (XK_x > XQ_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
                            {
                                    lambda_y = (-m_k1*m_C_FAM)*(pow(delta_x, 1-
m_alpha_FAM));

                                    gamma_y = (lambda_y)/(lambda_s - lambda_c);
                            }
                            else if (XK_y > XQ_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                            {
                                    lambda_x = (-m_k1*((cos(0.5*api*m_alpha_FAM))-
(m_C_FAM*sin(0.5*api*m_alpha_FAM))))*(pow(delta_y, 1-m_alpha_FAM));

                                    gamma_x = (lambda_x)/(lambda_w - lambda_c);
                            }
                    }

                    else if (XQ_x > XI_x && XI_y > XQ_y)
                    {

                            if (XQ_x > XK_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
                            {
                                    lambda_y = (-
m_k3*((((m_k1/m_k3)+(m_k2/m_k3))*cos(0.5*api*m_alpha_FAM)*sin(0.5*api*m_alpha_FAM))+(m_C_
FAM*(m_k2/m_k3)*(pow(cos(0.5*api*m_alpha_FAM),2)))-
(m_C_FAM*(m_k1/m_k3)*(pow(sin(0.5*api*m_alpha_FAM),2)))))*(pow(delta_x, 1-m_alpha_FAM));

                                    gamma_y = (lambda_y)/(lambda_sw - lambda_w);
                            }
```

```
                        else if (XK_y > XQ_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                        {
                                lambda_x = (-m_k1*((cos(0.5*api*m_alpha_FAM))-
(m_C_FAM*sin(0.5*api*m_alpha_FAM))))*(pow(delta_y, 1-m_alpha_FAM));

                                gamma_x = (lambda_x)/(lambda_w - lambda_c);
                        }
                }

                else if (XQ_x > XI_x && XQ_y > XI_y)
                {

                        if (XQ_x > XK_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
                        {
                                lambda_y = (-
m_k3*((((m_k1/m_k3)+(m_k2/m_k3))*cos(0.5*api*m_alpha_FAM)*sin(0.5*api*m_alpha_FAM))+(m_C_
FAM*(m_k2/m_k3)*(pow(cos(0.5*api*m_alpha_FAM),2)))-
(m_C_FAM*(m_k1/m_k3)*(pow(sin(0.5*api*m_alpha_FAM),2)))))*(pow(delta_x, 1-m_alpha_FAM));

                                gamma_y = (lambda_y)/(lambda_sw - lambda_w);
                        }
                        else if (XQ_y > XK_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                        {
                                lambda_x =
(m_k3*((cos(0.5*api*m_alpha_FAM)*((pow(sin(0.5*api*m_alpha_FAM),2))-
((m_k1/m_k2)*(pow(cos(0.5*api*m_alpha_FAM),2)))+(m_C_FAM*(1+(m_k1/m_k2))*cos(0.5*api*m_al
pha_FAM)*sin(0.5*api*m_alpha_FAM))))+(sin(0.5*api*m_alpha_FAM)*((((m_k1/m_k3)+(m_k2/m_k3)
)*cos(0.5*api*m_alpha_FAM)*sin(0.5*api*m_alpha_FAM))+(m_C_FAM*(m_k2/m_k3)*(pow(cos(0.5*ap
i*m_alpha_FAM),2)))-
(m_C_FAM*(m_k1/m_k3)*(pow(sin(0.5*api*m_alpha_FAM),2)))))))*(pow(delta_y, 1-
m_alpha_FAM));

                                gamma_x = (lambda_x)/(lambda_sw - lambda_s);
                        }
                }

                else if (XI_x > XQ_x && XQ_y > XI_y)
                {

                        if (XK_x > XQ_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
                        {
                                lambda_y = (-m_k1*m_C_FAM)*(pow(delta_x, 1-
m_alpha_FAM));

                                gamma_y = (lambda_y)/(lambda_s - lambda_c);
                        }
                        else if (XQ_y > XK_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                        {
                                lambda_x =
(m_k3*((cos(0.5*api*m_alpha_FAM)*((pow(sin(0.5*api*m_alpha_FAM),2))-
((m_k1/m_k2)*(pow(cos(0.5*api*m_alpha_FAM),2)))+(m_C_FAM*(1+(m_k1/m_k2))*cos(0.5*api*m_al
pha_FAM)*sin(0.5*api*m_alpha_FAM))))+(sin(0.5*api*m_alpha_FAM)*((((m_k1/m_k3)+(m_k2/m_k3)
)*cos(0.5*api*m_alpha_FAM)*sin(0.5*api*m_alpha_FAM))+(m_C_FAM*(m_k2/m_k3)*(pow(cos(0.5*ap
```

```
i*m_alpha_FAM),2)))-
(m_C_FAM*(m_k1/m_k3)*(pow(sin(0.5*api*m_alpha_FAM),2)))))))*(pow(delta_y, 1-
m_alpha_FAM));

                                        gamma_x = (lambda_x)/(lambda_sw - lambda_s);
                                }
                        }

                }
                else
                {

                        if (Geometric)
                        {
                                gamma_y = fabs(delta_y/delta_x)*0.5*sqrt(m_k1*m_k2);
                                gamma_x = fabs(delta_x/delta_y)*0.5*sqrt(m_k1*m_k2);
                        }
                        else
                        {
                                if (XI_x > XQ_x && XI_y > XQ_y)
                                {
                                        if (XK_x > XQ_x && fabs(XK_y - XQ_y) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                                        {

                                                gamma_y =
fabs(delta_y/delta_x)*((m_k1*m_k4)/(m_k1+m_k4));
                                        }
                                        else if (XK_y > XQ_y && fabs(XK_x - XQ_x) <= m_Ftol) //
The "&&..." addition is only present as an aditional check
                                        {

                                                gamma_x =
fabs(delta_x/delta_y)*((m_k1*m_k2)/(m_k1+m_k2));
                                        }
                                }

                                else if (XQ_x > XI_x && XI_y > XQ_y)
                                {
                                        if (XQ_x > XK_x && fabs(XK_y - XQ_y) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                                        {

                                                gamma_y =
fabs(delta_y/delta_x)*((m_k2*m_k3)/(m_k2+m_k3));
                                        }
                                        else if (XK_y > XQ_y && fabs(XK_x - XQ_x) <= m_Ftol) //
The "&&..." addition is only present as an aditional check
                                        {

                                                gamma_x =
fabs(delta_x/delta_y)*((m_k1*m_k2)/(m_k1+m_k2));
                                        }
                                }

                                else if (XQ_x > XI_x && XQ_y > XI_y)
                                {
```

```
                                    if (XQ_x > XK_x && fabs(XK_y - XQ_y) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                                    {

                                            gamma_y =
fabs(delta_y/delta_x)*((m_k2*m_k3)/(m_k2+m_k3));
                                    }
                                    else if (XQ_y > XK_y && fabs(XK_x - XQ_x) <= m_Ftol) //
The "&&..." addition is only present as an aditional check
                                    {

                                            gamma_x =
fabs(delta_x/delta_y)*((m_k3*m_k4)/(m_k3+m_k4));
                                    }
                            }

                            else if (XI_x > XQ_x && XQ_y > XI_y)
                            {
                                    if (XK_x > XQ_x && fabs(XK_y - XQ_y) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                                    {

                                            gamma_y =
fabs(delta_y/delta_x)*((m_k1*m_k4)/(m_k1+m_k4));
                                    }
                                    else if (XQ_y > XK_y && fabs(XK_x - XQ_x) <= m_Ftol) //
The "&&..." addition is only present as an aditional check
                                    {

                                            gamma_x =
fabs(delta_x/delta_y)*((m_k3*m_k4)/(m_k3+m_k4));
                                    }
                            }
                    }
            }


       RnPoint normalVec = m_C.u_getNormalVector_0u(PI, PK);

       if ((fabs(normalVec.u_getElement_0u(1)) <= m_Ftol) ? (ret = gamma_x) : (ret =
gamma_y));

       return ret;
}
realS FAM2DSGDiscritization::computeMatrixContributionAtBoundary(const int ihat, const
int itilde, const int khat, const int qhat, const ArrayS<realS>& alpha1, const
ArrayS<realS>& alpha2, const bool atBoundary, const bool edgeAtBoundary)
{
       realS ret=0.0;
       double delta_x = 0.0;
       double delta_y = 0.0;
       double gamma_x = 0.0;
       double gamma_y = 0.0;

       Pair<int,int> PQ(0, qhat);
    Pair<int,int> PI(2, ihat);
       Pair<int,int> PK(1, khat);
```

```
	RnPoint XQ = m_C.u_getCoordinate_0u(PQ);
	double XQ_x = XQ.u_getElement_0u(0);
	double XQ_y = XQ.u_getElement_0u(1);
	RnPoint XI = m_C.u_getCoordinate_0u(PI);
	double XI_x = XI.u_getElement_0u(0);
	double XI_y = XI.u_getElement_0u(1);
	RnPoint XK = m_C.u_getCoordinate_0u(PK);
	double XK_x = XK.u_getElement_0u(0);
	double XK_y = XK.u_getElement_0u(1);

	RnPoint v1 = XK;
	v1 -= XQ;
	RnPoint v2 = XI;
	v2 -= XQ;

	delta_x = XI_x - XQ_x;// Half actual delta_x in the grid!!!!
	delta_y = XI_y - XQ_y;// Half actual delta_y in the grid!!!!


	if (edgeAtBoundary)
	{
		if (fabs(alpha1[m_C.u_findBoundaryPosition_0u(PK)]) <= m_Ftol)
		{
			//We have a Neumann (i.e. flux) B.C.
			gamma_x = 0.0;
			gamma_y = 0.0;
		}
		else
		{
			//We have a Dirichlet (i.e. imposed pressure) B.C.
			gamma_x = fabs(delta_y/delta_x)*m_perm[ihat];
			gamma_y = fabs(delta_x/delta_y)*m_perm[ihat];
		}
	}
	else
	{
		ArrayS<int> neigh;
		m_C.u_getNeighbours_RF(PK, 2, neigh);// the '2' is hardcoded due to 2D
space, for different problem dimensions, use 'problemDimension'....?
		int ihatN = (neigh[0] == ihat) ? neigh[1] : neigh[0];

		gamma_x =
fabs(delta_y/delta_x)*((m_perm[ihat]*m_perm[ihatN])/(m_perm[ihat]+m_perm[ihatN
		gamma_y =
fabs(delta_x/delta_y)*((m_perm[ihat]*m_perm[ihatN])/(m_perm[ihat]+m_perm[ihatN

	}

	RnPoint normalVec = m_C.u_getNormalVector_0u(PI, PK);

	if ((fabs(normalVec.u_getElement_0u(1)) <= m_Ftol) ? (ret = gamma_x) : (ret =
gamma_y));

	return ret;
}
```

Appendix C: Finite Analytic Method Code Implementation (Anisotropic)

```
#include "FAMANI2DSGDiscritization.h"
#include <iostream>
#include <iomanip>
#include <cmath>
using namespace std;

FAMANI2DSGDiscritization::FAMANI2DSGDiscritization()
:	m_alpha_FAM(0.0),
	m_C_FAM(0.0),
	m_permxx(),
	m_permxy(),
	m_permyy(),
	m_C(),
	m_k1_1_1(0.0),
	m_k1_1_2(0.0),
	m_k1_2_2(0.0),
	m_k1_tilde_1(0.0),
	m_k1_tilde_2(0.0),
	m_k2_1_1(0.0),
	m_k2_1_2(0.0),
	m_k2_2_2(0.0),
	m_k2_tilde_1(0.0),
	m_k2_tilde_2(0.0),
	m_k3_1_1(0.0),
	m_k3_1_2(0.0),
	m_k3_2_2(0.0),
	m_k3_tilde_1(0.0),
	m_k3_tilde_2(0.0),
	m_k4_1_1(0.0),
	m_k4_1_2(0.0),
	m_k4_2_2(0.0),
	m_k4_tilde_1(0.0),
	m_k4_tilde_2(0.0),
	m_theta_tilde_1(0.0),
	m_theta_tilde_2(0.0),
	m_theta_tilde_3(0.0),
	m_theta_tilde_4(0.0),
	m_k1(0.0),
	m_k2(0.0),
	m_k3(0.0),
	m_k4(0.0),
	m_Ftol(0.00000001),
	m_useFAMScheme(true)
{
}
FAMANI2DSGDiscritization::~FAMANI2DSGDiscritization()
{
}
void FAMANI2DSGDiscritization::setup(const CSnShape& C, const ArrayS<realS>& permxx,
const ArrayS<realS>& permxy, const ArrayS<realS>& permyy)
{
```

```
	m_C = C;

	m_permxx = permxx;
	m_permxy = permxy;
	m_permyy = permyy;
}
void FAMANI2DSGDiscritization::computeEigenValuesAndRotation(const Pair<int,int> Pqhat,
const int ihat, const bool atBoundary)
{
	int i1 = 0;
	int i2 = 0;
	int i3 = 0;
	int i4 = 0;

	//Must find control volume neighbores "Pqhat", use a CSnShape 'getNeighbors....'

	if (!atBoundary)
	{
		ArrayS<int> neigh;
		m_C.u_getNeighbours_RF(Pqhat, 2, neigh);// the '2' is hardcoded due to 2D
space, for different problem dimensions, use 'problemDimension'....?
		i1 = neigh[0];
		i2 = neigh[1];
		i3 = neigh[2];
		i4 = neigh[3];

		//Must first find the correct quadrant for ihat:
		double FTOLL = 0.000001;
		Pair<int, int> Pihat(2, ihat);
		RnPoint XI = m_C.u_getCoordinate_0u(Pihat);
		RnPoint XQ = m_C.u_getCoordinate_0u(Pqhat);
		double XI_x = XI.u_getElement_0u(0);
		double XI_y = XI.u_getElement_0u(1);
		double XQ_x = XQ.u_getElement_0u(0);
		double XQ_y = XQ.u_getElement_0u(1);

		int whichQuad = -1;
		if ((XI_x-XQ_x) > FTOLL && (XI_y-XQ_y) > FTOLL)
		{
		  //1 quadrant
		  whichQuad = 1;
		}
		else if ((XQ_x-XI_x) > FTOLL && (XI_y-XQ_y) > FTOLL)
		{
		  //2 quadrant
		  whichQuad = 2;
		}
		else if ((XQ_x-XI_x) > FTOLL && (XQ_y-XI_y) > FTOLL)
		{
		  //3 quadrant
		  whichQuad = 3;
		}
		else if ((XI_x-XQ_x) > FTOLL && (XQ_y-XI_y) > FTOLL)
		{
		  //4 quadrant
		  whichQuad = 4;
		}
		 else
```

```
                {
                  cout << "Something is terribly wrong!!! Program is stopped... " << endl;
exit(0);
                }

                if (whichQuad == 1)//ihat in first quadrant
                {
                        if (i1 == ihat)
                        {
                        i1 = neigh[0];
                        i2 = neigh[1];
                        i3 = neigh[2];
                        i4 = neigh[3];
                        }
                        else if (i2 == ihat)
                        {
                        i1 = neigh[1];
                        i2 = neigh[2];
                        i3 = neigh[3];
                        i4 = neigh[0];
                        }
                        else if (i3 == ihat)
                        {
                        i1 = neigh[2];
                        i2 = neigh[3];
                        i3 = neigh[0];
                        i4 = neigh[1];
                        }
                        else if (i4 == ihat)
                        {
                        i1 = neigh[3];
                        i2 = neigh[0];
                        i3 = neigh[1];
                        i4 = neigh[2];
                        }
                }
                else if (whichQuad == 2)//ihat in second quadrant
                {
                        if (i1 == ihat)
                        {
                        i1 = neigh[3];
                        i2 = neigh[0];
                        i3 = neigh[1];
                        i4 = neigh[2];
                        }
                        else if (i2 == ihat)
                        {
                        i1 = neigh[0];
                        i2 = neigh[1];
                        i3 = neigh[2];
                        i4 = neigh[3];
                        }
                        else if (i3 == ihat)
                        {
                        i1 = neigh[1];
                        i2 = neigh[2];
                        i3 = neigh[3];
                        i4 = neigh[0];
```

```
        }
        else if (i4 == ihat)
        {
        i1 = neigh[2];
        i2 = neigh[3];
        i3 = neigh[0];
        i4 = neigh[1];
        }
}
else if (whichQuad == 3)//ihat in third quadrant
{
        if (i1 == ihat)
        {
        i1 = neigh[2];
        i2 = neigh[3];
        i3 = neigh[0];
        i4 = neigh[1];
        }
        else if (i2 == ihat)
        {
        i1 = neigh[3];
        i2 = neigh[0];
        i3 = neigh[1];
        i4 = neigh[2];
        }
        else if (i3 == ihat)
        {
        i1 = neigh[0];
        i2 = neigh[1];
        i3 = neigh[2];
        i4 = neigh[3];
        }
        else if (i4 == ihat)
        {
        i1 = neigh[1];
        i2 = neigh[2];
        i3 = neigh[3];
        i4 = neigh[0];
        }
}
else if (whichQuad == 4)//ihat in fourth quadrant
{
        if (i1 == ihat)
        {
        i1 = neigh[1];
        i2 = neigh[2];
        i3 = neigh[3];
        i4 = neigh[0];
        }
        else if (i2 == ihat)
        {
        i1 = neigh[2];
        i2 = neigh[3];
        i3 = neigh[0];
        i4 = neigh[1];
        }
        else if (i3 == ihat)
        {
```

```
                i1 = neigh[3];
                i2 = neigh[0];
                i3 = neigh[1];
                i4 = neigh[2];
                }
                else if (i4 == ihat)
                {
                i1 = neigh[0];
                i2 = neigh[1];
                i3 = neigh[2];
                i4 = neigh[3];
                }
        }

        m_k1_1_1 = m_permxx[i1];
        m_k1_1_2 = m_permxy[i1];
        m_k1_2_2 = m_permyy[i1];

        m_k2_1_1 = m_permxx[i2];
        m_k2_1_2 = m_permxy[i2];
        m_k2_2_2 = m_permyy[i2];

        m_k3_1_1 = m_permxx[i3];
        m_k3_1_2 = m_permxy[i3];
        m_k3_2_2 = m_permyy[i3];

        m_k4_1_1 = m_permxx[i4];
        m_k4_1_2 = m_permxy[i4];
        m_k4_2_2 = m_permyy[i4];

        }

//*********************************************************************//

Eigen::VectorXd vec_yaxis(2);
vec_yaxis(0) = 0;
vec_yaxis(1) = 1;

Eigen::VectorXd eigvec_1_1(2);
Eigen::VectorXd eigvec_1_2(2);
Eigen::VectorXd eigvec_2_1(2);
Eigen::VectorXd eigvec_2_2(2);
Eigen::VectorXd eigvec_3_1(2);
Eigen::VectorXd eigvec_3_2(2);
Eigen::VectorXd eigvec_4_1(2);
Eigen::VectorXd eigvec_4_2(2);
double eigval_1_1 = 0.0;
double eigval_1_2 = 0.0;
double eigval_2_1 = 0.0;
double eigval_2_2 = 0.0;
double eigval_3_1 = 0.0;
double eigval_3_2 = 0.0;
double eigval_4_1 = 0.0;
double eigval_4_2 = 0.0;
```

```
       eigval_1_1 = 0.5*(m_k1_2_2 + m_k1_1_1 +
sqrt((m_k1_1_1*m_k1_1_1)+(4*m_k1_1_2*m_k1_1_2)-
(2*m_k1_1_1*m_k1_2_2)+(m_k1_2_2*m_k1_2_2)));
       eigval_1_2 = 0.5*(m_k1_2_2 + m_k1_1_1 -
sqrt((m_k1_1_1*m_k1_1_1)+(4*m_k1_1_2*m_k1_1_2)-
(2*m_k1_1_1*m_k1_2_2)+(m_k1_2_2*m_k1_2_2)));
       eigval_2_1 = 0.5*(m_k2_2_2 + m_k2_1_1 +
sqrt((m_k2_1_1*m_k2_1_1)+(4*m_k2_1_2*m_k2_1_2)-
(2*m_k2_1_1*m_k2_2_2)+(m_k2_2_2*m_k2_2_2)));
       eigval_2_2 = 0.5*(m_k2_2_2 + m_k2_1_1 -
sqrt((m_k2_1_1*m_k2_1_1)+(4*m_k2_1_2*m_k2_1_2)-
(2*m_k2_1_1*m_k2_2_2)+(m_k2_2_2*m_k2_2_2)));
       eigval_3_1 = 0.5*(m_k3_2_2 + m_k3_1_1 +
sqrt((m_k3_1_1*m_k3_1_1)+(4*m_k3_1_2*m_k3_1_2)-
(2*m_k3_1_1*m_k3_2_2)+(m_k3_2_2*m_k3_2_2)));
       eigval_3_2 = 0.5*(m_k3_2_2 + m_k3_1_1 -
sqrt((m_k3_1_1*m_k3_1_1)+(4*m_k3_1_2*m_k3_1_2)-
(2*m_k3_1_1*m_k3_2_2)+(m_k3_2_2*m_k3_2_2)));
       eigval_4_1 = 0.5*(m_k4_2_2 + m_k4_1_1 +
sqrt((m_k4_1_1*m_k4_1_1)+(4*m_k4_1_2*m_k4_1_2)-
(2*m_k4_1_1*m_k4_2_2)+(m_k4_2_2*m_k4_2_2)));
       eigval_4_2 = 0.5*(m_k4_2_2 + m_k4_1_1 -
sqrt((m_k4_1_1*m_k4_1_1)+(4*m_k4_1_2*m_k4_1_2)-
(2*m_k4_1_1*m_k4_2_2)+(m_k4_2_2*m_k4_2_2)));

       if (fabs(m_k1_1_2) < m_Ftol) (m_k1_1_2 = m_Ftol);
       if (fabs(m_k2_1_2) < m_Ftol) (m_k2_1_2 = m_Ftol);
       if (fabs(m_k3_1_2) < m_Ftol) (m_k3_1_2 = m_Ftol);
       if (fabs(m_k4_1_2) < m_Ftol) (m_k4_1_2 = m_Ftol);

       eigvec_1_1(0) = eigval_1_1 - m_k1_2_2;
       eigvec_1_1(1) = m_k1_1_2;
       eigvec_1_2(0) = eigval_1_2 - m_k1_2_2;
       eigvec_1_2(1) = m_k1_1_2;

       eigvec_2_1(0) = eigval_2_1 - m_k2_2_2;
       eigvec_2_1(1) = m_k2_1_2;
       eigvec_2_2(0) = eigval_2_2 - m_k2_2_2;
       eigvec_2_2(1) = m_k2_1_2;

       eigvec_3_1(0) = eigval_3_1 - m_k3_2_2;
       eigvec_3_1(1) = m_k3_1_2;
       eigvec_3_2(0) = eigval_3_2 - m_k3_2_2;
       eigvec_3_2(1) = m_k3_1_2;

       eigvec_4_1(0) = eigval_4_1 - m_k4_2_2;
       eigvec_4_1(1) = m_k4_1_2;
       eigvec_4_2(0) = eigval_4_2 - m_k4_2_2;
       eigvec_4_2(1) = m_k4_1_2;

       m_k1_tilde_1 = eigval_1_1;
       m_k1_tilde_2 = eigval_1_2;
       m_k2_tilde_1 = eigval_2_1;
       m_k2_tilde_2 = eigval_2_2;
       m_k3_tilde_1 = eigval_3_1;
       m_k3_tilde_2 = eigval_3_2;
       m_k4_tilde_1 = eigval_4_1;
       m_k4_tilde_2 = eigval_4_2;
```

```
	m_theta_tilde_1 =
acos(eigvec_1_1.dot(vec_yaxis)/(eigvec_1_1.norm()*vec_yaxis.norm()));
	m_theta_tilde_2 =
acos(eigvec_2_1.dot(vec_yaxis)/(eigvec_2_1.norm()*vec_yaxis.norm()));
	m_theta_tilde_3 =
acos(eigvec_3_1.dot(vec_yaxis)/(eigvec_3_1.norm()*vec_yaxis.norm()));
	m_theta_tilde_4 =
acos(eigvec_4_1.dot(vec_yaxis)/(eigvec_4_1.norm()*vec_yaxis.norm()));

}
void FAMANI2DSGDiscritization::computeHelpValuesUVAndPhiForAlpha(double& phi_1_1, double&
phi_2_1, double& phi_1_2, double& phi_2_2,
	double& u_1_1, double& u_1_2, double& u_2_1, double& u_2_2, double& u_3_1, double&
u_3_2, double& u_4_1, double& u_4_2,
	double& v_1_1, double& v_1_2, double& v_2_1, double& v_2_2, double& v_3_1, double&
v_3_2, double& v_4_1, double& v_4_2, const double& alpha)
{
	double api = atan(1.0)*4;

	double a_1_1_y = 0.0;
	double a_1_1_x = 0.0;
	double a_1_2_y = 0.0;
	double a_1_2_x = 0.0;
	double a_2_1_y = 0.0;
	double a_2_1_x = 0.0;
	double a_2_2_y = 0.0;
	double a_2_2_x = 0.0;
	double a_3_1_y = 0.0;
	double a_3_1_x = 0.0;
	double a_3_2_y = 0.0;
	double a_3_2_x = 0.0;
	double a_4_1_y = 0.0;
	double a_4_1_x = 0.0;
	double a_4_2_y = 0.0;
	double a_4_2_x = 0.0;
	double b_1_1_y = 0.0;
	double b_1_1_x = 0.0;
	double b_1_2_y = 0.0;
	double b_1_2_x = 0.0;
	double b_2_1_y = 0.0;
	double b_2_1_x = 0.0;
	double b_2_2_y = 0.0;
	double b_2_2_x = 0.0;
	double b_3_1_y = 0.0;
	double b_3_1_x = 0.0;
	double b_3_2_y = 0.0;
	double b_3_2_x = 0.0;
	double b_4_1_y = 0.0;
	double b_4_1_x = 0.0;
	double b_4_2_y = 0.0;
	double b_4_2_x = 0.0;

	double x_1_y = 1.0;//the below is VERY important: setting for when 'x' or 'y' is
zero, which in turn can cancel the other (which is why it is set to one)
	double x_1_x = 0.0;
	double x_2_y = 1.0;
	double x_2_x = 0.0;
```

```
        double x_3_y = 1.0;
        double x_3_x = 0.0;
        double x_4_y = 1.0;
        double x_4_x = 0.0;
        double y_1_y = 0.0;
        double y_1_x = 1.0;
        double y_2_y = 0.0;
        double y_2_x = 1.0;
        double y_3_y = 0.0;
        double y_3_x = 1.0;
        double y_4_y = 0.0;
        double y_4_x = 1.0;

        double x_hat_1_y =
((x_1_y*cos(m_theta_tilde_1))+(y_1_y*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_1);
        double x_hat_1_x =
((x_1_x*cos(m_theta_tilde_1))+(y_1_x*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_1);
        double x_hat_2_y =
((x_2_y*cos(m_theta_tilde_2))+(y_2_y*sin(m_theta_tilde_2)))/sqrt(m_k2_tilde_1);
        double x_hat_2_x =
((x_2_x*cos(m_theta_tilde_2))+(y_2_x*sin(m_theta_tilde_2)))/sqrt(m_k2_tilde_1);
        double x_hat_3_y =
((x_3_y*cos(m_theta_tilde_3))+(y_3_y*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_1);
        double x_hat_3_x =
((x_3_x*cos(m_theta_tilde_3))+(y_3_x*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_1);
        double x_hat_4_y =
((x_4_y*cos(m_theta_tilde_4))+(y_4_y*sin(m_theta_tilde_4)))/sqrt(m_k4_tilde_1);
        double x_hat_4_x =
((x_4_x*cos(m_theta_tilde_4))+(y_4_x*sin(m_theta_tilde_4)))/sqrt(m_k4_tilde_1);
        double y_hat_1_y = ((y_1_y*cos(m_theta_tilde_1))-
(x_1_y*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_2);
        double y_hat_1_x = ((y_1_x*cos(m_theta_tilde_1))-
(x_1_x*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_2);
        double y_hat_2_y = ((y_2_y*cos(m_theta_tilde_2))-
(x_2_y*sin(m_theta_tilde_2)))/sqrt(m_k2_tilde_2);
        double y_hat_2_x = ((y_2_x*cos(m_theta_tilde_2))-
(x_2_x*sin(m_theta_tilde_2)))/sqrt(m_k2_tilde_2);
        double y_hat_3_y = ((y_3_y*cos(m_theta_tilde_3))-
(x_3_y*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_2);
        double y_hat_3_x = ((y_3_x*cos(m_theta_tilde_3))-
(x_3_x*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_2);
        double y_hat_4_y = ((y_4_y*cos(m_theta_tilde_4))-
(x_4_y*sin(m_theta_tilde_4)))/sqrt(m_k4_tilde_2);
        double y_hat_4_x = ((y_4_x*cos(m_theta_tilde_4))-
(x_4_x*sin(m_theta_tilde_4)))/sqrt(m_k4_tilde_2);

        double theta_hat_1_y = (0)-m_theta_tilde_1;
        double theta_hat_1_x = (0.5*api)-m_theta_tilde_1;
        double theta_hat_2_y = (api)-m_theta_tilde_2;
        double theta_hat_2_x = (0.5*api)-m_theta_tilde_2;
        double theta_hat_3_y = (-api)-m_theta_tilde_3;
        double theta_hat_3_x = (-0.5*api)-m_theta_tilde_3;
        double theta_hat_4_y = (0)-m_theta_tilde_4;
        double theta_hat_4_x = (-0.5*api)-m_theta_tilde_4;
        double radius_1_y = sqrt((x_hat_1_y*x_hat_1_y)+(y_hat_1_y*y_hat_1_y));
        double radius_1_x = sqrt((x_hat_1_x*x_hat_1_x)+(y_hat_1_x*y_hat_1_x));
        double radius_2_y = sqrt((x_hat_2_y*x_hat_2_y)+(y_hat_2_y*y_hat_2_y));
        double radius_2_x = sqrt((x_hat_2_x*x_hat_2_x)+(y_hat_2_x*y_hat_2_x));
```

```
double radius_3_y = sqrt((x_hat_3_y*x_hat_3_y)+(y_hat_3_y*y_hat_3_y));
double radius_3_x = sqrt((x_hat_3_x*x_hat_3_x)+(y_hat_3_x*y_hat_3_x));
double radius_4_y = sqrt((x_hat_4_y*x_hat_4_y)+(y_hat_4_y*y_hat_4_y));
double radius_4_x = sqrt((x_hat_4_x*x_hat_4_x)+(y_hat_4_x*y_hat_4_x));

double dx_dx_1_y = cos(m_theta_tilde_1)/sqrt(m_k1_tilde_1);
double dx_dx_1_x = cos(m_theta_tilde_1)/sqrt(m_k1_tilde_1);
double dx_dx_2_y = cos(m_theta_tilde_2)/sqrt(m_k2_tilde_1);
double dx_dx_2_x = cos(m_theta_tilde_2)/sqrt(m_k2_tilde_1);
double dx_dx_3_y = cos(m_theta_tilde_3)/sqrt(m_k3_tilde_1);
double dx_dx_3_x = cos(m_theta_tilde_3)/sqrt(m_k3_tilde_1);
double dx_dx_4_y = cos(m_theta_tilde_4)/sqrt(m_k4_tilde_1);
double dx_dx_4_x = cos(m_theta_tilde_4)/sqrt(m_k4_tilde_1);
double dy_dy_1_y = cos(m_theta_tilde_1)/sqrt(m_k1_tilde_2);
double dy_dy_1_x = cos(m_theta_tilde_1)/sqrt(m_k1_tilde_2);
double dy_dy_2_y = cos(m_theta_tilde_2)/sqrt(m_k2_tilde_2);
double dy_dy_2_x = cos(m_theta_tilde_2)/sqrt(m_k2_tilde_2);
double dy_dy_3_y = cos(m_theta_tilde_3)/sqrt(m_k3_tilde_2);
double dy_dy_3_x = cos(m_theta_tilde_3)/sqrt(m_k3_tilde_2);
double dy_dy_4_y = cos(m_theta_tilde_4)/sqrt(m_k4_tilde_2);
double dy_dy_4_x = cos(m_theta_tilde_4)/sqrt(m_k4_tilde_2);


double dtheta_dx_1_y = (-y_hat_1_y/(pow(radius_1_y, 2)))*(1-alpha);
double dtheta_dx_1_x = (-y_hat_1_x/(pow(radius_1_x, 2)))*(1-alpha);
double dtheta_dx_2_y = (-y_hat_2_y/(pow(radius_2_y, 2)))*(1-alpha);
double dtheta_dx_2_x = (-y_hat_2_x/(pow(radius_2_x, 2)))*(1-alpha);
double dtheta_dx_3_y = (-y_hat_3_y/(pow(radius_3_y, 2)))*(1-alpha);
double dtheta_dx_3_x = (-y_hat_3_x/(pow(radius_3_x, 2)))*(1-alpha);
double dtheta_dx_4_y = (-y_hat_4_y/(pow(radius_4_y, 2)))*(1-alpha);
double dtheta_dx_4_x = (-y_hat_4_x/(pow(radius_4_x, 2)))*(1-alpha);
double dtheta_dy_1_y = (x_hat_1_y/(pow(radius_1_y, 2)))*(1-alpha);
double dtheta_dy_1_x = (x_hat_1_x/(pow(radius_1_x, 2)))*(1-alpha);
double dtheta_dy_2_y = (x_hat_2_y/(pow(radius_2_y, 2)))*(1-alpha);
double dtheta_dy_2_x = (x_hat_2_x/(pow(radius_2_x, 2)))*(1-alpha);
double dtheta_dy_3_y = (x_hat_3_y/(pow(radius_3_y, 2)))*(1-alpha);
double dtheta_dy_3_x = (x_hat_3_x/(pow(radius_3_x, 2)))*(1-alpha);
double dtheta_dy_4_y = (x_hat_4_y/(pow(radius_4_y, 2)))*(1-alpha);
double dtheta_dy_4_x = (x_hat_4_x/(pow(radius_4_x, 2)))*(1-alpha);

double dr_dx_1_y = (x_hat_1_y/radius_1_y)*(pow(radius_1_y, -alpha))*(1-alpha);
double dr_dx_1_x = (x_hat_1_x/radius_1_x)*(pow(radius_1_x, -alpha))*(1-alpha);
double dr_dx_2_y = (x_hat_2_y/radius_2_y)*(pow(radius_2_y, -alpha))*(1-alpha);
double dr_dx_2_x = (x_hat_2_x/radius_2_x)*(pow(radius_2_x, -alpha))*(1-alpha);
double dr_dx_3_y = (x_hat_3_y/radius_3_y)*(pow(radius_3_y, -alpha))*(1-alpha);
double dr_dx_3_x = (x_hat_3_x/radius_3_x)*(pow(radius_3_x, -alpha))*(1-alpha);
double dr_dx_4_y = (x_hat_4_y/radius_4_y)*(pow(radius_4_y, -alpha))*(1-alpha);
double dr_dx_4_x = (x_hat_4_x/radius_4_x)*(pow(radius_4_x, -alpha))*(1-alpha);
double dr_dy_1_y = (y_hat_1_y/radius_1_y)*(pow(radius_1_y, -alpha))*(1-alpha);
double dr_dy_1_x = (y_hat_1_x/radius_1_x)*(pow(radius_1_x, -alpha))*(1-alpha);
double dr_dy_2_y = (y_hat_2_y/radius_2_y)*(pow(radius_2_y, -alpha))*(1-alpha);
double dr_dy_2_x = (y_hat_2_x/radius_2_x)*(pow(radius_2_x, -alpha))*(1-alpha);
double dr_dy_3_y = (y_hat_3_y/radius_3_y)*(pow(radius_3_y, -alpha))*(1-alpha);
double dr_dy_3_x = (y_hat_3_x/radius_3_x)*(pow(radius_3_x, -alpha))*(1-alpha);
double dr_dy_4_y = (y_hat_4_y/radius_4_y)*(pow(radius_4_y, -alpha))*(1-alpha);
double dr_dy_4_x = (y_hat_4_x/radius_4_x)*(pow(radius_4_x, -alpha))*(1-alpha);

a_1_1_y = (pow(radius_1_y, 1-alpha))*cos(theta_hat_1_y*(1-alpha));
```

```
	a_1_1_x = (pow(radius_1_x, 1-alpha))*cos(theta_hat_1_x*(1-alpha));
	a_2_1_y = (pow(radius_2_y, 1-alpha))*cos((theta_hat_2_y-(0.5*api))*(1-alpha));
	a_2_1_x = (pow(radius_2_x, 1-alpha))*cos((theta_hat_2_x-(0.5*api))*(1-alpha));
	a_3_1_y = (pow(radius_3_y, 1-alpha))*cos((theta_hat_3_y+api)*(1-alpha));//The plus
or minus PI has to do with the direction of orientation. Seeing as how our calcualtion go
from -PI to +PI,
	a_3_1_x = (pow(radius_3_x, 1-alpha))*cos((theta_hat_3_x+api)*(1-alpha));//when
calculating from the positive side (2 quadrant), one must have +PI, and vice versa
(applied one all 3 quadrant equations)
	a_4_1_y = (pow(radius_4_y, 1-alpha))*cos((theta_hat_4_y+(0.5*api))*(1-alpha));
	a_4_1_x = (pow(radius_4_x, 1-alpha))*cos((theta_hat_4_x+(0.5*api))*(1-alpha));

	b_1_1_y = (pow(radius_1_y, 1-alpha))*sin(theta_hat_1_y*(1-alpha));
	b_1_1_x = (pow(radius_1_x, 1-alpha))*sin(theta_hat_1_x*(1-alpha));
	b_2_1_y = -(pow(radius_2_y, 1-alpha))*sin((theta_hat_2_y-(0.5*api))*(1-alpha));
	b_2_1_x = -(pow(radius_2_x, 1-alpha))*sin((theta_hat_2_x-(0.5*api))*(1-alpha));
	b_3_1_y = -(pow(radius_3_y, 1-alpha))*sin((theta_hat_3_y+api)*(1-alpha));
	b_3_1_x = -(pow(radius_3_x, 1-alpha))*sin((theta_hat_3_x+api)*(1-alpha));
	b_4_1_y = (pow(radius_4_y, 1-alpha))*sin((theta_hat_4_y+(0.5*api))*(1-alpha));
	b_4_1_x = (pow(radius_4_x, 1-alpha))*sin((theta_hat_4_x+(0.5*api))*(1-alpha));

	a_1_2_y = (m_k1_1_2*dx_dx_1_y*((dr_dx_1_y*cos(theta_hat_1_y*(1-alpha)))+(-
dtheta_dx_1_y*(pow(radius_1_y, 1-alpha))*sin(theta_hat_1_y*(1-alpha)))))
			+(m_k1_2_2*dy_dy_1_y*((dr_dy_1_y*cos(theta_hat_1_y*(1-alpha)))+(-
dtheta_dy_1_y*(pow(radius_1_y, 1-alpha))*sin(theta_hat_1_y*(1-alpha)))));
	a_1_2_x = (m_k1_1_1*dx_dx_1_x*((dr_dx_1_x*cos(theta_hat_1_x*(1-alpha)))+(-
dtheta_dx_1_x*(pow(radius_1_x, 1-alpha))*sin(theta_hat_1_x*(1-alpha)))))
			+(m_k1_1_2*dy_dy_1_x*((dr_dy_1_x*cos(theta_hat_1_x*(1-alpha)))+(-
dtheta_dy_1_x*(pow(radius_1_x, 1-alpha))*sin(theta_hat_1_x*(1-alpha)))));
	a_2_2_y = (m_k2_1_2*dx_dx_2_y*((dr_dx_2_y*cos((theta_hat_2_y-(0.5*api))*(1-
alpha)))+(-dtheta_dx_2_y*(pow(radius_2_y, 1-alpha))*sin((theta_hat_2_y-(0.5*api))*(1-
alpha)))))
			+(m_k2_2_2*dy_dy_2_y*((dr_dy_2_y*cos((theta_hat_2_y-(0.5*api))*(1-
alpha)))+(-dtheta_dy_2_y*(pow(radius_2_y, 1-alpha))*sin((theta_hat_2_y-(0.5*api))*(1-
alpha)))));
	a_2_2_x = (m_k2_1_1*dx_dx_2_x*((dr_dx_2_x*cos((theta_hat_2_x-(0.5*api))*(1-
alpha)))+(-dtheta_dx_2_x*(pow(radius_2_x, 1-alpha))*sin((theta_hat_2_x-(0.5*api))*(1-
alpha)))))
			+(m_k2_1_2*dy_dy_2_x*((dr_dy_2_x*cos((theta_hat_2_x-(0.5*api))*(1-
alpha)))+(-dtheta_dy_2_x*(pow(radius_2_x, 1-alpha))*sin((theta_hat_2_x-(0.5*api))*(1-
alpha)))));
	a_3_2_y = (m_k3_1_2*dx_dx_3_y*((dr_dx_3_y*cos((theta_hat_3_y+api)*(1-alpha)))+(-
dtheta_dx_3_y*(pow(radius_3_y, 1-alpha))*sin((theta_hat_3_y+api)*(1-alpha)))))
			+(m_k3_2_2*dy_dy_3_y*((dr_dy_3_y*cos((theta_hat_3_y+api)*(1-
alpha)))+(-dtheta_dy_3_y*(pow(radius_3_y, 1-alpha))*sin((theta_hat_3_y+api)*(1-
alpha)))));
	a_3_2_x = (m_k3_1_1*dx_dx_3_x*((dr_dx_3_x*cos((theta_hat_3_x+api)*(1-alpha)))+(-
dtheta_dx_3_x*(pow(radius_3_x, 1-alpha))*sin((theta_hat_3_x+api)*(1-alpha)))))
			+(m_k3_1_2*dy_dy_3_x*((dr_dy_3_x*cos((theta_hat_3_x+api)*(1-
alpha)))+(-dtheta_dy_3_x*(pow(radius_3_x, 1-alpha))*sin((theta_hat_3_x+api)*(1-
alpha)))));
	a_4_2_y = (m_k4_1_2*dx_dx_4_y*((dr_dx_4_y*cos((theta_hat_4_y+(0.5*api))*(1-
alpha)))+(-dtheta_dx_4_y*(pow(radius_4_y, 1-alpha))*sin((theta_hat_4_y+(0.5*api))*(1-
alpha)))))
			+(m_k4_2_2*dy_dy_4_y*((dr_dy_4_y*cos((theta_hat_4_y+(0.5*api))*(1-
alpha)))+(-dtheta_dy_4_y*(pow(radius_4_y, 1-alpha))*sin((theta_hat_4_y+(0.5*api))*(1-
alpha)))));
```

```
      a_4_2_x = (m_k4_1_1*dx_dx_4_x*((dr_dx_4_x*cos((theta_hat_4_x+(0.5*api))*(1-
alpha)))+(-dtheta_dx_4_x*(pow(radius_4_x, 1-alpha))*sin((theta_hat_4_x+(0.5*api))*(1-
alpha)))))
                     +(m_k4_1_2*dy_dy_4_x*((dr_dy_4_x*cos((theta_hat_4_x+(0.5*api))*(1-
alpha)))+(-dtheta_dy_4_x*(pow(radius_4_x, 1-alpha))*sin((theta_hat_4_x+(0.5*api))*(1-
alpha)))));

      b_1_2_y = (m_k1_1_2*dx_dx_1_y*((dr_dx_1_y*sin(theta_hat_1_y*(1-
alpha)))+(dtheta_dx_1_y*(pow(radius_1_y, 1-alpha))*cos(theta_hat_1_y*(1-alpha)))))
                     +(m_k1_2_2*dy_dy_1_y*((dr_dy_1_y*sin(theta_hat_1_y*(1-
alpha)))+(dtheta_dy_1_y*(pow(radius_1_y, 1-alpha))*cos(theta_hat_1_y*(1-alpha)))));
      b_1_2_x = (m_k1_1_1*dx_dx_1_x*((dr_dx_1_x*sin(theta_hat_1_x*(1-
alpha)))+(dtheta_dx_1_x*(pow(radius_1_x, 1-alpha))*cos(theta_hat_1_x*(1-alpha)))))
                     +(m_k1_1_2*dy_dy_1_x*((dr_dy_1_x*sin(theta_hat_1_x*(1-
alpha)))+(dtheta_dy_1_x*(pow(radius_1_x, 1-alpha))*cos(theta_hat_1_x*(1-alpha)))));
      b_2_2_y = -((m_k2_1_2*dx_dx_2_y*((dr_dx_2_y*sin((theta_hat_2_y-(0.5*api))*(1-
alpha)))+(dtheta_dx_2_y*(pow(radius_2_y, 1-alpha))*cos((theta_hat_2_y-(0.5*api))*(1-
alpha)))))
                       +(m_k2_2_2*dy_dy_2_y*((dr_dy_2_y*sin((theta_hat_2_y-
(0.5*api))*(1-alpha)))+(dtheta_dy_2_y*(pow(radius_2_y, 1-alpha))*cos((theta_hat_2_y-
(0.5*api))*(1-alpha))))));
      b_2_2_x = -((m_k2_1_1*dx_dx_2_x*((dr_dx_2_x*sin((theta_hat_2_x-(0.5*api))*(1-
alpha)))+(dtheta_dx_2_x*(pow(radius_2_x, 1-alpha))*cos((theta_hat_2_x-(0.5*api))*(1-
alpha)))))
                       +(m_k2_1_2*dy_dy_2_x*((dr_dy_2_x*sin((theta_hat_2_x-
(0.5*api))*(1-alpha)))+(dtheta_dy_2_x*(pow(radius_2_x, 1-alpha))*cos((theta_hat_2_x-
(0.5*api))*(1-alpha))))));
      b_3_2_y = -((m_k3_1_2*dx_dx_3_y*((dr_dx_3_y*sin((theta_hat_3_y+api)*(1-
alpha)))+(dtheta_dx_3_y*(pow(radius_3_y, 1-alpha))*cos((theta_hat_3_y+api)*(1-alpha)))))
                       +(m_k3_2_2*dy_dy_3_y*((dr_dy_3_y*sin((theta_hat_3_y+api)*(1-
alpha)))+(dtheta_dy_3_y*(pow(radius_3_y, 1-alpha))*cos((theta_hat_3_y+api)*(1-
alpha))))));
      b_3_2_x = -((m_k3_1_1*dx_dx_3_x*((dr_dx_3_x*sin((theta_hat_3_x+api)*(1-
alpha)))+(dtheta_dx_3_x*(pow(radius_3_x, 1-alpha))*cos((theta_hat_3_x+api)*(1-alpha)))))
                       +(m_k3_1_2*dy_dy_3_x*((dr_dy_3_x*sin((theta_hat_3_x+api)*(1-
alpha)))+(dtheta_dy_3_x*(pow(radius_3_x, 1-alpha))*cos((theta_hat_3_x+api)*(1-
alpha))))));
      b_4_2_y = (m_k4_1_2*dx_dx_4_y*((dr_dx_4_y*sin((theta_hat_4_y+(0.5*api))*(1-
alpha)))+(dtheta_dx_4_y*(pow(radius_4_y, 1-alpha))*cos((theta_hat_4_y+(0.5*api))*(1-
alpha)))))
                     +(m_k4_2_2*dy_dy_4_y*((dr_dy_4_y*sin((theta_hat_4_y+(0.5*api))*(1-
alpha)))+(dtheta_dy_4_y*(pow(radius_4_y, 1-alpha))*cos((theta_hat_4_y+(0.5*api))*(1-
alpha)))));
      b_4_2_x = (m_k4_1_1*dx_dx_4_x*((dr_dx_4_x*sin((theta_hat_4_x+(0.5*api))*(1-
alpha)))+(dtheta_dx_4_x*(pow(radius_4_x, 1-alpha))*cos((theta_hat_4_x+(0.5*api))*(1-
alpha)))))
                     +(m_k4_1_2*dy_dy_4_x*((dr_dy_4_x*sin((theta_hat_4_x+(0.5*api))*(1-
alpha)))+(dtheta_dy_4_x*(pow(radius_4_x, 1-alpha))*cos((theta_hat_4_x+(0.5*api))*(1-
alpha)))));


      u_1_1 = ((a_4_1_y*b_1_2_y)-(a_4_2_y*b_1_1_y))/((a_1_1_y*b_1_2_y)-
(a_1_2_y*b_1_1_y));
      u_1_2 = (-(a_4_1_y*a_1_2_y)+(a_4_2_y*a_1_1_y))/((a_1_1_y*b_1_2_y)-
(a_1_2_y*b_1_1_y));
      u_2_1 = ((a_1_1_x*b_2_2_x)-(a_1_2_x*b_2_1_x))/((a_2_1_x*b_2_2_x)-
(a_2_2_x*b_2_1_x));
```

```
	u_2_2 = (-(a_1_1_x*a_2_2_x)+(a_1_2_x*a_2_1_x))/((a_2_1_x*b_2_2_x)-
(a_2_2_x*b_2_1_x));
	u_3_1 = ((a_2_1_y*b_3_2_y)-(a_2_2_y*b_3_1_y))/((a_3_1_y*b_3_2_y)-
(a_3_2_y*b_3_1_y));
	u_3_2 = (-(a_2_1_y*a_3_2_y)+(a_2_2_y*a_3_1_y))/((a_3_1_y*b_3_2_y)-
(a_3_2_y*b_3_1_y));
	u_4_1 = ((a_3_1_x*b_4_2_x)-(a_3_2_x*b_4_1_x))/((a_4_1_x*b_4_2_x)-
(a_4_2_x*b_4_1_x));
	u_4_2 = (-(a_3_1_x*a_4_2_x)+(a_3_2_x*a_4_1_x))/((a_4_1_x*b_4_2_x)-
(a_4_2_x*b_4_1_x));

	v_1_1 = ((b_4_1_y*b_1_2_y)-(b_4_2_y*b_1_1_y))/((a_1_1_y*b_1_2_y)-
(a_1_2_y*b_1_1_y));
	v_1_2 = (-(b_4_1_y*a_1_2_y)+(b_4_2_y*a_1_1_y))/((a_1_1_y*b_1_2_y)-
(a_1_2_y*b_1_1_y));
	v_2_1 = ((b_1_1_x*b_2_2_x)-(b_1_2_x*b_2_1_x))/((a_2_1_x*b_2_2_x)-
(a_2_2_x*b_2_1_x));
	v_2_2 = (-(b_1_1_x*a_2_2_x)+(b_1_2_x*a_2_1_x))/((a_2_1_x*b_2_2_x)-
(a_2_2_x*b_2_1_x));
	v_3_1 = ((b_2_1_y*b_3_2_y)-(b_2_2_y*b_3_1_y))/((a_3_1_y*b_3_2_y)-
(a_3_2_y*b_3_1_y));
	v_3_2 = (-(b_2_1_y*a_3_2_y)+(b_2_2_y*a_3_1_y))/((a_3_1_y*b_3_2_y)-
(a_3_2_y*b_3_1_y));
	v_4_1 = ((b_3_1_x*b_4_2_x)-(b_3_2_x*b_4_1_x))/((a_4_1_x*b_4_2_x)-
(a_4_2_x*b_4_1_x));
	v_4_2 = (-(b_3_1_x*a_4_2_x)+(b_3_2_x*a_4_1_x))/((a_4_1_x*b_4_2_x)-
(a_4_2_x*b_4_1_x));

	phi_1_1 = (u_1_1*u_2_1*u_3_1*u_4_1) + (u_1_1*u_2_2*v_3_1*u_4_1) +
(u_1_1*u_2_1*u_3_2*v_4_1) + (u_1_1*u_2_2*v_3_2*v_4_1) + (v_1_1*u_2_1*u_3_1*u_4_2) +
(v_1_1*u_2_2*v_3_1*u_4_2) + (v_1_1*u_2_1*u_3_2*v_4_2) + (v_1_1*u_2_2*v_3_2*v_4_2);
	phi_2_1 = (u_1_1*v_2_1*u_3_1*u_4_1) + (u_1_1*v_2_2*v_3_1*u_4_1) +
(u_1_1*v_2_1*u_3_2*v_4_1) + (u_1_1*v_2_2*v_3_2*v_4_1) + (v_1_1*v_2_1*u_3_1*u_4_2) +
(v_1_1*v_2_2*v_3_1*u_4_2) + (v_1_1*v_2_1*u_3_2*v_4_2) + (v_1_1*v_2_2*v_3_2*v_4_2);
	phi_1_2 = (u_1_2*u_2_1*u_3_1*u_4_1) + (u_1_2*u_2_2*v_3_1*u_4_1) +
(u_1_2*u_2_1*u_3_2*v_4_1) + (u_1_2*u_2_2*v_3_2*v_4_1) + (v_1_2*u_2_1*u_3_1*u_4_2) +
(v_1_2*u_2_2*v_3_1*u_4_2) + (v_1_2*u_2_1*u_3_2*v_4_2) + (v_1_2*u_2_2*v_3_2*v_4_2);
	phi_2_2 = (u_1_2*v_2_1*u_3_1*u_4_1) + (u_1_2*v_2_2*v_3_1*u_4_1) +
(u_1_2*v_2_1*u_3_2*v_4_1) + (u_1_2*v_2_2*v_3_2*v_4_1) + (v_1_2*v_2_1*u_3_1*u_4_2) +
(v_1_2*v_2_2*v_3_1*u_4_2) + (v_1_2*v_2_1*u_3_2*v_4_2) + (v_1_2*v_2_2*v_3_2*v_4_2);

}
void FAMANI2DSGDiscritization::computeHelpValuesUVAndPhi(double& phi_1_1, double&
phi_2_1, double& phi_1_2, double& phi_2_2,
	double& u_1_1, double& u_1_2, double& u_2_1, double& u_2_2, double& u_3_1, double&
u_3_2, double& u_4_1, double& u_4_2,
	double& v_1_1, double& v_1_2, double& v_2_1, double& v_2_2, double& v_3_1, double&
v_3_2, double& v_4_1, double& v_4_2)
{
	double api = atan(1.0)*4;

	double a_1_1_y = 0.0;
	double a_1_1_x = 0.0;
	double a_1_2_y = 0.0;
	double a_1_2_x = 0.0;
	double a_2_1_y = 0.0;
	double a_2_1_x = 0.0;
	double a_2_2_y = 0.0;
```

```
        double a_2_2_x = 0.0;
        double a_3_1_y = 0.0;
        double a_3_1_x = 0.0;
        double a_3_2_y = 0.0;
        double a_3_2_x = 0.0;
        double a_4_1_y = 0.0;
        double a_4_1_x = 0.0;
        double a_4_2_y = 0.0;
        double a_4_2_x = 0.0;
        double b_1_1_y = 0.0;
        double b_1_1_x = 0.0;
        double b_1_2_y = 0.0;
        double b_1_2_x = 0.0;
        double b_2_1_y = 0.0;
        double b_2_1_x = 0.0;
        double b_2_2_y = 0.0;
        double b_2_2_x = 0.0;
        double b_3_1_y = 0.0;
        double b_3_1_x = 0.0;
        double b_3_2_y = 0.0;
        double b_3_2_x = 0.0;
        double b_4_1_y = 0.0;
        double b_4_1_x = 0.0;
        double b_4_2_y = 0.0;
        double b_4_2_x = 0.0;

        double x_1_y = 1.0;//the below is VERY important: setting for when 'x' or 'y' is
zero, which in turn can cancel the other (which is why it is set to one)
        double x_1_x = 0.0;
        double x_2_y = 1.0;
        double x_2_x = 0.0;
        double x_3_y = 1.0;
        double x_3_x = 0.0;
        double x_4_y = 1.0;
        double x_4_x = 0.0;
        double y_1_y = 0.0;
        double y_1_x = 1.0;
        double y_2_y = 0.0;
        double y_2_x = 1.0;
        double y_3_y = 0.0;
        double y_3_x = 1.0;
        double y_4_y = 0.0;
        double y_4_x = 1.0;

        double x_hat_1_y =
((x_1_y*cos(m_theta_tilde_1))+(y_1_y*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_1);
        double x_hat_1_x =
((x_1_x*cos(m_theta_tilde_1))+(y_1_x*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_1);
        double x_hat_2_y =
((x_2_y*cos(m_theta_tilde_2))+(y_2_y*sin(m_theta_tilde_2)))/sqrt(m_k2_tilde_1);
        double x_hat_2_x =
((x_2_x*cos(m_theta_tilde_2))+(y_2_x*sin(m_theta_tilde_2)))/sqrt(m_k2_tilde_1);
        double x_hat_3_y =
((x_3_y*cos(m_theta_tilde_3))+(y_3_y*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_1);
        double x_hat_3_x =
((x_3_x*cos(m_theta_tilde_3))+(y_3_x*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_1);
        double x_hat_4_y =
((x_4_y*cos(m_theta_tilde_4))+(y_4_y*sin(m_theta_tilde_4)))/sqrt(m_k4_tilde_1);
```

```
	double x_hat_4_x =
((x_4_x*cos(m_theta_tilde_4))+(y_4_x*sin(m_theta_tilde_4)))/sqrt(m_k4_tilde_1);
	double y_hat_1_y = ((y_1_y*cos(m_theta_tilde_1))-
(x_1_y*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_2);
	double y_hat_1_x = ((y_1_x*cos(m_theta_tilde_1))-
(x_1_x*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_2);
	double y_hat_2_y = ((y_2_y*cos(m_theta_tilde_2))-
(x_2_y*sin(m_theta_tilde_2)))/sqrt(m_k2_tilde_2);
	double y_hat_2_x = ((y_2_x*cos(m_theta_tilde_2))-
(x_2_x*sin(m_theta_tilde_2)))/sqrt(m_k2_tilde_2);
	double y_hat_3_y = ((y_3_y*cos(m_theta_tilde_3))-
(x_3_y*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_2);
	double y_hat_3_x = ((y_3_x*cos(m_theta_tilde_3))-
(x_3_x*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_2);
	double y_hat_4_y = ((y_4_y*cos(m_theta_tilde_4))-
(x_4_y*sin(m_theta_tilde_4)))/sqrt(m_k4_tilde_2);
	double y_hat_4_x = ((y_4_x*cos(m_theta_tilde_4))-
(x_4_x*sin(m_theta_tilde_4)))/sqrt(m_k4_tilde_2);


	double theta_hat_1_y = (0)-m_theta_tilde_1;
	double theta_hat_1_x = (0.5*api)-m_theta_tilde_1;
	double theta_hat_2_y = (api)-m_theta_tilde_2;
	double theta_hat_2_x = (0.5*api)-m_theta_tilde_2;
	double theta_hat_3_y = (-api)-m_theta_tilde_3;
	double theta_hat_3_x = (-0.5*api)-m_theta_tilde_3;
	double theta_hat_4_y = (0)-m_theta_tilde_4;
	double theta_hat_4_x = (-0.5*api)-m_theta_tilde_4;
	double radius_1_y = sqrt((x_hat_1_y*x_hat_1_y)+(y_hat_1_y*y_hat_1_y));
	double radius_1_x = sqrt((x_hat_1_x*x_hat_1_x)+(y_hat_1_x*y_hat_1_x));
	double radius_2_y = sqrt((x_hat_2_y*x_hat_2_y)+(y_hat_2_y*y_hat_2_y));
	double radius_2_x = sqrt((x_hat_2_x*x_hat_2_x)+(y_hat_2_x*y_hat_2_x));
	double radius_3_y = sqrt((x_hat_3_y*x_hat_3_y)+(y_hat_3_y*y_hat_3_y));
	double radius_3_x = sqrt((x_hat_3_x*x_hat_3_x)+(y_hat_3_x*y_hat_3_x));
	double radius_4_y = sqrt((x_hat_4_y*x_hat_4_y)+(y_hat_4_y*y_hat_4_y));
	double radius_4_x = sqrt((x_hat_4_x*x_hat_4_x)+(y_hat_4_x*y_hat_4_x));

	double dx_dx_1_y = cos(m_theta_tilde_1)/sqrt(m_k1_tilde_1);
	double dx_dx_1_x = cos(m_theta_tilde_1)/sqrt(m_k1_tilde_1);
	double dx_dx_2_y = cos(m_theta_tilde_2)/sqrt(m_k2_tilde_1);
	double dx_dx_2_x = cos(m_theta_tilde_2)/sqrt(m_k2_tilde_1);
	double dx_dx_3_y = cos(m_theta_tilde_3)/sqrt(m_k3_tilde_1);
	double dx_dx_3_x = cos(m_theta_tilde_3)/sqrt(m_k3_tilde_1);
	double dx_dx_4_y = cos(m_theta_tilde_4)/sqrt(m_k4_tilde_1);
	double dx_dx_4_x = cos(m_theta_tilde_4)/sqrt(m_k4_tilde_1);
	double dy_dy_1_y = cos(m_theta_tilde_1)/sqrt(m_k1_tilde_2);
	double dy_dy_1_x = cos(m_theta_tilde_1)/sqrt(m_k1_tilde_2);
	double dy_dy_2_y = cos(m_theta_tilde_2)/sqrt(m_k2_tilde_2);
	double dy_dy_2_x = cos(m_theta_tilde_2)/sqrt(m_k2_tilde_2);
	double dy_dy_3_y = cos(m_theta_tilde_3)/sqrt(m_k3_tilde_2);
	double dy_dy_3_x = cos(m_theta_tilde_3)/sqrt(m_k3_tilde_2);
	double dy_dy_4_y = cos(m_theta_tilde_4)/sqrt(m_k4_tilde_2);
	double dy_dy_4_x = cos(m_theta_tilde_4)/sqrt(m_k4_tilde_2);

	double dtheta_dx_1_y = (-y_hat_1_y/(pow(radius_1_y, 2)))*(1-m_alpha_FAM);
	double dtheta_dx_1_x = (-y_hat_1_x/(pow(radius_1_x, 2)))*(1-m_alpha_FAM);
	double dtheta_dx_2_y = (-y_hat_2_y/(pow(radius_2_y, 2)))*(1-m_alpha_FAM);
	double dtheta_dx_2_x = (-y_hat_2_x/(pow(radius_2_x, 2)))*(1-m_alpha_FAM);
```

```
	double dtheta_dx_3_y = (-y_hat_3_y/(pow(radius_3_y, 2)))*(1-m_alpha_FAM);
	double dtheta_dx_3_x = (-y_hat_3_x/(pow(radius_3_x, 2)))*(1-m_alpha_FAM);
	double dtheta_dx_4_y = (-y_hat_4_y/(pow(radius_4_y, 2)))*(1-m_alpha_FAM);
	double dtheta_dx_4_x = (-y_hat_4_x/(pow(radius_4_x, 2)))*(1-m_alpha_FAM);
	double dtheta_dy_1_y = (x_hat_1_y/(pow(radius_1_y, 2)))*(1-m_alpha_FAM);
	double dtheta_dy_1_x = (x_hat_1_x/(pow(radius_1_x, 2)))*(1-m_alpha_FAM);
	double dtheta_dy_2_y = (x_hat_2_y/(pow(radius_2_y, 2)))*(1-m_alpha_FAM);
	double dtheta_dy_2_x = (x_hat_2_x/(pow(radius_2_x, 2)))*(1-m_alpha_FAM);
	double dtheta_dy_3_y = (x_hat_3_y/(pow(radius_3_y, 2)))*(1-m_alpha_FAM);
	double dtheta_dy_3_x = (x_hat_3_x/(pow(radius_3_x, 2)))*(1-m_alpha_FAM);
	double dtheta_dy_4_y = (x_hat_4_y/(pow(radius_4_y, 2)))*(1-m_alpha_FAM);
	double dtheta_dy_4_x = (x_hat_4_x/(pow(radius_4_x, 2)))*(1-m_alpha_FAM);

	double dr_dx_1_y = (x_hat_1_y/radius_1_y)*(pow(radius_1_y, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dx_1_x = (x_hat_1_x/radius_1_x)*(pow(radius_1_x, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dx_2_y = (x_hat_2_y/radius_2_y)*(pow(radius_2_y, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dx_2_x = (x_hat_2_x/radius_2_x)*(pow(radius_2_x, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dx_3_y = (x_hat_3_y/radius_3_y)*(pow(radius_3_y, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dx_3_x = (x_hat_3_x/radius_3_x)*(pow(radius_3_x, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dx_4_y = (x_hat_4_y/radius_4_y)*(pow(radius_4_y, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dx_4_x = (x_hat_4_x/radius_4_x)*(pow(radius_4_x, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dy_1_y = (y_hat_1_y/radius_1_y)*(pow(radius_1_y, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dy_1_x = (y_hat_1_x/radius_1_x)*(pow(radius_1_x, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dy_2_y = (y_hat_2_y/radius_2_y)*(pow(radius_2_y, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dy_2_x = (y_hat_2_x/radius_2_x)*(pow(radius_2_x, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dy_3_y = (y_hat_3_y/radius_3_y)*(pow(radius_3_y, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dy_3_x = (y_hat_3_x/radius_3_x)*(pow(radius_3_x, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dy_4_y = (y_hat_4_y/radius_4_y)*(pow(radius_4_y, -m_alpha_FAM))*(1-
m_alpha_FAM);
	double dr_dy_4_x = (y_hat_4_x/radius_4_x)*(pow(radius_4_x, -m_alpha_FAM))*(1-
m_alpha_FAM);

	a_1_1_y = (pow(radius_1_y, 1-m_alpha_FAM))*cos(theta_hat_1_y*(1-m_alpha_FAM));
	a_1_1_x = (pow(radius_1_x, 1-m_alpha_FAM))*cos(theta_hat_1_x*(1-m_alpha_FAM));
	a_2_1_y = (pow(radius_2_y, 1-m_alpha_FAM))*cos((theta_hat_2_y-(0.5*api))*(1-
m_alpha_FAM));
	a_2_1_x = (pow(radius_2_x, 1-m_alpha_FAM))*cos((theta_hat_2_x-(0.5*api))*(1-
m_alpha_FAM));
	a_3_1_y = (pow(radius_3_y, 1-m_alpha_FAM))*cos((theta_hat_3_y+api)*(1-
m_alpha_FAM));//The plus or minus PI has to do with the direction of orientation. Seeing
as how our calcualtion go from -PI to +PI,
	a_3_1_x = (pow(radius_3_x, 1-m_alpha_FAM))*cos((theta_hat_3_x+api)*(1-
m_alpha_FAM));//when calculating from the positive side (2 quadrant), one must have +PI,
and vice versa (applied one all 3 quadrant equations)
```

```
	a_4_1_y = (pow(radius_4_y, 1-m_alpha_FAM))*cos((theta_hat_4_y+(0.5*api))*(1-
m_alpha_FAM));
	a_4_1_x = (pow(radius_4_x, 1-m_alpha_FAM))*cos((theta_hat_4_x+(0.5*api))*(1-
m_alpha_FAM));

	b_1_1_y = (pow(radius_1_y, 1-m_alpha_FAM))*sin(theta_hat_1_y*(1-m_alpha_FAM));
	b_1_1_x = (pow(radius_1_x, 1-m_alpha_FAM))*sin(theta_hat_1_x*(1-m_alpha_FAM));
	b_2_1_y = -(pow(radius_2_y, 1-m_alpha_FAM))*sin((theta_hat_2_y-(0.5*api))*(1-
m_alpha_FAM));
	b_2_1_x = -(pow(radius_2_x, 1-m_alpha_FAM))*sin((theta_hat_2_x-(0.5*api))*(1-
m_alpha_FAM));
	b_3_1_y = -(pow(radius_3_y, 1-m_alpha_FAM))*sin((theta_hat_3_y+api)*(1-
m_alpha_FAM));
	b_3_1_x = -(pow(radius_3_x, 1-m_alpha_FAM))*sin((theta_hat_3_x+api)*(1-
m_alpha_FAM));
	b_4_1_y = (pow(radius_4_y, 1-m_alpha_FAM))*sin((theta_hat_4_y+(0.5*api))*(1-
m_alpha_FAM));
	b_4_1_x = (pow(radius_4_x, 1-m_alpha_FAM))*sin((theta_hat_4_x+(0.5*api))*(1-
m_alpha_FAM));

	a_1_2_y = (m_k1_1_2*dx_dx_1_y*((dr_dx_1_y*cos(theta_hat_1_y*(1-m_alpha_FAM)))+(-
dtheta_dx_1_y*(pow(radius_1_y, 1-m_alpha_FAM))*sin(theta_hat_1_y*(1-m_alpha_FAM)))))
			+(m_k1_2_2*dy_dy_1_y*((dr_dy_1_y*cos(theta_hat_1_y*(1-
m_alpha_FAM)))+(-dtheta_dy_1_y*(pow(radius_1_y, 1-m_alpha_FAM))*sin(theta_hat_1_y*(1-
m_alpha_FAM)))));
	a_1_2_x = (m_k1_1_1*dx_dx_1_x*((dr_dx_1_x*cos(theta_hat_1_x*(1-m_alpha_FAM)))+(-
dtheta_dx_1_x*(pow(radius_1_x, 1-m_alpha_FAM))*sin(theta_hat_1_x*(1-m_alpha_FAM)))))
			+(m_k1_1_2*dy_dy_1_x*((dr_dy_1_x*cos(theta_hat_1_x*(1-
m_alpha_FAM)))+(-dtheta_dy_1_x*(pow(radius_1_x, 1-m_alpha_FAM))*sin(theta_hat_1_x*(1-
m_alpha_FAM)))));
	a_2_2_y = (m_k2_1_2*dx_dx_2_y*((dr_dx_2_y*cos((theta_hat_2_y-(0.5*api))*(1-
m_alpha_FAM)))+(-dtheta_dx_2_y*(pow(radius_2_y, 1-m_alpha_FAM))*sin((theta_hat_2_y-
(0.5*api))*(1-m_alpha_FAM)))))
			+(m_k2_2_2*dy_dy_2_y*((dr_dy_2_y*cos((theta_hat_2_y-(0.5*api))*(1-
m_alpha_FAM)))+(-dtheta_dy_2_y*(pow(radius_2_y, 1-m_alpha_FAM))*sin((theta_hat_2_y-
(0.5*api))*(1-m_alpha_FAM)))));
	a_2_2_x = (m_k2_1_1*dx_dx_2_x*((dr_dx_2_x*cos((theta_hat_2_x-(0.5*api))*(1-
m_alpha_FAM)))+(-dtheta_dx_2_x*(pow(radius_2_x, 1-m_alpha_FAM))*sin((theta_hat_2_x-
(0.5*api))*(1-m_alpha_FAM)))))
			+(m_k2_1_2*dy_dy_2_x*((dr_dy_2_x*cos((theta_hat_2_x-(0.5*api))*(1-
m_alpha_FAM)))+(-dtheta_dy_2_x*(pow(radius_2_x, 1-m_alpha_FAM))*sin((theta_hat_2_x-
(0.5*api))*(1-m_alpha_FAM)))));
	a_3_2_y = (m_k3_1_2*dx_dx_3_y*((dr_dx_3_y*cos((theta_hat_3_y+api)*(1-
m_alpha_FAM)))+(-dtheta_dx_3_y*(pow(radius_3_y, 1-
m_alpha_FAM))*sin((theta_hat_3_y+api)*(1-m_alpha_FAM)))))
			+(m_k3_2_2*dy_dy_3_y*((dr_dy_3_y*cos((theta_hat_3_y+api)*(1-
m_alpha_FAM)))+(-dtheta_dy_3_y*(pow(radius_3_y, 1-
m_alpha_FAM))*sin((theta_hat_3_y+api)*(1-m_alpha_FAM)))));
	a_3_2_x = (m_k3_1_1*dx_dx_3_x*((dr_dx_3_x*cos((theta_hat_3_x+api)*(1-
m_alpha_FAM)))+(-dtheta_dx_3_x*(pow(radius_3_x, 1-
m_alpha_FAM))*sin((theta_hat_3_x+api)*(1-m_alpha_FAM)))))
			+(m_k3_1_2*dy_dy_3_x*((dr_dy_3_x*cos((theta_hat_3_x+api)*(1-
m_alpha_FAM)))+(-dtheta_dy_3_x*(pow(radius_3_x, 1-
m_alpha_FAM))*sin((theta_hat_3_x+api)*(1-m_alpha_FAM)))));
	a_4_2_y = (m_k4_1_2*dx_dx_4_y*((dr_dx_4_y*cos((theta_hat_4_y+(0.5*api))*(1-
m_alpha_FAM)))+(-dtheta_dx_4_y*(pow(radius_4_y, 1-
m_alpha_FAM))*sin((theta_hat_4_y+(0.5*api))*(1-m_alpha_FAM)))))
```

```
                    +(m_k4_2_2*dy_dy_4_y*((dr_dy_4_y*cos((theta_hat_4_y+(0.5*api))*(1-
m_alpha_FAM)))+(-dtheta_dy_4_y*(pow(radius_4_y, 1-
m_alpha_FAM))*sin((theta_hat_4_y+(0.5*api))*(1-m_alpha_FAM)))));
      a_4_2_x = (m_k4_1_1*dx_dx_4_x*((dr_dx_4_x*cos((theta_hat_4_x+(0.5*api))*(1-
m_alpha_FAM)))+(-dtheta_dx_4_x*(pow(radius_4_x, 1-
m_alpha_FAM))*sin((theta_hat_4_x+(0.5*api))*(1-m_alpha_FAM))))
                    +(m_k4_1_2*dy_dy_4_x*((dr_dy_4_x*cos((theta_hat_4_x+(0.5*api))*(1-
m_alpha_FAM)))+(-dtheta_dy_4_x*(pow(radius_4_x, 1-
m_alpha_FAM))*sin((theta_hat_4_x+(0.5*api))*(1-m_alpha_FAM)))));

      b_1_2_y = (m_k1_1_2*dx_dx_1_y*((dr_dx_1_y*sin(theta_hat_1_y*(1-
m_alpha_FAM)))+(dtheta_dx_1_y*(pow(radius_1_y, 1-m_alpha_FAM))*cos(theta_hat_1_y*(1-
m_alpha_FAM))))
                    +(m_k1_2_2*dy_dy_1_y*((dr_dy_1_y*sin(theta_hat_1_y*(1-
m_alpha_FAM)))+(dtheta_dy_1_y*(pow(radius_1_y, 1-m_alpha_FAM))*cos(theta_hat_1_y*(1-
m_alpha_FAM)))));
      b_1_2_x = (m_k1_1_1*dx_dx_1_x*((dr_dx_1_x*sin(theta_hat_1_x*(1-
m_alpha_FAM)))+(dtheta_dx_1_x*(pow(radius_1_x, 1-m_alpha_FAM))*cos(theta_hat_1_x*(1-
m_alpha_FAM))))
                    +(m_k1_1_2*dy_dy_1_x*((dr_dy_1_x*sin(theta_hat_1_x*(1-
m_alpha_FAM)))+(dtheta_dy_1_x*(pow(radius_1_x, 1-m_alpha_FAM))*cos(theta_hat_1_x*(1-
m_alpha_FAM)))));
      b_2_2_y = -((m_k2_1_2*dx_dx_2_y*((dr_dx_2_y*sin((theta_hat_2_y-(0.5*api))*(1-
m_alpha_FAM)))+(dtheta_dx_2_y*(pow(radius_2_y, 1-m_alpha_FAM))*cos((theta_hat_2_y-
(0.5*api))*(1-m_alpha_FAM)))))
                     +(m_k2_2_2*dy_dy_2_y*((dr_dy_2_y*sin((theta_hat_2_y-
(0.5*api))*(1-m_alpha_FAM)))+(dtheta_dy_2_y*(pow(radius_2_y, 1-
m_alpha_FAM))*cos((theta_hat_2_y-(0.5*api))*(1-m_alpha_FAM))))));
      b_2_2_x = -((m_k2_1_1*dx_dx_2_x*((dr_dx_2_x*sin((theta_hat_2_x-(0.5*api))*(1-
m_alpha_FAM)))+(dtheta_dx_2_x*(pow(radius_2_x, 1-m_alpha_FAM))*cos((theta_hat_2_x-
(0.5*api))*(1-m_alpha_FAM)))))
                     +(m_k2_1_2*dy_dy_2_x*((dr_dy_2_x*sin((theta_hat_2_x-
(0.5*api))*(1-m_alpha_FAM)))+(dtheta_dy_2_x*(pow(radius_2_x, 1-
m_alpha_FAM))*cos((theta_hat_2_x-(0.5*api))*(1-m_alpha_FAM))))));
      b_3_2_y = -((m_k3_1_2*dx_dx_3_y*((dr_dx_3_y*sin((theta_hat_3_y+api)*(1-
m_alpha_FAM)))+(dtheta_dx_3_y*(pow(radius_3_y, 1-
m_alpha_FAM))*cos((theta_hat_3_y+api)*(1-m_alpha_FAM)))))
                     +(m_k3_2_2*dy_dy_3_y*((dr_dy_3_y*sin((theta_hat_3_y+api)*(1-
m_alpha_FAM)))+(dtheta_dy_3_y*(pow(radius_3_y, 1-
m_alpha_FAM))*cos((theta_hat_3_y+api)*(1-m_alpha_FAM))))));
      b_3_2_x = -((m_k3_1_1*dx_dx_3_x*((dr_dx_3_x*sin((theta_hat_3_x+api)*(1-
m_alpha_FAM)))+(dtheta_dx_3_x*(pow(radius_3_x, 1-
m_alpha_FAM))*cos((theta_hat_3_x+api)*(1-m_alpha_FAM)))))
                     +(m_k3_1_2*dy_dy_3_x*((dr_dy_3_x*sin((theta_hat_3_x+api)*(1-
m_alpha_FAM)))+(dtheta_dy_3_x*(pow(radius_3_x, 1-
m_alpha_FAM))*cos((theta_hat_3_x+api)*(1-m_alpha_FAM))))));
      b_4_2_y = (m_k4_1_2*dx_dx_4_y*((dr_dx_4_y*sin((theta_hat_4_y+(0.5*api))*(1-
m_alpha_FAM)))+(dtheta_dx_4_y*(pow(radius_4_y, 1-
m_alpha_FAM))*cos((theta_hat_4_y+(0.5*api))*(1-m_alpha_FAM)))))
                    +(m_k4_2_2*dy_dy_4_y*((dr_dy_4_y*sin((theta_hat_4_y+(0.5*api))*(1-
m_alpha_FAM)))+(dtheta_dy_4_y*(pow(radius_4_y, 1-
m_alpha_FAM))*cos((theta_hat_4_y+(0.5*api))*(1-m_alpha_FAM)))));
      b_4_2_x = (m_k4_1_1*dx_dx_4_x*((dr_dx_4_x*sin((theta_hat_4_x+(0.5*api))*(1-
m_alpha_FAM)))+(dtheta_dx_4_x*(pow(radius_4_x, 1-
m_alpha_FAM))*cos((theta_hat_4_x+(0.5*api))*(1-m_alpha_FAM)))))
                    +(m_k4_1_2*dy_dy_4_x*((dr_dy_4_x*sin((theta_hat_4_x+(0.5*api))*(1-
m_alpha_FAM)))+(dtheta_dy_4_x*(pow(radius_4_x, 1-
m_alpha_FAM))*cos((theta_hat_4_x+(0.5*api))*(1-m_alpha_FAM)))));
```

```
	u_1_1 = ((a_4_1_y*b_1_2_y)-(a_4_2_y*b_1_1_y))/((a_1_1_y*b_1_2_y)-
(a_1_2_y*b_1_1_y));
	u_1_2 = (-(a_4_1_y*a_1_2_y)+(a_4_2_y*a_1_1_y))/((a_1_1_y*b_1_2_y)-
(a_1_2_y*b_1_1_y));
	u_2_1 = ((a_1_1_x*b_2_2_x)-(a_1_2_x*b_2_1_x))/((a_2_1_x*b_2_2_x)-
(a_2_2_x*b_2_1_x));
	u_2_2 = (-(a_1_1_x*a_2_2_x)+(a_1_2_x*a_2_1_x))/((a_2_1_x*b_2_2_x)-
(a_2_2_x*b_2_1_x));
	u_3_1 = ((a_2_1_y*b_3_2_y)-(a_2_2_y*b_3_1_y))/((a_3_1_y*b_3_2_y)-
(a_3_2_y*b_3_1_y));
	u_3_2 = (-(a_2_1_y*a_3_2_y)+(a_2_2_y*a_3_1_y))/((a_3_1_y*b_3_2_y)-
(a_3_2_y*b_3_1_y));
	u_4_1 = ((a_3_1_x*b_4_2_x)-(a_3_2_x*b_4_1_x))/((a_4_1_x*b_4_2_x)-
(a_4_2_x*b_4_1_x));
	u_4_2 = (-(a_3_1_x*a_4_2_x)+(a_3_2_x*a_4_1_x))/((a_4_1_x*b_4_2_x)-
(a_4_2_x*b_4_1_x));

	v_1_1 = ((b_4_1_y*b_1_2_y)-(b_4_2_y*b_1_1_y))/((a_1_1_y*b_1_2_y)-
(a_1_2_y*b_1_1_y));
	v_1_2 = (-(b_4_1_y*a_1_2_y)+(b_4_2_y*a_1_1_y))/((a_1_1_y*b_1_2_y)-
(a_1_2_y*b_1_1_y));
	v_2_1 = ((b_1_1_x*b_2_2_x)-(b_1_2_x*b_2_1_x))/((a_2_1_x*b_2_2_x)-
(a_2_2_x*b_2_1_x));
	v_2_2 = (-(b_1_1_x*a_2_2_x)+(b_1_2_x*a_2_1_x))/((a_2_1_x*b_2_2_x)-
(a_2_2_x*b_2_1_x));
	v_3_1 = ((b_2_1_y*b_3_2_y)-(b_2_2_y*b_3_1_y))/((a_3_1_y*b_3_2_y)-
(a_3_2_y*b_3_1_y));
	v_3_2 = (-(b_2_1_y*a_3_2_y)+(b_2_2_y*a_3_1_y))/((a_3_1_y*b_3_2_y)-
(a_3_2_y*b_3_1_y));
	v_4_1 = ((b_3_1_x*b_4_2_x)-(b_3_2_x*b_4_1_x))/((a_4_1_x*b_4_2_x)-
(a_4_2_x*b_4_1_x));
	v_4_2 = (-(b_3_1_x*a_4_2_x)+(b_3_2_x*a_4_1_x))/((a_4_1_x*b_4_2_x)-
(a_4_2_x*b_4_1_x));

	phi_1_1 = (u_1_1*u_2_1*u_3_1*u_4_1) + (u_1_1*u_2_2*v_3_1*u_4_1) +
(u_1_1*u_2_1*u_3_2*v_4_1) + (u_1_1*u_2_2*v_3_2*v_4_1) + (v_1_1*u_2_1*u_3_1*u_4_2) +
(v_1_1*u_2_2*v_3_1*u_4_2) + (v_1_1*u_2_1*u_3_2*v_4_2) + (v_1_1*u_2_2*v_3_2*v_4_2);
	phi_2_1 = (u_1_1*v_2_1*u_3_1*u_4_1) + (u_1_1*v_2_2*v_3_1*u_4_1) +
(u_1_1*v_2_1*u_3_2*v_4_1) + (u_1_1*v_2_2*v_3_2*v_4_1) + (v_1_1*v_2_1*u_3_1*u_4_2) +
(v_1_1*v_2_2*v_3_1*u_4_2) + (v_1_1*v_2_1*u_3_2*v_4_2) + (v_1_1*v_2_2*v_3_2*v_4_2);
	phi_1_2 = (u_1_2*u_2_1*u_3_1*u_4_1) + (u_1_2*u_2_2*v_3_1*u_4_1) +
(u_1_2*u_2_1*u_3_2*v_4_1) + (u_1_2*u_2_2*v_3_2*v_4_1) + (v_1_2*u_2_1*u_3_1*u_4_2) +
(v_1_2*u_2_2*v_3_1*u_4_2) + (v_1_2*u_2_1*u_3_2*v_4_2) + (v_1_2*u_2_2*v_3_2*v_4_2);
	phi_2_2 = (u_1_2*v_2_1*u_3_1*u_4_1) + (u_1_2*v_2_2*v_3_1*u_4_1) +
(u_1_2*v_2_1*u_3_2*v_4_1) + (u_1_2*v_2_2*v_3_2*v_4_1) + (v_1_2*v_2_1*u_3_1*u_4_2) +
(v_1_2*v_2_2*v_3_1*u_4_2) + (v_1_2*v_2_1*u_3_2*v_4_2) + (v_1_2*v_2_2*v_3_2*v_4_2);

}
void FAMANI2DSGDiscritization::computeAlphaAndC(const Pair<int,int> Pqhat, const int
ihat, const bool atBoundary)
{
	if (!atBoundary)
	{

		//*********************************************//
```

```
                //This (under) must probably be changed when isotropic testing is
finished//
                m_alpha_FAM = NewtonSecant();

                m_C_FAM = ComputeC();
                //*********************************************//

                cout << "alpha = " << m_alpha_FAM << endl;

                //$$$$$$$$$$$$$$$ JUST A TEMPORARY SOLUTION $$$$$$$$$$$$$$$$$$
                m_useFAMScheme = (fabs(m_alpha_FAM) < 1.0) ? true : false;

        }
        else
        {
                //NOT NECCESARY!!!!
        }
}
realS FAMANI2DSGDiscritization::computeRHSContribution(const int ihat, const int khat,
const int qhat, const ArrayS<realS>& g)
{
        double ret = 0.0;
        Pair<int,int> PQ(0, qhat);
        Pair<int,int> PK(1, khat);
        Pair<int,int> PI(2, ihat);
        double delta_x = 0.0;
        double delta_y = 0.0;

        RnPoint XQ = m_C.u_getCoordinate_0u(PQ);
        double XQ_x = XQ.u_getElement_0u(0);
        double XQ_y = XQ.u_getElement_0u(1);
        RnPoint XI = m_C.u_getCoordinate_0u(PI);
        double XI_x = XI.u_getElement_0u(0);
        double XI_y = XI.u_getElement_0u(1);

        delta_x = fabs(XI_x - XQ_x);// Half actual delta_x in the grid!!!!
        delta_y = fabs(XI_y - XQ_y);// Half actual delta_y in the grid!!!!


        RnPoint normalVec = m_C.u_getNormalVector_0u(PI, PK);

        if (m_C.u_boundaryIndex_0u(PK))
        {
                if ((fabs(normalVec.u_getElement_0u(1)) <= m_Ftol) ? (ret =
(delta_y/delta_x)*m_permxx[ihat]*g[m_C.u_findBoundaryPosition_0u(PK)]) : (ret =
(delta_x/delta_y)*m_permxx[ihat]*g[m_C.u_findBoundaryPosition_0u(PK)]));
        }

        return (-1.0)*ret;
}
realS FAMANI2DSGDiscritization::AlphaFunction(const double& alpha)
{
        double api = atan(1.0)*4;
        double alpha_func_result = 0.0;

        double phi_1_1 = 0.0;
        double phi_2_1 = 0.0;
        double phi_1_2 = 0.0;
```

```
	double phi_2_2 = 0.0;
	double u_1_1 = 0.0;
	double u_1_2 = 0.0;
	double u_2_1 = 0.0;
	double u_2_2 = 0.0;
	double u_3_1 = 0.0;
	double u_3_2 = 0.0;
	double u_4_1 = 0.0;
	double u_4_2 = 0.0;
	double v_1_1 = 0.0;
	double v_1_2 = 0.0;
	double v_2_1 = 0.0;
	double v_2_2 = 0.0;
	double v_3_1 = 0.0;
	double v_3_2 = 0.0;
	double v_4_1 = 0.0;
	double v_4_2 = 0.0;

	computeHelpValuesUVAndPhiForAlpha(phi_1_1, phi_2_1, phi_1_2, phi_2_2, u_1_1,
u_1_2, u_2_1, u_2_2, u_3_1, u_3_2, u_4_1, u_4_2, v_1_1, v_1_2, v_2_1, v_2_2, v_3_1,
v_3_2, v_4_1, v_4_2, alpha);

	alpha_func_result = (phi_1_1*phi_2_2) - (phi_1_2*phi_2_1) - phi_1_1 - phi_2_2 +
1.0;

	return alpha_func_result;
}
realS FAMANI2DSGDiscritization::NewtonSecant()
{
	double alpha_nm1 = 0.1;//m_Ftol;
	double alpha_n = 1.0-0.101;//m_Ftol;
	double F_derive = 0.0;
	double F_func = 0.0;
	bool converged = false;

	while (!converged)
	{
		F_func = AlphaFunction(alpha_n);
		F_derive = (F_func - AlphaFunction(alpha_nm1))/(alpha_n - alpha_nm1);
//Secant appox.
		cout << "AlphaFunction(alpha_nm1) = " << AlphaFunction(alpha_nm1) << endl;
		alpha_nm1 = alpha_n;
		alpha_n -= (F_func/F_derive); //Newton update
		cout << "alpha_n = " << alpha_n << " F_func = " << F_func << " F_derive = "
<< F_derive << " F_func/F_derive = " << F_func/F_derive << endl;
		converged = (fabs(alpha_n - alpha_nm1) < m_Ftol) ? true : false;
//Convergence requirement
	}

	return alpha_n;
}
realS FAMANI2DSGDiscritization::ComputeC ()
{
	double api = atan(1.0)*4;
	double c_from_phi = 0.0;

	double phi_1_1 = 0.0;
	double phi_2_1 = 0.0;
```

```
	double phi_1_2 = 0.0;
	double phi_2_2 = 0.0;
	double u_1_1 = 0.0;
	double u_1_2 = 0.0;
	double u_2_1 = 0.0;
	double u_2_2 = 0.0;
	double u_3_1 = 0.0;
	double u_3_2 = 0.0;
	double u_4_1 = 0.0;
	double u_4_2 = 0.0;
	double v_1_1 = 0.0;
	double v_1_2 = 0.0;
	double v_2_1 = 0.0;
	double v_2_2 = 0.0;
	double v_3_1 = 0.0;
	double v_3_2 = 0.0;
	double v_4_1 = 0.0;
	double v_4_2 = 0.0;

	computeHelpValuesUVAndPhi(phi_1_1, phi_2_1, phi_1_2, phi_2_2, u_1_1, u_1_2, u_2_1,
u_2_2, u_3_1, u_3_2, u_4_1, u_4_2, v_1_1, v_1_2, v_2_1, v_2_2, v_3_1, v_3_2, v_4_1,
v_4_2);

	c_from_phi = (phi_1_1 - 1)/(-phi_2_1);
	//c_from_phi = (phi_1_2)/(1 - phi_2_2);

	return c_from_phi;
}
void FAMANI2DSGDiscritization::computeLambdas(const int ihat, const int itilde, const int
khat, const int qhat, double& lambda_c, double& lambda_w, double& lambda_sw, double&
lambda_s, double& lambda_c_x, double& lambda_c_y, double& lambda_sw_x, double&
lambda_sw_y)
{
	double api = atan(1.0)*4;
	double theta_xycoord = 0.0;

	double flux_part_1_1 = 0.0;
	double flux_part_1_2 = 0.0;
	double flux_part_2_1 = 0.0;
	double flux_part_2_2 = 0.0;

	Pair<int,int> PQ(0, qhat);
    Pair<int,int> PI(2, ihat);
	Pair<int,int> PK(1, khat);

	RnPoint XQ = m_C.u_getCoordinate_0u(PQ);
	double XQ_x = XQ.u_getElement_0u(0);
	double XQ_y = XQ.u_getElement_0u(1);
	RnPoint XI = m_C.u_getCoordinate_0u(PI);
	double XI_x = XI.u_getElement_0u(0);
	double XI_y = XI.u_getElement_0u(1);
	RnPoint XK = m_C.u_getCoordinate_0u(PK);
	double XK_x = XK.u_getElement_0u(0);
	double XK_y = XK.u_getElement_0u(1);

	double delta_x = fabs(XI_x - XQ_x);// Half actual delta_x in the grid
	double delta_y = fabs(XI_y - XQ_y);// Half actual delta_y in the grid,
```

```
	double x_hat_1 =
((delta_x*cos(m_theta_tilde_1))+(delta_y*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_1);
	double y_hat_1 =
((delta_y*cos(m_theta_tilde_1))+(delta_x*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_2);
	double radius_hat_1 = sqrt((x_hat_1*x_hat_1) + (y_hat_1*y_hat_1));
	double theta_hat_1 = (api/4) - m_theta_tilde_1;

	double x_hat_2 =
((delta_x*cos(m_theta_tilde_2))+(delta_y*sin(m_theta_tilde_2)))/sqrt(m_k2_tilde_1);
	double y_hat_2 =
((delta_y*cos(m_theta_tilde_2))+(delta_x*sin(m_theta_tilde_2)))/sqrt(m_k2_tilde_2);
	double radius_hat_2 = sqrt((x_hat_2*x_hat_2) + (y_hat_2*y_hat_2));
	double theta_hat_2 = (3*api/4) - m_theta_tilde_2;

	double x_hat_3 =
((delta_x*cos(m_theta_tilde_3))+(delta_y*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_1);
	double y_hat_3 =
((delta_y*cos(m_theta_tilde_3))+(delta_x*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_2);
	double radius_hat_3 = sqrt((x_hat_3*x_hat_3) + (y_hat_3*y_hat_3));
	double theta_hat_3 = (-3*api/4) - m_theta_tilde_3;

	double x_hat_4 =
((delta_x*cos(m_theta_tilde_4))+(delta_y*sin(m_theta_tilde_4)))/sqrt(m_k4_tilde_1);
	double y_hat_4 =
((delta_y*cos(m_theta_tilde_4))+(delta_x*sin(m_theta_tilde_4)))/sqrt(m_k4_tilde_2);
	double radius_hat_4 = sqrt((x_hat_4*x_hat_4) + (y_hat_4*y_hat_4));
	double theta_hat_4 = (-api/4) - m_theta_tilde_4;


	double phi_1_1 = 0.0;
	double phi_2_1 = 0.0;
	double phi_1_2 = 0.0;
	double phi_2_2 = 0.0;
	double u_1_1 = 0.0;
	double u_1_2 = 0.0;
	double u_2_1 = 0.0;
	double u_2_2 = 0.0;
	double u_3_1 = 0.0;
	double u_3_2 = 0.0;
	double u_4_1 = 0.0;
	double u_4_2 = 0.0;
	double v_1_1 = 0.0;
	double v_1_2 = 0.0;
	double v_2_1 = 0.0;
	double v_2_2 = 0.0;
	double v_3_1 = 0.0;
	double v_3_2 = 0.0;
	double v_4_1 = 0.0;
	double v_4_2 = 0.0;

	computeHelpValuesUVAndPhi(phi_1_1, phi_2_1, phi_1_2, phi_2_2, u_1_1, u_1_2, u_2_1,
u_2_2, u_3_1, u_3_2, u_4_1, u_4_2, v_1_1, v_1_2, v_2_1, v_2_2, v_3_1, v_3_2, v_4_1,
v_4_2);

	double A_1 = 1.0;
	double B_1 = A_1*m_C_FAM;

	double A_2 = (A_1*u_2_1) + (B_1*v_2_1);
```

```
	double B_2 = (A_1*u_2_2) + (B_1*v_2_2);
	double A_3 = (A_2*u_3_1) + (B_2*v_3_1);
	double B_3 = (A_2*u_3_2) + (B_2*v_3_2);
	double A_4 = (A_3*u_4_1) + (B_3*v_4_1);
	double B_4 = (A_3*u_4_2) + (B_3*v_4_2);


	lambda_c = (A_1*(pow(radius_hat_1, 1-m_alpha_FAM))*cos(theta_hat_1*(1-
m_alpha_FAM))) + (B_1*(pow(radius_hat_1, 1-m_alpha_FAM))*sin(theta_hat_1*(1-
m_alpha_FAM)));
	lambda_w = (A_2*(pow(radius_hat_2, 1-m_alpha_FAM))*cos((theta_hat_2-(api/2))*(1-
m_alpha_FAM))) - (B_2*(pow(radius_hat_2, 1-m_alpha_FAM))*sin((theta_hat_2-(api/2))*(1-
m_alpha_FAM)));
	lambda_sw = (A_3*(pow(radius_hat_3, 1-m_alpha_FAM))*cos((theta_hat_3+api)*(1-
m_alpha_FAM))) - (B_3*(pow(radius_hat_3, 1-m_alpha_FAM))*sin((theta_hat_3+api)*(1-
m_alpha_FAM)));
	lambda_s = (A_4*(pow(radius_hat_4, 1-m_alpha_FAM))*cos((theta_hat_4+(api/2))*(1-
m_alpha_FAM))) + (B_4*(pow(radius_hat_4, 1-m_alpha_FAM))*sin((theta_hat_4+(api/2))*(1-
m_alpha_FAM)));

			if (XI_x > XQ_x && XI_y > XQ_y)
			{

				if (XK_x > XQ_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
				{
					theta_xycoord = 0.0;
					delta_y = 0.0;
					x_hat_1 =
((delta_x*cos(m_theta_tilde_1))+(delta_y*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_1);
					y_hat_1 =
((delta_y*cos(m_theta_tilde_1))+(delta_x*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_2);
					radius_hat_1 = sqrt((x_hat_1*x_hat_1) +
(y_hat_1*y_hat_1));
					theta_hat_1 = theta_xycoord - m_theta_tilde_1;

					flux_part_1_1 = (A_1*(pow(radius_hat_1, -
m_alpha_FAM))*(x_hat_1/radius_hat_1)*cos(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, -
m_alpha_FAM))*(x_hat_1/radius_hat_1)*sin(theta_hat_1*(1-m_alpha_FAM)));
					flux_part_1_2 = (-A_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(-y_hat_1/(radius_hat_1*radius_hat_1))*sin(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, 1-m_alpha_FAM))*(-
y_hat_1/(radius_hat_1*radius_hat_1))*cos(theta_hat_1*(1-m_alpha_FAM)));
					flux_part_2_1 = (A_1*(pow(radius_hat_1, -
m_alpha_FAM))*(y_hat_1/radius_hat_1)*cos(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, -
m_alpha_FAM))*(y_hat_1/radius_hat_1)*sin(theta_hat_1*(1-m_alpha_FAM)));
					flux_part_2_2 = (-A_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(x_hat_1/(radius_hat_1*radius_hat_1))*sin(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(x_hat_1/(radius_hat_1*radius_hat_1))*cos(theta_hat_1*(1-m_alpha_FAM)));

					lambda_c_y =
delta_x*((m_k1_1_2*(cos(m_theta_tilde_1)/sqrt(m_k1_tilde_1))*(flux_part_1_1+flux_part_1_2
))+(m_k1_2_2*(cos(m_theta_tilde_1)/sqrt(m_k1_tilde_2))*(flux_part_2_1+flux_part_2_2)));
				}
```

```
                        else if (XK_y > XQ_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                        {
                                theta_xycoord = api/2;
                                delta_x = 0.0;
                                x_hat_1 =
((delta_x*cos(m_theta_tilde_1))+(delta_y*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_1);
                                y_hat_1 =
((delta_y*cos(m_theta_tilde_1))+(delta_x*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_2);
                                radius_hat_1 = sqrt((x_hat_1*x_hat_1) +
(y_hat_1*y_hat_1));
                                theta_hat_1 = theta_xycoord - m_theta_tilde_1;

                                flux_part_1_1 = (A_1*(pow(radius_hat_1, -
m_alpha_FAM))*(x_hat_1/radius_hat_1)*cos(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, -
m_alpha_FAM))*(x_hat_1/radius_hat_1)*sin(theta_hat_1*(1-m_alpha_FAM)));
                                flux_part_1_2 = (-A_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(-y_hat_1/(radius_hat_1*radius_hat_1))*sin(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, 1-m_alpha_FAM))*(-
y_hat_1/(radius_hat_1*radius_hat_1))*cos(theta_hat_1*(1-m_alpha_FAM)));
                                flux_part_2_1 = (A_1*(pow(radius_hat_1, -
m_alpha_FAM))*(y_hat_1/radius_hat_1)*cos(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, -
m_alpha_FAM))*(y_hat_1/radius_hat_1)*sin(theta_hat_1*(1-m_alpha_FAM)));
                                flux_part_2_2 = (-A_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(x_hat_1/(radius_hat_1*radius_hat_1))*sin(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(x_hat_1/(radius_hat_1*radius_hat_1))*cos(theta_hat_1*(1-m_alpha_FAM)));

                                lambda_c_x =
delta_y*((m_k1_1_1*(cos(m_theta_tilde_1)/sqrt(m_k1_tilde_1))*(flux_part_1_1+flux_part_1_2
))+(m_k1_1_2*(cos(m_theta_tilde_1)/sqrt(m_k1_tilde_2))*(flux_part_2_1+flux_part_2_2)));
                        }
                }

                else if (XQ_x > XI_x && XI_y > XQ_y)
                {

                        if (XQ_x > XK_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
                        {
                                theta_xycoord = -api;
                                delta_y = 0.0;
                                x_hat_3 =
((delta_x*cos(m_theta_tilde_3))+(delta_y*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_1);
                                y_hat_3 =
((delta_y*cos(m_theta_tilde_3))+(delta_x*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_2);
                                radius_hat_3 = sqrt((x_hat_3*x_hat_3) +
(y_hat_3*y_hat_3));
                                theta_hat_3 = theta_xycoord - m_theta_tilde_3;

                                flux_part_1_1 = (A_3*(pow(radius_hat_3, -
m_alpha_FAM))*(x_hat_3/radius_hat_3)*cos((theta_hat_3+api)*(1-m_alpha_FAM)))-
(B_3*(pow(radius_hat_3, -m_alpha_FAM))*(x_hat_3/radius_hat_3)*sin((theta_hat_3+api)*(1-
m_alpha_FAM)));
                                flux_part_1_2 = (-A_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(-y_hat_3/(radius_hat_3*radius_hat_3))*sin((theta_hat_3+api)*(1-
```

```
m_alpha_FAM)))-(B_3*(pow(radius_hat_3, 1-m_alpha_FAM))*(-
y_hat_3/(radius_hat_3*radius_hat_3))*cos((theta_hat_3+api)*(1-m_alpha_FAM)));
                                    flux_part_2_1 = (A_3*(pow(radius_hat_3, -
m_alpha_FAM))*(y_hat_3/radius_hat_3)*cos((theta_hat_3+api)*(1-m_alpha_FAM)))-
(B_3*(pow(radius_hat_3, -m_alpha_FAM))*(y_hat_3/radius_hat_3)*sin((theta_hat_3+api)*(1-
m_alpha_FAM)));
                                    flux_part_2_2 = (-A_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(x_hat_3/(radius_hat_3*radius_hat_3))*sin((theta_hat_3+api)*(1-
m_alpha_FAM)))-(B_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(x_hat_3/(radius_hat_3*radius_hat_3))*cos((theta_hat_3+api)*(1-
m_alpha_FAM)));

                                    lambda_sw_y =
delta_x*((m_k3_1_2*(cos(m_theta_tilde_3)/sqrt(m_k3_tilde_1))*(flux_part_1_1+flux_part_1_2
))+(m_k3_2_2*(cos(m_theta_tilde_3)/sqrt(m_k3_tilde_2))*(flux_part_2_1+flux_part_2_2)));
                                }
                                else if (XK_y > XQ_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                                {
                                    theta_xycoord = api/2;
                                    delta_x = 0.0;
                                    x_hat_1 =
((delta_x*cos(m_theta_tilde_1))+(delta_y*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_1);
                                    y_hat_1 =
((delta_y*cos(m_theta_tilde_1))+(delta_x*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_2);
                                    radius_hat_1 = sqrt((x_hat_1*x_hat_1) +
(y_hat_1*y_hat_1));
                                    theta_hat_1 = theta_xycoord - m_theta_tilde_1;

                                    flux_part_1_1 = (A_1*(pow(radius_hat_1, -
m_alpha_FAM))*(x_hat_1/radius_hat_1)*cos(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, -
m_alpha_FAM))*(x_hat_1/radius_hat_1)*sin(theta_hat_1*(1-m_alpha_FAM)));
                                    flux_part_1_2 = (-A_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(-y_hat_1/(radius_hat_1*radius_hat_1))*sin(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, 1-m_alpha_FAM))*(-
y_hat_1/(radius_hat_1*radius_hat_1))*cos(theta_hat_1*(1-m_alpha_FAM)));
                                    flux_part_2_1 = (A_1*(pow(radius_hat_1, -
m_alpha_FAM))*(y_hat_1/radius_hat_1)*cos(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, -
m_alpha_FAM))*(y_hat_1/radius_hat_1)*sin(theta_hat_1*(1-m_alpha_FAM)));
                                    flux_part_2_2 = (-A_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(x_hat_1/(radius_hat_1*radius_hat_1))*sin(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(x_hat_1/(radius_hat_1*radius_hat_1))*cos(theta_hat_1*(1-m_alpha_FAM)));

                                    lambda_c_x =
delta_y*((m_k1_1_1*(cos(m_theta_tilde_1)/sqrt(m_k1_tilde_1))*(flux_part_1_1+flux_part_1_2
))+(m_k1_1_2*(cos(m_theta_tilde_1)/sqrt(m_k1_tilde_2))*(flux_part_2_1+flux_part_2_2)));
                                }
                        }

                        else if (XQ_x > XI_x && XQ_y > XI_y)
                        {

                                if (XQ_x > XK_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
                                {
```

```
                                    theta_xycoord = -api;
                                    delta_y = 0.0;
                                    x_hat_3 =
((delta_x*cos(m_theta_tilde_3))+(delta_y*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_1);
                                    y_hat_3 =
((delta_y*cos(m_theta_tilde_3))+(delta_x*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_2);
                                    radius_hat_3 = sqrt((x_hat_3*x_hat_3) +
(y_hat_3*y_hat_3));
                                    theta_hat_3 = theta_xycoord - m_theta_tilde_3;

                                    flux_part_1_1 = (A_3*(pow(radius_hat_3, -
m_alpha_FAM))*(x_hat_3/radius_hat_3)*cos((theta_hat_3+api)*(1-m_alpha_FAM)))-
(B_3*(pow(radius_hat_3, -m_alpha_FAM))*(x_hat_3/radius_hat_3)*sin((theta_hat_3+api)*(1-
m_alpha_FAM)));
                                    flux_part_1_2 = (-A_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(-y_hat_3/(radius_hat_3*radius_hat_3))*sin((theta_hat_3+api)*(1-
m_alpha_FAM)))-(B_3*(pow(radius_hat_3, 1-m_alpha_FAM))*(-
y_hat_3/(radius_hat_3*radius_hat_3))*cos((theta_hat_3+api)*(1-m_alpha_FAM)));
                                    flux_part_2_1 = (A_3*(pow(radius_hat_3, -
m_alpha_FAM))*(y_hat_3/radius_hat_3)*cos((theta_hat_3+api)*(1-m_alpha_FAM)))-
(B_3*(pow(radius_hat_3, -m_alpha_FAM))*(y_hat_3/radius_hat_3)*sin((theta_hat_3+api)*(1-
m_alpha_FAM)));
                                    flux_part_2_2 = (-A_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(x_hat_3/(radius_hat_3*radius_hat_3))*sin((theta_hat_3+api)*(1-
m_alpha_FAM)))-(B_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(x_hat_3/(radius_hat_3*radius_hat_3))*cos((theta_hat_3+api)*(1-
m_alpha_FAM)));

                                    lambda_sw_y =
delta_x*((m_k3_1_2*(cos(m_theta_tilde_3)/sqrt(m_k3_tilde_1))*(flux_part_1_1+flux_part_1_2
))+(m_k3_2_2*(cos(m_theta_tilde_3)/sqrt(m_k3_tilde_2))*(flux_part_2_1+flux_part_2_2)));
                                }
                                else if (XQ_y > XK_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                                {
                                    theta_xycoord = (-api)/2;
                                    delta_x = 0.0;
                                    x_hat_3 =
((delta_x*cos(m_theta_tilde_3))+(delta_y*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_1);
                                    y_hat_3 =
((delta_y*cos(m_theta_tilde_3))+(delta_x*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_2);
                                    radius_hat_3 = sqrt((x_hat_3*x_hat_3) +
(y_hat_3*y_hat_3));
                                    theta_hat_3 = theta_xycoord - m_theta_tilde_3;

                                    flux_part_1_1 = (A_3*(pow(radius_hat_3, -
m_alpha_FAM))*(x_hat_3/radius_hat_3)*cos((theta_hat_3+api)*(1-m_alpha_FAM)))-
(B_3*(pow(radius_hat_3, -m_alpha_FAM))*(x_hat_3/radius_hat_3)*sin((theta_hat_3+api)*(1-
m_alpha_FAM)));
                                    flux_part_1_2 = (-A_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(-y_hat_3/(radius_hat_3*radius_hat_3))*sin((theta_hat_3+api)*(1-
m_alpha_FAM)))-(B_3*(pow(radius_hat_3, 1-m_alpha_FAM))*(-
y_hat_3/(radius_hat_3*radius_hat_3))*cos((theta_hat_3+api)*(1-m_alpha_FAM)));
                                    flux_part_2_1 = (A_3*(pow(radius_hat_3, -
m_alpha_FAM))*(y_hat_3/radius_hat_3)*cos((theta_hat_3+api)*(1-m_alpha_FAM)))-
(B_3*(pow(radius_hat_3, -m_alpha_FAM))*(y_hat_3/radius_hat_3)*sin((theta_hat_3+api)*(1-
m_alpha_FAM)));
```

```
                                flux_part_2_2 = (-A_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(x_hat_3/(radius_hat_3*radius_hat_3))*sin((theta_hat_3+api)*(1-
m_alpha_FAM)))-(B_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(x_hat_3/(radius_hat_3*radius_hat_3))*cos((theta_hat_3+api)*(1-
m_alpha_FAM)));

                                lambda_sw_x =
delta_y*((m_k3_1_1*(cos(m_theta_tilde_3)/sqrt(m_k3_tilde_1))*(flux_part_1_1+flux_part_1_2
))+(m_k3_1_2*(cos(m_theta_tilde_3)/sqrt(m_k3_tilde_2))*(flux_part_2_1+flux_part_2_2)));
                        }
                }

                else if (XI_x > XQ_x && XQ_y > XI_y)
                {

                        if (XK_x > XQ_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
                        {
                                theta_xycoord = 0.0;
                                delta_y = 0.0;
                                x_hat_1 =
((delta_x*cos(m_theta_tilde_1))+(delta_y*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_1);
                                y_hat_1 =
((delta_y*cos(m_theta_tilde_1))+(delta_x*sin(m_theta_tilde_1)))/sqrt(m_k1_tilde_2);
                                radius_hat_1 = sqrt((x_hat_1*x_hat_1) +
(y_hat_1*y_hat_1));
                                theta_hat_1 = theta_xycoord - m_theta_tilde_1;

                                flux_part_1_1 = (A_1*(pow(radius_hat_1, -
m_alpha_FAM))*(x_hat_1/radius_hat_1)*cos(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, -
m_alpha_FAM))*(x_hat_1/radius_hat_1)*sin(theta_hat_1*(1-m_alpha_FAM)));
                                flux_part_1_2 = (-A_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(-y_hat_1/(radius_hat_1*radius_hat_1))*sin(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, 1-m_alpha_FAM))*(-
y_hat_1/(radius_hat_1*radius_hat_1))*cos(theta_hat_1*(1-m_alpha_FAM)));
                                flux_part_2_1 = (A_1*(pow(radius_hat_1, -
m_alpha_FAM))*(y_hat_1/radius_hat_1)*cos(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, -
m_alpha_FAM))*(y_hat_1/radius_hat_1)*sin(theta_hat_1*(1-m_alpha_FAM)));
                                flux_part_2_2 = (-A_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(x_hat_1/(radius_hat_1*radius_hat_1))*sin(theta_hat_1*(1-
m_alpha_FAM)))+(B_1*(pow(radius_hat_1, 1-
m_alpha_FAM))*(x_hat_1/(radius_hat_1*radius_hat_1))*cos(theta_hat_1*(1-m_alpha_FAM)));

                                lambda_c_y =
delta_x*((m_k1_1_2*(cos(m_theta_tilde_1)/sqrt(m_k1_tilde_1))*(flux_part_1_1+flux_part_1_2
))+(m_k1_2_2*(cos(m_theta_tilde_1)/sqrt(m_k1_tilde_2))*(flux_part_2_1+flux_part_2_2)));
                        }
                        else if (XQ_y > XK_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                        {
                                theta_xycoord = (-api)/2;
                                delta_x = 0.0;
                                x_hat_3 =
((delta_x*cos(m_theta_tilde_3))+(delta_y*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_1);
                                y_hat_3 =
((delta_y*cos(m_theta_tilde_3))+(delta_x*sin(m_theta_tilde_3)))/sqrt(m_k3_tilde_2);
```

```
                                    radius_hat_3 = sqrt((x_hat_3*x_hat_3) +
(y_hat_3*y_hat_3));
                                    theta_hat_3 = theta_xycoord - m_theta_tilde_3;

                                    flux_part_1_1 = (A_3*(pow(radius_hat_3, -
m_alpha_FAM))*(x_hat_3/radius_hat_3)*cos((theta_hat_3+api)*(1-m_alpha_FAM)))-
(B_3*(pow(radius_hat_3, -m_alpha_FAM))*(x_hat_3/radius_hat_3)*sin((theta_hat_3+api)*(1-
m_alpha_FAM)));
                                    flux_part_1_2 = (-A_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(-y_hat_3/(radius_hat_3*radius_hat_3))*sin((theta_hat_3+api)*(1-
m_alpha_FAM)))-(B_3*(pow(radius_hat_3, 1-m_alpha_FAM))*(-
y_hat_3/(radius_hat_3*radius_hat_3))*cos((theta_hat_3+api)*(1-m_alpha_FAM)));
                                    flux_part_2_1 = (A_3*(pow(radius_hat_3, -
m_alpha_FAM))*(y_hat_3/radius_hat_3)*cos((theta_hat_3+api)*(1-m_alpha_FAM)))-
(B_3*(pow(radius_hat_3, -m_alpha_FAM))*(y_hat_3/radius_hat_3)*sin((theta_hat_3+api)*(1-
m_alpha_FAM)));
                                    flux_part_2_2 = (-A_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(x_hat_3/(radius_hat_3*radius_hat_3))*sin((theta_hat_3+api)*(1-
m_alpha_FAM)))-(B_3*(pow(radius_hat_3, 1-
m_alpha_FAM))*(x_hat_3/(radius_hat_3*radius_hat_3))*cos((theta_hat_3+api)*(1-
m_alpha_FAM)));

                                    lambda_sw_x =
delta_y*((m_k3_1_1*(cos(m_theta_tilde_3)/sqrt(m_k3_tilde_1))*(flux_part_1_1+flux_part_1_2
))+(m_k3_1_2*(cos(m_theta_tilde_3)/sqrt(m_k3_tilde_2))*(flux_part_2_1+flux_part_2_2)));
                                }
                        }
}
realS FAMANI2DSGDiscritization::computeMatrixContribution(const int ihat, const int
itilde, const int khat, const int qhat, const ArrayS<realS>& alpha1, const ArrayS<realS>&
alpha2, const bool atBoundary)
{
       realS ret=0.0;
       double api = atan(1.0)*4;
       double delta_x = 0.0;
       double delta_y = 0.0;
       double theta = 0.0;
       double radius = 0.0;
       double gamma_x = 0.0;
       double gamma_y = 0.0;

       Pair<int,int> PQ(0, qhat);
    Pair<int,int> PI(2, ihat);
       Pair<int,int> PK(1, khat);

       RnPoint XQ = m_C.u_getCoordinate_0u(PQ);
       double XQ_x = XQ.u_getElement_0u(0);
       double XQ_y = XQ.u_getElement_0u(1);
       RnPoint XI = m_C.u_getCoordinate_0u(PI);
       double XI_x = XI.u_getElement_0u(0);
       double XI_y = XI.u_getElement_0u(1);
       RnPoint XK = m_C.u_getCoordinate_0u(PK);
       double XK_x = XK.u_getElement_0u(0);
       double XK_y = XK.u_getElement_0u(1);

       RnPoint v1 = XK;
       v1 -= XQ;
       RnPoint v2 = XI;
```

```
	v2 -= XQ;

	theta = acos(v1.u_dot_0u(v2)/(v1.u_norm2_0u()*v2.u_norm2_0u()));

	delta_x = fabs(XI_x - XQ_x);// Half actual delta_x in the grid!!!!
	delta_y = fabs(XI_y - XQ_y);// Half actual delta_y in the grid!!!!
	radius = sqrt((delta_x*delta_x)+(delta_y*delta_y));
	bool Geometric = false; //This should be turned true and “m_notHarmonicScheme =
false” if the geometric mean calcualtions are desired

		if (m_useFAMScheme)
		{
			double lambda_c = 0.0;
			double lambda_w = 0.0;
			double lambda_s = 0.0;
			double lambda_sw = 0.0;
			double lambda_c_x = 0.0;
			double lambda_c_y = 0.0;
			double lambda_sw_x = 0.0;
			double lambda_sw_y = 0.0;

			computeLambdas(ihat, itilde, khat, qhat/*This might be sketchy, must
ask!!!*/, lambda_c, lambda_w, lambda_sw, lambda_s, lambda_c_x, lambda_c_y, lambda_sw_x,
lambda_sw_y);

			if (XI_x > XQ_x && XI_y > XQ_y)
			{

				if (XK_x > XQ_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
				{
					gamma_y = (-lambda_c_y)/(lambda_s - lambda_c);
				}
				else if (XK_y > XQ_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
				{
					gamma_x = (-lambda_c_x)/(lambda_w - lambda_c);
				}
			}

			else if (XQ_x > XI_x && XI_y > XQ_y)
			{

				if (XQ_x > XK_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
				{
					gamma_y = (lambda_sw_y)/(lambda_sw - lambda_w);
				}
				else if (XK_y > XQ_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
				{
					gamma_x = (-lambda_c_x)/(lambda_w - lambda_c);
				}
			}

			else if (XQ_x > XI_x && XQ_y > XI_y)
			{
```

```
                        if (XQ_x > XK_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
                        {
                                gamma_y = (lambda_sw_y)/(lambda_sw - lambda_w);
                        }
                        else if (XQ_y > XK_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                        {
                                gamma_x = (lambda_sw_x)/(lambda_sw - lambda_s);
                        }
                }

                else if (XI_x > XQ_x && XQ_y > XI_y)
                {

                        if (XK_x > XQ_x && fabs(XK_y - XQ_y) <= m_Ftol) // The "&&..."
addition is only present as an aditional check
                        {
                                //lambda_y = (-m_k1*m_C_FAM)*(pow(delta_x, 1-
m_alpha_FAM));

                                gamma_y = (-lambda_c_y)/(lambda_s - lambda_c);
                        }
                        else if (XQ_y > XK_y && fabs(XK_x - XQ_x) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                        {
                                gamma_x = (lambda_sw_x)/(lambda_sw - lambda_s);
                        }
                }

        }
        else
        {

                if (Geometric)
                {
                        gamma_y = fabs(delta_y/delta_x)*0.5*sqrt(m_k1*m_k2);
                        gamma_x = fabs(delta_x/delta_y)*0.5*sqrt(m_k1*m_k2);
                }
                else
                {
                        if (XI_x > XQ_x && XI_y > XQ_y)
                        {
                                if (XK_x > XQ_x && fabs(XK_y - XQ_y) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                                {

                                        gamma_y =
fabs(delta_y/delta_x)*((m_k1_tilde_1*m_k4_tilde_1)/(m_k1_tilde_1+m_k4_tilde_1));
                                }
                                else if (XK_y > XQ_y && fabs(XK_x - XQ_x) <= m_Ftol) //
The "&&..." addition is only present as an aditional check
                                {

                                        gamma_x =
fabs(delta_x/delta_y)*((m_k1_tilde_1*m_k2_tilde_1)/(m_k1_tilde_1+m_k2_tilde_1));
                                }
                        }
```

```
                        else if (XQ_x > XI_x && XI_y > XQ_y)
                        {
                                if (XQ_x > XK_x && fabs(XK_y - XQ_y) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                                {

                                        gamma_y =
fabs(delta_y/delta_x)*((m_k2_tilde_1*m_k3_tilde_1)/(m_k2_tilde_1+m_k3_tilde_1));
                                }
                                else if (XK_y > XQ_y && fabs(XK_x - XQ_x) <= m_Ftol) //
The "&&..." addition is only present as an aditional check
                                {

                                        gamma_x =
fabs(delta_x/delta_y)*((m_k1_tilde_1*m_k2_tilde_1)/(m_k1_tilde_1+m_k2_tilde_1));
                                }
                        }

                        else if (XQ_x > XI_x && XQ_y > XI_y)
                        {
                                if (XQ_x > XK_x && fabs(XK_y - XQ_y) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                                {

                                        gamma_y =
fabs(delta_y/delta_x)*((m_k2_tilde_1*m_k3_tilde_1)/(m_k2_tilde_1+m_k3_tilde_1));
                                }
                                else if (XQ_y > XK_y && fabs(XK_x - XQ_x) <= m_Ftol) //
The "&&..." addition is only present as an aditional check
                                {

                                        gamma_x =
fabs(delta_x/delta_y)*((m_k3_tilde_1*m_k4_tilde_1)/(m_k3_tilde_1+m_k4_tilde_1));
                                }
                        }

                        else if (XI_x > XQ_x && XQ_y > XI_y)
                        {
                                if (XK_x > XQ_x && fabs(XK_y - XQ_y) <= m_Ftol) // The
"&&..." addition is only present as an aditional check
                                {

                                        gamma_y =
fabs(delta_y/delta_x)*((m_k1_tilde_1*m_k4_tilde_1)/(m_k1_tilde_1+m_k4_tilde_1));
                                }
                                else if (XQ_y > XK_y && fabs(XK_x - XQ_x) <= m_Ftol) //
The "&&..." addition is only present as an aditional check
                                {

                                        gamma_x =
fabs(delta_x/delta_y)*((m_k3_tilde_1*m_k4_tilde_1)/(m_k3_tilde_1+m_k4_tilde_1));
                                }
                        }
                }
                //Fungerer kun ved isotropi, må bruke en annen skjema (MPFA)

        }
```

```
	RnPoint normalVec = m_C.u_getNormalVector_0u(PI, PK);

	if ((fabs(normalVec.u_getElement_0u(1)) <= m_Ftol) ? (ret = gamma_x) : (ret =
gamma_y));

	return ret;
}
realS FAMANI2DSGDiscritization::computeMatrixContributionAtBoundary(const int ihat, const
int itilde, const int khat, const int qhat, const ArrayS<realS>& alpha1, const
ArrayS<realS>& alpha2, const bool atBoundary, const bool edgeAtBoundary)
{
	realS ret=0.0;
	double delta_x = 0.0;
	double delta_y = 0.0;
	double gamma_x = 0.0;
	double gamma_y = 0.0;

	Pair<int,int> PQ(0, qhat);
    Pair<int,int> PI(2, ihat);
	Pair<int,int> PK(1, khat);

	RnPoint XQ = m_C.u_getCoordinate_0u(PQ);
	double XQ_x = XQ.u_getElement_0u(0);
	double XQ_y = XQ.u_getElement_0u(1);
	RnPoint XI = m_C.u_getCoordinate_0u(PI);
	double XI_x = XI.u_getElement_0u(0);
	double XI_y = XI.u_getElement_0u(1);
	RnPoint XK = m_C.u_getCoordinate_0u(PK);
	double XK_x = XK.u_getElement_0u(0);
	double XK_y = XK.u_getElement_0u(1);

	RnPoint v1 = XK;
	v1 -= XQ;
	RnPoint v2 = XI;
	v2 -= XQ;

	delta_x = XI_x - XQ_x;// Half actual delta_x in the grid!!!!
	delta_y = XI_y - XQ_y;// Half actual delta_y in the grid!!!!


	if (edgeAtBoundary)
	{
		if (fabs(alpha1[m_C.u_findBoundaryPosition_0u(PK)]) <= m_Ftol)
		{
			//We have a Neumann (i.e. flux) B.C.
			gamma_x = 0.0;
			gamma_y = 0.0;
		}
		else
		{
			//We have a Dirichlet (i.e. imposed pressure) B.C.
			gamma_x = fabs(delta_y/delta_x)*m_permxx[ihat];
			gamma_y = fabs(delta_x/delta_y)*m_permxx[ihat];

		}
	}
	else
```

```
	{
		ArrayS<int> neigh;
		m_C.u_getNeighbours_RF(PK, 2, neigh);// the '2' is hardcoded due to 2D
space, for different problem dimensions, use 'problemDimension'....?
		int ihatN = (neigh[0] == ihat) ? neigh[1] : neigh[0];

		gamma_x =
fabs(delta_y/delta_x)*((m_permxx[ihat]*m_permxx[ihatN])/(m_permxx[ihat]+m_permxx[ihatN]))
;		gamma_y =
fabs(delta_x/delta_y)*((m_permxx[ihat]*m_permxx[ihatN])/(m_permxx[ihat]+m_permxx[ihatN]))
;


	}

	RnPoint normalVec = m_C.u_getNormalVector_0u(PI, PK);

	if ((fabs(normalVec.u_getElement_0u(1)) <= m_Ftol) ? (ret = gamma_x) : (ret =
gamma_y));

	return ret;
}
```

Appendix D: Setup Routines for the Finite Analytic Method

Setup Routine for Isotropic Finite Analytic Method:

```
void TempmodEI::u_setupFAM2DSG_1u(const int &PeriodicBC, const CSnShape &C, const ArrayS<
realS > &gExternalSurfaces,
       const ArrayS< realS > &alpha1ExternalSurfaces, const ArrayS< realS >
&alpha2ExternalSurfaces, const ArrayS< realS > &gWells,
       const ArrayS< realS > &alpha1Wells, const ArrayS< realS > &alpha2Wells, const
ArrayS<realS>& perm)
{

  m_C = C;

  //New treatment of BC's such that wells and holes can also be treated
!!!**************'
  ArrayS< realS > m_g;
  ArrayS< realS > m_alpha1;
  ArrayS< realS > m_alpha2;
  m_C.u_createConstBoundaryConditionData_0g(2,gExternalSurfaces,gWells,m_g);

m_C.u_createConstBoundaryConditionData_0g(0,alpha1ExternalSurfaces,alpha1Wells,m_alpha1);

m_C.u_createConstBoundaryConditionData_0g(1,alpha2ExternalSurfaces,alpha2Wells,m_alpha2);

  MeshShape SV;
  SV . u_setShape_1u(1);
  int mLength = m_C . u_cellCount_0u(m_C . u_cellDimension_0u());
  SV . u_changeElement_1u(0, mLength);
  MeshShape Smat;
  Smat . u_setShape_1u(2);
  Smat . u_changeElement_1u(0, mLength);
  Smat . u_changeElement_1u(1, mLength);

  int qLength = m_C . u_cellCount_0u(0);
  ArrayS< realS > innerHelp;
  ArrayS< ArrayS< MeshPoint > > AMP(mLength);
  ArrayS< MeshPoint > innerAMP;
  ArrayS< int > innerLengthAMP(mLength);
  ArrayS< ArrayS< realS > > Out(mLength);
  ArrayS< realS > OutBC(mLength, 0.0);
  for (int ihatloc = 0;(ihatloc < mLength); (ihatloc ++))
    {
      (innerAMP = m_C . u_getMatrixStencil_CDSmet_0u(Smat, ihatloc));
      innerLengthAMP . u_changeElement_1u(ihatloc, innerAMP . u_getSize_0u());
      AMP.u_changeElement_1u(ihatloc, innerAMP);
    }

  int qhat;
  int z;
  int k;
  int kMax;
```

```
int itildeMax;
int ihatn;
int nuFIndex;
int khat;
int cellDim = m_C . u_cellDimension_0u();
Pair< int, int > PQ(0, 0);
Pair< int, int > PGn(cellDim, 0);
Pair< int, int > PGnm1((cellDim - 1), 0);
ArrayS< int > ar1(2);
ArrayS< int > ar2(2);
ArrayS< int > ar3(2);
ArrayS< int > ar5(3);
ar1 . u_changeElement_1u(0, 0);
ar1 . u_changeElement_1u(1, cellDim);
ar2 . u_changeElement_1u(0, 0);
ar2 . u_changeElement_1u(1, cellDim);
ar3 . u_changeElement_1u(0, 0);
ar3 . u_changeElement_1u(1, (cellDim - 1));
ar5 . u_changeElement_1u(0, 0);
ar5 . u_changeElement_1u(1, (cellDim - 1));
ar5 . u_changeElement_1u(2, cellDim);
int v_ar1[MAXLOCARRAYSIZE];
int v_ar3[MAXLOCARRAYSIZE];
int v_ar5[MAXLOCARRAYSIZE];
MeshShape sh1;
MeshShape sh2;
MeshShape sh3;
MeshShape sh5;
MeshPoint mp1;
MeshPoint mp2;
MeshPoint mp3;
MeshPoint mp5;
(v_ar1[0] = qLength);
(v_ar3[0] = qLength);
(v_ar5[0] = qLength);
realS integrationValue;
ArrayS< ArrayS< int > > kzTab;
ArrayS< int > kTab;
ArrayS< int > zTab;
int bkTab[ARRAYMAXSIZE];
int bzTab[ARRAYMAXSIZE];
int LT;
int LGIE;
int Lq;
ArrayS< int > ari(2);
ari . u_changeElement_1u(0, cellDim);
ari . u_changeElement_1u(1, 0);
int v_ari[MAXLOCARRAYSIZE];
(v_ari[0] = mLength);
MeshShape shi;
MeshPoint mpi;

FAM2DSGDiscritization FAMNum;
FAMNum.setup(m_C, perm);

for (int ihat = 0;(ihat < mLength); (ihat ++))
  {
    PGn . u_updateSecond_1u(ihat);
```

```
(LT = innerLengthAMP[ihat]);
(Lq = m_C . u_getStencil_Omet_0u(PGn, 0));
(v_ari[1] = Lq);
shi . u_setShape_1u(2, v_ari);
mpi . u_setPoint_1u(shi, 0);
mpi . u_changeElement_1u(0, ihat);
innerHelp . u_copy_1u(LT, 0.0);
for (int q = 0;(q < Lq); (q ++))
  {
    mpi . u_changeElement_1u(1, q);
    (qhat = m_C . u_gammaF_Omet_0u(ari, mpi));
    PQ . u_updateSecond_1u(qhat);
          bool atBoundary = m_C.u_boundaryIndex_0u(PQ);
          FAMNum.computeAlphaAndC(PQ, ihat, atBoundary);
          (kMax = m_C . u_getStencil_Omet_0u(PQ, (cellDim - 1)));
    (itildeMax = m_C . u_getStencil_Omet_0u(PQ, cellDim));
    (v_ar1[1] = itildeMax);
    (v_ar3[1] = kMax);
    (v_ar5[1] = kMax);
    (v_ar5[2] = 2);
    sh1 . u_setShape_1u(2, v_ar1);
    mp1 . u_setPoint_1u(sh1, 0);
    sh2 . u_setShape_1u(2, v_ar1);
    mp2 . u_setPoint_1u(sh2, 0);
    sh3 . u_setShape_1u(2, v_ar3);
    mp3 . u_setPoint_1u(sh3, 0);
    sh5 . u_setShape_1u(3, v_ar5);
    mp5 . u_setPoint_1u(sh5, 0);
    mp1 . u_changeElement_1u(0, qhat);
    mp2 . u_changeElement_1u(0, qhat);
    mp3 . u_changeElement_1u(0, qhat);
    mp5 . u_changeElement_1u(0, qhat);

   //Now computes the "special" kz-table
   m_C.u_findTopPathIndices_0g(ihat, q, kzTab);
   kTab = kzTab[0];
   zTab = kzTab[1];
   //Note that each of the two rows in kzTab are arrays of equal length
   LGIE = kTab.u_getSize_0u();

   for (int iv=0; iv<LGIE; iv++)
   {
     k = kTab[iv];
     z = zTab[iv];

     mp3 . u_changeElement_1u(1, k);
     mp5 . u_changeElement_1u(1, k);
     mp5 . u_changeElement_1u(2, z);
     khat = m_C.u_gammaF_Omet_0u(ar3, mp3);
     PGnm1.u_updateSecond_1u(khat);

     PGn . u_updateSecond_1u(ihat);
     nuFIndex = m_C.u_nuF_Omet_0u(ihat, ar5, mp5);
     mp1 . u_changeElement_1u(1, nuFIndex);

     if (atBoundary)
     {
       mp2.u_changeElement_1u(1,0);//important when at boundary !
```

```
                integrationValue = FAMNum.computeRHSContribution(ihat, khat, qhat, m_g);
              OutBC.u_changeElement_1u(ihat,OutBC[ihat] + integrationValue);
            }

            for (int t=0; t<LT; t++)
            {
                integrationValue = 0.0;
              MeshPoint t_18;
              AMP[ihat].u_getElement_0g(t, t_18);
              int itilde = m_C.u_itildeFromSparsityPattern_0u(qhat,t_18);

              if (itilde >= 0)
              {
                      mp2.u_changeElement_1u(1,itilde);
                      int ihatm;
                      t_18.u_getElement_0g(1, ihatm);

                    Pair<int,int> Pkhatt(1,khat);
                       Pair<int,int> Pihatm(2,ihatm);
                       Pair<int,int> Pihatt(2,ihat);
                       bool edgeAtBoundary = m_C.u_boundaryIndex_0u(Pkhatt);
                       int Idiag = (ihat == ihatm) ? 2 : 1;
//*****************************************************************************
*******
                       if (!atBoundary)
                             {
                                    if (m_C.u_isNeighbour_0u(Pkhatt, Pihatm))
                                    {
                                     integrationValue =
FAMNum.computeMatrixContribution(ihat, itilde, khat, qhat, m_alpha1, m_alpha2,
atBoundary);
                                    }
                                    else
                                    {
                                   integrationValue = 0.0;
                                    }
                             }
                      else
                             {
                                    if (!edgeAtBoundary)
                                    {
                                           if (m_C.u_isNeighbour_0u(Pkhatt, Pihatm))
                                           {
                                                  integrationValue =
FAMNum.computeMatrixContributionAtBoundary(ihat, itilde, khat, qhat, m_alpha1, m_alpha2,
atBoundary, edgeAtBoundary);
                                           }
                                           else
                                           {
                                                  integrationValue = 0.0;
                                           }
                                    }
                                    else
                                    {
                                           if (ihat == ihatm)
                                           { //NOTE: Here the following SHOULD be true:
vertex IS at boundary and edge IS at boundary!!!
```

```
                                                integrationValue =
FAMNum.computeMatrixContributionAtBoundary(ihat, itilde, khat, qhat, m_alpha1, m_alpha2,
atBoundary, edgeAtBoundary);
                                        }
                                        else
                                        {
                                                integrationValue = 0.0;
                                        }
                                }
                        }

//*************************************************************************************
*******
                        innerHelp.u_changeElement_1u(t,innerHelp[t] + (pow(-
1.0,Idiag)*integrationValue));

                    }
                  }//end of for loop for 't' and 'LT'
                }
              }
             Out.u_changeElement_1u(ihat,innerHelp);
           }


  //must now fill the m_Spart and the m_RHSpart variables:
  //m_Spart . u_copy_1u(Smat, AMP, Out);
  //m_RHSpart . u_copy_1u(SV, OutBC);
   if (m_deltat <= 0.0)
   {
     m_Spart.u_copy_1u(Smat,AMP,Out);
     m_RHSpart.u_copy_1u(SV,OutBC);
   }
   //If we are to solve a parabolic problem, we do the additional setup here:
   else if (m_deltat > 0.0)
   {
     //We first create the necessary vector consisting of one's.
     ArrayS<realS> help(mLength, 1.0);
     m_Vpart.u_copy_1u(SV,help);

     for (int ihat = 0; ihat < mLength; ihat++)
     {
       CSnShapeGIndex GI(cellDim,ihat);
       realS vol = m_C.u_getVolume_0u(GI);
       LT = innerLengthAMP[ihat];
       innerHelp = Out[ihat];
       //NBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNB
       //We divide by the negative volume
       //in order to fix the
       //correct sign for the parabolic problem:
       //IS THIS CORRECT???
       //NBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNB
       for (int t=0; t<LT; t++)
       {
        innerHelp.u_changeElement_1u(t,innerHelp[t]/(vol));
       }
       Out.u_changeElement_1u(ihat,innerHelp);
       OutBC.u_changeElement_1u(ihat,OutBC[ihat]/(-vol));
     }
```

```
      m_Spart.u_copy_1u(Smat,AMP,Out);
      m_RHSpart.u_copy_1u(SV,OutBC);
    }

  if ((PeriodicBC == 1))
    {

      //-------------------------------------------------------------
      //In order to obtain a unique solution, pressure must be
      //prescribed at a single point in the domain of the PDE.
      //We prescribe the pressure (i.e. the solution) to be
      //zero (0.0) at the middle of the domain.
      //-------------------------------------------------------------

       ArrayS< MeshPoint > matrowMcentral;
       (m_Spart . u_getSparsityPoints_0u() . u_getElement_0g)((mLength / 2),
matrowMcentral);
       MeshPoint mpr;
       realS perVal;
       int mL = matrowMcentral . u_getSize_0u();
       for (int m = 0;(m < mL); (m ++))
         {
           matrowMcentral . u_getElement_0g(m, mpr);
           {
             int c_19;
             mpr . u_getElement_0g(0, c_19);
             {
               int l_19;
               mpr . u_getElement_0g(1, l_19);
               (perVal = ((c_19 == l_19) ? 1.0 : 0.0));
             }
             m_Spart . u_setData_1u(mpr, perVal);
           }
         }

      //must also make sure that the right hand side is zero for "specified row".
       ArrayS< MeshPoint > matrowH;
       (m_RHSpart . u_getSparsityPoints_0u() . u_getElement_0g)((mLength / 2), matrowH);
       matrowH . u_getElement_0g(0, mpr);
       m_RHSpart . u_setData_1u(mpr, 0.0);
    }

}
```

Setup Routine for Anisotropic Finite Analytic Method:

```
void TempmodEI::u_setupFAMANI2DSG_1u(const int &PeriodicBC, const CSnShape &C,     const
ArrayS< realS > &gExternalSurfaces,
       const ArrayS< realS > &alpha1ExternalSurfaces, const ArrayS< realS >
&alpha2ExternalSurfaces, const ArrayS< realS > &gWells,
       const ArrayS< realS > &alpha1Wells, const ArrayS< realS > &alpha2Wells, const
ArrayS<realS>& perm_xx, const ArrayS<realS>& perm_xy, const ArrayS<realS>& perm_yy)
{

  m_C = C;

  ArrayS< realS > m_g;
  ArrayS< realS > m_alpha1;
  ArrayS< realS > m_alpha2;
  m_C.u_createConstBoundaryConditionData_0g(2,gExternalSurfaces,gWells,m_g);

m_C.u_createConstBoundaryConditionData_0g(0,alpha1ExternalSurfaces,alpha1Wells,m_alpha1);

m_C.u_createConstBoundaryConditionData_0g(1,alpha2ExternalSurfaces,alpha2Wells,m_alpha2);


  MeshShape SV;
  SV . u_setShape_1u(1);
  int mLength = m_C . u_cellCount_0u(m_C . u_cellDimension_0u());
  SV . u_changeElement_1u(0, mLength);
  MeshShape Smat;
  Smat . u_setShape_1u(2);
  Smat . u_changeElement_1u(0, mLength);
  Smat . u_changeElement_1u(1, mLength);

  int qLength = m_C . u_cellCount_0u(0);
  ArrayS< realS > innerHelp;
  ArrayS< ArrayS< MeshPoint > > AMP(mLength);
  ArrayS< MeshPoint > innerAMP;
  ArrayS< int > innerLengthAMP(mLength);
  ArrayS< ArrayS< realS > > Out(mLength);
  ArrayS< realS > OutBC(mLength, 0.0);
  for (int ihatloc = 0;(ihatloc < mLength); (ihatloc ++))
    {
      (innerAMP = m_C . u_getMatrixStencil_CDSmet_0u(Smat, ihatloc));
      innerLengthAMP . u_changeElement_1u(ihatloc, innerAMP . u_getSize_0u());
      AMP.u_changeElement_1u(ihatloc, innerAMP);
    }

  int qhat;
  int z;
  int k;
  int kMax;
  int itildeMax;
  int ihatn;
  int nuFIndex;
  int khat;
  int cellDim = m_C . u_cellDimension_0u();
  Pair< int, int > PQ(0, 0);
  Pair< int, int > PGn(cellDim, 0);
  Pair< int, int > PGnm1((cellDim - 1), 0);
```

```
ArrayS< int > ar1(2);
ArrayS< int > ar2(2);
ArrayS< int > ar3(2);
ArrayS< int > ar5(3);
ar1 . u_changeElement_1u(0, 0);
ar1 . u_changeElement_1u(1, cellDim);
ar2 . u_changeElement_1u(0, 0);
ar2 . u_changeElement_1u(1, cellDim);
ar3 . u_changeElement_1u(0, 0);
ar3 . u_changeElement_1u(1, (cellDim - 1));
ar5 . u_changeElement_1u(0, 0);
ar5 . u_changeElement_1u(1, (cellDim - 1));
ar5 . u_changeElement_1u(2, cellDim);
int v_ar1[MAXLOCARRAYSIZE];
int v_ar3[MAXLOCARRAYSIZE];
int v_ar5[MAXLOCARRAYSIZE];
MeshShape sh1;
MeshShape sh2;
MeshShape sh3;
MeshShape sh5;
MeshPoint mp1;
MeshPoint mp2;
MeshPoint mp3;
MeshPoint mp5;
(v_ar1[0] = qLength);
(v_ar3[0] = qLength);
(v_ar5[0] = qLength);

realS integrationValue;
ArrayS< ArrayS< int > > kzTab;
ArrayS< int > kTab;
ArrayS< int > zTab;
int bkTab[ARRAYMAXSIZE];
int bzTab[ARRAYMAXSIZE];
int LT;
int LGIE;
int Lq;
ArrayS< int > ari(2);
ari . u_changeElement_1u(0, cellDim);
ari . u_changeElement_1u(1, 0);
int v_ari[MAXLOCARRAYSIZE];
(v_ari[0] = mLength);
MeshShape shi;
MeshPoint mpi;

FAMANI2DSGDiscritization FAMNum;
FAMNum.setup(m_C, perm_xx, perm_xy, perm_yy);

for (int ihat = 0;(ihat < mLength); (ihat ++))
  {
    PGn . u_updateSecond_1u(ihat);
    (LT = innerLengthAMP[ihat]);
    (Lq = m_C . u_getStencil_Omet_0u(PGn, 0));
    (v_ari[1] = Lq);
    shi . u_setShape_1u(2, v_ari);
    mpi . u_setPoint_1u(shi, 0);
    mpi . u_changeElement_1u(0, ihat);
    innerHelp . u_copy_1u(LT, 0.0);
```

```
for (int q = 0;(q < Lq); (q ++))
  {
    mpi . u_changeElement_1u(1, q);
    (qhat = m_C . u_gammaF_Omet_0u(ari, mpi));
    PQ . u_updateSecond_1u(qhat);
          bool atBoundary = m_C.u_boundaryIndex_0u(PQ);
          FAMNum.computeEigenValuesAndRotation(PQ, ihat, atBoundary);
          FAMNum.computeAlphaAndC(PQ, ihat, atBoundary);
          (kMax = m_C . u_getStencil_Omet_0u(PQ, (cellDim - 1)));
    (itildeMax = m_C . u_getStencil_Omet_0u(PQ, cellDim));
    (v_ar1[1] = itildeMax);
    (v_ar3[1] = kMax);
    (v_ar5[1] = kMax);
    (v_ar5[2] = 2);
    sh1 . u_setShape_1u(2, v_ar1);
    mp1 . u_setPoint_1u(sh1, 0);
    sh2 . u_setShape_1u(2, v_ar1);
    mp2 . u_setPoint_1u(sh2, 0);
    sh3 . u_setShape_1u(2, v_ar3);
    mp3 . u_setPoint_1u(sh3, 0);
    sh5 . u_setShape_1u(3, v_ar5);
    mp5 . u_setPoint_1u(sh5, 0);
    mp1 . u_changeElement_1u(0, qhat);
    mp2 . u_changeElement_1u(0, qhat);
    mp3 . u_changeElement_1u(0, qhat);
    mp5 . u_changeElement_1u(0, qhat);

   //Now computes the "special" kz-table
   m_C.u_findTopPathIndices_0g(ihat, q, kzTab
   kTab = kzTab[0];
   zTab = kzTab[1];
   //Note that each of the two rows in kzTab are arrays of equal length
   LGIE = kTab.u_getSize_0u();

   for (int iv=0; iv<LGIE; iv++)
   {
     k = kTab[iv];
     z = zTab[iv];

     mp3 . u_changeElement_1u(1, k);
     mp5 . u_changeElement_1u(1, k);
     mp5 . u_changeElement_1u(2, z);
     khat = m_C.u_gammaF_Omet_0u(ar3, mp3);
     PGnm1.u_updateSecond_1u(khat);

     PGn . u_updateSecond_1u(ihat);
     nuFIndex = m_C.u_nuF_Omet_0u(ihat, ar5, mp5);
     mp1 . u_changeElement_1u(1, nuFIndex);


     if (atBoundary)
     {
       mp2.u_changeElement_1u(1,0);//important when at boundary !

          integrationValue = FAMNum.computeRHSContribution(ihat, khat, qhat, m_g);

       OutBC.u_changeElement_1u(ihat,OutBC[ihat] + integrationValue);
     }
```

```
            for (int t=0; t<LT; t++)
            {
                 integrationValue = 0.0;
              MeshPoint t_18;
              AMP[ihat].u_getElement_0g(t, t_18);
              int itilde = m_C.u_itildeFromSparsityPattern_0u(qhat,t_18);//Should probably
check this later!!??
              if (itilde >= 0)
              {
                      mp2.u_changeElement_1u(1,itilde);
                      int ihatm;
                      t_18.u_getElement_0g(1, ihatm);

                    Pair<int,int> Pkhatt(1,khat);
                       Pair<int,int> Pihatm(2,ihatm);
                       Pair<int,int> Pihatt(2,ihat);
                       bool edgeAtBoundary = m_C.u_boundaryIndex_0u(Pkhatt);
                       int Idiag = (ihat == ihatm) ? 2 : 1;
//*********************************************************************************
*******
                       if (!atBoundary)
                             {
                                    if (m_C.u_isNeighbour_0u(Pkhatt, Pihatm))
                                    {
                                     integrationValue =
FAMNum.computeMatrixContribution(ihat, itilde, khat, qhat, m_alpha1, m_alpha2,
atBoundary);
                                    }
                                    else
                                    {
                                   integrationValue = 0.0;
                                    }
                             }
                         else
                             {
                                    if (!edgeAtBoundary)
                                    {
                                           if (m_C.u_isNeighbour_0u(Pkhatt, Pihatm)
                                           {
                                                  integrationValue =
FAMNum.computeMatrixContributionAtBoundary(ihat, itilde, khat, qhat, m_alpha1, m_alpha2,
atBoundary, edgeAtBoundary);
                                                  //integrationValue = 0.5;
                                           }
                                           else
                                           {
                                                  integrationValue = 0.0;
                                           }
                                    }
                                    else
                                    {
                                           if (ihat == ihatm)
                                           { //NOTE: Here the following SHOULD be true:
vertex IS at boundary and edge IS at boundary!!!
                                                  integrationValue =
FAMNum.computeMatrixContributionAtBoundary(ihat, itilde, khat, qhat, m_alpha1, m_alpha2,
atBoundary, edgeAtBoundary);
```

```
                                        }
                                        else
                                        {
                                                integrationValue = 0.0;
                                        }
                                }
                        }

//*************************************************************************************
*******
                        innerHelp.u_changeElement_1u(t,innerHelp[t] + (pow(-
1.0,Idiag)*integrationValue));

                }
              }//end of for loop for 't' and 'LT'
            }
          }
         Out.u_changeElement_1u(ihat,innerHelp);
       }


  //must now fill the m_Spart and the m_RHSpart variables:
   if (m_deltat <= 0.0)
   {
     //WE SOLVE THE ELLIPTIC PROBLEM
     //must now fill the m_Spart and the m_RHSpart variables:
     m_Spart.u_copy_1u(Smat,AMP,Out);
     m_RHSpart.u_copy_1u(SV,OutBC);
   }
   //If we are to solve a parabolic problem, we do the additional setup here:
   else if (m_deltat > 0.0)
   {
     //We first create the necessary vector consisting of one's.
     ArrayS<realS> help(mLength, 1.0);
     m_Vpart.u_copy_1u(SV,help);

     //We then divide the non-zero elements of m_RHSpart and m_Spart with the actual
element volumes:
     for (int ihat = 0; ihat < mLength; ihat++)
     {
       CSnShapeGIndex GI(cellDim,ihat);
       realS vol = m_C.u_getVolume_0u(GI);
       LT = innerLengthAMP[ihat];
       innerHelp = Out[ihat];
       //NBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNB
       //We divide by the negative volume
       //in order to fix the
       //correct sign for the parabolic problem:
       //IS THIS CORRECT???
       //NBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNBNB
       for (int t=0; t<LT; t++)
       {
        innerHelp.u_changeElement_1u(t,innerHelp[t]/(vol));
       }
       Out.u_changeElement_1u(ihat,innerHelp);
       OutBC.u_changeElement_1u(ihat,OutBC[ihat]/(-vol));
     }
     m_Spart.u_copy_1u(Smat,AMP,Out);
```

```
      m_RHSpart.u_copy_1u(SV,OutBC);
    }

  if ((PeriodicBC == 1))
    {

     //-------------------------------------------------------------
     //In order to obtain a unique solution, pressure must be
     //prescribed at a single point in the domain of the PDE.
     //We prescribe the pressure (i.e. the solution) to be
     //zero (0.0) at the middle of the domain.
     //-------------------------------------------------------------

      ArrayS< MeshPoint > matrowMcentral;
      (m_Spart . u_getSparsityPoints_0u() . u_getElement_0g)((mLength / 2),
matrowMcentral);
      MeshPoint mpr;
      realS perVal;
      int mL = matrowMcentral . u_getSize_0u();
      for (int m = 0;(m < mL); (m ++))
        {
          matrowMcentral . u_getElement_0g(m, mpr);
          {
            int c_19;
            mpr . u_getElement_0g(0, c_19);
            {
              int l_19;
              mpr . u_getElement_0g(1, l_19);
              (perVal = ((c_19 == l_19) ? 1.0 : 0.0));
            }
            m_Spart . u_setData_1u(mpr, perVal);
          }
        }

     //must also make sure that the right hand side is zero for "specified row".
      ArrayS< MeshPoint > matrowH;
      (m_RHSpart . u_getSparsityPoints_0u() . u_getElement_0g)((mLength / 2), matrowH);
      matrowH . u_getElement_0g(0, mpr);
      m_RHSpart . u_setData_1u(mpr, 0.0);
    }

}
```