UNIVERSITY OF CENTRAL OKLAHOMA
Edmond, Oklahoma
Dr. Joe C. Jackson College of Graduate Studies

A THREE-DIMENSIONAL COMPUTATIONAL FLUID DYNAMICS MODEL FOR FLOW THROUGH POROUS MEDIA

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for

the degree of

MASTER OF SCIENCE IN ENGINEERING PHYSICS

By

CHRIS KISER

Oklahoma City, Oklahoma

2011

A THREE-DIMENSIONAL COMPUTATIONAL FLUID DYNAMICS MODEL FOR FLOW THROUGH POROUS MEDIA

A THESIS
APPROVED FOR THE DEPARTMENT OF ENGINEERING PHYSICS

June 8, 2011

By [illegible]
Dr. Evan Lemley, Chairman

David R. Martin
Dr. David Martin

[illegible]
Dr. Jaehoon Seong

ACKNOWLEDGEMENTS

I would first like to thank Dr. Evan Lemley for agreeing to be my thesis advisor and for allowing me to be a part of his research group. My experience with the group helped mold my research in loss coefficients and flow in porous media. I am grateful for Dr. Lemley’s insight and support throughout the lengthy programming and simulation process involved in my thesis.

I would also like to thank the Lord, my family, and my friends for the love and support shown to me all along the way.

TABLE OF CONTENTS

ABSTRACT OF THESIS

University of Central Oklahoma

Edmond, Oklahoma

NAME: Chris Kiser

TITLE OF THESIS: A Three-Dimensional Computational Fluid Dynamics Model for Flow Through Porous Media

DIRECTOR OF THESIS: Evan Lemley

PAGES: 53

ABSTRACT: This thesis covers the development of a model for fluid flow which incorporates computational fluid dynamics simulations using three-dimensional planar porous media networks. Porous media are introduced along with applications and the need for computational models is discussed. Previous experiments and models are presented as well as features of the current model. This model constructs three-dimensional planar networks from cylindrical pipes and elbows of varying length, diameter, and angle. Simulations are carried out using a finite volume based computational fluid dynamics software. A methodology is provided to discuss the source code of the executable created to automate the modeling process. This process begins with the network creation from an existing code, which generates sets of random pore networks called "realizations" and ends with linear and polynomial regressions used to provide curve fits for Darcy's law and Forchheimer's equation. Findings of these parameters are presented for varying porosity values of Berea sandstone simulated with single phase liquid water. Results show that the model follows Forchheimer's equation for certain porosities and follows experimental results. Finally, remarks on future work are given and a closing summary is presented.

CHAPTER 1

INTRODUCTION

Porous media exist in a number of naturally occurring substances and man-made materials. Rocks and soils are common examples of natural porous media. Other porous media examples include biological sources such as our skin and bones. Packed beds of beads, cement, and ceramics are all examples of man made porous media. In each example there is a medium or material which contains void spaces known as pores that fluid may pass through. Figure 1 below is an illustration of a porous medium with the pore space and medium space labeled. The pores shown have varying shape, length, diameter, and connectivity.

Porous media have numerous applications in applied sciences areas such as geoscience and petroleum engineering. Many applications involving porous media are

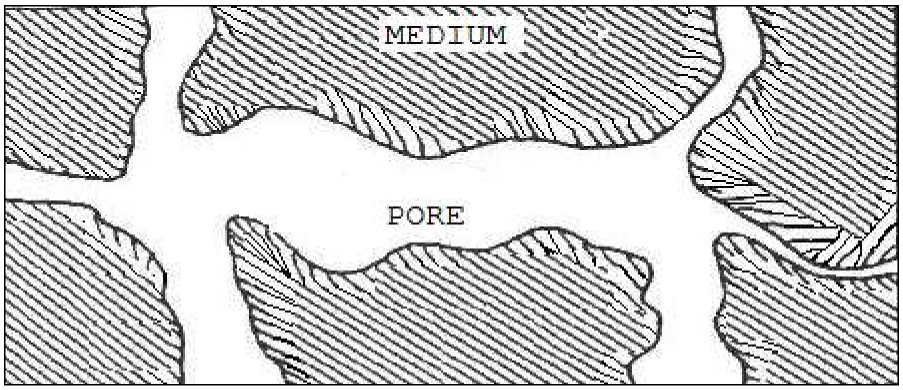

Figure 1. Two-dimensional illustration of a porous medium.

related to aquifers and petroleum reservoirs, which are porous media that contain resources like water, natural gas, and crude oil. Understanding how fluid travels through these porous media, especially at high flow rates, is essential in recovery of these resources.

Porous media may be studied experimentally or by using computational models. Experiments involve taking samples of a porous media and applying a pressure gradient from one end of the sample to the other[1]. This process emulates what occurs naturally in porous media, but it can be expensive and difficult to implement. Computational models make idealizations of real porous media by treating the pore space as an interconnected network of common geometric shapes such as spheres and cylinders. Models designed in this way are well suited for numerical methods used to determine the desired properties.

The goal of this work is to create a stochastic, computational model for porous media, namely Berea sandstone. This model is given the name Porous Media Computational Fluid Dynamics or PMCFD. PMCFD consists of pore space networks composed of cylinders connected by elbows. These cylinders and elbows have varying geometry (length, diameter, spatial orientation, etc.) determined by another model known as Flow

Through Porous Media or FTPM[2-4]. FTPM generates data files, called "realizations", which contain geometric data based on measurements taken from real Berea sandstone[5]. These data files are used by PMCFD to create three-dimensional pore space networks within a plane. Figure 2 below is an example of a pore space network created using PMCFD. These networks are created, meshed into finite volumes, and simulated with fluid flow using an Ansys® Fluid Dynamics software package comprised of the programs Gambit®[6] and Fluent®[7] (See http://www.ansys.com for more information on these products). Finally, PMCFD uses linear and polynomial regressions to curve fit pressure and volume flow rate data for comparison to two empirical equations, Darcy's law[8] and Forchheimer's equation[9]. The PMCFD model and all of its functions described above are automated by an executable file called *pmcfd.exe*.

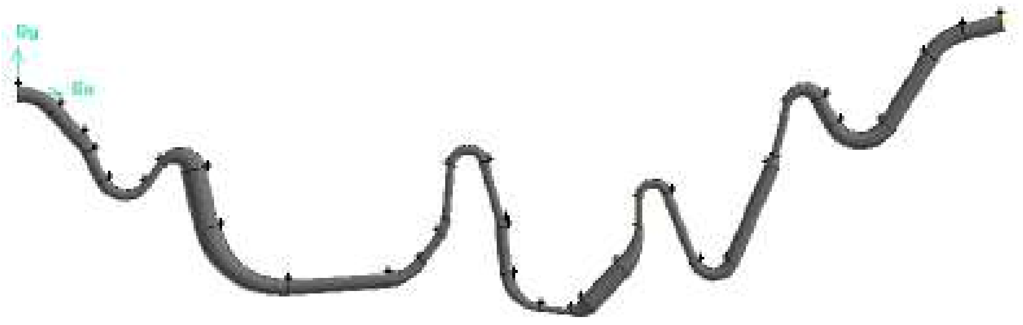

Figure 2. Pore space network created with the PMCFD model.

CHAPTER 2

BACKGROUND

Porous media contain networks of interconnected pore space that differ depending on the medium and its dimensions. An important property of porous media is the permeability, which is a measure of a porous medium's capability of transferring fluid throughout the pore space within the medium. Henry Darcy discovered this property in 1856 when he developed an experimental apparatus to measure water flowing through vertically oriented sand packed columns[1,8]. His experiments led to him to propose a relationship known as Darcy's law in Equation 1 below.

$$q = K(h_1 - h_2)/L \qquad \text{(Eq. 1)}$$

Here q (m/s) is the specific discharge from the column, K (m/s) is a proportionality constant, L (m) is the height of the column, and h_1-h_2 (m) is the difference in fluid height above the sand pack. It can be shown that for a one dimensional, horizontally oriented column, Equation 1 may be restated as Equation 2.

$$-\frac{dP}{dx} = \frac{\mu}{\kappa} u_f \qquad \text{(Eq. 2)}$$

Here dP/dx (N/m^3) is the pressure gradient along the x axis or length of the column, μ (N-s/m^2) is the fluid viscosity, u_f

(m/s) is the filtration velocity or ratio of total pore space volume flow rate (m^3/s) to total pore space area (m^2), and κ (m^2) is the average medium permeability. Permeability is also expressed in Darcy (D) or milliDarcy (mD) units where 1 D = $9.869233 \times 10^{-13} m^2$.

Equation 2 describes pressure losses in a porous medium due to viscous effects as a linear relationship between pressure gradient and filtration velocity. This relationship is generally considered valid for small flow rates within a medium. A good way to define flow rates within a medium is to use a dimensionless parameter known as Reynolds number shown in Equation 3.

$$\mathrm{Re} = \frac{\rho u D}{\mu} \qquad \text{(Eq. 3)}$$

Here ρ (kg/m^3) is the fluid density, u (m/s) is the mean velocity, D (m) is the characteristic diameter, and μ ($N\text{-}s/m^2$) is the fluid viscosity. Equation 2 is generally considered to describe flow in porous media for $0 < Re < 1$ which occurs often in different media[10].

In 1901, Philippe Forchheimer discovered that for higher flow rates ($Re > 1$) in porous media, the pressure gradient begins to deviate from a linear relationship[9]. He proposed adding a quadratic term to Darcy's law to describe this deviation as seen in Equation 4 on the next page.

$$-\frac{dP}{dx} = \frac{\mu}{\kappa} u_f + \rho \beta u_f^2 \qquad \text{(Eq. 4)}$$

This is known as Forchheimer's equation and β (m^{-1}) is often referred to as Forchheimer's coefficient. The quadratic term relates pressure losses within a porous media to inertial dissipation. Forchheimer's equation has been shown[10,11] to be valid for $Re < 100$ and $Re < 300$, respectively.

Both Equation 2 and Equation 4 are helpful in understanding flow in porous media, but solving for the unknown parameters is not simple. To experimentally determine κ and β requires constructing an apparatus that can apply a wide variety of pressure gradients across numerous samples of porous media. This can be an expensive and time consuming process. Therefore, computational models have been developed to simulate flow in porous media and numerically solve Equations 2 and 4 for κ and β.

Several models have been created to study flow in porous media as well other properties. Many of the models are stochastic[2-5,12-21] meaning that pore space geometry within model is determined randomly according to pre-determined distribution functions.

Balhoff and Wheeler[12] used a pore-scale network model to study the applicability of Forchheimer’s equation for variable sized sphere packings as well as x-ray computed microtomography images of sandstone. The pore space in their model is comprised of pores connected by throats in sinusoidal shaped ducts.

Lin and Slattery[13] used a three-dimensional, face-centered, cubic network to create a porous media model. The pore space was represented as circular ducts with radii based as a sinusoidal function of axial position.

Adler, Jacquin, and Quiblier[14] created a three-dimensional, homogeneous, isotropic porous medium composed of cubes which were assigned to be a solid or a liquid based off a probability of 0 and 1, respectively. The pore space was modeled on two average statistical properties measured from thin sections of Fontainbleau sandstone.

Quiblier[15] developed a three-dimensional model that is based on statistical data taken thin samples of porous media subjected to a light source. A threshold for minimum light intensity was established and the reflected and transmitted light were used to determine medium and pore space, respectively.

Koplik and Lasseter[16], Thauvin and Mohanty[17], and Rajaram, Ferrand, and Celia[18] modeled porous media using a lattice structure made up of variable sized spherical pore bodies connected by cylindrical pore throats. Haring and Greenkorn[19], Chatzis and Dullien[20], and Androutsopoulos and Mann[21] all modeled pore space within a medium using randomly generated cylindrical segments from various pore size distributions.

The model used in this work is based largely on the works of Handy, Kiser, Lemley, Lao, Papavassiliou, Neeman, Simpson, Yanuka, Dullien, and Elrick[2-5,22-24]. Some of these works[22-24] involve development and use of an automated code to perform computational fluid dynamics (CFD) simulations on microscale elbows and bifurcations. Other works[2-5] involve the concepts and use of the porous media model FTPM.

FTPM is a computational method developed to determine the correlation between permeability and Forchheimer's coefficient and to investigate the factors that affect the correlation[3]. FTPM creates two-dimensional and three-dimensional pore space networks comprised of cylindrical pipes with varying size, orientation, and connectivity. Probability distribution functions, specifically experimental pore radius distributions of randomly packed glass beads (177 to 350μm) and Berea sandstone from Yanuka, Dullien, and Elrick[5], are used to generate the geometric properties of the pore space networks. Collections of the pore space networks are known as “realizations” in FTPM and geometric data describing the realizations is written to text files by FTPM.

FTPM assumes flow as steady-state, fully-developed, and incompressible within each pore network. With these assumptions, FTPM uses the Poiseuille[25] equation, conservation of mass, and mechanical energy balance equations (including assumed energy losses) along with appropriate boundary conditions to form a closed, nonlinear system of equations that are solved using Newton’s method[3] for root finding.

Finally, FTPM computes κ and β using a Monte Carlo[3] style process to check the running averages for several realizations of a given porous medium. The results of κ and β are compared to experimental results of Jones[26] for Berea sandstone or the empirical equation of Ergun and Orning[27] for packed beds of glass beads. The development of the PMCFD model is focused on extending the FTPM model by removing the assumptions based on fully-developed flow, Poiseuille’s equation, and the mechanical energy balance equation with known energy losses.

CHAPTER 3

METHODOLOGY

The implementation of the PMCFD model was achieved by writing a computer code, entitled *pmcfd.cpp*, which is based on the works of several researchers[2-5,22-24]. The PMCFD code was created using the procedural based programming language, *C++*[28]. The code is composed of 24 custom functions, contains more than 150 variables, and can be used with the Windows XP and Windows 7 operating systems. Several key ideas used in the PMCFD code were gained through review of the code created by Handy[22,23] for use with microscale elbows and bifurcations and the text by Horstmann and Budd[28]. The PMCFD code can be divided into two main methods which are: modeling of a porous medium and processing the resulting data to obtain κ and β. See Figures 3 and 4 on the next page for flowcharts of the two coding methods.

The modeling method of PMCFD utilizes many of the custom functions contained in the code. The first of these functions prompts the user to select a FTPM data file or realization. These realizations contain all of the geometrical data (i.e. lengths, diameters, angles, etc.) required to create pore space networks.

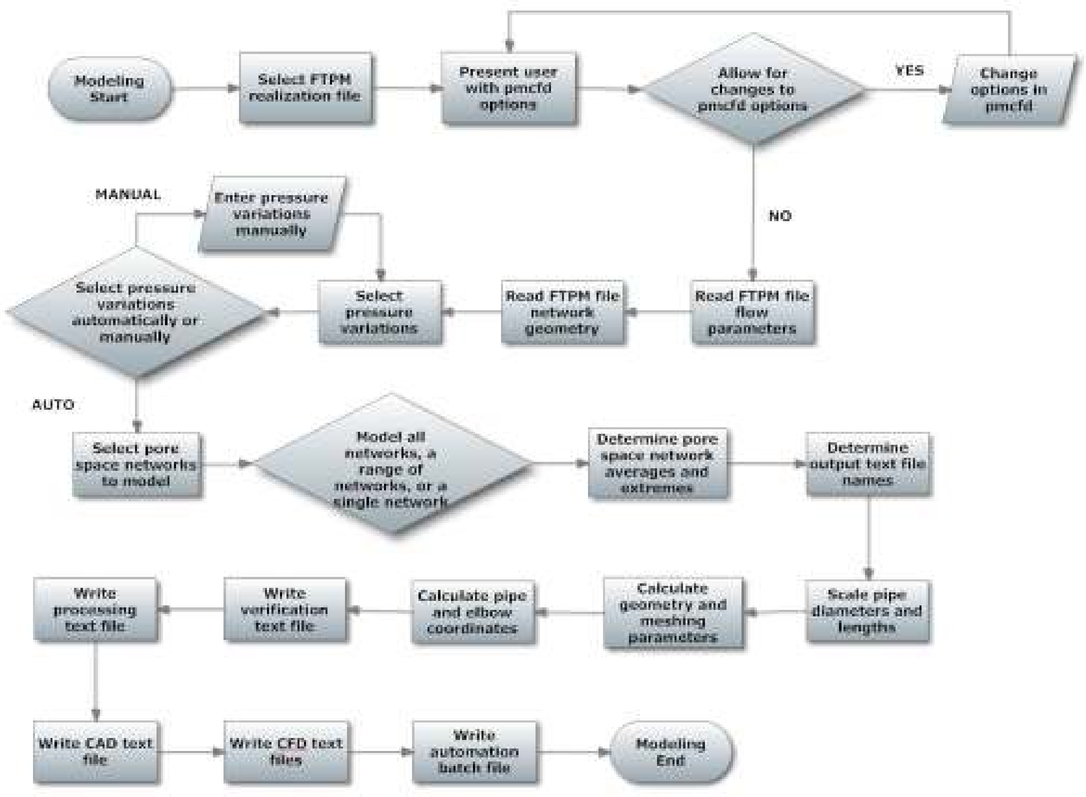
Modeling Start
Select FTPM realization file
Present user with pmcfd options
Allow for changes to pmcfd options
YES
Change options in pmcfd
NO
Read FTPM file flow parameters
Read FTPM file network geometry
Select pressure variations
Select pressure variations automatically or manually
MANUAL
Enter pressure variations manually
AUTO
Select pore space networks to model
Model all networks, a range of networks, or a single network
Determine pore space network averages and extremes
Determine output text file names
Scale pipe diameters and lengths
Calculate geometry and meshing parameters
Calculate pipe and elbow coordinates
Write verification text file
Write processing text file
Write CAD text file
Write CFD text files
Write automation batch file
Modeling End

Figure 3. Flowchart of the PMCFD modeling method.

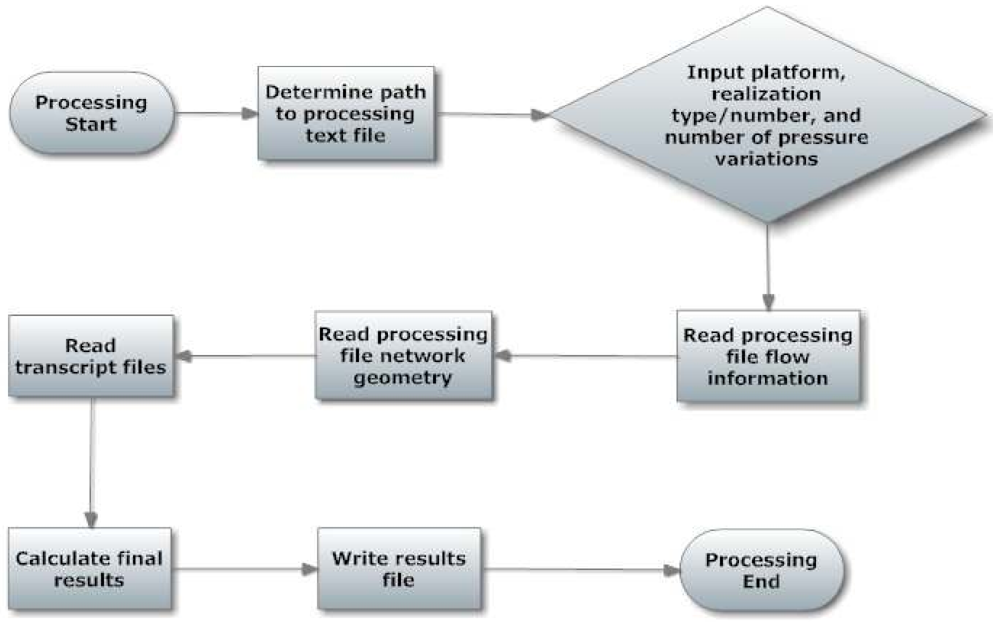
Processing Start
Determine path to processing text file
Input platform, realization type/number, and number of pressure variations
Read processing file flow information
Read processing file network geometry
Read transcript files
Calculate final results
Write results file
Processing End

Figure 4. Flowchart of the PMCFD processing method.

Each realization is composed of a variable number of pore space networks, which are in turn made up of a variable number of cylindrical pipes and interconnecting junctions. The junctions can be either elbows or bifurcations depending on the specifications made in FTPM. For PMCFD, all junctions in FTPM are pre-selected to be elbows. The number of pore space networks for a realization is determined by the specified porosity[3] in FTPM, which is shown in Equation 5 below.

$$\phi = \frac{V_{pores}}{V_{medium}} \qquad \text{(Eq. 5)}$$

Here V_{pores} (m^3) is the volume of the pore space within a medium and V_{medium} (m^3) is the total volume of the medium. See Appendix I for a sample of the geometric data contained within a realization text file for Berea sandstone with porosity of 15.0%.

Next, the user is presented with 20 different options, set to default values, which can be modified to control the output of the code. See Table 1 for a list of options and choices associated with each. These options control several aspects of the pore space network generation, meshing, and CFD simulation processes. Once the user has made any desired changes in the options shown in Table 1, the code proceeds to the next step of reading data.

PMCFD Options		
Option	**Description**	**Choices**
1	mesh volumes created in CAD software	1 - no 2 – yes
2	cylindrical pipe volume mesh type	1 - cooper 2 - tetrahedral
3	optimization criteria for reading .TRN files	1 - continuity || xyz momentum residuals 2 - xyz momentum residuals 3 - continuity residuals 4 - inlet and outlet pressure within 1% of their specified values
4	platform to use for modeling	1 – linux 2 - windows
5	number of iterations to perform	Any integer value greater than zero
6	pore space network inlet pressure	Any value greater than zero expressed in Pascals
7	number of pressure variations to use	Any integer value greater than zero
8	momentum under-relaxation factor to use	Any value between 0 and 1
9	pressure under-relaxation factor to use	Any value between 0 and 1
A	momentum discretization scheme	1 – second order upwind 6 – third-order MUSCL
B	pressure discretization scheme	10 – standard 12 – second order
C	pressure-velocity coupling	20 – SIMPLE 21 – SIMPLEC
D	pressure variation multiplier	Any value greater than zero expressed in Pascals not to exceed the ratio of option 6 to option 7
E	pore space network fluid type	1 – water-liquid 2 – water-vapor 3 - air
F	pore space network fluid density	Any value expressed in kg/m^3 between 0 and 13600
G	pore space network fluid viscosity	Any value expressed in $N\text{-}s/m^2$ between 0 and 1

Option	Description	Choices
H	residual convergence criteria	Any value between 0.000001 and 0.01
I	minimum allowed elbow angle in pore space networks	Any value expressed in degrees between 10 and 90
J	pore space network scale factor	1 – 1×10^{-4} 2 – 1×10^{-5} 3 – 1×10^{-6}
K	number of processors to use per node on cluster	Any integer value between 1 and 6

Table 1. Available options and choices in the PMCFD code.

The PMCFD code reads the chosen realization text file created by FTPM in two functions. In the first function, the code reads fixed data from the text file describing the chosen porous medium. This data may include: total number of pore space networks and junctions, number of junctions for each pore space network, domain length, domain volume, domain density, domain viscosity, Reynolds numbers, etc. In the second function, the code reads data describing each pore space network contained within the realization text file. For each junction within a network, the cylindrical pipe length, diameter, orientation angle, and position are recorded by the PMCFD code. Therefore, all the information required to model a given porous medium is obtained.

The next step in the modeling process is to select the pressure variations or *dP* to apply across each of the pore space networks. This is essential to the verification of Equation 4 that takes place later in the processing section of the code. Pressure variations are chosen based on options *6*, *7*, and *D* seen in Table 1. The inlet pressure to a network is kept constant while the outlet pressure is set to a multiple of the pressure variation multiplier. Thus for 3 pressure variations with an inlet pressure of 100 Pa and a pressure variation multiplier of 25 Pa, the outlet pressures would be specified as 75, 50, and 25 Pa, respectively. This function also allows the user to specify the outlet pressures manually.

Once the pressure variations are decided, the modeling process moves on to the pore space network selection function. This function allows the user to choose from three options. The first option is for the code to model all of the networks contained within the chosen realization text file. The second option is for the code to model a range of the networks within the file. The final option is for the code to model a single network within the file. This function is essential for running trials with different parameters on a subset of networks or for troubleshooting a defective network.

The next two functions in the modeling portion of the PMCFD code determine the pore space network averages, extremes, and file path names using the chosen realization text file.

The averages include the average cylindrical pipe diameter and pipe length. The extremes include the shortest and longest pipe lengths with corresponding pipe diameters as well as the smallest and largest pipe diameters with corresponding pipe lengths. These values are all stored to be reported by another function of the code. The file path names are based on the realization text file name, the pore space network numbers, and the chosen pressure variations.

This brings us to functions controlling the computer-aided design (CAD) and finite volume meshing software known as Gambit®[6]. The first is a scaling function used to increase all of the diameters and lengths of the cylindrical pipes from micrometers (μm) to meters (m). This is done to avoid meshing errors in Gambit® due to finite volume tolerances. Additional scaling or increasing is required for pipes that are shorter than 50μm or narrower than 10μm.

The next function calculates parameters associated with the geometry and meshing of each junction and pipe. Because each pipe has a different diameter, the junctions or elbows must be allowed to contract or expand to connect pipes of different diameters. Elbow starting and ending angles are allowed to range from 90° to -90° in increments of 1° meaning each resulting elbow created has an angle between 0° and 180°.

However, due to limitations in the CAD section of Gambit®, the actual elbow angle has a minimum limit set by option *I* listed in Table 1.

This function also calculates the average pore space network diameter using Equation 6.

$$D_{avg} = \frac{1}{L_{total}} \sum D_i L_i \qquad \text{(Eq. 6)}$$

Here, D_{avg} (m) is the average diameter of the network, L_{total} (m) is the total length of a network, and D_i (m) and L_i (m) are the diameter and length of each pipe and elbow with the network.

Finally, this function calculates the mesh interval sizes or number of discretized volumes for the elbows and pipes of the network. PMCFD requires each pore space network to maintain 100 cross sectional faces throughout the network as seen in Figure 5 below. Size functions are attached to each of the contracting or expanding elbows to maintain 100 faces.

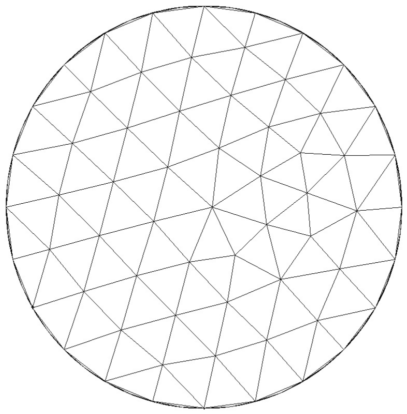

Figure 5. Cross section of a meshed cylinder in PMCFD.

These size functions are based on a geometric series with a closed form from the text by Spiegel, Lipschutz, and Liu[29] shown in Equation 7 below.

$$s = a\frac{1-r^n}{1-r} \qquad \text{(Eq. 7)}$$

Here, *s* is the sum of the series or the length of the elbow, *a* is the first term of the series or the initial mesh size, *r* is the common ratio or growth-rate of the size function, and ar^n is the last term in the series or the final mesh size. See Figure 6 on the next page for an example of size functions attached to a section of meshed network. Mesh interval sizes were also based on a mesh resolution study from a previous work[24] involving the Gambit® software.

The third function relating to geometry creation and meshing involves calculation of the pipe and elbow coordinates for the networks. These coordinates are based off of the scaled diameters, lengths, and angles for the pipes and elbows within each network. Once these coordinates are calculated, the x axis network length or *dx* is calculated and stored.

The remaining functions in the PMCFD modeling process involve the creation of several text files. The first function generates a verification file which lists all of the information read from the realization text file in a

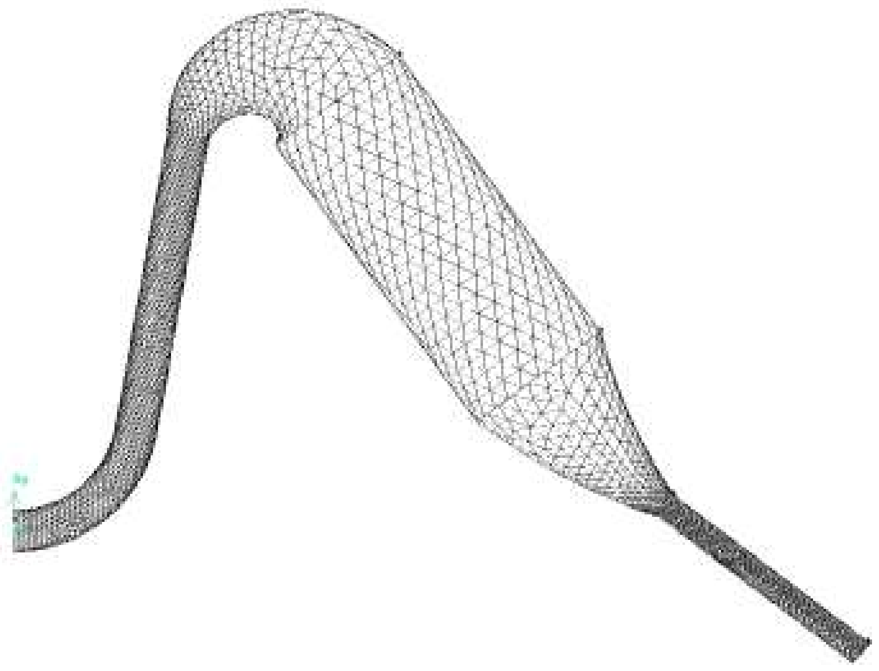

Figure 6. Size functions attached to a section of meshed pore space network.

reorganized, easy-to-read format. The second function creates a processing file written to be used in the processing functions of PMCFD. The third function writes a journal file used by the Gambit® software to create and mesh all of the pore space networks selected by the user.

The fourth function generates a set of journal files that are used by the CFD software known as Fluent®[7] to simulate flow in the meshed networks. Fluent® is used to solve the steady-state, incompressible Navier-Stokes[30] equations for pressure and velocity components within the pore space networks. Fluent performs CFD using the finite volume method with the parameters specified in Table 2 on the next page. Please refer to the

Fluent® Parameters		
Parameter	**Description**	**Choice**
grid/scale	scales a meshed pore space networks by a specified amount	1×10^{-6}
convergence criteria	defines a stopping point for simulations based on the continuity and momentum residuals	1×10^{-5}
boundary condition type	defines the type of boundaries used for the pore space network inlet and outlet	pressure
p-v-coupling	specifies how the pressure and momentum equations are coupled	SIMPLE
momentum under-relaxation factor	multiplier used to improve the momentum solutions	0.25
pressure under-relaxation factor	multiplier used to improve the pressure solution	0.75
momentum discretization scheme	determines how the momentum equations are solved within a meshed pore space network	Second Order Upwind
pressure discretization scheme	determines how the pressure equation is solved within a meshed pore space network	Second Order
iterations	determines how many times the pressure and momentum equations are solved for a pore space network	3000

Table 2. Parameters used in Fluent® for CFD simulations.

text by Versteeg and Malalasekera[30] for information on finite volume CFD simulations. The number of Fluent® journal files generated by PMCFD is equal to the number of created pore space networks multiplied by the number of pressure variations used.

The fifth and final text writing function creates a batch file that automates the execution of the Gambit® and Fluent® journal files. The end result of the modeling process is a set of transcript files equal in number to the Fluent® journal files. These transcript files contain pressure and volume flow rate values calculated by Fluent®.

After the transcript files are generated, the processing functions of PMCFD are utilized. The first two functions read the processing text file generated previously by PMCFD. From this file, PMCFD can determine how many transcript files need to be read as well as the pathway to each file. The next function involves reading of the transcript files based on the criteria specified in option *3* shown in Table 1. From each transcript file, the inlet and outlet pressures (Pa) along with the volume flow rate (m^3/s) are recorded for a pore space network.

Once all of the pressure and volume flow rates are recorded, the next function solves Equation 4 for κ and β in a series of steps. Step one calculates the pressure gradient, *dP/dx*, for all of the chosen pressure variations. This is done using averages of the recorded inlet pressures, outlet pressures, and the x axis network lengths. Step two calculates the filtration velocities, u_f, using Equation 8 below.

$$u_f = \frac{4}{\pi} \sum \frac{Q_i}{D_{i,avg}^2} \qquad \text{(Eq. 8)}$$

Here Q_i (m^3/s) and $D_{i,avg}$ (m^2) are recorded volume flow rate and average pore space diameter calculated using Equation 6, respectively, for each pore space network. Once dP/dx and u_f are known for every chosen pressure variation, step three performs linear and polynomial regressions to fit the available data to a line and second order polynomial, respectively. This is done according to the text by Chapra and Canale[31], using Equations 9 and 10.

$$y = a_0 + a_1 x \qquad \text{(Eq. 9)}$$

$$y = a_0 + a_1 x + a_2 x^2 \qquad \text{(Eq. 10)}$$

Equations 9 and 10 are solved for coefficients a_0, a_1, and a_2 and these are used to solve Equations 2 and 4, respectively. In the case of polynomial regression, matrix algebra was performed using Cramer's rule described in the text by Lindeburg[32]. These line and curve fits are used to test PMCFD's ability to describe flow in a porous medium using Darcy's[8] law and Forchheimer's[9] equation.

The last processing function in PMCFD writes results to a text file for the user. This file contains the pressure gradient and filtration velocity for each pressure variation. The file also stores the results of linear and polynomial regression including a correlation coefficient, r^2, which describes how well Equations 9 and 10 represent the given data.

Finally, the file displays the optimized values of pressure and volume flow rate read from the transcript files. See Appendix II for a sample of PMCFD functions discussed in the preceding sections. Also see Appendix III for a results file written by PMCFD that was used to model Berea sandstone with 5 pressure variations and a porosity of 10.0%.

CHAPTER 4

RESULTS AND DISCUSSION

The PMCFD model was used to generate five models of Berea sandstone from five realization text files generated by FTPM with porosity values of 10.0, 12.5, 15.0, 17.5, and 20.0%. These models contained 22, 28, 33, 39, and 44 pore space networks, respectively. An inlet pressure of 3 x 10^{12} Pa or 3 TPa was used at the inlet of each network and outlet pressures of 2.7, 2.4, 2.1, 1.8, and 1.5 TPa were used at the outlets of each network. The resulting flow rates in the models produced an average *Re* in the range of 100 to 300. The fluid modeled was liquid water at room temperature.

Modeling took place on a PC desktop as well as a 3 node, 24 processor cluster on the campus of the University of Central Oklahoma (UCO). Models generated and processed on the PC desktop took an average of six days to complete with models on the cluster completing in about half the time.

The transcript files created by Fluent® for each model were processed by PMCFD to obtain values of inlet pressure, outlet pressure, and volume flow rate for each pore space network. Pressure gradients and filtration velocities were calculated for each variation of pressure within a model. Finally, values of κ and β were calculated for each model using linear and polynomial regression. Figure 7 below shows a plot of *dP/dx* vs u_f for the 10% porosity model with regression lines included. Tables 3 and 4 on the next page list the results for both linear and polynomial regressions for all five porosity models along with the calculated values of Darcy permeability, κ_D, Forchheimer permeability, κ_F, and Forchheimer coefficient, β.

From Figure 7 along with Tables 3 and 4, it appears that polynomial regression provides a better curve fit to the *dP/dx* vs u_f data for every porosity modeled. This is seen by examining the r^2 value in Tables 3 and 4.

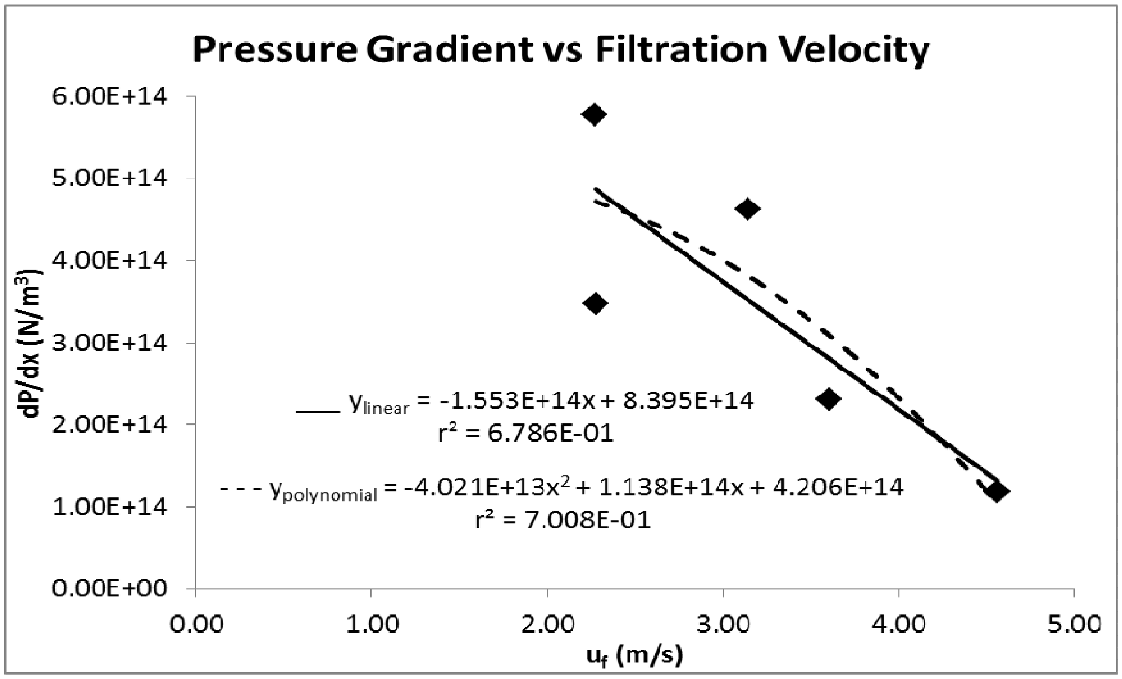

Figure 7. Pressure gradient vs filtration velocity for the 10% porosity model.

Linear Regression Results $y=a_0+a_1x$				
φ(%)	**a_0 (10^{14})**	**a_1 (10^{14})**	**r^2**	**κ_D (mD)**
10.0	8.39	-1.55	0.68	-0.64
12.5	3.00	0.23	0.02	4.31
15.0	11.2	-1.74	0.58	-0.57
17.5	-1.16	1.54	0.26	0.64
20.0	12.2	-1.60	0.47	-0.62

Table 3. Linear regression results of PMCFD for five porosity models.

Polynomial Regression Results $y=a_0+a_1x+a_2x^2$						
φ(%)	**a_0 (10^{14})**	**a_1 (10^{14})**	**a_2 (10^{14})**	**r^2**	**κ_F (mD)**	**β ($10^{10}m^{-1}$)**
10.0	4.21	1.14	-0.40	0.70	0.87	-4.03
12.5	12.4	-4.61	0.57	0.14	-0.21	5.75
15.0	4.33	1.77	-0.42	0.60	0.56	-4.24
17.5	31.1	16.2	-1.72	0.40	0.06	-17.2
20.0	-11.2	8.84	-1.11	0.62	0.11	-11.1

Table 4. Polynomial regression results of PMCFD for five porosity models.

A value of r^2 closer to one means the regression is able to describe the desired trend within the data. A value closer to zero means the desired trend cannot be described by the data. While this observation does appear to validate Forchheimer's equation over Darcy's law using PMCFD with higher flow rates, some of the regression results are inconclusive due to small values of r^2 and negative values of κ_D, κ_F, and β.

Figure 8 on the next page is a comparison of the magnitude of β vs κ_F for the five porosities modeled to experimental results of Jones[26] as well as computational results of Lao, Papavassiliou, and Neeman[3] using FTPM.

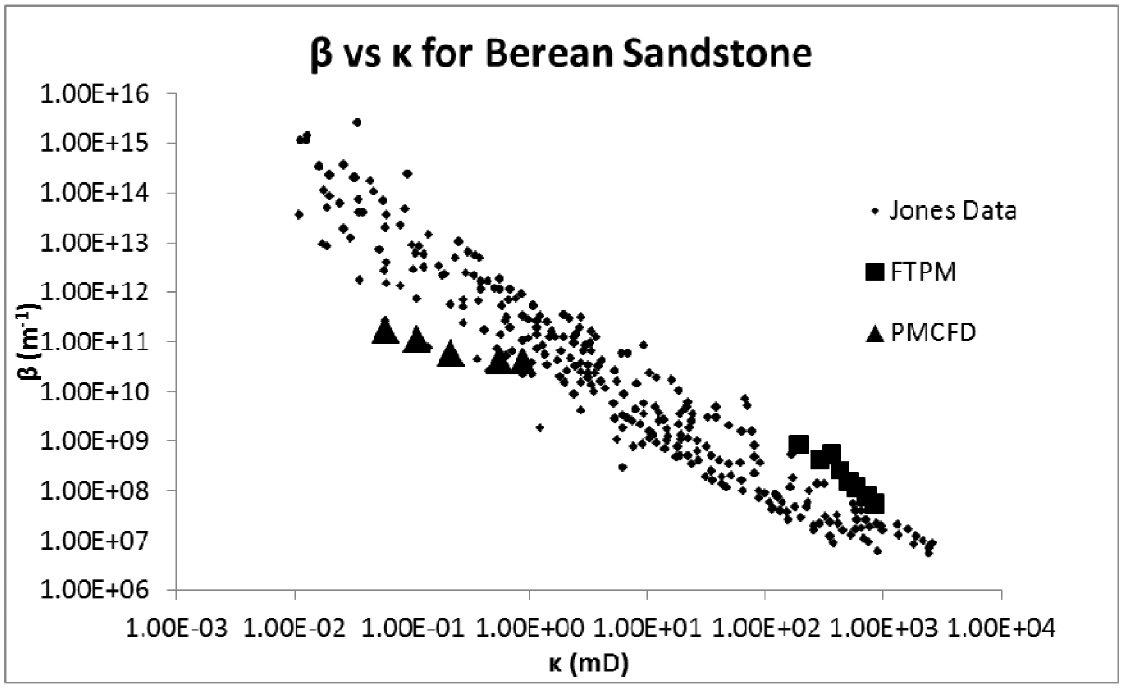

Figure 8. Comparison of PMCFD to Jones and FTPM for Berea sandstone.

All data seen in Figure 8 is for Berea sandstone. The results show better agreement between PMCFD and Jones than PMCFD and FTPM.

Currently, PMCFD is designed to model a porous medium as a collection of pore space networks comprised of cylindrical pipes connected by expanding or contracting elbows. This model is a simplification of the complex nature of porous media shown in Figure 1. Future work with this model would involve modifying PMCFD to allow for junctions or bifurcations within a network. Other considerations would include: improving the performance of UCO's cluster to decrease the time taken to model using PMCFD, modifying PMCFD to model packed beds of glass beads as well as other porous media, and general improvement of the CFD portion of PMCFD to ensure the best possible values of pressure and volume flow rate are obtained.

CHAPTER 5

SUMMARY

A three-dimensional model was developed to model fluid flow in networks of a porous medium. The PMCFD model is operated by an executable file named *pmcfd.exe* which was written using the *C++* computer language. The PMCFD code can be broken into two methods dealing with modeling of flow in a porous media and processing of the resulting data. Modeling of flow is achieved by reading a realization text file created by FTPM, generating and meshing the pore space networks in the Gambit® software, and simulating flow within the networks using the Fluent® CFD software. Processing the resulting data is achieved by reading transcript files generated by Fluent®, calculating pressure gradients and filtration velocities from that data, and performing linear and polynomial regressions on that data to solve for the unknown coefficients of Darcy’s law and Forchheimer’s equation.

PMCFD was used to model five porosities of Berea sandstone. Five pressure variations were applied to all of the networks within the five models. Regression analysis was performed to show the validity of using both Darcy’s law and Forchheimer’s equation to describe fluid flow using PMCFD. Results show Forchheimer’s equation does describe flow better than Darcy’s law in PMCFD at the higher flow rates simulated. Results also show agreement between PMCFD and experimental results obtained by Jones[22].

REFERENCES

[1] J. Bear, Dynamics of Fluids in Porous Media, (Dover, New York, 1972).

[2] E.C. Lemley, D.V. Papavassiliou, and H.J. Neeman, Non-Darcy Flow Pore Network Simulation Development and Validation of a 3D Model, Proceedings of FEDSM2007, 5th Joint ASME/JSME Fluids Engineering Conference, paper FEDSM2007-37278.

[3] H-W. Lao, D.V. Papavassiliou, and H.J. Neeman, A Pore Network Model for the Calculation of Non-Darcy Flow Coefficients in Fluid Flow Through Porous Media, Chem. Eng. Comm. 191, 1301-1338 (2004).

[4] H.J. Neeman, H-W. Lao, D. Simpson, and D.V. Papavassiliou, Multiscale Characterization of Porous Media Properties for Hydrocarbon Reservoir Simulation, Proc. SPIE 4528, 87 (2001).

[5] M. Yanuka, F.A.L. Dullien, and D.E. Elrick, Percolation Proccesses and Porous Media, J.Colloid Interface Sci. 112(1), 24-41 (1986).

[6] Gambit 2.4 User's Guide, (Fluent, Inc., New Hampshire, 2007).

[7] Fluent 6.3 User's Guide, (Fluent, Inc., New Hampshire, 2006).

[8] H. Darcy and P. Bobeck, The Public Fountains of the City of Dijon, (Kendall Hunt, Iowa, 2004).

[9] P. Forchheimer, Wasserbewegung durch Boden. Zeit. Ver. Deut. Ing., 45, 1781-1788 (1901).

[10] R.D. Barree, and M.W. Conway, Beyond Beta Factors: A Complete Model for Darcy, Forchheimer, and Trans-Forchheimer Flow in Porous Media, SPE 89325, SPE Annual Technical Conference, Houston, Texas. (2004).

[11] H. Huang and J Ayoub, Applicability of the Forchheimer Equation for Non-Darcy Flow in Porous Media, SPE 102715, SPE Annual Technical Conference, San Antonio, Texas, (2006).

[12] M.T. Balhoff and M.F. Wheeler, A Predictive Pore-Scale Model for Non-Darcy Flow in Anisotropic Porous Media. SPE 110838, SPE Annual Technical Conference, Anaheim, California. (2007).

[13] C.-Y. Lin and J.C. Slattery, Three-Dimensional, Randomized, Network Model for Two-Phase Flow Through Porous Media. AIChE J. 28(2), 311-324 (1982).

[14] P.M. Adler, C.G. Jacquin, and J.A. Quiblier, Flow in Simulated Porous Media, International Journal of Multiphase Flow 16(4), 691-712 (1990).

[15] J.A. Quiblier, A New Three-Dimensional Modeling Technique for Studying Porous Media. J. Coll. Int. Sci. 98(1), 84-102 (1984).

[16] J. Koplik and T.J. Lasseter, Two-Phase Flow in Random Network Models of Porous Media. SPE J. (February), 89-100 (1985).

[17] F. Thauvin, and K.K. Mohanty, Network modeling of Non-Darcy flow through porous media, Transport in porous media, 31, 19-37 (1998).

[18] H. Rajaram, L.A. Ferrand, and M.A. Celia, Prediction of Relative Permeabilities for Unconsolidated Soils using Pore-scale Network. Water Resources Research. Vol. 3, No. 1. pp 43-52 (1997).

[19] R.E. Haring and R.A. Greenkorn, A Statistical Model of a Porous Medium with Nonuniform Pores. AICHE J. 16(3), 477-483 (1970).

[20] I. Chatzis and F.A.L. Dullien, Modelling Pore Structure by 2-D and 3-D Networks with Application to Sandstones. J. Can. Pet. Tech. 16(1), 97-108 (1977).

[21] G.P. Androutsopoulos and R. Mann, Evaluation of Mercury Porosimeter Experiments using a Network Pore Structure Model. Chem. Eng. Sci. 34S, 1203-1212 (1979).

[22] T.A. Handy, E.C. Lemley, D.V. Papavassiliou, and H.J. Neeman, Loss Coefficients in Microelbows, Proceedings of FEDSM2009, 2009 ASME Fluids Engineering Conference, paper FEDSM2009-78517.

[23] T.A. Handy, E.C. Lemley, D.V. Papavassiliou, and H.J. Neeman, Simulations to Determine Laminar Loss Coefficients for Flow in Circular Ducts with Arbitrary Planar Bifurcation

Geometries, Proceedings of FEDSM2008, 2008 ASME Fluids Engineering Conference, paper FEDSM2008-55181.

[24] C.C. Kiser, T.A. Handy, E.C. Lemley, D.V. Papavassiliou, and H.J. Neeman, Reynolds Number Dependence for Laminar Flow Loss Coefficients in Tee and Wye Junctions, Proceedings of ASME 2010 3rd Joint US-European Fluids Engineering Summer Meeting and 8th International Conference on Nanochannels, Microchannels, and Minichannels, paper FEDSM2010-ICNMM2010-31026.

[25] F. White, Fluid Mechanics 7th Ed., (McGraw-Hill, New Jersey, 2010).

[26] S.C. Jones, Using the Inertial Coefficient, B, To Characterize Heterogeneity in Reservoir Rock, SPE 16949, SPE Annual Technical Conference, Dallas, Texas. (1987).

[27] S. Ergun, and A. A. Orning, Fluid Flow Through Randomly Packed Columns and Fluidized Beds, Ind. Eng. Chem., 41(6), 1179-1184 (1949).

[28] C. Horstmann and T. Budd, Big C++, (John Wiley & Sons, New Jersey, 2005).

[29] M.R. Spiegel, S. Lipschutz, and J. Liu, Mathematical Handbook of Formulas and Tables 3rd Ed., (McGraw-Hill, New York, 2009).

[30] H.K. Versteeg and W Malalasekera, An Introduction to Computational Fluid Dynamics: The Finite Volume Method 2nd Ed., (Pearson, New York, 2007).

[31] S.C. Chapra and R. P. Canale, Numerical Methods for Engineers 5th Ed., (McGraw-Hill, New York, 2006).

[32] M.R. Lindeburg, FE Review Manual, (Professional Publications Inc., California, 2006).

APPENDIX I

The following is a sample of geometric data contained within a realization text file for Berea sandstone with porosity of 15%.

```
PipeNetworkRealization:
  Domain Geometry Values:
    Computational Rank:  2
    Physical Rank:       2
    Domain Length:       [0.001 0.001]
    Domain Volume:       1e-06
  Structure Values:
    Maximum Number of Children Per Junction:  1
  Network Counts:
    Number of Networks:
      Total:   33
    Number of Network Junctions:
      Network #  0:       6
      Network #  1:      11
      Network #  2:       9
      Network #  3:      10
      Network #  4:       9
      Network #  5:       7
      Network #  6:      12
      Network #  7:       9
      Network #  8:       6
      Network #  9:       9
      Network # 10:       8
      Network # 11:      10
      Network # 12:      12
      Network # 13:      10
      Network # 14:       6
      Network # 15:       9
      Network # 16:      10
      Network # 17:       8
      Network # 18:      10
      Network # 19:      10
      Network # 20:       8
      Network # 21:       7
      Network # 22:      10
      Network # 23:      11
      Network # 24:       6
      Network # 25:      12
      Network # 26:       7
      Network # 27:      10
      Network # 28:      10
      Network # 29:       9
      Network # 30:       6
      Network # 31:       9
      Network # 32:      10
      Total:     296
    Pipe Networks:
      Network #0:
      Network #1:
      Network #2:
      Network #3:
      Network #4:
      Network #5:
      Network #6:
      Network #7:
      Network #8:
      Network #9:
      Network #10:
      Network #11:
      Network #12:
      Network #13:
      Network #14:
      Network #15:
      Network #16:
      Network #17:
      Network #18:
      Network #19:
      Network #20:
      Network #21:
      Network #22:
```

```
    Network #23:
    Network #24:
    Network #25:
    Network #26:
    Network #27:
    Network #28:
    Network #29:
    Network #30:
    Network #31:
    Network #32:
Network Representation Arrays:
  Network #0, Junction #0:
    Parent ID:           -1
    Number of Children:  1
    Child #0:  1
    Pipe Length in m:       0
    Pipe Diameter in m:     3.27709e-06
    Pipe Angle in degrees:  [0]
    Position in m:          [0 0.000840188]
  Network #0, Junction #1:
    Parent ID:           0
    Number of Children:  1
    Child #0:  2
    Pipe Length in m:       0.000222839
    Pipe Diameter in m:     3.27709e-06
    Pipe Angle in degrees:  [4]
    Position in m:          [0.000222296 0.000855732]
  Network #0, Junction #2:
    Parent ID:           1
    Number of Children:  1
    Child #0:  3
    Pipe Length in m:       0.000471054
    Pipe Diameter in m:     5.75291e-05
    Pipe Angle in degrees:  [-38]
    Position in m:          [0.000593492 0.000565722]
  Network #0, Junction #3:
    Parent ID:           2
    Number of Children:  1
    Child #0:  4
    Pipe Length in m:       0.000199227
    Pipe Diameter in m:     5.6621e-05
    Pipe Angle in degrees:  [49]
    Position in m:          [0.000724197 0.00071608]
  Network #0, Junction #4:
    Parent ID:           3
    Number of Children:  1
    Child #0:  5
    Pipe Length in m:       0.000310637
    Pipe Diameter in m:     4.32605e-05
    Pipe Angle in degrees:  [-39]
    Position in m:          [0.000965607 0.00052059]
  Network #0, Junction #5:
    Parent ID:           4
    Number of Children:  0
    Pipe Length in m:       0.000142165
    Pipe Diameter in m:     2.71664e-05
    Pipe Angle in degrees:  [76]
    Position in m:          [0.001 0.000658532]
  Network #1, Junction #0:
    Parent ID:           -1
    Number of Children:  1
    Child #0:  1
    Pipe Length in m:       0
    Pipe Diameter in m:     3.11892e-05
    Pipe Angle in degrees:  [0]
    Position in m:          [0 0.000394383]
  Network #1, Junction #1:
    Parent ID:           0
    Number of Children:  1
    Child #0:  2
    Pipe Length in m:       5.11241e-05
    Pipe Diameter in m:     3.11892e-05
    Pipe Angle in degrees:  [81]
    Position in m:          [7.99757e-06 0.000444878]
  Network #1, Junction #2:
    Parent ID:           1
    Number of Children:  1
    Child #0:  3
    Pipe Length in m:       5.87071e-05
    Pipe Diameter in m:     3.30425e-05
    Pipe Angle in degrees:  [-56]
    Position in m:          [4.08262e-05 0.000396207]
  Network #1, Junction #3:
    Parent ID:           2
    Number of Children:  1
    Child #0:  4
    Pipe Length in m:       0.000309549
    Pipe Diameter in m:     2.37547e-05
    Pipe Angle in degrees:  [-27]
    Position in m:          [0.000316636 0.000255675]
  Network #1, Junction #4:
    Parent ID:           3
    Number of Children:  1
    Child #0:  5
    Pipe Length in m:       4.08927e-05
    Pipe Diameter in m:     8.83812e-05
    Pipe Angle in degrees:  [-8]
    Position in m:          [0.000357131 0.000249984]
```

APPENDIX II

The following is a sample of some important functions used in the PMCFD code.

```
void simulate_networks_or_process_previous_simulations(string& decide, string&
user_fpath, string& path, string p_filename, string r_filename, string&
r_filepath)
{
    //VARIABLE DECLARATION
    bool valid=true, check=true;
    char choice1, choice2, choice3, choice4;
    /*EXE_PATH_LENGTH SUBJECT TO CHANGE BASED ON NUMBER OF CHARACTERS IN
    THIS PROGRAM'S NAME (INCLUDING UNDERSCORES AND THE .EXE EXTENSION) */
    int exe_path_length=16, path_storage_length;
    string path_storage, temp_ntype, temp_num, temp_nopv, temp_ptype, n="\n",
    u="_", s="\\", cp="Current path: ", pv="pv";

    //RETURN PATH TO USE FOR FILE WRITING
    char buffer[MAX_PATH];//always use MAX_PATH for filepaths
    GetModuleFileName(NULL,buffer,sizeof(buffer));
    path_storage=buffer;
    path_storage_length=path_storage.length();
    path=path_storage.substr(0,(path_storage_length-exe_path_length));

    //DECIDE WHETHER TO SIMULATE NEW NETWORKS OR PROCESS OLD SIMULATIONS
    while (valid)
    {
        cout << "\nPlease enter the number corresponding to the network code" <<
        " choice.\n";
        cout << "1: Simulate Networks\n" << "2: Process Previous Simulation " <<
        "Data\n";
        cin.get(choice1);
        if (choice1!='\n') cin.ignore();
        if (choice1!='1' && choice1!='2')
        {
        cout << "\nError, please try again.\n";
        }
        if (choice1=='1' || choice1=='2') valid=false;
        if (choice1=='1') decide="simulate";
        if (choice1=='2') decide="process";
    }

    //DETERMINE THE FILE FOLDER TO OPEN TO LOOK FOR THE PTRNs AND VTRNs
    if (decide=="process")
    {
        while (check)
        {
            valid=true;
            temp_ntype="@@@@@";
            temp_ptype="@@@@@";
            temp_num="##";
            temp_nopv="##";
            cout << "\nDETERMINE PATH TO PROCESSING_INFO.txt\n";
            cout << n+cp+path+temp_ptype+u+temp_ntype+temp_num+u+temp_nopv <<
            pv+s+p_filename+n;
            while (valid)
            {
                cout << "\nPlease enter the number corresponding to the " <<
                "platform type.\n";
                cout << "1: windows\n" << "2: linux\n";
                cin.get(choice2);
                if (choice2!='\n') cin.ignore();
                if (choice2!='1' && choice2!='2')
                {
                cout << "\nError, please try again.\n";
                }
                if (choice2=='1' || choice2=='2')
                valid=false;
                if (choice2=='1') temp_ptype="windows";
                if (choice2=='2') temp_ptype="linux";
            }
            valid=true;
            cout << n+cp+path+temp_ptype+u+temp_ntype+temp_num+u+temp_nopv <<
            pv+s+p_filename+n;
            while (valid)
            {
                cout << "\nPlease enter the number corresponding to the " <<
                "network type.\n";
                cout << "1: full\n" << "2: exitable\n" << "3: realization\n" <<
                "4: pipe\n";
                cin.get(choice3);
                if (choice3!='\n') cin.ignore();
                if (choice3!='1' && choice3!='2' && choice3!='3' &&
                choice3!='4')
                {
                cout << "\nError, please try again.\n";
                }
                if (choice3=='1' || choice3=='2' || choice3=='3' ||
                choice3=='4')
                valid=false;
                if (choice3=='1') temp_ntype="full";
                if (choice3=='2') temp_ntype="exitable";
                if (choice3=='3') temp_ntype="realization";
                if (choice3=='4') temp_ntype="pipe";
            }
            cout << n+cp+path+temp_ptype+u+temp_ntype+temp_num+u+temp_nopv <<
            pv+s+p_filename+n;
            cout << "\nPlease enter the network realization number (include " <<
            "'0' if <10).\n";
            cin >> temp_num;
            cin.ignore();
            cout << n+cp+path+temp_ptype+u+temp_ntype+temp_num+u+temp_nopv <<
            pv+s+p_filename+n;
            cout << "\nPlease enter the number of pressure variations.\n";
            cin >> temp_nopv;
            cin.ignore();
            cout << n+cp+path+temp_ptype+u+temp_ntype+temp_num+u+temp_nopv <<
            pv+s+p_filename+n;
            cout << "\nIs this the correct path (y/n)?\n";
            valid=true;
            while (valid)
            {
                cin.get(choice4);
                if (choice4!='\n') cin.ignore();
                if (choice4!='y' && choice4!='Y' && choice4!='n' &&
                choice4!='N')
                {
                    cout << "\nError, please try again.\n";
                }
                if (choice4=='y' || choice4=='Y' || choice4=='n' ||
                choice4=='N')
                {
                    valid=false;
                }
                if (choice4=='y' || choice4=='Y') check=false;
            }
        }
        user_fpath=path+temp_ptype+u+temp_ntype+temp_num+u+temp_nopv+pv+s+
        p_filename;
        r_filepath=path+temp_ptype+u+temp_ntype+temp_num+u+temp_nopv+pv+s+
        r_filename;
    }

    /*cout << "\nsimulate_networks_or_process_previous_simulations function " <<
    "successful.\n";*/
}


void get_info_for_network_simulation(string& i_filename, string& o_filename,
string& g_o_filename, string& ntype, string& num, string& P_L_V_M_T, string&
M_V, string& net_bat, string& time_log, string& F_S_F_C, int& N_o_C_P, string&
P_T, vector<string>& shell_script, vector<string>& hosts_filename, int& F_I,
int& N_o_P_V, double& F_I_P, double& F_M_U_R_F, double& F_P_U_R_F, int& F_M_D,
int& F_P_D, int& P_V_C, double& P_V_M, double& D_D, double& D_V, double& F_C_C,
string& F_T, double& M_E_A, double& F_S, char& O_C, string& L_F_N, string&
S_F_S)
{
    //VARIABLE DECLARATIONS
    bool check=true, valid=true, value_good;
    char choice, response, type_change, value_change[12];
    double value_convert2=0, MPV=0;
    int count, value_convert1=0;
    string spacesaver="montecarloincompressible1phaseflow_out.txt_", u="_",
    node, fmd, fpd, pvc, oc;

    //GET VALID FILE TYPE AND NUMBER AND ASK FOR CONFIRMATION OF FILENAME
    while (check)
    {
        cout << "\nPlease enter the number corresponding to the network " <<
        "type.\n";
        cout << "1: full\n" << "2: exitable\n" << "3: realization\n" <<
        "4: pipenetwork\n";
        cin.get(choice);
        if (choice!='\n') cin.ignore();
        if (choice!='1' && choice!='2' && choice!='3' && choice!='4')
        {
            cout << "\nError, please try again.\n";
            valid=false;
        }
        else
        {
            cout << "\nPlease enter the network file number (include '0' if <"
            << " 10).\n";
            cin >> num;
            cin.ignore();
            if (choice=='1')
            {
                ntype="full";
                i_filename=spacesaver+"full00"+num+".txt";
            }
            if (choice=='2')
            {
                ntype="exitable";
                i_filename=spacesaver+"exitable00"+num+".txt";
            }
            if (choice=='3')
            {
                ntype="realization";
                i_filename=spacesaver+"realization00"+num+".txt";
            }
            if (choice=='4')
            {
                ntype="pipe";
                i_filename="pipenetwork_"+num+".txt";
            }
            o_filename=ntype+num+".txt";
            g_o_filename="gambit_"+ntype+num+".txt";
            time_log="timelog_"+ntype+num+".txt";
        }
        while (valid)
        {
            cout << "\nThe file name to read is " << i_filename << endl;
            cout << "\nIs this correct y/n?\n";
            cin.get(response);
            if (response!='\n') cin.ignore();
            if (response=='y' || response=='Y')
            {
                valid=false;
                check=false;
            }
            else if (response=='n' || response=='N')
            {
                valid=false;
                check=true;
            }
            else cout << "\nError, please select y/n.\n";
        }
        valid=true;
    }

    //SET DEFAULT PARAMETERS
    M_V="yes";
    P_L_V_M_T="tetrahedral";
    O_C='4';
    oc="specified pressure tolerance of 1%";
    F_S_F_C="Linux - The Cluster";
    P_T="linux";
    F_I=3000;
    F_I_P=3e12;
    F_M_D=1;
    fmd="Second Order Upwind";
    F_M_U_R_F=0.25;
```

```
F_P_D=12;
fpd="Second Order";
F_P_U_R_F=0.75;
N_o_P_V=5;
P_V_C=20;
pvc="SIMPLE";
P_V_M=3e11;
D_D=998.2;
D_V=0.001003;
F_C_C=1e-005;
F_T="water-liquid";
M_E_A=10;
F_S=1e-006;
N_o_C_P=6;
S_F_S="1e-006scale";

//DISPLAY DEFAULTS AND ALLOW FOR MODIFICATIONS
check=true;
while (check)
{
    cout << "\nELBOW NETWORK DEFAULTS\n\n";
    cout << "1) Mesh Volumes?: " << M_V << endl;
    cout << "2) Pipe Length Volume Mesh Type: " << P_L_V_M_T << endl;
    cout << "3) Optimization Criteria for reading .TRN files: " << O_C <<
    " - " << oc << endl;
    cout << "4) Fluent Script Files written for: " << F_S_F_C << endl;
    cout << "5) Number of Iterations in Fluent: " << F_I << endl;
    cout << "6) Network Inlet Pressure in Fluent: " << F_I_P << "Pa\n";
    cout << "7) Number of Pressure Variations to Apply in Fluent: " <<
    N_o_P_V << endl;
    cout << "8) Momentum Under-Relaxation factor in Fluent: " <<
    F_M_U_R_F << endl;
    cout << "9) Pressure Under-Relaxation factor in Fluent: " <<
    F_P_U_R_F << endl;
    cout << "A) Momentum Discretization Scheme in Fluent: " <<
    F_M_D << " - " << fmd << endl;
    cout << "B) Pressure Discretization Scheme in Fluent: " <<
    F_P_D << " - " << fpd << endl;
    cout << "C) Pressure Velocity Coupling in Fluent: " <<
    P_V_C << " - " << pvc << endl;
    cout << "D) Pressure Variation Multiplier in Fluent: " <<
    P_V_M << endl;
    cout << "E) Network Fluid Type: " << F_T << endl;
    cout << "F) Network Domain Density: " << D_D << "kg/m^3\n";
    cout << "G) Network Domain Viscosity: " << D_V << "N-s/m^2\n";
    cout << "H) Fluent Convergence Criteria: " << F_C_C << endl;
    cout << "I) Minimum Elbow Angle allowed in networks: " << M_E_A <<
    " degrees\n";
    cout << "J) Fluent scale: " << F_S << endl;
    if (F_S_F_C=="Linux - The Cluster")
    {
        cout << "K) Number of Cluster Processors to use per node: " <<
        N_o_C_P << endl;
    }
    cout << "\nWould you like to change any of these defaults (y/n)?\n";
    cin.get(response);
    if (response!='\n') cin.ignore();
    if (response=='n' || response=='N' || response=='y' || response=='Y')
    {
        valid=false;
    }
    while (valid)
    {
        cout << "\nError, please select y/n.\n";
        cin.get(response);
        if (response!='\n') cin.ignore();
        if (response=='n' || response=='N' || response=='y' ||
        response=='Y') valid=false;
    }
    if (response=='n' || response=='N') check=false;
    if (response=='y' || response=='Y')
    {
        valid=true;
        cout << "\nWhich default would you like to change ";
        if (F_S_F_C=="Linux - The Cluster") cout << "(1-K)?\n";
        if (F_S_F_C=="Windows - This Computer") cout << "(1-J)?\n";
        cin.get(response);
        if (response!='\n') cin.ignore();
    }
```

```
while (valid)
{
    if (response=='1' || response=='2' || response=='3' ||
    response=='4' || response=='5' || response=='6' ||
    response=='7' || response=='8' || response=='9' ||
    response=='A' || response=='B' || response=='C' ||
    response=='D' || response=='E' || response=='F' ||
    response=='G' || response=='H' || response=='I' ||
    response=='J' || response=='K')
    {
        valid=false;
    }
    else
    {   cout << "Error, please select ";
        if (F_S_F_C=="Linux - The Cluster") cout << "(1-K)?\n";
        if (F_S_F_C=="Windows - This Computer") cout << "(1-J)?\n";
        cin.get(response);
        if (response!='\n') cin.ignore();
    }
}
if (response=='1')
{
    cout << "\nDo you want to Mesh the Volumes (1 or 2)?\n";
    cout << "1) no\n";
    cout << "2) yes\n";
    cin.get(type_change);
    if (type_change!='\n') cin.ignore();
    if (type_change=='1')
    cout << "\nThe Fluent files will not be written!!!\n";
}
if (response=='2')
{
    cout << "\nPlease choose an volume mesh type (1 or 2).\n";
    cout << "1) cooper\n";
    cout << "2) tetrahedral\n";
    cin.get(type_change);
    if (type_change!='\n') cin.ignore();
}
if (response=='3')
{
    cout << "\nWhich optimization criteria would you like to use to " <<
    "read the .TRN files after simulation (1 through 4)?\n";
    cout << "1) continuity or xyz momentum residuals\n";
    cout << "2) xyz momentum residuals\n";
    cout << "3) continuity residuals\n";
    cout << "4) fluent inlet and outlet pressures are within 1% of" <<
    " their specified values\n";
    cin.get(type_change);
    if (type_change!='\n') cin.ignore();
}
if (response=='4')
{
    cout << "\nWhich platform would you like the Fluent Script Files" <<
    " written for (1 or 2)?\n";
    cout << "1) Linux - The Cluster\n";
    cout << "2) Windows - This Computer\n";
    cin.get(type_change);
    if (type_change!='\n') cin.ignore();
}
if (response=='E')
{
    cout << "\nWhich fluid type would you like to use in Fluent " <<
    "(1 through 3)?\n";
    cout << "1) water-liquid\n";
    cout << "2) water-vapor\n";
    cout << "3) air\n";
    cin.get(type_change);
    if (type_change!='\n') cin.ignore();
}
if (response=='J')
{
    cout << "\nWhat scale would you like use in Fluent (1 through " <<
    "3)?\n";
    cout << "1) 1e-004\n";
    cout << "2) 1e-005\n";
    cout << "3) 1e-006\n";
    cin.get(type_change);
    if (type_change!='\n') cin.ignore();
}
```

```
if (response=='5' || response=='6' || response=='7' || response=='8' ||
response=='9' || response=='A' || response=='B' || response=='C' ||
response=='D' || response=='F' || response=='G' || response=='H' ||
response=='I' || response=='K')
{
    value_good=true;
    count=0;
    MPV=F_I_P/N_o_P_V;
    if (response=='5')
    {
        cout << "\nHow many iterations would you like to use in " <<
        "Fluent (>0)?\n";
    }
    if (response=='6')
    {
        cout << "\nWhat Inlet Pressure (Pa) would you like to use in" <<
        " Fluent (>0)?\n";
    }
    if (response=='7')
    {
        cout << "\nHow many Pressure Variations would you like to " <<
        "use in Fluent (better curve fits are given for 5+ " <<
        "variations)?\n";
    }
    if (response=='8')
    {
        cout << "\nWhat momentum under-relaxation factor would you " <<
        "like to use in Fluent (must be between 0 and 1)?\n";
    }
    if (response=='9')
    {
        cout << "\nWhat pressure under-relaxation factor would you " <<
        "like to use in Fluent (must be between 0 and 1)?\n";
    }
    if (response=='A')
    {
        cout << "\nWhat momentum discretization scheme would you " <<
        "like to use in Fluent (enter the appropriate number)?\n";
        cout << "1 - Second Order Upwind\n";
        cout << "6 - Third-Order MUSCL\n";
    }
    if (response=='B')
    {
        cout << "\nWhat pressure discretization scheme would you " <<
        "like to use in Fluent (enter the appropriate number)?\n";
        cout << "10 - Standard\n";
        cout << "12 - Second Order\n";
    }
    if (response=='C')
    {
        cout << "\nWhat pressure velocity coupling would you like " <<
        "to use in Fluent (enter the appropriate number)?\n";
        cout << "20 - SIMPLE\n";
        cout << "21 - SIMPLEC\n";
    }
    if (response=='D')
    {
        cout << "\nWhat pressure variation multiplier would you like" <<
        " to use in Fluent (must not exceed " << MPV << ")?\n";
    }
    if (response=='F')
    {
        cout << "\nWhat fluid density (kg/m^3) would you like to use" <<
        " (must be between 0 and 13600)?\n";
    }
    if (response=='G')
    {
        cout << "\nWhat fluid viscosity (N-s/m^2) would you like to " <<
        "use (must be between 0 and 1)?\n";
    }
    if (response=='H')
    {
        cout << "\nWhat Convergence criteria would you like to use " <<
        "in Fluent (must be between 0.000001 and 0.01)?\n";
    }
    if (response=='I')
    {
        cout << "\nWhat minimum angle (degrees) would you like to " <<
        "allow in the networks (must be between 10 and 90)?\n";
    }
    if (response=='K')
    {
        cout << "\nHow many processors would you like to use on the " <<
        "cluster per node(1 to 6)?\n";
    }
    cin >> value_change;
    cin.ignore(12,'\n');
    if (value_change[0]=='-') value_good=false;
    while (value_good)
    {
        if ((response=='5' || response=='7' || response=='A' ||
        response=='B' || response=='C' || response=='K') &&
        isdigit(value_change[count])) count++;
        else if ((response=='6' || response=='8' || response=='9' ||
        response=='D' || response=='F' || response=='G' ||
        response=='H' || response=='I') &&
        (isdigit(value_change[count]) || value_change[count]=='.'))
        {
            count++;
        }
        else value_good=false;
        if (count>strlen(value_change)-1) value_good=false;
    }
    if (count>strlen(value_change)-1 && (response=='5' ||
    response=='7' || response=='A' || response=='B' || response=='C' ||
    response=='K')) value_convert1=atoi(value_change);
    if (count>strlen(value_change)-1 && (response=='6' ||
    response=='8' || response=='9' || response=='D' || response=='F' ||
    response=='G' || response=='H' || response=='I'))
    {
        value_convert2=atof(value_change);
    }
}
if (response=='1' || response=='2' || response=='3' || response=='4' ||
response=='5' || response=='6' || response=='7' || response=='8' ||
response=='9' || response=='A' || response=='B' || response=='C' ||
response=='D' || response=='E' || response=='F' || response=='G' ||
response=='H' || response=='I' || response=='J' || response=='K')
{
    valid=true;
}
while (valid)
{
    if (response=='1' && (type_change=='1' || type_change=='2'))
    {
        if (type_change=='1') M_V="no";
        if (type_change=='2') M_V="yes";
    }
    else if (response=='2' && (type_change=='1' || type_change=='2'))
    {
        if (type_change=='1') P_L_V_M_T="tetrahedral";
        if (type_change=='2') P_L_V_M_T="cooper";
    }
    else if (response=='3' && (type_change=='1' || type_change=='2' ||
    type_change=='3' || type_change=='4'))
    {
        if (type_change=='1')
        {
            oc="continuity or xyz momentum residuals";
            O_C='1';
        }
        if (type_change=='2')
        {
            oc="xyz momentum residuals";
            O_C='2';
        }
        if (type_change=='3')
        {
            oc="continuity residuals";
            O_C='3';
        }
        if (type_change=='4')
        {
            oc="specified pressure tolerance of 1%";
            O_C='4';
        }
    }
    else if (response=='4' && (type_change=='1' || type_change=='2'))
```

```
{
    if (type_change=='1')
    {
        F_S_F_C="Linux - The Cluster";
        P_T="linux";
    }
    if (type_change=='2')
    {
        F_S_F_C="Windows - This Computer";
        P_T="windows";
    }
}
else if (response=='5' && count>strlen(value_change)-1 &&
value_convert1>0) F_I=value_convert1;
else if (response=='6' && count>strlen(value_change)-1 &&
value_convert2>0)
{
    F_I_P=value_convert2;
    P_V_M=0.5*F_I_P/N_o_P_V;
}
else if (response=='7' && count>strlen(value_change)-1 &&
value_convert1>=1)
{
    N_o_P_V=value_convert1;
    MPV=F_I_P/N_o_P_V;
    if (0.5*MPV>P_V_M) P_V_M=0.5*MPV;
}
else if ((response=='8' || response=='9' || response=='G') &&
count>strlen(value_change)-1 &&
value_convert2>0 && value_convert2<=1)
{
    if (response=='8') F_M_U_R_F=value_convert2;
    if (response=='9') F_P_U_R_F=value_convert2;
    if (response=='G') D_V=value_convert2;
}
else if (response=='A' && count>strlen(value_change)-1 &&
(value_convert1==1 || value_convert1==6 ))
{
    F_M_D=value_convert1;
    if (F_M_D==1) fmd="Second Order Upwind";
    if (F_M_D==6) fmd="Third-Order MUSCL";
}
else if (response=='B' && count>strlen(value_change)-1 &&
(value_convert1==10 || value_convert1==12 ))
{
    F_P_D=value_convert1;
    if (F_P_D==10) fpd="Standard";
    if (F_P_D==12) fpd="Second Order";
}
else if (response=='C' && count>strlen(value_change)-1 &&
(value_convert1==20 || value_convert1==21 ))
{
    P_V_C=value_convert1;
    if (P_V_C==20) pvc="SIMPLE";
    if (P_V_C==21) pvc="SIMPLEC";
}
else if (response=='D' && count>strlen(value_change)-1 &&
value_convert2>0 && value_convert2<=MPV) P_V_M=value_convert2;
else if (response=='E' && (type_change=='1' || type_change=='2' ||
type_change=='3'))
{
    if (type_change=='1')
    {
        F_T="water-liquid";
        D_D=998.2;
        D_V=1.003e-03;
    }
    if (type_change=='2')
    {
        F_T="water-vapor";
        D_D=0.5542;
        D_V=1.34e-05;
    }
    if (type_change=='3')
    {
        F_T="air";
        D_D=1.225;
        D_V=1.7894e-05;
    }
```

```
            }
            else if (response=='F' && count>strlen(value_change)-1 &&
            value_convert2>0 && value_convert2<=13600) D_D=value_convert2;
            else if (response=='H' && count>strlen(value_change)-1 &&
            value_convert2>=1e-006 && value_convert2<=1e-002)
            {
                F_C_C=value_convert2;
            }
            else if (response=='I' && count>strlen(value_change)-1 &&
            value_convert2>=10 && value_convert2<=90) M_E_A=value_convert2;
            else if (response=='J' && (type_change=='1' || type_change=='2' ||
            type_change=='3'))
            {
                if (type_change=='1')
                {
                    F_S=1e-004;
                    S_F_S="1e-004scale";
                }
                if (type_change=='2')
                {
                    F_S=1e-005;
                    S_F_S="1e-005scale";
                }
                if (type_change=='3')
                {
                    F_S=1e-006;
                    S_F_S="1e-006scale";
                }
            }
            else if (response=='K' && count>strlen(value_change)-1 &&
            value_convert1<=6 && value_convert1>0) N_o_C_P=value_convert1;
            else cout << "\nThe entry was invalid.  Please try again\n\n";
            valid=false;
        }
        valid=true;
    }

    //NAME THE NETWORK BATCH FILE & SHELL SCRIPT FILE ACCORDING TO TYPE
    net_bat=P_T+"_batch_"+ntype+num+".bat";
    if (P_T=="linux")
    {
        cout << "You have selected the cluster (linux) to run the " <<
        "simulations, please specify a folder name to be used on the cluster" <<
        " (e.g. 12point5percentporosity)\n";
        getline (cin,L_F_N,'\n');
        for (int i=0; i<3; i++)
        {
            //CONVERT NODE NUMBER INTO STRING
            ostringstream nodeoutstr;
            nodeoutstr << i;
            node="node"+nodeoutstr.str();
            shell_script[i]=node+"_shell_script.sh";
            hosts_filename[i]=node+"_fluent_hosts.hosts";
        }
    }

    //cout << "\nget_simulation_info function successful.\n";
}



void calculate_junction_coordinates(int T_N_o_N, int T_N_o_J, vector<int> N_J,
vector<double> S_P_L, vector<double> S_P_D, vector<double> C_E_A, vector<double>
S_P_R, vector<double> E_R, vector<double> D_A, vector<double>& S_X_C,
vector<double>& S_Y_C, vector<double>& M_X_C, vector<double>& M_Y_C,
vector<double>& E_X_C, vector<double>& E_Y_C, vector<int>& C_S_N,
vector<double>& Ac_D_L_x, double& Av_D_L_x, double& T_D_L_x, int S_N_N, int
E_N_N, double F_S)

{
    //VARIABLE DECLARATIONS AND INITIALIZATION
    int CNoJ=0, NNoJ=0, CToJ=0, jn, start_angle_quad, mid_angle_quad,
    end_angle_quad, counter=0;
    double start_angle, mid_angle, end_angle, act_start_angle, act_mid_angle,
    act_end_angle, cur_x, cur_y, start_radius, end_radius, length, eoc_length,
    domain_start, domain_end, numnet=E_N_N-S_N_N+1, ZT, pi;
    ZT=1e-10;
    pi=atan(1.0)*4;
    T_D_L_x=0;
```

```
//CALCULATE ALL COORDINATES FOR EACH JUNCTION IN ALL NETWORKS
 for (int i=0; i<T_N_o_N; i++)
 {
     C_S_N[i]=counter;
     NNoJ=N_J[i];
     CToJ=CNoJ+NNoJ;
     cur_x=0;
     cur_y=0;
     for (int j=CNoJ; j<CToJ-1; j++)
     {
         start_angle=C_E_A[j];
         end_angle=C_E_A[j+1];
         mid_angle=(start_angle+end_angle)/2;
         start_radius=S_P_R[j];
         end_radius=S_P_R[j+1];
         length=S_P_L[j+1];
         eoc_length=0.25*S_P_L[j+1];//must equal E_L in function above
         //CALCULATE ELBOW AND SURFACE PLANE COORDINATES FOR UNEQUAL ANGLES
         if (start_angle!=end_angle)
         {
             //DETERMINE QUADRANTS FOR START, MIDDLE, & END ANGLES
             if (start_angle<=0 && mid_angle<0 && end_angle<0 &&
             start_angle>end_angle)
             {
                 start_angle_quad=1;
                 mid_angle_quad=1;
                 end_angle_quad=1;
             }
             if (start_angle>0 && mid_angle>0 && end_angle>=0 &&
             start_angle>end_angle)
             {
                 start_angle_quad=2;
                 mid_angle_quad=2;
                 end_angle_quad=2;
             }
             if (start_angle<0 && mid_angle<0 && end_angle<=0 &&
             start_angle<end_angle)
             {
                 start_angle_quad=3;
                 mid_angle_quad=3;
                 end_angle_quad=3;
             }
             if (start_angle>=0 && mid_angle>0 && end_angle>0 &&
             start_angle<end_angle)
             {
                 start_angle_quad=4;
                 mid_angle_quad=4;
                 end_angle_quad=4;
             }
             if (start_angle>0 && mid_angle>0 && end_angle<0)
             {
                 start_angle_quad=2;
                 mid_angle_quad=2;
                 end_angle_quad=1;
             }
             if (start_angle>0 && mid_angle<=0 && end_angle<0)
             {
                 start_angle_quad=2;
                 mid_angle_quad=1;
                 end_angle_quad=1;
             }
             if (start_angle<0 && mid_angle<0 && end_angle>0)
             {
                 start_angle_quad=3;
                 mid_angle_quad=3;
                 end_angle_quad=4;
             }
             if (start_angle<0 && mid_angle>=0 && end_angle>0)
             {
                 start_angle_quad=3;
                 mid_angle_quad=4;
                 end_angle_quad=4;
             }
             //DETERMINE ACTUAL START, MIDDLE, & END ANGLES
             if (start_angle_quad==1 || start_angle_quad==2)
             act_start_angle=(start_angle+90)*pi/180;
             if (start_angle_quad==3 || start_angle_quad==4)
             act_start_angle=(start_angle+270)*pi/180;
```

```
    if (mid_angle_quad==1 || mid_angle_quad==2)
    act_mid_angle=(mid_angle+90)*pi/180;
    if (mid_angle_quad==3 || mid_angle_quad==4)
    act_mid_angle=(mid_angle+270)*pi/180;
    if (end_angle_quad==1 || end_angle_quad==2)
    act_end_angle=(end_angle+90)*pi/180;
    if (end_angle_quad==3 || end_angle_quad==4)
    act_end_angle=(end_angle+270)*pi/180;
    //DETERMINE ELBOW AND SURFACE PLANE VERTICES COORDINATES
    S_X_C[counter]=cur_x;
    S_Y_C[counter]=cur_y;
    M_X_C[counter]=cur_x+E_R[j]*(cos(act_mid_angle)-
    cos(act_start_angle));
    M_Y_C[counter]=cur_y+E_R[j]*(sin(act_mid_angle)-
    sin(act_start_angle));
    E_X_C[counter]=cur_x+E_R[j]*(cos(act_end_angle)-
    cos(act_start_angle));
    E_Y_C[counter]=cur_y+E_R[j]*(sin(act_end_angle)-
    sin(act_start_angle));
    //DETERMINE X DOMAIN LENGTH
    if (j==CNoJ) domain_start=S_X_C[counter];
    //CORRECT COORDINATES BELOW ZERO TOLERANCE
    if (fabs(S_X_C[counter])<ZT) S_X_C[counter]=0;
    if (fabs(S_Y_C[counter])<ZT) S_Y_C[counter]=0;
    if (fabs(M_X_C[counter])<ZT) M_X_C[counter]=0;
    if (fabs(M_Y_C[counter])<ZT) M_Y_C[counter]=0;
    if (fabs(E_X_C[counter])<ZT) E_X_C[counter]=0;
    if (fabs(E_Y_C[counter])<ZT) E_Y_C[counter]=0;
    //UPDATE POSITION & COUNTER
    cur_x=E_X_C[counter];
    cur_y=E_Y_C[counter];
    counter++;
}
//CALCULATE EXPANSION OR CONTRACTION AND SURFACE PLANE COORDINATES
//FOR EQUAL ANGLES
if (start_angle==end_angle)
{
    //DETERMINE ACTUAL ANGLES FOR TOP & BOTTOM VERTICES
    if (end_angle<=0) act_end_angle=(end_angle+90)*pi/180;
    if (end_angle>0) act_end_angle=(end_angle-90)*pi/180;
    //DETERMINE EOC PIPE AND SURFACE PLANE VERTICES COORDINATES
    S_X_C[counter]=cur_x;
    S_Y_C[counter]=cur_y;
    M_X_C[counter]=cur_x+0.5*eoc_length*cos(end_angle*pi/180);
    M_Y_C[counter]=cur_y+0.5*eoc_length*sin(end_angle*pi/180);
    E_X_C[counter]=cur_x+eoc_length*cos(end_angle*pi/180);
    E_Y_C[counter]=cur_y+eoc_length*sin(end_angle*pi/180);
    //DETERMINE X DOMAIN LENGTH
    if (j==CNoJ) domain_start=S_X_C[counter];
    //CORRECT COORDINATES BELOW ZERO TOLERANCE
    if (fabs(S_X_C[counter])<ZT) S_X_C[counter]=0;
    if (fabs(S_Y_C[counter])<ZT) S_Y_C[counter]=0;
    if (fabs(M_X_C[counter])<ZT) M_X_C[counter]=0;
    if (fabs(M_Y_C[counter])<ZT) M_Y_C[counter]=0;
    if (fabs(E_X_C[counter])<ZT) E_X_C[counter]=0;
    if (fabs(E_Y_C[counter])<ZT) E_Y_C[counter]=0;
    //UPDATE POSITION & COUNTER
    cur_x=E_X_C[counter];
    cur_y=E_Y_C[counter];
    counter++;
}
//DETERMINE PIPE LENGTH AND SURFACE PLANE VERTICES COORDINATES
if (end_angle<=0) act_end_angle=(end_angle+90)*pi/180;
if (end_angle>0) act_end_angle=(end_angle-90)*pi/180;
S_X_C[counter]=cur_x;
S_Y_C[counter]=cur_y;
M_X_C[counter]=cur_x+0.5*length*cos(end_angle*pi/180);
M_Y_C[counter]=cur_y+0.5*length*sin(end_angle*pi/180);
E_X_C[counter]=cur_x+length*cos(end_angle*pi/180);
E_Y_C[counter]=cur_y+length*sin(end_angle*pi/180);
//DETERMINE X DOMAIN LENGTH
if (j==CToJ-2) domain_end=E_X_C[counter];
//CORRECT COORDINATES BELOW ZERO TOLERANCE
if (fabs(S_X_C[counter])<ZT) S_X_C[counter]=0;
if (fabs(S_Y_C[counter])<ZT) S_Y_C[counter]=0;
if (fabs(E_X_C[counter])<ZT) E_X_C[counter]=0;
if (fabs(E_Y_C[counter])<ZT) E_Y_C[counter]=0;
//UPDATE POSITION & COUNTER
cur_x=E_X_C[counter];
```

```
            cur_y=E_Y_C[counter];
            counter++;
        }
        //CALCULATE ACTUAL DOMAIN LENGTH ALONG THE X DIRECTION INCLUDING SCALING
        Ac_D_L_x[i]=F_S*domain_end-domain_start;
        CNoJ+=N_J[i];
    }

    //CALCULATE AVERAGE DOMAIN LENGTH FOR NETWORKS SELECTED
    for (int i=S_N_N; i<=E_N_N; i++) T_D_L_x+=Ac_D_L_x[i];
    Av_D_L_x=T_D_L_x/numnet;

    //cout << "\ncalculate_junction_coordinates function successful.\n";
}


void calculate_final_results(int S_N_N, int E_N_N, vector<double> R_N,
vector<double>& R_N_A, vector<double>& R_N_S_D, vector<double> P_D_A, double&
T_A, vector<double> T_V_F_R, vector<double>& F_V, vector<double>& P_G, double&
F_C, double& F_N_P, double Av_D_L_x, vector<double> N_O_P, vector<double> N_I_P,
vector<double> A_V_F_R, double& P_C_C, double D_D, double D_V, double& P_R_C_1,
double& P_R_C_2, double& P_R_C_3, int N_o_P_V, vector<double>& R_N_I_A, double&
L_R_C_1, double& L_R_C_2, double& L_C_C, double& D_N_P, double& I_P_D,
vector<bool> N_S)
{
    //VARIABLE DECLARATIONS I
    double pi=atan(1.0)*4, nopsum, nipsum, avfrsum, rnsum, rndev_pow2_sum,
    pda_sum=0, pda_pow2_sum=0, avgnop, avgnip;
    int counter=S_N_N*N_o_P_V, numnet=E_N_N-S_N_N+1, badnet;

    for (int i=0; i<N_o_P_V; i++)
    {
        //INITIALIZE BAD NETWORK, SUM, AND AVERAGE VARIABLES
        badnet=0;
        nopsum=0;
        nipsum=0;
        avfrsum=0;
        rnsum=0;
        avgnop=0;
        avgnip=0;
        rndev_pow2_sum=0;
        //SUM THE VOLUME FLOW RATES, DIAMETER AVERAGES, PRESSURES, AND REYNOLDS
        for (int j=S_N_N; j<=E_N_N; j++)
        {
            counter=i+j*N_o_P_V;
            if (N_S[counter])
            {
                nopsum+=N_O_P[counter];
                nipsum+=N_I_P[counter];
                avfrsum+=A_V_F_R[counter];
                rnsum+=R_N[counter];
                if (i==0)
                {
                    pda_pow2_sum+=pow(P_D_A[j], 2);
                    pda_sum+=P_D_A[j];
                }
            }
            else badnet++;
        }
        //CALCULATE REYNOLDS VALUES, PRESSURE GRADIENT AND FILTRATION VELOCITY
        avgnip=nipsum/(numnet-badnet);
        avgnop=nopsum/(numnet-badnet);
        if (i==0) I_P_D=pda_sum/(numnet-badnet);
        R_N_A[i]=rnsum/(numnet-badnet);
        for (int j=S_N_N; j<=E_N_N; j++)
        {
            counter=i+j*N_o_P_V;
            if (R_N[counter]>0 && N_S[counter])
            {
                rndev_pow2_sum+=pow((R_N[counter]-R_N_A[i]), 2);
            }
        }
        if ((numnet-badnet)>1)
        {
            R_N_S_D[i]=sqrt(rndev_pow2_sum/(numnet-badnet-1));
        }
        else
        {
            R_N_S_D[i]=0;
        cout << "Standard deviation of Reynolds number could not be " <<
        "calculated for pressure variation " << i+1 << " due to lack of " <<
        "two or more network values\n";
    }
    P_G[i]=(avgnip-avgnop)/Av_D_L_x;
    T_A=pi/4*pda_pow2_sum;
    T_V_F_R[i]=avfrsum;
    F_V[i]=T_V_F_R[i]/T_A;
    R_N_I_A[i]=(D_D/D_V)*F_V[i]*I_P_D;
}

//VARIABLE DECLARATIONS II
double a0=0, a1=0, a2=0, b00=0, b01=0, b02=0, b10=0, b11=0, b12=0, b20=0,
b21=0, b22=0, c0=0, c1=0, c2=0, detb=0, detb0=0, detb1=0, detb2=0, x_sum=0,
y_sum=0, prod_xy_sum=0, x_pow2_sum=0, y_pow2_sum=0, x_sum_pow2=0,
y_sum_pow2=0, fv_sum=0, fv_pow2_sum=0, fv_pow3_sum=0, fv_pow4_sum=0,
pg_sum=0, prod_fv_pg_sum=0, prod_fv_pow2_pg_sum=0, r_pow2, S_t=0,
S_r_linear=0, S_r_polynomial=0, x_avg, y_avg;

//DETERMINE FORCHHEIMER EQUATION FIT AND CORRELATION COEFFICIENTS
for (int i=0; i<N_o_P_V; i++)
{
    x_sum+=F_V[i];
    y_sum+=P_G[i];
    prod_xy_sum+=F_V[i]*P_G[i];
    x_pow2_sum+=pow(F_V[i], 2);
    y_pow2_sum+=pow(P_G[i], 2);
    fv_sum+=F_V[i];
    fv_pow2_sum+=pow(F_V[i], 2);
    fv_pow3_sum+=pow(F_V[i], 3);
    fv_pow4_sum+=pow(F_V[i], 4);
    pg_sum+=P_G[i];
    prod_fv_pg_sum+=F_V[i]*P_G[i];
    prod_fv_pow2_pg_sum+=pow(F_V[i], 2)*P_G[i];
}
x_avg=x_sum/N_o_P_V;
x_sum_pow2=pow(x_sum, 2);
y_avg=y_sum/N_o_P_V;
y_sum_pow2=pow(y_sum, 2);
a1=(N_o_P_V*prod_xy_sum-x_sum*y_sum)/(N_o_P_V*x_pow2_sum-x_sum_pow2);
a0=y_avg-a1*x_avg;
L_R_C_1=a0;
L_R_C_2=a1;
b00=N_o_P_V;
b01=fv_sum;
b02=fv_pow2_sum;
b10=fv_sum;
b11=fv_pow2_sum;
b12=fv_pow3_sum;
b20=fv_pow2_sum;
b21=fv_pow3_sum;
b22=fv_pow4_sum;
c0=pg_sum;
c1=prod_fv_pg_sum;
c2=prod_fv_pow2_pg_sum;
detb=b00*(b11*b22-b12*b21)+b01*(b12*b20-b10*b22)+b02*(b10*b21-b11*b20);
detb0=c0*(b11*b22-b12*b21)+b01*(b12*c2-c1*b22)+b02*(c1*b21-b11*c2);
detb1=b00*(c1*b22-b12*c2)+c0*(b12*b20-b10*b22)+b02*(b10*c2-c1*b20);
detb2=b00*(b11*c2-c1*b21)+b01*(c1*b20-b10*c2)+c0*(b10*b21-b11*b20);
a0=detb0/detb;
a1=detb1/detb;
a2=detb2/detb;
P_R_C_1=a0;
P_R_C_2=a1;
P_R_C_3=a2;
for (int i=0; i<N_o_P_V; i++)
{
    S_t+=pow((P_G[i]-y_avg), 2);
    S_r_linear+=pow((P_G[i]-L_R_C_1-L_R_C_2*F_V[i]), 2);
    S_r_polynomial+=pow((P_G[i]-P_R_C_1-P_R_C_2*F_V[i]-P_R_C_3*
    pow(F_V[i], 2)), 2);
}
r_pow2=(S_t-S_r_linear)/S_t;
L_C_C=r_pow2;
r_pow2=(S_t-S_r_polynomial)/S_t;
P_C_C=r_pow2;

//DETERMINE FORCHHEIMER COEFFICIENT AND PERMEABILITIES
D_N_P=D_V/L_R_C_2;
```

```
    F_N_P=D_V/P_R_C_2;
    F_C=P_R_C_3/D_D;

    //cout << "\ncalculate_final_results function successful.\n";
}
```

APPENDIX III

The following is a results file written by PMCFD to model Berea sandstone with 5 pressure variations and a porosity of 10%. The optimized transcript file data has been omitted.

```
RESULTS for realization00_5pv

         Inlet Pressure:       3e+012Pa
  Average Domain Length:   0.00258851m
Idealized Pipe Diameter: 5.18219e-005m

Outlet Pressure(Pa)    dP/dx(Pa/m)       u_f(m/s)  Avg. Reynolds  Reynolds S.D. Ideal Avg. Reynolds
          2.7e+012   1.18643e+014        4.55766         228.53        287.952             235.056
          2.4e+012   2.30341e+014        3.60164        182.925        181.265             185.751
          2.1e+012   3.47741e+014        2.27356        129.872        116.954             117.256
          1.8e+012    4.6228e+014        3.14493        215.344        275.667             162.196
          1.5e+012   5.77743e+014        2.27088        129.765        152.441             117.118


Linear Equation General Form: a0+a1*x

Linear Regression Fit Coefficients
a0= 8.39465e+014 a1=-1.55254e+014 r^2=     0.678645

Darcy's Law General Form: -dP/dx=(mu/kappa)u_f

User Specified Value
   mu=     0.001003N-s/m^2

Calculated Value
kappa=-6.46037e-018m^2
     =    -0.637589mD


Second Order Polynomial General Form: a0+a1*x+a2*x^2

Polynomial Regression Fit Coefficients
a0=  4.2062e+014 a1=   1.138e+014 a2=-4.02132e+013 r^2=     0.700763

Forchheimer Equation General Form: -dP/dx=(mu/kappa)u_f+(rho*beta)u_f^2

User Specified Values
  rho=        998.2kg/m^3   mu=     0.001003N-s/m^2

Calculated Values
kappa= 8.81371e-018m^2    beta=-4.02857e+010m^-1
     =     0.869845mD
```