The Development of a CFD Program to Model the Pressure Distribution of a Nanocomposite Melt Flow through a Twin-Screw Extruder

Craig Lewandowski

University of Minnesota Latin Honors Thesis
Academic Year: 2005-2006

Thesis Advisor: Dr. Traian Dumitrica

NASA Goddard Space Flight Center
Principal Investigator: Dan Powell

Abstract

A computational fluid dynamics code based on Lattice Boltzmann principles was adapted to predict the maximum pressure produced by flowing nanocomposite melts through a manifold. The manifold analyzed is attached to the twin-screw extruder associated with the NASA Oriented Nanocomposite Extrusion project. A Lattice Boltzmann model was programmed in MATLAB, and the code accepts user inputs of polymer selection, carbon nanotube concentration, and mass flow rate. These values are combined with physical properties of the nanocomposite and default model parameters. By modeling the manifold geometry, the appropriate flow boundaries were determined, and a solution matrix was produced. The velocity of simulated flow through this matrix was computed, and the appropriate conversions made to yield the maximum pressure.

Table of Contents

1. Introduction and Background…………………………………………...……….4

1.1 Project Motivation………………………………………………………4

1.2 Lattice Boltzmann Overview……………………………………………5

1.3 Project Objective Summary…………………………………………..7

2. Modeling and Simulation…………………………………………...……....7

3. Results and Discussion………………………………………………...……..13

4. Conclusions…………………………………………………………...…....16

5. References……………………………………………………………...…..18

6. Appendices……………………………………………………………...….19

Appendix A – Lattice Boltzmann Derivation and Discussion…………………...19

Appendix B - Model Parameters Associated with the D2Q9 LBM Model……...24

Appendix C – MATLAB Lattice Boltzmann Code……..…………………..…...25

1. Introduction and Background

1.1 Project Motivation

The next generation of NASA missions will necessitate revolutionary materials. In spaceflight, the driving cost is almost invariably mass, and mass is largely dictated by material selection. These future materials will be lighter, stronger, and multifunctional. Thermal conductivity, electrical conductivity, and radiation shielding are all desired properties.

It is anticipated that these materials will be achieved through nanotechnology, and presently, many efforts are focused on carbon nanotubes. Carbon nanotubes (CNTs) were discovered in 1991 by the Japanese electron microscopist Sumio Iijima. Both single-walled and multi-walled CNTs demonstrate extraordinary properties, with single-walled CNTs being notably superior. NASA is presently attempting to insert CNTs into polymer melts to fabricate the next generation of space materials.

This is being accomplished at the Goddard Space Flight Center in a collaborative project with the University of Maryland (College Park). A Werner & Pfleiderer 28mm twin-screw extruder, located at the University, is being used to melt polymer pellets and mix them with CNTs. This combination of polymer melts and CNTs results in the production of nanocomposites. Figure 1 shows a nanocomposite film exiting the twin-screw extruder.

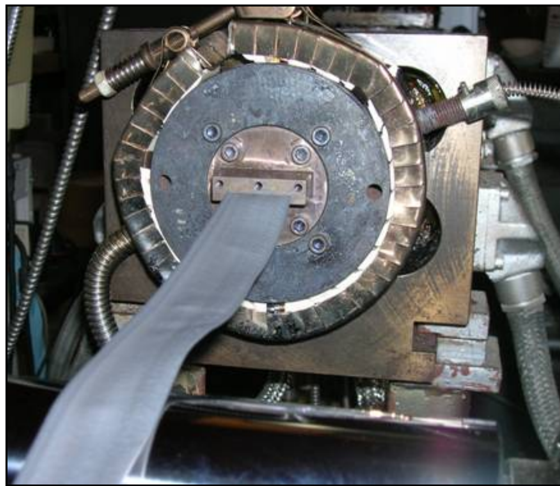

Figure 1. Nanocomposite Exiting the Twin-Screw Extruder

The nanocomposites are produced in sheets, and the ultimate objective is sheets characterized by a 1m width, 3mm height, and continuous length. The width and height are governed by a die which can be varied and is located at outlet of the twin-screw extruder. The die is manufactured in such a way so as to allow for the attachment of a microchannel array. Alignment of the CNTs is required in order to achieve multifunctionality, and this project aims to align the CNTs mechanically through the microchannel array.

Shear stresses induced by the manifold and channel walls in the array are to orient the CNTs, and a considerable amount of pressure is required in order to make this feasible. In the fall of 2004, this pressure resulted in the shattering of the silicon array then being used. At this time, this was the array attachment for the 3 in die, which constituted a major setback both financially and in terms of process development. Presently, the timetable dictates that the 1m die and array attachment will be completed in 2010. Were a rupture of a 1m manifold or microchannel array to occur, the results would be catastrophic as six-figure (minimum) manufacturing costs are anticipated.

Thus, the development of a software code to predict the behavior of the nanocomposites in the six-inch manifold will be extremely beneficial. This CFD code is to be programmed in MATLAB and based on Lattice Boltzmann principles. Specifically, this code will be utilized to calculate the maximum pressure of the laminar flow through the extrusion die. The user input-variables include polymer selection, carbon nanotube concentration, and mass flow rate.

At present, a 3 in (width) die is being used, and a 6 in die and corresponding microchannel array are in development. In two years, assuming no budgetary complications, a 1 ft die will be manufactured before the ultimate scale-up to 1m. Through alterations to the geometry specified in the program, this code will be applicable to any of these designs and any additional models that are developed.

1.2 Lattice Boltzmann Overview

In recent years, Lattice Boltzmann models (LBMs) have become increasingly popular due to their ease of implementation, extensibility, and computational efficiency [1]. Consequently, Lattice Boltzmann has become a viable alternative to traditional CFD methods. Traditional CFD relies on kinetic theory to produce continuous

macroscopic equations. These continuous equations are then discretized using a mesh, and the complication of transitioning from continuous to discrete variables can introduce errors, especially in the case of an inadequate mesh.

In contrast, the Lattice Boltzmann method discretizes variables on a microscopic level by defining particles at discrete locations with discrete velocities at discrete moments in time. Lattice Boltzmann refers to the lattice of nodes employed to solve a given problem and the Boltzmann equation from which the method is derived. In the Boltzmann equation,

$$\frac{\partial f}{\partial t} + \bar{v}\frac{\partial f}{\partial \bar{x}} + \bar{F}\frac{\partial f}{\partial \bar{v}} = \Omega(f), \tag{1}$$

f is the probability that a particle will be at a position between $\bar{x}$ and $\bar{x} + d\bar{x}$ with a velocity between $\bar{v}$ and $\bar{v} + d\bar{v}$ at time t. Thus, $f = f(\bar{x}, \bar{v}, t)$ where $\bar{x}$ and $\bar{v}$ are position and velocity vectors, respectively, and t represents time. In Equation 1, $\bar{F}$ is the external force causing the particle motion and $\Omega(f)$ is the collision operator governing interactions between the particles themselves.

The premises behind this equation allow for an exact realization of particle dynamics and have made Lattice Boltzmann a popular technique for solving small-scale fluid-flow problems. Information pertaining to the derivation and application of the Lattice Boltzmann equations is included in Appendix A.

There are several variations of LBMs, each model representing a flow in a different manner. The D2Q9 model shown in Figure 1 is a two-dimensional model (D2) with nine possible velocity vectors (Q9). At a given time step, the center particle may travel to any of the eight surrounding nodes, or it may remain stationary. Thus, a velocity vector equal to zero constitutes the ninth possible vector.

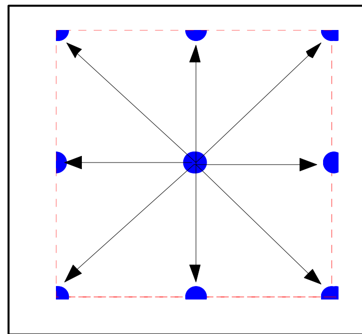

Figure 1. Visualization of D2Q9 Lattice Boltzmann Model [2]

Lattice Boltzmann can be used to simulate three-dimensional flows with models such as the D3Q19, which has motion in three dimensions and 19 associated velocity vectors.

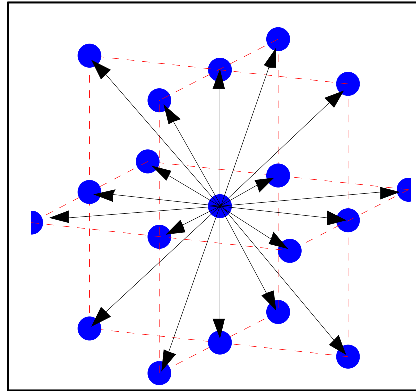

Figure 2. Visualization of D3Q19 Lattice Boltzmann Model [2]

Information detailing how the particle distribution accounts for boundary conditions, time steps, and other factors is included in Appendix A.

1.3 Project Objective Summary

The Lattice Boltzmann method was to be employed to predict the maximum pressure in the manifold attached to the twin-screw extruder based on known parameters. The calculations performed in the code are not necessarily required to determine the pressure with pinpoint accuracy. Rather, this code is to serve as a guide available to the individuals highly involved in this project. The results produced by the code may be one of any number of factors taken into consideration when preparing for extrusion runs.

2. Modeling and Simulation

A model created in ProEngineer (ProE), a CAD program, of the six-inch manifold being considered in the analysis is shown in Figure 3. This model was designed by NASA engineer Greg Martins.

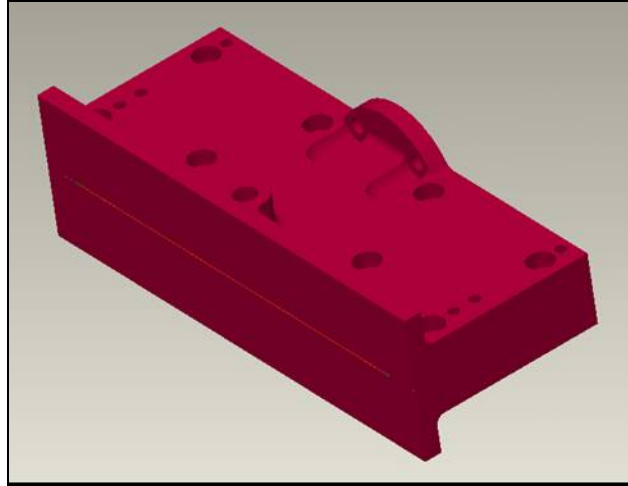

Figure 3. Complete ProE Model of 6 in Manifold

Figure 4 contains a view of the cross-section of the manifold. In this image, the nanocomposites flow over the green and orange areas.

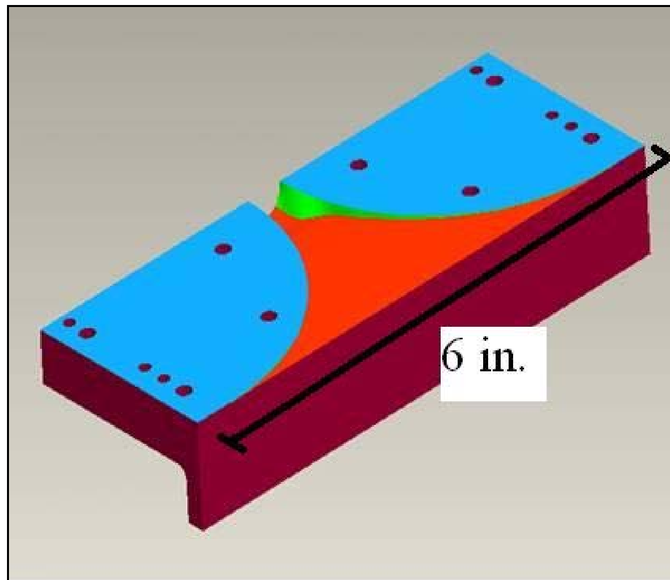

Figure 4. Cross-Section of ProE Model of 6 in Manifold

The manifold inlet is characterized by a circular cross-sectional area with a diameter of one inch, and this corresponds to the outlet geometry of the twin-screw extruder. Consequently, all manifolds utilized throughout the Oriented Nanocomposite Extrusion project have this inlet geometry. The area of the manifold inlet is 0.785 in^2.

Nanocomposite sheets are the product of interest; therefore, the geometry of the manifold outlet is rectangular with a width of 6 in and an area of 0.057 in^2. The cross-sectional area between these two planes is continuously decreasing.

Knowing the inlet and outlet geometries and the manner of the variation of the cross-sectional area between these planes, it was possible to represent the manifold in MATLAB. A MATLAB graph featuring the significant manifold boundaries is shown in Figure 5.

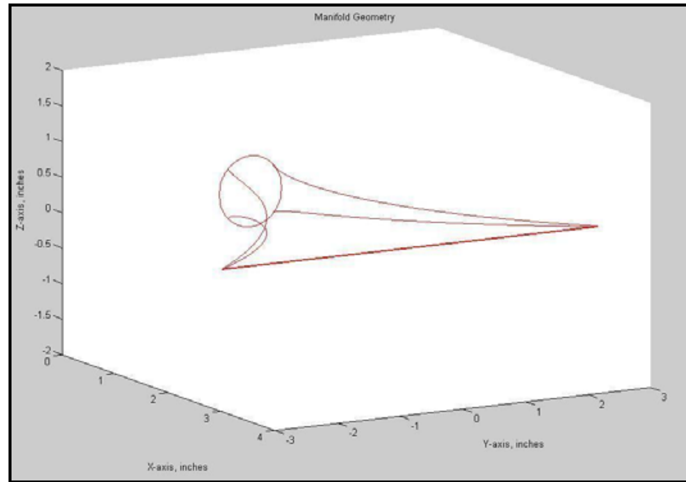

Figure 5. MATLAB Model of Significant Manifold Boundaries

From Figure 5, it is apparent that the manifold contains two symmetry planes. It was anticipated that solution resolution would be limited, and it had been hoped that a solution could be obtained using only one quadrant of the manifold. While the flow through the upper half imposes a gravity-induced load on the flow through the lower half, this force contribution is assumed to be negligible. A representation of a solitary manifold quadrant is contained in Figure 6.

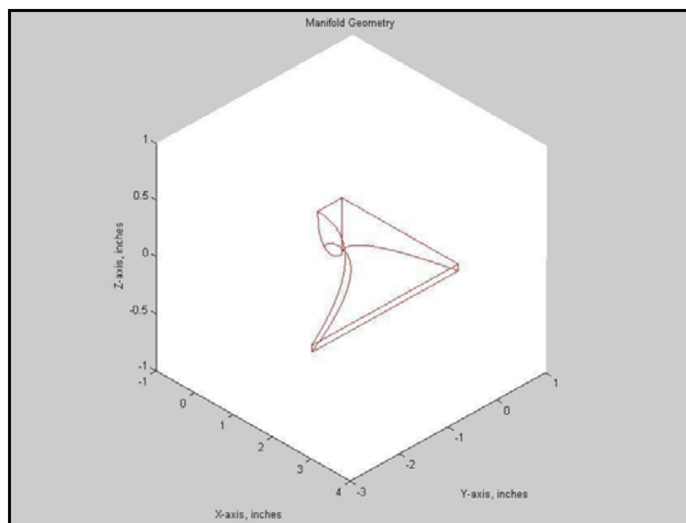

Figure 6. MATLAB Model of a Manifold Quadrant

Modeling only one quadrant of the manifold with the D2Q9 model, the symmetry plane bisecting the width effectively became an opening. As a result, the flow exited the manifold through this plane and produced erroneous results. In the Lattice Boltzmann model used, two walls are required to contain the flow, and only one symmetry plane could be employed. The lower half of the manifold is shown in Figure 7.

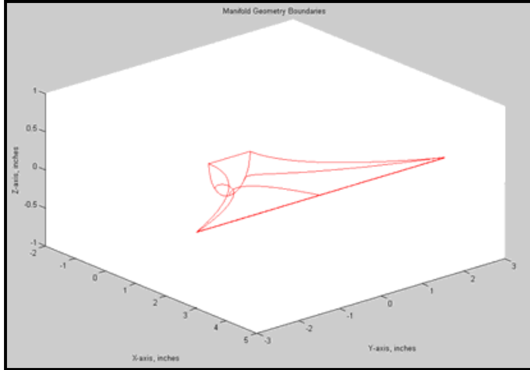

Figure 7. MATLAB Model of the Manifold Lower-Half Boundaries

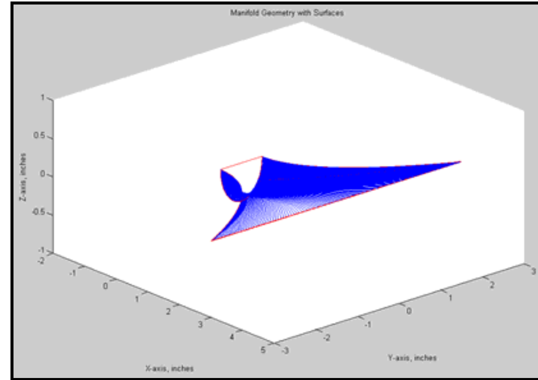

Figure 8. MATLAB Model of the Manifold Lower-Half Surfaces

Having determined the manifold boundaries, arrays of user-specified resolution were swept to map the surfaces. This is shown in Figure 8. The points generated by these sweeping arrays were then used to produce two-dimensional planes extending from the inlet to the outlet. Given that a D2Q9 Lattice Boltzmann model was to be utilized in this simulation, it was only possible to model planes with both an inlet and outlet. Consequently, only planes near the upper surface of the geometry shown in Figure 8 could be analyzed, and this constitutes a limitation of the model.

It was necessary to convert the plane of interest into matrix format to be analyzed with the Lattice Boltzmann method. Each location in the matrix represents a node, which is initially set to a one (flow) or a zero (no flow). The size of the matrix is directly connected to the accuracy of the result. However, a large matrix will also greatly augment the time required to obtain a solution.

In this simulation, a 60 x 30 matrix was selected due to the relative ease of the unit conversion. The 60 rows correlate to the manifold width of 6 in, and the 30 columns represent the 3 in length. As a result, the effective length units applied in the code are inches*10, and all other variables incorporating length are adjusted accordingly.

The code appropriately places ones and zeros in the solution matrix by treating the defining lines of the 2D planes as boundaries. Iterations effectively begin on one side of the matrix with values set to zero. After crossing the boundary into the flow domain, the values are set to one. Upon reaching the opposing boundary, the values are set to zero until the other edge of the matrix is reached.

Several fluid parameters are needed to model the flow through this matrix. Three of these variables are set by the user when running the program. Upon initiating the program, the operator receives prompts for the polymer selection, the concentration of carbon nanotubes, and the mass flow rate. At present, the polymer options are polystyrene (PS), high-density polyethylene (HDPE), and low-density polyethylene (LDPE) as these are the polymers being investigated with the most frequency. After a polymer is assigned, the code proceeds to a small material library in which the density is designated. These material properties were obtained from www.matweb.com. Once the concentration of carbon nanotubes is established, the mass-weighted density is calculated.

The mass flow rate ($\dot{m}$) input allows for the determination of the outlet velocity by definition:

$$V_{outlet} = \frac{\dot{m}}{\rho \cdot A_{outlet}}. \tag{2}$$

In this expression, V_{outlet} is the velocity at the manifold outlet, ρ is the mass-weighted density, and A_{outlet} is the area of the outlet. The units on the mass flow rate input are set to kg/s as these are the units associated with the actual twin-screw extruder feeders.

Assuming an incompressible flow, the density will remain unaltered throughout the manifold. As a result, the density and area at the manifold inlet are known, and the velocity at the inlet is determined with the continuity equation. At the outlet, the nanocomposite melt flow is effectively a jet. Thus, assuming atmospheric pressure at the manifold outlet, the static pressure can be determined at the inlet through Bernoulli's Equation,

$$p_{inlet} = p_{outlet} + \frac{1}{2} \cdot \rho \cdot \left(V_{outlet}^2 - V_{inlet}^2\right). \tag{3}$$

These values are then used to determine the pressure gradient inducing the flow,

$$\frac{dp}{dx} = \frac{\Delta p}{dL} = \frac{p_{inlet} - p_{outlet}}{L} . \tag{4}$$

Equation 4 is the formula for the pressure gradient associated with this model, in which L represents the manifold length. This assumes that the pressure gradient is constant, and as the cross-sectional area is continuously decreasing, this is a decent assumption.

Another important parameter is the dynamic viscosity of the nanocomposite melt. This variable is estimated as 600 Pa*s [3]. However, in reality, viscosity in polymer melts is strongly dependent on temperature, and the temperature in the twin-screw extruder is routinely varied. There are several NASA employees providing material characterization support, and their impending results could be used to formulate an improved viscosity model in the program.

Values in the code directly associated with the general D2Q9 model are listed in Table 1 in Appendix B. Brief descriptions of their significance are also included in the table.

The portion of the program in which the iterations were run was based on a MathWorks Lattice Boltzmann program modeling the flow of water between parallel plates [4]. The solution matrix of model consisted of 40 rows, 20 columns, and the resulting 800 nodes. An arbitrary pressure gradient forces water through the channel, and the output is a velocity profile which is compared to a theoretical profile.

This model was modified to accommodate the flow through the manifold. The pressure gradient steadily alters the flow by means outlined in Appendix A, and the velocity profile of the base model is converted to a velocity array. The simulation ceases when convergence is reached, the maximum number of iterations is obtained, or the values become unrealistic.

A convergence limit of 10^{-8} is specified in the code. At each time step, the average velocity is calculated and compared to the value determined by the previous time step. Unity is subtracted from the respective ratio, and the absolute value of this quantity is designed to steadily approach zero. When this value becomes less than 10^{-8}, the iterations stop.

The iterations are also discontinued when a predetermined number of iterations is reached. The default setting in the code is 3000 iterations, though both the convergence limit and the maximum number of iterations are defined at the beginning of the code to facilitate adjustments.

Finally, the solution concludes if the absolute value of the average velocity exceeds 100 in/s. A melt flow velocity of this magnitude is unattainable in the twin-screw extruder. Should the simulation arrive at an unrealistic velocity value, it is most likely the result of an unrealistic user-input mass flow rate.

Upon iteration termination, the simulation compiles a velocity array, and selects the maximum and minimum velocity values. These values are related to pressure through the Navier-Stokes Equation,

$$\rho\left(\frac{\partial V}{\partial t}+\bar{V}\cdot\nabla\bar{V}\right)=-\nabla p+\rho\cdot\bar{g}+\mu\nabla^{2}\bar{V}\,. \tag{5}$$

Upon completion of this calculation, the maximum pressure is determined, and this value is output to the screen.

5. Results and Discussion

The complete and commented code used to determine the maximum pressure value is included in Appendix C. The program was designed to accept any combination of variable values; thus, the potential quantity of results is considerable.

In addition to detailing the maximum pressure throughout the manifold, plots of the solution matrix geometry, fluid obstacles, and the medial axis are generated with each simulation. The solution matrix geometry is shown in Figure 9. This is a plot of the solution matrix with the flow area (ones) set to white and the area outside of the flow set to black (zeros).

Solution Matrix Geometry

Figure 9. MATLAB Output of Solution Matrix Geometry

Figure 10 is an image of the solution matrix in its numerical form.

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0
0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0
0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0
```

Figure 10. Solution Matrix in Numerical Form

Additionally, the code outputs images of manifold walls serving as the fluid obstacles and the medial axis associated with the solution matrix. These graphics are included in Figures 11 and 12, respectively.

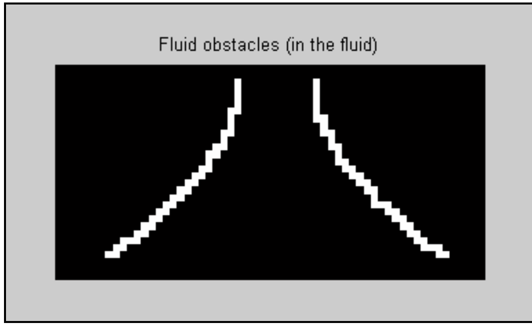

Figure 11. Fluid Obstacles Output

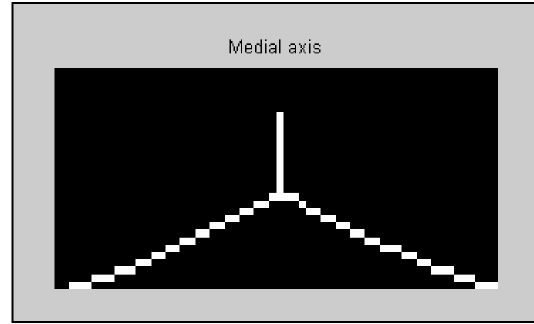

Figure 12. Matrix Medial Axis Output

Furthermore, a plot of the convergence path is generated. Assuming reasonable input variables, the convergence variable will be sizeable through the early iterations and quickly approach zero. An image of the convergence plot is featured in Figure 13.

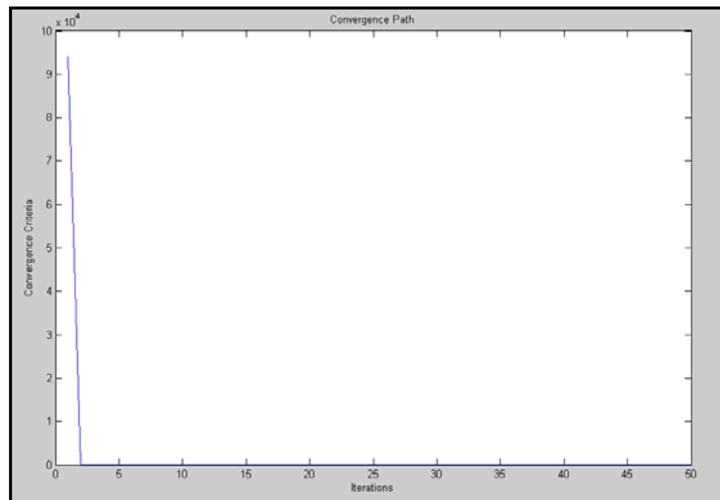

Figure 13. Sample Convergence Plot

Figure 9 shows the flow boundaries to be jagged, and this is the result of limitations on the size of the solution matrix. The size of the solution matrix is inhibited primarily by computing power. It is expected that in order to obtain acceptably fine boundaries, the size of the solution matrix would have to be increased from 60 x 30 to 6000 x 3000. This would increase the node quantity by a factor of 10,000. An average simulation with the current settings requires approximately two minutes. Assuming that computing time is directly correlated to nodal density, then increasing the nodes to this desired resolution would result in a simulation duration of 333 hours.

In addition to the lack of available computing power, another factor limiting the accuracy of this model is the fixed dynamic viscosity. To improve upon this, a concrete

relationship between the polymer melt viscosity and manifold temperature must be established.

Furthermore, the flow through the physical manifold is a three-dimensional flow, and the developed code provides a two-dimensional analysis. Thorough attempts were made to ascertain a basic D3Q19 model programmed in MATLAB; however, no code could be found. The premise of completely designing and implementing a three-dimensional Lattice Boltzmann model is far beyond any reasonable scope associated with this project.

When beginning this research, Dan Powell, the principal investigator of the Oriented Nanocomposite Extrusion project stated that there was enough work to compose a doctoral thesis. While the validity of this contention was initially unclear, the assertion became increasingly legitimate as work progressed. Major improvements could be made to the model by building three-dimensional geometry, and minor upgrades could be made by accounting for Van der Waals interactions between the carbon nanotubes. The collection of potential enhancements could easily occupy a researcher for years.

4. Conclusions

A D2Q9 Lattice Boltzmann model was programmed in MATLAB and used to ascertain the maximum pressure caused by a nanocomposite melt flow through a manifold attached to a twin-screw extruder. The manifold geometry was constructed in MATLAB and this geometry was used to acquire the solution matrix. Most physical properties associated with the nanocomposite melt flow are specified in the code, though polymer selection, carbon nanotube concentration, and mass flow rate are accepted as user inputs. The Lattice Boltzmann model determines the velocity array of the flow through the extruder and calculates the maximum pressure based on this array.

There are a number of limitations associated with this model. The solution matrix is characterized by a relatively low nodal density due to lack of computing power. Also, the precise dynamic viscosity of the nanocomposite melt is unknown, and the program employs an approximation. Furthermore, the two-dimensional model is utilized to approximate a three-dimensional flow as a result of the project scope.

In spite of these limitations, the code serves as another tool available to NASA and/or the University of Maryland with which the pressure in the manifold can be

predicted. Moreover, the program represents a solid foundation upon which researchers could develop a more robust model in the future.

5. References

1. Korner, Carolin, T. Pohl, U. Rude, N. Thurey, and T. Zeiser. "Parallel Lattice Boltzmann Methods for CFD Applications." Institut fűr Informatik. 27 Feb. 2006 <http://www10.informatik.uni-erlangen.de/Publications/Papers/LBMCFD_LNCSE51.pdf>.

2. Aydilek, Ahmet, and E. Kutay. "The Lattice Boltzmann Method." 3 March 2006 <http://www.latticeboltzmann.com>.

3. Kim, Byung, K. Shon, and H. Jeong. "Modification of Polystyrene by Reactive Extrusion with Peroxide and Trimethylolpropane Triacrylate." 7 Oct. 2003. Journal of Applied Polymer Science. 4 Apr. 2006 <http://home.pusan.ac.kr/~goyeun/pdf/2004-4.pdf>.

4. Schena, Gianni. "Lattice Boltzmann LBE, Geometry." July 2005. The MathWorks. 12 Dec. 2005 <http://www.mathworks.com>.

6. Appendices

Appendix A: Lattice Boltzmann Derivation and Discussion

Mathematical Derivation and Discussion of the Lattice Boltzmann Model

The Lattice-Boltmann model is mathematically derived from the Boltzmann equation, which is a partial differential equation describing the evolution of the single particle distribution function *f* in phase space. This function is defined in such a way that $f(\overline{x},\overline{v},t)$ is the probability of particles to being located within a phase space control element $d\overline{x}d\overline{v}$ about $\overline{x}$ and $\overline{v}$ at time t. In the control element, $\overline{x}$ represents the spatial position vector, and $\overline{v}$ is particle velocity vector.

The transport equation for the $f(\overline{x},\overline{v},t)$ can be expressed by the Boltzmann equation as:

$$\frac{\partial f}{\partial t}+\overline{v}\frac{\partial f}{\partial \overline{x}}+\overline{F}\frac{\partial f}{\partial \overline{v}}=\Omega(f)\,. \tag{6}$$

In Equation 6, *Ω(f)* is the collision term. At low Mach numbers, *Ω(f)* can be simplified to the Bhatnagar-Gross-Krook (BGK) approximation, and this is performed in the code associated with this project. The BGK model (a.k.a. "single relaxation time approximation") is expressed as

$$Q(f)=-\frac{1}{\lambda}(f-f^{(0)})\,. \tag{7}$$

In this formula, λ represents the relaxation time controlling the rate at which equilibrium is reached, and $f^{(0)}$ is Maxwell-Boltzmann equilibrium distribution function. BGK relaxation fulfills conservation of mass and momentum

To obtain the Lattice Boltzmann model, a finite set of velocity vectors $\overline{v_i}$ (i = 0,…, n) is created. The quantity of velocity vectors used depends on the LBM model selected. For example, the D2Q9 model employs nine velocities while the D3Q19 model incorporates 19.

D2Q9 => 9 velocities, i = 0,…,8

D3Q19 => 19 velocities, i = 0,…18

The velocity space is discretized using these vectors to produce the velocity-discrete Boltzmann equation,

$$\frac{\partial f_i}{\partial t}+\overline{v_i}\frac{\partial f_i}{\partial \overline{x}}+\overline{F}\frac{\partial f}{\partial \overline{v_i}}=-\frac{1}{\lambda}\left(f_i-f_i^{eq}\right). \tag{8}$$

$$f_i\left(\bar{x},t\right)=f(\bar{x},\bar{v}_i,t)$$

For all Lattice Boltzmann models, a suitable equilibrium distribution is of the form

$$f_i^{eq}=\rho w_i\left[1+\frac{3}{c^2}\bar{v}_i\cdot\bar{u}+\frac{9}{2c^4}\left(\bar{v}_i\cdot\bar{u}\right)^2-\frac{3}{2c^2}\bar{u}\cdot\bar{u}\right]. \tag{9}$$

In Equation 9, c is $\Delta x/\Delta t$, where Δx is normalized to one in 2D and 3D models. $\bar{u}$ represents macroscopic flow velocity, and w_i is a weighting factor that depends on the particular Lattice Boltzmann model chosen. The weighting factor values are obtained from tables. f_i^{eq} is derived from the Maxwell-Boltzmann distribution function, $f^{(0)}$,such that the velocity moments up to fourth order are identical with those of $f^{(0)}$.

To obtain the main equation of LBM, Equation 8, is discretized numerically in a particular manner. The discretization of space and time is accomplished by an explicit finite difference approximation. By scaling the lattice spacing, the time step, and the discrete velocities appropriately,

$$f_i\left(\bar{x}+\bar{e}_i\Delta t,t+\Delta t\right)-f_i\left(\bar{x},t\right)=-\frac{1}{\tau}\left[f_i\left(\bar{x},t\right)-f_i^{eq}(\bar{x},t)\right]. \tag{10}$$

In Equation 10, $\tau=\lambda/\Delta t$, the dimensionless relaxation time. The right side of Equation 10 represents the "collision step", while the left side is the "streaming step". For the collision step, f_i^{eq} must be calculated at each cell and at each time step using:

$$\rho=\int_{-\infty}^{\infty}fd\bar{v}=\sum_{i=0}^{N}f_i=\sum_{i=0}^{N}f_i^{eq} \tag{11}$$

and

$$\rho\bar{u}=\int_{-\infty}^{\infty}\bar{v}fd\bar{v}=\sum_{i=0}^{N}\bar{v}_i f_i^{eq}\ . \tag{12}$$

ρ and $\rho\bar{u}$ are the macroscopic density and momentum, respectively. Therefore, these equations function as continuity of mass and continuity of momentum.

It can be advantageous to split the update process into two equations:

$$f_i^{out}\left(\bar{x},t\right)=f_i^{in}\left(\bar{x},t\right)-\frac{1}{\tau}\left[f_i^{in}\left(\bar{x},t\right)-f_i^{eq}\left(\bar{x},t\right)\right], \tag{13}$$

$$f_i^{in}\left(\overline{x}+\overline{v_i}\Delta t, t+\Delta t\right)=f_i^{out}\left(\overline{v}, t\right). \quad (14)$$

When applying this methodology, f_i^{out} is the distribution after collision but before propagation, and f_i^{in} is distribution after collision and propagation which simultaneously translates to the values entering the neighboring cell as data for the next time step

The boundary conditions associated with the Lattice Boltzmann model are achieved through what is termed the "bounce-back rule" on wall nodes. Assuming that $\overline{x}$ belongs to the wall:

$$f_i^{out}\left(\overline{x}, t\right)=f_{\bar{i}}^{in}\left(\overline{x}, t\right) \quad (15)$$

$$\overline{v_{\bar{i}}}=-\overline{v_i} \quad (16)$$

$$f_{\bar{i}}\left(\overline{x}, t\right)=f\left(\overline{x}, \overline{v_{\bar{i}}}, t\right)=f\left(\overline{x}, -\overline{v_i}, t\right) \quad (17)$$

Thus, the bounce-back rule rotates the distribution function on a wall node, returning a particle back to the fluid with the opposite momentum in the subsequent time step, producing a wall velocity equal to zero.

Appendix B: Model Parameters Associated with the D2Q9 LBM Model

Table 1. D2Q9 Parameters [1]

Parameter	Significance	D2Q9 Value
c^2	Speed of Sound Squared	1/3
(vi)x (vi)y	x discrete velocity vectors y discrete velocity vectors	(0 1 0 -1 0 1 -1 -1 1) (0 0 1 0 -1 1 1 -1 -1)
wi wi wi wi	weighting factor associated with |vi|^2 = 0 weighting factor associated with |vi|^2 = 1 weighting factor associated with |vi|^2 = 2 weighting factor associated with |vi|^2 = 3	4/9 1/9 1/36 1/72

Appendix C. MATLAB Lattice Boltzmann Code

```
%L-B D2Q9 Model of Nanocomposite Flow through an Extrusion Manifold
%University of Minnesota
%Craig Lewandowski

function nanocomposite()
close all, clear all
tic                         %Start clock
surf_res = .01;             %Declare the surface resolution in the geometry model
Max_Iters = 3000;           %Set the maximum number of iterations
tolerance = 1.0e-8;         %Tolerance to declare convergence

%Acquire User-Specified Data
fprintf('Lattice Boltzmann Model for Maximum Pressure in the TSE\n\n')
fprintf('Input Polymer (HDPE = 1, LDPE = 2, PS = 3): ')
material = input('');
fprintf('Input the CNT Concentration in Percent: ')
conc = input('');
fprintf('Input the flowrate in kg/s: ')
flow_rate_m = input('');            %Inputs the flow rate in metric units (kg/s)
fprintf('\n')
flow_rate = flow_rate_m*.068522;    %Converts flow rate to slugs/s

%Material Library
if material == 1            %1 = HDPE
   rho_poly_lb = 0.0343/1000;    %Density of high-density polyethylene (lb/(10*in^3))
end
if material == 2            %2 = LDPE
   rho_poly_lb = 0.0343/1000;    %Density of low-density polyethylene (lb/(10*in^3))
end
if material == 3            %3 = PS
   rho_poly_lb = .0381/1000;     %Density of polystyrene (lb/(10*in)^3)
end
rho_poly = rho_poly_lb * .031081;   %Convert density to slugs/((10*in)^3)
rho_cnt  = 0.0441/1000 * .031081;   %Carbon nanotube density (slugs/(10*in)^3)
density = (1-(conc/100))*rho_poly + (conc/100)*rho_cnt; %Mass-weighted density
A_1 = ((pi/2)*(1)^2)*100;    %Inlet area in (in*10 * in*10)
A_2 = 5.7/2;                 %Outlet area in (in*10 * in*10)
V_2 = flow_rate/(density * A_2); %Velocity at the manifold outlet

%Manifold Geometry Modeling
figure(2)
micro_m = .039370078740157 * 10^-3;  %One micrometer equivalent in terms of inches - conversion
%Set the curved die-inlet geometry
r =  0.5;   %Inlet is a circle w/ raduis = 0.5 in
y_inlet_c = -r : .01 : r;
x_inlet_c = 0 .* y_inlet_c;
z_inlet_c = - sqrt(r^2 - y_inlet_c.^2);
plot3(x_inlet_c, y_inlet_c, z_inlet_c, 'r-')
axis([-2 5 -3 3 -1 1])
xlabel('X-axis, inches'), ylabel('Y-axis, inches'), zlabel('Z-axis, inches')
title('Manifold Geometry with Surfaces')
%grid on
hold on
```

```
%Set the line above the circular inlet
x_up = 0 : .01 : 3;
y_in_up = -r : .01 : r;
x_in_up = 0 .* y_in_up;
z_in_up = 0 .* y_in_up;
plot3(x_in_up, y_in_up, z_in_up, 'r-')
hold on

%Set the upper surface of the die-outlet geometry
y_outlet_up = -3 : .01 : 3;
x_outlet_up = 0 .* y_outlet_up + 3;   %Assumes 3 inches between the inlet and outlet surfaces
z_outlet_up = 0 .* y_outlet_up;
plot3(x_outlet_up, y_outlet_up, z_outlet_up, 'r-')
hold on

%Set the lower surface of the die-outlet geometry
t = 120 * micro_m;       %Die outlet thickness = 240 micrometers
y_outlet_down = -3 : .01 : 3;
x_outlet_down = 0 .* y_outlet_down + 3;   %Assumes 3 inches between the inlet and outlet surfaces
z_outlet_down = 0 .* y_outlet_down - t;
plot3(x_outlet_down, y_outlet_down, z_outlet_down, 'r-')
hold on

%Set the left surface of the die-outlet geometry
z_outlet_out = -t : .001 : 0;
y_outlet_out = 0 .* z_outlet_out - 3;
x_outlet_out = 0 .* z_outlet_out + 3;
plot3(x_outlet_out, y_outlet_out, z_outlet_out, 'r-')
hold on

%Set the right surface of the die-outlet geometry
z_outlet_in = -t : .001 : 0;
y_outlet_in = 0 .* z_outlet_in + 3;
x_outlet_in = 0 .* z_outlet_in + 3;
plot3(x_outlet_in, y_outlet_in, z_outlet_in, 'r-')
hold on

%BEGIN MORE COMPLICATED GEOMETRY
%Set the line directly below the central line of the die geometry
x_down = x_up;
y_down = 0 .* x_down;
z_down = (-.5 - (-t))/(-sqrt(3)) .* sqrt(x_down) - .5;
plot3(x_down, y_down, z_down, 'r-')
hold on

%Set the left line on the upper surface connecting the inlet and outlet
x_out_surf_up = 0 : .01 : 3;
y_out_surf_up = -.277777778 * (x_out_surf_up .^ 2) - .5;
z_out_surf_up = 0 .* x_out_surf_up;
plot3(x_out_surf_up, y_out_surf_up, z_out_surf_up, 'r-')
hold on

%Set the right line on the upper surface connecting the inlet and outlet
x_out_surf_up_r = 0 : .01 : 3;
y_out_surf_up_r = .277777778 * (x_out_surf_up_r .^ 2) + .5;
```

```
z_out_surf_up_r = 0 .* x_out_surf_up_r;
plot3(x_out_surf_up_r, y_out_surf_up_r, z_out_surf_up_r, 'r-')
hold on

%Set the left line on the lower surface connecting the inlet and outlet
angle_deg = 45;         %Angle for solution purposes set in degrees
angle = angle_deg * pi/180;
x_out_surf_down = 0 : .01 : 3;
y_out_surf_down = ((-cos(angle)*r - (-3))/(-(3^2))) * (x_out_surf_down .^ 2) - cos(angle)*r;
z_out_surf_down = ((-cos(angle)*r - (-t))/(-sqrt(3))) * sqrt(x_out_surf_down) - cos(angle)*r;
plot3(x_out_surf_down, y_out_surf_down, z_out_surf_down, 'r-')
hold on

%Set the right line on the lower surface connecting the inlet and outlet
angle_deg = 45;         %Angle for solution purposes set in degrees
angle = angle_deg * pi/180;
x_out_surf_down_r = 0 : .01 : 3;
y_out_surf_down_r = ((cos(angle)*r - 3)/((-3^2))) * (x_out_surf_down_r .^ 2) + cos(angle)*r;
z_out_surf_down_r = -((cos(angle)*r - (-t))/(-sqrt(3))) * sqrt(x_out_surf_down_r) - cos(angle)*r;
plot3(x_out_surf_down_r, y_out_surf_down_r, z_out_surf_down_r, 'r-')
hold on

%NOTE: IT IS THESE THREE LINES COMBINED WITH THE CENTRAL LINE THAT DEFINE
%THE BOUNDARIES THAT WILL BE NECESSARY FOR THE SOLUTION
%Rewriting the central line expressions:
%x_up = 0 : .01 : 3;        y_up = 0 .* x_up;
%z_up = 0 .* x_up;          plot3(x_up, y_up, z_up, 'r-')

theta_deg = -90;
theta = theta_deg * pi/180;
theta_deg_s = theta_deg + 45;
theta_s = theta_deg_s * pi/180;
incr = 1;                       %Angle increment in degrees
y_outlet = 0;                   %Initial position at outlet
n_iter = 45 / incr;             %number of iterations to 45 degrees
p_incr_b = 3 / n_iter;          %increment of bottom line on outlet in inches
count = 0;                      %count the number of iterations

%Creating the lower surface of the die geometry
while theta_deg <= -45
y_inlet = -cos(theta) * r;
x_inlet = 0;                    %x position at the inlet in inches
x_outlet = 3;                   %x position at the outlet in inches
z_outlet = -t;                  %z position at the outlet in inches
z_inlet = sin(theta) * r;       %z position at the inlet in inches

c = (y_inlet - y_outlet)/(x_inlet^2 - (x_outlet^2));  %x-y slope factor for quadratic
b = y_outlet - c * x_outlet^2;    %x-y y-intercept
d = (z_inlet - z_outlet)/(x_inlet^.5 - x_outlet^.5);  %x-z slope factor for quadratic
e = z_outlet - d * x_outlet^.5;    %x-z z-intercept

x_surf = 0: surf_res : 3;
y_surf = c .* x_surf .^ 2 + b;
z_surf = d .* (((y_surf - b) ./ c) .^ .5).^ .5 + e;
plot3(x_surf, y_surf, z_surf, '-')
```

```
xlabel('X-axis, inches'), ylabel('Y-axis, inches'), zlabel('Z-axis, inches')
hold on

z_outlet_s = -t;
p_incr_s  = t / n_iter;
%Creating the side surface of the die geometry
y_outlet_s = -3;                    %iterate y along the base of the outlet
y_inlet_s = -cos(theta_s) * r;
x_inlet_s = 0;                      %x position at the inlet in inches
x_outlet_s = 3;                     %x position at the outlet in inches
z_inlet_s = sin(theta_s) * r;       %z position at the inlet in inches

c_s = (y_inlet_s - y_outlet_s)/(x_inlet_s^2 - (x_outlet_s^2));  %x-y slope factor for quadratic
b_s = y_outlet_s - c_s * x_outlet_s^2;    %x-y y-intercept
d_s = (z_inlet_s - z_outlet_s)/(x_inlet_s^.5 - x_outlet_s^.5);  %x-z slope factor for quadratic
e_s = z_outlet_s - d_s * x_outlet_s^.5;    %x-z z-intercept

x_surf = 0: surf_res : 3;
y_surf = c_s .* x_surf.^ 2 + b_s;
z_surf = d_s .* (((y_surf - b_s) ./ c_s) .^ .5).^ .5 + e_s;
plot3(x_surf, y_surf, z_surf, '-')
xlabel('X-axis, inches'), ylabel('Y-axis, inches'), zlabel('Z-axis, inches')
hold on

%Base Iterations
y_outlet = y_outlet - p_incr_b;   %iterate y along the base of the outlet
theta_deg = theta_deg + incr;
theta = theta_deg * pi/180;
count = count + 1;

%Side Iterations
theta_deg_s = theta_deg_s + incr;
theta_s = theta_deg_s * pi/180;
end

%Build the right side of the manifold
theta_deg = -90;
theta = theta_deg * pi/180;
theta_deg_s = theta_deg - 45;
theta_s = theta_deg_s * pi/180;
incr = 1;                           %Angle increment in degrees
y_outlet = 0;                       %Initial position at outlet
p_incr_b = 3 / n_iter;              %increment of bottom line on outlet in inches
count = 0;                          %count the number of iterations

%Creating the lower surface of the die geometry
while theta_deg >= -135
y_inlet = -cos(theta) * r;
x_inlet = 0;                        %x position at the inlet in inches
x_outlet = 3;                       %x position at the outlet in inches
z_outlet = -t;                      %z position at the outlet in inches
z_inlet = sin(theta) * r;           %z position at the inlet in inches

c = (y_inlet - y_outlet)/(x_inlet^2 - (x_outlet^2));  %x-y slope factor for quadratic
b = y_outlet - c * x_outlet^2;    %x-y y-intercept
```

```
d = (z_inlet - z_outlet)/(x_inlet^.5 - x_outlet^.5);  %x-z slope factor for quadratic
e = z_outlet - d * x_outlet^.5;    %x-z z-intercept

x_surf = 0: surf_res : 3;
y_surf = c .* x_surf .^ 2 + b;
z_surf = d .* (((y_surf - b) ./ c) .^ .5).^ .5 + e;
plot3(x_surf, y_surf, z_surf, '-')
hold on

z_outlet_s = -t;
p_incr_s  = t / n_iter;
%Creating the side surface of the die geometry
y_outlet_s = 3;                    %iterate y along the base of the outlet
y_inlet_s = -cos(theta_s) * r;
x_inlet_s = 0;                     %x position at the inlet in inches
x_outlet_s = 3;                     %x position at the outlet in inches
z_inlet_s = sin(theta_s) * r;       %z position at the inlet in inches

c_s = (y_inlet_s - y_outlet_s)/(x_inlet_s^2 - (x_outlet_s^2));  %x-y slope factor for quadratic
b_s = y_outlet_s - c_s * x_outlet_s^2;    %x-y y-intercept
d_s = (z_inlet_s - z_outlet_s)/(x_inlet_s^.5 - x_outlet_s^.5);  %x-z slope factor for quadratic
e_s = z_outlet_s - d_s * x_outlet_s^.5;    %x-z z-intercept

x_surf = 0: surf_res : 3;
y_surf = c_s .* x_surf.^ 2 + b_s;
z_surf = d_s .* (((y_surf - b_s) ./ c_s) .^ .5).^ .5 + e_s;
plot3(x_surf, y_surf, z_surf, '-')
xlabel('X-axis, inches'), ylabel('Y-axis, inches'), zlabel('Z-axis, inches')
hold on

%Base Iterations
y_outlet = y_outlet + p_incr_b;   %iterate y along the base of the outlet
theta_deg = theta_deg - incr;
theta = theta_deg * pi/180;
count = count + 1;

%Side Iterations
theta_deg_s = theta_deg_s - incr;
theta_s = theta_deg_s * pi/180;
end
hold off

%Rebuilding the die boundaries and plotting seperately
figure(1)
plot3(x_inlet_c, y_inlet_c, z_inlet_c, 'r-')
axis([-2 5 -3 3 -1 1])
xlabel('X-axis, inches'), ylabel('Y-axis, inches'), zlabel('Z-axis, inches')
title('Manifold Geometry Boundaries')
%grid on
hold on
plot3(x_in_up, y_in_up, z_in_up, 'r-')
hold on
plot3(x_outlet_up, y_outlet_up, z_outlet_up, 'r-')
hold on
plot3(x_outlet_down, y_outlet_down, z_outlet_down, 'r-')
```

```
hold on
plot3(x_outlet_down, y_outlet_down, z_outlet_down, 'r-')
hold on
plot3(x_outlet_out, y_outlet_out, z_outlet_out, 'r-')
hold on
plot3(x_outlet_in, y_outlet_in, z_outlet_in, 'r-')
hold on
plot3(x_down, y_down, z_down, 'r-')
hold on
plot3(x_out_surf_up, y_out_surf_up, z_out_surf_up, 'r-')
hold on
plot3(x_out_surf_up_r, y_out_surf_up_r, z_out_surf_up_r, 'r-')
hold on
plot3(x_out_surf_down, y_out_surf_down, z_out_surf_down, 'r-')
hold on
plot3(x_out_surf_down_r, y_out_surf_down_r, z_out_surf_down_r, 'r-')
hold off

%Create matrix plane for the left side of the manifold
%Adjusting manifold geometry for matrix solution
y_outlet_s = -30;                    %iterate y along the base of the outlet
y_inlet_s = -10 * r;
x_inlet_s = 0;                       %x position at the inlet in inches
x_outlet_s = 3*10;
c_s = (y_inlet_s - y_outlet_s)/(x_inlet_s^2 - (x_outlet_s^2));  %x-y slope factor for quadratic
b_s = y_outlet_s - c_s * x_outlet_s^2;    %x-y y-intercept
x = 1;                               %Declare x-iteration variable
y2 = -29;                            %Set y-iteration variable to outer edge of manifold
matrix_l = [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1]; %Manually set 1st row of matrix plane
matrixa = zeros([1,1]);         %Declare iteration matrix
while x < 30
    y = c_s * x ^ 2 + b_s;       %y parameter based on matrix geometry
    while y2 < 0                 %Traversing the matrix plane...
        if y2 < y                %If the iteration varible doesn't cross into the manifold
            point = 0;           %Set location to zero
        else
            point = 1;           %When the variable crosses, set location to 1
        end
        matrixa = [matrixa, point]; %Add point to iteration matrix
        y2 = y2 + 1;                %Iterate y
    end
    matrix_l = [matrix_l; matrixa];     %Add iteration matrix to matrix plane
    matrixa = zeros([1,1]);         %Reset iteration matrix
        x=x+1;                      %Iterate x (progressing towards outlet)
        y2=-29;                     %Reset y
end
%Create matrix plane for the right side of the manifold
y_outlet_s = 30;                     %iterate y along the base of the outlet
y_inlet_s = 10 * r;
x_inlet_s = 0;                       %x position at the inlet in inches
x_outlet_s = 3*10;
c_s = (y_inlet_s - y_outlet_s)/(x_inlet_s^2 - (x_outlet_s^2));  %x-y slope factor for quadratic
b_s = y_outlet_s - c_s * x_outlet_s^2;    %x-y y-intercept
x = 1;                               %Declare x-iteration variable
y2 = 0;                              %Set y-iteration variable to outer edge of manifold
```

```
matrix_r = [1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]; %Manually set 1st row of matrix plane
matrixa = ones([1,1]);         %Declare iteration matrix
while x < 30
    y = c_s * x ^ 2 + b_s;       %y parameter based on matrix geometry
    while y2 < 29                 %Traversing the matrix plane...
        if y2 < y               %If the iteration varible doesn't cross into the manifold
            point = 1;          %Set location to one
        else
            point = 0;          %When the variable crosses, set location to zero
        end
        matrixa = [matrixa, point]; %Add point to iteration matrix
        y2 = y2 + 1;                %Iterate y
        end
    matrix_r = [matrix_r; matrixa];     %Add iteration matrix to matrix plane
    matrixa = ones([1,1]);         %Reset iteration matrix
        x=x+1;                     %Iterate x (progressing towards outlet)
        y2=0;                    %Reset y
end
matrix = [matrix_l, matrix_r];
%matrix                          %Option of displaying solution matrix plane
Channel2D = matrix;

L = 30;                     %Channel length in inches*10
p2 = 14.7/100;              %Pressure at the manifold outlet in lbf/((10*in)^2) (atmospheric)
V_1 = (A_2*V_2)/(A_1);      %Continuity Eqn.
p1 = p2 + .5 * density *(V_2^2 - V_1^2); %Bernoulli's Eqn.
dPdL = -(p1 - p2)/L;         %Pressure Gradient - Drives Flow
Len_Channel_2D=30;               %Die length (x-direction) in inches*10
Channel_2D_half_Width=30;        %Die exit half-width in inches*10
Width=Channel_2D_half_Width*2;   %Channel width in inches*10
[Nr Mc] = size(Channel2D);       %Number of rows and columns in solution space
porosity=nnz(Channel2D==1)/(Nr*Mc); %Define porosity
cs2=1/3;                         %Speed of light in LB Model
cP_visco=12.5/10;               %Estimated dynamic viscosity of a polymer melt at 662F/350C in slugs/(10*in*s)
Lky_visco=cP_visco/density;    %Lattice Viscosity
omega=(Lky_visco/cs2+0.5).^-1; %Relaxation Time
uy_fin_max=dPdL*(Channel_2D_half_Width.^2)/(2*Lky_visco);
uy0=-V_1;                       %Initial y-velocity
ux0=.0001;                      %Initial x-velocity
uyf_av=uy_fin_max*(2/3);       %Avg. y-velocity
x_profile=([-Channel_2D_half_Width:+Channel_2D_half_Width-1]+0.5);
uy_analy_profile=uy_fin_max .* (1-( x_profile/Channel_2D_half_Width).^2);
av_vel_t=1.e+10;              %Intialization
inb=2; oub=2;                  %Inlet and Outlet Buffers
inlet=ones(inb,Mc);
outlet=ones(oub,Mc);
wb=2;                         %Wall Buffer
[Nr Mc] = size(Channel2D);     %Update size

%Display the Solution Matrix
figure(3)
imshow(Channel2D,[])        %Display the 2D geometry plane to be analyzed
title('Solution Matrix Geometry')

inb = 2; oub = 2;
```

```
Channel2D=logical(Channel2D);   %Convert matrix to logical matrix
%Obstacles for bounce back (in front of the grain)
Obstacles=bwperim(Channel2D,8); %Perimeter of grains for bounce back boundary condition
border=logical(ones(Nr,Mc));
border([1:inb,Nr-oub:Nr],[wb-1:Mc])=0;
Obstacles=Obstacles.*(border);
figure(4)
imshow(Obstacles); title('Fluid obstacles (in the fluid)' );

Medial_axis=bwmorph(Channel2D,'thin',Inf);  %Define medial axis
figure(5)
imshow(Medial_axis); title('Medial axis');  %Plot medial axis
[iabw1 jabw1]=find(Channel2D==1);           %Indices i,j, of active lattice locations
lena=length(iabw1);                          %Number of active locations (cells)
ija= (jabw1-1)*Nr+iabw1;                     %Equivalent single index (i,j) ->> ija for active locations
[iobs jobs]=find(Obstacles);                 %Same as above
lenobs=length(iobs);
ijobs= (jobs-1)*Nr+iobs;
% Medial axis of the pore space
[ima jma] = find(Medial_axis);               %Same as above
lenma=length(ima);
ijma= (jma-1)*Nr+ima;
%Internal wet locations: wet & not obstables
[iawint jawint] = find((Channel2D==1 & ~Obstacles));
lenwint=length(iawint);                      %Number of internal wet locations
ijaint= (jawint-1)*Nr+iawint;
NxM=Nr*Mc;

%Coordinate System Orientation: E N W S NE NW SW SE ZERO
%zero => rest particle (RP)
%    y^
%  6 2 5           ^         NW  N  NE
%  3 9 1 ... +x-> +y         W   RP  E
%  7 4 8                    SW  S  SE
%   -y
% x & y components of velocities , +x is to est , +y is to nord
East=1; North=2; West=3; South=4; NE=5; NW=6; SW=7; SE=8; RP=9;
N_c=9;          %Number of Directions
C_x=[1 0 -1  0 1 -1 -1  1 0];              %x parameters of D2Q9 Model
C_y=[0 1  0 -1 1  1 -1 -1 0];              %y parameters of DeQ9 Model
C=[C_x; C_y];                               %Combining terms in a single matrix

%Bounce back technique
%After collision the fluid elements densities f are sent back to the
%lattice node they come from with opposite direction.
%Indices opposite to 1:8 for fast inversion after bounce
ic_op = [3 4 1 2 7 8 5 6];

%Periodic boundary conditions - reinjection technique
yi2=[Nr , 1:Nr , 1];                %Allows for implementation of periodic bound. cond.
w0=16/36. ; w1=4/36. ; w2=1/36.;    %Directional weights (density weights)
W=[w1 w1 w1 w1 w2 w2 w2 w2 w0];
cs2=1/3; cs2x2=2*cs2; cs4x2=2*cs2.^2;   %LBM constants related to speed of sound
%f1=1/cs2; f2=1/cs2x2; f3=1/cs4x2;
f1=3; f2=4.5; f3=1.5;
```

```
%Variable Assignments
f=zeros(Nr, Mc, N_c);   %Array of fluid density distribution-Creates 9 [30, 30] matrices
feq=zeros(Nr,Mc,N_c);   %Array f at equilibrium
rho=ones(Nr,Mc);        %Macroscopic density
temp1=zeros(Nr,Mc);
ux=zeros(Nr,Mc);   uy=zeros(Nr,Mc); uyout=zeros(Nr,Mc);  %For dimensionless velocities
uxsq=zeros(Nr,Mc); uysq=zeros(Nr,Mc);   usq=zeros(Nr,Mc);   %For higher-order velocities
%Initialization arrays: start values in the wet area
for ia=1:lena %Start values in the active cells only - 0 outside
   i=iabw1(ia);
   j=jabw1(ia);
   f(i,j,:)=1/9; %Initial Density Set as Uniform
end
uy(ija)=uy0;        %Initialize fluid velocities
ux(ija)=ux0;
rho(ija)=density;   %Initialize density
force = -dPdL*(1/6)*1*[0 -1 0 1 -1 -1 1  1  0]';    %External force based on pressure gradient
StopFlag=false;         %StopFlag set to logical zero
Max_Iter=Max_Iters;     %Recalling limit on number of iterations
Check_Iter=1;           %Frequency of iteration check
Output_Every=20;
Cur_Iter=0;             %Initialize iteration counter
toler=tolerance;        %Recalling tolerance
Cond_path=[];           %Records values of convergence
density_path=[];        %Recording average density values during convergence

while(~StopFlag)        %While StopFlag remains logical false...
  Cur_Iter=Cur_Iter + 1;  %Iterate counter
  rho=sum(f,3);           %Density modifications
  if Cur_Iter > 1         %Beyond the initial iteration
     ux=zeros(Nr,Mc);    %
     uy=zeros(Nr,Mc);
     for ic=1 : N_c-1;
        ux = ux + C_x(ic).*f(:,:,ic);
        uy = uy + C_y(ic).*f(:,:,ic);
     end
  end
  ux(ija)=ux(ija)./rho(ija); uy(ija)=uy(ija)./rho(ija);
  uxsq(ija)=ux(ija).^2; uysq(ija)=uy(ija).^2;
  usq(ija)=uxsq(ija)+uysq(ija);
  rt0 = w0.*rho; rt1 = w1.*rho; rt2 = w2.*rho;     %Weighted densities: rest particle, principal axis, diagonals
%Equilibrium Distribution
%Main Directions (N, S, E, W)
   feq(ija)= rt1(ija) .*(1 +f1*ux(ija) +f2*uxsq(ija) -f3*usq(ija));
   feq(ija+NxM*(2-1))= rt1(ija) .*(1 +f1*uy(ija) +f2*uysq(ija) -f3*usq(ija));
   feq(ija+NxM*(3-1))= rt1(ija) .*(1 -f1*ux(ija) +f2*uxsq(ija) -f3*usq(ija));
   feq(ija+NxM*(4-1))= rt1(ija) .*(1 -f1*uy(ija) +f2*uysq(ija) -f3*usq(ija));
%Diagonal Motion
   feq(ija+NxM*(5-1))= rt2(ija) .*(1 +f1*(+ux(ija)+uy(ija)) +f2*(+ux(ija)+uy(ija)).^2 -f3.*usq(ija));
   feq(ija+NxM*(6-1))= rt2(ija) .*(1 +f1*(-ux(ija)+uy(ija)) +f2*(-ux(ija)+uy(ija)).^2 -f3.*usq(ija));
   feq(ija+NxM*(7-1))= rt2(ija) .*(1 +f1*(-ux(ija)-uy(ija)) +f2*(-ux(ija)-uy(ija)).^2 -f3.*usq(ija));
   feq(ija+NxM*(8-1))= rt2(ija) .*(1 +f1*(+ux(ija)-uy(ija)) +f2*(+ux(ija)-uy(ija)).^2 -f3.*usq(ija));
%Rest particle ic = 9 (No motion)
   feq(ija+NxM*(9-1))= rt0(ija) .*(1 - f3*usq(ija));
%Collision term
```

```
  f=(1.-omega).*f + omega.*feq;
%External body force due to pressure gradient
  for ic = 1 : N_c
    for ia=1:lena
      i=iabw1(ia);
      j=jabw1(ia);
      f(i,j,ic)= f(i,j,ic) + force(ic);
    end
  end
%Forward propagation step and bounce back
  feq = f;
   for ic = 1 : 1 : N_c-1    %Selects the velocity layer
    ic2=ic_op(ic);          %Selects the layer of the velocity opposite to ic for BB
    temp1=feq(:,:,ic);
    %Wet locations that are NOT on the border to other wet locations
    for ia=1:1:lenwint  %Number of internal (i.e. not border) wet locations
      i=iawint(ia);   %Treat only the wet space
      j=jawint(ia);
      i2 = i+C_y(ic); % Expected final locations to move
      j2 = (j-1)+C_x(ic);
      i2=yi2(i2+1);   %i2 is corrected for PBC when necessary (flow out re-fed to inlet)
      f(i2,j2,ic)=temp1(i,j);   %Generates a circular shift
    end
   %Wet locations on borders
     for ia=1:1:lenobs      % wet border locations
      i=iobs(ia);  j=jobs(ia);  %Treat only the wet space
      i2 = i + C_y(ic);       % Expected final locations to move
      j2 = j + C_x(ic)-1;
      i2 = yi2(i2+1);         %i2 corrected for PBC
      if Channel2D(i2,j2-1) == 0   %New position (i2,j2) is dry
        f(i,j,ic2) = temp1(i,j); %Direction inversion: bounce-back in the opposite direction ic2
      else                        %Otherwise, normal propagation from (i,j) to (i2,j2) on layer ic
        f(i2,j2,ic) = temp1(i,j);
      end
     end
   end  % ends of forward fropagation step & bounce back sections
 %Recalculate  uy as uyout to test convergence
  rho=sum(f,3);
  uyout= zeros(Nr,Mc);
  for ic = 1 : N_c-1;
    f(:,:,ic);
    uyout= uyout + C_y(ic) .* f(:,:,ic);  %Dimensionless flow velocity out
  end
 if mod(Cur_Iter,Check_Iter) == 0
    vect=rho(ija);
    vect=vect(:);
    cur_density=mean(vect);
    vect=uy(ija); av_vel_int= mean(vect);   %Seepage velocity (in the wet area)
    av_vel_int=av_vel_int*porosity;         %Avg. vel. on the wet & dry area
    av_vel_tp1 = av_vel_int;
    Condition=abs( abs(av_vel_t/av_vel_tp1 )-1);    %Should approach 0
    Cond_path=[Cond_path, Condition];        %Records the convergence path (value)
    density_path=[density_path, cur_density];
    av_vel_t=av_vel_tp1;
    %Stop iterating if the tolerance is achieved, if the maximum number
```

```
        %of iterations is achieved, or if the absolute velocity exeedes 100
        %in/s (unrealistic)
        if (Condition < toler) | (Cur_Iter > Max_Iter) | (abs(av_vel_int) > 10)
            StopFlag=true;
            display( 'Iterations Stopped')
            display('Convergence Criteria Met, Max Iterations Reached, or Unrealistic Solution' )
            display( ['Current iteration: ',num2str(Cur_Iter),...
                ' Max Number of iter: ',num2str(Max_Iter)] )
            break           %Terminates the while loop
        end
    end
     if (mod(Cur_Iter,Output_Every)==0)
        rho=sum(f,3);
        %figure(7); imshow(rho,[0.1 0.9]); title(' rho');   %Optional density plot
        up=2;
        V_max = sqrt((max(abs(uy(Nr-up,:))))^2+ (max(abs(ux(Nr-up,:))))^2);  %Retrieve the maximum velocity from the velocity array
        V_min = sqrt((min(abs(uy(Nr-up,:))))^2+ (min(abs(ux(Nr-up,:))))^2);  %Retrieve the maximum velocity from the velocity array
        pause(2);
     end
end %Conclusion of the main iteration loop
figure(6)
plot(Cond_path(2:end));     %Plot the convergence path
title('Convergence Path')
xlabel('Iterations')
ylabel('Convergence Criteria')
p_conv_max = 100*(p1*L*(density*38640 + cP_visco*((V_max-V_1)/L)^2 - density*V_max*(V_max-V_1)/L)); %Navier Stokes
p_conv_min = 100*(p1*L*(density*38640 + cP_visco*((V_min-V_1)/L)^2 - density*V_min*(V_min-V_1)/L)); %Navier-Stokes

if p_conv_max > p_conv_min          %Determine if v_max or v_min has a stronger impact on the pressure
    p = p_conv_max * 100;           %Convert pressure to normal psi
else
    p = pconv_min * 100;
end
    fprintf('\nThe maximum pressure in psi is: ')
disp(p)             %Display the maximum pressure
toc                 %Conclude timer
```