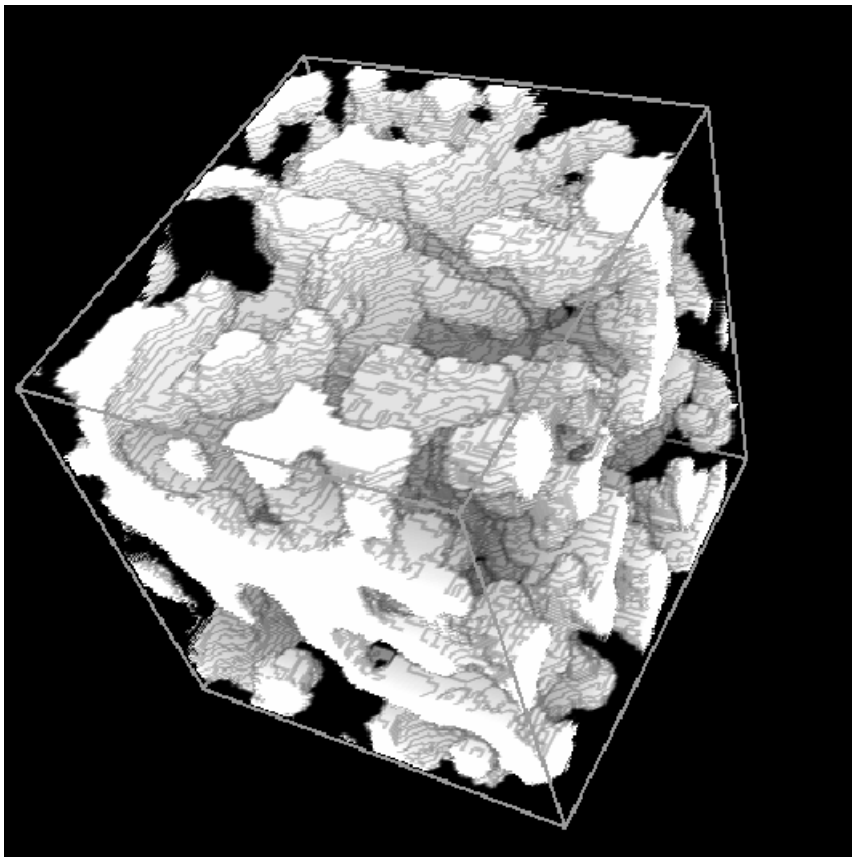
NISTIR 7416

A Stokes Permeability Solver for Three-Dimensional Porous Media



Dale P. Bentz
Nicos S. Martys

National Institute of Standards and Technology
Technology Administration, U.S. Department of Commerce

NISTIR 7416

A Stokes Permeability Solver for Three-Dimensional Porous Media

Dale P. Bentz
Nicos S. Martys
Materials and Construction Research Division
Building and Fire Research Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8615

April 2007

U.S. DEPARTMENT OF COMMERCE
Carlos M. Gutierrez, Secretary
TECHNOLOGY ADMINISTRATION
Robert C. Cresanti, Under Secretary of Commerce for Technology
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY
William Jeffrey, Director

Abstract

A set of C and Fortran computer programs have been developed for computing the permeability of three-dimensional porous media under incompressible Stokes flow conditions. In this NISTIR, the background for developing these codes is reviewed and the detailed code listings are also presented in an appendix. The codes are also available for free downloading from the NIST anonymous ftp site at ftp://ftp.nist.gov/pub/bfrl/bentz/permsolver. While currently being employed to simulate the permeability of model pervious concretes, the codes have been utilized in the past on a wide variety of model and real (computer tomography) three-dimensional microstructures.

Table of Contents

List of Figures

List of Tables

1 Introduction

For many years, research in the Materials and Construction Research Division (formerly the Building Materials Division) at the National Institute of Standards and Technology (NIST) has focused on understanding the relationship between microstructure and properties, particularly for cement-based materials [1]. For many materials, one key material property is their resistance to fluid flow under a pressure gradient, e.g., their permeability. For the case of laminar (slow, incompressible) flow, the fluid flow can be conveniently described by the linear Stokes equations [2]. This paper presents a set of computer codes written in the C and Fortran programming languages to accomplish two purposes: 1) a pre-processor written in C to convert a model or real three-dimensional microstructure (represented by a three-dimensional set of voxels) into lists of coordinates representing the voxels where pressures, x-velocity, y-velocity, or z-velocity components will be present and 2) a set of Fortran programs to numerically solve the linear Stokes equations in three dimensions, using a finite difference scheme in conjunction with the artificial compressibility relaxation algorithm [2, 3].

The remainder of this NISTIR is organized as follows. Section 2 provides a brief computational background to the solution of the linear Stokes equations. Section 3 provides an overview of the two sets of computer codes and their operation. Section 4 presents some validation results obtained for cylindrical and square tubes and some model results recently obtained for pervious concretes. Section 5 provides a summary and the complete code listings are given in an Appendix. The permeability codes have been used in a wide variety of applications at NIST during the past 15 years [2, 4-9]. With the publication of this NISTIR, the codes are being made available to the general public for their utilization; the computer programs may be downloaded from the NIST anonymous ftp site at ftp://ftp.nist.gov/pub/bfrl/bentz/permsolver.

DISCLAIMER

The U.S. Department of Commerce makes no warranty, expressed or implied, to users of the three-dimensional Stokes solver and its associated computer programs, and accepts no responsibility for its use. Users of these computer programs assume sole responsibility under Federal law for determining the appropriateness of its use in any particular application; for any conclusions drawn from the results of its use; and for any actions taken or not taken as a result of analyses performed using these tools.

Users are warned that these codes are intended for use only by those competent in the field of computational materials science and are intended only to supplement the informed judgment of the qualified user. The software package may or may not have predictive value when applied to a specific set of factual circumstances. Lack of accurate predictions by the model could lead to erroneous conclusions with regard to materials selection and design. All results should be evaluated by an informed user.

INTENT AND USE

The algorithms, procedures, and computer programs described in this report constitute a methodology for computing the permeability of porous materials. They have been compiled from the best knowledge and understanding currently available, but have important limitations that must be understood and considered by the user. The programs are intended for use by persons competent in the field of computational materials science and with some familiarity with computers.

2 Computational Background

For slow incompressible steady-state flow, the linear Stokes equations may be simply written as [2]:

$$\eta \nabla^2 \vec{V}(\vec{r}) = \vec{\nabla} p(\vec{r}) \tag{1}$$

$$\vec{\nabla} \bullet \vec{V}(\vec{r}) = 0 \tag{2}$$

where $\vec{V}$ and p are the local velocity vector and pressure fields at position $\vec{r}$, respectively, and η is the fluid viscosity. The boundary conditions include the fluid velocity vanishing at all fluid-solid interfaces and an applied pressure gradient across two of the faces comprising the three-dimensional microstructure. For computational simplicity, typically, a pressure of 1 unit per voxel is applied across the microstructure. It is assumed that all pores in the three-dimensional microstructure are completely filled with fluid (saturated). Thus, in the text that follows, pore and fluid-filled pore are used interchangeably.

To numerically solve the Stokes equations, a finite-difference scheme in conjunction with the artificial compressibility relaxation algorithm is employed [3, 4, 10]. The three-dimensional set of voxels representing the three-dimensional microstructure are converted into a marker-and-cell mesh [4], where pressures are defined at the nodes (centers of each voxel) and fluid velocity components are defined along the center of bonds connecting neighboring nodes (see Figure 1). For such a realization of the microstructure, it is natural to define the pore/solid interface to be located at the midpoint of bonds connecting nearest neighbor nodes and for the pore/solid surface to be oriented perpendicular to the bond. A non-centered-difference-based realization of the ∇^2 operator has been derived to maintain at least 2nd order accuracy of the Stokes equation and to force the fluid velocities to zero at the pore solid interface. It is convenient to divide the ∇^2 operator into two parts. One corresponds to the direction parallel to the velocity being updated and the second corresponds to the directions transverse (perpendicular) to that velocity, $\nabla^2 = \nabla_{\parallel}^2 + \nabla_{\perp}^2$ (i.e., if we are updating the y component of the velocity V^y, then x and z are the transverse directions).

For the marker and cell mesh used, the $\nabla_{\parallel}^2$ operator, when discretized takes the usual form:

$$\nabla_{\parallel}^2 \vec{V}_0 \approx \vec{V}_{-1} + \vec{V}_1 - 2\vec{V}_0 \tag{3}$$

where the lattice spacing is taken as equal to one. The subscripts indicate neighboring mesh sites (0 corresponds to where the velocity is being evaluated, 1 is the forward one, and –1 is the backward one).

The discretized form of the $\nabla_{\perp}^2$ operator has 6 forms that depend on where the nearest solid voxels, in the transverse direction, are located. For example, the $\nabla_{\perp}^2$ operator will be

different if a pore node has solid voxels to its right and left or if a solid voxel is on the left and there is one pore voxel and then a solid voxel to the right (see Figure 2). Note, the zero velocity at the pore/ solid surface is now taken at the midpoint of the edge of the voxel corresponding to the solid. The distance of this edge point to its neighboring velocity is ½ of a lattice spacing.

Consider the case where there are two pore voxels sandwiched between solid voxels in the x direction (Figure 2). If one takes

$$\vec{V} = u\hat{i} + v\hat{j} + w\hat{k} \tag{4}$$

to determine $\frac{\partial^2 v_2}{\partial x^2}$, where v_2 is now the y component of the velocity at site 2, we first Taylor expand v_2 to represent its transverse neighbor's values:

$$\begin{aligned} v_1 &= v_2 - \frac{1}{2}\frac{\partial v_2}{\partial x} + \frac{1}{8}\frac{\partial^2 v_2}{\partial x^2} - \frac{1}{48}\frac{\partial^3 v_2}{\partial x^3} \\ v_3 &= v_2 + \frac{\partial v_2}{\partial x} + \frac{1}{2}\frac{\partial^2 v_2}{\partial x^2} + \frac{1}{6}\frac{\partial^3 v_2}{\partial x^3} \\ v_4 &= v_2 + \frac{3}{2}\frac{\partial v_2}{\partial x} + \frac{9}{8}\frac{\partial^2 v_2}{\partial x^2} + \frac{9}{16}\frac{\partial^3 v_2}{\partial x^3} \end{aligned} \tag{5}$$

Furthermore, because of the no-slip boundary condition, v_1=v_4=0.

Solving for $\frac{\partial^2 v_2}{\partial x^2}$, we obtain

$$\frac{\partial^2 v_2}{\partial x^2} = \frac{8}{3}v_3 - \frac{16}{3}v_2 \tag{6}$$

All six forms of the $\nabla_\perp^2$ operator have been derived in this fashion and the results are documented in the Stokes solver code listing found in the appendix. For each velocity component (x, y, and z), this results in thirty-six possible different cases depending on which of the neighbors within a distance of two voxels are also porosity voxels.

Once the finite difference solution converges sufficiently, the permeability, k, of the porous medium is calculated by volume averaging the local fluid velocity (in the direction of the flow) and applying the Darcy equation:

$$u = -\frac{k}{\eta}\frac{\Delta P}{L} \tag{6}$$

where u is the average fluid velocity in the direction of the flow (x-direction) for the porous media and L is the length of the sample porous medium across which there is an applied pressure difference of ΔP [2]. For a given porous medium, three separate runs of the computer codes may be conducted to determine the (different) permeabilities for flow in the x, y, and z directions, by simply transposing the indices in the (three) nested loops used to read in the starting microstructure.

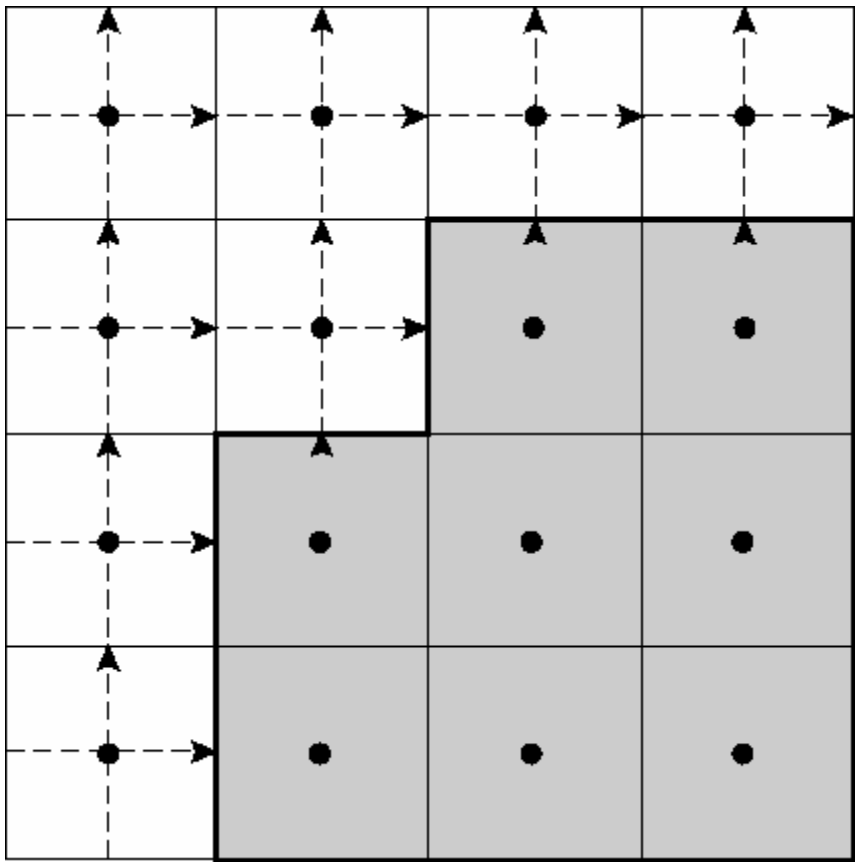


Figure 1. A piece of a pore space-solid interface in two dimensions [10]. The dark lines are pixel boundaries, and the dashed lines are the superimposed MAC mesh for the fluid-flow computation. The arrows show the location where the fluid velocities are determined, and the black circles show the nodes where the pressures are determined.

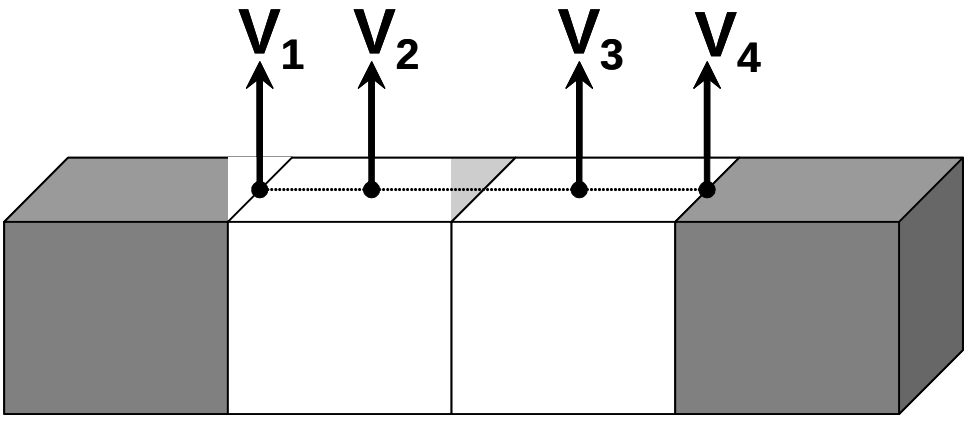


Figure 2. Schematic for the non-centered difference solution for the case described in the text, solving for v_2. Grey voxels illustrate solids and white voxels are porosity.

3 Computer Codes

The Stokes solver codes were originally developed in 1993, when computers were much slower and had much less memory than current systems. Therefore, a large effort was made to minimize the memory requirements of the codes and to develop codes that would be especially efficient on vector-based machines of that era. Rather than storing a complete representation of the three-dimensional microstructure, for example, simple lists of those nodes where pressures or x, y, or z components of the velocity vector are present were created and stored. (x,y,z) coordinates were stored in and extracted from a single 4-byte integer using logical **shift** and logical **and** operations. While the original version of the code was developed for (100 x 100 x 100) voxel microstructures and could therefore use an 8-bit representation (0-255) for each coordinate, the presented version of the code has a 9-bit size for each of these three coordinates. Thus, it could potentially be used on three-dimensional systems as large as 500 x 500 x 500, although currently configured for a (300 x 300 x 300) voxel microstructure. If execution for larger systems were desired, extensive modifications would have to be made to both the C and Fortran codes to take them to 10-bit (0-1023) versions, for example.

3.1 Pre-processor code: microperm.c

The microperm.c computer program is written in the C programming language and is menu-based. Thus, it is intended to be executed interactively. It can be compiled, on a Linux or UNIX-based machine for example, using a command such as:

cc microperm.c –o microperm –lm

When the program is first executed (by typing **./microperm** at the command prompt for example), the user will be prompted to enter a negative integer random number seed. Then, the main menu will be displayed and the interactive user session will begin. The main menu and a brief description of each option are as follows:

1) Add a flat inert aggregate to microstructure
2) Add spherical particles to microstructure
3) Add one-pixel solid particles to microstructure
4) Add a cylindrical tube to the microstructure
5) Place a square tube in the microstructure
6) Measure phase fractions
7) Measure phase fractions as a function of distance from aggregate surface
8) Measure single phase connectivity
9) Output permeability preprocessor file for x direction
10) Output permeability preprocessor file for y direction
11) Output permeability preprocessor file for z direction
12) Output permeability preprocessor file for pressures
13) Read in microstructure from one byte per pixel file
14) Read in microstructure from one integer (pixel) per line file

15) Exit

1) Add a flat inert aggregate to microstructure

This selection can be used to add a flat plate of a user-specified thickness through the center of the three-dimensional microstructure. Such a structure could represent a single aggregate surface in a simulated concrete microstructure, for example.

2) Add spherical particles to microstructure

This selection can be used to add up to 10 different size non-overlapping digitized spherical particles (diameters of 3 voxels up to 21 voxels) to the microstructure, to generate a starting porous media, for example. Once selected, the user must specify the number of different size spheres to place and the number and radius for each sphere size class. The spheres will be placed from largest to smallest at random locations in the 3-D microstructure such that they do not overlap any existing solids already present.

3) Add one-pixel solid particles to microstructure

This selection is similar to #2 except that one-voxel solid particles are added at random locations within the three-dimensional microstructure. The user simply specifies the number of these one-voxel particles to be placed. If both spherical and one-voxel particles are being placed in a microstructure, the user should first use menu selection #2 to place the spherical particles, followed by #3 for the one-voxel particles. If the one-voxel particles are placed first, there may be no spaces remaining available for the larger spherical particles.

4) Add a cylindrical tube to the microstructure

This selection places a digitized cylindrical tube of user-specified radius (the diameter will be equal to two*radius+1 in voxels so that the tube may be centered on a voxel in the yz plane) through the microstructure in the direction of the flow (pressure gradient). The user specifies the tube radius and the coordinates (y,z) at which the cylinder will be centered.

5) Place a square tube in the microstructure

This selection places a digitized square tube of user-specified size through the microstructure in the direction of the flow (pressure gradient). The user specifies the tube width (height) as an odd integer and the coordinates (y,z) at which it will be centered.

6) Measure phase fractions

This selection simply executes a complete scan of the three-dimensional microstructure and returns the counts of solid, aggregate, and porosity voxels, as well as the area (surface pixels) of the pore-solid (including aggregate) interfaces.

7) Measure phase fractions as a function of distance from aggregate surface

This selection returns the counts for porosity and solids as a function of distance from the aggregate surface in one-voxel increments, and can be used for quantifying interfacial transition zone (ITZ) microstructure.

8) Measure single phase connectivity

This selection examines the percolation characteristics of a single phase in what will ultimately be the direction of flow (the x direction). It returns a total voxel count for the phase of interest (porosity for example), the number of voxels accessible from one surface, and the number of voxels that are part of a percolated (connected) pathway through the microstructure.

9) Output permeability preprocessor file for x direction

This selection is one of the four that must be executed to establish the datafiles necessary for executing a permeability computation. It outputs a file, always called **xslist**, that contains an encoded list (one four byte integer contains a set of x, y, z coordinates) of voxels where an x-component of the velocity field will be present. The list is sorted into separate sublists for each of the 36 separate cases for subsequent application of the non-centered difference equations utilized in the Stokes solver.

10) Output permeability preprocessor file for y direction

This selection outputs the equivalent information as selection #9, but for the y direction and to a file named **yslist**.

11) Output permeability preprocessor file for z direction

This selection outputs the equivalent information as selection #9, but for the z direction and to a file named **zslist**.

12) Output permeability preprocessor file for pressures

This selection outputs the equivalent information as selection #9, but for the pressure fields and to a file named **pslist**. For this selection, there are not 36 cases to consider, so the encoded coordinates are not sorted but simply output as a single list.

13) Read in microstructure from one byte per pixel file

This is one of the two main selections used to read in an externally generated microstructure (model or real) from a file that must have the name **image3d**. It reads in a microstructure file that contains one byte per pixel (voxel) and is a simple list with the x dimension varying first, followed by the y dimension, and finally the z dimension. For a (300 x 300 x 300) voxel microstructure, the file should contain 27,000,000 bytes. A value of zero represents a porosity voxel and any other value (1 for example) represents solid.

14) Read in microstructure from one integer (pixel) per line file

This is one of the two main selections used to read in an externally generated microstructure (model or real) from a file that must have the name **micinput**. It reads in a microstructure file that contains one integer per pixel (voxel) and is a simple list with the x dimension varying first, followed by the y dimension, and finally the z dimension. For a (300 x 300 x 300) voxel microstructure, the file should contain 27,000,000 lines, with an integer value of 0 corresponding to porosity.

The user may modify this portion of the **microperm.c** program (subroutine readint) to effectively change the direction of the flow field (pressure gradient) for the 3-D microstructure. While the Stokes solver always applies the pressure gradient in the x-direction, within the **microperm.c** program, the order of assignment of coordinates to the microstructure can be varied (transposed) to effectively rotate the 3-D microstructure. In the program, the variables i, j, and k can be thought of as representing the x, y, and z directions, respectively, with i varying first, followed by j, then k. The 3-D array **cement** holds the input microstructure. Thus, to compute the permeability for the x, y, and z directions in the original 3-D microstructure, one of the following assignment statements can be made:

```
cement [i] [j] [k] = num;    /* Flow in x direction  of original microstructure */
cement [j] [k] [i] = num;    /* Flow in y direction  of original microstructure */
cement [k] [i] [j] = num;    /* Flow in z direction  of original microstructure */
```

Whenever this assignment statement is modified, of course, the **microperm.c** program must be recompiled prior to execution for the change to take effect.

15) Exit

This selection simply exits the program when the user is ready to quit and proceed to the permeability computation.

A typical interactive session might consist of reading in an external 3-D microstructure (selection #13 or #14), verifying the phase counts (selection #6), and establishing the datafiles necessary for the computation of permeability (selections #9 through #12) before finally exiting (selection #15).

3.2 Permeability (Stokes) solver: Fortran codes

The Stokes solver consists of a set of Fortran codes and input datafiles (such as **pslist**, **xslist**, **yslist**, and **zslist** described above). To compile the main **perm3d3.f** computer program, a typical statement might be:

f77 perm3d3.f –o perm3d3 -lm

where f77 should be replaced by the specific Fortran compiler being utilized. On a Linux or UNIX-based computer, the program could then be executed with a statement similar to

./perm3d3 >perm3d3.out &

where the Fortran UNIT=6 output is being piped to **perm3d3.out** and the **&** indicates that execution will take place in the background while the command prompt will be returned for the user's execution of other commands.

The input parameters for the permeability computation are contained in a file called **inpuf256** and include the following variables in the **perm3d3.f** program:

re - Reynold's number-type parameter for the solution algorithm
c2 - compressibility parameter for the artificial compressibilty algorithm
dt - time step parameter for the artificial compressibilty algorithm
ntot - number of solution iterations to perform between output statements
pl - pressure applied at left side of microstructure
pr - pressure applied at right side of microstructure
itsc - number of outer loop solution iterations to perform (=number of output lines)
vinit - initial guess for x-velocity component

Typically a pressure drop of 1 unit per voxel is specified. For example, as indicated by the listing for **inpuf256** in the appendix, for a (300 x 300 x 300) voxel microstructure, typically, pl=301. and pr=1. For most microstructures, the default values provided for these parameters in the appendix can be used. As this file is read in at runtime, if it is modified, the **perm3d3.f** program does not need to be recompiled for the changes to take effect.

The file **bcfeed256** contains the common variable list and sets the dimensions of the system and the number of expected porosity (pressure) nodes. The default values as contained in the version of the code provided in the appendix are for a 300 x 300 x 300 microstructure; hence the statement:

```
parameter (ix=304,iy=304,iz=304)
```

The values of ix, iy, and iz are 4 greater than 300 to allow for a two-voxel rim surrounding the microstructure for the implementation of the periodic boundary conditions. Additionally, the coordinates of the porosity sites are stored in a series of one-dimensional arrays. These should be dimensioned at a size greater than (the porosity fraction * the system size). The current statements are of the form

```
common/tabs1/lvxi(7500000),lvyi(7500000),lvzi(7500000)
```

indicating an expected porosity on the order of 7,500,000/300/300/300 = 27.8 %. If the system porosity is greater than this, all arrays dimensioned at 7,500,000 should be increased accordingly.

When the **bcfeed256** file is modified, the **perm3d3.f** program has to be recompiled for these changes to be implemented into the executable version (**perm3d3**).

The results of the computation are written to a velocity file (**perm256**) and a pressure file (**press256**). The **perm256** file contains the average x-component velocity computed at four different depths (distances) within the 3-D microstructure. As convergence is achieved, each of the four values should asymptote to a fixed value and the differences among the four of them will also generally become less (for a tube for example, the four values should be identical). Multiplying the average x-velocity returned in **perm256** by 1/re will provide the estimated permeability for the 3-D microstructure in units of square voxels. To convert to conventional units such as m^2, the user simply in turn multiplies this value by the scale factor for the microstructure (squared). For example, for a microstructure with a scale factor of 0.001 m per voxel, the value returned by **perm256** would be multiplied by (1/re)*0.001*0.001 to obtain the estimated permeability in m^2.

4 Example Applications

The codes had first been validated by computing the permeabilities of circular and square tubes. For a circular tube of radius r, the permeability, k, is given by [11, 12]:

$$k = \frac{r^2}{8} \tag{4}$$

For a square tube of width b, the equivalent expression is [12]:

$$k = \frac{0.422b^2}{12} \tag{5}$$

Three-dimensional microstructures with either a square or cylindrical tube of width or diameter of 25, respectively, were created and evaluated using the Stokes solver codes. For the square tube, the theoretical value for permeability of 21.979 (units of square pixels) can be compared to the solver's computed value of 21.976, giving an error of about 0.01 %. For the cylindrical tube, the area of the digitized cylinder in square pixels was 489, which is equivalent to a cylinder with a radius 12.476 pixels. Here, the theoretical value of 19.46 can be compared to the computed value of 19.09, giving an error of less than 2 %, some of which may be due to the "roughness" of the digitized circular perimeter in the model microstructure. Overall, the predictions of the code for square and circular tubes are very reasonable and serve as a validation of the equations, computational algorithms, and actual computer codes presented in this report.

As an example of a more concrete application, the codes have been recently employed for evaluating the permeability of various model pervious concretes. While first introduced over 30 years ago, recently pervious concretes are finding increased applications in driveways, parking lots, and other pavements where drainage issues and environmentally friendly construction are important [13-15]. It is basically a conventional concrete mixture in which the fine aggregates (sand) are removed and ultimately replaced by voids or porosity. Typical void porosities are on the order of 15 % to 25 % and measured permeabilities are on the order of 1×10^{-10} m^2 to 10×10^{-10} m^2 [13-15]. The Stokes solver has recently been applied to computing permeabilities for various microstructural models (300 x 300 x 300 voxels in size) that have been created in an attempt to simulate the real microstructure of pervious concretes. Table 1 summarizes some typical permeability values for real and model pervious concretes with porosities in the range of 20 % to 25 %. Clearly, the porous media model based on the three-dimensional reconstruction algorithm previously employed in reference [5] provides a microstructure whose permeability is closest to those observed experimentally. A more extensive comparison of the ability of various microstructural models to capture the reality of pervious concrete will be presented elsewhere [16].

Table 1. Experimental [13-15] and Model Permeabilities for Pervious Concretes

Experimental or Model	Source or Model Type	Porosity (%)	Permeability (m^2)
Experimental	[13]	22.4 %	1.54E-10
Experimental	[13]	24.9 %	4.04E-10
Experimental	[15]	22.5 %	3.3E-10
Experimental	[15]	23.2 %	6.6E-10
Experimental	[15]	23 %	2.3E-10
Experimental	[15]	22.6 %	3.3E-10
Experimental	[14]	21.8 %	4.8E-10
Experimental	[14]	24.6 %	3.3E-10
Model	Monosize spheres/shells [17]	26 %	2.22E-8
Model	Three size spheres/shells [17]	22.4 %	4.39E-9
Model	Reconstructed [5]	22.3 %	3.75E-10

5 Summary

A set of computer codes implementing a Stokes solver for estimating the permeability of a three-dimensional microstructure has been presented. Basic instructions for downloading and utilizing the computer software have been presented. The codes have been validated against known theoretical solutions for both cylindrical and square tubes, with excellent agreement between model and theoretical values. An example application of utilizing the codes to computer the permeabilities of pervious concretes has been briefly presented. It is envisioned that the codes will be useful to and used by a wide variety of researchers interested in porous media (rocks, bones, ceramics, concrete, catalysts, etc.).

6 References

[1] http://ciks.cbt.nist.gov/monograph, access verified March 20, 2007.

[2] Martys, N.S., Torquato, S., and Bentz, D.P., "Universal Scaling of Fluid Permeability for Sphere Packings," Physical Review E, Vol. 50 (1), 403-408, 1994.

[3] Peyret, R., and Taylor, T.D., Computational Methods for Fluid Flow, Springer-Verlag, New York, 1983.

[4] Schwartz, L.M., Martys, N., Bentz, D.P., Garboczi, E.J., and Torquato, S., "Cross Property Relations and Permeability Estimation in Model Porous Media," Physical Review E, Vol. 48 (6), 4584-4591, 1993.

[5] Bentz, D.P., and Martys, N.S., "Hydraulic Radius and Transport in Reconstructed Model Three-Dimensional Porous Media," Transport in Porous Media, Vol. 17 (3), 221-238, 1994.

[6] Skamser, D.J., Bentz, D.P., Coverdale, R.T., Spotz, M.S., Martys, N., Jennings, H.M., and Johnson, D.L., "Calculation of the Thermal Conductivity and Gas Permeability in a Uniaxial Bundle of Fibers," Journal of the American Ceramic Society, Vol. 77 (10), 2669-2680, 1994.

[7] Bentz, D.P., Garboczi, E.J. and Quenard, D.A., "Modelling Drying Shrinkage in Porous Materials Using Image Reconstruction: Application to Porous Vycor Glass," Modelling and Simulation in Materials Science and Engineering, Vol. 6, 211-236, 1998.

[8] Quenard, D.A., Xu, K., Kunzel, H.M., Bentz, D.P., and Martys, N.S., "Microstructure and Transport Properties of Porous Building Materials," Materials and Structures, Vol. 31 (209), 317-324, 1998.

[9] Bentz, D.P., Quenard, D.A., Kunzel, H.M., Baruchel, J., Peyrin, F., Martys, N.S., and Garboczi, E.J., "Microstructure and Transport Properties of Porous Building Materials. II: Three-Dimensional X-ray Tomographic Studies," Materials and Structures, Vol. 33, 147-153, 2000.

[10] Martys, N., and Garboczi, E.J., "Length Scales Relating Fluid Permeability and Electrical Conductivity in Random Two-Dimensional Model Porous Media," Physical Review B, Vol. 46, 6080, 1992.

[11] Middleman, S., Fundamentals of Polymer Processing, McGraw-Hill, New York, 1977.

[12] Sisavath, S., Jing, X., and Zimmerman, R.W., "Laminar Flow through Irregularly-Shaped Pores in Sedimentary Rocks," Transport in Porous Media, Vol. 45, 41-62, 2001.

[13] Montes, F., and Haselbach, L., "Measuring Hydraulic Conductivity on Pervious Concrete," Environmental Engineering Science, Vol. 23 (6), 960-969, 2006.

[14] Neithalath, N., Weiss, J., and Olek, J., "Characterizing Enhanced Porosity Concrete Using Electrical Impedance to Predict Acoustic and Hydraulic Performance," Cement and Concrete Research, Vol. 36 (11), 2074-2085, 2006.

[15] Schaefer, V.R., Wang, K., Suleiman, M.T., and Kevern, J.T., "Mix Design Development for Pervious Concrete in Cold Weather Climates," Final Report, Report 2006-1, National Concrete Pavement Technology Center, Feb. 2006.

[16] Bentz, D.P., "Virtual Pervious Concrete: Microstructure, Percolation, and Permeability," submitted to ACI Materials Journal, 2007.

[17] Bentz, D.P., Snyder, K.A., and Garboczi, E.J., "A Hard Core/Soft Shell Microstructural Model for Studying Percolation and Transport in Three-Dimensional Composite Media," NISTIR **6265**, U.S. Department of Commerce, January 1999.

Appendix – Computer Code Listings

microperm.c

```
#include <stdio.h>
#include <math.h>


/*#########################################################################*/
/*                                                                         */
/*      Program: microperm.C                                               */
/*      Programmer: Dale P. Bentz                                          */
/*            Building and Fire Research Laboratory                        */
/*                National Institute of Standards and Technology           */
/*                100 Bureau Drive  Stop 8615                              */
/*                Building 226 Room B-362                                  */
/*                Gaithersburg, MD  20899-8615                             */
/*                (301) 975-5865       FAX (301) 990-6891                  */
/*      Date: 1/07                                                         */
/*                                                                         */
/*      Purpose: To produce the pressure, x, y, and z lists for a          */
/*               permeability run using the 3-D Stokes solver              */
/*                                                                         */
/*#########################################################################*/

#define SYSSIZE 300

/* phase identifiers */
#define POROSITY 0
#define SOLIDS 1
#define SURFID 6
#define BURNT 7
#define AGG 2

/* number of different classes of spheres allowed */
#define NUMSIZES 10

/* definitions for portable random number generator */
#define M1 259200
#define IA1 7141
#define IC1 54773
#define RM1 (1.0/M1)
#define M2 134456
#define IA2 8121
```

```
#define IC2 28411
#define RM2 (1.0/M2)
#define M3 243000
#define IA3 4561
#define IC3 51349

/* 3-D microstructure is stored in 3-D array cement */
struct porelist{
        int px,py,pz;
        struct porelist *nextpore;
        };

struct porelist *portype [37];
static unsigned short int cement [SYSSIZE+1] [SYSSIZE+1] [SYSSIZE+1];
int aggsize,*seed;

/* Random number generator */
#define IA 16807
#define IM 2147483647
#define IQ 127773
#define IR 2836
#define NTAB 32
#define EPS (1.2E-07)
#define MAX(a,b) (a>b)?a:b
#define MIN(a,b) (a<b)?a:b

/* Random number generator ran1 from Computers in Physics */
/* Vol. 6 (5), 522-524, 1992, Press and Teukolsky */
/* To generate real random numbers 0.0-1.0 */
/* Should be seeded with a negative integer */
double ran1(idum)
int *idum;
{
      int j,k;
         static int iv[NTAB],iy=0;
         void nrerror();
      static double NDIV = 1.0/(1.0+(IM-1.0)/NTAB);
      static double RNMX = (1.0-EPS);
      static double AM = (1.0/IM);

         if ((*idum <= 0) || (iy == 0)) {
                *idum = MAX(-*idum,*idum);
           for(j=NTAB+7;j>=0;j--) {
                        k = *idum/IQ;
                        *idum = IA*(*idum-k*IQ)-IR*k;
                        if(*idum < 0) *idum += IM;
```

```
                        if(j < NTAB) iv[j] = *idum;
                }
                iy = iv[0];
        }
        k = *idum/IQ;
        *idum = IA*(*idum-k*IQ)-IR*k;
        if(*idum<0) *idum += IM;
        j = iy*NDIV;
        iy = iv[j];
        iv[j] = *idum;
        return MIN(AM*iy,RNMX);
}
#undef IA
#undef IM
#undef IQ
#undef IR
#undef NTAB
#undef EPS
#undef MAX
#undef MIN

/* Routine to add a cylindrical tube to the microstructure */
void addtube(){
        int ix,iy,iz,radtube,ztube,ytube;
        float dist;

        for(ix=1;ix<=SYSSIZE;ix++){
        for(iy=1;iy<=SYSSIZE;iy++){
        for(iz=1;iz<=SYSSIZE;iz++){
                cement [ix] [iy] [iz]=1;
        }
        }
        }

        printf("Enter radius of tube in pixels \n");
        scanf("%d",&radtube);
        printf("%d \n",radtube);

        printf("Enter y and z coordinates at which to center tube \n");
        scanf("%d %d",&ytube,&ztube);
        printf("%d %d \n",ytube,ztube);

        for(iy=ytube-radtube;iy<=(ytube+radtube);iy++){
        for(iz=ztube-radtube;iz<=(ztube+radtube);iz++){
                dist=sqrt((float)((iy-ytube)*(iy-ytube)+(iz-ztube)*(iz-ztube)));
```

```
                if((dist-0.5)<=(float)radtube){
                        for(ix=1;ix<=SYSSIZE;ix++){
                        cement [ix] [iy] [iz]=POROSITY;

                        }
                }
        }
        }
}

/* Routine to add a square tube to the microstructure */
void addsqtub(){
        int ix,iy,iz,radtube,ztube,ytube;

        for(ix=1;ix<=SYSSIZE;ix++){
        for(iy=1;iy<=SYSSIZE;iy++){
        for(iz=1;iz<=SYSSIZE;iz++){
                cement [ix] [iy] [iz]=1;
        }
        }
        }

        printf("Enter width of tube in pixels (odd integer) \n");
        scanf("%d",&radtube);
        printf("%d \n",radtube);

        printf("Enter y and z coordinates at which to center tube \n");
        scanf("%d %d",&ytube,&ztube);
        printf("%d %d \n",ytube,ztube);

        for(iy=ytube-(int)(radtube/2);iy<=(ytube+(int)(radtube/2));iy++){
        for(iz=ztube-(int)(radtube/2);iz<=(ztube+(int)(radtube/2));iz++){
        for(ix=1;ix<=SYSSIZE;ix++){
                cement [ix] [iy] [iz]=POROSITY;

        }
        }
        }
}

/* Routine to add a flat plate aggregate through the 3-D microstructure */
void addagg(){
        int ix,iy,iz;

        printf("Enter aggregate thickness (even number of pixels) \n");
        scanf("%d",&aggsize);
```

```
        printf("%d \n",aggsize);

/* Aggregate extends through entire system in x and y directions */
        for(iz=((SYSSIZE-aggsize+2)/2);iz<=((SYSSIZE+aggsize)/2);iz++){
        for(iy=1;iy<=SYSSIZE;iy++){
        for(ix=1;ix<=SYSSIZE;ix++){
                cement [ix] [iy] [iz]=AGG;
        }
        }
        }

}

/* routine to check or perform placement of sphere centered */
/* at location xin,yin,zin of radius radd */
/* wflg=1 check for fit of sphere */
/* wflg=2 place the sphere */
int chksph(xin,yin,zin,radd,wflg,phasein)
        int xin,yin,zin,radd,wflg,phasein;
        {
        int sflg,nofits,xp,yp,zp,i,j,k;
        float dist;

        nofits=0;       /* Flag indicating if placement is possible */

/* Check all pixels within the digitized sphere volume */
        for(i=xin-radd;((i<=xin+radd)&&(nofits==0));i++){
        for(j=yin-radd;((j<=yin+radd)&&(nofits==0));j++){
        for(k=zin-radd;((k<=zin+radd)&&(nofits==0));k++){

                xp=i;
                yp=j;
                zp=k;
                /* use periodic boundary conditions for sphere placement */
                if(xp<1) {xp+=SYSSIZE;}
                if(yp<1) {yp+=SYSSIZE;}
                if(zp<1) {zp+=SYSSIZE;}
                if(xp>SYSSIZE) {xp-=SYSSIZE;}
                if(yp>SYSSIZE) {yp-=SYSSIZE;}
                if(zp>SYSSIZE) {zp-=SYSSIZE;}

        /* Compute distance from center of sphere to this pixel */
                dist=sqrt((float)((i-xin)*(i-xin)+(j-yin)*(j-yin)+(k-zin)*(k-zin)));
                if((dist-0.5)<=(float)radd){
                                if(wflg==2){
                                        cement [xp] [yp] [zp] =phasein;
```

```
                        }
                    if((wflg==1)&&(cement [xp] [yp] [zp] !=POROSITY)){
                            nofits=1;
                    }
                }
        }
        }
        }

        /* return flag indicating if sphere will fit */
        return(nofits);
}

/* routine to place spheres of various sizes and phases at random */
/* locations in 3-D microstructure */
void gsphere(numgen,numeach,sizeeach,pheach)
        int numgen;
        long int numeach[NUMSIZES];
        int sizeeach[NUMSIZES],pheach[NUMSIZES];
        {
        int count,x,y,z,radius,ig,kg,phsph;
        long int jg;
        float rx,ry,rz;

/* Generate spheres of each size class in turn (largest first) */
        for(ig=0;ig<numgen;ig++){

                radius=sizeeach[ig];    /* radius for this class */
                phsph=pheach[ig];       /* phase for this class */
                /* loop for each sphere in this size class */
                for(jg=1;jg<=numeach[ig];jg++){

                        do{
                        /* generate a random center location for the sphere */
                                rx=ran1(seed);
                                ry=ran1(seed);
                                rz=ran1(seed);
                                x=(int)((float)SYSSIZE*rx)+1;
                                y=(int)((float)SYSSIZE*ry)+1;
                                z=(int)((float)SYSSIZE*rz)+1;
                                /* see if the sphere will fit at x,y,z */
                                count=chksph(x,y,z,radius,1,phsph);
                        } while(count!=0);

                        /* place the sphere at x,y,z */
                        printf("%d %d %d %d \n",x,y,z,radius);
```

```
                        count=chksph(x,y,z,radius,2,phsph);
                }
        }
}

/* routine to obtain user input and create a starting microstructure */
void create(){
        int numsize,sphrad [NUMSIZES],sphid [NUMSIZES];
        long int sphnum [NUMSIZES],inval1;
        int isph,inval;

        printf("Enter number of different size spheres to use (maximum is 10) \n");
        scanf("%d",&numsize);
        printf("%d \n",numsize);

        if((numsize>0)&&(numsize<(NUMSIZES+1))){
                printf("Enter number and size for each class (largest radius 1st) \n");

        /* Obtain input for each size class of spheres */
                for(isph=0;isph<numsize;isph++){
                        printf("Enter number of spheres of class %d \n",isph+1);
                        scanf("%ld",&inval1);
                        printf("%ld \n",inval1);
                        sphnum[isph]=inval1;

                        printf("Enter radius of spheres of class %d \n",isph+1);
                        printf("(Integer <=10 please) \n");
                        scanf("%d",&inval);
                        printf("%d \n",inval);
                        sphrad[isph]=inval;
                        sphid[isph]=1;
                }
                gsphere(numsize,sphnum,sphrad,sphid);
        }
}

/* routine to place one pixel solid particles at random
  unoccupied locations in 3-D system                              */
void genfill(ntopl,idtouse)
        long int ntopl;  /* Number of filler pixels to place */
        int idtouse;     /* Phase to be assigned to filler */
        {
        int pfill,xfill,yfill,zfill;
        long int ifill;
        float rxf,ryf,rzf;
```

```
/* Place each filler particle in turn */
        for(ifill=1;ifill<=ntopl;ifill++){

                pfill=0;
                /* place this filler pixel at a random unoccupied location */
                while (pfill==0){
                                rxf=ran1(seed);
                                ryf=ran1(seed);
                                rzf=ran1(seed);

                                xfill=(int)((float)SYSSIZE*rxf)+1;
                                yfill=(int)((float)SYSSIZE*ryf)+1;
                                zfill=(int)((float)SYSSIZE*rzf)+1;

                                if(cement [xfill] [yfill] [zfill] ==POROSITY){
                                        pfill=1;
                                        cement [xfill] [yfill] [zfill]=idtouse;
                                }
                }
        }
}

/* subroutine to obtain user input as to filler to be added */
void filler(){
        long int nadd;
        int fillid;

        printf("Enter number of one-pixel solid particles to add \n");
        scanf("%ld",&nadd);
        printf("%ld \n",nadd);
        fillid=1;

/* If phase ID is valid, call routine to place the filler */
        genfill(nadd,fillid);
}

/* routine to assess global phase fractions present in 3-D system */
void measure(){
        long int npor,nsolids,nagg,nsurf;
        int i,j,k,i1,j1,k1;

/* counters for the various phase fractions */
        npor=0;
        nsolids=0;
        nagg=0;
        nsurf=0;
```

```
/* Check all pixels in 3-D microstructure */
        for(i=1;i<=SYSSIZE;i++){
        for(j=1;j<=SYSSIZE;j++){
        for(k=1;k<=SYSSIZE;k++){

                if(cement [i] [j] [k]==POROSITY) {npor+=1;
                                                i1=i+1;
                                                if(i1>SYSSIZE){i1-=SYSSIZE;}
                                                if(cement [i1] [j] [k]!=POROSITY){nsurf+=1;}
                                                i1=i-1;
                                                if(i1<=0){i1+=SYSSIZE;}
                                                if(cement [i1] [j] [k]!=POROSITY){nsurf+=1;}
                                                j1=j+1;
                                                if(j1>SYSSIZE){j1-=SYSSIZE;}
                                                if(cement [i] [j1] [k]!=POROSITY){nsurf+=1;}
                                                j1=j-1;
                                                if(j1<=0){j1+=SYSSIZE;}
                                                if(cement [i] [j1] [k]!=POROSITY){nsurf+=1;}
                                                k1=k+1;
                                                if(k1>SYSSIZE){k1-=SYSSIZE;}
                                                if(cement [i] [j] [k1]!=POROSITY){nsurf+=1;}
                                                k1=k-1;
                                                if(k1<=0){k1+=SYSSIZE;}
                                                if(cement [i] [j] [k1]!=POROSITY){nsurf+=1;}
                                                        }
                if(cement [i] [j] [k]==SOLIDS) {nsolids+=1;}
                if(cement [i] [j] [k]==AGG) {nagg+=1;}
        }
        }
        }

/* Output results */
        printf("Porosity= %ld \n",npor);
        printf("Solids= %ld \n",nsolids);
        printf("Aggregate= %ld \n",nagg);
        printf("Surface pixels= %ld \n",nsurf);
}

/* Routine to measure phase fractions as a function of distance from */
/* aggregate surface                                                 */
void measagg(){
        int phase [11],ptot;
        int icnt,ix,iy,iz,phid,idist;

        printf("Distance  Porosity  Solids \n");
```

```
/* Increase distance from aggregate in increments of one */
        for(idist=1;idist<=(SYSSIZE-aggsize)/2;idist++){

        /* Initialize phase counts for this distance */
                for(icnt=0;icnt<11;icnt++){
                        phase[icnt]=0;
                }
                ptot=0;

/* Check all pixels which are this distance from aggregate surface */
                for(iy=1;iy<=SYSSIZE;iy++){
                for(ix=1;ix<=SYSSIZE;ix++){
                        /* Pixel above aggregate surface */
                        iz=((SYSSIZE-aggsize+2)/2)-idist;
                        phid=cement [ix] [iy] [iz];
                        ptot+=1;
                        phase[phid]+=1;

                        /* Pixel below aggregate surface */
                        iz=((SYSSIZE+aggsize)/2)+idist;
                        phid=cement [ix] [iy] [iz];
                        ptot+=1;
                        phase[phid]+=1;
                }
                }

                /* Output results for this distance from surface */
                printf("%d   %d   %d  \n",idist,phase[0],phase[1]);

        }
}

/* routine to assess the connectivity (percolation) of a single phase */
/* Two matrices are used here: one to store the recently burnt locations */
/*              the other to store the newly found burnt locations */
void connect(){
        long int ntop,nthrough,ncur,nnew,ntot;
        int i,inew,j,k,nmatx[90000],nmaty[90000],nmatz[90000];
        int ii,jj,kk;
        int xcn,ycn,zcn,npix,x1,y1,z1,igood,nnewx[90000],nnewy[90000],nnewz[90000];
        int jnew,icur;

        printf("Enter phase to analyze 0) pores 1) solids \n");
        scanf("%d",&npix);
        printf("%d \n",npix);
```

```
/* counters for number of pixels of phase accessible from top surface */
/* and number which are part of a percolated pathway */

        ntop=0;
        nthrough=0;

        /* percolation is assessed from left to right only */
        i=1;

        for(k=1;k<=SYSSIZE;k++){
        for(j=1;j<=SYSSIZE;j++){

                ncur=0;
                ntot=0;
                igood=0;        /* Indicates if bottom has been reached */
                if(cement [i] [j] [k]==npix){
                        /* Start a burn front */
                        cement [i] [j] [k]=BURNT;
                        ntot+=1;
                        ncur+=1;
                        /* burn front is stored in matrices nmat* */
                        /* and nnew* */
                        nmatx[ncur]=1;
                        nmaty[ncur]=j;
                        nmatz[ncur]=k;
                        /* Burn as long as new (fuel) pixels are found */
                        do{
                                nnew=0;
                                for(inew=1;inew<=ncur;inew++){
                                        xcn=nmatx[inew];
                                        ycn=nmaty[inew];
                                        zcn=nmatz[inew];

                                        /* Check all six neighbors */
                                        for(jnew=1;jnew<=6;jnew++){
                                                x1=xcn;
                                                y1=ycn;
                                                z1=zcn;
                                                if(jnew==1){x1-=1;}
                                                if(jnew==2){x1+=1;}
                                                if(jnew==3){y1-=1;}
                                                if(jnew==4){y1+=1;}
                                                if(jnew==5){z1-=1;}
                                                if(jnew==6){z1+=1;}
                                                if(z1<1){z1+=SYSSIZE;}
```

```
                        if(z1>SYSSIZE){z1-=SYSSIZE;}
                        if(y1<1){y1+=SYSSIZE;}
                        if(y1>SYSSIZE){y1-=SYSSIZE;}

        /* Nonperiodic in x-direction */
                if((x1>=1)&&(x1<=SYSSIZE)){
                        if(cement [x1] [y1] [z1]==npix){
                          ntot+=1;
                          cement [x1] [y1] [z1]=BURNT;
                                nnew+=1;
                                if(nnew>=90000){
                                 printf("error in size of nnew \n");
                                 }
                           nnewx[nnew]=x1;
                           nnewy[nnew]=y1;
                           nnewz[nnew]=z1;
        /* See if bottom of system has been reached */
                           if(x1==SYSSIZE){igood=1;}
                        }
                }
                }
        }
        if(nnew>0){
                ncur=nnew;
                /* update the burn front matrices */
                for(icur=1;icur<=ncur;icur++){
                        nmatx[icur]=nnewx[icur];
                        nmaty[icur]=nnewy[icur];
                        nmatz[icur]=nnewz[icur];
                }
        }
}while (nnew>0);

ntop+=ntot;
if(igood==1){
        nthrough+=ntot;
        if(npix==0){
        for(ii=1;ii<=SYSSIZE;ii++){
        for(jj=1;jj<=SYSSIZE;jj++){
        for(kk=1;kk<=SYSSIZE;kk++){

        if(cement [ii] [jj] [kk]==BURNT){
                cement [ii] [jj] [kk]=BURNT+1;
                }
        }
        }
```

```
                        }
                        }
                }
                else{
                        if(npix==0){
                        for(ii=1;ii<=SYSSIZE;ii++){
                        for(jj=1;jj<=SYSSIZE;jj++){
                        for(kk=1;kk<=SYSSIZE;kk++){

                        if(cement [ii] [jj] [kk]==BURNT){
                                cement [ii] [jj] [kk]=BURNT+2;
                                }
                        }
                        }
                        }
                        }
                }
        }

}
}

printf("Phase ID= %d \n",npix);
printf("Number accessible from top= %ld \n",ntop);
printf("Number contained in through pathways= %ld \n",nthrough);

/* return the burnt sites to their original phase values */
for(i=1;i<=SYSSIZE;i++){
for(j=1;j<=SYSSIZE;j++){
for(k=1;k<=SYSSIZE;k++){

        if((cement [i] [j] [k]==BURNT)||(cement [i] [j] [k]==(BURNT+2))){
/* Change disconnected porosity to solids */
                if(npix!=0){
                        cement [i] [j] [k]=npix;
                }
                else{

                        cement [i][j][k]=SOLIDS;
                }
        }
        if(cement [i] [j] [k]==(BURNT+1)){
                cement [i] [j] [k]=npix;
        }
}
}
```

```
	}
}

/* Routine to evaluate boundary condition in y-direction for x-velocity matrix */
int evalyx(x1,x2,ypx,zpx)
	int x1,x2,ypx,zpx;
{
	int valback,jlow,jhigh,j2,test;

	test=1;
	jlow=ypx-1;
	if(jlow<1){jlow=SYSSIZE;}
	jhigh=ypx+1;
	if(jhigh>SYSSIZE){jhigh=1;}

	if((cement [x1] [jlow] [zpx]==0)&&(cement [x2] [jlow] [zpx]==0)){
		test*=2;
	}
	if((cement [x1] [jhigh] [zpx]==0)&&(cement [x2] [jhigh] [zpx]==0)){
		test*=3;
	}

	valback=0;
	switch (test) {
		case 1:
		/* Neither neighbor has an x-velocity component */
			valback=1;
			break;
		case 2:
			j2=jlow-1;
			if(j2<1){j2=SYSSIZE;}
			if((cement [x1] [j2] [zpx]==0)&&(cement [x2] [j2] [zpx]==0)){
			/* 2 immediate lower neighbors have x-velocity components */
				valback=5;
			}
			else{
			/* only immediate lower neighbor has an x-velocity component */
				valback=3;
			}
			break;
		case 3:
			j2=jhigh+1;
			if(j2>SYSSIZE){j2=1;}
			if((cement [x1] [j2] [zpx]==0)&&(cement [x2] [j2] [zpx]==0)){
			/* 2 immediate upper neighbors have x-velocity components */
				valback=4;
```

```
                        }
                        else{
                        /* only immediate upper neighbor has an x-velocity component */
                                valback=2;
                        }
                        break;
                case 6:
                /* Both neighbors have an x-velocity component */
                        valback=6;
                        break;
                default:
                        printf("Error in evalyx routine \n");
                        break;
                }

        if(valback==0){printf("Error in valback in evalyx \n");}
        return(valback);
}

/* Routine to evaluate boundary condition in z-direction for x-velocity matrix */
int evalzx(x1,x2,ypx,zpx)
        int x1,x2,ypx,zpx;
{
        int valback,klow,khigh,k2,test;

        test=1;
        klow=zpx-1;
        if(klow<1){klow=SYSSIZE;}
        khigh=zpx+1;
        if(khigh>SYSSIZE){khigh=1;}

        if((cement [x1] [ypx] [klow]==0)&&(cement [x2] [ypx] [klow]==0)){
                test*=2;
        }
        if((cement [x1] [ypx] [khigh]==0)&&(cement [x2] [ypx] [khigh]==0)){
                test*=3;
        }

        valback=0;
        switch (test) {
                case 1:
                        valback=1;
                        break;
                case 2:
                        k2=klow-1;
                        if(k2<1){k2=SYSSIZE;}
```

```
                        if((cement [x1] [ypx] [k2]==0)&&(cement [x2] [ypx] [k2]==0)){
                                valback=5;
                        }
                        else{
                                valback=3;
                        }
                        break;
                case 3:
                        k2=khigh+1;
                        if(k2>SYSSIZE){k2=1;}
                        if((cement [x1] [ypx] [k2]==0)&&(cement [x2] [ypx] [k2]==0)){
                                valback=4;
                        }
                        else{
                                valback=2;
                        }
                        break;
                case 6:
                        valback=6;
                        break;
                default:
                        printf("Error in evalzx routine \n");
                        break;
                }

        if(valback==0){printf("Error in valback in evalzx \n");}
        return(valback);
}

/* Routine to output list of encoded coordinates where x-velocities are present */
void permx()
{
        struct porelist *newpore,*oldpore,*outpore;
        int typey,typez,typex;
        long int encode;
        int ix,ix1,jx,kx,jx1,kx1,ix2;
        int ity;
        long int count [37];
        FILE *pxout;

/* Initialize array of linked list pointers to NULL values */
/* There are 36 separate types of boundary conditions */
        for(ity=1;ity<=36;ity++){
                portype [ity]=NULL;
                count [ity]=0;
        }
```

```
	for(ix=1;ix<=SYSSIZE;ix++){
		ix1=ix+1;
		ix2=ix1;
		if(ix1>SYSSIZE){ix1=1;}
	for(jx=2;jx<=(SYSSIZE+1);jx++){
		jx1=jx;
		if(jx1>SYSSIZE){jx1=1;}
	for(kx=2;kx<=(SYSSIZE+1);kx++){
		kx1=kx;
		if(kx1>SYSSIZE){kx1=1;}

/* If both location are porous, then flow in x-direction is possible */
	if((cement [ix] [jx1] [kx1]==0)&&(cement [ix1] [jx1] [kx1]==0)){
		typey=evalyx(ix,ix1,jx1,kx1);  /* Determine y-direction BC */
		typez=evalzx(ix,ix1,jx1,kx1);  /* Determine z-direction BC */
		typex=6*(typey-1)+typez;
		newpore=(struct porelist *)malloc(sizeof(struct porelist));
		newpore->px=ix2;
		newpore->py=jx;
		newpore->pz=kx;
		/* Add new element to front of appropriate list */
		newpore->nextpore=portype [typex];
		portype [typex]=newpore;
		count [typex]+=1;  /* Increment appropriate counter */
	}

	}
	}
	}
/* Output results */
	pxout=fopen("xslist","w");
	for(ix=1;ix<=36;ix++){
		fprintf(pxout,"%ld \n",count [ix]);
		outpore=portype [ix];
		while (outpore!=NULL){
		/* Offset all three coordinates by 1 in preparation for adding a 1 pixel rim around
the system */
		encode=outpore->px+1+512*(outpore->py+1)+512*512*(outpore->pz+1);
			fprintf(pxout,"%ld \n",encode);
			oldpore=outpore;
			outpore=outpore->nextpore;
			free(oldpore);
		}
	}
	fclose(pxout);
```

```
}


/* Routine to evaluate boundary condition in y-direction for z-velocity matrix */
int evalyz(xpx,ypx,z1,z2)
        int xpx,ypx,z1,z2;
{
        int valback,jlow,jhigh,j2,test;

        test=1;
        jlow=ypx-1;
        if(jlow<1){jlow=SYSSIZE;}
        jhigh=ypx+1;
        if(jhigh>SYSSIZE){jhigh=1;}

        if((cement [xpx] [jlow] [z1]==0)&&(cement [xpx] [jlow] [z2]==0)){
                test*=2;
        }
        if((cement [xpx] [jhigh] [z1]==0)&&(cement [xpx] [jhigh] [z2]==0)){
                test*=3;
        }

        valback=0;
        switch (test) {
                case 1:
                        valback=1;
                        break;
                case 2:
                        j2=jlow-1;
                        if(j2<1){j2=SYSSIZE;}
                        if((cement [xpx] [j2] [z1]==0)&&(cement [xpx] [j2] [z2]==0)){
                                valback=5;
                        }
                        else{
                                valback=3;
                        }
                        break;
                case 3:
                        j2=jhigh+1;
                        if(j2>SYSSIZE){j2=1;}
                        if((cement [xpx] [j2] [z1]==0)&&(cement [xpx] [j2] [z2]==0)){
                                valback=4;
                        }
                        else{
                                valback=2;
```

```
                }
                break;
        case 6:
                valback=6;
                break;
        default:
                printf("Error in evalyz routine \n");
                break;
        }

    if(valback==0){printf("Error in valback in evalyz \n");}
    return(valback);
}

/* Routine to evaluate boundary condition in x-direction for z-velocity matrix */
int evalxz(xpx,ypx,z1,z2)
    int xpx,ypx,z1,z2;
{
    int valback,ilow,ihigh,i2,test;

    test=1;
    ilow=xpx-1;
    if(ilow<1){ilow=SYSSIZE;}
    ihigh=xpx+1;
    if(ihigh>SYSSIZE){ihigh=1;}

    if((cement [ilow] [ypx] [z1]==0)&&(cement [ilow] [ypx] [z2]==0)){
        test*=2;
    }
    if((cement [ihigh] [ypx] [z1]==0)&&(cement [ihigh] [ypx] [z2]==0)){
        test*=3;
    }

    valback=0;
    switch (test) {
        case 1:
            valback=1;
            break;
        case 2:
            i2=ilow-1;
            if(i2<1){i2=SYSSIZE;}
            if((cement [i2] [ypx] [z1]==0)&&(cement [i2] [ypx] [z2]==0)){
                valback=5;
            }
            else{
                valback=3;
```

```
                }
                break;
        case 3:
                i2=ihigh+1;
                if(i2>SYSSIZE){i2=1;}
                if((cement [i2] [ypx] [z1]==0)&&(cement [i2] [ypx] [z2]==0)){
                        valback=4;
                }
                else{
                        valback=2;
                }
                break;
        case 6:
                valback=6;
                break;
        default:
                printf("Error in evalxz routine \n");
                break;
        }

    if(valback==0){printf("Error in valback in evalxz \n");}
    return(valback);
}

/* Routine to output list of encoded coordinates where z-velocities are present */
void permz()
{
    struct porelist *newpore,*oldpore,*outpore;
    int typey,typez,typex;
    long int encode;
    int ix,jx,kx,kx1,ix1,jx1,kx2;
    int ity;
    long int count [37];
    FILE *pzout;

/* Initialize array of linked list pointers to NULL values */
    for(ity=1;ity<=36;ity++){
        portype [ity]=NULL;
        count [ity]=0;
    }

    for(kx=1;kx<=SYSSIZE;kx++){
        kx1=kx+1;
        kx2=kx1;
        if(kx1>SYSSIZE){kx1=1;}
    for(jx=2;jx<=(SYSSIZE+1);jx++){
```

```
                jx1=jx;
                if(jx1>SYSSIZE){jx1=1;}
        for(ix=2;ix<=(SYSSIZE+1);ix++){
                ix1=ix;
                if(ix1>SYSSIZE){ix1=1;}

        if((cement [ix1] [jx1] [kx]==0)&&(cement [ix1] [jx1] [kx1]==0)){
                typey=evalyz(ix1,jx1,kx,kx1);
                typex=evalxz(ix1,jx1,kx,kx1);
                typez=6*(typex-1)+typey;
                newpore=(struct porelist *)malloc(sizeof(struct porelist));
                newpore->px=ix;
                newpore->py=jx;
                newpore->pz=kx2;
                newpore->nextpore=portype [typez];
                portype [typez]=newpore;
                count [typez]+=1;
        }

        }
        }
        }

        pzout=fopen("zslist","w");
        for(ix=1;ix<=36;ix++){
                fprintf(pzout,"%ld \n",count [ix]);
                outpore=portype [ix];
                while (outpore!=NULL){
                encode=outpore->px+1+512*(outpore->py+1)+512*512*(outpore->pz+1);
                        fprintf(pzout,"%ld \n",encode);
                        oldpore=outpore;
                        outpore=outpore->nextpore;
                        free(oldpore);
                }
        }
        fclose(pzout);

}

/* Routine to evaluate boundary condition in x-direction for y-velocity matrix */
int evalxy(xpx,y1,y2,zpx)
        int xpx,y1,y2,zpx;
{
        int valback,ilow,ihigh,i2,test;

        test=1;
```

```
ilow=xpx-1;
if(ilow<1){ilow=SYSSIZE;}
ihigh=xpx+1;
if(ihigh>SYSSIZE){ihigh=1;}

if((cement [ilow] [y1] [zpx]==0)&&(cement [ilow] [y2] [zpx]==0)){
        test*=2;
}
if((cement [ihigh] [y1] [zpx]==0)&&(cement [ihigh] [y2] [zpx]==0)){
        test*=3;
}

valback=0;
switch (test) {
        case 1:
                valback=1;

                break;
        case 2:
                i2=ilow-1;
                if(i2<1){i2=SYSSIZE;}
                if((cement [i2] [y1] [zpx]==0)&&(cement [i2] [y2] [zpx]==0)){
                        valback=5;
                }
                else{
                        valback=3;
                }
                break;
        case 3:
                i2=ihigh+1;
                if(i2>SYSSIZE){i2=1;}
                if((cement [i2] [y1] [zpx]==0)&&(cement [i2] [y2] [zpx]==0)){
                        valback=4;
                }
                else{
                        valback=2;
                }
                break;
        case 6:
                valback=6;
                break;
        default:
                printf("Error in evalxy routine \n");
                break;
        }
```

```
        if(valback==0){printf("Error in valback in evalxy \n");}
        return(valback);
}


/* Routine to evaluate boundary condition in z-direction for y-velocity matrix */
int evalzy(xpx,y1,y2,zpx)
        int xpx,y1,y2,zpx;
{
        int valback,klow,khigh,k2,test;

        test=1;
        klow=zpx-1;
        if(klow<1){klow=SYSSIZE;}
        khigh=zpx+1;
        if(khigh>SYSSIZE){khigh=1;}

        if((cement [xpx] [y1] [klow]==0)&&(cement [xpx] [y2] [klow]==0)){
                test*=2;
        }
        if((cement [xpx] [y1] [khigh]==0)&&(cement [xpx] [y2] [khigh]==0)){
                test*=3;
        }

        valback=0;

        switch (test) {
                case 1:
                        valback=1;
                        break;
                case 2:
                        k2=klow-1;
                        if(k2<1){k2=SYSSIZE;}
                        if((cement [xpx] [y1] [k2]==0)&&(cement [xpx] [y2] [k2]==0)){
                                valback=5;
                        }
                        else{
                                valback=3;
                        }
                        break;
                case 3:
                        k2=khigh+1;
                        if(k2>SYSSIZE){k2=1;}
                        if((cement [xpx] [y1] [k2]==0)&&(cement [xpx] [y2] [k2]==0)){
                                valback=4;
                        }
```

```
                else{
                        valback=2;
                }
                break;
        case 6:
                valback=6;
                break;
        default:
                printf("Error in evalzy routine \n");
                break;
        }

    if(valback==0){printf("Error in valback in evalzy \n");}
    return(valback);
}

/* Routine to output list of encoded coordinates where y-velocities are present */
void permy()
{
    struct porelist *newpore,*oldpore,*outpore;
    int typey,typez,typex;
    long int encode;
    int ix,jx1,jx,kx,ix1,kx1,jx2;
    int ity;
    long int count [37];
    FILE *pyout;

/* Initialize array of linked list pointers to NULL values */
    for(ity=1;ity<=36;ity++){
        portype [ity]=NULL;
        count [ity]=0;
    }


    for(jx=1;jx<=SYSSIZE;jx++){
        jx1=jx+1;
        jx2=jx1;
        if(jx1>SYSSIZE){jx1=1;}
    for(ix=2;ix<=(SYSSIZE+1);ix++){
        ix1=ix;
        if(ix1>SYSSIZE){ix1=1;}
    for(kx=2;kx<=(SYSSIZE+1);kx++){
        kx1=kx;
        if(kx1>SYSSIZE){kx1=1;}

    if((cement [ix1] [jx] [kx1]==0)&&(cement [ix1] [jx1] [kx1]==0)){
```

```
                typex=evalxy(ix1,jx,jx1,kx1);
                typez=evalzy(ix1,jx,jx1,kx1);
                typey=6*(typex-1)+typez;
                newpore=(struct porelist *)malloc(sizeof(struct porelist));
                newpore->px=ix;
                newpore->py=jx2;
                newpore->pz=kx;

                newpore->nextpore=portype [typey];
                portype [typey]=newpore;
                count [typey]+=1;
        }

        }
        }
        }

        pyout=fopen("yslist","w");
        for(ix=1;ix<=36;ix++){
                fprintf(pyout,"%ld \n",count [ix]);
                outpore=portype [ix];
                while (outpore!=NULL){
                encode=outpore->px+1+512*(outpore->py+1)+512*512*(outpore->pz+1);
                        fprintf(pyout,"%ld \n",encode);
                        oldpore=outpore;
                        outpore=outpore->nextpore;
                        free(oldpore);
                }
        }
        fclose(pyout);

}

/* Routine to output list of encoded coordinates where pressures are present */
void permp()
{
        struct porelist *newpore,*oldpore,*outpore;
        long int encode;
        long int count;
        int ix,jx,kx,jx1,kx1;
        FILE *ppout;

        count=0;
        portype[1]=NULL;
        for(ix=2;ix<=SYSSIZE;ix++){
        for(jx=1;jx<=(SYSSIZE+1);jx++){
```

```
            jx1=jx;
            if(jx1>SYSSIZE){jx1=1;}
        for(kx=1;kx<=(SYSSIZE+1);kx++){
                kx1=kx;
                if(kx1>SYSSIZE){kx1=1;}

                if(cement [ix] [jx1] [kx1]==0){
                newpore=(struct porelist *)malloc(sizeof(struct porelist));
                newpore->px=ix;
                newpore->py=jx;
                newpore->pz=kx;
                newpore->nextpore=portype [1];
                portype [1]=newpore;
                count+=1;

                }
        }
        }
        }

        ppout=fopen("pslist","w");
        fprintf(ppout,"%ld \n",count);
        outpore=portype [1];
        while (outpore!=NULL){
                encode=outpore->px+1+512*(outpore->py+1)+512*512*(outpore->pz+1);
                fprintf(ppout,"%ld \n",encode);
                oldpore=outpore;
                outpore=outpore->nextpore;
                free(oldpore);
        }
        fclose(ppout);

}

readbyte(){
        int i,j,k,iy,num,num1;
        unsigned char cin;
        long int count,tcnt,ix;
        FILE *infile;

        count=tcnt=0;
        i=j=k=1;
        infile=fopen("image3d","r");

        for(ix=1;ix<=(SYSSIZE*SYSSIZE*SYSSIZE);ix++){
                if(feof(infile)!=0){
```

```
                printf("End of file encountered \n");
                printf("ix is %ld \n",ix);
        }
        fscanf(infile,"%c",&cin);
        num=(unsigned int)cin;
        if(num<0){num+=256;}
                if(num==0){
                        count+=1;
                }
                cement [i] [j] [k]=num;
                i+=1;
                if(i>SYSSIZE){
                        j+=1;
                        i=1;
                        if(j>SYSSIZE){
                                k+=1;
                                j=1;
                        }
                }
                tcnt+=1;
    }

    printf("tcnt is %ld and pore count is %ld \n",tcnt,count);
    fclose(infile);
}

readint(){
    int i,j,k,iy,num,num1;
    unsigned char c,cnum;
    long int count,tcnt,ix;
    FILE *infile;

    count=tcnt=0;
    i=j=k=1;
    infile=fopen("micinput","r");

    for(ix=1;ix<=(SYSSIZE*SYSSIZE*SYSSIZE);ix++){
        if(feof(infile)!=0){
                printf("End of file encountered \n");
                printf("ix is %ld \n",ix);
        }
        fscanf(infile,"%1d\n",&num);
                if(num==0){
                        count+=1;
                }
                cement [k] [i] [j]=num;
```

```
                        i+=1;
                        if(i>SYSSIZE){
                                j+=1;
                                i=1;
                                if(j>SYSSIZE){
                                        k+=1;
                                        j=1;
                                }
                        }
                        tcnt+=1;
        }

        printf("tcnt is %ld and pore count is %ld \n",tcnt,count);
        fclose(infile);
}

main(){
        int userc;      /* User choice from menu */
        int nseed,ig,jg,kg;

        printf("Enter random number seed value \n");
        scanf("%d",&nseed);
        printf("%d \n",nseed);
        seed=(&nseed);

/* Initialize counters and system paramters */
        aggsize=0;

/* clear the 3-D system to all porosity to start */
        for(ig=1;ig<=SYSSIZE;ig++){
        for(jg=1;jg<=SYSSIZE;jg++){
        for(kg=1;kg<=SYSSIZE;kg++){
                cement [ig] [jg] [kg]=0;
        }
        }
        }

/* present menu and execute user choice */
        do{
                printf(" \n Input User Choice \n");
                printf("1) Add a flat inert aggregate to microstructure \n");
                printf("2) Add spherical particles to microstructure \n");
                printf("3) Add one-pixel solid particles to microstructure \n");
                printf("4) Add a cylindrical tube to the microstructure \n");
                printf("5) Place a square tube in the microstructure \n");
                printf("6) Measure phase fractions \n");
```

```
printf("7) Measure phase fractions as a function ");
printf("   of distance from aggregate surface \n");
printf("8) Measure single phase connectivity \n");
printf("9) Output permeability preprocessor file for x direction \n");
printf("10) Output permeability preprocessor file for y direction \n");
printf("11) Output permeability preprocessor file for z direction \n");
printf("12) Output permeability preprocessor file for pressures \n");
printf("13) Read in microstructure from one byte per pixel file \n");
printf("14) Read in microstructure from one integer (pixel) per line file \n");
printf("15) Exit \n");

scanf("%d",&userc);
printf("%d \n",userc);
fflush(stdout);


switch (userc) {

                case 1:
                        addagg();
                        break;
                case 2:
                        create();
                        break;
                case 3:
                        filler();
                        break;
                case 4:
                        addtube();
                        break;
                case 5:
                        addsqtub();
                        break;
                case 6:
                        measure();
                        break;
                case 7:
                        measagg();
                        break;
                case 8:
                        connect();
                        break;
                case 9:
                        permx();
                        break;
                case 10:
```

```
                permy();
                break;
            case 11:
                permz();
                break;
            case 12:
                permp();
                break;
            case 13:
                readbyte();
                break;
            case 14:
                readint();
                break;
            default:
                break;
            }
        } while (userc<15);
}
```

Permeability – Stokes Solver Fortran Codes

perm3d3.f

```
      include 'bcfeed256'
      open(11,file='inpuf256')
      read(11,*)re
      read(11,*)c2
      read(11,*)dt
      read(11,*)ntot
      read(11,*)pl
      read(11,*)pr
      read(11,*)itsc
      read(11,*)vinit
      rre=1/re
      close(11)
      ixx=ix-1
      ixxx=ixx-1
      iyy=iy-1
      iyyy=iyy-1
      izz=iz-1
      izzz=izz-1
      cof1=16./3.
      cof2=8./3.
      dtc2=dt*c2
      c2=(.25/dt-rre)*2/dt

      call readvx
      call readvy
      call readvz
      call readp

C     Initialize the pressures and the compute initial fields
      call initpr
      call fields

C     Now iterate towards a solution
      do 3 it=1,itsc
      do 100 i=1,ntot

C     Update velocities
      call newvx
      call newvy
      call newvz
```

```
C     Apply periodic boundary conditions
      call perbcxyz
C     Update pressures
      call newpress

  100 continue
C     Output new pressure and velocity files
      call fields
    3 continue
      stop
      end

      include 'readvx.f'
      include 'readvy.f'
      include 'readvz.f'
      include 'readp.f'
      include 'initp2.f'
      include 'newpress.f'
      include 'newvxf.f'
      include 'newvyf.f'
      include 'newvzf.f'
      include 'newperbc.f'
      include 'perbcall.f'
      include 'fields.f'
```

inpuf256

0.01
1500000.
0.0004
400
301.
1.
80
0.001

bcfeed256

```
parameter (ix=304,iy=304,iz=304)
common/a1/values(ix,iy,iz,4)
common/limit/rre,dt,cof1,cof2,dtc2,pl,pr,c2
common/vss/vinit
common/tabs1/lvxi(7500000),lvyi(7500000),lvzi(7500000)
common/tabs2/lvxj(7500000),lvyj(7500000),lvzj(7500000)
common/tabs3/lvxk(7500000),lvyk(7500000),lvzk(7500000)
common/tabs4/lvpi(7500000),lvpj(7500000),lvpk(7500000)
common/labels/ixx,iyy,izz,ixxx,iyyy,izzz
common/tcv/xtc(36),ytc(36),ztc(36),itp
integer*4   xtc,ytc,ztc,itp
integer*2 lvxi,lvyi,lvzi,lvpi
integer*2 lvxj,lvyj,lvzj,lvpj
integer*2 lvxk,lvyk,lvzk,lvpk
```

readvx.f

```
      subroutine readvx
      integer*4 kk,j,numin
      include 'bcfeed256'

C     Open xslist file and read in lists of coordinates for x-velocity
C     components for the 36 possible cases
C     Use logical shifts and ands to convert single integer value
C     to the three (x,y,z) coordinates --- 9 bits per value (0-511)
      open(15,file='xslist')
      mask=511
      read(15,*)numin
      do 85 j=1,numin
      read(15,*)ivxspot
      lvxk(j)=jishft(ivxspot,-18)
      lvxi(j)=jiand(ivxspot,mask)
   85 lvxj(j)=jiand(jishft(ivxspot,-9),mask)

      xtc(1)=numin
      kk=numin
      do 80 k=2,36
      read(15,*)numin

      do 90 j=1,numin
      kk=kk+1
      read(15,*)ivxspot
      lvxk(kk)=jishft(ivxspot,-18)
      lvxi(kk)=jiand(ivxspot,mask)
   90 lvxj(kk)=jiand(jishft(ivxspot,-9),mask)

      xtc(k)=xtc(k-1)+numin
   80 continue

      close(15)
      return
      end
```

readvy.f

```
      subroutine readvy
      integer*4 kk,numin,j
      include 'bcfeed256'

      open(11,file='yslist')
      mask=511
      read(11,*)numin
      do 85 j=1,numin
      read(11,*)ivyspot
      lvyk(j)=jishft(ivyspot,-18)
      lvyi(j)=jiand(ivyspot,mask)
 85   lvyj(j)=jiand(jishft(ivyspot,-9),mask)
      ytc(1)=numin

      kk=numin
      do 80 k=2,36
      read(11,*)numin

      do 90 j=1,numin
      kk=kk+1
      read(11,*)ivyspot
      lvyk(kk)=jishft(ivyspot,-18)
      lvyi(kk)=jiand(ivyspot,mask)
 90   lvyj(kk)=jiand(jishft(ivyspot,-9),mask)

      ytc(k)=ytc(k-1)+numin
 80   continue

      close(11)
      return
      end
```

readvz.f

```
      subroutine readvz
      integer*4 kk,j,numin
      include 'bcfeed256'

      open(16,file='zslist')
      mask=511
      read(16,*)numin
      do 85 j=1,numin
      read(16,*)ivzspot
      lvzk(j)=jishft(ivzspot,-18)
      lvzi(j)=jiand(ivzspot,mask)
 85   lvzj(j)=jiand(jishft(ivzspot,-9),mask)
      ztc(1)=numin

      kk=numin
      do 80 k=2,36
      read(16,*)numin

      do 90 j=1,numin
      kk=kk+1
      read(16,*)ivzspot
      lvzk(kk)=jishft(ivzspot,-18)
      lvzi(kk)=jiand(ivzspot,mask)
 90   lvzj(kk)=jiand(jishft(ivzspot,-9),mask)

      ztc(k)=ztc(k-1)+numin
 80   continue

      close(16)
      return
      end
```

readp.f

```
      subroutine readp
      integer*4 j
      include 'bcfeed256'

C     Open pslist file and read in the coordinates of all pressure locations
C     in the three-dimensional microstructure
      open(17,file='pslist')
      mask=511
      read(17,*)itp

      do 90 j=1,itp
      read(17,*)ipspot
      lvpk(j)=jishft(ipspot,-18)
      lvpi(j)=jiand(ipspot,mask)
 90   lvpj(j)=jiand(jishft(ipspot,-9),mask)

      close(17)
      return
      end
```

initp2.f

```
      subroutine initpr
      integer pinit,p2,xinit,yinit,zinit
      integer*4 k6,mask,jj,ii,kk
      include 'bcfeed256'

      write(6,*)'At beginning of initpr'
      mask=511
      do 111 i=1,ix
      do 111 j=1,iy
      do 111 k=1,iz
      values(i,j,k,1)=0.
      values(i,j,k,2)=0.
      values(i,j,k,3)=0.
  111 values(i,j,k,4)=0.
      write(6,*)'After inititalization to 0 '

      pinc=(pl-pr)/float(ixx-3)
      do 50 k6=1,itp
      kk=lvpk(k6)
      jj=lvpj(k6)
      ii=lvpi(k6)
      fi=float(ii-2)
      pd=pl-fi*pinc
   50 values(ii,jj,kk,4)=pd
      write(6,*)'After initialization of pressures'

      do 51 j=2,iyyy
      do 51 k=2,izzz
      values(2,j,k,4)=pl
   51 values(ixxx,j,k,4)=pr
        write(6,*)'After pressure addition to edges'
      do 61 k6=xtc(35)+1,xtc(36)

      kk=lvxk(k6)
      ii=lvxi(k6)
      jj=lvxj(k6)
   61 values(ii,jj,kk,1)=vinit
      write(6,*)'After initialization of velocities'
      xinit=1
      call perbc(xinit)
      return
      end
```

newpress.f

```
      subroutine newpress
      integer pout,pin,vxin,vyin,vzin
      integer*4 k,jj,ii,kk,mask
      include 'bcfeed256'

      mask=511
C       routine to update pressure matrix
C       first update all nodes from pressure list
      do 105 k=1,itp
           kk=lvpk(k)
           ii=lvpi(k)
           jj=lvpj(k)
           i1=ii+1
           j1=jj+1
           k1=kk+1
           dot=-dtc2*(values(i1,jj,kk,1)-values(ii,jj,kk,1)+
     &values(ii,j1,kk,2)-values(ii,jj,kk,2)+
     &values(ii,jj,k1,3)-values(ii,jj,kk,3))
  105 values(ii,jj,kk,4)=values(ii,jj,kk,4)+dot

C       now update pressures on left and right edge
C       left edge is at x=2,  right edge is at x=ixxx
      do 106 j=2,iyyy
      j1=j+1
      do 106 k4=2,izzz
      k1=k4+1
      dot1=-dtc2*(values(3,j,k4,1)-values(2,j,k4,1)+
     &values(2,j1,k4,2)-values(2,j,k4,2)+
     &values(2,j,k1,3)-values(2,j,k4,3))
      values(2,j,k4,4)=values(2,j,k4,4)+dot1
  106 values(ixxx,j,k4,4)=values(ixxx,j,k4,4)+dot1

      return
      end
```

newvxf.f

```
      subroutine newvx
      integer*4 k,ii,jj,kk,mask
      include 'bcfeed256'

      mask=511
C       Case 1
C       No neighbors in y and z directions
      do 100 k=1,xtc(1)

            kk=lvxk(k)
            ii=lvxi(k)
            jj=lvxj(k)
            il=ii-1
            ih=ii+1
            d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
            d2vx=d2vx-8.*values(ii,jj,kk,1)
            d2vx=d2vx-8.*values(ii,jj,kk,1)
  100 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 2
C       No neighbors in y and one neighbor (+1) in z direction
      do 200 k=xtc(1)+1,xtc(2)

            kk=lvxk(k)
            ii=lvxi(k)
            jj=lvxj(k)
            il=ii-1
            ih=ii+1
            kh=kk+1
            d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
            d2vx=d2vx-8.*values(ii,jj,kk,1)
            d2vx=d2vx+cof2*values(ii,jj,kh,1)-cof1*values(ii,jj,kk,1)
  200 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 3
C       No neighbors in y and one neighbor (-1) in z direction
      do 300 k=xtc(2)+1,xtc(3)

            kk=lvxk(k)
```

```
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          kl=kk-1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx-8.*values(ii,jj,kk,1)
          d2vx=d2vx+cof2*values(ii,jj,kl,1)-cof1*values(ii,jj,kk,1)
  300 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 4
C       No neighbors in y and two neighbors (+1,+2) in z direction
      do 400 k=xtc(3)+1,xtc(4)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          k1=kk+1
          k2=k1+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx-8.*values(ii,jj,kk,1)
          d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,jj,k1,1)
  400 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 5
C       No neighbors in y and two neighbors (-1,-2) in z direction
      do 500 k=xtc(4)+1,xtc(5)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          k1=kk-1
          k2=k1-1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx-8.*values(ii,jj,kk,1)
          d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
```

```
     &2.*values(ii,jj,k1,1)
  500 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 6
C        No neighbors in y and two neighbors (-1,+1) in z direction
      do 600 k=xtc(5)+1,xtc(6)

           kk=lvxk(k)
           ii=lvxi(k)
           jj=lvxj(k)
           il=ii-1
           ih=ii+1
           kl=kk-1
           kh=kk+1
           d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
           d2vx=d2vx-8.*values(ii,jj,kk,1)
           d2vx=d2vx+values(ii,jj,kl,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jj,kh,1)
  600 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 7
C        One neighbor (+1) in y and none in z direction
      do 700 k=xtc(6)+1,xtc(7)

           kk=lvxk(k)
           ii=lvxi(k)
           jj=lvxj(k)
           il=ii-1
           ih=ii+1
           jh=jj+1
           d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
           d2vx=d2vx+cof2*values(ii,jh,kk,1)-cof1*values(ii,jj,kk,1)
           d2vx=d2vx-8.*values(ii,jj,kk,1)
  700 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 8
C        One neighbor (+1) in y and one (+1) in z direction
      do 800 k=xtc(7)+1,xtc(8)

           kk=lvxk(k)
           ii=lvxi(k)
```

```
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          jh=jj+1
          kh=kk+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx+cof2*values(ii,jh,kk,1)-cof1*values(ii,jj,kk,1)
          d2vx=d2vx+cof2*values(ii,jj,kh,1)-cof1*values(ii,jj,kk,1)
  800 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 9
C       One neighbor (+1) in y and one (-1) in z direction
      do 900 k=xtc(8)+1,xtc(9)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          jh=jj+1
          kl=kk-1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx+cof2*values(ii,jh,kk,1)-cof1*values(ii,jj,kk,1)
          d2vx=d2vx+cof2*values(ii,jj,kl,1)-cof1*values(ii,jj,kk,1)
  900 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 10
C       One neighbor (+1) in y and two (+1,+2) in z direction
      do 1000 k=xtc(9)+1,xtc(10)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          jh=jj+1
          k1=kk+1
          k2=k1+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx+cof2*values(ii,jh,kk,1)-cof1*values(ii,jj,kk,1)
          d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
```

```
     &2.*values(ii,jj,k1,1)
 1000 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C        Case 11
C         One neighbor (+1) in y and two (-1,-2) in z direction
      do 1100 k=xtc(10)+1,xtc(11)

           kk=lvxk(k)
           ii=lvxi(k)
           jj=lvxj(k)
           il=ii-1
           ih=ii+1
           jh=jj+1
           k1=kk-1
           k2=k1-1
           d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
           d2vx=d2vx+cof2*values(ii,jh,kk,1)-cof1*values(ii,jj,kk,1)
           d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,jj,k1,1)
 1100 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C        Case 12
C         One neighbor (+1) in y and two (-1,+1) in z direction
      do 1200 k=xtc(11)+1,xtc(12)

           kk=lvxk(k)
           ii=lvxi(k)
           jj=lvxj(k)
           il=ii-1
           ih=ii+1
           jh=jj+1
           kl=kk-1
           kh=kk+1
           d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
           d2vx=d2vx+cof2*values(ii,jh,kk,1)-cof1*values(ii,jj,kk,1)
           d2vx=d2vx+values(ii,jj,kl,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jj,kh,1)
 1200 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C        Case 13
C         One neighbor (-1) in y and none in z direction
```

```
      do 1300 k=xtc(12)+1,xtc(13)

            kk=lvxk(k)
            ii=lvxi(k)
            jj=lvxj(k)
            il=ii-1
            ih=ii+1
            jl=jj-1
            d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
            d2vx=d2vx+cof2*values(ii,jl,kk,1)-cof1*values(ii,jj,kk,1)
            d2vx=d2vx-8.*values(ii,jj,kk,1)
 1300 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 14
C       One neighbor (-1) in y and one (+1) in z direction
      do 1400 k=xtc(13)+1,xtc(14)

            kk=lvxk(k)
            ii=lvxi(k)
            jj=lvxj(k)
            il=ii-1
            ih=ii+1
            jl=jj-1
            kh=kk+1
            d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
            d2vx=d2vx+cof2*values(ii,jl,kk,1)-cof1*values(ii,jj,kk,1)
            d2vx=d2vx+cof2*values(ii,jj,kh,1)-cof1*values(ii,jj,kk,1)
 1400 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 15
C       One neighbor (-1) in y and one (-1) in z direction
      do 1500 k=xtc(14)+1,xtc(15)

            kk=lvxk(k)
            ii=lvxi(k)
            jj=lvxj(k)
            il=ii-1
            ih=ii+1
            jl=jj-1
            kl=kk-1
            d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
```

```
          d2vx=d2vx+cof2*values(ii,jl,kk,1)-cof1*values(ii,jj,kk,1)
          d2vx=d2vx+cof2*values(ii,jj,kl,1)-cof1*values(ii,jj,kk,1)
 1500 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C        Case 16
C        One neighbor (-1) in y and two (+1,+2) in z direction
      do 1600 k=xtc(15)+1,xtc(16)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          jl=jj-1
          k1=kk+1
          k2=k1+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx+cof2*values(ii,jl,kk,1)-cof1*values(ii,jj,kk,1)
          d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,jj,k1,1)
 1600 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C        Case 17
C        One neighbor (-1) in y and two (-1,-2) in z direction
      do 1700 k=xtc(16)+1,xtc(17)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          jl=jj-1
          k1=kk-1
          k2=k1-1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx+cof2*values(ii,jl,kk,1)-cof1*values(ii,jj,kk,1)
          d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,jj,k1,1)
 1700 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C        Case 18
```

```
C        One neighbor (-1) in y and two (-1,+1) in z direction
      do 1800 k=xtc(17)+1,xtc(18)

            kk=lvxk(k)
            ii=lvxi(k)
            jj=lvxj(k)
            il=ii-1
            ih=ii+1
            jl=jj-1
            kl=kk-1
            kh=kk+1
            d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
            d2vx=d2vx+cof2*values(ii,jl,kk,1)-cof1*values(ii,jj,kk,1)
            d2vx=d2vx+values(ii,jj,kl,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jj,kh,1)
 1800 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C        Case 19
C        Two neighbors (+1,+2) in y and none in z direction
      do 1900 k=xtc(18)+1,xtc(19)

            kk=lvxk(k)
            ii=lvxi(k)
            jj=lvxj(k)
            il=ii-1
            ih=ii+1
            j1=jj+1
            j2=j1+1
            d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
            d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,j1,kk,1)
            d2vx=d2vx-8.*values(ii,jj,kk,1)
 1900 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C        Case 20
C        Two neighbors (+1,+2) in y and one (+1) in z direction
      do 2000 k=xtc(19)+1,xtc(20)

            kk=lvxk(k)
            ii=lvxi(k)
            jj=lvxj(k)
            il=ii-1
```

```
          ih=ii+1
          kh=kk+1
          j1=jj+1
          j2=j1+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,j1,kk,1)
          d2vx=d2vx+cof2*values(ii,jj,kh,1)-cof1*values(ii,jj,kk,1)
 2000 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 21
C       Two neighbors (+1,+2) in y and one (-1) in z direction
      do 2100 k=xtc(20)+1,xtc(21)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          kl=kk-1
          j1=jj+1
          j2=j1+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,j1,kk,1)
          d2vx=d2vx+cof2*values(ii,jj,kl,1)-cof1*values(ii,jj,kk,1)
 2100 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 22
C       Two neighbors (+1,+2) in y and two (+1,+2) in z direction
      do 2200 k=xtc(21)+1,xtc(22)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          k1=kk+1
          k2=k1+1
          j1=jj+1
          j2=j1+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
```

```
     &values(ih,jj,kk,1)
           d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,j1,kk,1)
           d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,jj,k1,1)
  2200 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 23
C       Two neighbors (+1,+2) in y and two (-1,-2) in z direction
      do 2300 k=xtc(22)+1,xtc(23)

           kk=lvxk(k)
           ii=lvxi(k)
           jj=lvxj(k)
           il=ii-1
           ih=ii+1
           k1=kk-1
           k2=k1-1
           j1=jj+1
           j2=j1+1
           d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
           d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,j1,kk,1)
           d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,jj,k1,1)
  2300 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 24
C       Two neighbors (+1,+2) in y and two (-1,+1) in z direction
      do 2400 k=xtc(23)+1,xtc(24)

           kk=lvxk(k)
           ii=lvxi(k)
           jj=lvxj(k)
           il=ii-1
           ih=ii+1
           kl=kk-1
           kh=kk+1
           j1=jj+1
           j2=j1+1
           d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
           d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
```

```
     &2.*values(ii,j1,kk,1)
           d2vx=d2vx+values(ii,jj,kl,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jj,kh,1)
  2400 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 25
C       Two neighbors (-1,-2) in y and none in z direction
      do 2500 k=xtc(24)+1,xtc(25)

           kk=lvxk(k)
           ii=lvxi(k)
           jj=lvxj(k)
           il=ii-1
           ih=ii+1
           j1=jj-1
           j2=j1-1
           d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
           d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,j1,kk,1)
           d2vx=d2vx-8.*values(ii,jj,kk,1)
  2500 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 26
C       Two neighbors (-1,-2) in y and one (+1) in z direction
      do 2600 k=xtc(25)+1,xtc(26)

           kk=lvxk(k)
           ii=lvxi(k)
           jj=lvxj(k)
           il=ii-1
           ih=ii+1
           kh=kk+1
           j1=jj-1
           j2=j1-1
           d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
           d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,j1,kk,1)
           d2vx=d2vx+cof2*values(ii,jj,kh,1)-cof1*values(ii,jj,kk,1)
  2600 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 27
```

```
C       Two neighbors (-1,-2) in y and one (-1) in z direction
      do 2700 k=xtc(26)+1,xtc(27)

            kk=lvxk(k)
            ii=lvxi(k)
            jj=lvxj(k)
            il=ii-1
            ih=ii+1
            kl=kk-1
            j1=jj-1
            j2=j1-1
            d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
            d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,j1,kk,1)
            d2vx=d2vx+cof2*values(ii,jj,kl,1)-cof1*values(ii,jj,kk,1)
 2700 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 28
C       Two neighbors (-1,-2) in y and two (+1,+2) in z direction
      do 2800 k=xtc(27)+1,xtc(28)

            kk=lvxk(k)
            ii=lvxi(k)
            jj=lvxj(k)
            il=ii-1
            ih=ii+1
            k1=kk+1
            k2=k1+1
            j1=jj-1
            j2=j1-1
            d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
            d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,j1,kk,1)
            d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,jj,k1,1)
 2800 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 29
C       Two neighbors (-1,-2) in y and two (-1,-2) in z direction
      do 2900 k=xtc(28)+1,xtc(29)

            kk=lvxk(k)
```

```
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          k1=kk-1
          k2=k1-1
          j1=jj-1
          j2=j1-1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,j1,kk,1)
          d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,jj,k1,1)
 2900 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 30
C       Two neighbors (-1,-2) in y and two (-1,+1) in z direction
      do 3000 k=xtc(29)+1,xtc(30)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          kl=kk-1
          kh=kk+1
          j1=jj-1
          j2=j1-1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx-.2*values(ii,j2,kk,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,j1,kk,1)
          d2vx=d2vx+values(ii,jj,kl,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jj,kh,1)
 3000 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 31
C       Two neighbors (-1,+1) in y and none in z direction
      do 3100 k=xtc(30)+1,xtc(31)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
```

```
          il=ii-1
          ih=ii+1
          jl=jj-1
          jh=jj+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx+values(ii,jl,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jh,kk,1)
          d2vx=d2vx-8.*values(ii,jj,kk,1)
 3100 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C        Case 32
C        Two neighbors (-1,+1) in y and one (+1) in z direction
      do 3200 k=xtc(31)+1,xtc(32)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          kh=kk+1
          jl=jj-1
          jh=jj+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx+values(ii,jl,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jh,kk,1)
          d2vx=d2vx+cof2*values(ii,jj,kh,1)-cof1*values(ii,jj,kk,1)
 3200 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C        Case 33
C        Two neighbors (-1,+1) in y and one (-1) in z direction
      do 3300 k=xtc(32)+1,xtc(33)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          kl=kk-1
          jl=jj-1
          jh=jj+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
```

```
          d2vx=d2vx+values(ii,jl,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jh,kk,1)
          d2vx=d2vx+cof2*values(ii,jj,kl,1)-cof1*values(ii,jj,kk,1)
 3300 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 34
C       Two neighbors (-1,+1) in y and two (+1,+2) in z direction
      do 3400 k=xtc(33)+1,xtc(34)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          k1=kk+1
          k2=k1+1
          jl=jj-1
          jh=jj+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx+values(ii,jl,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jh,kk,1)
          d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
     &2.*values(ii,jj,k1,1)
 3400 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 35
C       Two neighbors (-1,+1) in y and two (-1,-2) in z direction
      do 3500 k=xtc(34)+1,xtc(35)

          kk=lvxk(k)
          ii=lvxi(k)
          jj=lvxj(k)
          il=ii-1
          ih=ii+1
          k1=kk-1
          k2=k1-1
          jl=jj-1
          jh=jj+1
          d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
          d2vx=d2vx+values(ii,jl,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jh,kk,1)
          d2vx=d2vx-.2*values(ii,jj,k2,1)-5.*values(ii,jj,kk,1)+
```

```
     &2.*values(ii,jj,k1,1)
 3500 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

C       Case 36
C       All neighbors in y and z directions
       do 3600 k=xtc(35)+1,xtc(36)

            kk=lvxk(k)
            ii=lvxi(k)
            jj=lvxj(k)
            il=ii-1
            ih=ii+1
            jl=jj-1
            jh=jj+1
            kl=kk-1
            kh=kk+1
            d2vx=values(il,jj,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ih,jj,kk,1)
            d2vx=d2vx+values(ii,jl,kk,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jh,kk,1)
            d2vx=d2vx+values(ii,jj,kl,1)-2.*values(ii,jj,kk,1)+
     &values(ii,jj,kh,1)
 3600 values(ii,jj,kk,1)=values(ii,jj,kk,1)+dt*
     &(-values(ii,jj,kk,4)+values(il,jj,kk,4)+d2vx*rre)

       return
       end
```

newvyf.f

```
      subroutine newvy
      integer*4 k,kk,ii,jj,mask
      include 'bcfeed256'

      mask=511
C       Case 1
C       No neighbors in x and z directions
      do 102 k=1,ytc(1)

           kk=lvyk(k)
           ii=lvyi(k)
           jj=lvyj(k)
           jl=jj-1
           jh=jj+1
           d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
           d2vx=d2vx-8.*values(ii,jj,kk,2)
           d2vx=d2vx-8.*values(ii,jj,kk,2)
  102 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 2
C       No neighbors in x and one neighbor (+1) in z direction
      do 202 k=ytc(1)+1,ytc(2)

           kk=lvyk(k)
           ii=lvyi(k)
           jj=lvyj(k)
           jl=jj-1
           jh=jj+1
           kh=kk+1
           d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
           d2vx=d2vx-8.*values(ii,jj,kk,2)
           d2vx=d2vx+cof2*values(ii,jj,kh,2)-cof1*values(ii,jj,kk,2)
  202 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 3
C       No neighbors in x and one neighbor (-1) in z direction
      do 302 k=ytc(2)+1,ytc(3)

           kk=lvyk(k)
```

```
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          kl=kk-1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx-8.*values(ii,jj,kk,2)
          d2vx=d2vx+cof2*values(ii,jj,kl,2)-cof1*values(ii,jj,kk,2)
  302 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 4
C       No neighbors in x and two neighbors (+1,+2) in z direction
      do 402 k=ytc(3)+1,ytc(4)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          k1=kk+1
          k2=k1+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx-8.*values(ii,jj,kk,2)
          d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
     &2.*values(ii,jj,k1,2)
  402 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 5
C       No neighbors in x and two neighbors (-1,-2) in z direction
      do 502 k=ytc(4)+1,ytc(5)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          k1=kk-1
          k2=k1-1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx-8.*values(ii,jj,kk,2)
          d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
```

```
     &2.*values(ii,jj,k1,2)
   502 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 6
C        No neighbors in x and two neighbors (-1,+1) in z direction
      do 602 k=ytc(5)+1,ytc(6)

           kk=lvyk(k)
           ii=lvyi(k)
           jj=lvyj(k)
           jl=jj-1
           jh=jj+1
           kl=kk-1
           kh=kk+1
           d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
           d2vx=d2vx-8.*values(ii,jj,kk,2)
           d2vx=d2vx+values(ii,jj,kl,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jj,kh,2)
    602 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 7
C        One neighbor (+1) in x and none in z direction
      do 702 k=ytc(6)+1,ytc(7)

           kk=lvyk(k)
           ii=lvyi(k)
           jj=lvyj(k)
           jl=jj-1
           jh=jj+1
           ih=ii+1
           d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
           d2vx=d2vx+cof2*values(ih,jj,kk,2)-cof1*values(ii,jj,kk,2)
           d2vx=d2vx-8.*values(ii,jj,kk,2)
    702 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 8
C        One neighbor (+1) in x and one (+1) in z direction
      do 802 k=ytc(7)+1,ytc(8)

           kk=lvyk(k)
           ii=lvyi(k)
```

```
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          ih=ii+1
          kh=kk+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx+cof2*values(ih,jj,kk,2)-cof1*values(ii,jj,kk,2)
          d2vx=d2vx+cof2*values(ii,jj,kh,2)-cof1*values(ii,jj,kk,2)
  802 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 9
C       One neighbor (+1) in x and one (-1) in z direction
      do 902 k=ytc(8)+1,ytc(9)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          ih=ii+1
          kl=kk-1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx+cof2*values(ih,jj,kk,2)-cof1*values(ii,jj,kk,2)
          d2vx=d2vx+cof2*values(ii,jj,kl,2)-cof1*values(ii,jj,kk,2)
  902 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 10
C       One neighbor (+1) in x and two (+1,+2) in z direction
      do 1002 k=ytc(9)+1,ytc(10)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          ih=ii+1
          k1=kk+1
          k2=k1+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx+cof2*values(ih,jj,kk,2)-cof1*values(ii,jj,kk,2)
          d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
```

```
     &2.*values(ii,jj,k1,2)
  1002 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 11
C        One neighbor (+1) in x and two (-1,-2) in z direction
      do 1102 k=ytc(10)+1,ytc(11)

           kk=lvyk(k)
           ii=lvyi(k)
           jj=lvyj(k)
           jl=jj-1
           jh=jj+1
           ih=ii+1
           k1=kk-1
           k2=k1-1
           d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
           d2vx=d2vx+cof2*values(ih,jj,kk,2)-cof1*values(ii,jj,kk,2)
           d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
     &2.*values(ii,jj,k1,2)
  1102 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 12
C        One neighbor (+1) in x and two (-1,+1) in z direction
      do 1202 k=ytc(11)+1,ytc(12)

           kk=lvyk(k)
           ii=lvyi(k)
           jj=lvyj(k)
           jl=jj-1
           jh=jj+1
           ih=ii+1
           kl=kk-1
           kh=kk+1
           d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
           d2vx=d2vx+cof2*values(ih,jj,kk,2)-cof1*values(ii,jj,kk,2)
           d2vx=d2vx+values(ii,jj,kl,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jj,kh,2)
  1202 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 13
C        One neighbor (-1) in x and none in z direction
```

```
      do 1302 k=ytc(12)+1,ytc(13)

            kk=lvyk(k)
            ii=lvyi(k)
            jj=lvyj(k)
            jl=jj-1
            jh=jj+1
            il=ii-1
            d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
            d2vx=d2vx+cof2*values(il,jj,kk,2)-cof1*values(ii,jj,kk,2)
            d2vx=d2vx-8.*values(ii,jj,kk,2)
 1302 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 14
C       One neighbor (-1) in x and one (+1) in z direction
      do 1402 k=ytc(13)+1,ytc(14)

            kk=lvyk(k)
            ii=lvyi(k)
            jj=lvyj(k)
            jl=jj-1
            jh=jj+1
            il=ii-1
            kh=kk+1
            d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
            d2vx=d2vx+cof2*values(il,jj,kk,2)-cof1*values(ii,jj,kk,2)
            d2vx=d2vx+cof2*values(ii,jj,kh,2)-cof1*values(ii,jj,kk,2)
 1402 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 15
C       One neighbor (-1) in x and one (-1) in z direction
      do 1502 k=ytc(14)+1,ytc(15)

            kk=lvyk(k)
            ii=lvyi(k)
            jj=lvyj(k)
            jl=jj-1
            jh=jj+1
            il=ii-1
            kl=kk-1
            d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
```

```
          d2vx=d2vx+cof2*values(il,jj,kk,2)-cof1*values(ii,jj,kk,2)
          d2vx=d2vx+cof2*values(ii,jj,kl,2)-cof1*values(ii,jj,kk,2)
 1502 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 16
C        One neighbor (-1) in x and two (+1,+2) in z direction
      do 1602 k=ytc(15)+1,ytc(16)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          il=ii-1
          k1=kk+1
          k2=k1+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx+cof2*values(il,jj,kk,2)-cof1*values(ii,jj,kk,2)
          d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
     &2.*values(ii,jj,k1,2)
 1602 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 17
C        One neighbor (-1) in x and two (-1,-2) in z direction
      do 1702 k=ytc(16)+1,ytc(17)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          il=ii-1
          k1=kk-1
          k2=k1-1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx+cof2*values(il,jj,kk,2)-cof1*values(ii,jj,kk,2)
          d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
     &2.*values(ii,jj,k1,2)
 1702 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 18
```

```
C        One neighbor (-1) in x and two (-1,+1) in z direction
      do 1802 k=ytc(17)+1,ytc(18)

            kk=lvyk(k)
            ii=lvyi(k)
            jj=lvyj(k)
            jl=jj-1
            jh=jj+1
            il=ii-1
            kl=kk-1
            kh=kk+1
            d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
            d2vx=d2vx+cof2*values(il,jj,kk,2)-cof1*values(ii,jj,kk,2)
            d2vx=d2vx+values(ii,jj,kl,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jj,kh,2)
 1802 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 19
C        Two neighbors (+1,+2) in x and none in z direction
      do 1902 k=ytc(18)+1,ytc(19)

            kk=lvyk(k)
            ii=lvyi(k)
            jj=lvyj(k)
            jl=jj-1
            jh=jj+1
            i1=ii+1
            i2=i1+1
            d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
            d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
     &2.*values(i1,jj,kk,2)
            d2vx=d2vx-8.*values(ii,jj,kk,2)
 1902 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 20
C        Two neighbors (+1,+2) in x and one (+1) in z direction
      do 2002 k=ytc(19)+1,ytc(20)

            kk=lvyk(k)
            ii=lvyi(k)
            jj=lvyj(k)
            jl=jj-1
```

```
          jh=jj+1
          kh=kk+1
          i1=ii+1
          i2=i1+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
     &2.*values(i1,jj,kk,2)
          d2vx=d2vx+cof2*values(ii,jj,kh,2)-cof1*values(ii,jj,kk,2)
 2002 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 21
C        Two neighbors (+1,+2) in x and one (-1) in z direction
      do 2102 k=ytc(20)+1,ytc(21)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          kl=kk-1
          i1=ii+1
          i2=i1+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
     &2.*values(i1,jj,kk,2)
          d2vx=d2vx+cof2*values(ii,jj,kl,2)-cof1*values(ii,jj,kk,2)
 2102 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 22
C        Two neighbors (+1,+2) in x and two (+1,+2) in z direction
      do 2202 k=ytc(21)+1,ytc(22)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          k1=kk+1
          k2=k1+1
          i1=ii+1
          i2=i1+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
```

```
     &values(ii,jh,kk,2)
           d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
     &2.*values(i1,jj,kk,2)
           d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
     &2.*values(ii,jj,k1,2)
  2202 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 23
C       Two neighbors (+1,+2) in x and two (-1,-2) in z direction
      do 2302 k=ytc(22)+1,ytc(23)

           kk=lvyk(k)
           ii=lvyi(k)
           jj=lvyj(k)
           jl=jj-1
           jh=jj+1
           k1=kk-1
           k2=k1-1
           i1=ii+1
           i2=i1+1
           d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
           d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
     &2.*values(i1,jj,kk,2)
           d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
     &2.*values(ii,jj,k1,2)
  2302 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 24
C       Two neighbors (+1,+2) in x and two (-1,+1) in z direction
      do 2402 k=ytc(23)+1,ytc(24)

           kk=lvyk(k)
           ii=lvyi(k)
           jj=lvyj(k)
           jl=jj-1
           jh=jj+1
           kl=kk-1
           kh=kk+1
           i1=ii+1
           i2=i1+1
           d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
           d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
```

```
     &2.*values(i1,jj,kk,2)
           d2vx=d2vx+values(ii,jj,kl,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jj,kh,2)
  2402 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 25
C        Two neighbors (-1,-2) in x and none in z direction
       do 2502 k=ytc(24)+1,ytc(25)

           kk=lvyk(k)
           ii=lvyi(k)
           jj=lvyj(k)
           jl=jj-1
           jh=jj+1
           i1=ii-1
           i2=i1-1
           d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
           d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
     &2.*values(i1,jj,kk,2)
           d2vx=d2vx-8.*values(ii,jj,kk,2)
  2502 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 26
C        Two neighbors (-1,-2) in x and one (+1) in z direction
       do 2602 k=ytc(25)+1,ytc(26)

           kk=lvyk(k)
           ii=lvyi(k)
           jj=lvyj(k)
           jl=jj-1
           jh=jj+1
           kh=kk+1
           i1=ii-1
           i2=i1-1
           d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
           d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
     &2.*values(i1,jj,kk,2)
           d2vx=d2vx+cof2*values(ii,jj,kh,2)-cof1*values(ii,jj,kk,2)
  2602 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 27
```

```
C       Two neighbors (-1,-2) in x and one (-1) in z direction
      do 2702 k=ytc(26)+1,ytc(27)

            kk=lvyk(k)
            ii=lvyi(k)
            jj=lvyj(k)
            jl=jj-1
            jh=jj+1
            kl=kk-1
            i1=ii-1
            i2=i1-1
            d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
            d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
     &2.*values(i1,jj,kk,2)
            d2vx=d2vx+cof2*values(ii,jj,kl,2)-cof1*values(ii,jj,kk,2)
 2702 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 28
C       Two neighbors (-1,-2) in x and two (+1,+2) in z direction
      do 2802 k=ytc(27)+1,ytc(28)

            kk=lvyk(k)
            ii=lvyi(k)
            jj=lvyj(k)
            jl=jj-1
            jh=jj+1
            k1=kk+1
            k2=k1+1
            i1=ii-1
            i2=i1-1
            d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
            d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
     &2.*values(i1,jj,kk,2)
            d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
     &2.*values(ii,jj,k1,2)
 2802 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 29
C       Two neighbors (-1,-2) in x and two (-1,-2) in z direction
      do 2902 k=ytc(28)+1,ytc(29)

            kk=lvyk(k)
```

```
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          k1=kk-1
          k2=k1-1
          i1=ii-1
          i2=i1-1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
     &2.*values(i1,jj,kk,2)
          d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
     &2.*values(ii,jj,k1,2)
 2902 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 30
C       Two neighbors (-1,-2) in x and two (-1,+1) in z direction
      do 3002 k=ytc(29)+1,ytc(30)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          kl=kk-1
          kh=kk+1
          i1=ii-1
          i2=i1-1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx-.2*values(i2,jj,kk,2)-5.*values(ii,jj,kk,2)+
     &2.*values(i1,jj,kk,2)
          d2vx=d2vx+values(ii,jj,kl,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jj,kh,2)
 3002 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 31
C       Two neighbors (-1,+1) in x and none in z direction
      do 3102 k=ytc(30)+1,ytc(31)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
```

```
          jl=jj-1
          jh=jj+1
          il=ii-1
          ih=ii+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx+values(il,jj,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ih,jj,kk,2)
          d2vx=d2vx-8.*values(ii,jj,kk,2)
 3102 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 32
C        Two neighbors (-1,+1) in x and one (+1) in z direction
      do 3202 k=ytc(31)+1,ytc(32)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          kh=kk+1
          il=ii-1
          ih=ii+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx+values(il,jj,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ih,jj,kk,2)
          d2vx=d2vx+cof2*values(ii,jj,kh,2)-cof1*values(ii,jj,kk,2)
 3202 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 33
C        Two neighbors (-1,+1) in x and one (-1) in z direction
      do 3302 k=ytc(32)+1,ytc(33)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          kl=kk-1
          il=ii-1
          ih=ii+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
```

```
          d2vx=d2vx+values(il,jj,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ih,jj,kk,2)
          d2vx=d2vx+cof2*values(ii,jj,kl,2)-cof1*values(ii,jj,kk,2)
 3302 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 34
C        Two neighbors (-1,+1) in x and two (+1,+2) in z direction
      do 3402 k=ytc(33)+1,ytc(34)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          k1=kk+1
          k2=k1+1
          il=ii-1
          ih=ii+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx+values(il,jj,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ih,jj,kk,2)
          d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
     &2.*values(ii,jj,k1,2)
 3402 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C        Case 35
C        Two neighbors (-1,+1) in x and two (-1,-2) in z direction
      do 3502 k=ytc(34)+1,ytc(35)

          kk=lvyk(k)
          ii=lvyi(k)
          jj=lvyj(k)
          jl=jj-1
          jh=jj+1
          k1=kk-1
          k2=k1-1
          il=ii-1
          ih=ii+1
          d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
          d2vx=d2vx+values(il,jj,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ih,jj,kk,2)
          d2vx=d2vx-.2*values(ii,jj,k2,2)-5.*values(ii,jj,kk,2)+
```

```
     &2.*values(ii,jj,k1,2)
 3502 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

C       Case 36
C       All neighbors in x and z directions
       do 3602 k=ytc(35)+1,ytc(36)

            kk=lvyk(k)
            ii=lvyi(k)
            jj=lvyj(k)
            jl=jj-1
            jh=jj+1
            il=ii-1
            ih=ii+1
            kl=kk-1
            kh=kk+1
            d2vx=values(ii,jl,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jh,kk,2)
            d2vx=d2vx+values(il,jj,kk,2)-2.*values(ii,jj,kk,2)+
     &values(ih,jj,kk,2)
            d2vx=d2vx+values(ii,jj,kl,2)-2.*values(ii,jj,kk,2)+
     &values(ii,jj,kh,2)
 3602 values(ii,jj,kk,2)=values(ii,jj,kk,2)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jl,kk,4)+d2vx*rre)

       return
       end
```

newvzf.f

```
      subroutine newvz
      integer*4 k,kk,ii,jj,mask
      include 'bcfeed256'

      mask=511
C       Case 1
C       No neighbors in y and x directions
      do 103 k=1,ztc(1)

            kk=lvzk(k)
            ii=lvzi(k)
            jj=lvzj(k)
            kl=kk-1
            kh=kk+1
            d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
            d2vx=d2vx-8.*values(ii,jj,kk,3)
            d2vx=d2vx-8.*values(ii,jj,kk,3)
  103 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 2
C       No neighbors in y and one neighbor (+1) in x direction
      do 203 k=ztc(1)+1,ztc(2)

            kk=lvzk(k)
            ii=lvzi(k)
            jj=lvzj(k)
            kl=kk-1
            kh=kk+1
            ih=ii+1
            d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
            d2vx=d2vx-8.*values(ii,jj,kk,3)
            d2vx=d2vx+cof2*values(ih,jj,kk,3)-cof1*values(ii,jj,kk,3)
  203 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 3
C       No neighbors in y and one neighbor (-1) in x direction
      do 303 k=ztc(2)+1,ztc(3)

            kk=lvzk(k)
```

```
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          il=ii-1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx-8.*values(ii,jj,kk,3)
          d2vx=d2vx+cof2*values(il,jj,kk,3)-cof1*values(ii,jj,kk,3)
  303 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 4
C       No neighbors in y and two neighbors (+1,+2) in x direction
      do 403 k=ztc(3)+1,ztc(4)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          i1=ii+1
          i2=i1+1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx-8.*values(ii,jj,kk,3)
          d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(i1,jj,kk,3)
  403 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 5
C       No neighbors in y and two neighbors (-1,-2) in x direction
      do 503 k=ztc(4)+1,ztc(5)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          i1=ii-1
          i2=i1-1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx-8.*values(ii,jj,kk,3)
          d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
```

```
     &2.*values(i1,jj,kk,3)
   503 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 6
C        No neighbors in y and two neighbors (-1,+1) in x direction
      do 603 k=ztc(5)+1,ztc(6)

           kk=lvzk(k)
           ii=lvzi(k)
           jj=lvzj(k)
           kl=kk-1
           kh=kk+1
           il=ii-1
           ih=ii+1
           d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
           d2vx=d2vx-8.*values(ii,jj,kk,3)
           d2vx=d2vx+values(il,jj,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ih,jj,kk,3)
   603 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 7
C        One neighbor (+1) in y and none in x direction
      do 703 k=ztc(6)+1,ztc(7)

           kk=lvzk(k)
           ii=lvzi(k)
           jj=lvzj(k)
           kl=kk-1
           kh=kk+1
           jh=jj+1
           d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
           d2vx=d2vx+cof2*values(ii,jh,kk,3)-cof1*values(ii,jj,kk,3)
           d2vx=d2vx-8.*values(ii,jj,kk,3)
   703 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 8
C        One neighbor (+1) in y and one (+1) in x direction
      do 803 k=ztc(7)+1,ztc(8)

           kk=lvzk(k)
           ii=lvzi(k)
```

```
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          jh=jj+1
          ih=ii+1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx+cof2*values(ii,jh,kk,3)-cof1*values(ii,jj,kk,3)
          d2vx=d2vx+cof2*values(ih,jj,kk,3)-cof1*values(ii,jj,kk,3)
  803 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 9
C       One neighbor (+1) in y and one (-1) in x direction
      do 903 k=ztc(8)+1,ztc(9)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          jh=jj+1
          il=ii-1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx+cof2*values(ii,jh,kk,3)-cof1*values(ii,jj,kk,3)
          d2vx=d2vx+cof2*values(il,jj,kk,3)-cof1*values(ii,jj,kk,3)
  903 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 10
C       One neighbor (+1) in y and two (+1,+2) in x direction
      do 1003 k=ztc(9)+1,ztc(10)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          jh=jj+1
          i1=ii+1
          i2=i1+1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx+cof2*values(ii,jh,kk,3)-cof1*values(ii,jj,kk,3)
          d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
```

```
     &2.*values(i1,jj,kk,3)
 1003 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C        Case 11
C         One neighbor (+1) in y and two (-1,-2) in x direction
      do 1103 k=ztc(10)+1,ztc(11)

           kk=lvzk(k)
           ii=lvzi(k)
           jj=lvzj(k)
           kl=kk-1
           kh=kk+1
           jh=jj+1
           i1=ii-1
           i2=i1-1
           d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
           d2vx=d2vx+cof2*values(ii,jh,kk,3)-cof1*values(ii,jj,kk,3)
           d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(i1,jj,kk,3)
 1103 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C        Case 12
C         One neighbor (+1) in y and two (-1,+1) in x direction
      do 1203 k=ztc(11)+1,ztc(12)

           kk=lvzk(k)
           ii=lvzi(k)
           jj=lvzj(k)
           kl=kk-1
           kh=kk+1
           jh=jj+1
           il=ii-1
           ih=ii+1
           d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
           d2vx=d2vx+cof2*values(ii,jh,kk,3)-cof1*values(ii,jj,kk,3)
           d2vx=d2vx+values(il,jj,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ih,jj,kk,3)
 1203 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C        Case 13
C         One neighbor (-1) in y and none in x direction
```

```
      do 1303 k=ztc(12)+1,ztc(13)

            kk=lvzk(k)
            ii=lvzi(k)
            jj=lvzj(k)
            kl=kk-1
            kh=kk+1
            jl=jj-1
            d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
            d2vx=d2vx+cof2*values(ii,jl,kk,3)-cof1*values(ii,jj,kk,3)
            d2vx=d2vx-8.*values(ii,jj,kk,3)
 1303 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 14
C       One neighbor (-1) in y and one (+1) in x direction
      do 1403 k=ztc(13)+1,ztc(14)

            kk=lvzk(k)
            ii=lvzi(k)
            jj=lvzj(k)
            kl=kk-1
            kh=kk+1
            jl=jj-1
            ih=ii+1
            d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
            d2vx=d2vx+cof2*values(ii,jl,kk,3)-cof1*values(ii,jj,kk,3)
            d2vx=d2vx+cof2*values(ih,jj,kk,3)-cof1*values(ii,jj,kk,3)
 1403 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 15
C       One neighbor (-1) in y and one (-1) in x direction
      do 1503 k=ztc(14)+1,ztc(15)

            kk=lvzk(k)
            ii=lvzi(k)
            jj=lvzj(k)
            kl=kk-1
            kh=kk+1
            jl=jj-1
            il=ii-1
            d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
```

```
          d2vx=d2vx+cof2*values(ii,jl,kk,3)-cof1*values(ii,jj,kk,3)
          d2vx=d2vx+cof2*values(il,jj,kk,3)-cof1*values(ii,jj,kk,3)
 1503 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 16
C       One neighbor (-1) in y and two (+1,+2) in x direction
      do 1603 k=ztc(15)+1,ztc(16)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          jl=jj-1
          i1=ii+1
          i2=i1+1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx+cof2*values(ii,jl,kk,3)-cof1*values(ii,jj,kk,3)
          d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(i1,jj,kk,3)
 1603 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 17
C       One neighbor (-1) in y and two (-1,-2) in x direction
      do 1703 k=ztc(16)+1,ztc(17)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          jl=jj-1
          i1=ii-1
          i2=i1-1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx+cof2*values(ii,jl,kk,3)-cof1*values(ii,jj,kk,3)
          d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(i1,jj,kk,3)
 1703 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 18
```

```
C       One neighbor (-1) in y and two (-1,+1) in x direction
      do 1803 k=ztc(17)+1,ztc(18)

            kk=lvzk(k)
            ii=lvzi(k)
            jj=lvzj(k)
            kl=kk-1
            kh=kk+1
            jl=jj-1
            il=ii-1
            ih=ii+1
            d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
            d2vx=d2vx+cof2*values(ii,jl,kk,3)-cof1*values(ii,jj,kk,3)
            d2vx=d2vx+values(il,jj,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ih,jj,kk,3)
 1803 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 19
C       Two neighbors (+1,+2) in y and none in x direction
      do 1903 k=ztc(18)+1,ztc(19)

            kk=lvzk(k)
            ii=lvzi(k)
            jj=lvzj(k)
            kl=kk-1
            kh=kk+1
            j1=jj+1
            j2=j1+1
            d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
            d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(ii,j1,kk,3)
            d2vx=d2vx-8.*values(ii,jj,kk,3)
 1903 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 20
C       Two neighbors (+1,+2) in y and one (+1) in x direction
      do 2003 k=ztc(19)+1,ztc(20)

            kk=lvzk(k)
            ii=lvzi(k)
            jj=lvzj(k)
            kl=kk-1
```

```
          kh=kk+1
          ih=ii+1
          j1=jj+1
          j2=j1+1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(ii,j1,kk,3)
          d2vx=d2vx+cof2*values(ih,jj,kk,3)-cof1*values(ii,jj,kk,3)
 2003 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 21
C       Two neighbors (+1,+2) in y and one (-1) in x direction
      do 2103 k=ztc(20)+1,ztc(21)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          il=ii-1
          j1=jj+1
          j2=j1+1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(ii,j1,kk,3)
          d2vx=d2vx+cof2*values(il,jj,kk,3)-cof1*values(ii,jj,kk,3)
 2103 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 22
C       Two neighbors (+1,+2) in y and two (+1,+2) in x direction
      do 2203 k=ztc(21)+1,ztc(22)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          i1=ii+1
          i2=i1+1
          j1=jj+1
          j2=j1+1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
```

```
     &values(ii,jj,kh,3)
           d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(ii,j1,kk,3)
           d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(i1,jj,kk,3)
  2203 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 23
C       Two neighbors (+1,+2) in y and two (-1,-2) in x direction
      do 2303 k=ztc(22)+1,ztc(23)

           kk=lvzk(k)
           ii=lvzi(k)
           jj=lvzj(k)
           kl=kk-1
           kh=kk+1
           i1=ii-1
           i2=i1-1
           j1=jj+1
           j2=j1+1
           d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
           d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(ii,j1,kk,3)
           d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(i1,jj,kk,3)
  2303 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 24
C       Two neighbors (+1,+2) in y and two (-1,+1) in x direction
      do 2403 k=ztc(23)+1,ztc(24)

           kk=lvzk(k)
           ii=lvzi(k)
           jj=lvzj(k)
           kl=kk-1
           kh=kk+1
           il=ii-1
           ih=ii+1
           j1=jj+1
           j2=j1+1
           d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
           d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
```

```
     &2.*values(ii,j1,kk,3)
           d2vx=d2vx+values(il,jj,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ih,jj,kk,3)
  2403 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 25
C       Two neighbors (-1,-2) in y and none in x direction
      do 2503 k=ztc(24)+1,ztc(25)

           kk=lvzk(k)
           ii=lvzi(k)
           jj=lvzj(k)
           kl=kk-1
           kh=kk+1
           j1=jj-1
           j2=j1-1
           d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
           d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(ii,j1,kk,3)
           d2vx=d2vx-8.*values(ii,jj,kk,3)
  2503 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 26
C       Two neighbors (-1,-2) in y and one (+1) in x direction
      do 2603 k=ztc(25)+1,ztc(26)

           kk=lvzk(k)
           ii=lvzi(k)
           jj=lvzj(k)
           kl=kk-1
           kh=kk+1
           ih=ii+1
           j1=jj-1
           j2=j1-1
           d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
           d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(ii,j1,kk,3)
           d2vx=d2vx+cof2*values(ih,jj,kk,3)-cof1*values(ii,jj,kk,3)
  2603 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 27
```

```
C       Two neighbors (-1,-2) in y and one (-1) in x direction
      do 2703 k=ztc(26)+1,ztc(27)

            kk=lvzk(k)
            ii=lvzi(k)
            jj=lvzj(k)
            kl=kk-1
            kh=kk+1
            il=ii-1
            j1=jj-1
            j2=j1-1
            d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
            d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(ii,j1,kk,3)
            d2vx=d2vx+cof2*values(il,jj,kk,3)-cof1*values(ii,jj,kk,3)
 2703 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 28
C       Two neighbors (-1,-2) in y and two (+1,+2) in x direction
      do 2803 k=ztc(27)+1,ztc(28)

            kk=lvzk(k)
            ii=lvzi(k)
            jj=lvzj(k)
            kl=kk-1
            kh=kk+1
            i1=ii+1
            i2=i1+1
            j1=jj-1
            j2=j1-1
            d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
            d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(ii,j1,kk,3)
            d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(i1,jj,kk,3)
      values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)
 2803 continue

C       Case 29
C       Two neighbors (-1,-2) in y and two (-1,-2) in x direction
      do 2903 k=ztc(28)+1,ztc(29)
```

```
          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          i1=ii-1
          i2=i1-1
          j1=jj-1
          j2=j1-1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(ii,j1,kk,3)
          d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(i1,jj,kk,3)
 2903 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 30
C       Two neighbors (-1,-2) in y and two (-1,+1) in x direction
      do 3003 k=ztc(29)+1,ztc(30)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          il=ii-1
          ih=ii+1
          j1=jj-1
          j2=j1-1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx-.2*values(ii,j2,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(ii,j1,kk,3)
          d2vx=d2vx+values(il,jj,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ih,jj,kk,3)
 3003 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 31
C       Two neighbors (-1,+1) in y and none in x direction
      do 3103 k=ztc(30)+1,ztc(31)

          kk=lvzk(k)
          ii=lvzi(k)
```

```
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          jl=jj-1
          jh=jj+1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx+values(ii,jl,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jh,kk,3)
          d2vx=d2vx-8.*values(ii,jj,kk,3)
 3103 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C        Case 32
C        Two neighbors (-1,+1) in y and one (+1) in x direction
      do 3203 k=ztc(31)+1,ztc(32)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          ih=ii+1
          jl=jj-1
          jh=jj+1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx+values(ii,jl,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jh,kk,3)
          d2vx=d2vx+cof2*values(ih,jj,kk,3)-cof1*values(ii,jj,kk,3)
 3203 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C        Case 33
C        Two neighbors (-1,+1) in y and one (-1) in x direction
      do 3303 k=ztc(32)+1,ztc(33)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          il=ii-1
          jl=jj-1
          jh=jj+1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
```

```
     &values(ii,jj,kh,3)
           d2vx=d2vx+values(ii,jl,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jh,kk,3)
           d2vx=d2vx+cof2*values(il,jj,kk,3)-cof1*values(ii,jj,kk,3)
    3303 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 34
C       Two neighbors (-1,+1) in y and two (+1,+2) in x direction
        do 3403 k=ztc(33)+1,ztc(34)

           kk=lvzk(k)
           ii=lvzi(k)
           jj=lvzj(k)
           kl=kk-1
           kh=kk+1
           i1=ii+1
           i2=i1+1
           jl=jj-1
           jh=jj+1
           d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
           d2vx=d2vx+values(ii,jl,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jh,kk,3)
           d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(i1,jj,kk,3)
    3403 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 35
C       Two neighbors (-1,+1) in y and two (-1,-2) in x direction
        do 3503 k=ztc(34)+1,ztc(35)

           kk=lvzk(k)
           ii=lvzi(k)
           jj=lvzj(k)
           kl=kk-1
           kh=kk+1
           i1=ii-1
           i2=i1-1
           jl=jj-1
           jh=jj+1
           d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
           d2vx=d2vx+values(ii,jl,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jh,kk,3)
```

```
          d2vx=d2vx-.2*values(i2,jj,kk,3)-5.*values(ii,jj,kk,3)+
     &2.*values(i1,jj,kk,3)
 3503 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

C       Case 36
C       All neighbors in y and x directions
      do 3603 k=ztc(35)+1,ztc(36)

          kk=lvzk(k)
          ii=lvzi(k)
          jj=lvzj(k)
          kl=kk-1
          kh=kk+1
          jl=jj-1
          jh=jj+1
          il=ii-1
          ih=ii+1
          d2vx=values(ii,jj,kl,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jj,kh,3)
          d2vx=d2vx+values(ii,jl,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ii,jh,kk,3)
          d2vx=d2vx+values(il,jj,kk,3)-2.*values(ii,jj,kk,3)+
     &values(ih,jj,kk,3)
 3603 values(ii,jj,kk,3)=values(ii,jj,kk,3)+dt*
     &(-values(ii,jj,kk,4)+values(ii,jj,kl,4)+d2vx*rre)

      return
      end
```

newperbc.f

```
      subroutine perbc(elemid)
      integer elemid
      include 'bcfeed256'

      isize=ix-4
      irght=isize+2
C     routine to update boundary elements of matrix

      do 101 k=1,2
      do 101 j=1,iy
      do 101 i=1,iz
      i1=i
      j1=j
      k1=isize+k
      if(i1.lt.3) i1=isize+i1
      if(j1.lt.3) j1=isize+j1
      if(i1.gt.irght) i1=i1-isize
      if(j1.gt.irght) j1=j1-isize
  101 values(i,j,k,elemid)=values(i1,j1,k1,elemid)

      do 201 k=ix-1,ix
      do 201 j=1,iy
      do 201 i=1,iz
      i1=i
      j1=j
      k1=k-isize
      if(i1.lt.3) i1=isize+i1
      if(j1.lt.3) j1=isize+j1
      if(i1.gt.irght) i1=i1-isize
      if(j1.gt.irght) j1=j1-isize
  201 values(i,j,k,elemid)=values(i1,j1,k1,elemid)

      do 301 k=3,iz-2
      do 301 j=1,2
      do 301 i=1,ix
      i1=i
      j1=j+isize
      if(i1.lt.3) i1=isize+i1
      if(i1.gt.irght) i1=i1-isize
  301 values(i,j,k,elemid)=values(i1,j1,k,elemid)

      do 401 k=3,iz-2
      do 401 j=iy-1,iy
```

```
   do 401 i=1,ix
   i1=i
   j1=j-isize
   if(i1.lt.3) i1=isize+i1
   if(i1.gt.irght) i1=i1-isize
401 values(i,j,k,elemid)=values(i1,j1,k,elemid)

   do 501 k=3,iz-2
   do 501 j=3,iy-2
   do 501 i=1,2
   i1=i+isize
501 values(i,j,k,elemid)=values(i1,j,k,elemid)

   do 601 k=3,iz-2
   do 601 j=3,iy-2
   do 601 i=ix-1,ix
   i1=i-isize
601 values(i,j,k,elemid)=values(i1,j,k,elemid)

   return
   end
```

perbcall.f

```
      subroutine perbcxyz
      include 'bcfeed256'

      isize=ix-4
      irght=isize+2
C     routine to update boundary elements of matrix

      do 101 k=1,2
      do 101 j=1,iy
      do 101 i=1,iz
      i1=i
      j1=j
      k1=isize+k
      if(i1.lt.3) i1=isize+i1
      if(j1.lt.3) j1=isize+j1
      if(i1.gt.irght) i1=i1-isize
      if(j1.gt.irght) j1=j1-isize
      values(i,j,k,3)=values(i1,j1,k1,3)
      values(i,j,k,2)=values(i1,j1,k1,2)
  101 values(i,j,k,1)=values(i1,j1,k1,1)

      do 201 k=ix-1,ix
      do 201 j=1,iy
      do 201 i=1,iz
      i1=i
      j1=j
      k1=k-isize
      if(i1.lt.3) i1=isize+i1
      if(j1.lt.3) j1=isize+j1
      if(i1.gt.irght) i1=i1-isize
      if(j1.gt.irght) j1=j1-isize
      values(i,j,k,3)=values(i1,j1,k1,3)
      values(i,j,k,2)=values(i1,j1,k1,2)
  201 values(i,j,k,1)=values(i1,j1,k1,1)

      do 301 k=3,iz-2
      do 301 j=1,2
      do 301 i=1,ix
      i1=i
      j1=j+isize
      if(i1.lt.3) i1=isize+i1
      if(i1.gt.irght) i1=i1-isize
```

```
   values(i,j,k,3)=values(i1,j1,k,3)
   values(i,j,k,2)=values(i1,j1,k,2)
301 values(i,j,k,1)=values(i1,j1,k,1)

   do 401 k=3,iz-2
   do 401 j=iy-1,iy
   do 401 i=1,ix
   i1=i
   j1=j-isize
   if(i1.lt.3) i1=isize+i1
   if(i1.gt.irght) i1=i1-isize
   values(i,j,k,3)=values(i1,j1,k,3)
   values(i,j,k,2)=values(i1,j1,k,2)
401 values(i,j,k,1)=values(i1,j1,k,1)

   do 501 k=3,iz-2
   do 501 j=3,iy-2
   do 501 i=1,2
   i1=i+isize
   values(i,j,k,3)=values(i1,j,k,3)
   values(i,j,k,2)=values(i1,j,k,2)
501 values(i,j,k,1)=values(i1,j,k,1)

   do 601 k=3,iz-2
   do 601 j=3,iy-2
   do 601 i=ix-1,ix
   i1=i-isize
   values(i,j,k,3)=values(i1,j,k,3)
   values(i,j,k,2)=values(i1,j,k,2)
601 values(i,j,k,1)=values(i1,j,k,1)

   return
   end
```

fields.f

```
      subroutine fields
      real pa,pb,pc,pd,va,vb,vc,vd
      include 'bcfeed256'

C     Routine to print out average x-velocity components and pressures
C     at four different depths in the 3-D microstructure
      area=1./float(iyyy-2)/float(izzz-2)
      va=0.
      vb=0.
      vc=0.
      vd=0.
      pc=0.
      pd=0.
      pa=0.0
      pb=0.0
      ix1=5
      ix2=ixxx/4
      ix3=ixxx/2
      ix4=ixxx-2
      do 251 j=3,iyyy
      do 251 k=3,izzz
      va=va+values(ix1,j,k,1)
      vb=vb+values(ix2,j,k,1)
      pa=pa+values(ix1,j,k,4)
      pb=pb+values(ix2,j,k,4)
      vc=vc+values(ix3,j,k,1)
      pc=pc+values(ix3,j,k,4)
      pd=pd+values(ix4,j,k,4)
  251 vd=vd+values(ix4,j,k,1)

      va=va*area
      vb=vb*area
      pa=pa*area
      pb=pb*area
      vc=vc*area
      vd=vd*area
      pc=pc*area
      pd=pd*area
      open(99,file='perm256')
      open(98,file='press256')
      write(99,114)va,vb,vc,vd
      write(98,114)pa,pb,pc,pd
  114 format(4(f16.9,1x))
```

```
call flush(99)
call flush(98)
return
end
```