2011

Reduced Order Model of a Spouted Fluidized Bed Utilizing Proper Orthogonal Decomposition

Stephanie R. Beck-Roth
West Virginia University

Reduced Order Model of a Spouted Fluidized Bed Utilizing Proper Orthogonal Decomposition

Stephanie R. Beck-Roth

Dissertation submitted to the
Eberly College of Arts and Sciences
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Mathematics

Mary Ann Clarke, Ph.D. (Chair)
Ian Christie, Ph.D.
Gary Ganser, Ph.D.
Adam Halasz, Ph.D.
Sreekanth Pannala, Ph.D.*

Department of Mathematics
West Virginia University

***Oak Ridge National Laboratory**
Oak Ridge, Tennessee

Morgantown, West Virginia
2011

Keywords: MFIX, Navier-Stokes, Proper Orthogonal Decomposition, Reduced Order Modeling, Spouted Bed

ABSTRACT

Reduced Order Model of a Spouted Fluidized Bed Utilizing Proper Orthogonal Decomposition

Stephanie R. Beck-Roth

A reduced order model utilizing proper orthogonal decomposition for approximation of gas and solids velocities as well as pressure, solids granular temperature and gas void fraction for use in multiphase incompressible fluidized beds is developed and presented. The methodology is then tested on data representing a flat-bottom spouted fluidized bed and comparative results against the software Multiphase Flow with Interphase eXchanges (MFIX) are provided. The governing equations for the model development are based upon those implemented in the (MFIX) software. The three reduced order models explored are projective, extrapolative and interpolative. The first is an extension of the system solution beyond an original time sequence. The second is a numerical approximation to a new solution based on a small selected parameter deviation from an existing CFD data set. Finally an interpolative methodology approximates a solution between two existing CFD data sets both which vary a single parameter.

Acknowledgements

This research was supported in part by an appointment to the National Energy Technology Laboratory Research Participation Program, sponsored by the U.S. Department of Energy and administered by the Oak Ridge Institute for Science and Education (ORISE). The many efforts of Steve Giessler, Professional Technologist, at West Virginia University currently serving as WVU Math Department cluster administrator are greatly appreciated. This work was improved through conversations with various members of the Multiphase Flow Research Group at the National Energy and Technology Laboratory in Morgantown, WV including but not limited to Chris Guenther, Mehrdad Shahnam, Sofiane Benyahia and Rahul Garg in addition to Sreekanth Pannala at Oak Ridge National Lab in Oak Ridge, TN.

Dedication

This work is dedicated to my family, in particular, my son Aaron and my daughter Clarissa. It has become a reality only through their sacrifice, support and encouragement.

Table of Contents

NOMENCLATURE

MFIX	Multiphase Flow with Interphase eXchanges
CFD	Computational Fluid Dynamics
POD	Proper Orthogonal Decomposition
ROM	Reduced Order Model
ODEs	Ordinary Differential Equations
PCA	Principal Components Analysis
SVD	Singular Value Decomposition
PSD	Power Spectral Density
cgs	(cm, g, s)
ε_g	Volume fraction of the fluid/gas phase (void fraction)
ε_s	Volume fraction of the solid phase
ρ_g	Density of the fluid/gas phase; g/cm^3
ρ_s	Density of the solid phase; g/cm^3
P_g	Pressure of the fluid/gas phase; Pa
P_s	Pressure of the solid phase; Pa
τ_g	Fluid/gas phase stress tensor; Pa
τ_s	Solid phase stress tensor; Pa
G	Gravity; cm/s^2
I_{gs}	Coefficient of momentum interphase exchange

δ_{ij}	Dirac Delta Function
μ_{gt}	Turbulent viscosity of the fluid/gas phase; g/cm·s
E	Coefficient of restitution for the collisions of the solid particles
η	Function of the coefficient of restitution
μ_s	Solids viscosity; g/(cm·s)
g_{0s}	Radial distribution function at contact
Θ_s	Granular temperature of the solid particles; cm^2/s^2
π	Pi was approximated to 3.14159265
d_p	Average diameter of the solid particles, cm
μ_{max}	Maximum value of the turbulent viscosity of the fluid phase; g/(cm·s)
μ_g	Molecular viscosity of the fluid phase; g/(cm·s)
μ_e	Eddy viscosity of the fluid phase; g/(cm·s)
l_s	Turbulence length-scale parameter; cm
$R_{\#}$	Reynolds number
φ_z	Basis function
K	Temporal covariance matrix (n x n)
A	Computational mesh matrix (N x n)
σ_i	Singular value (i^{th} largest)
λ_i	Eigenvalue (i^{th} largest)
NGU	Number of x-component gas velocity POD bases
NGV	Number of y-component gas velocity POD bases
NPU	Number of x-component particle velocity POD bases
NPV	Number of y-component particle velocity POD bases

NGP	Number of gas pressure POD bases
NPP	Number of particle pressure POD bases
NGF	Number of gas void fraction POD bases
NPT	Number of particle temperature POD bases
$U_g(\vec{x},t)$	POD representation for x-component gas velocity
$\mu_{U_g}(\vec{x})$	Time-averaged x-component gas velocity
$a_{U_{g_k}}(t)$	POD coefficient (k^{th}) for x-component of the gas velocity
$\varphi_{U_{g_k}}(\vec{x})$	POD basis function (k^{th}) for x-component of the gas velocity
$V_g(\vec{x},t)$	POD representation for y-component gas velocity
$\mu_{V_g}(\vec{x})$	Time-averaged y-component gas velocity
$a_{V_{g_k}}(t)$	POD coefficient (k^{th}) for y-component of the gas velocity
$\varphi_{V_{g_k}}(\vec{x})$	POD basis function (k^{th}) for y-component of the gas velocity
$U_s(\vec{x},t)$	POD representation for x-component particle velocity
$\mu_{U_s}(\vec{x})$	Time-averaged x-component particle velocity
$a_{U_{s_k}}(t)$	POD coefficient (k^{th}) for x-component of the particle velocity
$\varphi_{U_{s_k}}(\vec{x})$	POD basis function (k^{th}) for x-component of the particle velocity
$V_s(\vec{x},t)$	POD representation for y-component particle velocity
$\mu_{V_s}(\vec{x})$	Time-averaged y-component particle velocity
$a_{V_{s_k}}(t)$	POD coefficient (k^{th}) for y-component of the particle velocity
$\varphi_{V_{s_k}}(\vec{x})$	POD basis function (k^{th}) for y-component of the particle velocity
$P_g(\vec{x},t)$	POD representation for gas pressure
$\mu_{P_g}(\vec{x})$	Time-averaged gas pressure

$a_{P_{g_k}}(t)$	POD coefficient (k^{th}) for gas pressure
$\varphi_{P_{g_k}}(\vec{x})$	POD basis function (k^{th}) for gas pressure
$P_s(\vec{x},t)$	POD representation for particle pressure
$\mu_{P_s}(\vec{x})$	Time-averaged particle pressure
$a_{P_{s_k}}(t)$	POD coefficient (k^{th}) for particle pressure
$\varphi_{P_{s_k}}(\vec{x})$	POD basis function (k^{th}) for particle pressure
$\varepsilon_g(\vec{x},t)$	POD representation for gas void fraction
$\mu_{\varepsilon_g}(\vec{x})$	Time-averaged gas void fraction
$a_{\varepsilon_{g_k}}(t)$	POD coefficient (k^{th}) for gas void fraction
$\varphi_{\varepsilon_{g_k}}(\vec{x})$	POD basis function (k^{th}) for gas void fraction
$\theta_s(\vec{x},t)$	POD representation for particle temperature
$\mu_{\theta_s}(\vec{x})$	Time-averaged particle temperature
$a_{\theta_{s_k}}(t)$	POD coefficient (k^{th}) for particle temperature
$\varphi_{\theta_{s_k}}(\vec{x})$	POD basis function (k^{th}) for particle temperature
$\psi_w(\vec{x})$	Piecewise basis function

LIST OF TABLES AND FIGURES

1.0 Introduction

The US Department of Energy sponsors many research projects that share an overarching purpose to identify ways to preserve non-renewable energy resources by maximizing the amount of energy released when they are consumed, whether it be coal, oil or natural gas. In the context of this dissertation, computational methodology based upon proper orthogonal decomposition is developed to create a reduced order model of a multiphase fluidized bed system.

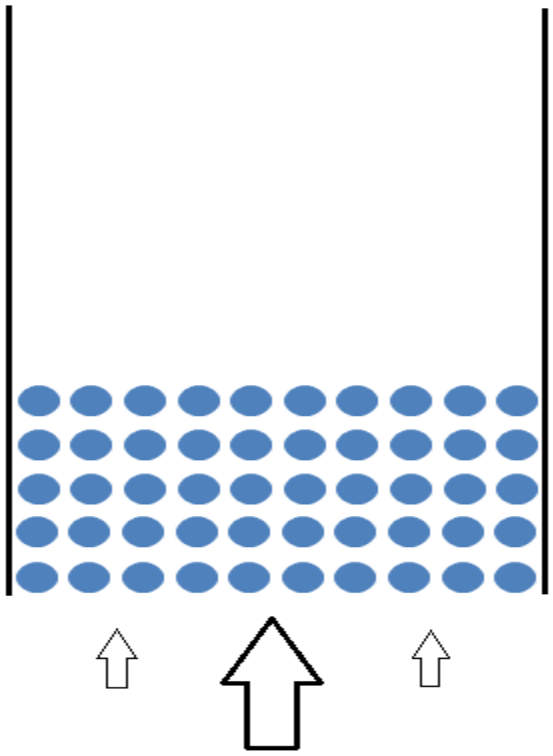
Many large scale applications utilizing highly accurate CFD models can take an extensive amount of time during the planning and design phase of power plant control algorithms. This is due in part to the heavy computational burden imposed by the CFD solvers as the equations governing the physical processes occurring in multiphase fluidized bed systems are complex and highly nonlinear. Reduced order models or ROMs can be used to reduce the design space of the possible combinations of the parameters of interest during simulations of physical processes as a sufficiently small sacrifice in accuracy will result in reduced computation time. This permits saving time early in the design phase and ultimately allows for more effort to be focused in the refinement of the physical process model via the more accurate full order model CFD solvers. In other words, the results of this dissertation will help to develop and monitor actual energy systems that utilize spouted fluidized beds, like coal reactors, to influence the optimization of coal burning plants that create electricity.

Mathematically, the computational model presented in this work was derived through an evaluation of the Navier Stokes momentum equations used to describe the flow fields of a fluidized bed. Eight selected variables (gas void fraction, solids granular temperature, both gas and solids pressure in addition to the two dimensional components of gas and solid velocities) found in the Navier Stokes conservation of momentum equations are individually represented by a collection of orthogonal basis functions. This variable specific collection of basis functions is summative and controls the expression of dominant spatial features at specific points in time as deviations from the observed mean behavior of the variable in a similarly posed known data set of a given fluidized bed. Thus, each variable's representation can be viewed as a combination of two functions: one that represents only spatial characteristics and another that represents only temporal variation.

Computational Fluid Dynamics (CFD) may employ several numerical methodologies to project flow fields of fluids and particles. Some of these methods use finite element, finite difference, or hybrid algorithms to describe the equations that govern a flow field. As with all computer models there is often an inverse relationship between the accuracy of the model and the computational run time necessary to calculate solutions. Those models that yield more accurate results often take orders of magnitude longer to identify stable solutions. Some researchers have shown that a numerical reduction to a CFD model is possible by identifying particular spatial and temporal components that capture a majority of the overall energy of any physical system. [1], [2], [3], [4], [5], [6] Ideally, by focusing only on principal energetic components that identify data correlations one can preserve solution accuracy and reduce the amount of computation time required to obtain results. This was accomplished in this work by utilizing proper orthogonal decomposition (POD). The aim of this POD methodology is to identify and subsequently generate the minimum number, *m,* of time specific eigenvalues for each variable in the Navier Stokes form that contribute to a minimum of 99% of the total representative *matrix energy* of an understood and previously modeled physical system. In particular, researchers can then use this technique and a variable's reduced order representation of a well understood fluidized bed system to make quick assessments of the same fluidized bed operating under sufficiently similar conditions.

In this research, there is select interest in the nature of the CFD model for a spouted fluidized bed, but the presented methodology is suitable for any 2D circulating fluidized bed. A spouted fluidized bed is any apparatus in which an opening is provided for the inflow of a gas or liquid at a sufficient velocity into a region usually containing particles. Spouted fluidized beds have a variety of uses from the coating of tablets to the combustion and gasification of coal. [7] The described numerical system is a two dimensional representation of a flat bottom spouted fluidized bed. Air is introduced to the particle bed from the bottom of the computational domain. As seen in Figure 1, a single central port provides high velocity gas flow (big arrow) which is supported by background airflow (small arrow) introduced everywhere else along the bottom edge of the spouted bed.

Figure 1: **Flat Bottom Spouted Bed**



Specifically discussed in relevant sections the research presented in this document is guided by previous work found in the literature in that it formulates a reduced order representation of a multiphase fluidized bed system through use of orthogonal basis functions obtained through proper orthogonal decomposition. This project's distinguishing and novel characteristic is the number and type of variables being represented through the orthogonal basis functions derived from a proper orthogonal decomposition of a training dataset provided by an existing computational fluid dynamic solver for use in model development of sufficiently similar physical processes. In particular, this research models void fraction of the gas in a cell as well as gas pressure using a reduced order representation. As such, this lends to a more technically correct approach to modeling flat bottom spouted fluidized beds than what is found in the literature. Also, the proposed approach deviates from traditional reduced order models of

fluidized beds using proper orthogonal decomposition in that such a representation of these variables results in a system of nonlinear equations. Traditionally, a linear system of equations is identified and the solution process is straightforward using matrix solvers such as LU-decomposition. A literature overview is presented on a section by section basis so that the reader may better understand the impact and relevance of each core methodology as it is being presented.

The material in this document is presented in eight chapters covering seven major topical areas. This introduction aims to provide a broad overview of the material found in the following chapters as well as the work's industrial applications. Chapter Two discusses the generalities of modeling gas and solid particles in a multiphase system. Chapter Three contains a literature overview of existing computational fluid dynamic solving methodologies. This is followed by a discussion of the governing equations used for development of the reduced order models presented in this work and concludes with a discussion of their weak solutions. Chapter Four encompasses on a per section basis the methodological areas covered in this work such as proper orthogonal decomposition, reduced order modeling, and the Levenberg-Marquardt nonlinear solver incorporated in this research. Background of each methodological component is provided at the start of each section and the relevance with regards to this research follows. Chapter Five contains a section on the comparsion of this works reduced order model using proper orthogonal decomposition with its corresponding full order model followed by three sections highlighting the results of the projective, extrapolative and interpolatory reduced order models. Chapters Six, Seven and Eight are devoted to conclusions drawn, future work and the work's bibliography, respectively. Source code and supplementary equations along with the partial derivatives comprising the entries of the Jacobian matrix required by the nonlinear solver are provided in the appendices which follow the main text of this document.

2.0 BACKGROUND

Given the current concerns of energy conservation, time and consideration are devoted to the identification of physical processes which may optimize the consumption of natural resources, like coal. Such identification is often done via use of computer simulations which mimic the actual fluidization process of coal particles in a particular type of fluidized bed. These computer simulations are then used to identify those physical processes which are most efficient and are used again to determine the appropriateness of the material handling procedures and mechanical designs of coal gasifiers utilized in power plants. Ultimately, the value of computer simulations is in a savings of both money and time.

The modeling of a mixture containing solid particles and fluid (or gas) within any fluidized bed system, and in particular a spouted fluidized bed, is treated in one of two manners. (1:) Either the behavior of the solid particles are viewed over individual elements and tracked via their respective positions or (2:) the solid particles are treated as a continuum. In the second case, particle behavior is modeled with a similar methodology to fluids. Instead of treating each particle individually, the equations that govern solid particle movements treat the particles as a unified collection of objects. Given the large number of particles and the computational complexity of the governing equations, the continuum description of particles is the more appropriate treatment of the behavior of solid flow fields when exploring a means to identify a reduced order model.

This document will address the development and accuracy of several types of reduced order models (ROMs) utilizing proper orthogonal decomposition (POD) of mixed element flows of a system of particles and a fluid (or gas) in a flat bottom spouted fluidized bed.

3.0 LITERATURE REVIEW OF CFD METHODOLOGIES

The material in this section provides an overview of the most common numerical techniques for depicting fluid/solid dynamics in physical systems. The study of the flow of only a single type of gas or fluid is commonly referred to as *single-phase flow.* When solid particles of potentially more than one type and/or, different gases or fluids are added to single-phase flow, the system is recast as *multi-phase flow.*

Computational dynamic methodologies that are employed for approximating numerical solutions of the fluidization process use both single-phase as well as multi-phase flow models depending upon the physical process occurring within the system. As such, the computational description of the flow may fall into one of three categories: finite volume, finite difference, or finite element. A brief overview of these methodologies is given below.

Finite volume methodologies partition the domain of interest into many different regions, frequently utilizing combinations of regular geometric shapes (e.g. squares, triangles). Each region (or control volume) is called a cell or element. Particular variables of interest such as velocity or pressure are calculated at the centroid of each cell. The differential form of any governing equations (such as those found in Equations 3.1.1 and 3.1.2) are integrated over the cell. This allows for interpolation of a particular variable at the centroid of each cell. One very important characteristic of finite volume methodologies is that they preserve the conservation laws of momentum, mass, energy and species in every cell and subsequently throughout an entire computational domain. [8]

Finite difference methodologies were introduced first by Euler in the 18th century. Implementation through a discretization of the computational domain, typically some kind of a grid is performed. Estimates of a solution are obtained at grid intersections or "nodes". Since the governing equations considered hold for any point in the domain, then they must hold at the nodes. Finite difference methodologies are based on the limit definition of a derivative

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \qquad (3.0.1)$$

whereby *h* has some fixed value instead of approaching 0. Since every continuous differentiable function is expressible as a Taylor series expansion about each point in its domain, in particular such a function will have a Taylor series expansion around every node in a discretized domain. Examining the Taylor series expansion at every node and dropping terms of higher order from the expansion allows for the formation of a linear system of equations where there is exactly one equation per node. Variations of finite difference methodologies relate to the particular frame of reference in the expansion about each node. Forward-difference schemes formulate differential relationship with successive nodes; backward-difference schemes form these relationships with the preceding nodes while central-difference schemes form their relationships with both preceding and successive nodes. [9] Finite difference methods are often very fast, as they require only simple arithmetic calculations to approximate solutions, but are often inconvenient to implement in an irregularly shaped domain.

Finite element methodology has its roots in the early 1940s and can be traced to work involving Alexander Hrennikoff and Richard Courant. [10] Its development came from the need to obtain solutions to complex structural problems involving elasticity and structural analysis in the fields of civil and aeronautical engineering. [11] Like finite difference methodology, finite element methodology requires that a continuous domain is represented as a finite collection of sub-domains, creating a mesh. However, in finite element methodology, governing equations are translated into a system of differential equations (DEs) and the numerical solution of those governing equations is obtained through approximations of the solution of these DEs. This is done through an empirical integration of the ODEs by use of numerical quadrature techniques such as Runge-Kutta or Euler's methods.

There have been numerous examinations of solid-fluid flows in fluidized beds using finite volume, finite difference and finite element methodologies to discern various characteristics and features of the system. Finite volume methodologies are implemented on a variety of systems such as flow around a cylinder [12], or multiphase flow in a circulating fluidized bed [13]. Burkardt [14] discusses treatment of the Navier Stokes equations using a finite element approach. Spouted fluidized beds were also examined in the literature. He et. al [15] examined the behavior of a solid-fluid system in a .91m cylinder with particles ranging from 3.3 to 6.7cm using finite differences. Pannala et. al. [16] examined the dynamics of spouted beds

as a means to coat nuclear fuel through use of open-source finite volume CFD software. Franca et. al. [17] discuss the effects of airflow on the drying process in spouted bed dryers. An examination of spouting particle velocity and minimum spouting flow rates is discussed in Duarte et.al. [18] For their analysis they use Fluent software which incorporates finite volume methodology in its CFD solver.

Although CFD methodologies are analytically similar for single and multi-phase systems, algorithms for implementing them do vary. Part of the reason for this is that the interfaces between various phases in each cell must be tracked. This additional algorithmic requirement is quite involved depending upon the interface tracking methodology chosen. Some of the most common interface tracking methods include volume of fluid methods, level-set methods, and front tracking methods. Volume of fluid and level set methodologies involve defining the interface line between the various phases on a fixed grid while front tracking methodologies impose an additional layer of complexity by tracking the motion of the interface explicitly. [19]

3.1 GOVERNING EQUATION DEFINITIONS

The equations of continuity that characterize this problem are the conservation of momentum equations of Navier Stokes. Cast in the volume of fluid formulation, these equations include approximations of pressure and velocity for both gas and solid phases in addition to the granular temperature of the solids and volume fraction of fluids present in any given computational cell. The numerical model is assumed to be divergence free indicating the assumption of incompressible fluid use only.

For purposes of this research, the assumption of a two dimensional slice extracted from a uniform three dimensional flat bottom spouted fluidized bed to represent the physical domain is made. The governing equations have been recast to include only one particle type of uniform size.

The primary software for validation of the proposed reduced order model is Multiphase Flow with Interface eXchanges (MFIX). This software, freely available from the National Energy and Technology Laboratory, is used to develop a comparative model from which the POD basis functions of the predictive models are obtained. The formulation of the governing equations and the variables used to represent each term in those equations is in large part directly comparable to that which is found in MFIX Equations documentation [20], derived in cgs (cm, g, s). MFIX documentation references to multiple solid phases have been recast with the subscript of "s" to indicate "single" or "solid" particle. When convenient, equations have been simplified. The governing equations also utilize Einstein summative notation [21] treating i and j as the first principal (x) and second principal (y) direction respectively, with several noted exceptions.

The equations which govern the flow of fluids and solids in multiphase fluidized beds are the Navier Stokes Equations. In particular, the gas and solid momentum equations govern the divergence free velocity of the gas (or fluid) and solid particles as they move throughout a fluid/solid system.

The gas velocity motion is governed by the Gas Momentum equation [20]:

$$\frac{\partial}{\partial t}(\varepsilon_g \rho_g U_{gi}) + \frac{\partial}{\partial x_j}(\varepsilon_g \rho_g U_{gj} U_{gi}) = -\varepsilon_g \left(\frac{\partial P_g}{\partial x_i}\right) + \frac{\partial \tau_{gij}}{\partial x_j} + \varepsilon_g \rho_g G_i - I_{gsi} \quad (3.1.1)$$

The solid continuum velocity motion is governed by the following Solids Momentum equation [20]:

$$\frac{\partial}{\partial t}(\varepsilon_s \rho_s U_{si}) + \frac{\partial}{\partial x_j}(\varepsilon_s \rho_s U_{sj} U_{si}) = -\varepsilon_s \left(\frac{\partial P_g}{\partial x_i}\right) + \frac{\partial \tau_{sij}}{\partial x_j} + \varepsilon_s \rho_s G_i + I_{gsi} \quad (3.1.2)$$

The subscripts g and s denote components belonging to gas and solids respectively. The volume fractions are represented by ε while the density of each phase is represented by ρ. The two dimensional velocity vectors are represented by U = <U_i, U_j> while pressure, a scalar, is represented by P. The stress of the phase is represented by τ while gravity is represented by G. The momentum of the interphase exchange, I, is the transfer of momentum to the solid phase from the gas phase. It incorporates information from the solid as well as gas phase components. Because of its nature its quantity is subtracted in Equation 3.1.1 (i.e. from the gas) and added in Equation 3.1.2 (i.e. to the solids).

The stress term, τ, is defined below for both the solid and gas phase where δ_{ij} is the Dirac delta function. [20]

$$\tau_{gij} = \mu_{gt}\left[\left(\frac{\partial U_{gi}}{\partial x_j} + \frac{\partial U_{gj}}{\partial x_i}\right) - \frac{2}{3}\frac{\partial U_{gi}}{\partial x_i}\delta_{ij}\right] \quad (3.1.3)$$

$$\tau_{sij} = \left(-P_s + \eta\mu_b\left(\frac{\partial U_{si}}{\partial x_i}\right)\delta_{ij} + 2\mu_s\left[\frac{1}{2}\left(\frac{\partial U_{si}}{\partial x_j} + \frac{\partial U_{sj}}{\partial x_i}\right) - \frac{1}{3}\left(\frac{\partial U_{si}}{\partial x_i}\right)\right]\right. \quad (3.1.4)$$

Here η represents a function of restitution. E in particular, is the restitution coefficient (See Equation 3.1.5), which is the ratio of relative speed of an object after collision divided by relative speed of the object before collision. [20]

$$\eta = \frac{1}{2}(1 + E) \qquad (3.1.5)$$

The values of μ_{gt} and μ_s are the gas and solids viscosity, respectively, with other μ subscripted variables representing individual components of viscosity as defined empirically below. [20]

$$\mu_s = \left(\frac{2 + \alpha}{3}\right)\left(\left(\frac{\mu_s{}^*}{g_{0s}\,\eta\,(2 - \eta)}\right)\left(1 + \frac{8}{5}\eta\varepsilon_s g_{0s}\right)\left(1 + \frac{8}{5}\eta\varepsilon_s g_{0s}(3\eta - 2)\right) + \frac{3}{5}\eta\mu_b\right) \qquad (3.1.6)$$

where

$$\mu_s{}^* = \frac{\varepsilon_s\rho_s g_{0s}\mu\Theta_s}{\varepsilon_s\rho_s g_{0s}\Theta_s + \left(\frac{2\beta_{gs}\mu}{\varepsilon_s\rho_s}\right)}, \qquad (3.1.7)$$

$$\mu = \frac{5}{96}\rho_s d_p\sqrt{\pi\Theta_s}\,, \qquad (3.1.8)$$

$$\mu_b = \frac{256}{5\pi}\mu\varepsilon_s{}^2 g_{0s}\,, \qquad (3.1.9)$$

$$\mu_{gt} = \min(\mu_{max}, \mu_g + \mu_e), \qquad (3.1.10)$$

and

$$\mu_e = 2l_s{}^2\varepsilon_g\rho_g\sqrt{\frac{1}{2}\left[\left(\frac{U_{g,x}}{\partial x} - \frac{U_{g,y}}{\partial y}\right) + \left(\frac{U_{g,x}}{\partial y} + \frac{U_{g,y}}{\partial x}\right)\right]}\,. \qquad (3.1.11)$$

The above equations are empirical relations used in MFIX and come directly from MFIX documentation. [20] Note that equation (3.1.8) contains the term d_p which is used to denote the average diameter of a particle, and equation (3.1.11) contains l_s which is the turbulence length scale parameter. The granular temperature of the solids phase is represented by Θ_s and is likewise defined empirically in MFIX as given below. [20]

$$\Theta_s = [(d_p\sqrt{\pi})\left(\frac{-2(1+e)\varepsilon_s\rho_s g_{0s}\left(\frac{U_{g,x}}{\partial x}+\frac{U_{g,y}}{\partial y}\right)}{24(1-E^2)\varepsilon_s\rho_s g_{0s}}\right.$$

$$\left.+\frac{\sqrt{\mathrm{K}_1+\frac{48(1-E^2)\varepsilon_s\rho_s g_{0s}}{d_p\sqrt{\pi}}(\mathrm{K}_2+\mathrm{K}_4)}}{24(1-E^2)\varepsilon_s\rho_s g_{0s}}\right)]^2 \quad (3.1.12)$$

$$\mathrm{K}_1 = (2(1+E)\varepsilon_s\rho_s g_{0s})^2\left(\frac{U_{g,x}}{\partial x}+\frac{U_{g,y}}{\partial y}\right)^2, \quad (3.1.13)$$

$$\mathrm{K}_2 = \left(\frac{4d_p\varepsilon_s\rho_s g_{0s}}{3\sqrt{\pi}}-\frac{2}{3}\mathrm{K}_3\right)\left(\frac{U_{g,x}}{\partial x}+\frac{U_{g,y}}{\partial y}\right)^2, \quad (3.1.14)$$

$$\mathrm{K}_3 = \frac{d_p\rho_s}{2}\left(\frac{\sqrt{\pi}\left(\frac{1}{2}(3E+1)+\frac{2}{5}(1+E)(3E-1)\varepsilon_s g_{0s}\right)}{3(3-E)}\right. \quad (3.1.15)$$

$$\left.+\frac{8\varepsilon_s g_{0s}(1+E)}{5\sqrt{\pi}}\right),$$

and

$$\mathrm{K}_4 = 2\mathrm{K}_3\left(\frac{U_{g,x}}{\partial y}+\frac{U_{g,y}}{\partial x}\right)\left(\frac{U_{g,x}}{\partial y}+\frac{U_{g,y}}{\partial x}\right). \quad (3.1.16)$$

Both gas and solid phase equations include the term, I_{gs}, a transfer of momentum between the phases at their interface, and it is defined as follows: [20]

$$I_{gs} = \beta_{gs}(U_g - U_s). \quad (3.1.17)$$

The coefficient for the interface force between the gas and solid phase, β_{gs} , is defined using the findings of Syamlal and O'Brien and is given below. [20]

$$\beta_{gs} = \frac{3\varepsilon_s\varepsilon_g\rho_g}{4V_{rm}d_p}\left(0.63+4.8\sqrt{\frac{V_{rm}}{R_\#}}\right)^2|U_g - U_s| \quad (3.1.18)$$

The Reynold's number of the particle is referred to as $R_\#$ and is given by

$$R_{\#} = \frac{d_p \rho_s |U_g - U_s|}{\mu_g}. \qquad (3.1.19)$$

Likewise, the definition of the interface force coefficient also includes the term V_{rm} given by

$$V_{rm} = \frac{1}{2}\left({\varepsilon_g}^{4.14} - 0.06 R_{\#} + \sqrt{(0.06 R_{\#})^2 + 0.12 R_{\#}\left(2B - {\varepsilon_g}^{4.14}\right) + {\varepsilon_g}^{17.396}} \right) \qquad (3.1.20)$$

where the piecewise function B is defined below. [20]

$$B = \begin{cases} 0.8{\varepsilon_g}^{1.28}; & if\ \varepsilon_g \leq 0.85 \\ {\varepsilon_g}^{2.65}; & if\ \varepsilon_g > 0.85 \end{cases}. \qquad (3.1.21)$$

Together, Equations 3.1.1 through 3.1.21 define the full order computational model for multiphase fluidized beds as implemented in MFIX. It is from a data set containing the full order model solution of a particular spouted fluidized bed in MFIX that a ROM will be constructed through the use of proper orthogonal decomposition.

3.2 WEAK SOLUTIONS

A classical or "strong" solution satisfies Equations 3.1.1 and 3.1.2 for every ordered pair (x,y) in a given two-dimensional domain. Since the solution of these equations holds for every point in that domain, the classical solution will still hold if a product of these equations with any chosen basis function denoted as φ_z below is taken. This classical solution continues to hold if said product is integrated over the problem's domain (Equations 3.2.1 and 3.2.2). Integration by parts allows for the transfer of some derivatives to the basis function, φ_z , provided that it is differentiable. The momentum equations in the form of Equations 3.2.1 and 3.2.2. are referred to as "weak." As an analytical solution to the Navier-Stokes equations remains at present unknown, this format permits the consideration of a larger class of solutions than what was originally possible. This larger class of solutions is comprised of those solutions where the basis function, φ_z , is defined as any differentiable function. Any solution from this larger class is a "weak solution" since it satisfies the weak formulation of the governing equations. [14] This formulation allows for the identification of piecewise solutions whereby each cell in the two-dimensional domain has its own "piece" of the solution. Thus, weak solutions are then sought on a per cell basis and the dynamics of the physical process occurring outside of the cell are irrelevant to the solution within each cell.

For this application, a weak solution of gas momentum is obtained by multiplying Equation 3.1.1 by φ_z and integrating over the two-dimensional domain, $D_x D_y$, yielding the following equation.

$$\iint_{D_x D_y} \frac{\partial}{\partial t}(\varepsilon_g \rho_g)\begin{bmatrix} U_{xg} \\ U_{yg} \end{bmatrix} \varphi_z \, dy \, dx + \int_{D_x} \varepsilon_g \rho_g U_{yg} \begin{bmatrix} U_{xg} \\ U_{yg} \end{bmatrix} \varphi_z \, dx - \iint_{D_x D_y} (\varepsilon_g \rho_g U_{yg}) \begin{bmatrix} U_{xg} \\ U_{yg} \end{bmatrix} \left(\frac{\partial}{\partial y}\varphi_z\right) dy \, dx + \int_{D_y} \varepsilon_g \rho_g U_{xg} \begin{bmatrix} U_{xg} \\ U_{yg} \end{bmatrix} \varphi_z \, dy$$

$$
\begin{aligned}
&-\iint_{D_y D_x} \left(\varepsilon_g \rho_g U_{xg}\right) \begin{bmatrix} U_{xg} \\ U_{yg} \end{bmatrix} \left(\frac{\partial}{\partial x}\varphi_z\right) dx\, dy \\
&= \iint_{D_x D_y} \left(-\varepsilon_g \begin{bmatrix} \frac{\partial}{\partial x} P_g \\ \frac{\partial}{\partial y} P_g \end{bmatrix} \varphi_z \right) dy\, dx \qquad (3.2.1) \\
&+ \iint_{D_x D_y} \varphi_z \begin{bmatrix} \frac{\partial}{\partial x} \tau_{gij} \\ \frac{\partial}{\partial y} \tau_{gij} \end{bmatrix} dy\, dx \\
&+ \iint_{D_x D_y} \varepsilon_g \rho_g \varphi_z \begin{bmatrix} 0.0 \\ -980.0 \end{bmatrix} dy\, dx \\
&- \iint_{D_x D_y} \beta_{gs} \left(\begin{bmatrix} U_{xg} \\ U_{yg} \end{bmatrix} - \begin{bmatrix} U_{xs} \\ U_{ys} \end{bmatrix} \right) \varphi_z\, dy\, dx
\end{aligned}
$$

Similarly, a weak solution for solid particle momentum is obtained and given next.

$$
\begin{aligned}
&\iint_{D_x D_y} \frac{\partial}{\partial t}(\varepsilon_s \rho_s) \begin{bmatrix} U_{xs} \\ U_{ys} \end{bmatrix} \varphi_z \, dy \, dx \\
&\quad + \int_{D_x} \varepsilon_s \rho_s U_{ys} \begin{bmatrix} U_{xs} \\ U_{ys} \end{bmatrix} \varphi_z \, dx \\
&\quad - \iint_{D_x D_y} (\varepsilon_s \rho_s U_{ys}) \begin{bmatrix} U_{xs} \\ U_{ys} \end{bmatrix} \left(\frac{\partial}{\partial y} \varphi_z \right) dy \, dx \\
&\quad + \int_{D_y} \varepsilon_s \rho_s U_{xs} \begin{bmatrix} U_{xs} \\ U_{ys} \end{bmatrix} \varphi_z \, dy \\
&\quad - \iint_{D_y D_x} (\varepsilon_s \rho_s U_{xs}) \begin{bmatrix} U_{xs} \\ U_{ys} \end{bmatrix} \left(\frac{\partial}{\partial x} \varphi_z \right) dx \, dy \\
&\quad = \iint_{D_x D_y} \left(-\varepsilon_s \begin{bmatrix} \frac{\partial}{\partial x} P_g \\ \frac{\partial}{\partial y} P_g \end{bmatrix} \varphi_z \right) dy \, dx \qquad (3.2.2) \\
&\quad + \iint_{D_x D_y} \varphi_z \begin{bmatrix} \frac{\partial}{\partial x} \tau_{sij} \\ \frac{\partial}{\partial y} \tau_{sij} \end{bmatrix} dy \, dx \\
&\quad + \iint_{D_x D_y} \varepsilon_s \rho_s \varphi_z \begin{bmatrix} 0.0 \\ -980.0 \end{bmatrix} dy \, dx \\
&\quad + \iint_{D_x D_y} \beta_{gs} \left(\begin{bmatrix} U_{xg} \\ U_{yg} \end{bmatrix} - \begin{bmatrix} U_{xs} \\ U_{ys} \end{bmatrix} \right) \varphi_z \, dy \, dx
\end{aligned}
$$

4.0 METHODOLOGY

This chapter on methodology is divided into five distinct sections. First in Section 4.1, a discussion of the Multiphase Flow Interphase eXchange (MFIX) program from which the full order model training data set originates is given. Then, the concept of POD and how it relates to the development of the POD reduced variable representation is conveyed in Section 4.2. Later, a discussion of the development of a novel methodology for obtaining three distinct types of ROMs and the solving methodologies employed based upon the POD algorithmic design for estimation of solutions under certain conditions is reviewed in Section 4.3. Subsequently in Section 4.4, a discussion of non-linear solvers is included. This is followed by a concluding discussion of the requirements for conforming to current computer aided process environment (CAPE-Open) standards in Section 4.5. This brief discussion of CAPE standards is included in this section as it directly influences some of the analytic and programming techniques implemented as well as methodologies which have been incorporated into this work.

The governing equations of the POD based ROMs are discussed in Section 4.3.1. Though there is an inherent relationship between the development of extrapolation and interpolation methodologies, they are discussed separately in Sections 4.3.2 and 4.3.3, respectively. The development of specific methodologies to use the POD based ROM to extend a solution to future time points is discussed in Section 4.3.4.

The first step in the development of a ROM is setting up the general framework of the physical system in question. Next, a numerical data sample of such a system is used to facilitate the development of the parameters necessary to reduce the dimension of the original model. Following this, an examination of the constructed POD variable representations is conducted to ensure that the required spatial characteristics of the training data set are captured (discussed in Section 5.1). Once a reduced order model of the physical system in question is identified (each discussed separately in Sections 5.2-5.4), the next step involves using it to approximate a solution of the same physical system operating under similar conditions. If the ROM agrees with some measure of tolerance to the original model of the physical system, then an exploration of the agreement of the reduced order model to a variety of similarly conditioned problems is conducted. The focus of this dissertation is to perform only the first three components of this

process and discuss the development of the methodologies used to formulate the equations in which the projective, extrapolatory and interpolatory methodologies are based.

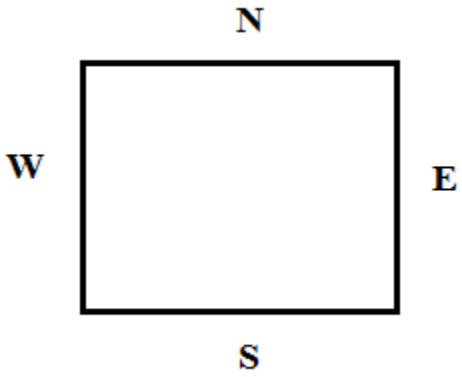
4.1 MFIX

Multiphase Flow Interphase eXchange (MFIX) is a software package developed and distributed by the National Energy Technology Laboratory (NETL). [20] Its general purpose is to provide detailed data describing chemical reactions, hydrodynamic flow and heat transfer in multiphase fluid-solids systems. [20] This software was applied successfully to geophysical-volcanological systems, aerospace and automotive applications. [22] The package's governing equations are those highlighted in Section 3.1 and are based on accepted forms from the field of computational fluid dynamics. It is for this reason the particular choices of Equations 3.1.3-3.1.21 were made. The best predictive ROM methodologies inherently should observe and operate under the same choice of Equations 3.1.1 and 3.1.2. By utilizing the same governing equations on which the training dataset is based, any bias away from an optimal solution strategy is minimized.

All peripheral conditions associated with a ROM come from a training dataset. In the case of this work, initial conditions, boundary conditions and other parameters for the ROM are imported from the file mfix.dat (the input parameter file for MFIX) and the respective .SP? files (where ? takes on the numbers 1-9 as well as A and B which contain the output variable data including gas and solid velocity components as well as pressure, solids temperature and void fraction) for each relevant variable. For example, the mesh for the ROM is identical to the MFIX mesh devoid of any ghost cells.

One important deviation from MFIX is noted. MFIX reports data at staggered grid points for velocity (x-components centered along the east and west edge and y-components along the north and south edge of each cell) while data for void fractions along with pressure and temperature are provided at the center of each cell. In order to maintain consistency throughout the models, it was necessary to convert all edge data for the velocities to the center of each. This was performed through linear interpolation.

Figure 2: **MFIX cell labeling**



4.2 PROPER ORTHOGONAL DECOMPOSITION

Proper Orthogonal Decomposition (POD), Karhunen-Loève Transform, Hotelling Transform, Principal Components Analysis (PCA) and Singular Value Decomposition (SVD) [23] are just a few of the given names that are used to describe the process of identifying a subset of uncorrelated data from a much larger set of somewhat correlated data with the aim of reducing the dimensionality of a problem. [2] This POD methodology has been incorporated in the fields of seismology, signal processing, image recognition and for modeling of geophysical data including atmospheric phenomena.

POD methodology is a useful tool in data analysis when the objective is to extract low dimensional approximate descriptions of high dimensional processes. [23] It is for this reason that POD analysis is increasingly popular in CFD modeling. Recently POD was used to quantify the magnitude of error in mesh design for modeling ocean gyre [24] as well as a method for reconstruction of missing data in subsonic airfoil and cylinder wake flow [25].

The computational model described in this dissertation was derived through an evaluation of the Navier Stokes conservation of momentum equations used to describe the flow fields of a fluidized bed. In the context of the proposed problem, the aim is to identify the minimum number of *m* time specific eigenvalues which contribute to at minimum 99% of the total energy of a particular spouted fluidized bed CFD simulation. A particular variable *F* is thought of as a function that varies across space and time, say $F(\vec{x}, t)$. An approximation of *F* is given in terms of a representation of spatial basis functions combined with temporal coefficients. Such a representation is given by

$$F(\vec{x}, t) \approx \mu_F(\vec{x}) + \sum_{k=1}^{m} a_k(t)\varphi_k(\vec{x}) \qquad (4.2.1)$$

where $\mu_F(\vec{x})$ denotes the mean value of F over time spanning the domain, and $\varphi_k(\vec{x})$ represents a set of spatial basis functions. POD basis coefficients or "modes," $a_k(t)$, are calculated for the time periods in the base model and vary only across time not space. This representation is similar to that given by Burkardt [26] with the exception that the POD representation here is

given as a deviation from the mean similar to the standard basis notation and representation in PCA and the other methodologies listed above.

A is an N x n matrix where N denotes the number of cells in the computational mesh that defines the physical domain and n the number of observed temporal *snapshots* from the same simulation; a snapshot is a collection of data that represents all variables in every spatial cell at a given point in time. Each *i*th column of A represents the vector denoted by $F(\vec{x},t) - \mu_F(\vec{x})$. Then, K is defined as the n x n temporal covariance matrix:

$$K = \frac{1}{n} A^T A \qquad (4.2.2)$$

where A^T denotes the transpose of A. [26]

The singular value decomposition (SVD) of A is considered as

$$A = U\Sigma V^T \qquad (4.2.3)$$

where U and V are orthogonal matrices, and Σ is a rectangular pseudo-diagonal matrix. The singular values of A are represented by σ_i where *i* denotes the *i*th singular value of A. Suppose λ_i is used to denote the *i*th largest eigenvalue of the matrix K, then the following relationship holds between matrices A and K (proof in Appendix 2)

$$\sigma_i{}^2 = n\lambda_i. \qquad (4.2.4)$$

By definition [26], the POD basis vectors are the first n left singular vectors of A given as the first n columns of U above.

The matrix energy of a matrix K, M(K), was defined by Gutman in 1978 [27] as the sum of the absolute value of the eigenvalues of the matrix, K, given as:

$$M(K) = \sum_{k=1}^{n} |\lambda_k| \,. \qquad (4.2.5)$$

Burkhardt [26] notes that the reduction in dimensionality is obtained by controlling the relative error, ε, of the reduced *m*-dimensional POD subspace. This is accomplished by selecting the smallest integer *m* satisfying the following requirement for a given ε, so that there is a capture of (1.0 - ε) matrix energy for the model:

$$\frac{\sum_{k=1}^{m} \sigma_k{}^2}{\sum_{k=1}^{n} \sigma_k{}^2} = \frac{\sum_{k=1}^{m} |\lambda_k|}{\sum_{k=1}^{n} |\lambda_k|} \geq 1.0 - \varepsilon. \qquad (4.2.6)$$

The process of obtaining POD basis functions is performed independently for each component of the gas and solids velocity vectors, gas and solids pressure, gas void fraction and solids temperature. NGU denotes the number of x-component gas velocity POD bases; NGV denotes the number of y-component gas velocity POD bases; NPU denotes the number of x-component particle velocity POD bases; NPV denotes the number of y-component particle velocity POD bases; NGP denotes the number of gas pressure POD bases; NPP denotes the number of particle pressure POD bases; NGF denotes the number of gas void fraction POD bases, and finally NPT denotes the number of particle temperature POD basis functions. The related POD coefficients are obtained through the SVD representation. For example, consider the formulation of the x-component gas velocity matrix A, the first NGU left singular vectors of A, found in the orthogonal columns of U constitute the set of orthogonal POD basis vectors. The set of NGU POD modes are found by multiplying the first NGU largest singular values by the first NGU rows of V. Note that each column of V corresponds to each successive time step of the data set. The representation for each of the approximated solutions is given below.

$$U_g(\vec{x},t) \approx \mu_{U_g}(\vec{x}) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(\vec{x}) \qquad (4.2.7)$$

$$V_g(\vec{x},t) \approx \mu_{V_g}(\vec{x}) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(\vec{x}) \qquad (4.2.8)$$

$$U_s(\vec{x},t) \approx \mu_{U_s}(\vec{x}) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(\vec{x}) \qquad (4.2.9)$$

$$V_s(\vec{x},t) \approx \mu_{V_s}(\vec{x}) + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(\vec{x}) \qquad (4.2.10)$$

$$P_g(\vec{x},t) \approx \mu_{P_g}(\vec{x}) + \sum_{k=1}^{NGP} a_{P_{g_k}}(t)\varphi_{P_{g_k}}(\vec{x}) \qquad (4.2.11)$$

$$P_s(\vec{x},t) \approx \mu_{P_s}(\vec{x}) + \sum_{k=1}^{NPP} a_{P_{s_k}}(t)\varphi_{P_{s_k}}(\vec{x}) \qquad (4.2.12)$$

$$\varepsilon_g(\vec{x},t) \approx \mu_{\varepsilon_g}(\vec{x}) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(\vec{x}) \qquad (4.2.13)$$

$$\theta_s(\vec{x},t) \approx \mu_{\theta_s}(\vec{x}) + \sum_{k=1}^{NPT} a_{\theta_{sk}}(t)\varphi_{\theta_{sk}}(\vec{x}) \qquad (4.2.14)$$

For predictive models, the POD basis functions obtained from these calculations are used to estimate a solution for a similarly posed system. These solution estimates are based solely on the general behavior of the system already understood and exhibited in pre-generated sample data. Any serious deviations from the original physical system may (and likely will) result in inaccurate conclusions. This is due in large part to the nature of POD reduced order modeling, in that a small perturbation in a input parameter of a highly nonlinear system will fundamentally change the spatial behavior of the system which then has the potential to impact the POD basis representation nontrivially. Provided a physical system solution with a sufficiently small change in some parameter from the original sample data system on which the POD basis was obtained is desired, then this methodology is appropriately interpreted.

4.3 REDUCED ORDER MODELING

The methodology for producing the ROM is divided primarily into 5 sections. Following in subsequent paragraphs of this section, a review of work previously done in this field related to the problem and its process is fully developed. The first step in the modeling process presented in Subsection 4.3.1 is the development of the governing equations using the POD basis functions in addition to the calculated time averages (while considering the POD modes as unknowns) from a set of sample data. In Subsection 4.3.2, a method to extrapolate a ROM solution for a sufficiently similar flat bottom spouted bed system is developed. Also in Subsection 4.3.3, a method for interpolation of a ROM solution between two flat bottom spouted bed systems with a variation in a single closely related parameter is presented. Finally, in Subsection 4.3.4 a discussion of the developments of methodologies under which the ROM solution is extended to later time points is given.

The motivation behind the development of any reduced order model is to identify key physical characteristics of some data set and subsequently reduce the dimensional representation of the data. POD based reduced order models are useful in characterizing the flow features in both single phase and multiphase systems. In typical multiphase system ROMs, gas/solids void fraction and pressure values are estimated outside of the POD representation and solution process. Exploring gas/solids void fraction, gas and solids pressure, as well as solids granular temerature through reduced POD variable representations distinguishes this work from that found in literature. A literature overview of relevant POD based reduced order models of both single phase and multiphase systems is presented below.

In single phase systems, properties of the velocity and/or vorticity of the fluid are often studied. Yuan et. al. developed a compressible gas phase only ROM for a fluidized bed. [6] Their ROM is derived from MFIX datasets. In their schema, no corrections to handle the staggered grid datasets are made prior to the POD basis calculations. They begin by reconstructing velocity components and pressure then estimate gas density using the ideal gas law. Small corrections to all variables are made following the POD formulation. Corrections are needed because initial estimates of gas velocity are made using previous time step data. Their corrections comprise a system of linear equations which are solved quickly. Providing solutions

converge, the data is accepted and the time step is advanced. Fang et. al. developed a POD reduced order model based upon finite element methodology to examine ocean flows on irregular coastlines. [28] Such a model is governed by the three dimensional Boussinesq equations. Their model uses a dynamically adaptive mesh, constructs two distinct sets of POD basis functions and in an effort to reduce the computation time of the POD simulation, a time dependent matrix is formulated from a series of time independent matrices instead of calculating a solution across every element in a finite element mesh, at every time step. Fang et. al. also developed a new approach for enhancing the accuracy of POD ROM for their ocean model by including the derivatives of the snapshots as well as those of the basis functions and formulating POD representations which incorporate those derivatives. [29] Suram, McCorkle and Bryden develop a POD based ROM for studying velocity flow in a hydraulic mixing nozzle. [4] Kalb and Deane [30] implemented a Galerkin projection of their POD basis modes and substituted the POD basis representation for fluid velocity flows into the two dimensional Navier-Stokes equations to examine wake flow around a bluff body. In particular, their work is aimed at discerning the changes in wake flow as Reynolds number is increased. They develop an intrinsic stabilization scheme aimed at identifying error of the POD solution estimates at time points not included in the training dataset. They then use this as a means for adjusting initial POD solution estimates by projecting this error onto time based POD eigenfunctions constructed by examining the spatial covariance matrix instead of the traditional temporal one and incorporate this adjustment into the spatial POD basis representation. Burkardt et. al. developed a POD reduced order model for gas phase only flows in a T-cell in addition to a model involving central Voronoi tesselations [1]. They then compare the two ROMs and discovered that they both performed equally well and their implementations are both significantly faster than traditional finite element solving methodologies alone. Aquino et. al. [31] demonstrated the ability to use a POD representation in their finite element schema for solving the Helmholtz equations. Weller et. al. examined fluid flow effects in POD based reduced order models in a two dimensional incompressible laminar fluid flow around a square cylinder in an effort to minimize the error in the POD model representation via calibration [5]. Similar work using Tikhonov regularization was shown to be effective [32] and these methods outperform the calibration technique proposed by Couplet et. al. [33].

Yuan et. al. also had success in constructing a ROM for a multiphase bubbling fluidized bed [6] containing both fluid and particles. In their model, POD reduction is performed only for pressure and velocity components of the gas and solids. Time coefficients from a previous iteration are used to estimate field variables at a current time step from the momentum balance equations. Likewise, densities, viscosities and the gas-solid drag force are calculated. A linear system of correction equations which exist for each variable are solved. Then, a correction factor is applied to the POD variable representation. Finally, a solid's volume correction equation is applied to the solid's void fraction and the gas void fraction is updated. If a solution has converged at this point, data is stored and calculation for the next time step begins. Brenner et. al. [34] explored both the impact of snapshot sampling and the best choice of the autocorrelation matrix for minimization of error in multi-phase velocity flow fields in ROMs utilizing a POD representation schema. They noted that the error in the POD representation decreased with the inclusion of additional snapshots. In particular, during the transient phase increasing the number of snapshots was informative while fewer snapshots were required during the quasi-steady state phase of the circulating fluidized bed system. They also examined the impact of coupling the directional components of velocity into a single POD representation as opposed to treating them separately by direction and attempting no coupling whatsoever. Their results showed accuracy is significantly improved when separate POD representations are formed for the directional velocity components. Cizmas et. al. examined reduced order models of multi-phase flows in fluidized beds. [35] In [36] several acceleration techniques were explored. An algorithm for splitting the database of snapshots into two distinct periods so that the number of POD basis functions required is reduced. Additionally, a strategy for reducing projection frequencies is discussed which involved freezing the system of equations for a set period of calculations over several iterations rather than estimating the solution at each time step.

Palacios et. al. [3] studied spatio-temporal patterns in two dimensional multiphase spouted fluidized beds leading them to the conclusion that even though this type of system exhibits chaotic features, a ROM is an appropriate model for fluid-particle interaction. The construction of POD basis functions and modes in multiphase spouted fluidized beds was conducted by Cizmas et. al. [35] but no projection of the governing Navier-Stokes PDE model onto the POD modes was performed.

As eluded to in the previous section, the POD basis functions represent the key spatial differences in the original dataset while the POD coefficients amplify or reduce the visible effects of the POD basis functions at any point in time. A reasonable assumption is that a data set generated from an analytical model inherently similar to the analytical model on which these basis functions were derived should have approximately similar POD basis functions. It is this assumption on which the methodologies for the ROM model developed in this section is based.

4.3.1 GOVERNING EQUATIONS FOR ROM METHODOLOGIES

The governing equations for these ROMs are derived utilizing the POD concept. The *m*-dimensional approximations of the gas and solid velocities, the gas and solid pressure, the void fraction as well as the solids granular temperature (Equations 4.2.7 - 4.2.14) are substituted into the weak solution of both the gas and solids momentum equations (Equations 3.2.1 and 3.2.2) to obtain an approximation to the two-dimensional flow field data.

Consider a set of piecewise basis functions, $\psi_w(\vec{x})$, for $\vec{x} = < x_1, x_2, \dots, x_i, \dots, x_m >$ which are defined by

$$\psi_w(\vec{x}) = \begin{cases} \varphi_w(x_i); & where\ i = w \\ 0; & otherwise \end{cases} . \qquad (4.3.1.1)$$

Assume a mapping of a vector, $\vec{z}$, exists along the diagonal of a diagonal matrix Z. For example, each *i*th entry in the vector ($z_i \in \vec{z}$) would map to $z_{i,i}$. Given this mapping any multiplication by a basis function defined as above (say $\psi_w(\vec{x})$) results in only nonzero entries when *w* is equal to the *i*th component of $\vec{x}$. Thus there is precisely one non-trivial equation for each cell in the grid.

While there are a number of possible choices of basis functions to use for the formulation of the weak solution per each directional component's momentum equation, the chosen basis function is based on the first respective identified POD basis function corresponding to the largest singular value for each variable. In this notation, $\varphi_{U_{g_1}}(x_i)$, for example, is also equal to the weak solution basis function at a cell by assignment. While any of the basis functions could have been chosen, the POD basis function contributing to the largest amount of matrix energy is used. Specifically, the weak formulation for the x-component of gas velocity will utilize the basis function defined by

$$\psi_w(\vec{x}) = \begin{cases} \varphi_{U_{g_1}}(x_i); & where\ i = w \\ 0; & otherwise \end{cases} . \qquad (4.3.1.2)$$

The weak formulation for the y-component of gas velocity will use the basis function defined as

$$\psi_w(\vec{x}) = \begin{cases} \varphi_{V_{g_1}}(x_i); & where\ i = w \\ 0; & otherwise \end{cases}. \qquad (4.3.1.3)$$

The weak formulation for the x-component and y-component of solid velocity, respectively, will use

$$\psi_w(\vec{x}) = \begin{cases} \varphi_{U_{s_1}}(x_i); & where\ i = w \\ 0; & otherwise \end{cases} \qquad (4.3.1.4)$$

and

$$\psi_w(\vec{x}) = \begin{cases} \varphi_{V_{s_1}}(x_i); & where\ i = w \\ 0; & otherwise \end{cases}. \qquad (4.3.1.5)$$

The resulting four equations (two directions per gas and solid mom one per directional component of gas as well as solids momentum) after substitution of the POD basis approximations for each variable following the mapping to a diagonal matrix yield the primary equations to solve for the predictive models. The weak formulation of each equation is developed utilizing the above basis function definitions, $\psi_w(\vec{x})$, yielding four non-trivial equations (again, one per directional component of gas as well as solids momentum). These equations are derived directly from Equations (3.1.1) and (3.1.2) by first obtaining the weak formulation of the solution and then substituting into the reduced representation of the eight examined variables. As such the interpretation of each parameter and variable presented in Section 3.1 still holds in the reduced representation which now follows.

The x-component of the Gas Momentum equation derived through substitution of the reduced variable representations into the first principal direction of Equation (3.2.1) is as follows:

$$\rho_g \left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i) \right] \left[\frac{\partial}{\partial t} \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i) \right) \right] \varphi_{U_{g_1}}(x_i)\, dx\, dy$$

$$
\begin{aligned}
&+ \rho_g \left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t) \varphi_{\varepsilon_{g_k}}(x_i) \right] \left[\frac{\partial}{\partial t} \left(\mu_{U_g}(x_i) \right. \right. \\
&\qquad \left. \left. + \sum_{k=1}^{NGU} a_{U_{g_k}}(t) \varphi_{U_{g_k}}(x_i) \right) \right] \varphi_{U_{g_1}}(x_i) dx\, dy \\
&+ \left[\frac{\partial}{\partial x} \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t) \varphi_{\varepsilon_{g_k}}(x_i) \right) \right] \rho_g \left[\mu_{U_g}(x_i) \right. \\
&\qquad \left. + \sum_{k=1}^{NGU} a_{U_{g_k}}(t) \varphi_{U_{g_k}}(x_i) \right]^2 \varphi_{U_{g_1}}(x_i) dx\, dy \\
&+ \left[\frac{\partial}{\partial x} \left(\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t) \varphi_{U_{g_k}}(x_i) \right) \right] \left[\mu_{U_g}(x_i) \right. \\
&\qquad \left. + \sum_{k=1}^{NGU} a_{U_{g_k}}(t) \varphi_{U_{g_k}}(x_i) \right] \left[\mu_{\varepsilon_g}(x_i) \right. \\
&\qquad \left. + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t) \varphi_{\varepsilon_{g_k}}(x_i) \right] \rho_g \varphi_{U_{g_1}}(x_i) dx\, dy \\
&+ \rho_g \left[\mu_{U_g}(x_i) \right. \\
&\qquad \left. + \sum_{k=1}^{NGU} a_{U_{g_k}}(t) \varphi_{U_{g_k}}(x_i) \right] \left[\frac{\partial}{\partial y} \left(\mu_{\varepsilon_g}(x_i) \right. \right. \\
&\qquad \left. \left. + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t) \varphi_{\varepsilon_{g_k}}(x_i) \right) \right] \left[\mu_{V_g}(x_i) \right. \\
&\qquad \left. + \sum_{k=1}^{NGV} a_{V_{g_k}}(t) \varphi_{V_{g_k}}(x_i) \right] \varphi_{U_{g_1}}(x_i) dx\, dy
\end{aligned}
\qquad (4.3.1.6)
$$

$$+\left[\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\rho_g\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)dx\,dy$$

$$+\left[\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\mu_{P_g}(x_i)+\sum_{k=1}^{NGP}a_{P_{g_k}}(t)\varphi_{P_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)dx\,dy$$

$$-\mu_{gt}\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)dx$$

$$+\mu_{gt}\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_1}}(x_i)\right)\right]dy\,dx$$

$$-\frac{4}{3}\mu_{gt}\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)dy$$

$$+\frac{4}{3}\mu_{gt}\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{g_1}}(x_i)\right)\right]dx\,dy$$

$$-\mu_{gt}\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)dx$$

$$+\mu_{gt}\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_1}}(x_i)\right)\right]dy\,dx$$

$$+\beta_{gs}\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)-\mu_{U_g}(x_i)-\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\varphi_{U_{g_1}}(x_i)dx\,dy=0$$

The y-component of the gas velocity derived through substitution of the reduced variable representations into the second principal direction of Equation (3.2.1) is represented by

$$
\begin{aligned}
&\left[\frac{\partial}{\partial t}\left(\mu_{\varepsilon g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right)\right]\left[\mu_{V g}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NGV} a_{V g_k}(t)\varphi_{V g_k}(x_i)\right]\rho_g\varphi_{V g_1}(x_i)\,dx\,dy\\
&+\left[\frac{\partial}{\partial t}\left(\mu_{V g}(x_i)+\sum_{k=1}^{NGV} a_{V g_k}(t)\varphi_{V g_k}(x_i)\right)\right]\left[\mu_{\varepsilon g}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right]\rho_g\varphi_{V g_1}(x_i)\,dx\,dy\\
&+\left[\frac{\partial}{\partial x}\left(\mu_{\varepsilon g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right)\right]\left[\mu_{U g}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NGU} a_{U g_k}(t)\varphi_{U g_k}(x_i)\right]\left[\mu_{V g}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NGV} a_{V g_k}(t)\varphi_{V g_k}(x_i)\right]\rho_g\varphi_{V g_1}(x_i)\,dx\,dy\\
&+\left[\frac{\partial}{\partial y}\left(\mu_{\varepsilon g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right)\right]\left[\mu_{V g}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NGV} a_{V g_k}(t)\varphi_{V g_k}(x_i)\right]^2\rho_g\varphi_{V g_1}(x_i)\,dx\,dy\\
&+\left[\mu_{\varepsilon g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right]\left[\mu_{U g}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NGU} a_{U g_k}(t)\varphi_{U g_k}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V g}(x_i)\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NGV} a_{V g_k}(t)\varphi_{V g_k}(x_i)\right)\right]\rho_g\varphi_{V g_1}(x_i)\,dx\,dy
\end{aligned}
\qquad (4.3.1.7)
$$

$$-\left[\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\mu_{P_g}(x_i)+\sum_{k=1}^{NGP}a_{P_{g_k}}(t)\varphi_{P_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy$$

$$+980.665\left[\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\rho_g\varphi_{V_{g_1}}(x_i)\,dx\,dy$$

$$-\mu_{\mathrm{gt}}\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx$$

$$+\mu_{\mathrm{gt}}\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]\,dx\,dy$$

$$-\mu_{\mathrm{gt}}\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_1}}(x_i)\right)\right]\,dy$$

$$+\mu_{\mathrm{gt}}\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_1}}(x_i)\right)\right]\,dx\,dy$$

$$-\frac{4}{3}\mu_{\mathrm{gt}}\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]\,dx$$

$$+\frac{4}{3}\mu_{\mathrm{gt}}\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]\,dx\,dy$$

$$+\beta_{gs}\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{sk}}(t)\varphi_{V_{sk}}(x_i)-\mu_{V_g}(x_i)-\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy=0.$$

The x-component for the solids momentum derived through substitution of the reduced variable representations into the first principal direction of Equation (3.2.2) is given by

$$
\begin{aligned}
&-\rho_s\left[\frac{\partial}{\partial t}\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{U_S}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NPU}a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)\right]\varphi_{U_{s_1}}(x_i)dx\,dy\\
&+\rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial t}\left(\mu_{U_S}(x_i)\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NPU}a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy\\
&-\rho_s\left[\frac{\partial}{\partial x}\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{U_S}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NPU}a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)\right]^2\varphi_{U_{s_1}}(x_i)\,dx\,dy\\
&+\rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_S}(x_i)\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NPU}a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)\right)\right]\left[\mu_{U_S}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NPU}a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)\right]\varphi_{U_{s_1}}(x_i)dx\,dy\\
&-\rho_s\left[\frac{\partial}{\partial y}\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{U_S}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NPU}a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)\right]\left[\mu_{V_S}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NPV}a_{V_{sk}}(t)\varphi_{V_{sk}}(x_i)\right]\varphi_{U_{s_1}}(x_i)dx\,dy
\end{aligned}
\qquad (4.3.1.8)
$$

$$+\ \rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_s}(x_i)\right.\right.$$
$$\left.+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\Bigg]\left[\mu_{V_s}(x_i)\right.$$
$$\left.+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\varphi_{U_{s_1}}(x_i)dx\,dy$$
$$+\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\mu_{P_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGP}a_{P_{g_k}}(t)\varphi_{P_{g_k}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy$$
$$+\left[\frac{\partial}{\partial x}\left(\mu_{P_s}(x_i)+\sum_{k=1}^{NPP}a_{P_{s_k}}(t)\varphi_{P_{s_k}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy$$
$$-0.5(1+\mathrm{E})\left[\frac{\partial}{\partial x}\left(\mu_{U_s}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\,(\mu_b)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy$$
$$-0.5(1+\mathrm{E})\left[\frac{\partial^2}{\partial x^2}\left(\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\mu_b\varphi_{U_{s_1}}(x_i)\,dx\,dy$$
$$-\mu_p\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dx$$
$$+\mu_p\left[\frac{\partial}{\partial y}\left(\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dy\,dx$$
$$-\frac{4}{3}\mu_p\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dy$$
$$+\frac{4}{3}\mu_p\left[\frac{\partial}{\partial x}\left(\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dx\,dy$$

$$
\begin{aligned}
&- \mu_p \left[\mu_{U_S}(x_i) + \sum_{k=1}^{NPU} a_{U_{S_k}}(t) \varphi_{U_{S_k}}(x_i) \right] \left[\frac{\partial}{\partial x} \left(\varphi_{U_{S_1}}(x_i) \right) \right] dx \\
&+ \mu_p \left[\frac{\partial}{\partial x} \left(\mu_{V_S}(x_i) + \sum_{k=1}^{NPV} a_{V_{S_k}}(t) \varphi_{V_{S_k}}(x_i) \right) \right] \left[\frac{\partial}{\partial x} \left(\varphi_{U_{S_1}}(x_i) \right) \right] dx \\
&- \beta \left[\mu_{U_S}(x_i) + \sum_{k=1}^{NPU} a_{U_{S_k}}(t) \varphi_{U_{S_k}}(x_i) - \mu_{U_g}(x_i) \right. \\
&\qquad \left. - \sum_{k=1}^{NGU} a_{U_{g_k}}(t) \varphi_{U_{g_k}}(x_i) \right] \left[\frac{\partial}{\partial x} \left(\varphi_{U_{S_1}}(x_i) \right) \right] dx \, dy = 0 \, .
\end{aligned}
$$

The y-component for the solid momentum derived through substitution of the reduced variable representations into the second principal direction of Equation (3.2.2) is given by:

$$
\begin{aligned}
&\rho_s \left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t) \varphi_{\varepsilon_{g_k}}(x_i) \right) \right] \left[\mu_{V_S}(x_i) \right. \\
&\qquad \left. + \sum_{k=1}^{NPV} a_{V_{S_k}}(t) \varphi_{V_{S_k}}(x_i) \right] \varphi_{V_{S_1}}(x_i) \, dx \, dy \\
&+ \rho_s \left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t) \varphi_{\varepsilon_{g_k}}(x_i) \right) \right] \left[\frac{\partial}{\partial t} \left(\mu_{V_S}(x_i) \right. \right. \\
&\qquad \left. \left. + \sum_{k=1}^{NPV} a_{V_{S_k}}(t) \varphi_{V_{S_k}}(x_i) \right) \right] \varphi_{V_{S_1}}(x_i) \, dx \, dy \\
&- \rho_s \left[\frac{\partial}{\partial x} \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t) \varphi_{\varepsilon_{g_k}}(x_i) \right) \right] \left[\mu_{V_S}(x_i) \right. \\
&\qquad \left. + \sum_{k=1}^{NPV} a_{V_{S_k}}(t) \varphi_{V_{S_k}}(x_i) \right] \left[\mu_{U_S}(x_i) \right. \\
&\qquad \left. + \sum_{k=1}^{NPU} a_{U_{S_k}}(t) \varphi_{U_{S_k}}(x_i) \right] \varphi_{V_{S_1}}(x_i) \, dx \, dy
\end{aligned}
$$

$$-\rho_s\left[\frac{\partial}{\partial y}\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]^2\varphi_{V_{s_1}}(x_i)\,dx\,dy$$

$$+\rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}x_i(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy$$

$$+\rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy$$

$$+\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{P_g}(x_i)+\sum_{k=1}^{NGP}a_{P_{g_k}}(t)\varphi_{P_{g_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy$$

$$+980.665\rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy$$

$$+\left[\frac{\partial}{\partial y}\left(\mu_{P_s}(x_i)+\sum_{k=1}^{NPP}a_{P_{s_k}}(t)\varphi_{P_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy$$

$$-0.5(1+E)\left[\frac{\partial}{\partial y}\,(\mu_b)\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy \qquad (4.3.1.9)$$

$$-0.5(1+E)\mu_b\left[\frac{\partial^2}{\partial y^2}\left(\mu_{V_S}(x_i)+\sum_{k=1}^{NPV}a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right)\right]\varphi_{V_{S_1}}(x_i)\,dx\,dy$$

$$-\mu_p\left[\frac{\partial}{\partial x}\left(\mu_{V_S}(x_i)+\sum_{k=1}^{NPV}a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right)\right]\varphi_{V_{S_1}}(x_i)\,dx$$

$$+\mu_p\left[\frac{\partial}{\partial x}\left(\mu_{V_S}(x_i)+\sum_{k=1}^{NPV}a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{S_1}}(x_i)\right)\right]dx\,dy$$

$$-\mu_p\left[\mu_{V_S}(x_i)+\sum_{k=1}^{NPV}a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{S_1}}(x_i)\right)\right]dy$$

$$+\mu_p\left[\frac{\partial}{\partial x}\left(\mu_{V_S}(x_i)+\sum_{k=1}^{NPV}a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{S_1}}(x_i)\right)\right]dx\,dy$$

$$-\frac{4}{3}\mu_p\left[\mu_{V_S}(x_i)+\sum_{k=1}^{NPV}a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{S_1}}(x_i)\right)\right]dx$$

$$+\frac{4}{3}\mu_p\left[\frac{\partial}{\partial y}\left(\mu_{V_S}(x_i)+\sum_{k=1}^{NPV}a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{S_1}}(x_i)\right)\right]dx\,dy$$

$$-\beta_{gs}\left[\mu_{V_S}(x_i)+\sum_{k=1}^{NPV}a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)-\mu_{V_g}(x_i)\right.$$

$$\left.-\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\varphi_{V_{S_1}}(x_i)\,dx\,dy=0.$$

Note that for any given cell ($x_i \in \vec{x}$) the above four equations form the collection of governing equations within that particular cell for the novel reduced order models presented in the next three sections.

4.3.2 EXTRAPOLATION

The development of a ROM which represents the spatial variation of a selected variable in a set of data as a function of time allows a user to predict a solution of a similarly posed problem which has a slight variation in a single parameter. In this particular instance, the POD basis functions used are those directly obtained from the original dataset and the unknown POD coefficients are calculated by designating all of the original model parameters while changing only a *single parameter*.

For the purpose of this dissertation, the coefficient of restitution serves as the suitable parameter of interest to alter. A predictive solution based upon the ROM of the original data set is generated. Note that this parameter change is the only deviation to the original parameter set. The POD basis functions used are derived from the original MFIX dataset. Only the POD coefficients are recalculated to reflect the parameter change. Suitably, this type of ROM requires the new value of the coefficient of restitution to be sufficiently near the original value used in the construction of the original MFIX training data set.

4.3.3 INTERPOLATION

As pointed out in Section 4.3.2, the POD ROM extrapolative methodology is for a parameter change that is sufficiently near the value of the same parameter in the original MFIX data set. Now, through interpolation, a parameter bound is created between two distinct MFIX training data sets from which the POD basis functions are extracted at the bounds. Then two extrapolatory solution estimates are obtained at the desired parameter's value, one from each of the data training sets. A combined solution based upon the average of the extrapolated solutions is created and then its POD basis representation is subsequently derived.

4.3.4 PROJECTIVE SOLUTION

Provided a reasonable representation for the basic variation in an original data set exists and is captured via POD ROM, it is possible to extend a solution for time points near the last approximated time step given in the original data set. The POD basis functions are used in the governing POD equations by solving the nonlinear equations for the unknown POD modes at successive time steps with the same change in time (delta t) that were present in the original POD model. As the new data is predicted, it is expected that the error will increase in a direct relationship with time, but to what extent and how quickly this may occur is less certain.

4.4 NONLINEAR EQUATION SOLVERS

The nonlinear cell equations given in Section 4.3.1 form the governing equations for both the projective and extrapolative ROMs and are the foundational basis of the research presented in this document. Typically, a Galerkin finite element scheme yields a set of linear equations due to the inherent piecewise solution defined across the domain. In such a system, a set of unknown variables for which there exists a single unknown variable per cell and subsequently a single equation or set of equations per cell is solved. In this particular instance, there exist multiple unknown variables per cell, the POD basis coefficients, and these same variables are unknown across all cells. Additionally, these unknown variables interact with each other in every cell in a nonlinear fashion. It is for this reason that a nonlinear equation solver is required for POD ROM.

Of the noted nonlinear system solving methodologies implemented in many of today's most widely used software packages, there are both benefits and shortcomings to each. Many nonlinear system solvers are based upon single nonlinear equation solvers. Perhaps the most popular are those based upon a variation of Newton's method. Newton's method seeks to update previous estimates of a solution by subtracting the product of the inverse of the Jacobian matrix evaluated using the previous iteration's values of the independent variable with its functional evaluation [37]. Broyden's method, which is a generalization of the secant method to nonlinear systems replaces the Jacobian evaluation in the Newton approximation with a finite difference approximation. [38] Another class of nonlinear solvers are those involving a trust region. Non-linear solvers of this class look for solutions which lie within some open neighborhood of a starting solution estimate which is assumed to be near the true solution.

Solving a system of nonlinear equations when there are more equations than there are unknown variables is a minimization problem. In general, nonlinear solvers of a system, $\vec{F} = \vec{0}$, of H equations with G unknowns where H > G require finding the vector $\vec{a}$ with dimension G that minimizes $\vec{F}(\vec{a})$ in the least squares sense using the l_2 norm ($\frac{1}{2}\left\|\vec{F}(\vec{a})\right\|^2$). The three most widely implemented algorithms include Gauss-Newton, trust-region reflective and the

Levenberg-Marquardt algorithm. The Levenberg-Marquardt algorithm is based upon the Gauss-Newton algorithm but incorporates trust region control at each step.

The Gauss-Newton algorithm involves solving a series of linear least squares fits to a nonlinear problem. Its advantage is that it does not require the calculation of second derivatives. [39] The trust-region algorithm is based upon the concept that the "next" solution lies within some region, often an ellipse, of a given search point which is usually taken to be the previous solution estimate. In this algorithm the size of the trust region is adjusted based upon the agreement of test solutions within the region. [40]

4.4.1 LEVENBERG – MARQUARDT SOLVER

A numerical solution of this least squares minimization problem was proposed by Levenberg [41] in 1944 and later in 1963 by Marquardt [42]. These algorithms form the basis of the Levenberg-Marquardt algorithm presented by Moré in 1977 [43]. Of importance, Moré's algorithm incorporates implicitly scaled variables, uses a damped Gauss-Newton method, and has better than linear convergence [44].

The Levenberg-Marquardt algorithm calculates $J^T J$ where J is the Jacobian matrix of the function evaluated at some current approximation of the solution vector, $\vec{a}$. The resulting matrix is equated with J^T evaluated at $\vec{a}$ multiplied by the functional value at $\vec{a}$. The residual vector of the difference between these two components is examined as below in Equation 4.3.1.1.

$$\vec{r} = J^T(\vec{a})J(\vec{a}) - J^T(\vec{a})F(\vec{a}) \qquad (4.3.1.1)$$

The solution vector, $\vec{a}$, will be updated if the residual vector is not within some set measure of tolerance. For the purposes of this work, this tolerance was assumed to be 1.0e-3. (Note that a decrease in tolerance will increase computation time.) The maximum number of iterations allowed for convergence was set to 10,000. The update of the solution vector is made based upon adding mu to the diagonal entries. The value of mu is set to some predetermined tolerance (in this implementation 1.0e-6) times the maximum entry in $J^T J$ and is updated during iterations in the following trust-region fashion. Given $h = \vec{a}_{n+1} - \vec{a}_n$, the calculated step size,

$$step = \frac{\left[\vec{F}(\vec{a}_n) - \vec{F}(\vec{a}_{n+1})\right]}{0.5h^T((mu * h) - \overrightarrow{J^T F}(\vec{a}_n))} \qquad (4.3.1.2)$$

is nonnegative, and mu is updated by

$$mu = \begin{cases} mu * \frac{1}{3}, & 1 - (2 * step - 1)^3 < \frac{1}{3} \\ mu * \left[1 - (2 * step - 1)^3\right], & 1 - (2 * step - 1)^3 \geq \frac{1}{3} \end{cases}. \qquad (4.3.1.3)$$

Otherwise, mu is multiplied by a successive factor of two for every previous iteration in which mu was nonnegative. The LU decomposition of $J^T J$ is formed and the new solution is obtained by back solving for the approximate new solution.

The unknown coefficients which are obtained are the POD basis coefficients. $\vec{a}$ is defined as a vector of these unknown coefficients and is given by

$$\vec{a} = \begin{bmatrix} a_{U_{g_1}} \\ \vdots \\ a_{U_{g_i}} \\ \vdots \\ a_{U_{g_{NGU}}} \\ a_{V_{g_1}} \\ \vdots \\ a_{V_{g_i}} \\ \vdots \\ a_{V_{g_{NGV}}} \\ a_{U_{s_1}} \\ \vdots \\ a_{U_{s_i}} \\ \vdots \\ a_{U_{s_{NPV}}} \\ a_{V_{s_1}} \\ \vdots \\ a_{V_{s_i}} \\ \vdots \\ a_{V_{s_{NPV}}} \\ a_{\varepsilon_{g_1}} \\ \vdots \\ a_{\varepsilon_{g_i}} \\ \vdots \\ a_{\varepsilon_{g_{NGF}}} \\ a_{P_{g_1}} \\ \vdots \\ a_{P_{g_i}} \\ \vdots \\ a_{P_{g_{NPG}}} \\ a_{P_{g_1}} \\ \vdots \\ a_{P_{g_i}} \\ \vdots \\ a_{P_{g_{NPP}}} \\ a_{T_{s_1}} \\ \vdots \\ a_{T_{s_i}} \\ \vdots \\ a_{T_{s_{NPT}}} \end{bmatrix} \qquad (4.3.1.4)$$

with dimension G = NGU + NGV + NPU + NPV + NGF + NGP + NPP + NPT, the number of all POD coefficients. F is defined as

$$\vec{F} = \begin{bmatrix} F_{U_{g_{x_1}}}(\vec{a}) \\ \vdots \\ F_{U_{g_{x_i}}}(\vec{a}) \\ \vdots \\ F_{U_{g_{x_N}}}(\vec{a}) \\ F_{V_{g_{x_1}}}(\vec{a}) \\ \vdots \\ F_{V_{g_{x_i}}}(\vec{a}) \\ \vdots \\ F_{V_{g_{x_N}}}(\vec{a}) \\ F_{U_{s_{x_1}}}(\vec{a}) \\ \vdots \\ F_{U_{s_{x_i}}}(\vec{a}) \\ \vdots \\ F_{U_{s_{x_N}}}(\vec{a}) \\ F_{V_{s_{x_1}}}(\vec{a}) \\ \vdots \\ F_{V_{s_{x_i}}}(\vec{a}) \\ \vdots \\ F_{V_{s_{x_N}}}(\vec{a}) \end{bmatrix} \qquad (4.3.1.5)$$

where $F_{U_{g_{x_i}}}(\vec{a})$ is the POD weak solution of the gas momentum x-component at the *i*th cell. $F_{V_{g_{x_i}}}(\vec{a})$ is the POD weak solution of the gas momentum y-component at the *i*th cell and similarly for the solids x and y components $F_{U_{s_{x_i}}}(\vec{a})$ and $F_{V_{s_{x_i}}}(\vec{a})$. This allows the introduction of the Jacobian of $\vec{F}$, denoted by Jac(F), where

$$Jac(F) = \begin{bmatrix} \frac{\partial}{\partial a_1}(F_{U_{g x_1}}(\vec{a})) & & \frac{\partial}{\partial a_j}(F_{U_{g x_1}}(\vec{a})) & & \frac{\partial}{\partial a_G}(F_{U_{g x_1}}(\vec{a})) \\ \vdots & & \vdots & & \vdots \\ \frac{\partial}{\partial a_1}(F_{U_{g x_i}}(\vec{a})) & & \frac{\partial}{\partial a_j}(F_{U_{g x_i}}(\vec{a})) & & \frac{\partial}{\partial a_G}(F_{U_{g x_i}}(\vec{a})) \\ \vdots & & \vdots & & \vdots \\ \frac{\partial}{\partial a_1}(F_{U_{g x_N}}(\vec{a})) & & \frac{\partial}{\partial a_j}(F_{U_{g x_N}}(\vec{a})) & & \frac{\partial}{\partial a_G}(F_{U_{g x_N}}(\vec{a})) \\ \frac{\partial}{\partial a_1}(F_{V_{g x_1}}(\vec{a})) & \cdots & \frac{\partial}{\partial a_j}(F_{V_{g x_1}}(\vec{a})) & \cdots & \frac{\partial}{\partial a_G}(F_{V_{g x_1}}(\vec{a})) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial}{\partial a_1}(F_{V_{g x_i}}(\vec{a})) & \cdots & \frac{\partial}{\partial a_j}(F_{V_{g x_i}}(\vec{a})) & \cdots & \frac{\partial}{\partial a_G}(F_{V_{g x_i}}(\vec{a})) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial}{\partial a_1}(F_{V_{g x_N}}(\vec{a})) & \cdots & \frac{\partial}{\partial a_j}(F_{V_{g x_N}}(\vec{a})) & \cdots & \frac{\partial}{\partial a_G}(F_{V_{g x_N}}(\vec{a})) \\ \frac{\partial}{\partial a_1}(F_{U_{s x_1}}(\vec{a})) & \vdots & \frac{\partial}{\partial a_j}(F_{U_{s x_1}}(\vec{a})) & \vdots & \frac{\partial}{\partial a_G}(F_{U_{s x_1}}(\vec{a})) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial}{\partial a_1}(F_{U_{s x_i}}(\vec{a})) & \cdots & \frac{\partial}{\partial a_j}(F_{U_{s x_i}}(\vec{a})) & \cdots & \frac{\partial}{\partial a_G}(F_{U_{s x_i}}(\vec{a})) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{\partial}{\partial a_1}(F_{U_{s x_N}}(\vec{a})) & \cdots & \frac{\partial}{\partial a_j}(F_{U_{s x_N}}(\vec{a})) & \cdots & \frac{\partial}{\partial a_G}(F_{U_{s x_N}}(\vec{a})) \\ \frac{\partial}{\partial a_1}(F_{V_{s x_1}}(\vec{a})) & & \frac{\partial}{\partial a_j}(F_{V_{s x_1}}(\vec{a})) & & \frac{\partial}{\partial a_G}(F_{V_{s x_1}}(\vec{a})) \\ \vdots & & \vdots & & \vdots \\ \frac{\partial}{\partial a_1}(F_{V_{s x_i}}(\vec{a})) & & \frac{\partial}{\partial a_j}(F_{V_{s x_i}}(\vec{a})) & & \frac{\partial}{\partial a_G}(F_{V_{s x_i}}(\vec{a})) \\ \vdots & & \vdots & & \vdots \\ \frac{\partial}{\partial a_1}(F_{V_{s x_N}}(\vec{a})) & & \frac{\partial}{\partial a_j}(F_{V_{s x_N}}(\vec{a})) & & \frac{\partial}{\partial a_G}(F_{V_{s x_N}}(\vec{a})) \end{bmatrix} \quad (4.3.1.6)$$

Such a formulation naturally requires the introduction of the partial derivatives of the directional component of the gas and solid momentum equations with respect to each of the POD coefficients. These are given in Appendix 1.

For any given POD variable approximation, the derivative with respect to x is given by

$$\frac{\partial}{\partial x}F(x_i,t) \cong \frac{\partial}{\partial x}[\mu_F(x_i)] + \frac{\partial}{\partial x}\left[\sum_{k=1}^{m} a_k(t)\varphi_k(x_i)\right]$$
$$= \frac{\partial}{\partial x}[\mu_F(x_i)] + \sum_{k=1}^{m} a_k(t)\left[\frac{\partial}{\partial x}\left(\varphi_k(x_i)\right)\right]. \quad (4.3.1.7)$$

The derivative with respect to y of a POD variable approximation is given by

$$\frac{\partial}{\partial y}F(x_i,t) \cong \frac{\partial}{\partial y}[\mu_F(x_i)] + \frac{\partial}{\partial y}\left[\sum_{k=1}^{m} a_k(t)\varphi_k(x_i)\right]$$
$$= \frac{\partial}{\partial y}[\mu_F(x_i)] + \sum_{k=1}^{m} a_k(t)\left[\frac{\partial}{\partial y}\left(\varphi_k(x_i)\right)\right]. \quad (4.3.1.8)$$

The derivative with respect to t of a POD variable approximation varies depending upon the ROM developed. In the event of an extrapolative or interpolative model not involving a change in velocity, the derivative with respect to t used is that from the MFIX data set. Otherwise, it is given by

$$\frac{\partial}{\partial t}F(x_i,t) \cong \frac{\partial}{\partial t}[\mu_F(x_i)] + \frac{\partial}{\partial t}\left[\sum_{k=1}^{m} a_k(t)\varphi_k(x_i)\right]$$
$$= \sum_{k=1}^{m} \varphi_k(x_i)\left[\frac{\partial}{\partial t}\left(a_k(t)\right)\right]. \quad (4.3.1.9)$$

while the derivative with respect to t of the forward time projection model incorporates the change in the time averaged variable between the original model and the most recent time projections of the newest POD approximation of the variable at the previous time step. It is given by

$$\frac{\partial}{\partial t}F(x_i,t) \cong \frac{\partial}{\partial t}[\mu_F(x_i)] + \frac{\partial}{\partial t}\left[\sum_{k=1}^{m} a_k(t)\varphi_k(x_i)\right]$$
$$= \frac{\partial}{\partial t}[\mu_F(x_i)] + \sum_{k=1}^{m} \varphi_k(x_i)\left[\frac{\partial}{\partial t}\left(a_k(t)\right)\right] \qquad (4.3.1.10)$$

where $\frac{\partial}{\partial t}[\mu_F(x_i)]$ is obtained by recalculating the mean value of F in cell x_i over all previously calculated time points including the new ones. It is given by

$$\frac{\partial}{\partial t}[\mu_F(x_i)] = \frac{\mu_{F_{NEW}}(x_i) - \mu_{F_{OLD}}(x_i)}{t_{NEW} - t_{OLD}} \qquad (4.3.1.11)$$

where $\mu_{F_{NEW}}(x_i)$ represents the calculated mean at the current time step over all previous mean values in cell x_i. $\mu_{F_{OLD}}(x_i)$ represents the calculated mean at the time step immediately prior to the time step for $\mu_{F_{NEW}}(x_i)$. Finally, t_{NEW} is the current value of time and t_{OLD} is the value of time at the $\mu_{F_{OLD}}(x_i)$ calculation.

4.5 IMPLEMENTATION

As with any numerical model, implementation is important. Precision of results and the speed in which results are desired are considered carefully. In an effort to expand the audience of users for these ROMs, the development of software to implement these methodologies took into consideration the general preferences of the audience for whom these ROMs would provide the most use.

4.5.1 CAPE - OPEN COMPLIANCE

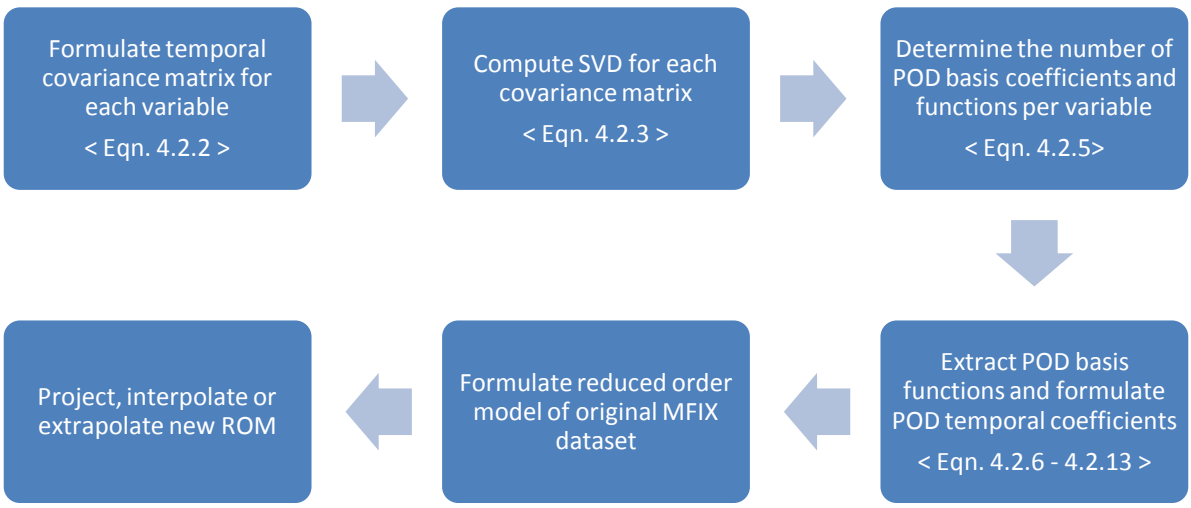
Computer-Aided Process Engineering (CAPE) Open standards are a set of internationally recognized rules and interfaces that were designed to aid in the development and inter-operability of various applications and their components. CAPE-Open standards are employed at algorithmic design and operational levels to facilitate the ease of open simulation [20]. This set of standards allows developers and users to create, modify and structure software developed under these standards to work almost seamlessly with other developers and users modeling interfaces. Keeping these standards in mind, the foundation for any algorithm's implementation utilizes an object oriented programming approach. Under this approach, routines related to a specific feature of the program are combined together as an "object." These objects share only that information which is absolutely necessary with other objects and their interactions and subsequent sharing of information is kept to a minimum. This type of framework eases the ability to change any inherent programming feature such as choice of nonlinear solver, type of foundational data set, or even the modeling framework itself. In this project, every effort was made to minimize overall model restrictions by treating physical parameters, domain structure, and POD parameters in a general sense which would make this software appealing to a much broader audience.

4.5.2 ALGORITHM DEVELOPMENT

Any CFD model must necessarily take into consideration the complexity of the algorithms used to calculate predictions of flow characteristics but most importantly the amount of computer memory required to make these calculations as well as performance speed. Likewise, both physical and computational limitations of the systems on which they are utilized must be accounted. While it is not safe to assume that every user has the capability to process a parallel implementation of the ROM based solvers offered in this dissertation, it has become common practice to implement new computational algorithms in this manner. For the implementation of ROM based solvers offered in this document as well as the construction of the ROM itself, OPEN-MPI was used. Both a parallel algorithm and parallel executable were created. For testing purposes, the POD modes and POD basis functions were calculated using 8 parallel processors. The testing was performed primarily on the WVU Math Department cluster, tron.math.wvu.edu, which is comprised of 64 Bit Dual Core AMD Opteron Processors 275; 2210MHz. The ROM based solvers utilized 40 parallel processors for the projection solvers and 20 parallel processors for the extrapolation and interpolation solvers.

The reduced order model solving process is highlighted in the Figure 3 flow chart below.

Figure 3: ROM – based solver algorithm flow chart



5.0 RESULTS

Input parameters and numerical solutions generated by MFIX for a flat bottom spouted fluidized bed were made available prior to development of the reduced order model. These solutions served as the construct for the POD basis functions, but it should be noted that only the parameters of the predictive simulations (no comparative data generated by MFIX) were made available prior to the prediction of the projection, interpolation, and extrapolation model scenarios.

A two-dimensional spouted fluidized bed was simulated in MFIX. The computational model for the bed was characterized with the following initial parameters: 55% solids void fraction (percentage of solids within a given cell), 200 cm/s y-velocity of gas in the spout, 30 cm/s y-velocity of gas around the spout, constant gas and solid viscosity, 0.05 cm particle diameter, 0.8 coefficient of restitution (measure of elasticity between the particles) and a 50 x 75 cell computational grid covering a 5 cm x 15 cm physical region. The POD methodology described in this dissertation was employed to characterize the data generated by MFIX simulation.

In this context, average relative error for a specific point in time, t, between the POD model and the MFIX model is defined as follows where x_i is the centered cell approximation for the methodological construct.

$$\frac{1}{NCELLS}\left[\sum_{i=1}^{NCELLS}\frac{POD(x_i,t)-MFIX(x_i,t)}{MFIX(x_i,t)}\right] \qquad (5.0.1)$$

5.1 POD ROM

In the following tables and figures, a comparison between MFIX simulation data and its ROM are given.

Table 1 illustrates the number of basis functions required to approximate each cell-centered variable of the Navier Stokes governing equations to assure that a reduced order model will capture 99% of its representative matrix energy across time of the simulation.

Table 1: Singular value variable summary

Variable	Number of POD Basis Functions	Percent of Energy captured	Largest Singular Value	Smallest Singular Value
U_g	78	99.0022%	11,490.9	354.5
V_g	76	99.0197%	18,229.5	625.4
U_s	77	99.0220%	2388.3	74.3
V_s	57	99.0147%	4137.5	126.6
ϵ_g	36	99.0425%	71.3	3.1
T_s	122	99.0172%	16,448.3	421.2
P_g	7	99.2495%	1,023,540.0	63,042.2
P_s	159	99.0020%	131,883.0	2,932.5

The number of POD basis functions and modes varies depending upon the desired amount of energy capture. However, it is noted that there is a slight difference in the required number of POD modes and basis functions depending upon the location of the nodal values approximated. Table 2 summarizes the number of modes and coefficients required at various energy level requirements as well as the nodal value numbers for 99% energy capture. Note that the number of POD basis functions needed to capure at a minimum 99% energy is much higher for solids pressure, 159, and solids granular temperature, 122, though this is still a sufficient reduction from the 1001 POD basis functions required to construct the full order model. While not known for certain, these are typically the two variables which are hardest to control in CFD simulation.

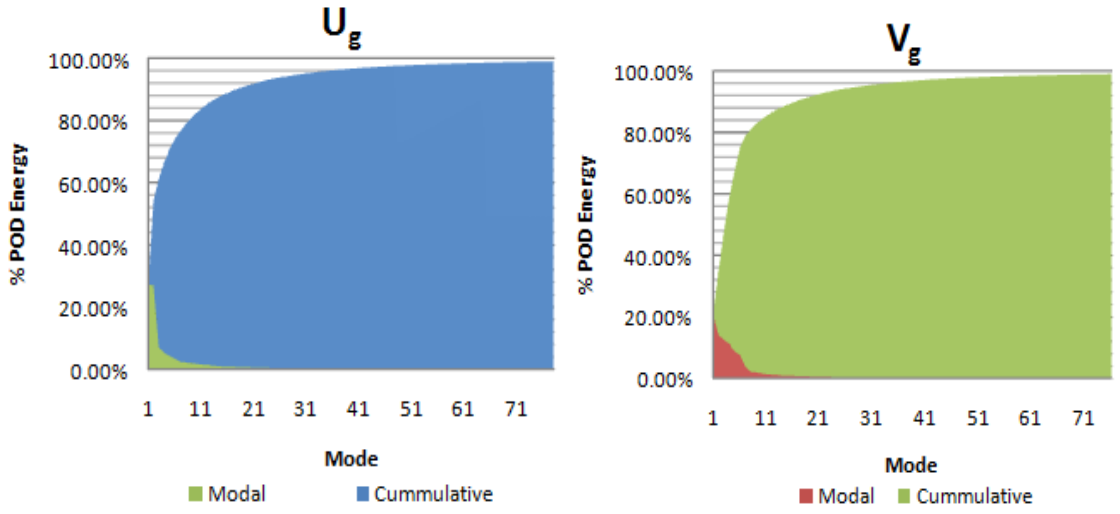
Table 2: Energy capture POD basis count summary

Variable	99% (Nodal Values)	99% (Center Values)	97.5% (Center Values)	95% (Center Values)	90% (Center Values)	85% (Center Values)	80% (Center Values)
U_g	72	78	48	31	18	12	9
V_g	85	76	45	29	17	11	8
U_s	73	77	45	27	13	8	6
V_s	81	57	29	16	9	7	5
ϵ_g	26	36	22	13	8	6	4
θ_s	122	122	79	49	25	14	9
P_g	5	7	4	3	2	2	2

P_s	167	159	89	47	20	12	8

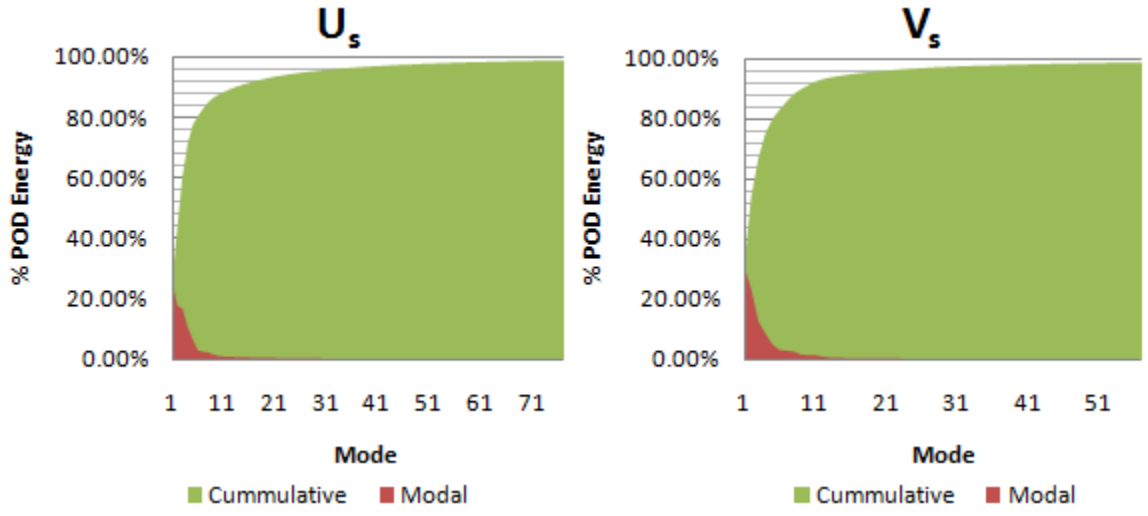
Figure 4 gives a visual interpretation of the POD energy spectrum across the eight variables in terms of energy percentage captured across the modes. The cumulative energy capture represents the total amount of energy captured by the largest *m* modes, while the modal energy captured illustrates only that proportion of total energy which the *m*th mode contributes to the overall energy capture. For example, the x-component of gas velocity requires the utilization of 78 modes to capture 99% of the cumulative energy, 12 modes for 85% cumulative energy capture and 9 for 80%. However, the first POD mode captures over 20% of the energy on its own. (This is also seen in Table 2.) Excluding the first dozen modes, the remaining modes of the x-component of gas velocity each contribute less than 1% to the total energy indicating their contribution to an approximate solution is very minor.

Figure 4: POD basis function percent energy captured by variable



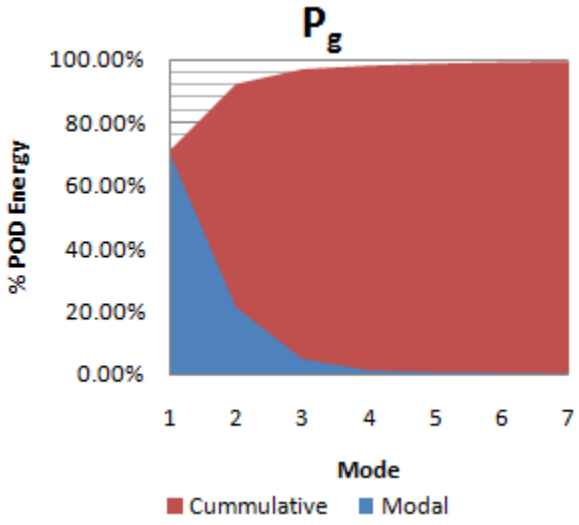
(a) (b)

Figure 4 (cont): **POD basis function percent energy captured by variable**

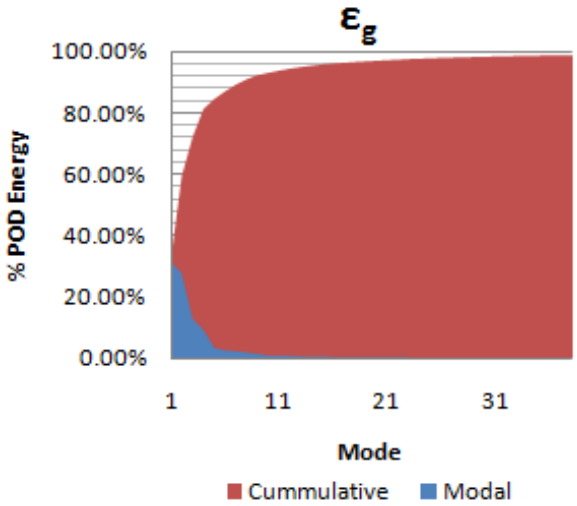


(c)

(d)

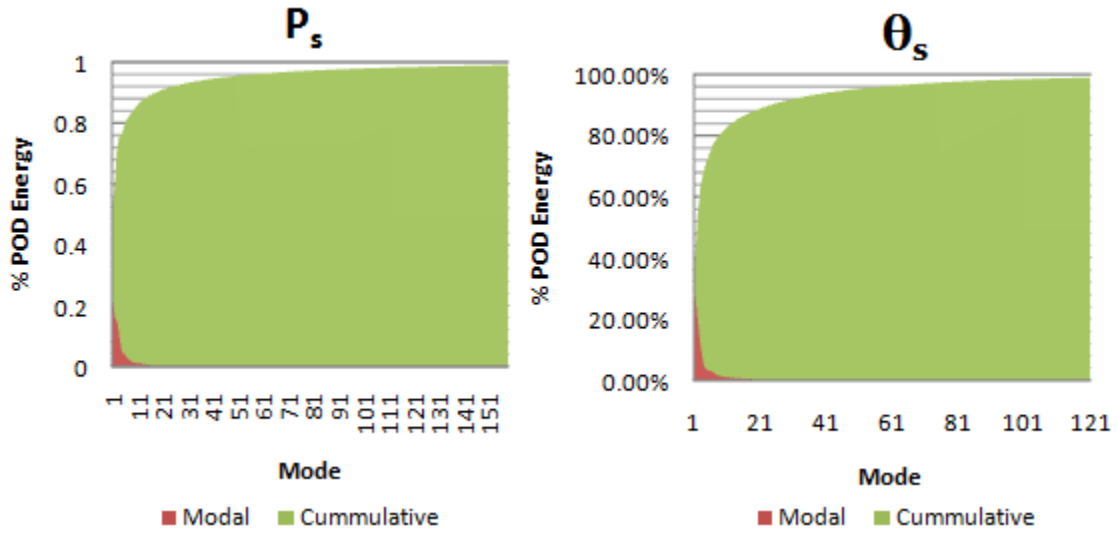


(e)



(f)

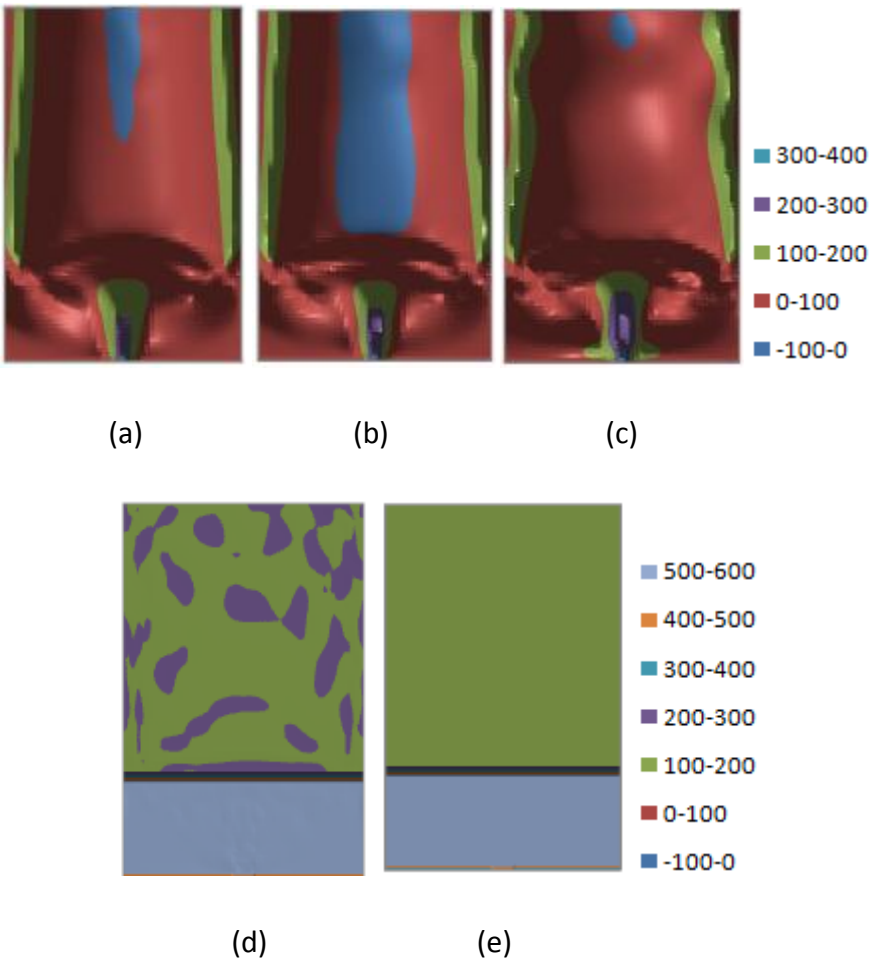
<u>Figure 4 (cont):</u> POD basis function percent energy captured by variable



(g) (h)

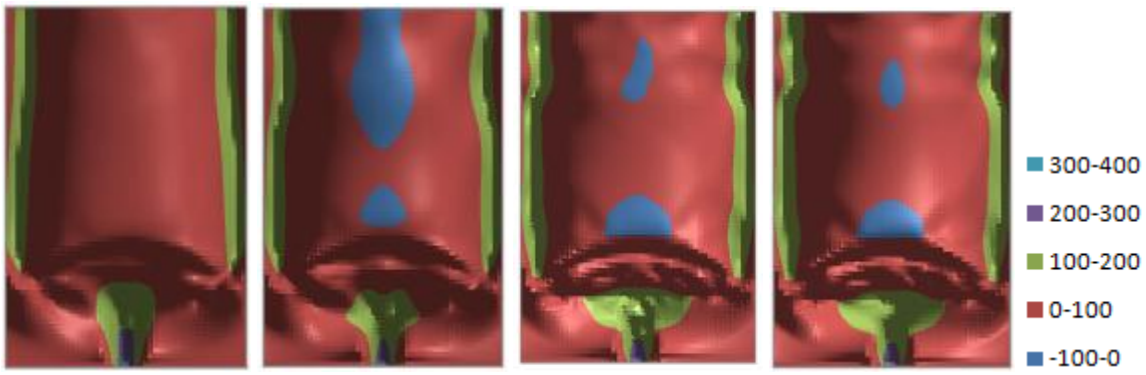
The images in Figure 5 illustrate approximations for the gas y-velocity, V_g, at time t=0s. Specifically, (a) illustrates the POD variable mean, (b) denotes the contribution of the first POD approximation plus the mean and (c) denotes the contribution of the second POD approximation plus the mean. Image (d) illustrates the full POD approximation which required 76 POD modes and basis functions to capture 99% of the energy of the MFIX model. Image (e) illustrates the MFIX model's approximation of V_g at time t=0s. Although at first glance it may seem that the images in (d) and (e) are markedly different, it is important to recognize the visual effect of having values just slightly above and below the 200 cm/s threshold. Relative error between (d) and (e) in Figure 5 is less than 0.5%.

Figure 5: V_g **at time t=0s (cm/s)**



The images in Figure 6 illustrate approximations for the gas y-velocity, V_g, at time t=5s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of V_g at time t=5s. Notice that there is sufficiently acceptable agreement between (c) and (d) as quantified in Figure 8 image (b).

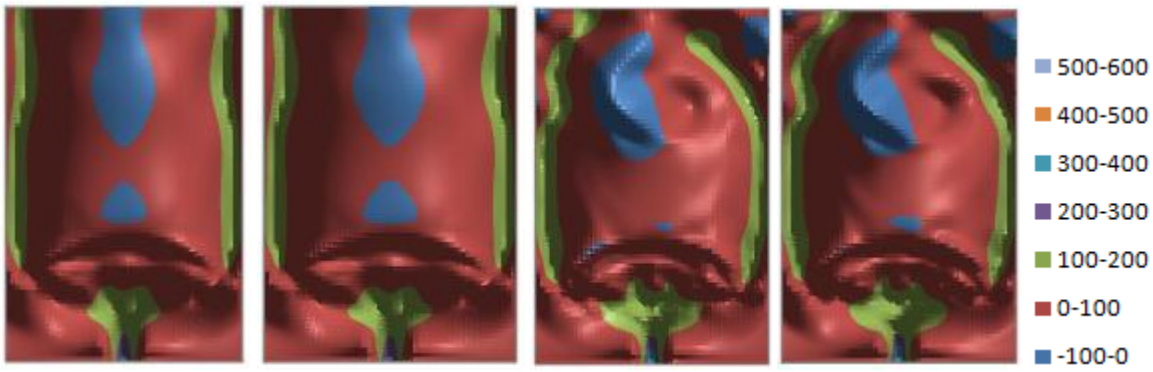
Figure 6: V_g **at time t=5s (cm/s)**



(a) (b) (c) (d)

The images in Figure 7 illustrate approximations for the gas y-velocity, V_g, at time t=10s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of V_g at time t=10s. Note that the curve of the velocity plume is captured in (c). Also, small regions near the top of the spout with non-positive y-velocities are captured in the reduced order model representation (c) though the size and precise location of these features vary slightly from the MFIX model (d).

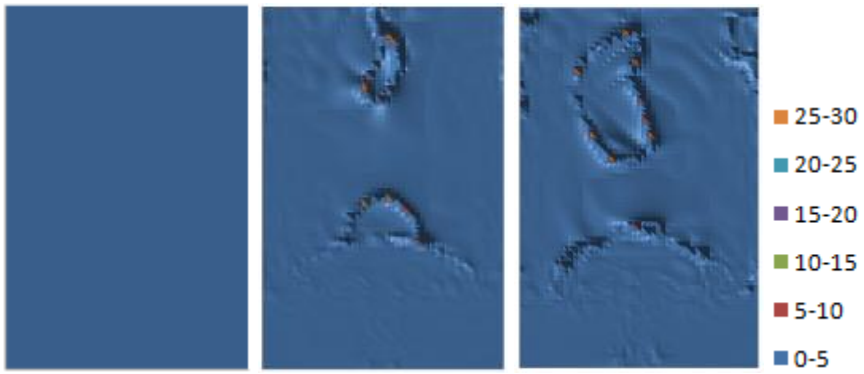
Figure 7: V_g **at time t=10s (cm/s)**



(a) (b) (c) (d)

Figure 8 images show the relative error between the POD approximations and the MFIX model at various times for the gas y-velocity, V_g. Image (a) illustrates the relative error at 0s while (b) and (c) depict the relative error between the models at 5s and 10s respectively. *As was expected, the errors in POD ROM representations generally appear along the boundaries of physical entities where change occurs rapidly.* This is illustrated below in figures (b) and (c) where there is noticeable variation in the outline of the velocity plume as well as at the top of the bed.

Figure 8: V_g relative error



(a) (b) (c)

Figure 9 provides the average relative error across the time span of the reduced order model for the gas y-velocity, V_g. At least 96% of the time, the average relative error is less than 0.5.

Figure 9: V_g **average relative error**

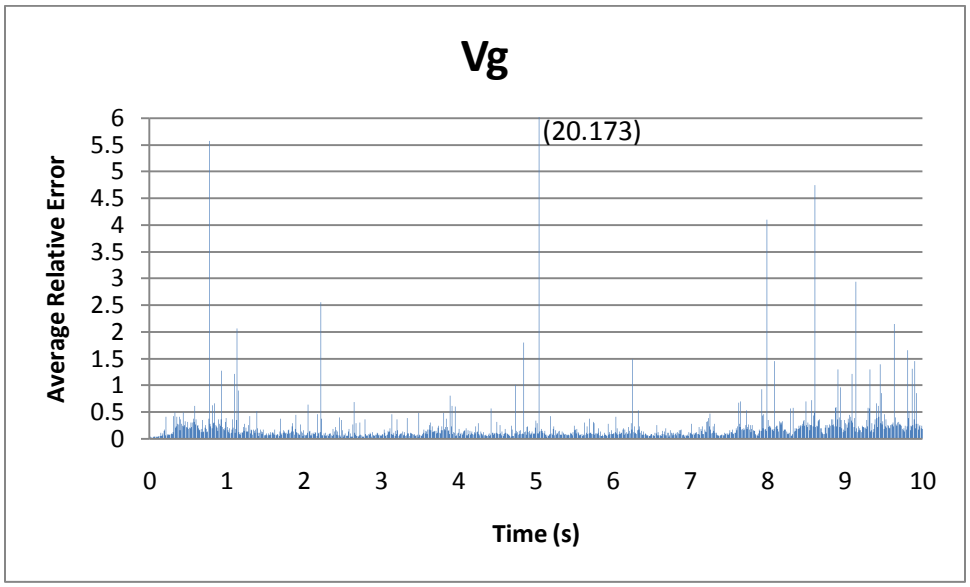
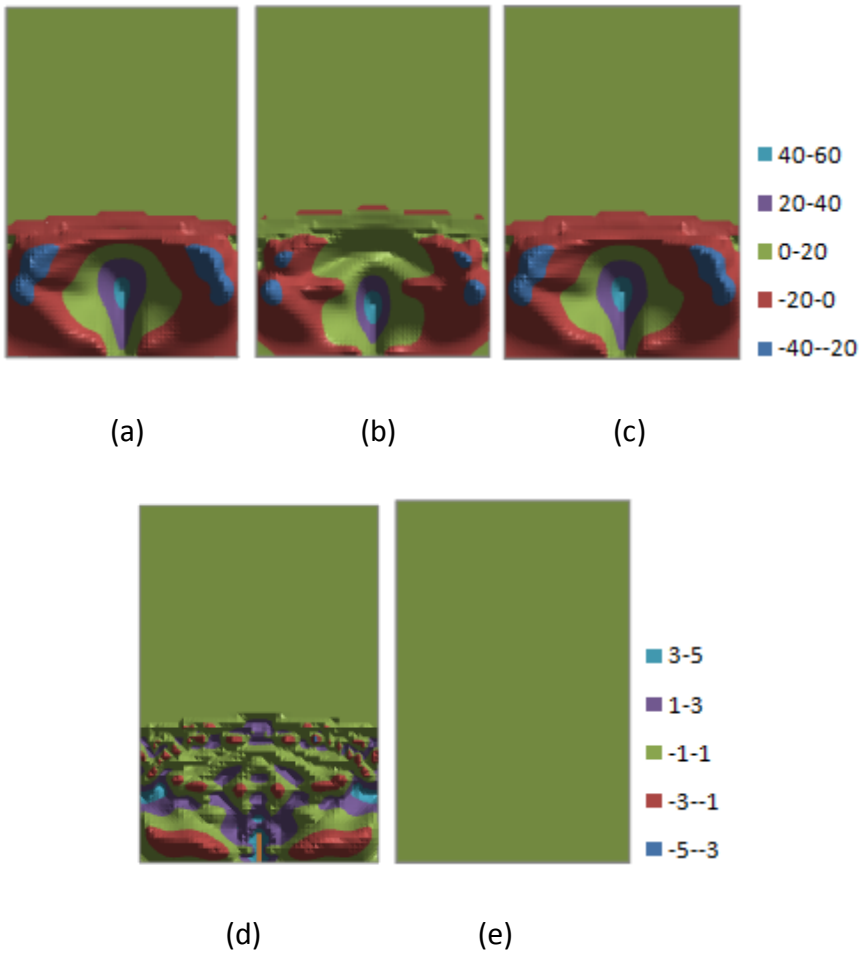


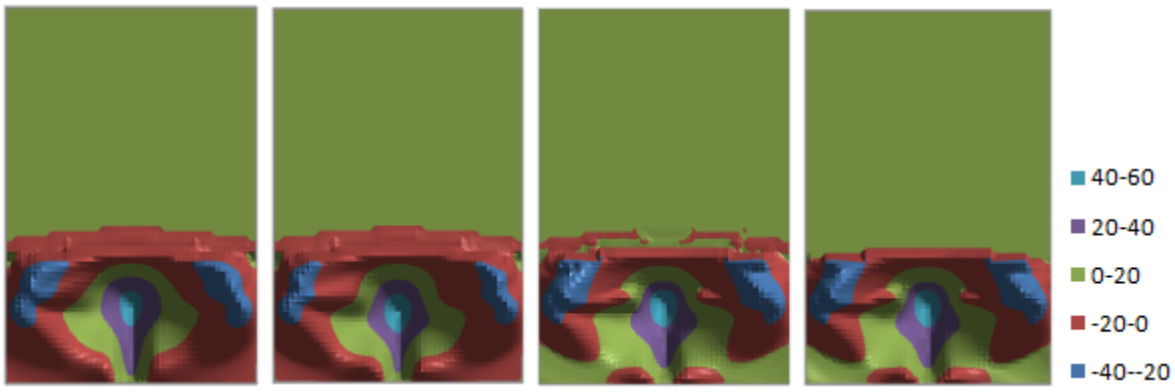
Figure 10 illustrates various partial POD approximations for the solids y-velocity, V_s, at time t=0s. Specifically, (a) illustrates the POD variable mean, (b) denotes the contribution of the first POD approximation plus the mean and (c) denotes the contribution of the second POD approximation plus the mean. Image (d) illustrates the full POD approximation which required 57 POD modes and basis functions to capture 99% of the energy of the MFIX model. Image (e) illustrates the MFIX model's approximation of V_s at time t=0s. Given the rapid transition of the particles from their resting state to their quasi-steady state during spouting it is not surprising that the largest error in their y-velocity POD representation is at the initial time step. The maximum relative error between figures (d) and (e) at this time step is 1.0.

Figure 10: V_s **at time t=0s (cm/s)**



The images in Figure 11 illustrate approximations for the particle y-velocity, V_s, at time t=5s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of V_s at time t=5s. Notice that the variation between the MFIX approximation and the POD ROM representation after five seconds has much better agreement than the initial conditions of Figure 9 (d) and (e).

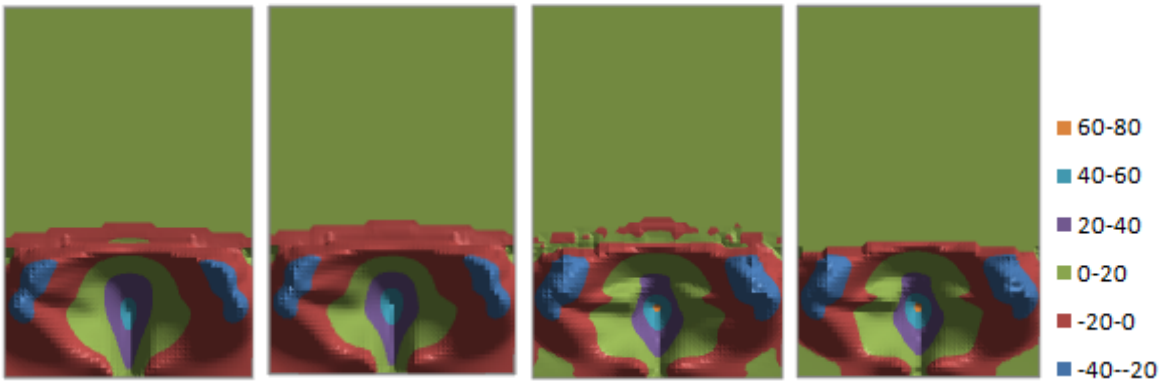
Figure 11: **V_s at time t=5s (cm/s)**



(a) (b) (c) (d)

The images in Figure 12 illustrate similar approximations for the particle y-velocity, V_s, at time t=10s. By the last time step, the agreement between (c) and (d) has become much better at the top of the spout. While there are noticeable regions above the spout in the POD ROM exhibiting small non-positive velocities, they are sufficiently near the top of the spout where the particles are heavily concentrated. Additionally, the tiny light blue region midway up the spout height in the center of the central jet is identified in the POD ROM.

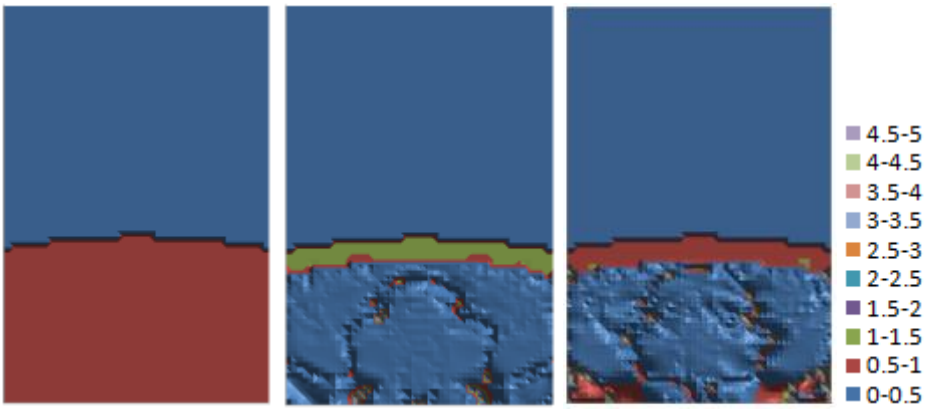
Figure 12: V_s **at time t=10s (cm/s)**



(a) (b) (c) (d)

Figure 13 images show the relative error between the POD approximations and the MFIX model at various times for the particle y-velocity, V_s. Image (a) illustrates the relative error at 0s while (b) and (c) depict the relative error between the models at 5s and 10s respectively. Excluding the initial time step, a majority of the particle y-velocity regions exhibit agreement within 0.5 relative error. Again, note that the boundary of the spout and band at the top of the spout where the particles are bouncing around yields the largest error. It is also informative that the velocity dead zones at the bottom of the bed where particles tend to build up is defined.

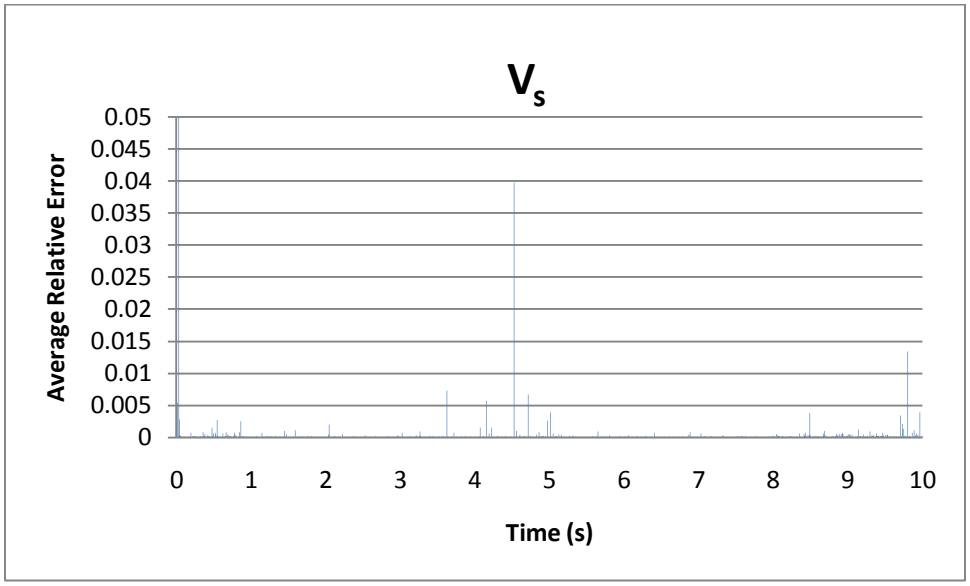
Figure 13: V_s **relative error**



(a) (b) (c)

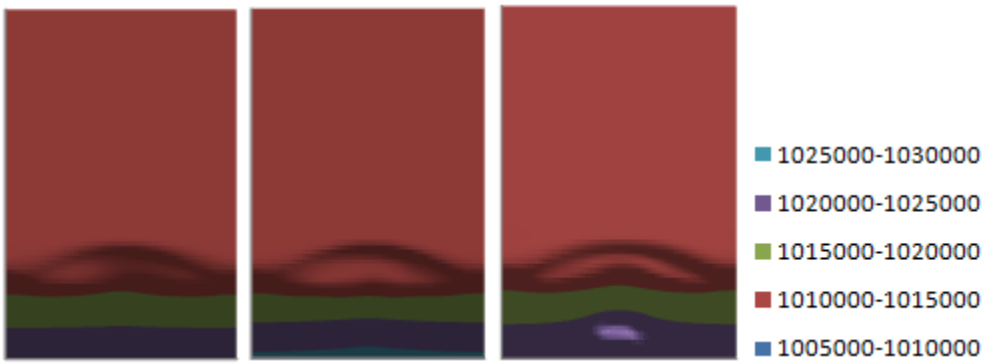
Figure 14 provides the average relative error across the time span of the reduced order model for the particle y-velocity, V_s. Except for at 5 time points (out of 1,001), the error of the y-velocity at each time step is very minimal (< 0.005). It is important to keep in mind this is due to a majority of the bed having 0 cm/s particle y-velocity, (i.e. there are no particles which enter the upper portion of the bed).

Figure 14: V_s average relative error

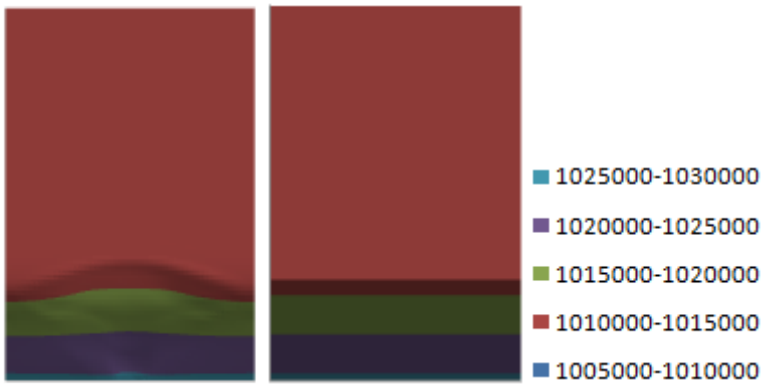


The images in Figure 15 illustrate approximations for the gas pressure, P_g, at time t=0s. Specifically, (a) illustrates the POD variable mean, (b) denotes the contribution of the first POD approximation plus the mean and (c) denotes the contribution of the second POD approximation plus the mean. Image (d) illustrates the full POD approximation which required 7 POD modes and basis functions to capture 99% of the energy of the MFIX model. Image (e) illustrates the MFIX model's approximation of P_g at time t=0s. Note the slight bulge at the top of the bed in the POD ROM representation which does not appear in the MFIX dataset.

Figure 15: P_g **at time t=0s (Pa)**



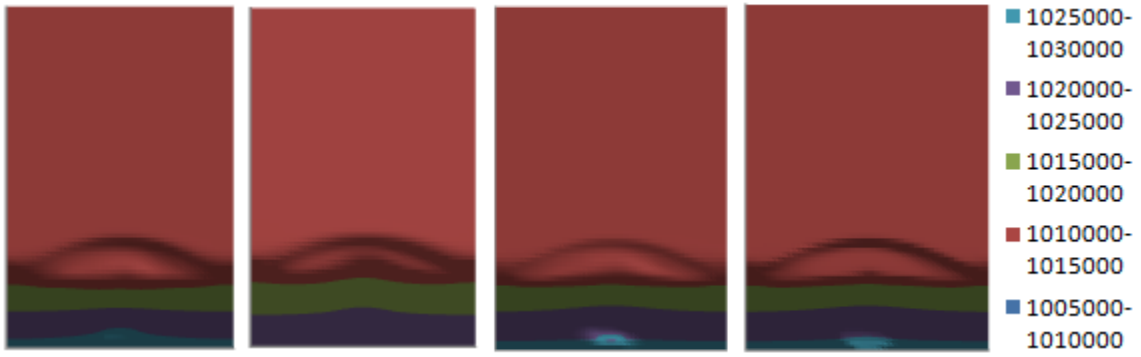
(a) (b) (c)



(d) (e)

The images in Figure 16 illustrate approximations for the gas pressure, P_g, at time t=5s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of P_g at time t=5s.

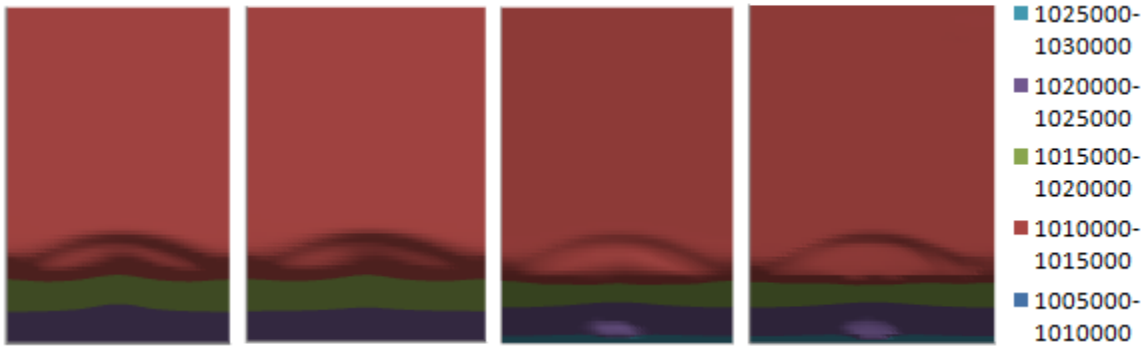
Figure 16: P_g **at time t=5s (Pa)**



(a) (b) (c) (d)

The images in Figure 17 illustrate approximations for the gas pressure, P_g, at time t=10s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of P_g at time t=10s. Notice that there is some slight variation in pressure near the walls at the top of the spout between the MFIX data set and the POD representation.

Figure 17: P_g **at time t=10s (Pa)**



(a) (b) (c) (d)

Figure 18 images show the relative error between the POD approximations and the MFIX model at various times for the gas pressure P_g. Image (a) illustrates the relative error at 0s while (b) and (c) depict the relative error between the models at 5s and 10s respectively.

Figure 18: P_g **relative error**

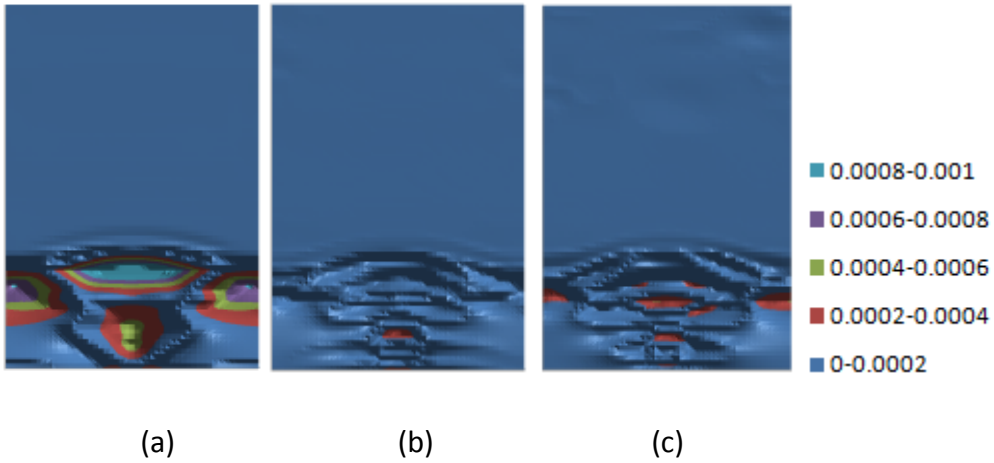
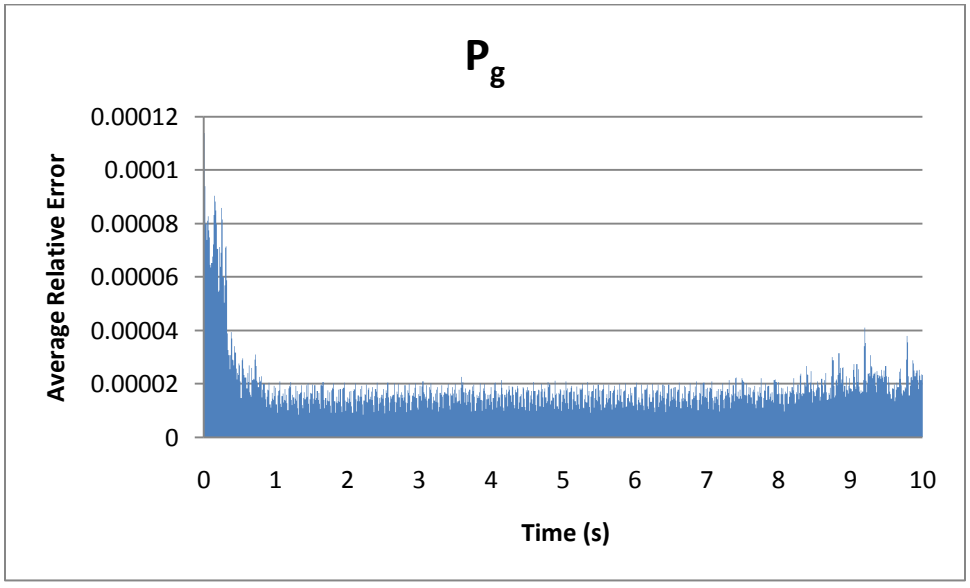


Figure 19 provides the average relative error across the time span of the reduced order model for the gas pressure, P_g. Notice that the relative error at each time step is incredibly small. This indicates a very good POD ROM pressure approximation.

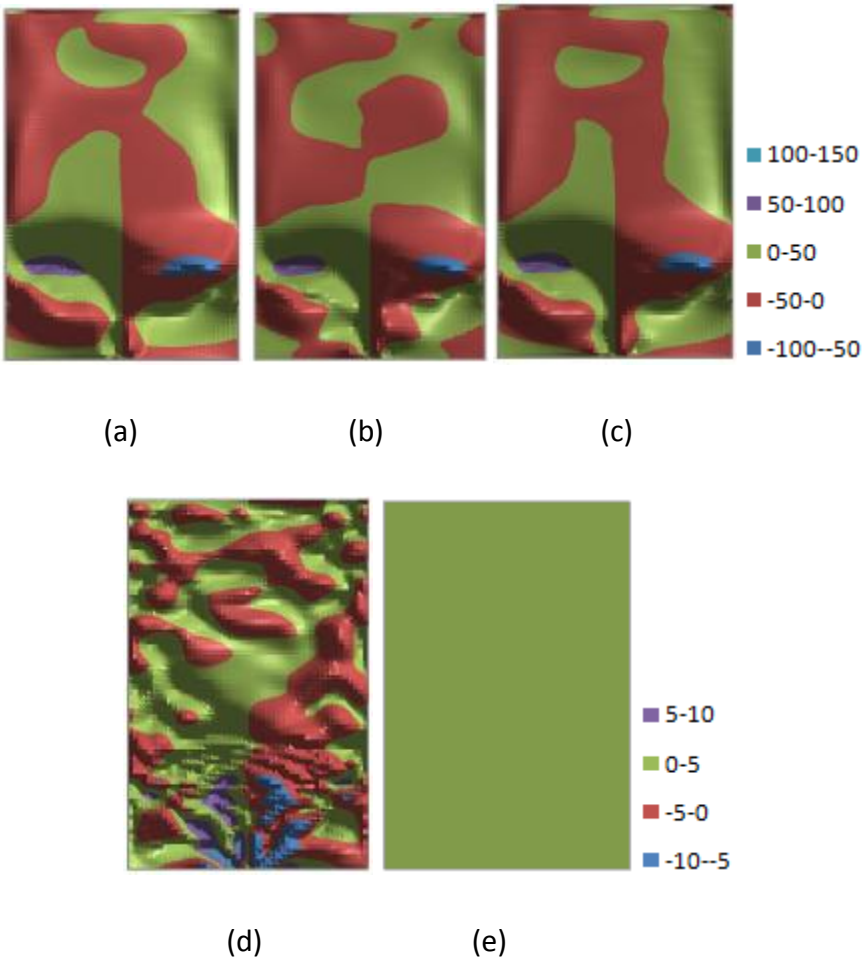
Figure 19: P_g **average relative error**



The images in Figure 20 illustrate approximations for the gas x-velocity, U_g, at time t=0s. Specifically, (a) illustrates the POD variable mean, (b) denotes the contribution of the first POD

approximation plus the mean and (c) denotes the contribution of the second POD approximation plus the mean. Image (d) illustrates the full POD approximation which required 78 POD modes and basis functions to capture 99% of the energy of the MFIX model. Image (e) illustrates the MFIX model's approximation of U_g at time t=0s. For the most part, one expects the x-component of gas velocity is symmetric with respect to the center of the central jet. With the exception of the upper regions of the bed where the velocity plume oscillates, this is the case.

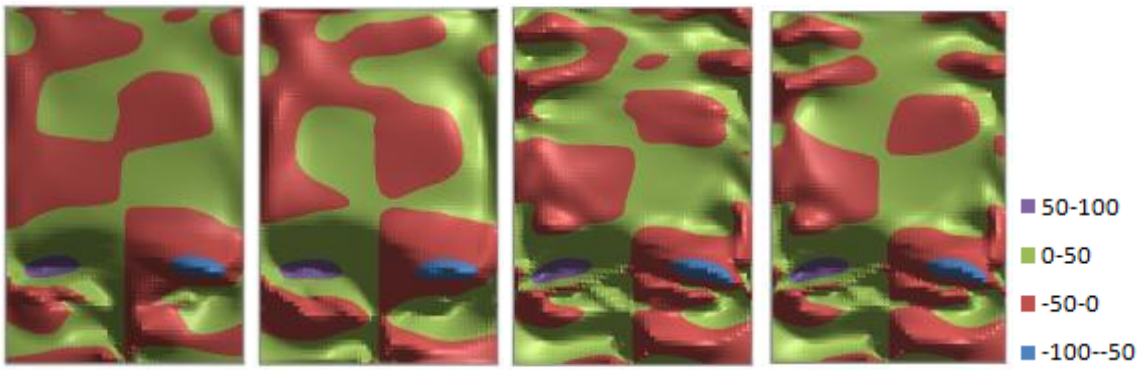
Figure 20: U_g at time t=0s (cm/s)



The images in Figure 21 illustrate approximations for the gas x-velocity, U_g, at time t=5s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of U_g at time

t=5s. Note the agreement between (c) and (d), although the x-velocity at the top of the bed is not quite symmetric with respect to the center of the central jet.

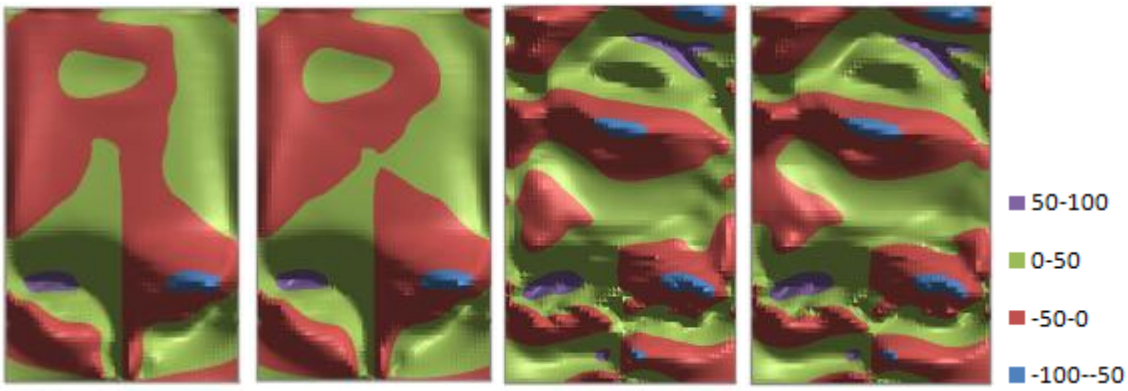
Figure 21: U_g at time t=5s (cm/s)



(a) (b) (c) (d)

The images in Figure 22 illustrate approximations for the gas x-velocity, U_g, at time t=10s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of U_g at time t=10s. Notice the agreement between (c) and (d).

Figure 22: U_g at time t=10s (cm/s)



(a) (b) (c) (d)

Figure 23 images show the relative error between the POD approximations and the MFIX model at various times for the gas x-velocity, U_g. Image (a) illustrates the relative error at 0s while (b) and (c) depict the relative error between the models at 5s and 10s respectively. Notice that the regions exhibiting high relative error at both five and ten seconds define the physical regions experiencing the most rapid transition of this variable in the model. This is due to the smoothing of results which is caused by the POD ROM'S reduced representation.

Figure 23: U_g relative error

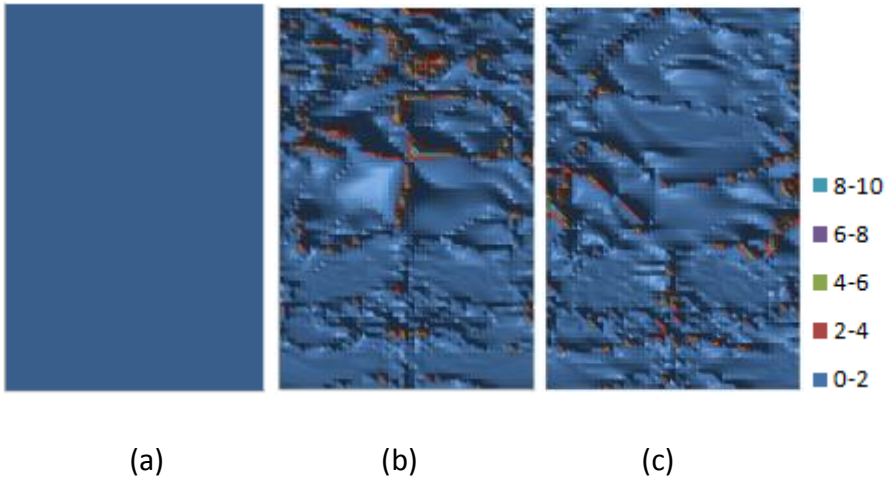


Figure 24 provides the average relative error across the time span of the reduced order model for the gas x-velocity, U_g.

Figure 24: U_g **average relative error**

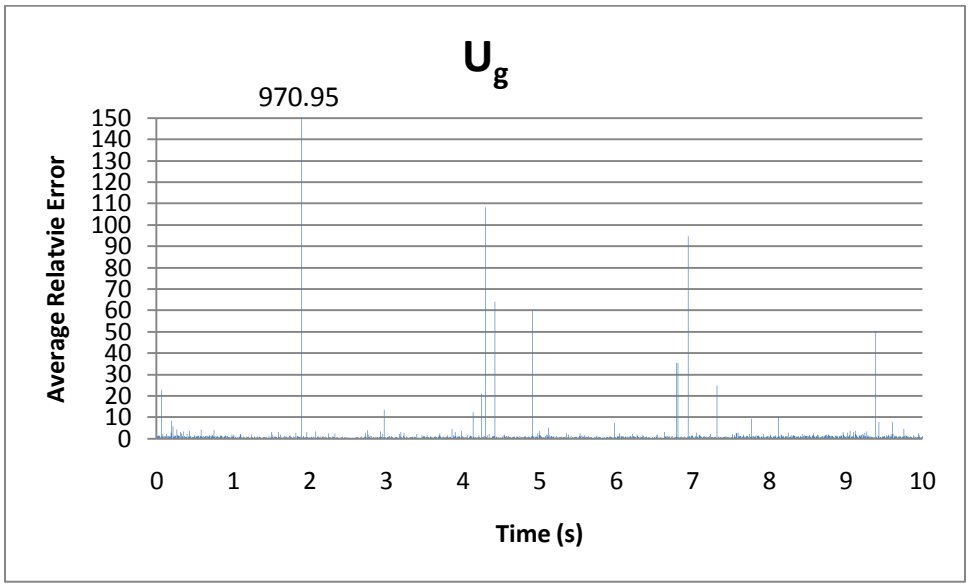
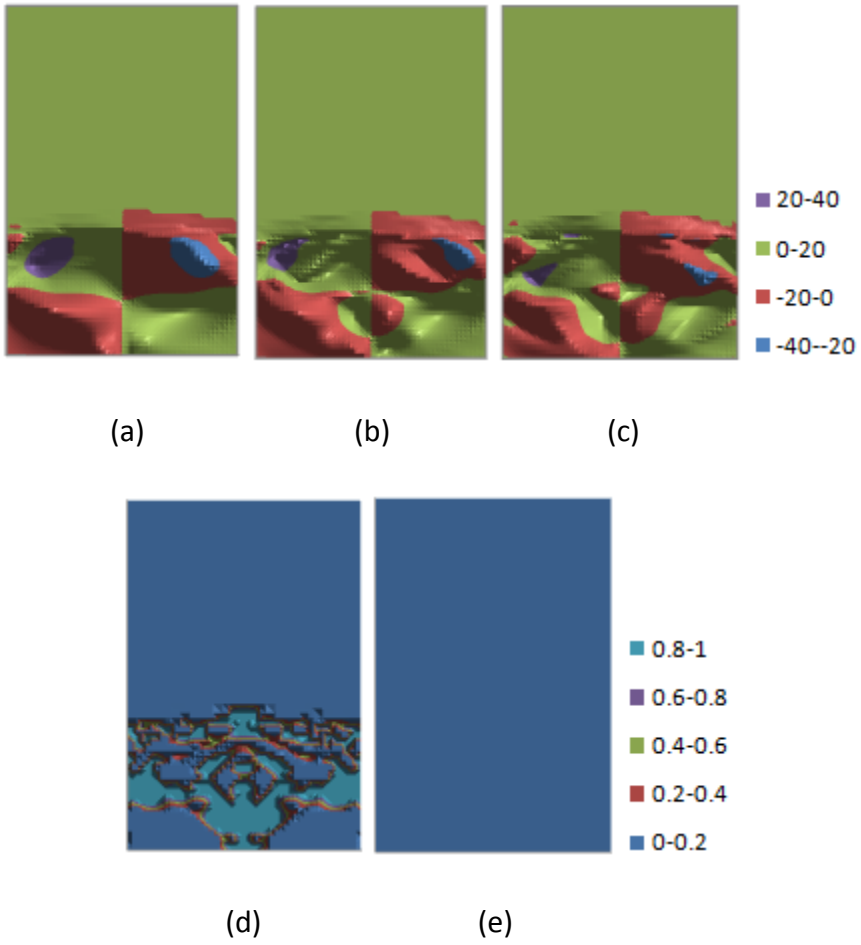


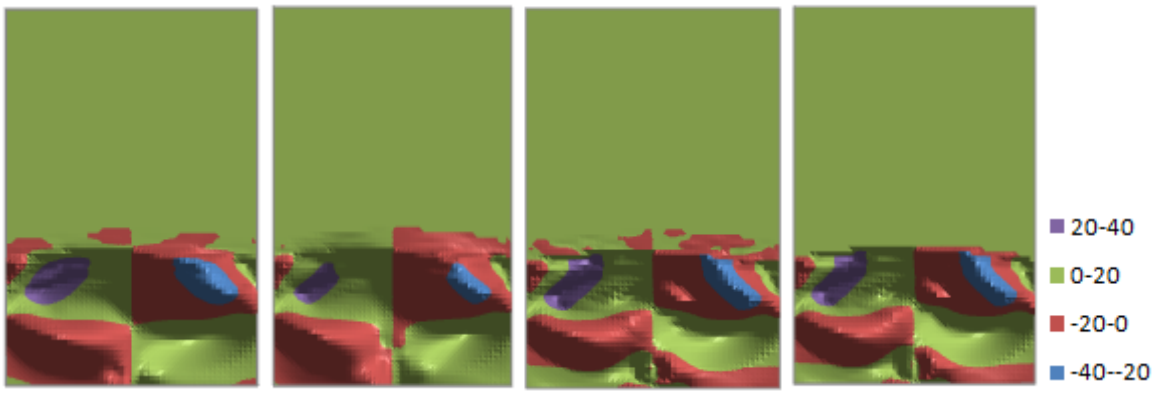
Figure 25 illustrates various partial POD approximations for the solids x-velocity, U_s, at time t=0s. Specifically, (a) illustrates the POD variable mean, (b) denotes the contribution of the first POD approximation plus the mean and (c) denotes the contribution of the second POD approximation plus the mean. Image (d) illustrates the full POD approximation which required 77 POD modes and basis functions to capture 99% of the energy of the MFIX model. Image (e) illustrates the MFIX model's approximation of U_s at time t=0s. The disagreement between Figure 24 (d) and (e) highlights the initialization effect.

Figure 25: U_s **at time t=0s (cm/s)**



The images in Figure 26 illustrate approximations for the particle x-velocity, U_s, at time t=5s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of U_s at time t=5s.

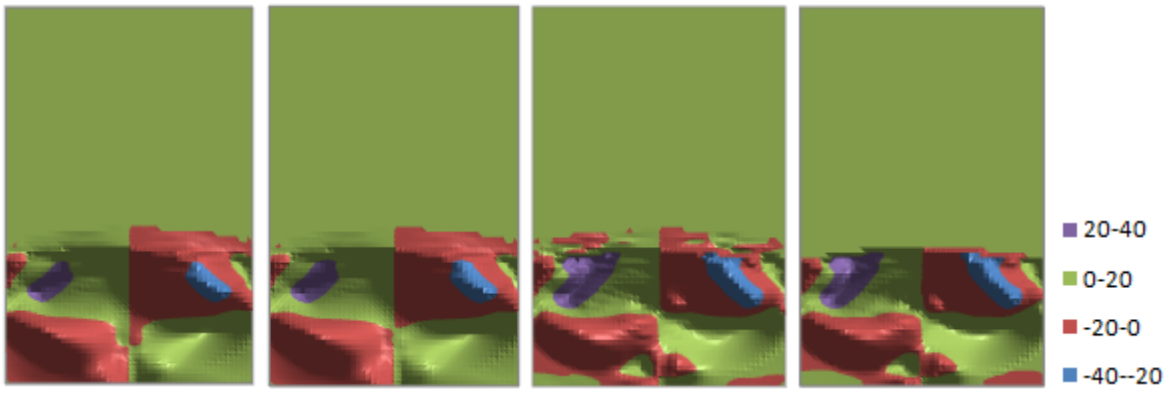
Figure 26: **U_s at time t=5s (cm/s)**



(a) (b) (c) (d)

The images in Figure 27 illustrate similar approximations for the particle x-velocity, U_s, at time t=10s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean.

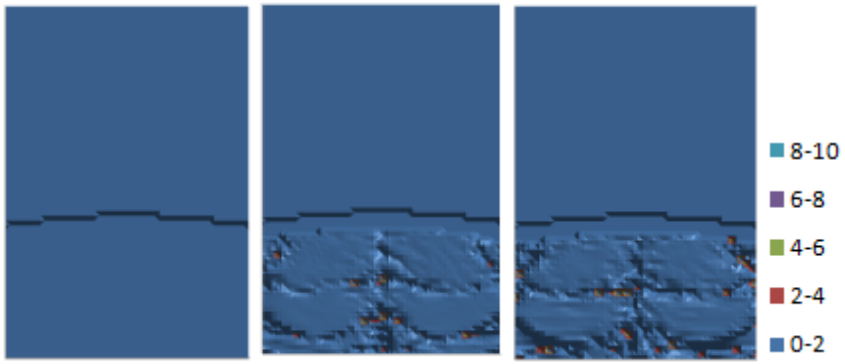
Figure 27: **U_s at time t=10s (cm/s)**



(a) (b) (c) (d)

Figure 28 images show the relative error between the POD approximations and the MFIX model at various times for the particle x-velocity, U_s. Image (a) illustrates the relative error at 0s while (b) and (c) depict the relative error between the models at 5s and 10s respectively. Again notice that the areas containing the largest relative error are the areas at the top of the spout, near the center of the jet and in the bottom corners where particles tend to pool.

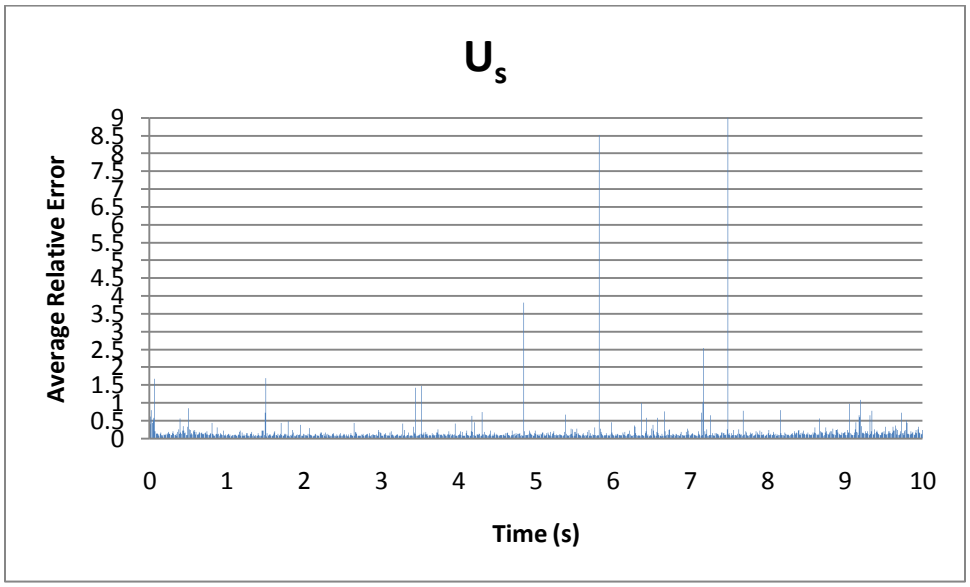
Figure 28: U_s **relative error**



(a) (b) (c)

Figure 29 provides the average relative error across the time span of the reduced order model for the particle x-velocity, U_s. 97% of the time steps have average relative error less than 0.5.

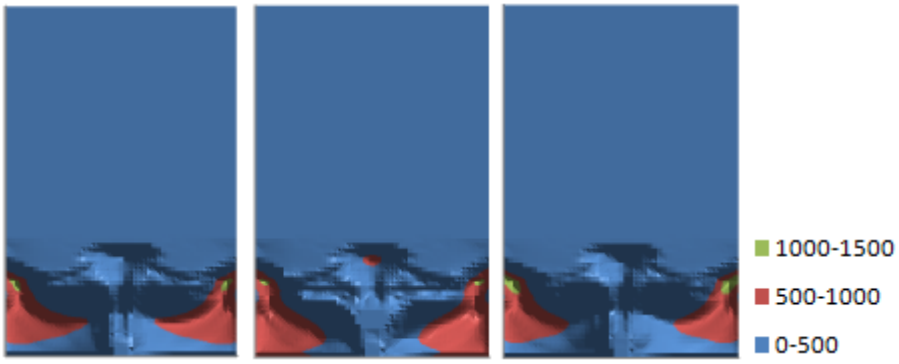
Figure 29: U_s **average relative error**



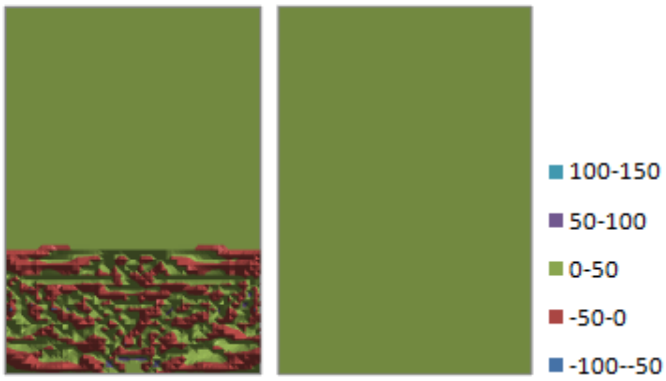
The images in Figure 30 illustrate approximations for the solid particle pressure, P_s, at time t=0s. Specifically, (a) illustrates the POD variable mean, (b) denotes the contribution of the first POD approximation plus the mean and (c) denotes the contribution of the second POD

approximation plus the mean. Image (d) illustrates the full POD approximation which required the largest number of all variables managed, 159, of POD modes and basis functions to capture 99% of the energy of the MFIX model. Image (e) illustrates the MFIX model's approximation of P_s at time t=0s.

Figure 30: P_s at time t=0s (Pa)



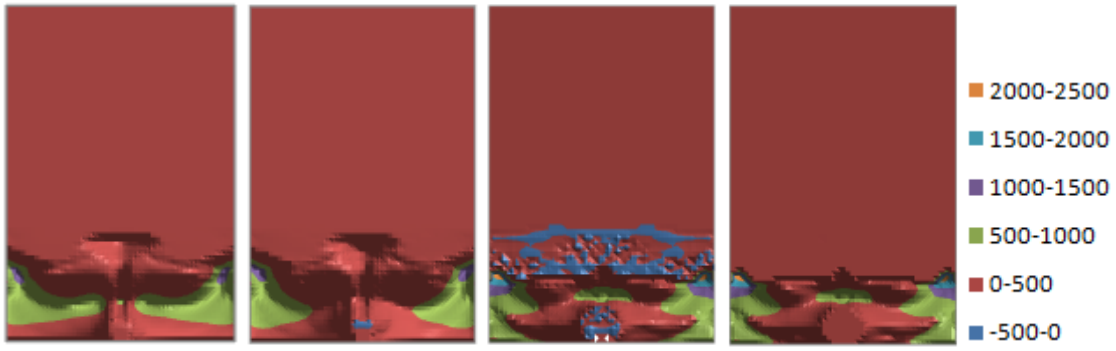
(a) (b) (c)



(d) (e)

The images in Figure 31 illustrate approximations for the solid particle pressure, P_s, at time t=5s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of P_s at time t=5s.

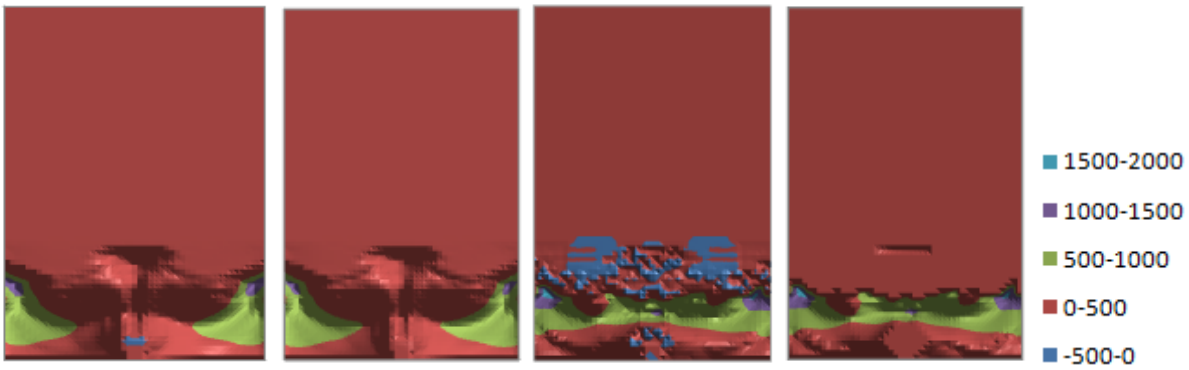
Figure 31: **P_s at time t=5s (Pa)**



(a) (b) (c) (d)

The images in Figure 32 illustrate approximations for the solid particle pressure, P_s, at time t=10s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of P_s at time t=10s.

Figure 32: **Ps at time t=10s (Pa)**

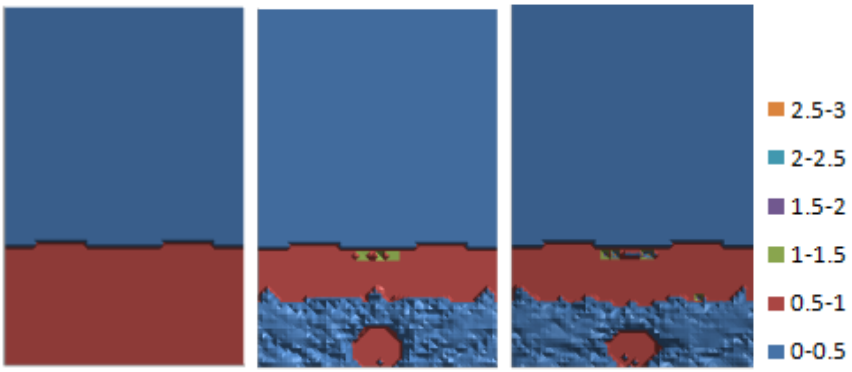


(a) (b) (c) (d)

Figure 33 images show the relative error between the POD approximations and the MFIX model at various times for the solid particle pressure P_s. Image (a) illustrates the relative error at 0s while (b) and (c) depict the relative error between the models at 5s and 10s respectively. As was seen in the two preceding figures the largest variation in pressure of the solid particles between

the MFIX data set and the POD ROM representation is at the top center of the spout. This is illustrated in Figure 33 (b) and (c).

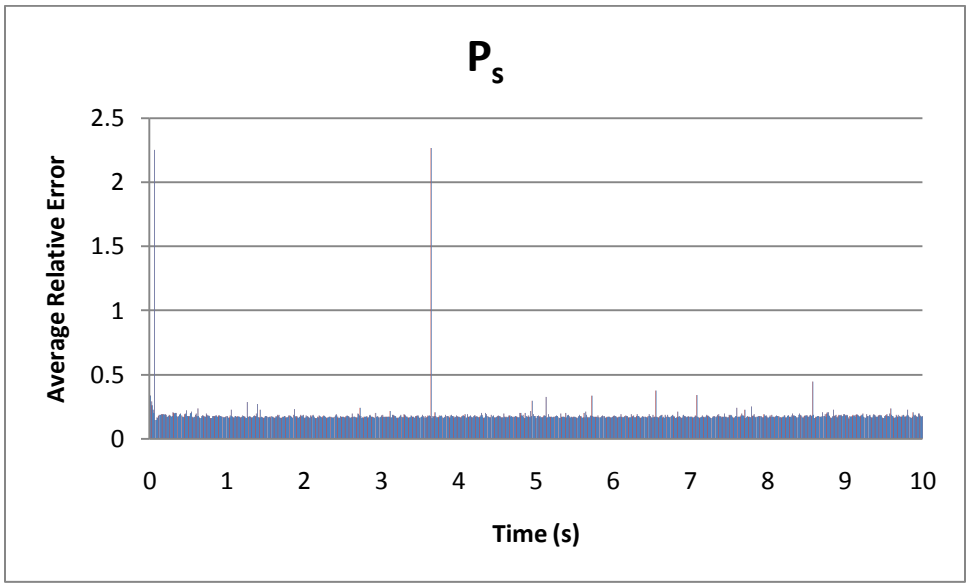
Figure 33: Ps relative error



(a) (b) (c)

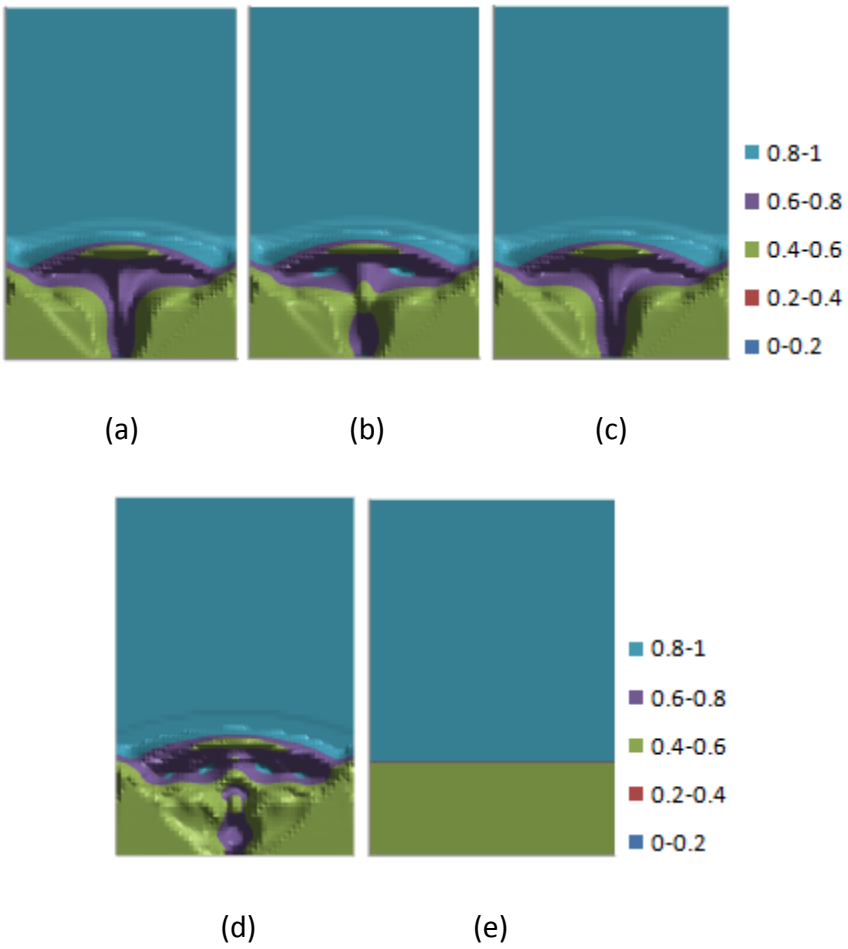
Figure 34 provides the average relative error across the time span of the reduced order model for the solid particle pressure, P_s. For the most part, the solid pressure relative error is less than 0.5 with the exception of two time points. In most CFD simulations, it is common to notice spikes in solids pressure which are the result of natural numerical artifacts. The POD model in this instance did not detect the spikes and therefore the average relative error at these time points is drastically higher.

Figure 34: **Ps average relative error**



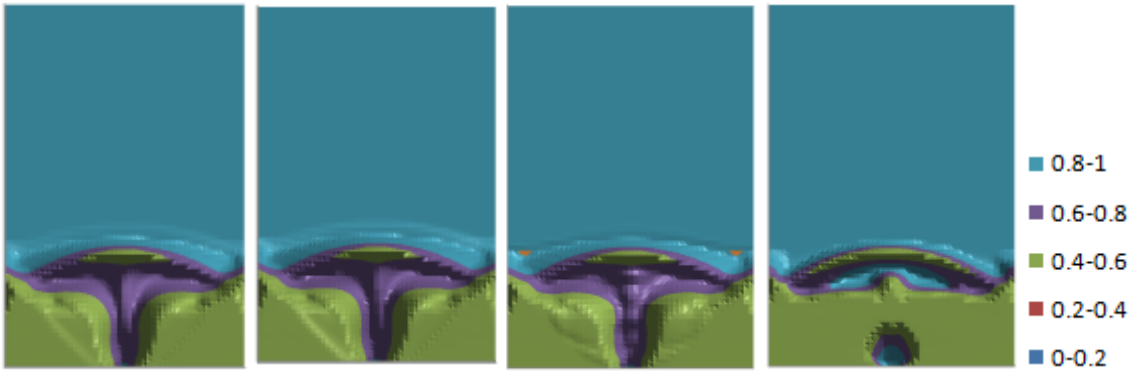
The images in Figure 35 illustrate approximations for the gas void fraction, ε_g, at time t=0s. Specifically, (a) illustrates the POD variable mean, (b) denotes the contribution of the first POD approximation plus the mean and (c) denotes the contribution of the second POD approximation plus the mean. Image (d) illustrates the full POD approximation which required 36 POD modes and basis functions to capture 99% of the energy of the MFIX model. Image (e) illustrates the MFIX model's approximation of ε_g at time t=0s. Recall that void fraction of both the gas and solid particles should be bounded between 0 and 1. These figures denote the proportion of gas found in each cell. Note that this bound holds even though it was not enforced during the approximation. The most variation between the MFIX data set and the POD ROM representation is again defined at the spout, the area experiencing the most physical change.

Figure 35: ε_g **at time t=0s**



The images in Figure 36 illustrate approximations for the gas void fraction, ε_g, at time t=5s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of ε_g at time t=5s.

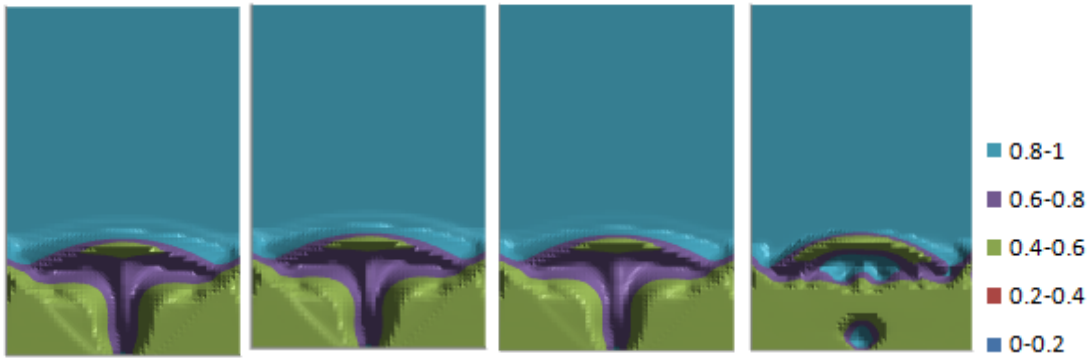
Figure 36: ε_g **at time t=5s**



(a) (b) (c) (d)

The images in Figure 37 illustrate approximations for the gas void fraction, ε_g, at time t=10s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of ε_g at time t=10s.

Figure 37: ε_g **at time t=10s**



(a) (b) (c) (d)

Figure 38 images show the relative error between the POD approximations and the MFIX model at various times for the gas void fraction, ε_g. Image (a) illustrates the relative error at 0s while (b) and (c) depict the relative error between the models at 5s and 10s respectively.

Figure 38: ε_g **relative error**

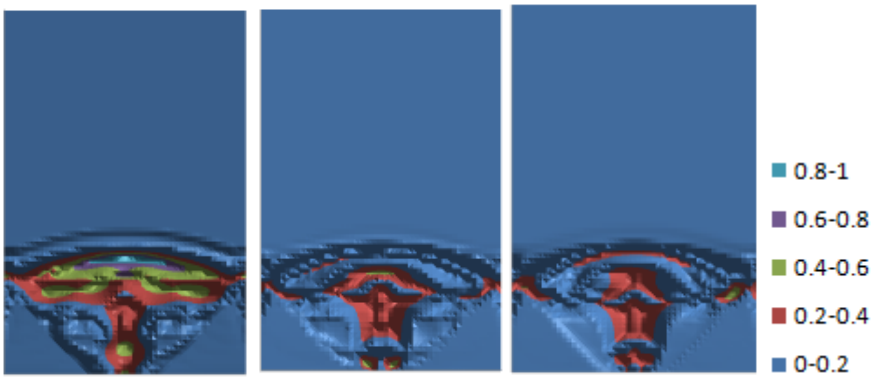


Figure 39 provides the average relative error across the time span of the reduced order model for the gas void fraction, ε_g. The average relative error of the POD ROM representation is less than 0.015 for every time step although it is heavily influenced by the upper portion of the bed which is in perfect agreement with the MFIX data set as those cells contain no particles.

Figure 39: ε_g **average relative error**

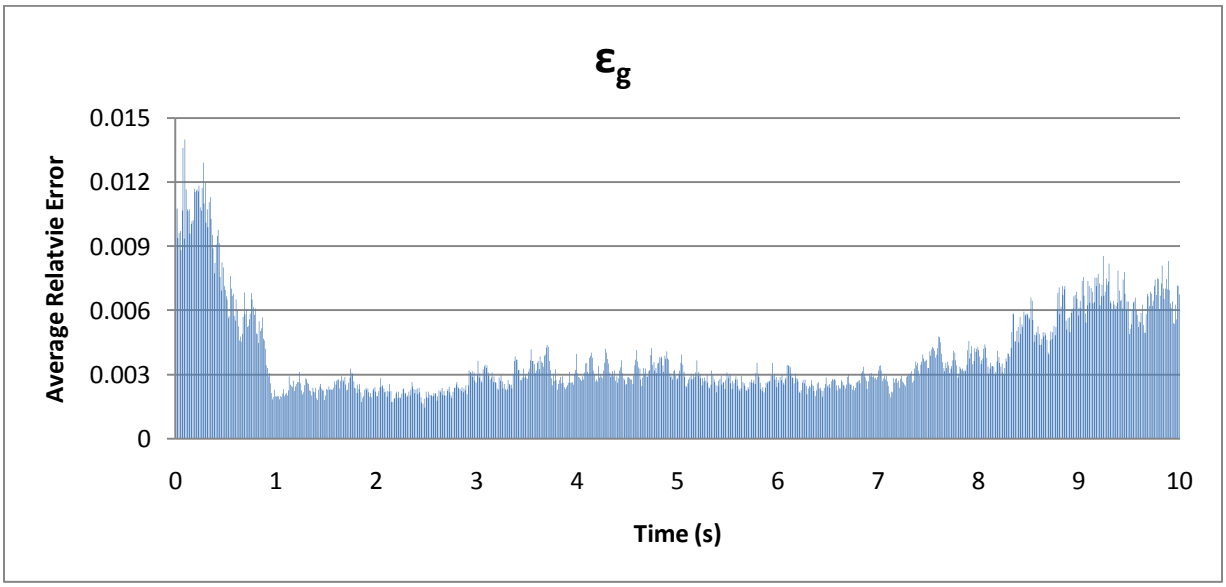
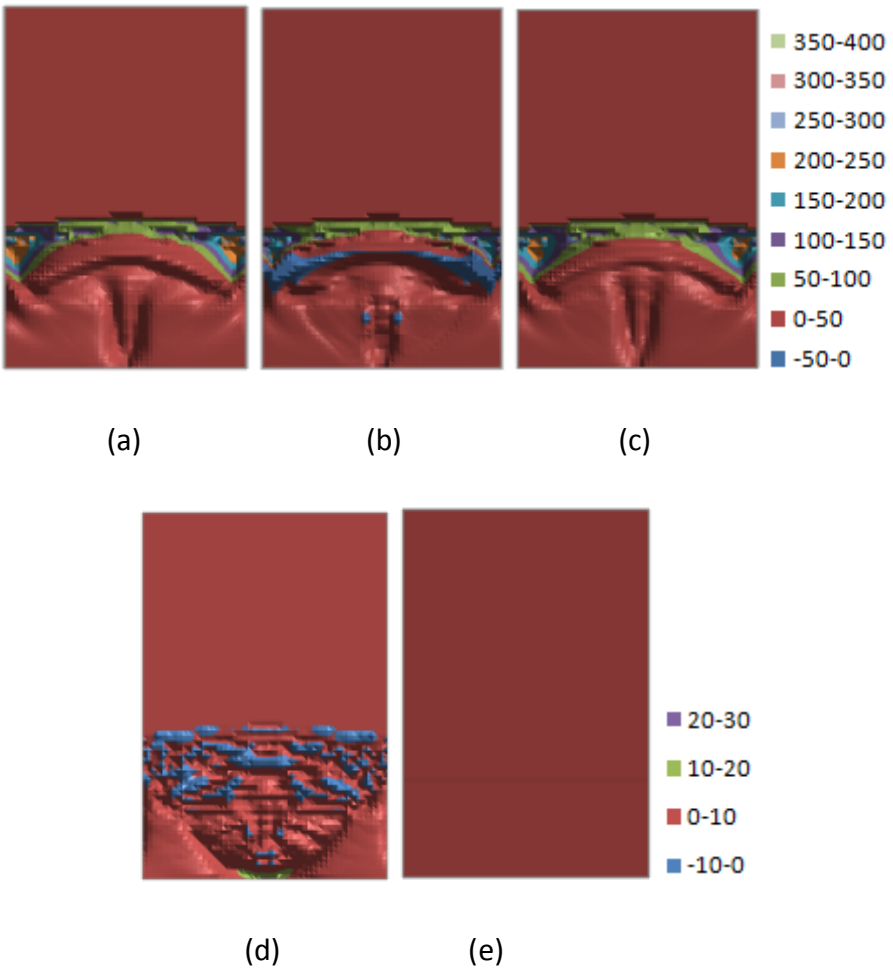


Figure 40 illustrates various partial POD approximations for the solids temperature, θ_s, at time t=0s. Specifically, (a) illustrates the POD variable mean, (b) denotes the contribution of the first POD approximation plus the mean and (c) denotes the contribution of the second POD

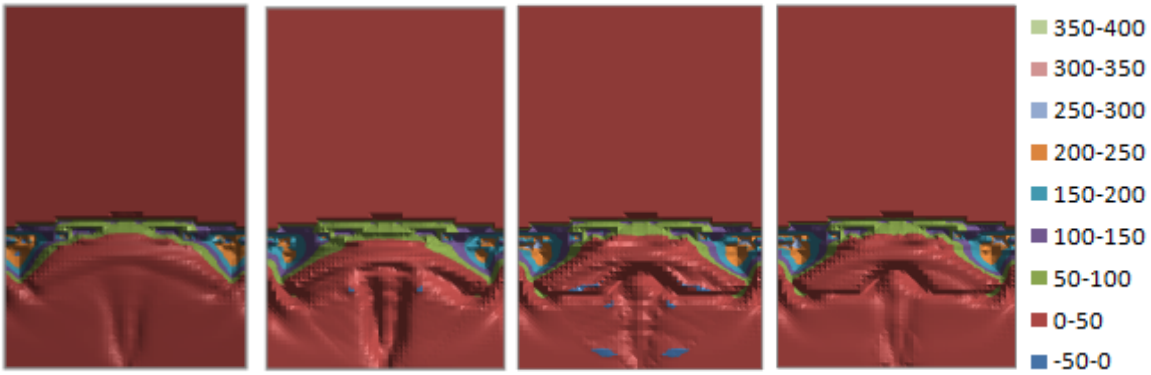
approximation plus the mean. Image (d) illustrates the full POD approximation which required 122 POD modes and basis functions to capture 99% of the energy of the MFIX model. Image (e) illustrates the MFIX model's approximation of θ_s at time t=0s.

Figure 40: θ_s at time t=0s (m^2/s^2)



The images in Figure 41 illustrate approximations for the temperature of the solid particles, θ_s, at time t=5s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of θ_s at time t=5s. The following two sets of figures exhibit fairly strong agreement between the POD ROM representation and the MFIX data set. The areas exhibiting the highest solid granular temperature exhibit good agreement in magnitude.

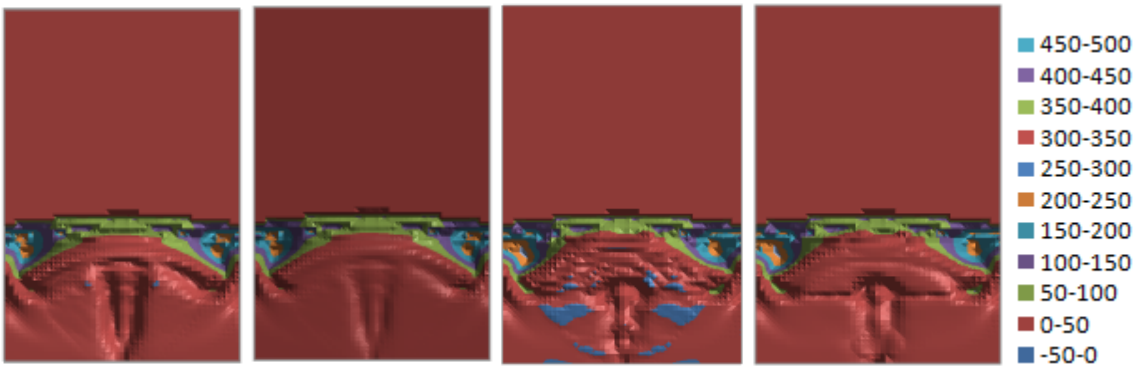
Figure 41: θ_s at time t=5s (m^2/s^2)



(a) (b) (c) (d)

The images in Figure 42 illustrate similar approximations for the temperature of the solid particles, θ_s, at time t=10s. Specifically, (a) denotes the contribution of the first POD approximation plus the mean and (b) denotes the contribution of the second POD approximation plus the mean. Finally (c) illustrates the full POD approximation while (d) illustrates the MFIX model's approximation of θ_s at time t=10s.

Figure 42: θ_s at time t=10s (m^2/s^2)

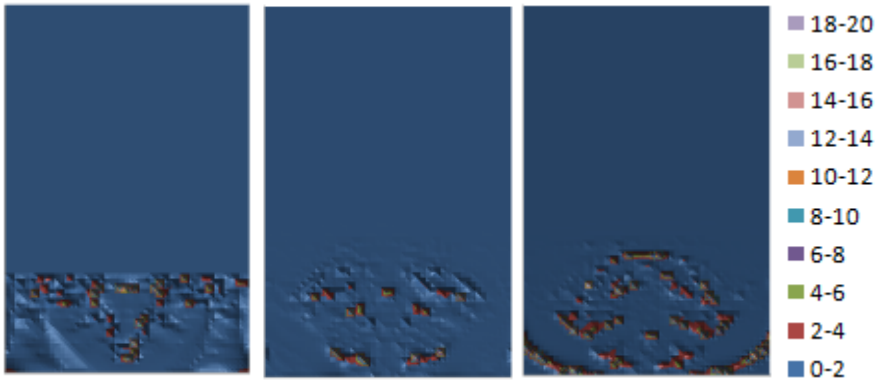


(a) (b) (c) (d)

Figure 43 images show the relative error between the POD approximations and the MFIX model at various times for the solid particle temperature, θ_s. Image (a) illustrates the relative error at 0s while (b) and (c) depict the relative error between the models at 5s and 10s respectively. The areas exhibiting the largest error tend to be along the boundaries of the spout at the initial time

step and the bottom right corners at the later time step as well as the central jet wall line which takes the shape of an inverted cone.

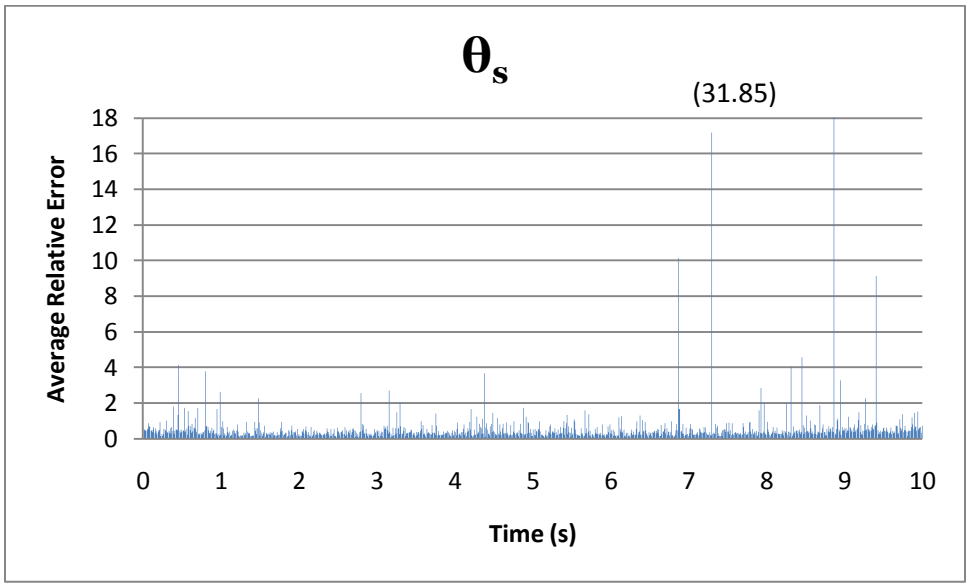
Figure 43: θ_s relative error



(a) (b) (c)

Figure 44 provides the average relative error across the time span of the reduced order model for the temperature of the solid particles, θ_s.

Figure 44: θ_s average relative error



Recall the objective of reduced order model development is ultimately to develop a model which reduces the order of the system used to approximate a solution. In this particular

case, the identification of a sufficient number of basis functions through POD methodology allows the capture of 99% of the energy for each variable identified in the Navier Stokes momentum equations for gas and solids. This has at most 0.01 relative error to the MFIX model. Because of these requirements, POD methodology allows for the reduction in the dimension of an accurate capture of the full MFIX model requiring n = 1,001 modes using all of the POD basis functions (eigenvectors) to a near approximation of the MFIX model. Only 7 POD basis functions were required in the case of gas pressure. Significantly more POD basis functions and modes were required for 99% total energy capture of the gas and solid velocity approximations, 72 and 85. Respectively, these numbers still represent less than 9% of the amount of basis functions needed for a full model capture in MFIX. The largest number of POD modes (159) required for a reduced order approximation was for the pressure of solid particles which still is an enormous reduction (84%) of the full order model formulation.

5.2 Projective Solution Model

Data for validation of the projection methodology at time points later than 10s were not available from the same version of the MFIX program used to create the POD basis functions. As a result, the newest version of MFIX was used to generate the comparison data. Thus, a quantification of the direct comparison is not valid as illustrated in the differences between images k and l in Figures 45-52.

The figures following in this section illustrate the results of the projected solution past some known time sequence. In this particular case, the results depict the extension of the solution from the original MFIX time sequence which ended at 10s. The images in (a), (c), (e), (g), and (i) in Figures 45-52 come from the POD ROM solution which are to be compared to their respective MFIX solution from a later run which extended the solution time sequence to 15s. The images in (b), (d), (f), (h), and (j) in Figures 45-52depict these solutions in MFIX. Images (a) and (b) illustrate the solution at 11s. Images (c) and (d) illustrate the solution at 12s. Images (e) and (f) illustrate the solution at 13s. Images (g) and (h) illustrate the solution at 14s. Finally, the 15s solution is shown in (i) and (j). Images (k) and (l) illustrate the change in the MFIX solution at 10s between the two separate runs (the original which ran to 10 seconds only on which the POD basis was constructed and the verification run which concluded at 15 seconds). Image (k) was the solution at 10s from the original run and image (l) was at 10s from the verification run.

As mentioned in previous sections, one of the shortcomings of the POD basis functions to predict future solutions is that if physical characteristics are not present in the data set used to generate the POD basis functions then it is very unlikely they will be present in any projected data. This is evidenced in some of the variables for the projection results. In particular, the projection results of the x component of the gas velocity, U_g, exhibits only slight change as time progresses in the freeboard region, the area above the top of the spouting bed, while the MFIX solution exhibits regions with both higher and lower velocities in several regions. Also, the velocities in the freeboard region for the MFIX comparison data set are more extreme than at any time point present in the initial 10 seconds worth of data used to generate the POD basis

functions. The projection results of the y component of the gas velocity, V_g, exhibit similar inconsistencies. This trend is observable in the remaining variables.

Another confounding factor related to the solution methodology is that the approximation error of the solution is distributed throughout every variable without enforcement of sufficient agreement on a per variable basis before the solution is accepted. For example, the 2nd POD coefficient contributes 20.15% matrix energy but the 97 POD coeffient for solids pressure contributes only 0.21% matrix energy. In the solution process, however, they are weighted equally important. This is further confounded by forcing a solution on a heavily over-determined system without recognizing that each equation in each cell is weighted equally. Excellent agreement in the freeboard region for all but the modes for the gas velocity variables reduces the necessary convergence criteria of the solution in the spouting region. Since this is true for every variable but the gas velocities this increases the likelihood that the freeboard region will change very little at subsequent iterations as the approximations are sufficiently close to balancing the governing equations and therefore result in smaller residuals. Thus, a sufficiently large number of unknown POD modes exhibiting sufficiently good agreement in a majority of the cells of the governing equations overcomes the disagreement of the governing equations in the spouting region resulting in the criteria for iteration convergence being met. Note that the solid particle pressure and temperature have the largest number of POD modes and together represent nearly half of all POD modes to be determined at each time step. Since their mode approximations converge fairly quickly to a "near" approximation, it isn't surprising that the convergence criteria was met before the gas velocities and changes in "small" regions of the solid particle velocities were not identified. This effect can be controlled and subsequently reduced by adjusting the tolerance of the Levenberg-Marquardt iterations, but this will impact overall calculation time.

The biggest changes are visible in the void fraction variable found in Figure 49. This is due largely to the "overfilling" or the lack of enforcement of the void fraction cap in these models. Note that there become regions containing more than 100% of the solid particles as indicated by the "red" regions while some regions now contain more than 100% of the gas which is indicated by the pink and periwinkle (light purplish blue) shading. One other change which is also discernible is the variation in the x component of the gas velocity found in Figure 45. The

right-most red boundary in the freeboard region above the spouted bed appears to be slowly drifting right and its boundary becomes more jagged and irregular as time progresses.

Figure 45: **U_g (cm/s) Projection Results**

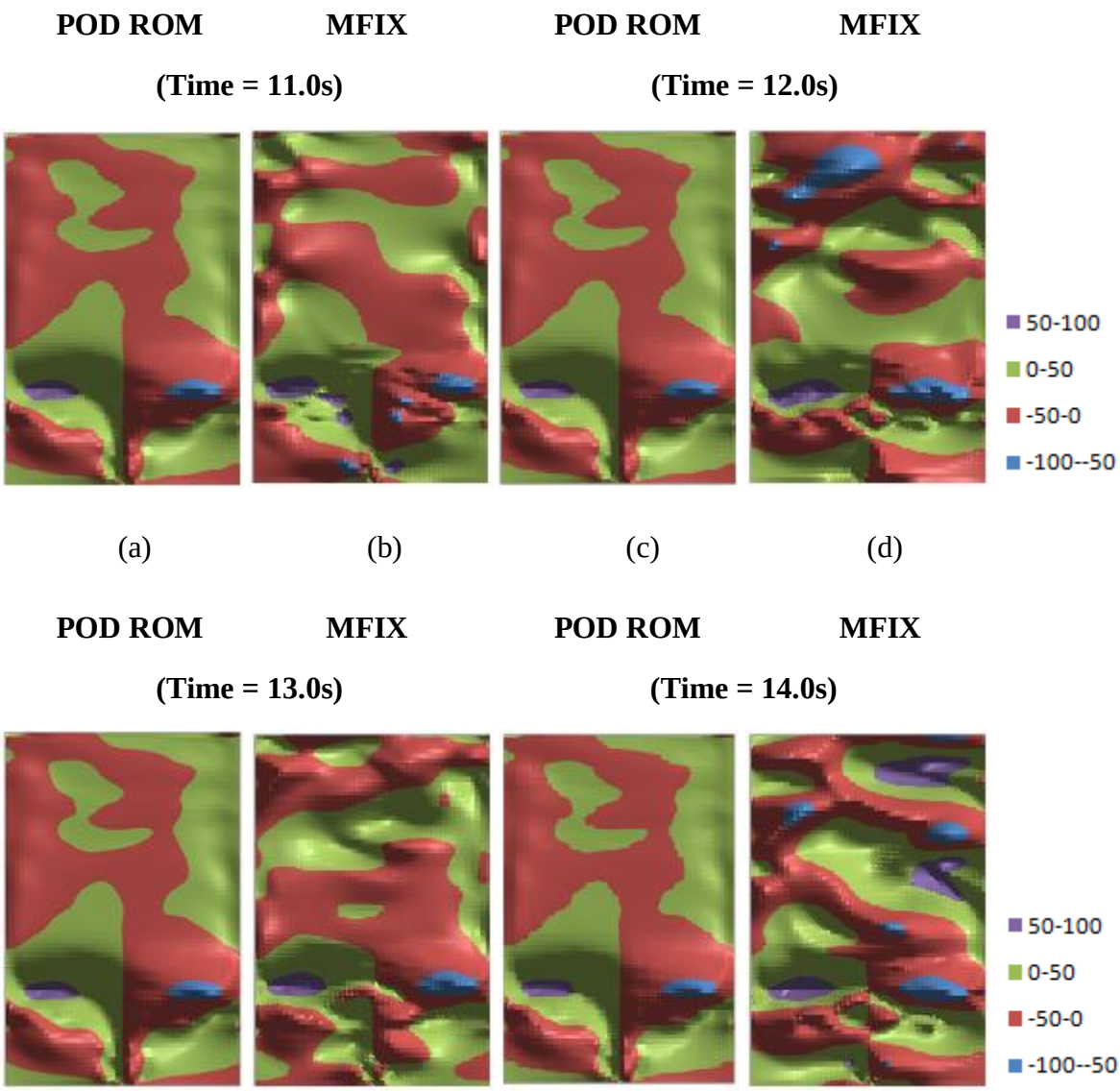
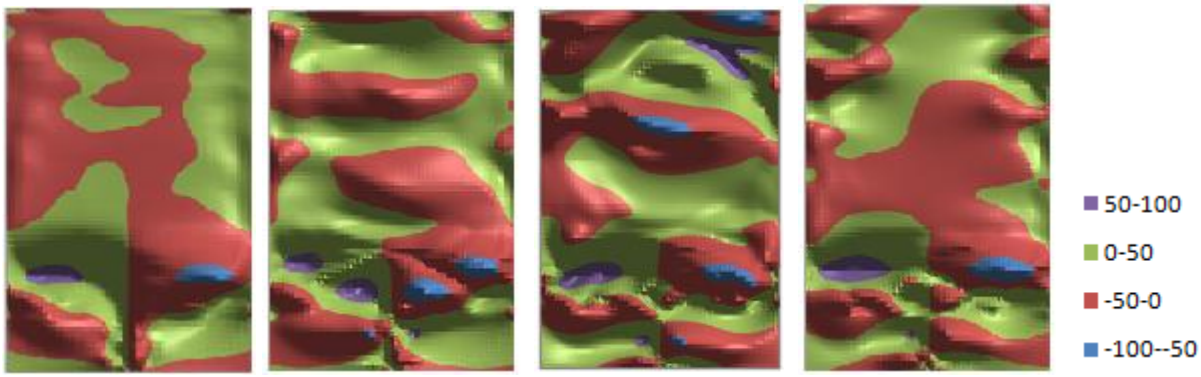


Figure 45 (cont): **U_g (cm/s) Projection Results**

POD ROM **MFIX** **Original** **Verification**

(Time = 15.0s) **(MFIX Time = 10.0s)**

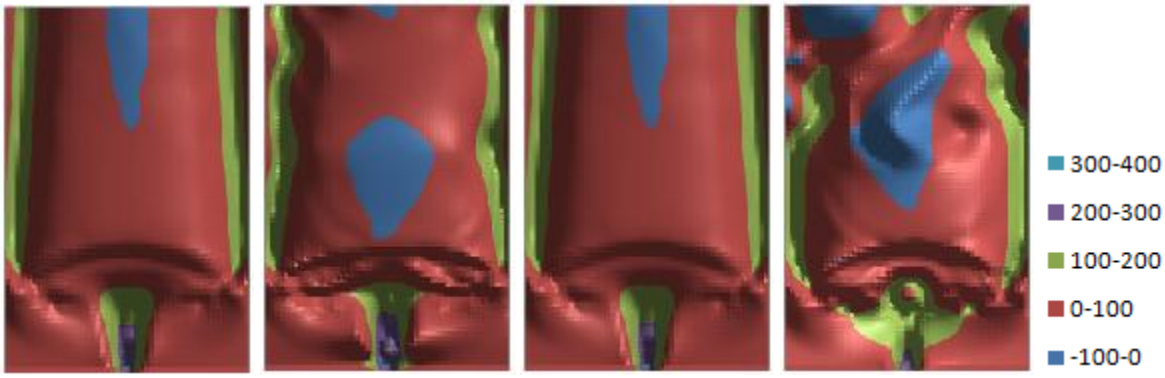


(i) (j) (k) (l)

Figure 46: **V_g (cm/s) Projection Results**

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 11.0s) **(Time = 12.0s)**

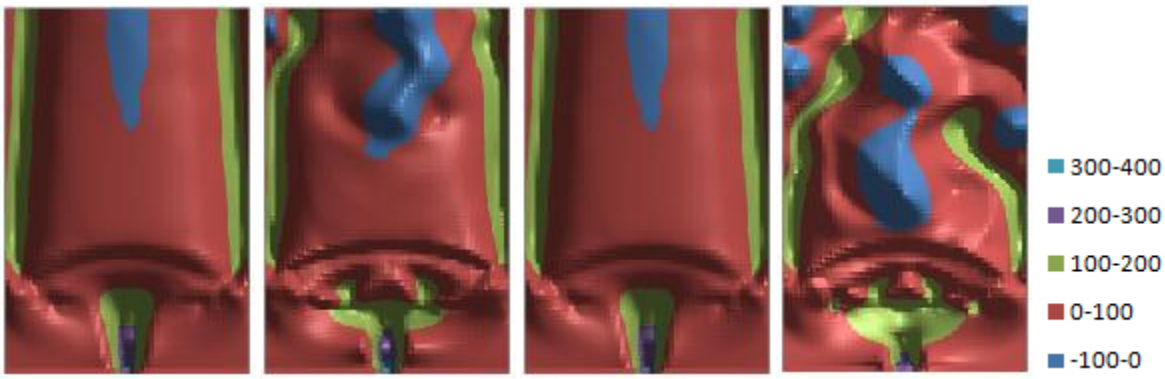


(a) (b) (c) (d)

Figure 46 (cont): V_g **(cm/s) Projection Results**

POD ROM **MFIX** **POD ROM** **MFIX**

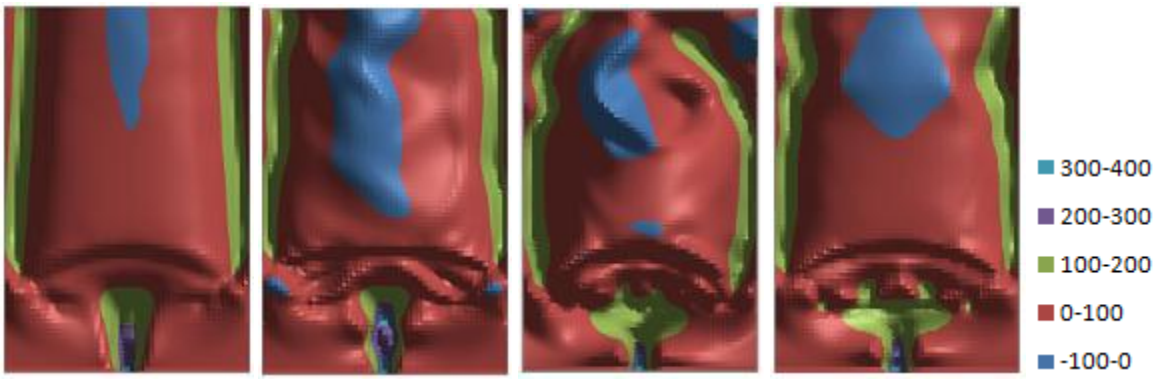
(Time = 13.0s) **(Time = 14.0s)**



(e) (f) (g) (h)

POD ROM **MFIX** **Original** **Verification**

(Time = 15.0s) **(MFIX Time = 10.0s)**

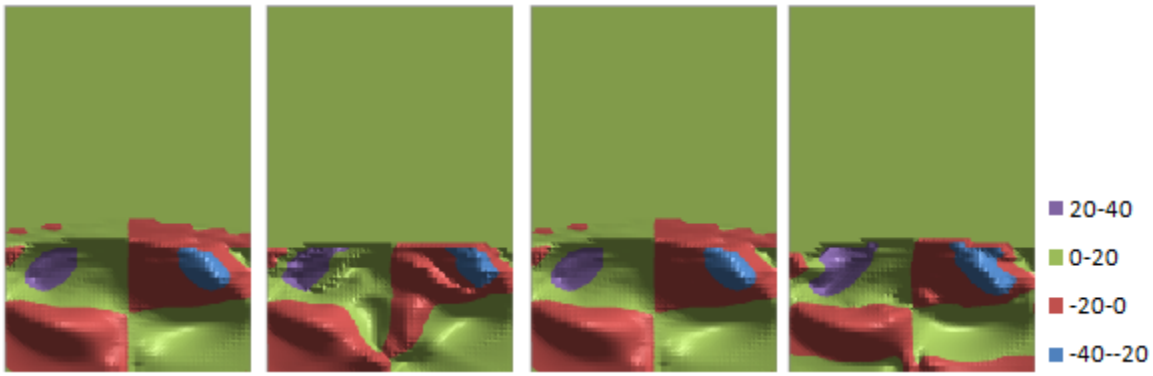


(i) (j) (k) (l)

Figure 47: U_s **(cm/s) Projection Results**

POD ROM **MFIX** **POD ROM** **MFIX**

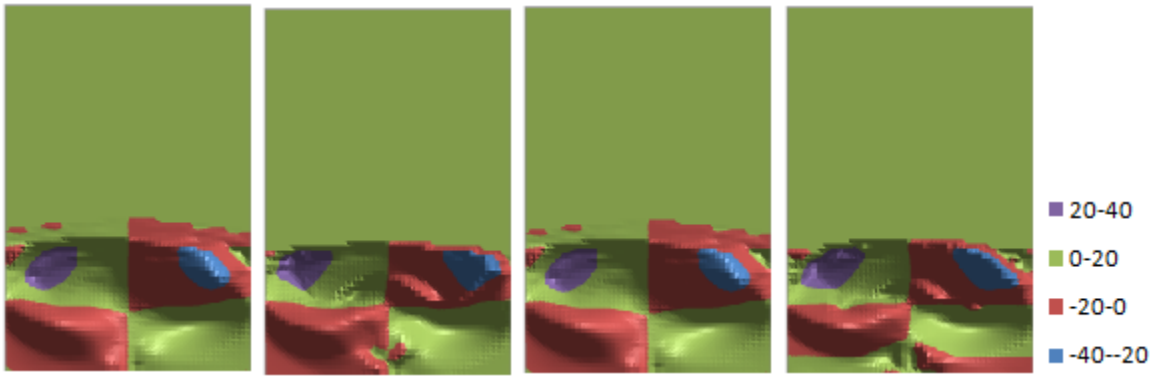
(Time = 11.0s) **(Time = 12.0s)**



(a) (b) (c) (d)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 13.0s) **(Time = 14.0s)**



(e) (f) (g) (h)

Figure 47 (cont): U_s (cm/s) Projection Results

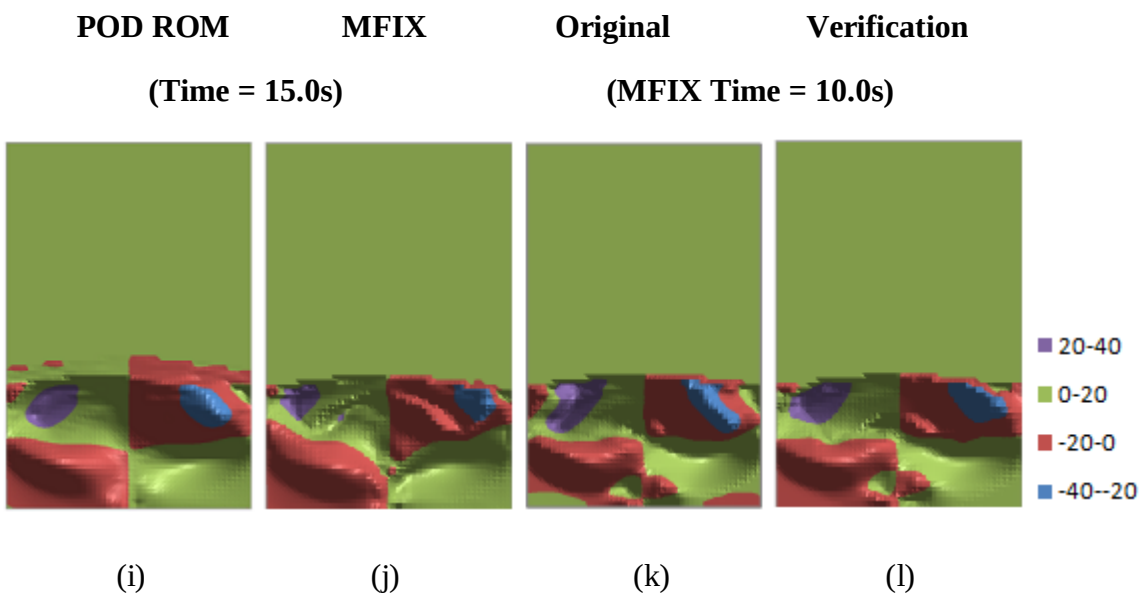


Figure 48: V_s (cm/s) Projection Results

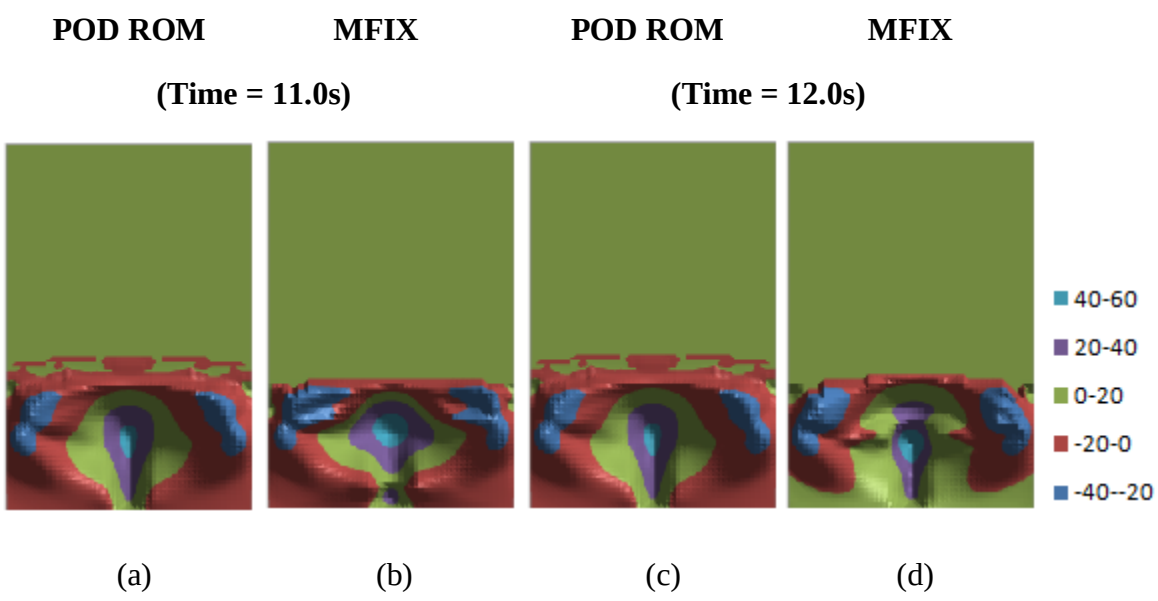
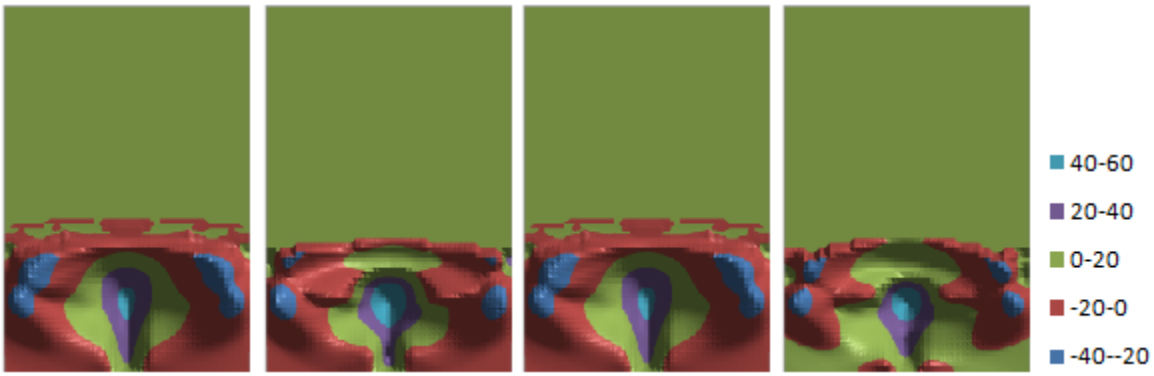


Figure 48 (cont): V_s **(cm/s) Projection Results**

POD ROM **MFIX** **POD ROM** **MFIX**

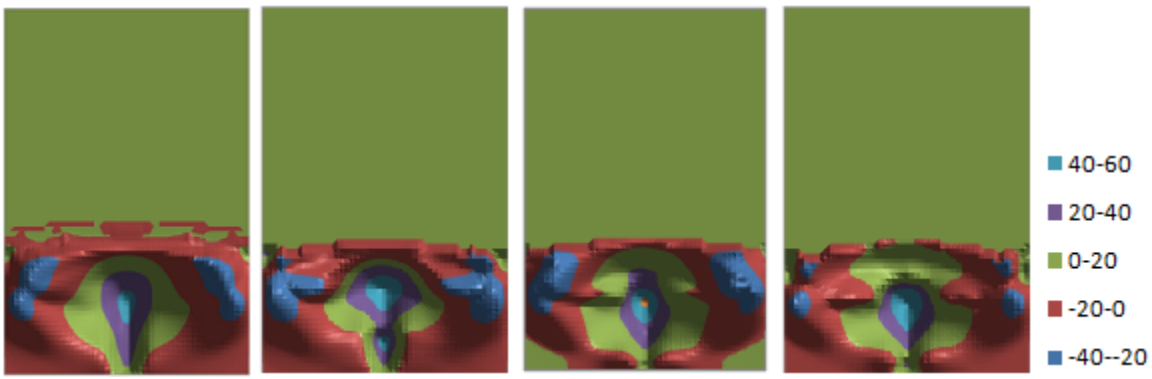
(Time = 13.0s) **(Time = 14.0s)**



(e) (f) (g) (h)

POD ROM **MFIX** **Original** **Verification**

(Time = 15.0s) **(MFIX Time = 10.0s)**

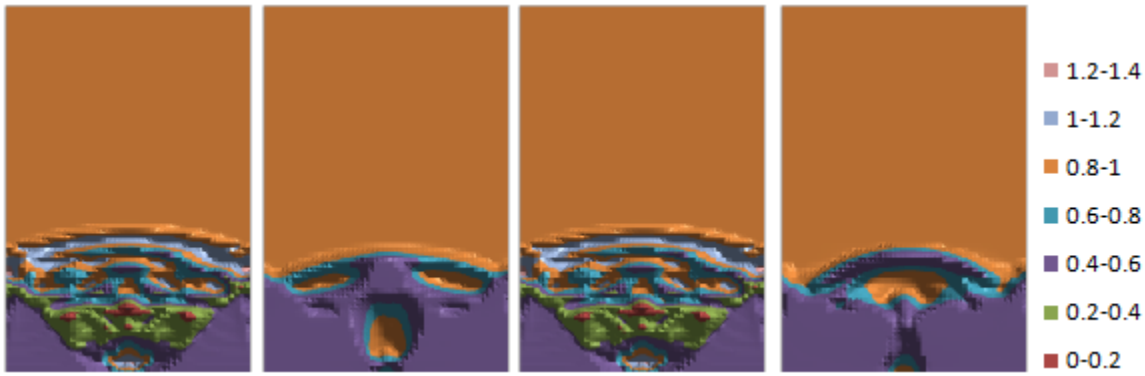


(i) (j) (k) (l)

Figure 49: ϵ_g **Projection Results**

POD ROM **MFIX** **POD ROM** **MFIX**

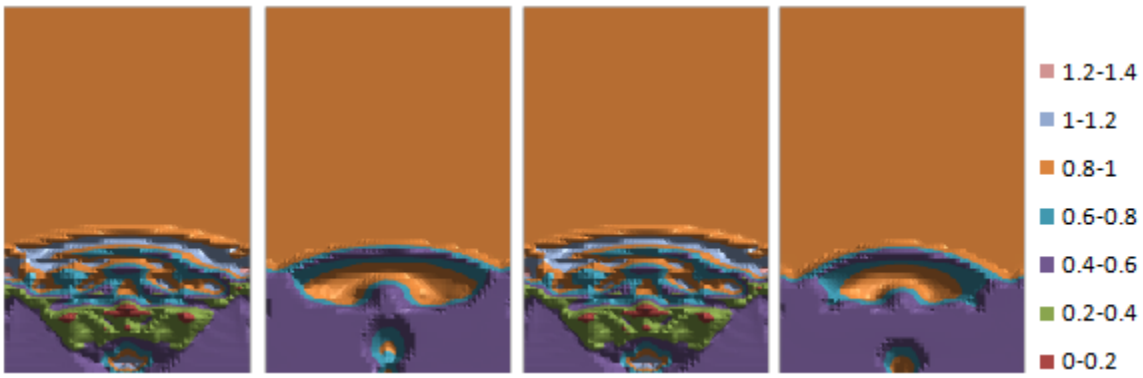
(Time = 11.0s) **(Time = 12.0s)**



(a) (b) (c) (d)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 13.0s) **(Time = 14.0s)**



(e) (f) (g) (h)

Figure 49 (cont): ϵ_g **Projection Results**

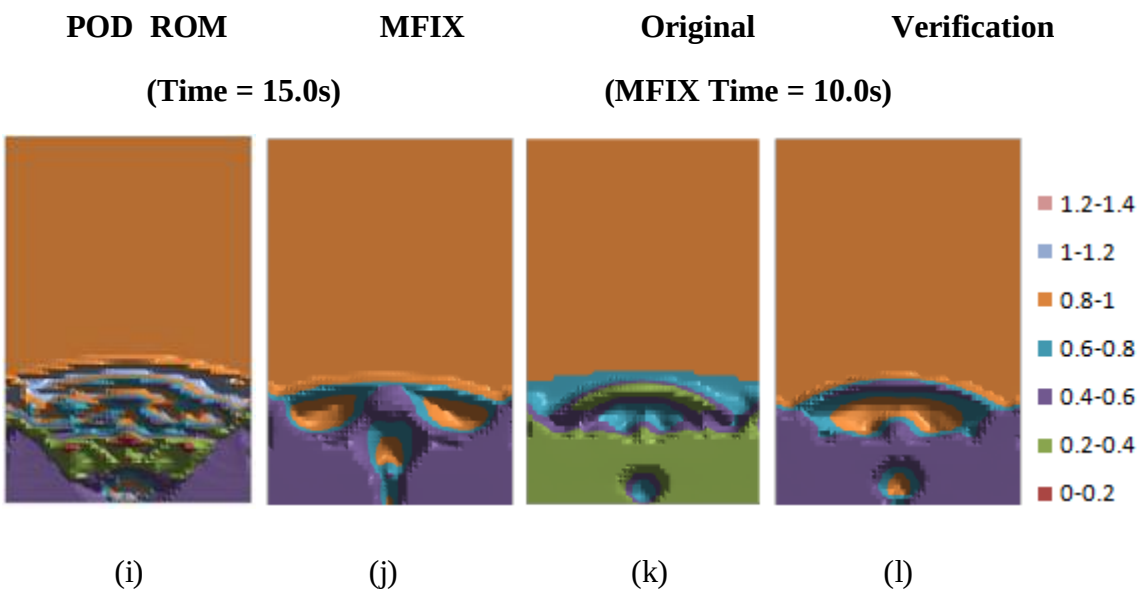


Figure 50: P_g **(Pa) Projection Results**

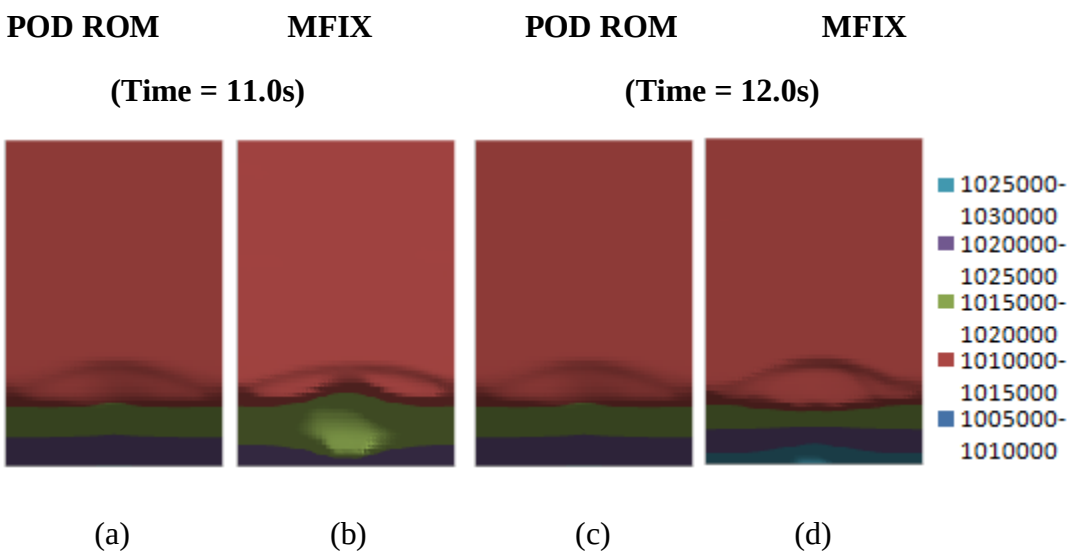
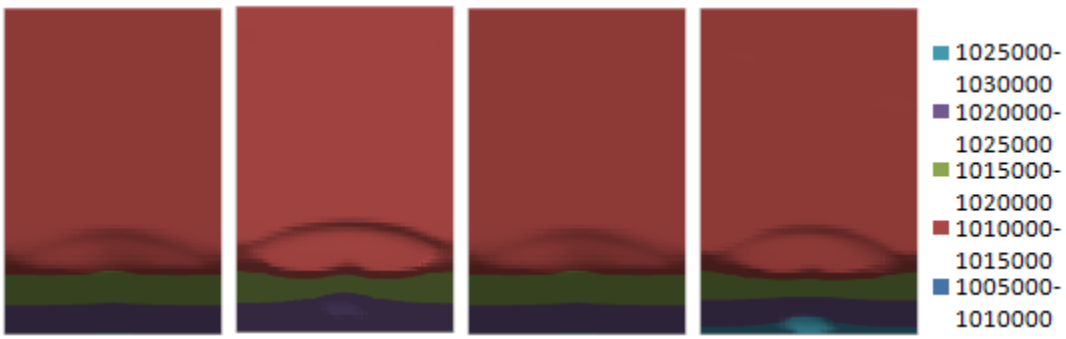


Figure 50 (cont): P_g **(Pa) Projection Results**

POD ROM **MFIX** **POD ROM** **MFIX**

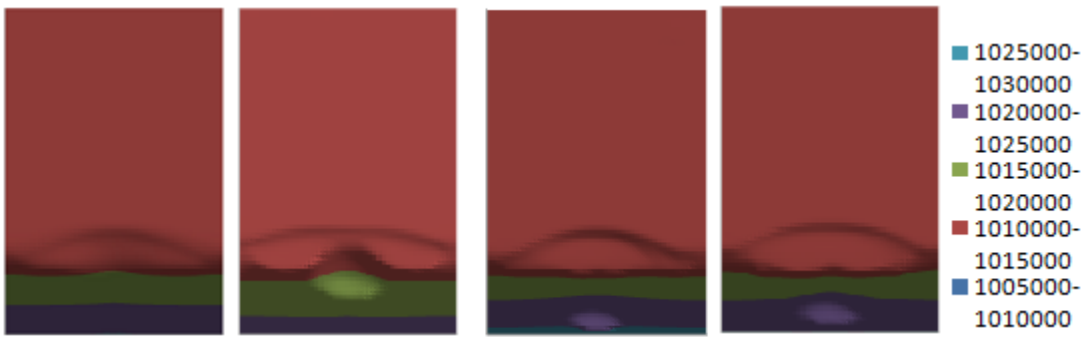
(Time = 13.0s) **(Time = 14.0s)**



(e) (f) (g) (h)

POD ROM **MFIX** **Original** **Verification**

(Time = 15.0s) **(MFIX Time = 10.0s)**

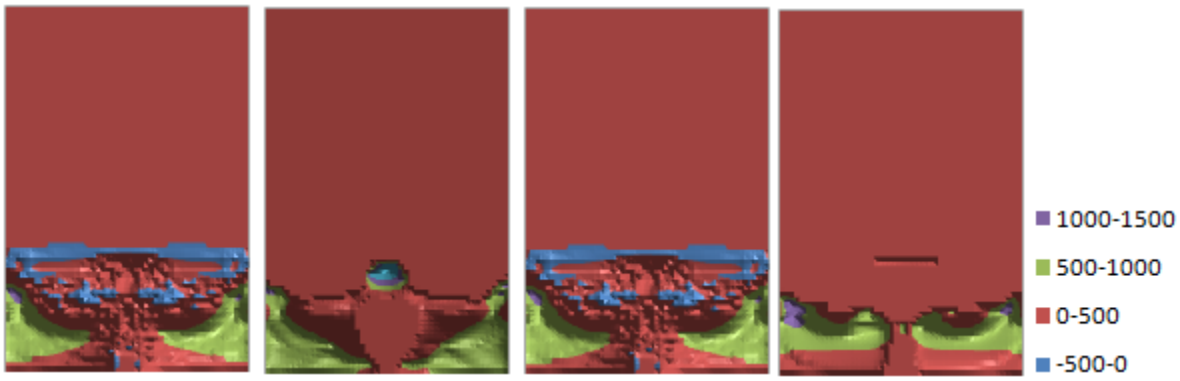


(i) (j) (k) (l)

Figure 51: P_s **(Pa) Projection Results**

POD ROM **MFIX** **POD ROM** **MFIX**

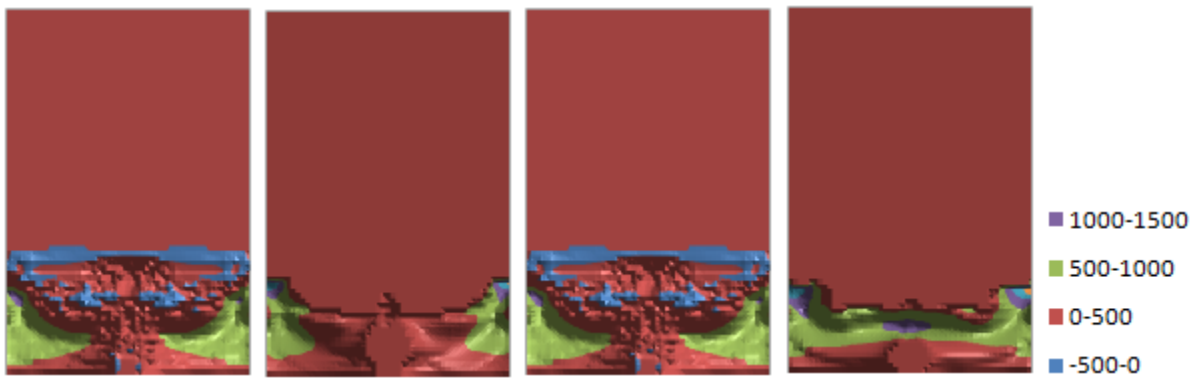
(Time = 11.0s) **(Time = 12.0s)**



(a) (b) (c) (d)

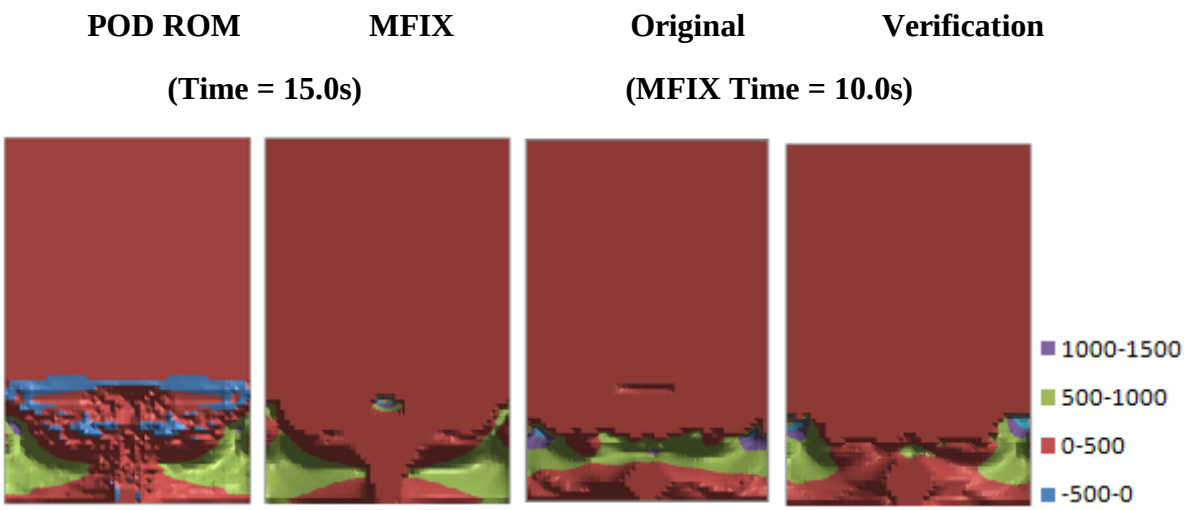
POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 13.0s) **(Time = 14.0s)**



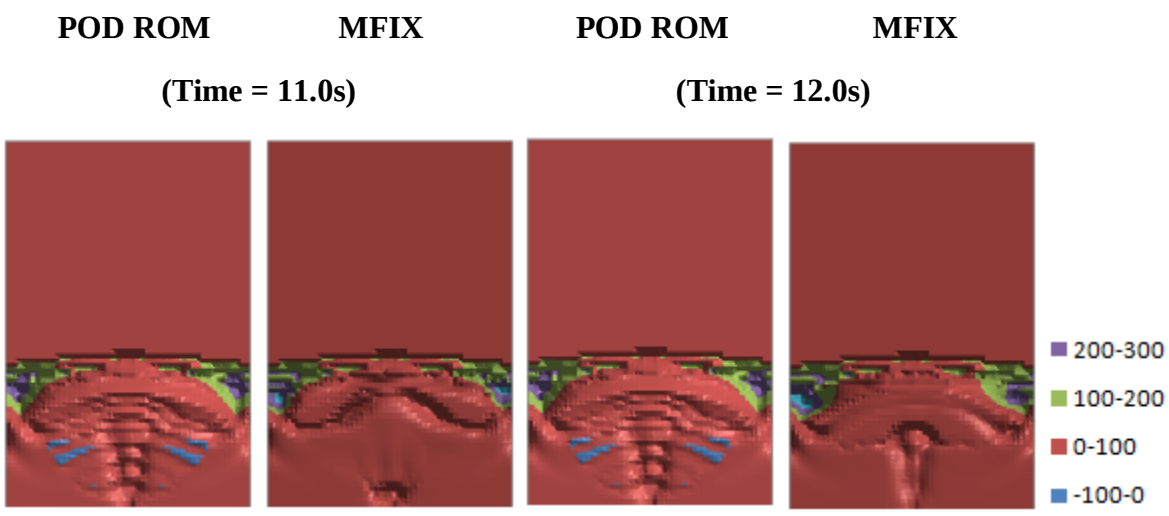
(e) (f) (g) (h)

Figure 51 (cont): P_s **(Pa) Projection Results**



(i) (j) (k) (l)

Figure 52: θ_s **(m^2/s^2) Projection Results**

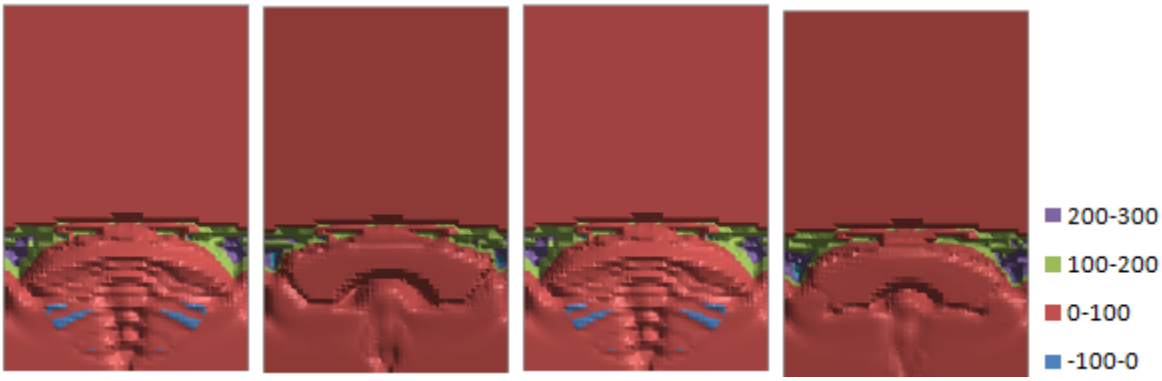


(a) (b) (c) (d)

Figure 52 (cont): θ_s (m^2/s^2) **Projection Results**

POD ROM **MFIX** **POD ROM** **MFIX**

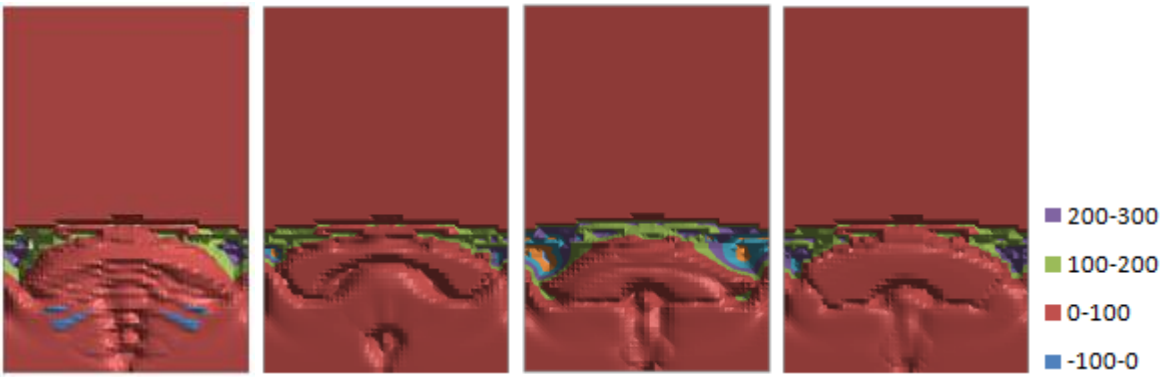
(Time = 13.0s) **(Time = 14.0s)**



(e) (f) (g) (h)

POD ROM **MFIX** **Original** **Verification**

(Time = 15.0s) **(MFIX Time = 10.0s)**



(i) (j) (k) (l)

5.3 Extrapolation of Parameter Model

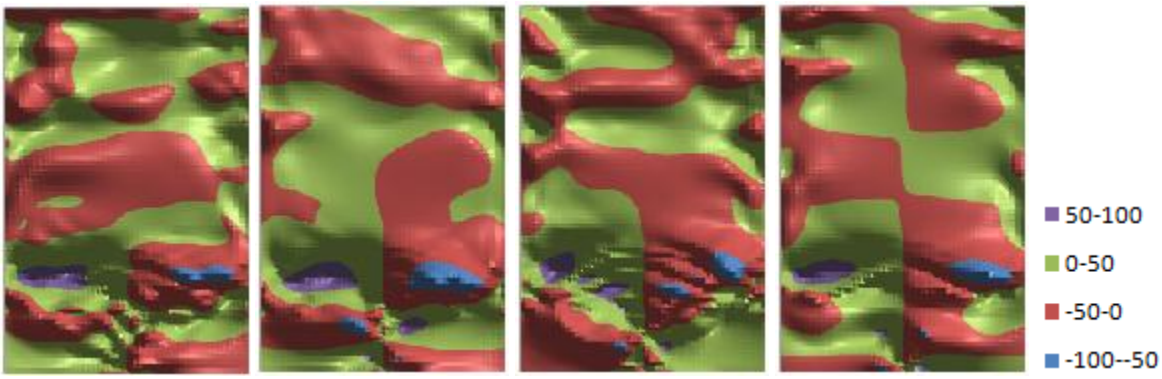
The figures in this section illustrate the extrapolation of a solution with a slight change in parameter from some known solution. Figures 53-60 highlight the results of an extrapolation from the original MFIX simulation which had the coefficient of restitution set at 0.8 to the value of 0.825. The images in (a), (c), (e), (g), (i), and (k) come from the POD ROM solution which are to be compared to their respective MFIX solution. The images in (b), (d), (f), (h), (j), and (l) depict these solutions in MFIX. Images (a) and (b) illustrate the solution at 1s. Images (c) and (d) illustrate the solution at 2s. Images (e) and (f) illustrate the solution at 3s. Images (g) and (h) illustrate the solution at 4s. Finally, the 5s solution is shown in (i) and (j) and the 10s solution is shown in (k) and (l).

In this particular model as evidenced in Figure 57, even though no restriction was placed upon gas void fraction it exhibited no underflow though overflow was present at several time points. In general, the agreement between the POD ROM extrapolation and the MFIX results tends to greatly improve around the 3 second mark. It appears the extrapolation of the x component of gas velocity is slightly behind the MFIX solution. As evidenced by the spouting region comparison between the 2, Figure 53 images (c) and (d), and 3, Figure 53 images (e) and (f), second marks. Solid particle velocity approximations in Figures 55 and 56 appear to be within 20 cm/s agreement for the examined time periods in a majority of the cells. The solid particle pressure and temperature regions exhibiting disagreement between the POD ROM and MFIX calculations are only slight. These occur predominantly in the spouting region and have only small pockets of slightly negative values. This is due in part to the existence of solid particles in the POD ROM approximations which are not present in the MFIX approximations.

Figure 53: U_g **(cm/s) Extrapolation Results for Coefficient of Restitution**

POD ROM **MFIX** **POD ROM** **MFIX**

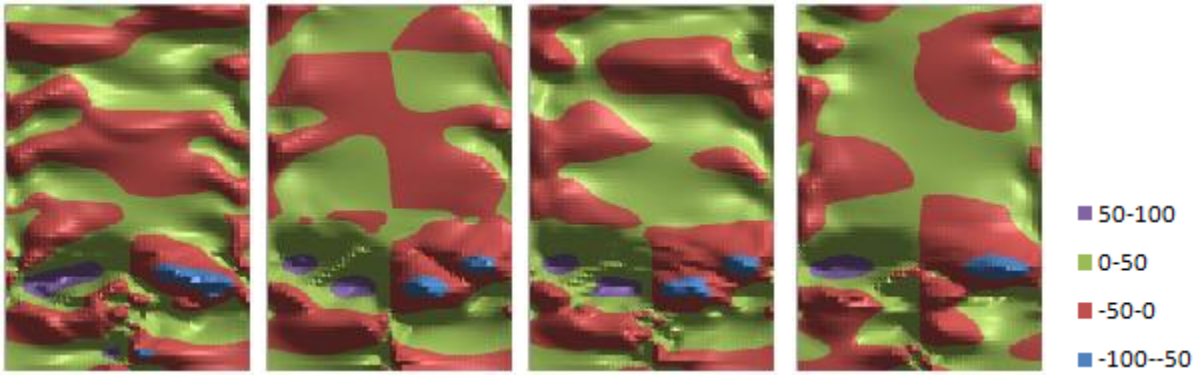
(Time = 1.0s) **(Time = 1.0s)**



(a) (b) (c) (d)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 2.0s) **(Time = 3.0s)**



(e) (f) (g) (h)

Figure 53 (cont): **U_g (cm/s) Extrapolation Results for Coefficient of Restitution**

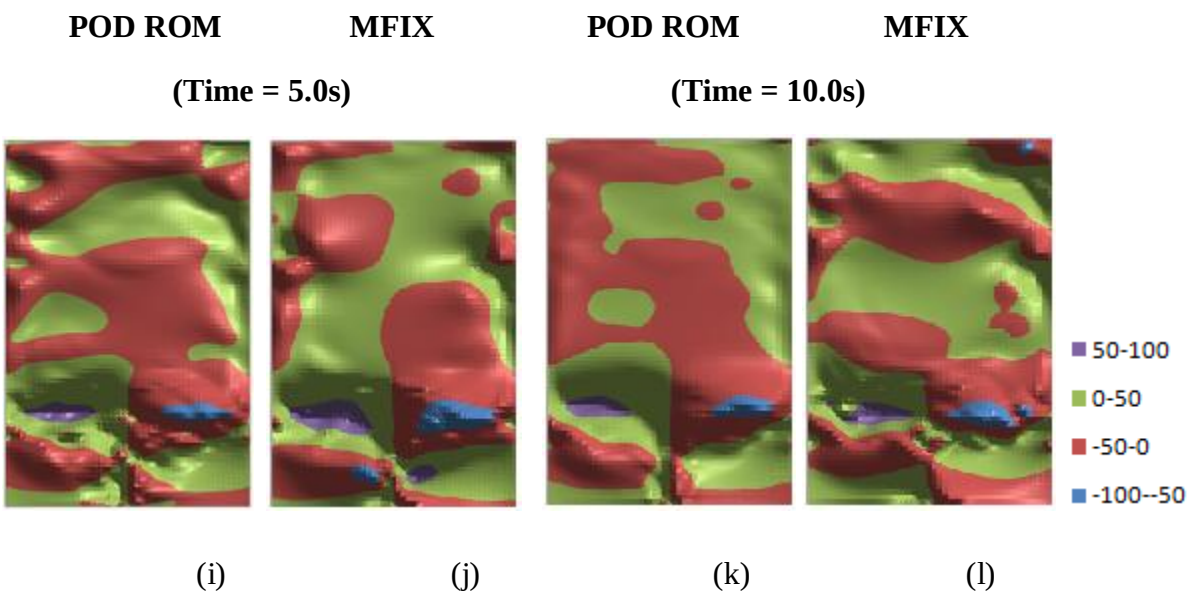


Figure 54: **V_g (cm/s) Extrapolation Results for Coefficient of Restitution**

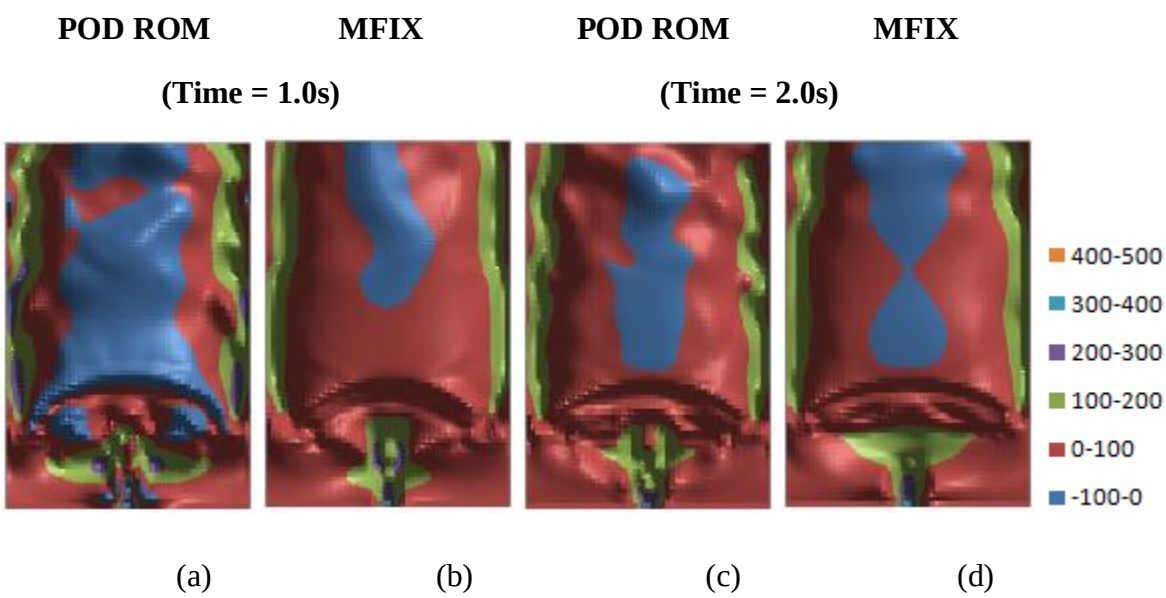
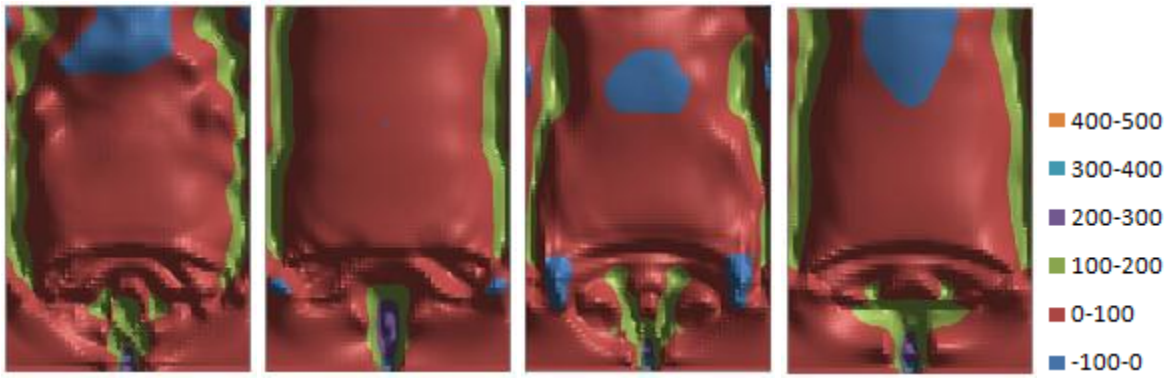


Figure 54 (cont): V_g **(cm/s) Extrapolation Results for Coefficient of Restitution**

POD ROM **MFIX** **POD ROM** **MFIX**

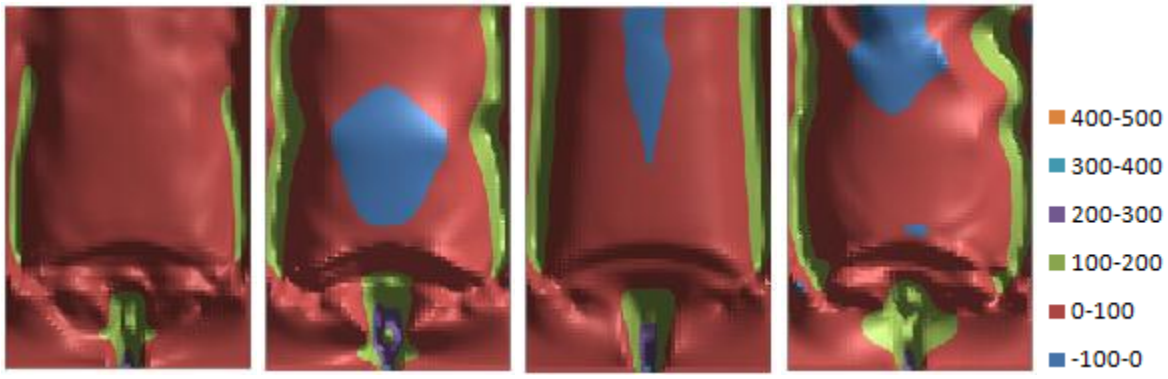
(Time = 3.0s) **(Time = 4.0s)**



(e) (f) (g) (h)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 5.0s) **(Time = 10.0s)**

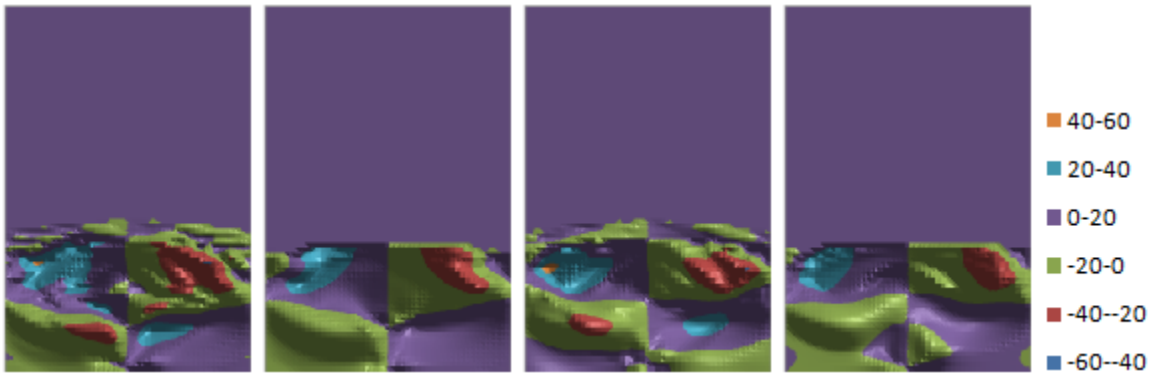


(i) (j) (k) (l)

Figure 55: U_s **(cm/s) Extrapolation Results for Coefficient of Restitution**

POD ROM **MFIX** **POD ROM** **MFIX**

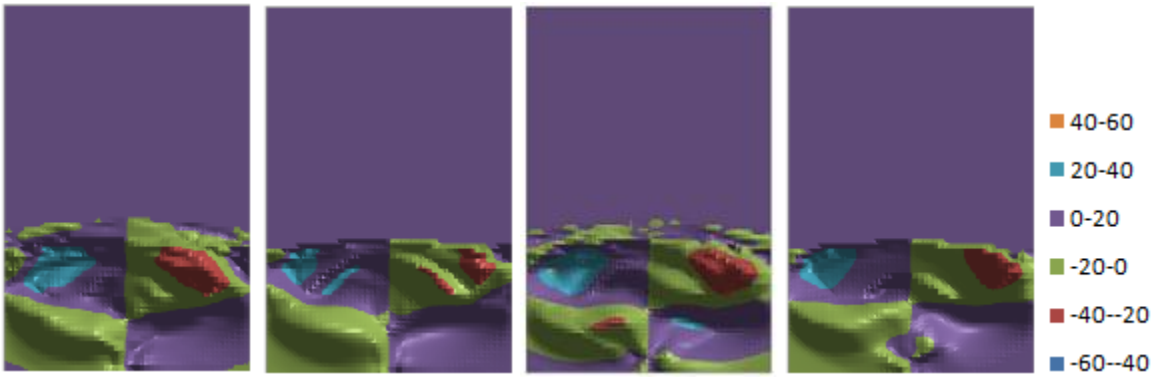
(Time = 1.0s) **(Time = 2.0s)**



(a) (b) (c) (d)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 3.0s) **(Time = 4.0s)**

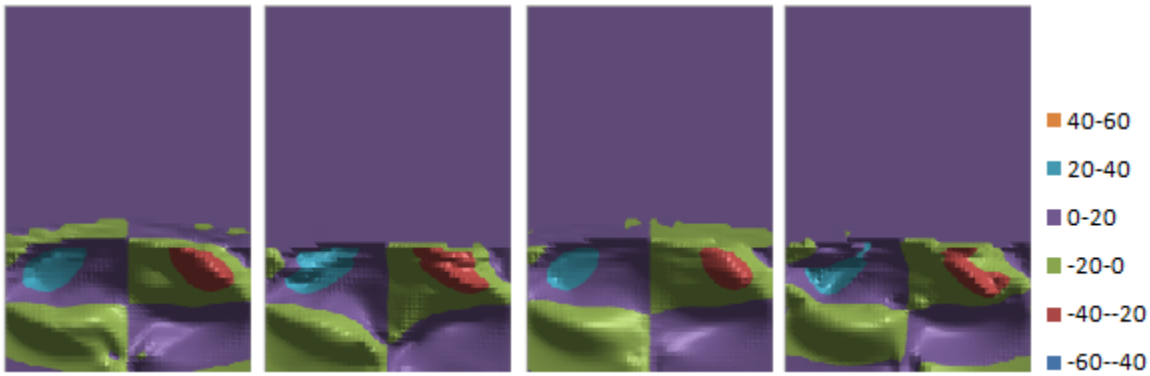


(e) (f) (g) (h)

Figure 55 (cont): **U_s (cm/s) Extrapolation Results for Coefficient of Restitution**

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 5.0s) **(Time = 10.0s)**

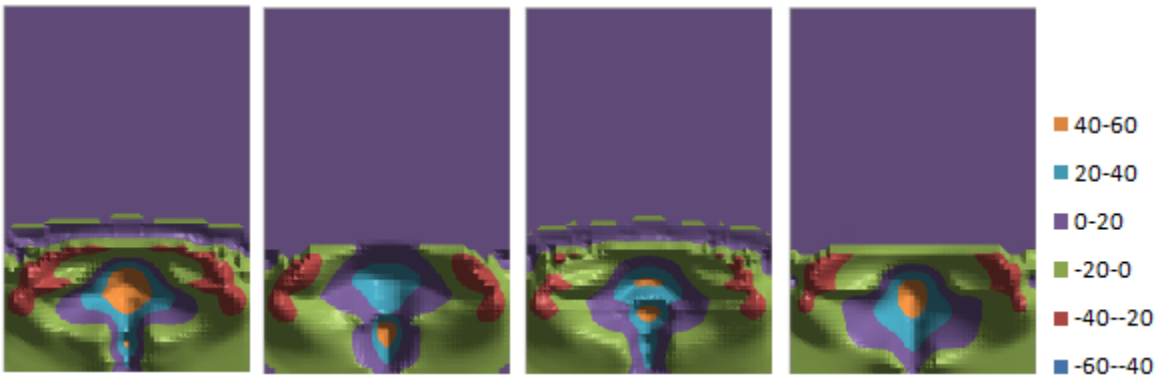


(i) (j) (k) (l)

Figure 56: **V_s (cm/s) Extrapolation Results for Coefficient of Restitution**

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 1.0s) **(Time = 2.0s)**

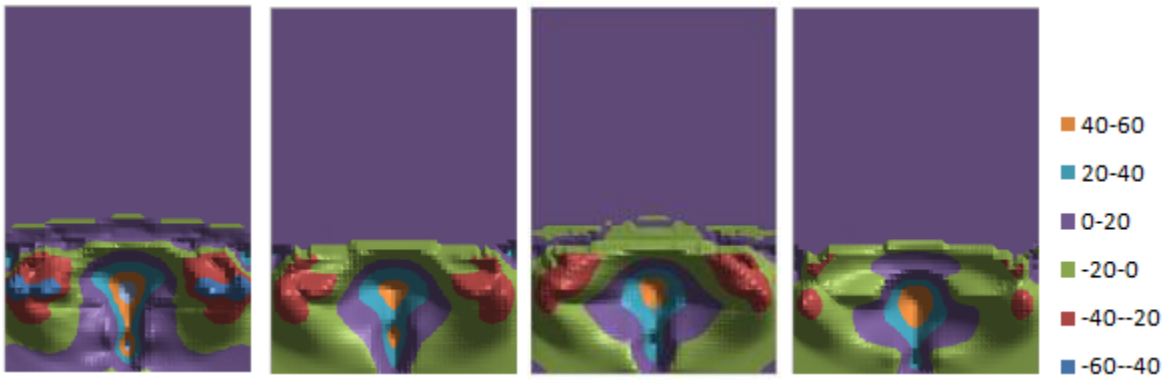


(a) (b) (c) (d)

Figure 56 (cont): V_s **(cm/s) Extrapolation Results for Coefficient of Restitution**

POD ROM **MFIX** **POD ROM** **MFIX**

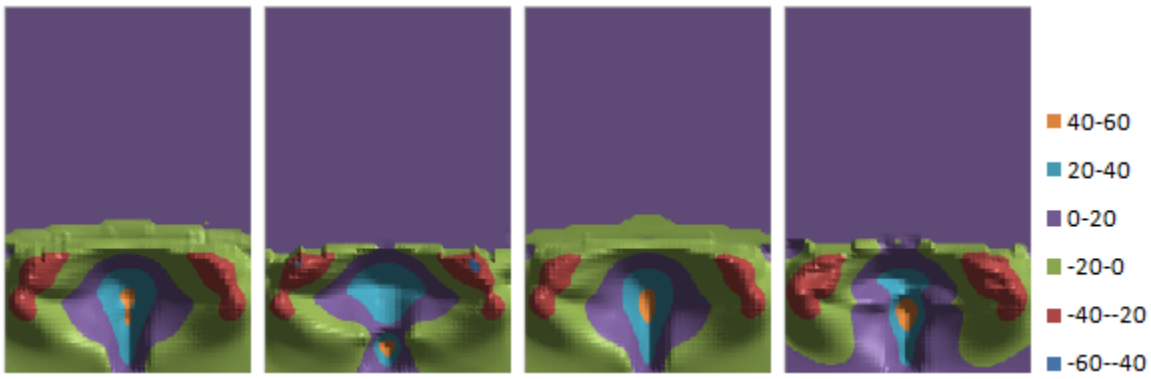
(Time = 3.0s) **(Time = 4.0s)**



(e) (f) (g) (h)

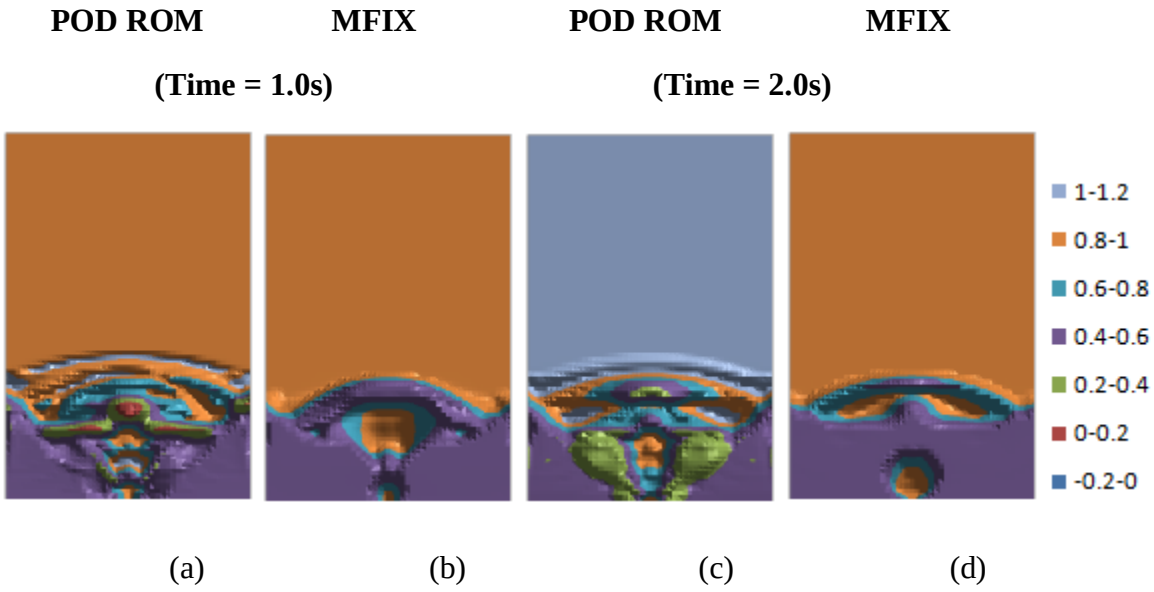
POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 5.0s) **(Time = 10.0s)**



(i) (j) (k) (l)

Figure 57: ϵ_g **Extrapolation Results for Coefficient of Restitution**



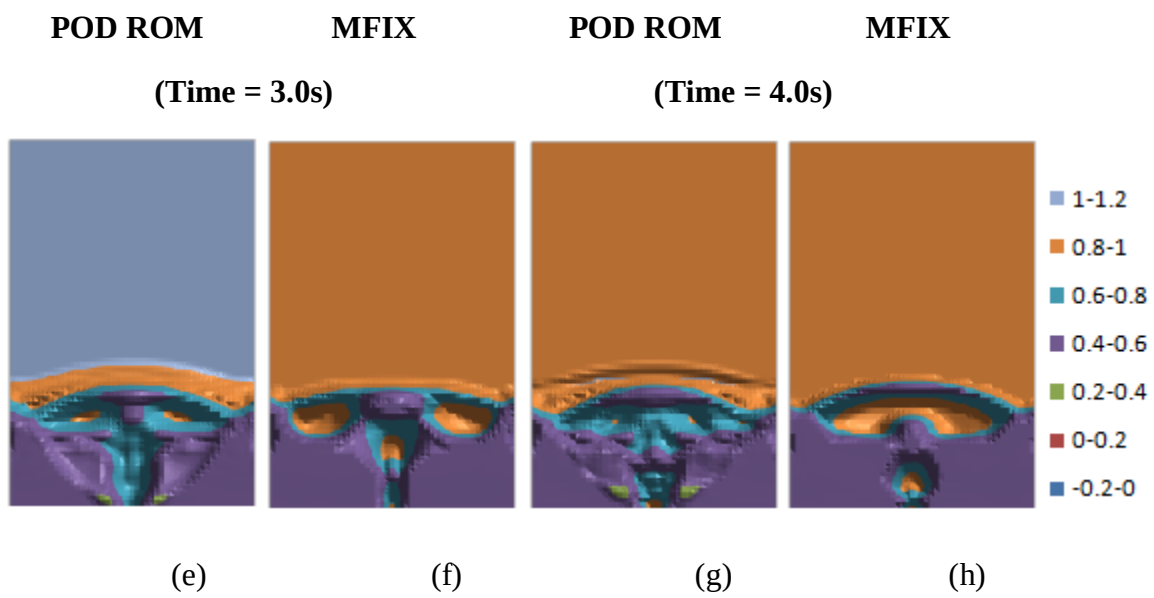


Figure 57 (cont): ϵ_g **Extrapolation Results for Coefficient of Restitution**

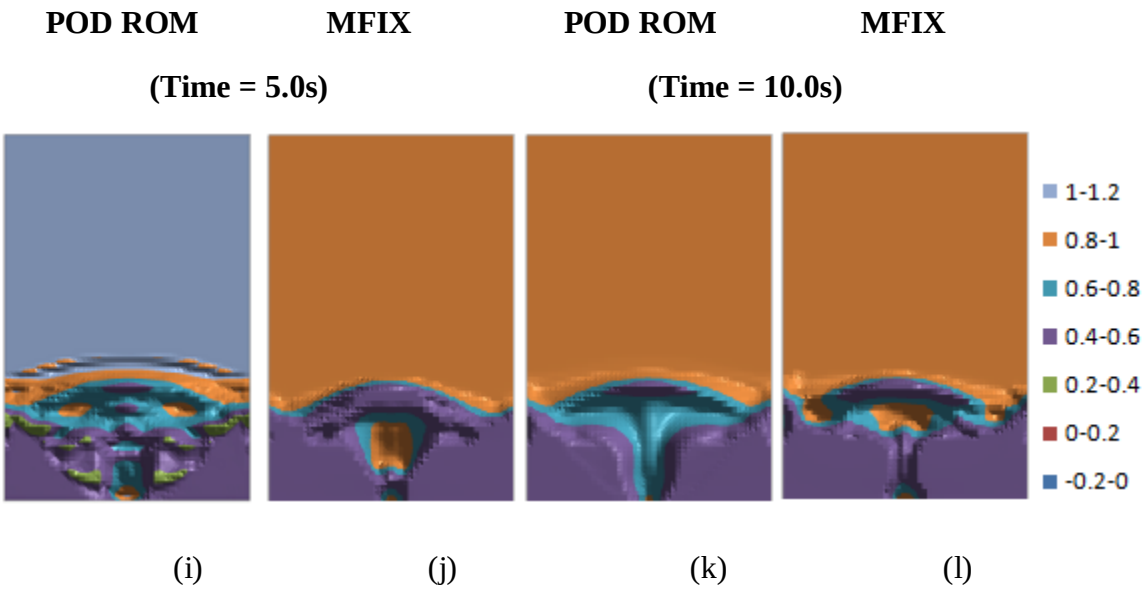


Figure 58: P_g **(Pa) Extrapolation Results for Coefficient of Restitution**

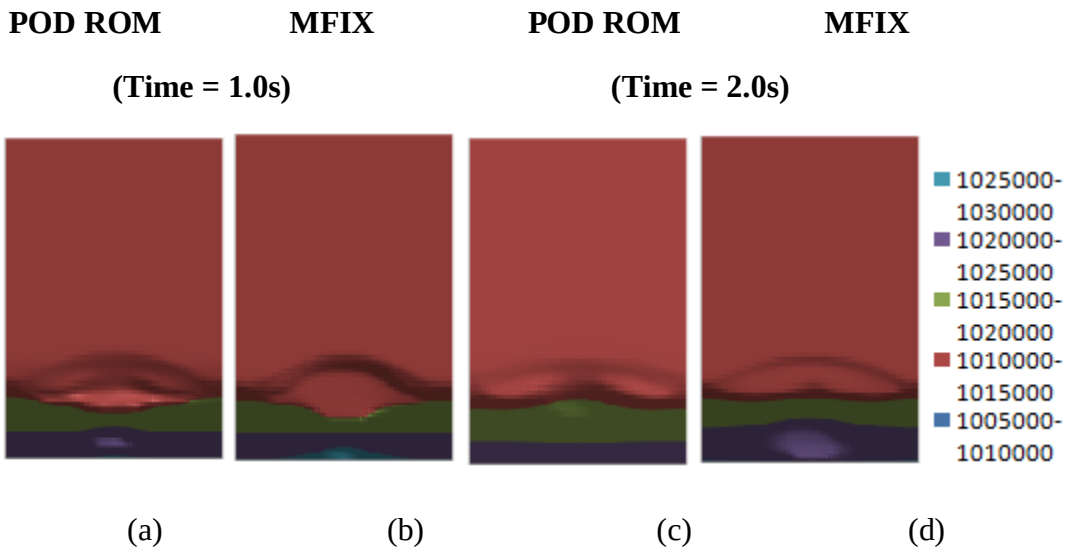
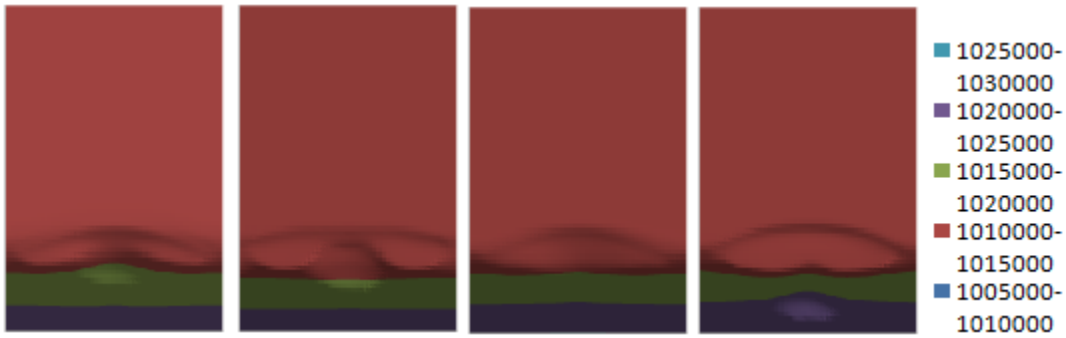


Figure 58 (cont): **P_g (Pa) Extrapolation Results for Coefficient of Restitution**

POD ROM **MFIX** **POD ROM** **MFIX**

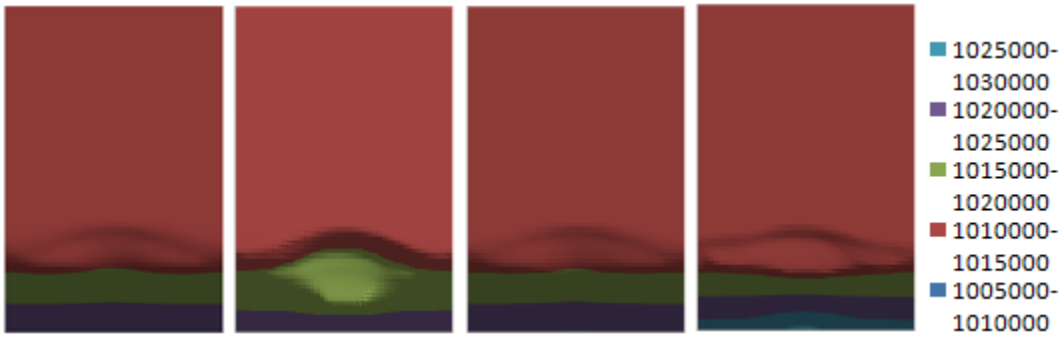
(Time = 3.0s) **(Time = 4.0s)**



(e) (f) (g) (h)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 5.0s) **(Time = 10.0s)**

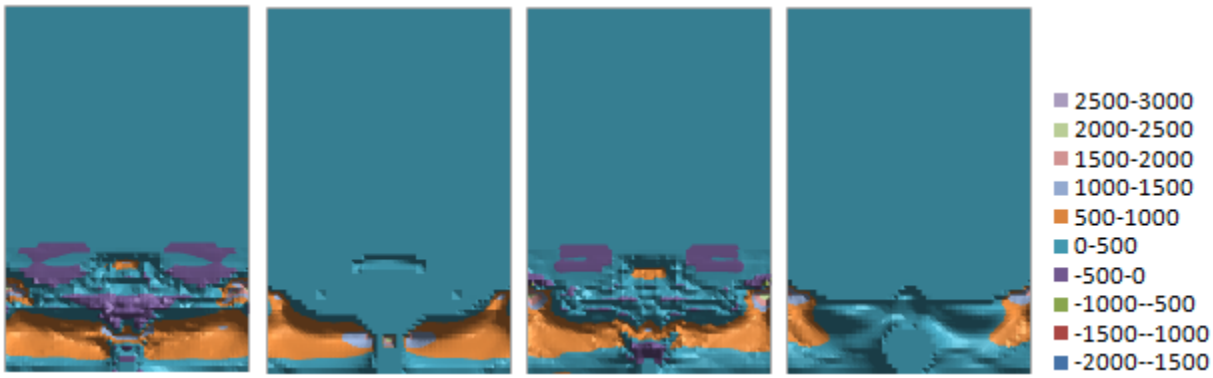


(i) (j) (k) (l)

Figure 59: P_s **(Pa) Extrapolation Results for Coefficient of Restitution**

POD ROM **MFIX** **POD ROM** **MFIX**

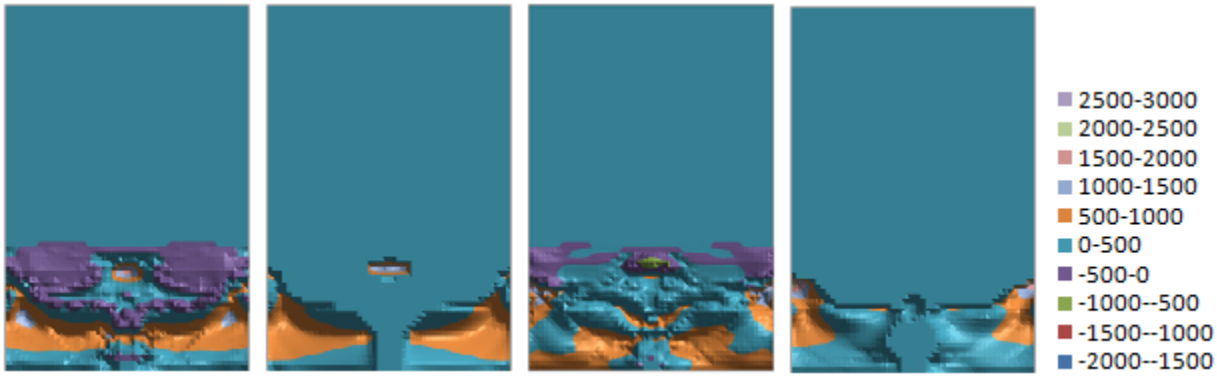
(Time = 1.0s) **(Time = 2.0s)**



(a) (b) (c) (d)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 3.0s) **(Time = 4.0s)**



(e) (f) (g) (h)

Figure 59 (cont): P_s (Pa) Extrapolation Results for Coefficient of Restitution

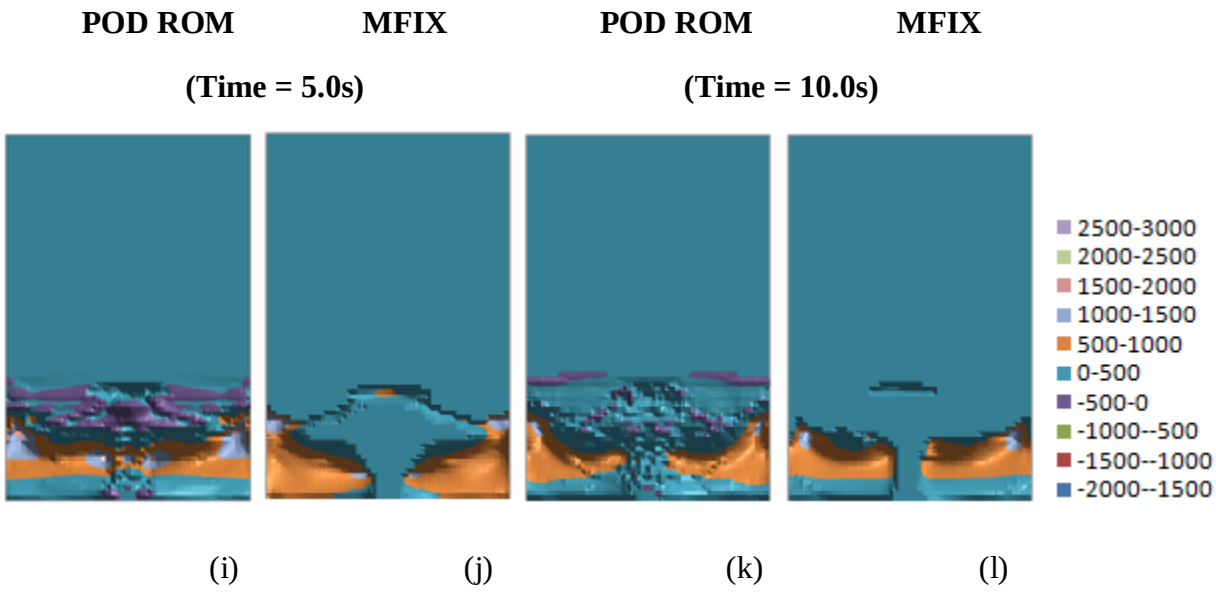


Figure 60: θ_s (m^2/s^2) Extrapolation Results for Coefficient of Restitution

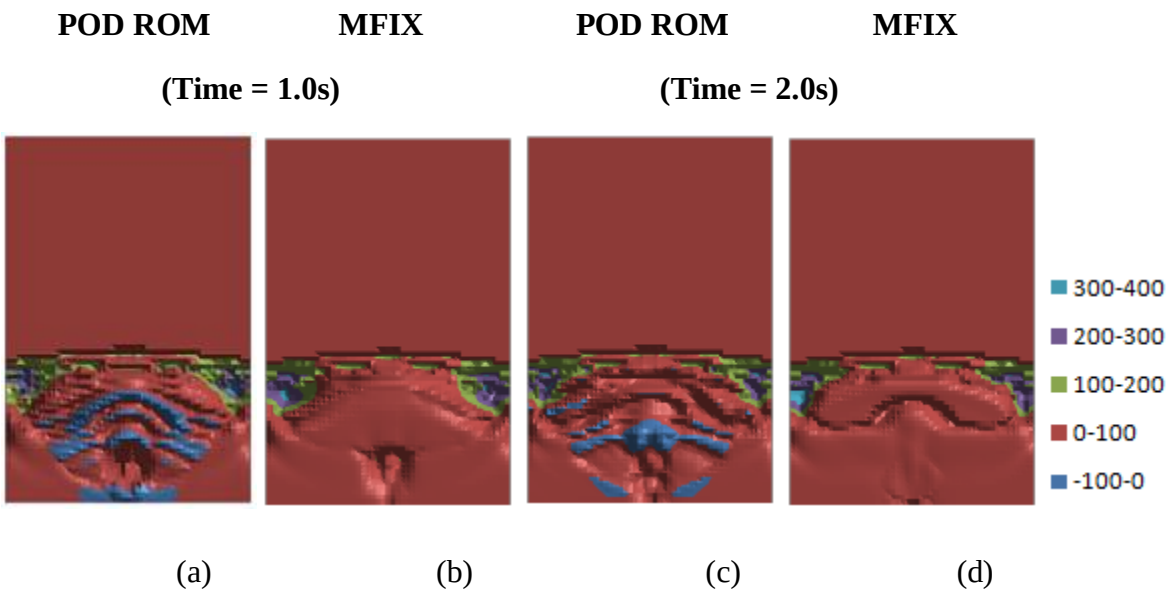
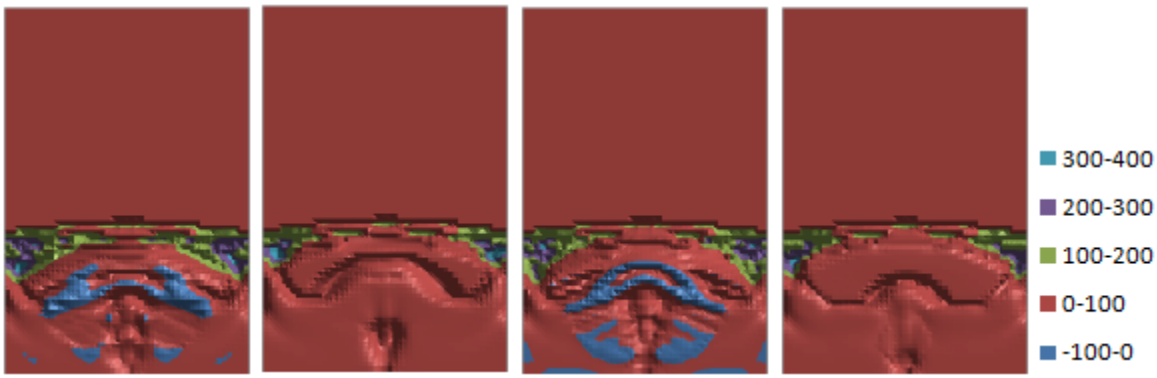


Figure 60 (cont): θ_s (m^2/s^2) **Extrapolation Results for Coefficient of Restitution**

POD ROM **MFIX** **POD ROM** **MFIX**

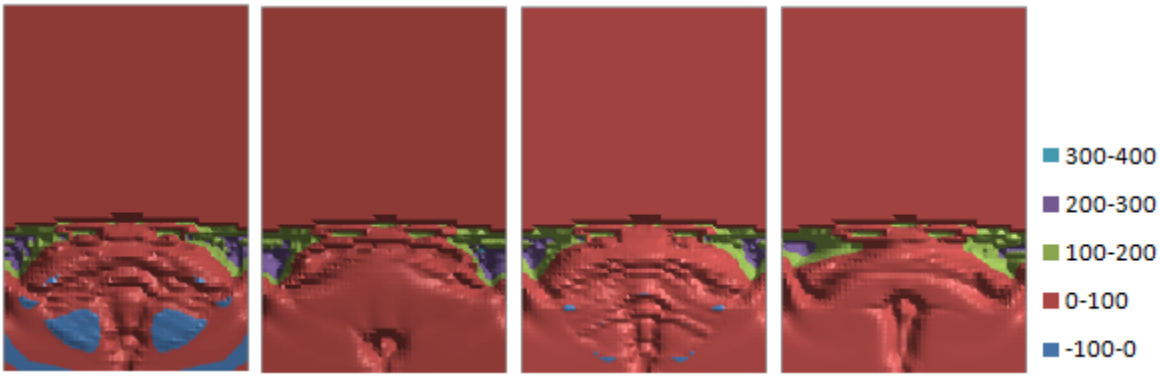
(Time = 3.0s) **(Time = 4.0s)**



(e) (f) (g) (h)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 5.0s) **(Time = 10.0s)**



(i) (j) (k) (l)

5.4 Interpolation of Parameter Model

Figures 61-68 highlight the results of an interpolation from two original MFIX simulations which had coefficient of restitution set at 0.8 and 0.85. This interpolative model approximates the solution of a spouted bed system where the value of the coefficient of restitution was set to 0.825. The images in (a), (c), (e), (g), (i), and (k) come from the POD ROM solution which are to be compared to their respective MFIX solution. The images in (b), (d), (f), (h), (j), and (l) depict these solutions in MFIX. Images (a) and (b) illustrate the solution at 1s. Images (c) and (d) illustrate the solution at 2s. Images (e) and (f) illustrate the solution at 3s. Images g and h illustrate the solution at 4s. Finally, the 5s solution is shown in (i) and (j) and the 10s solution is shown in (k) and (l).

Of the three predictive models presented in this paper, the interpolation model exhibits the best agreement overall. The central jet funnels (green areas) in Figure 62 bulge similarly at each second except for t=2s. The red and pale blue regions along the wall at the height of the spouted bed in Figure 63 which highlight particle absolute x velocities between 20 cm/s and 40 cm/s appear in relatively the same locations across the two models. In Figure 64, the y particle velocity funnels at the various time steps are similarly shaped between the two models and the high velocity locations in orange (40-60cm/s) appear in approximately the same locations in the funnel except at 10s. The gas void fraction illustrated in Figure 65 indicates some over approximation in the interpolative POD ROM model in the upper regions at the first couple of time steps. Gas pressure shown in Figure 66 is over estimated at 1s. Several physical features in this model were not captured in the interpolative POD ROM. The changes in pressure at the bottom of the bed above the central jet were not captured at most time steps. Recall that only 7 POD basis functions and modes impact the gas pressure calculations. This small number represents only approximately 1% of the total number of basis modes of all variables calculated at each time step. Again, since there is no weighting of basis modes in the solution of the model on a per variable basis, a high degree of accuracy was not expected in the pressure variable. Figure 67 illustrates the comparison between solid particle pressure of the interpolative POD ROM and MFIX models at the various time points. The large particle pressure regions compare favorably between the two models except at the 2s and 5s time points. The central spouting

region has only slight variability between the two models. In Figure 68, the solid particle temperature comparative figures are shown. The pockets of negative particle velocities along the central jet funnel correspond to those regions which exhibit disagreement in the gas void fraction. It seems likely that the incorrect presence of solid particles impact the results in this region.

Figure 61: U_g (cm/s) Interpolation Results

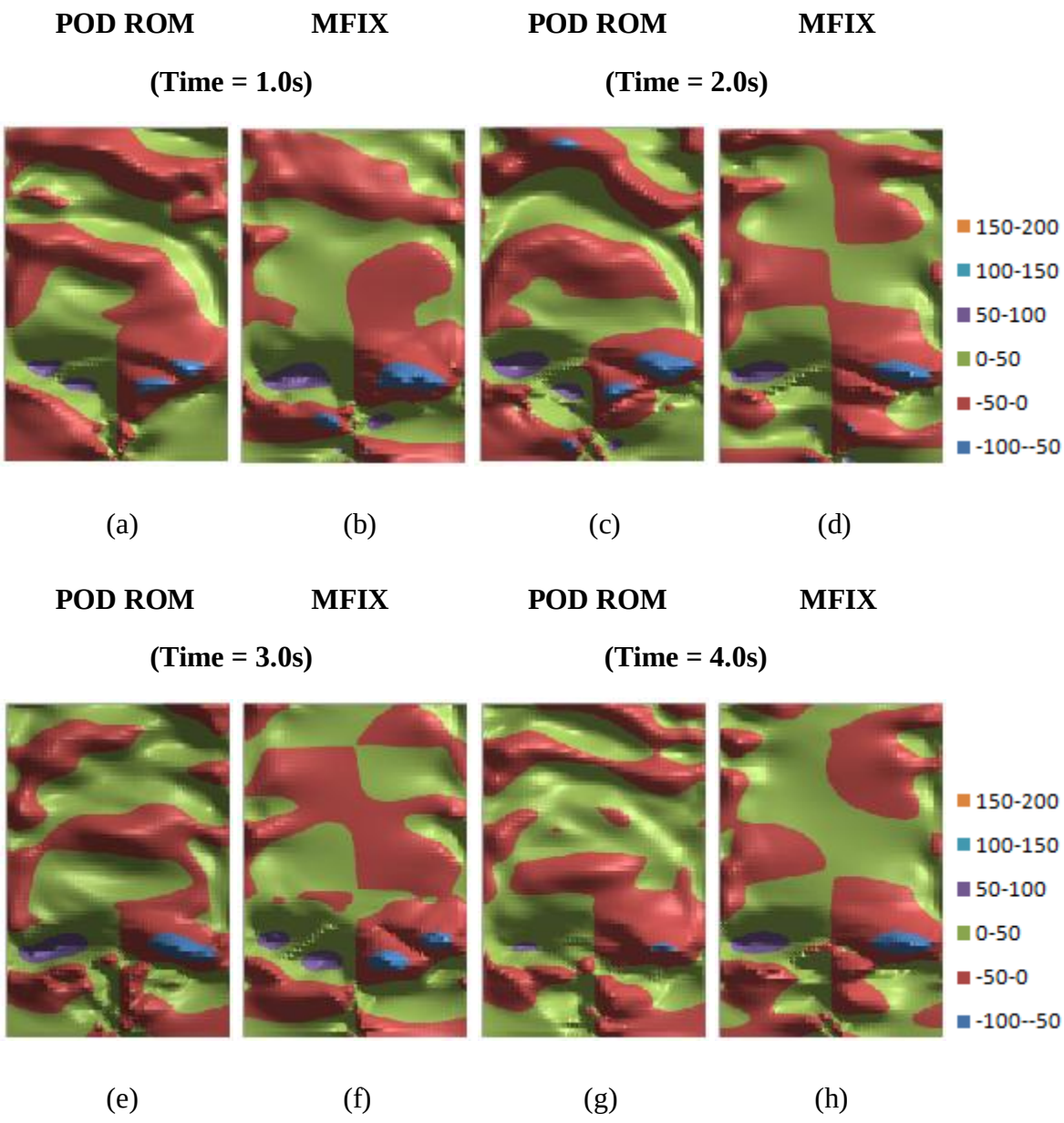
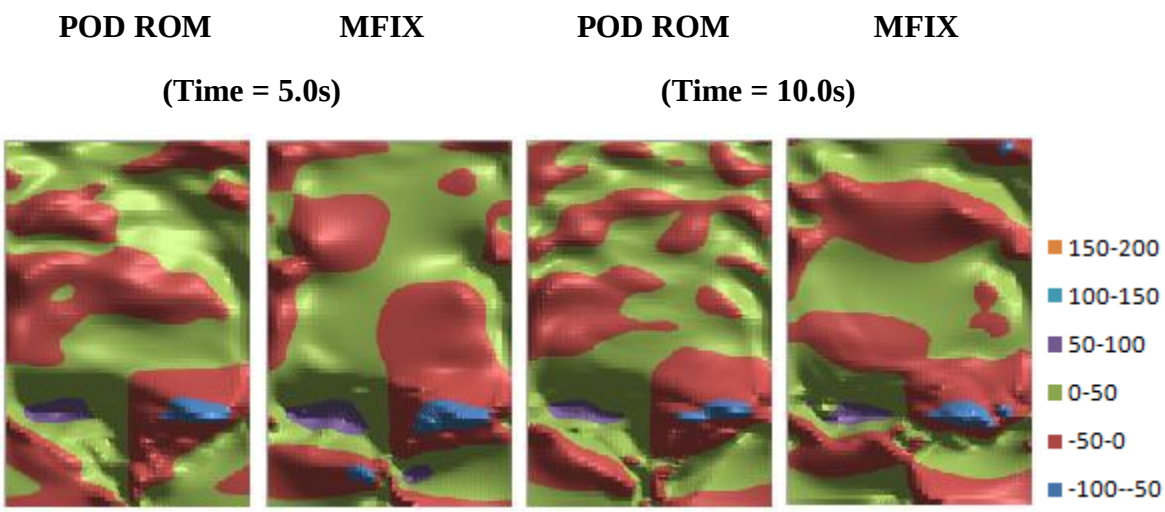
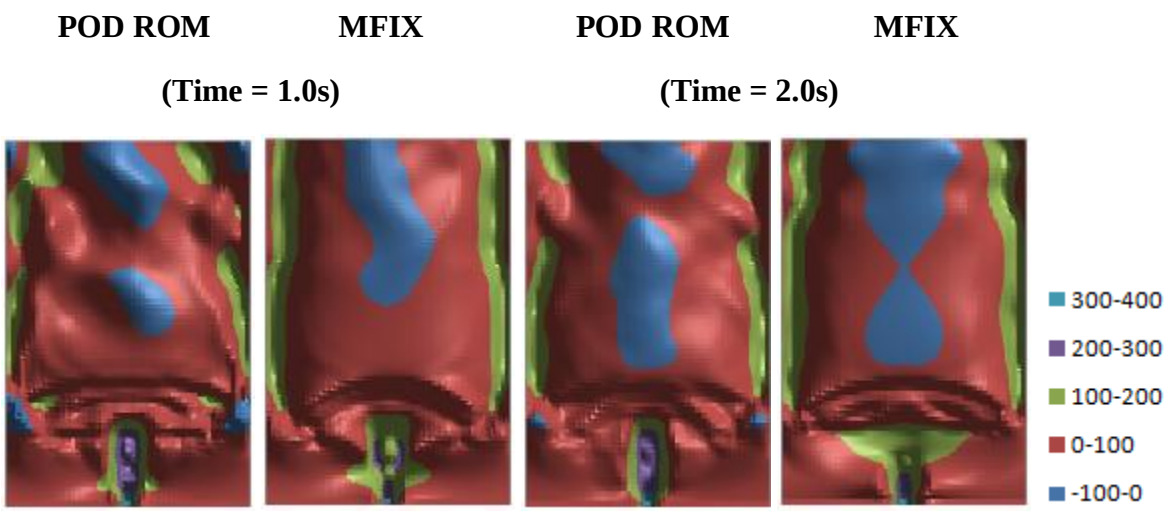


Figure 61 (cont): **U_g (cm/s) Interpolation Results**



(i) (j) (k) (l)

Figure 62: **V_g (cm/s) Interpolation Results**

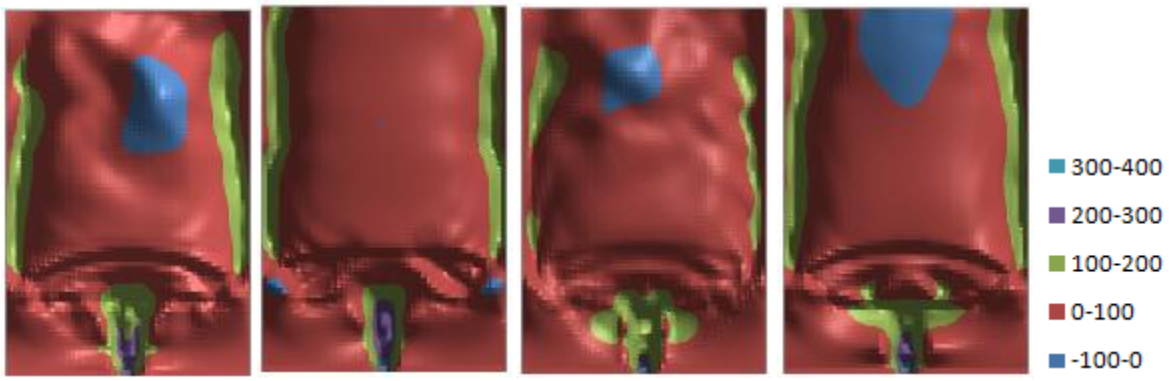


(a) (b) (c) (d)

Figure 62 (cont): **V_g (cm/s) Interpolation Results**

POD ROM **MFIX** **POD ROM** **MFIX**

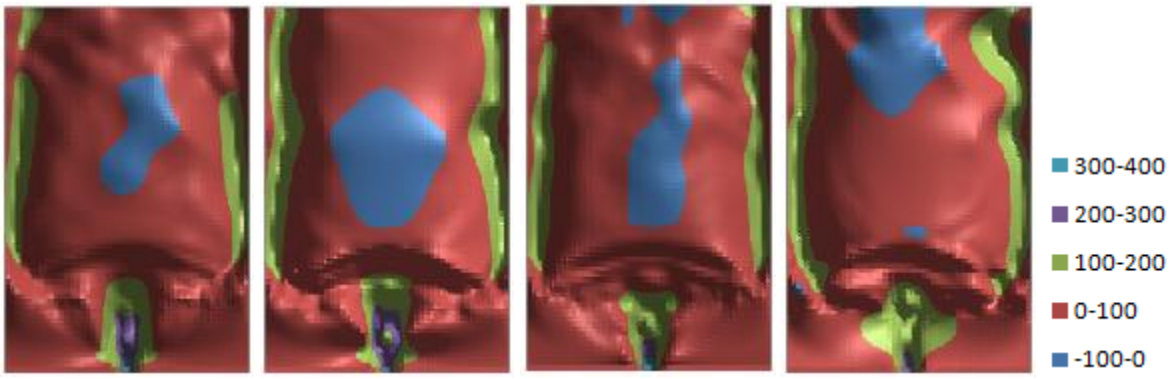
(Time = 3.0s) **(Time = 4.0s)**



(e) (f) (g) (h)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 5.0s) **(Time = 10.0s)**

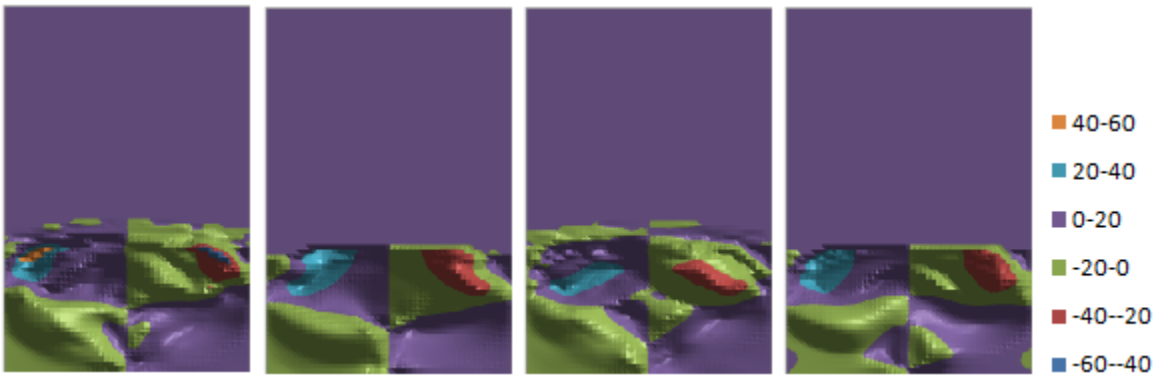


(i) (j) (k) (l)

Figure 63: U_s **(cm/s) Interpolation Results**

POD ROM **MFIX** **POD ROM** **MFIX**

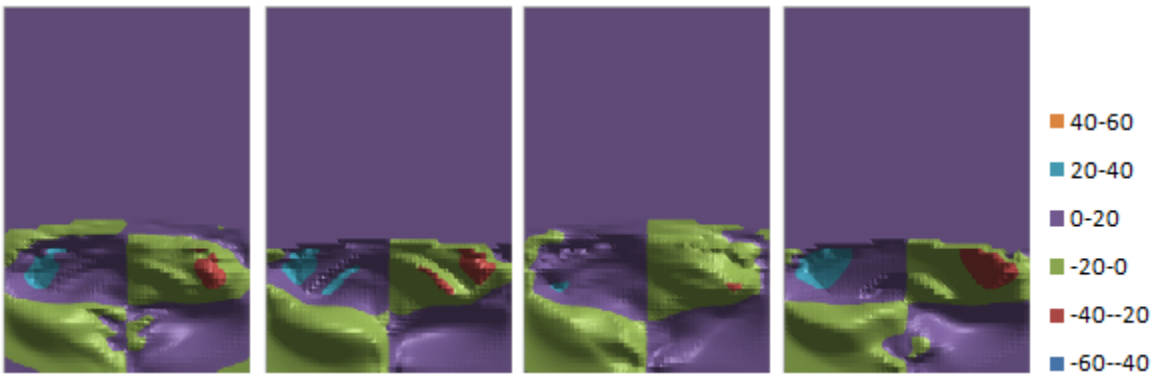
(Time = 1.0s) **(Time = 2.0s)**



(a) (b) (c) (d)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 3.0s) **(Time = 4.0s)**

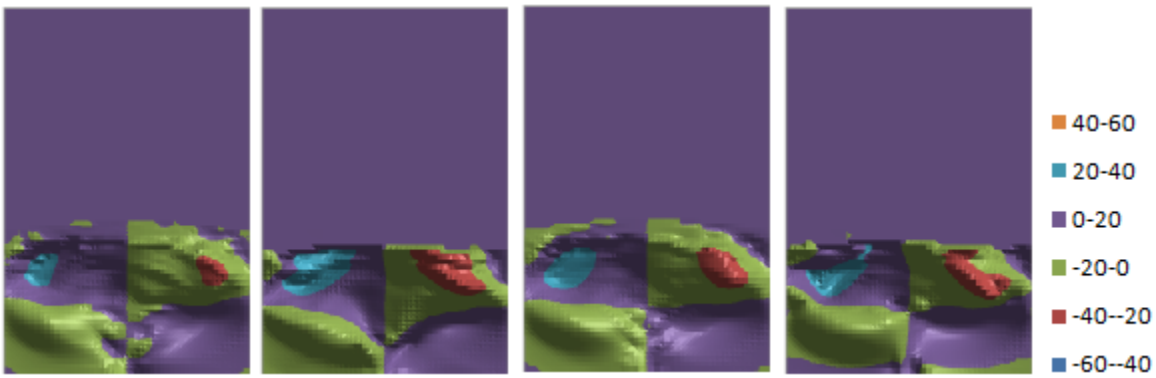


(e) (f) (g) (h)

Figure 63 (cont): U_s **(cm/s) Interpolation Results**

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 5.0s) **(Time = 10.0s)**

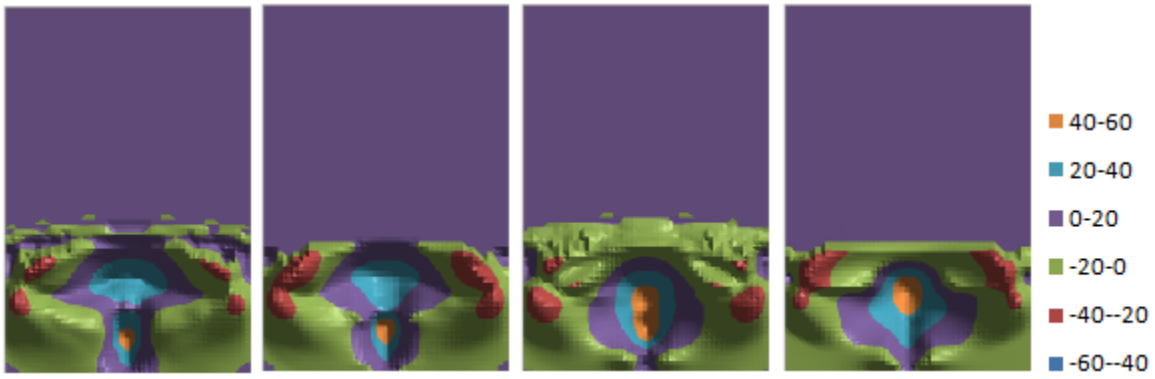


(i) (j) (k) (l)

Figure 64: V_s **(cm/s) Interpolation Results**

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 1.0s) **(Time = 2.0s)**

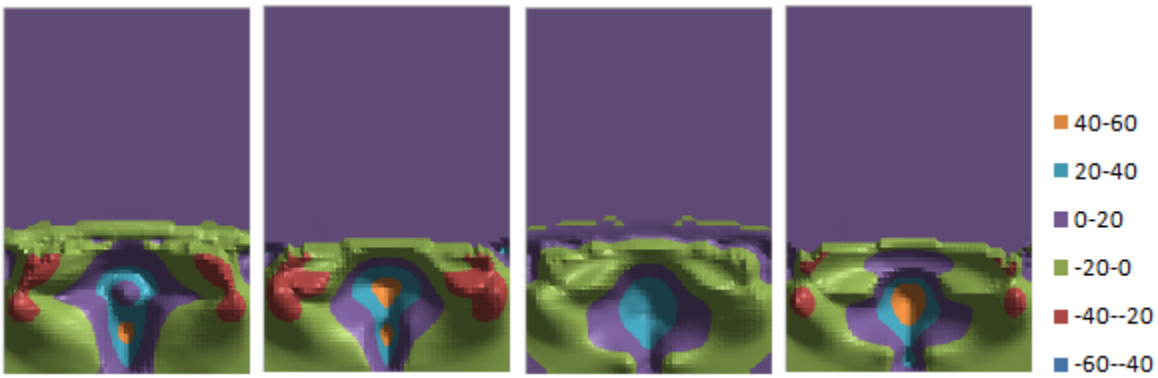


(a) (b) (c) (d)

Figure 64 (cont): V_s **(cm/s) Interpolation Results**

POD ROM **MFIX** **POD ROM** **MFIX**

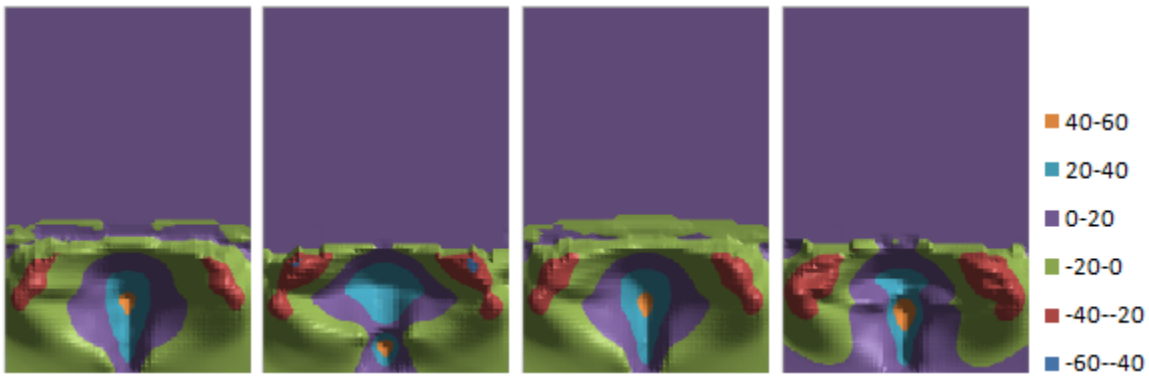
(Time = 3.0s) **(Time = 4.0s)**



(e) (f) (g) (h)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 5.0s) **(Time = 10.0s)**



(i) (j) (k) (l)

Figure 65: ϵ_g **Interpolation Results**

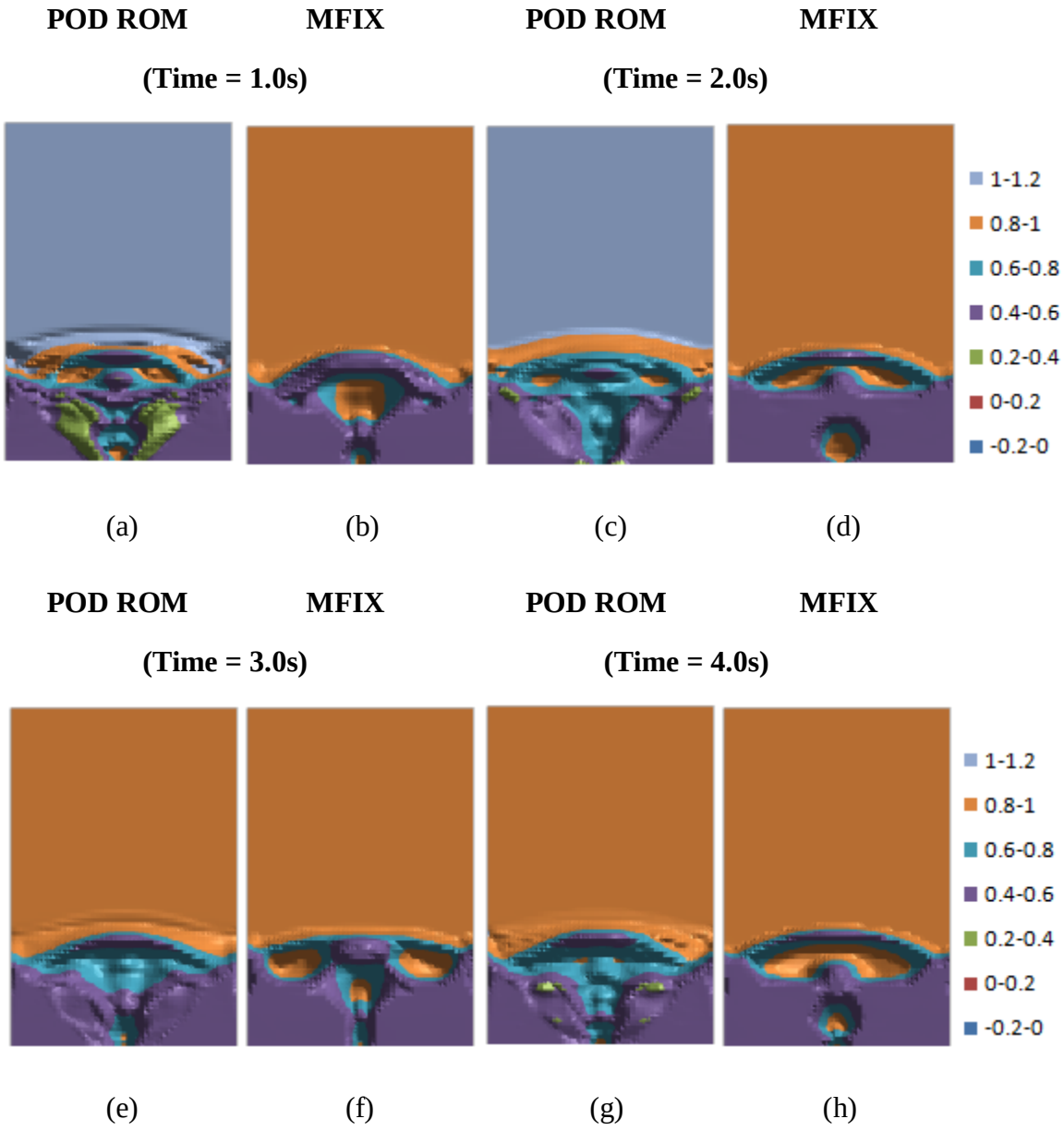


Figure 65 (cont): ϵ_g **Interpolation Results**

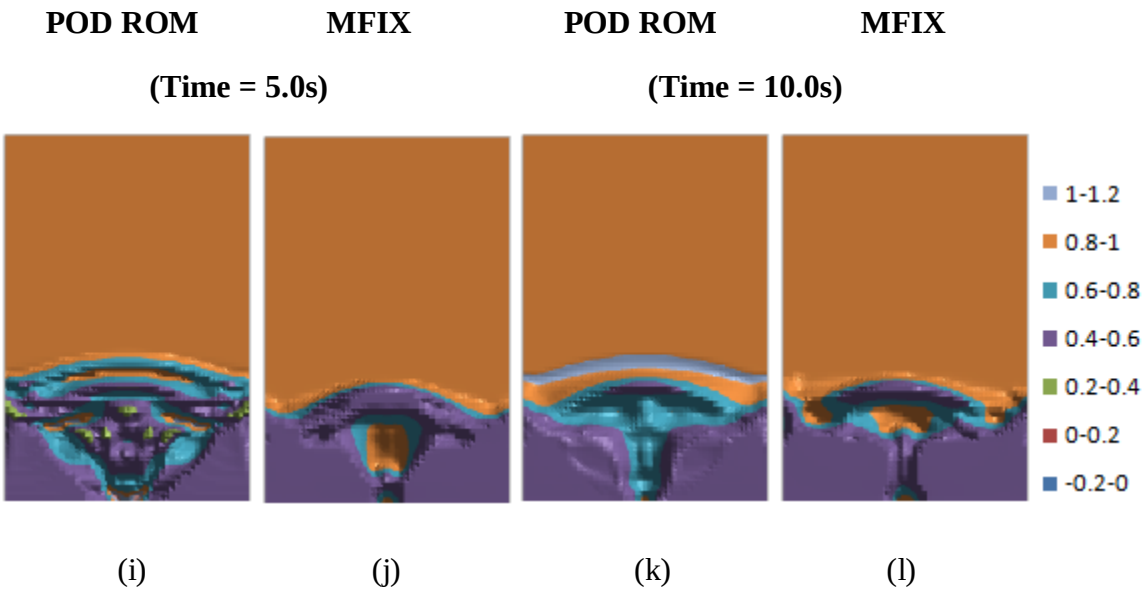


Figure 66: P_g **(Pa) Interpolation Results**

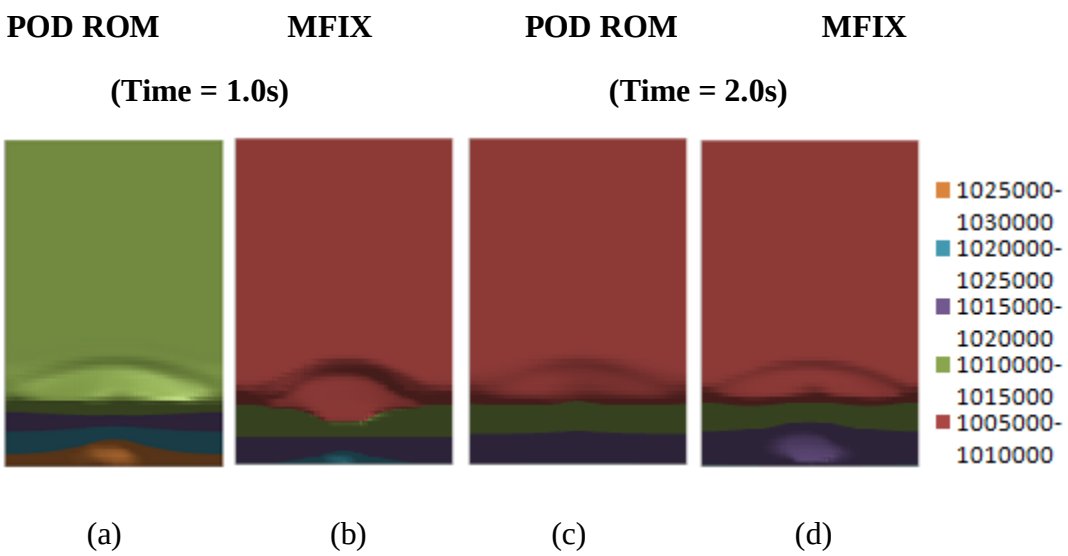
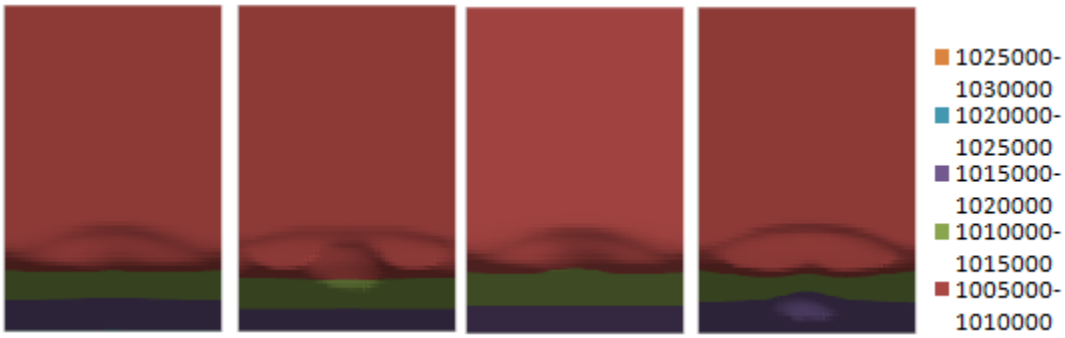


Figure 66 (cont): P_g **(Pa) Interpolation Results**

POD ROM **MFIX** **POD ROM** **MFIX**

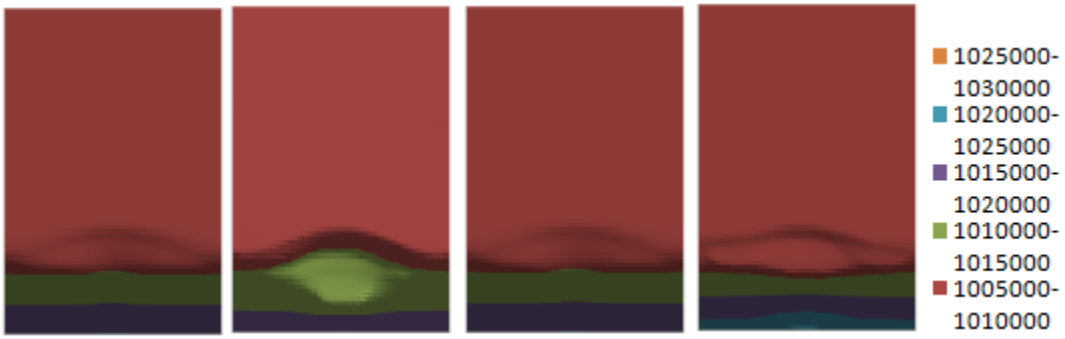
(Time = 3.0s) **(Time = 4.0s)**



(e) (f) (g) (h)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 5.0s) **(Time = 10.0s)**

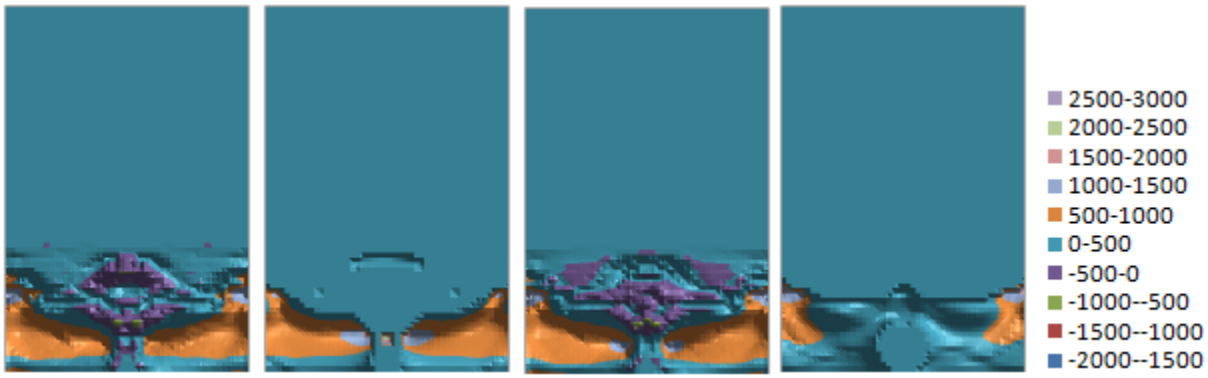


(i) (j) (k) (l)

Figure 67: P_s **(Pa) Interpolation Results**

POD ROM **MFIX** **POD ROM** **MFIX**

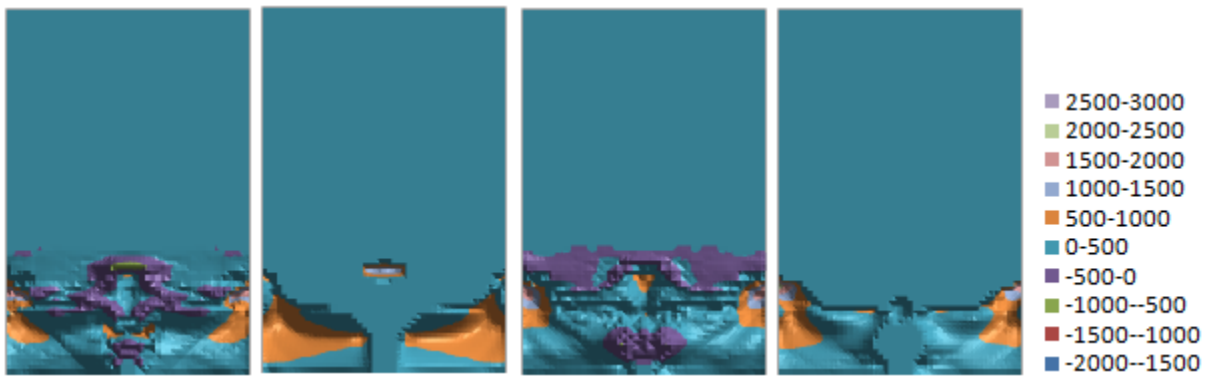
(Time = 1.0s) **(Time = 2.0s)**



(a) (b) (c) (d)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 3.0s) **(Time = 4.0s)**



(e) (f) (g) (h)

Figure 67 (cont): P_s **(Pa) Interpolation Results**

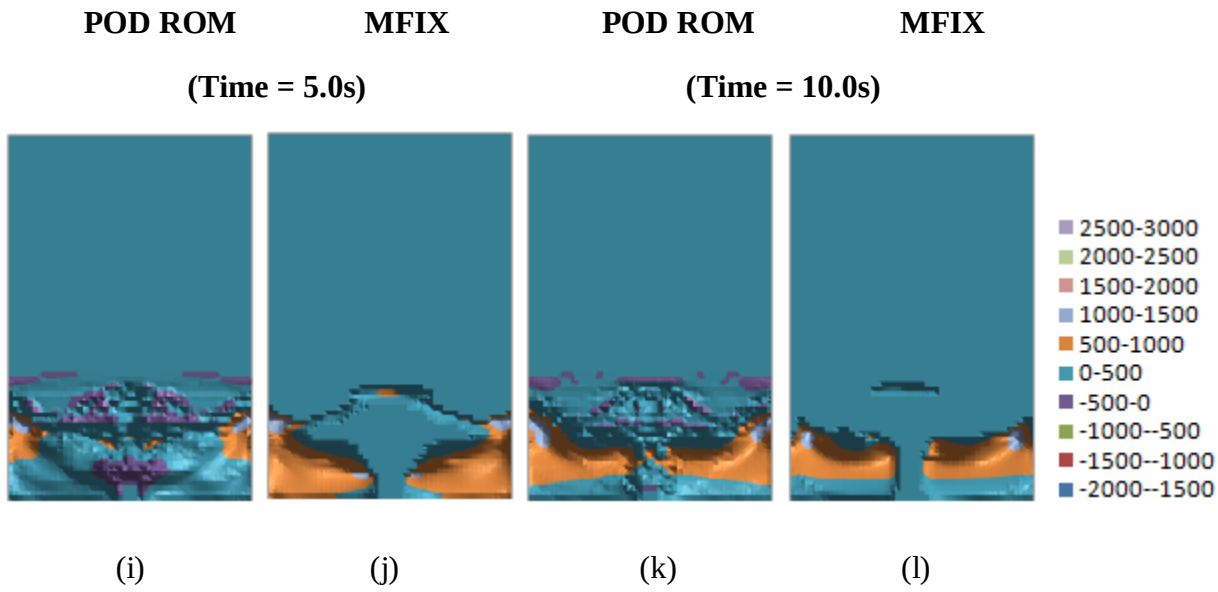


Figure 68: θ_s **(m^2/s^2) Interpolation Results**

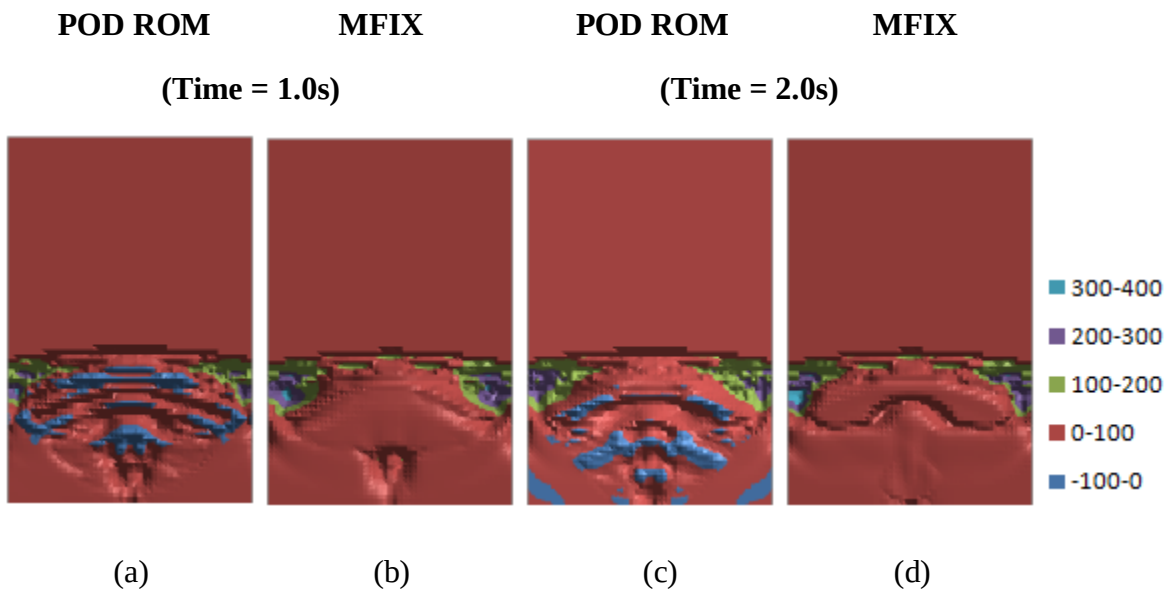
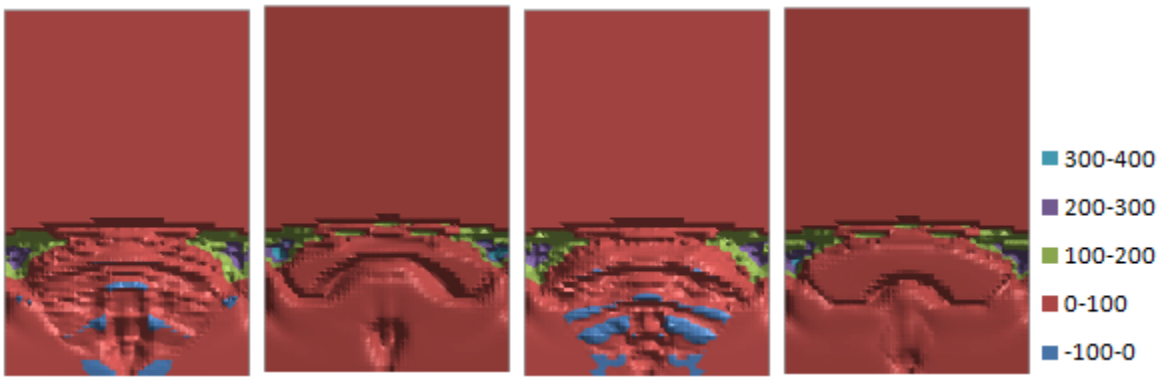


Figure 68 (cont): θ_s (m^2/s^2) **Interpolation Results**

POD ROM **MFIX** **POD ROM** **MFIX**

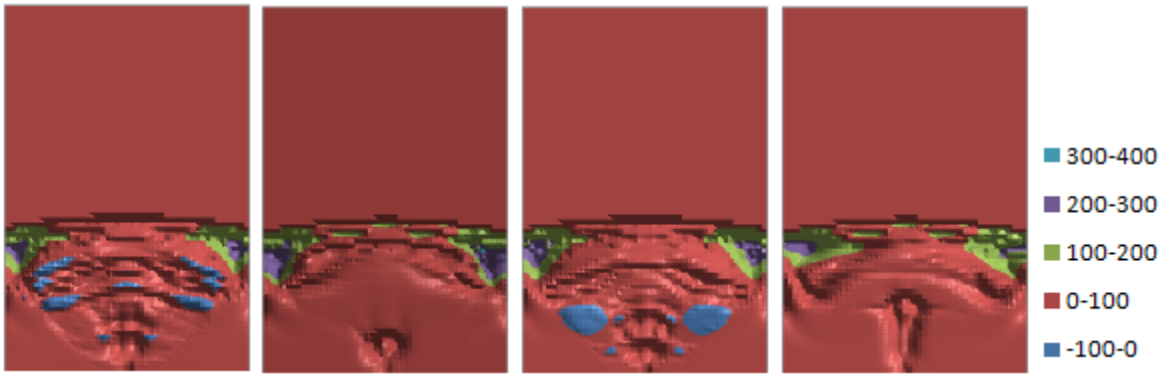
(Time = 3.0s) **(Time = 4.0s)**



(e) (f) (g) (h)

POD ROM **MFIX** **POD ROM** **MFIX**

(Time = 5.0s) **(Time = 10.0s)**



(i) (j) (k) (l)

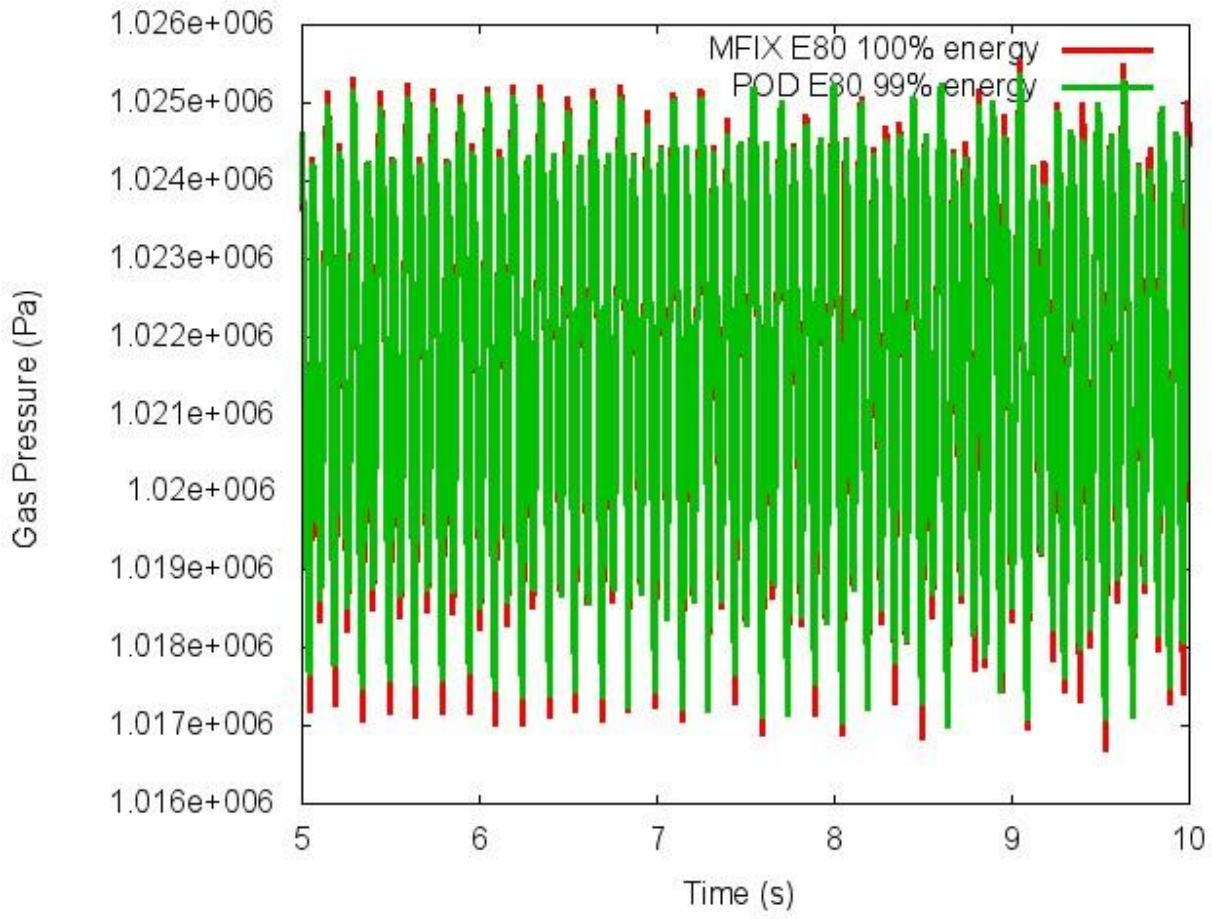
5.5 TIME SERIES ANALYSIS

Gas-solid flows exhibit chaotic phenomena because of the highly non-linear coupling between the gas and solids. As a result, a direct comparison of raw data at specific temporal points may not be the best way to assess the predicted dynamics from different models. As such, it is common to track variables in time and analyze the time-series. In this fashion, the temporal dynamics of a data set is analyzed and compared to other data sets through an examination of randomly selected spatial points within the computational domain. In this way, data can be correlated through temporal characteristics including frequency, amplitude and phase shift. In an effort to analyze only the stationary state behavior of the spouted bed system of interest in this dissertation, highly transient data prior to 5 seconds has been excluded from this analysis. In the following three subsections, time series analysis results are given for (1) POD reconstruction, (2) POD ROM validation and (3) POD ROM extrapolation. The fourth subsection contains power spectral density analysis for the POD ROM extrapolative model.

5.5.1 POD Reconstruction

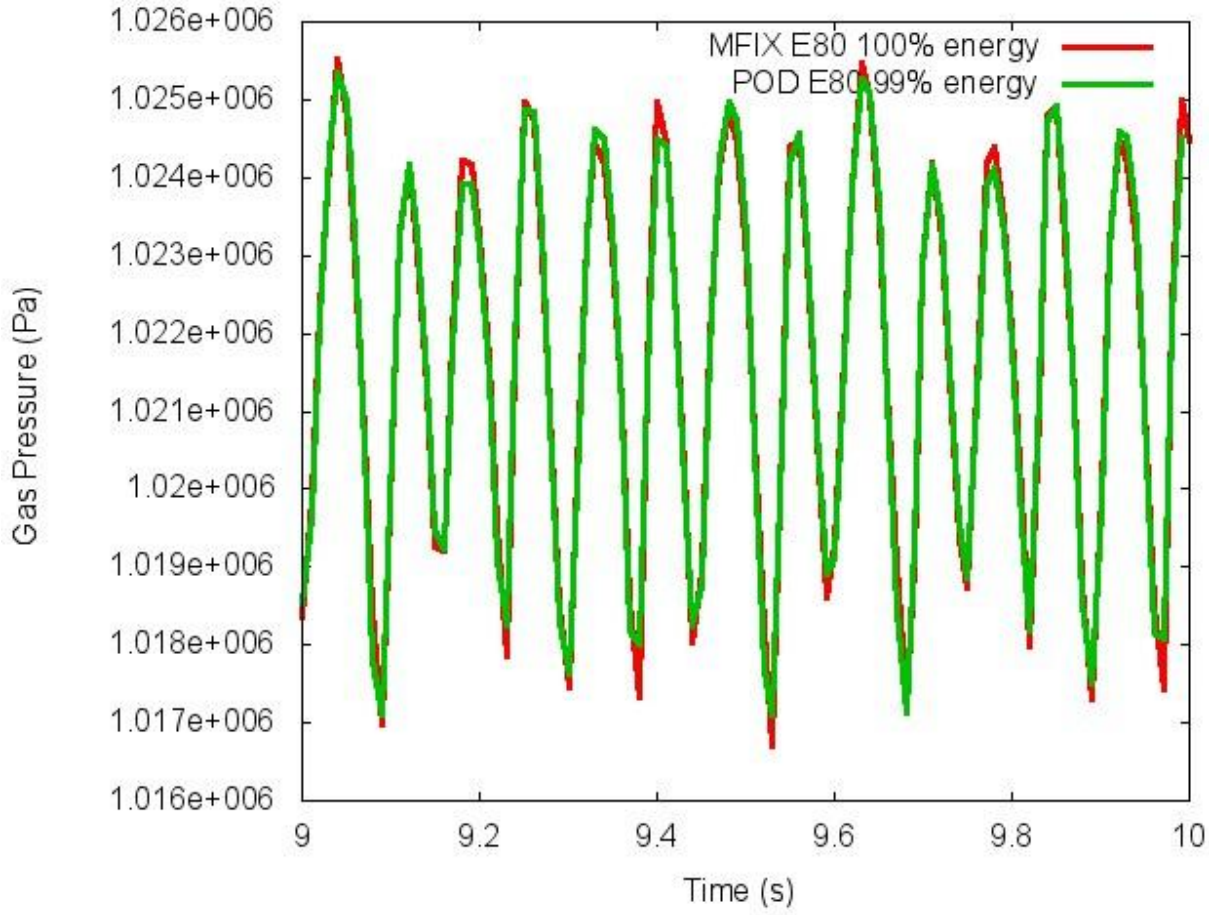
A randomly selected sample data point from the spouting region was used to perform the following time series analysis. Figure 69 illustrates gas pressure of the MFIX training data set versus the 99% matrix energy POD reconstruction at this data point where coefficient of restitution equals 0.80.

Figure 69: POD Reconstruction of Gas Pressure 5-10 second interval



Closer examination of the 9-10 second interval of Figure 69 reveals significant agreement. See Figure 70.

<u>Figure 70:</u> POD Reconstruction of Gas Pressure 9-10 second interval



5.5.2 POD ROM Validation

Likewise, a subsequent examination of the POD ROM validation (with 99% matrix energy capture) of gas pressure at coefficient of restitution value 0.80 exhibited good agreement with the MFIX training dataset at the same randomly selected sample data point from the spouting region. This is illustrated in Figures 71 and 72.

Figure 71: POD ROM Validation of Gas Pressure 5-10 second interval

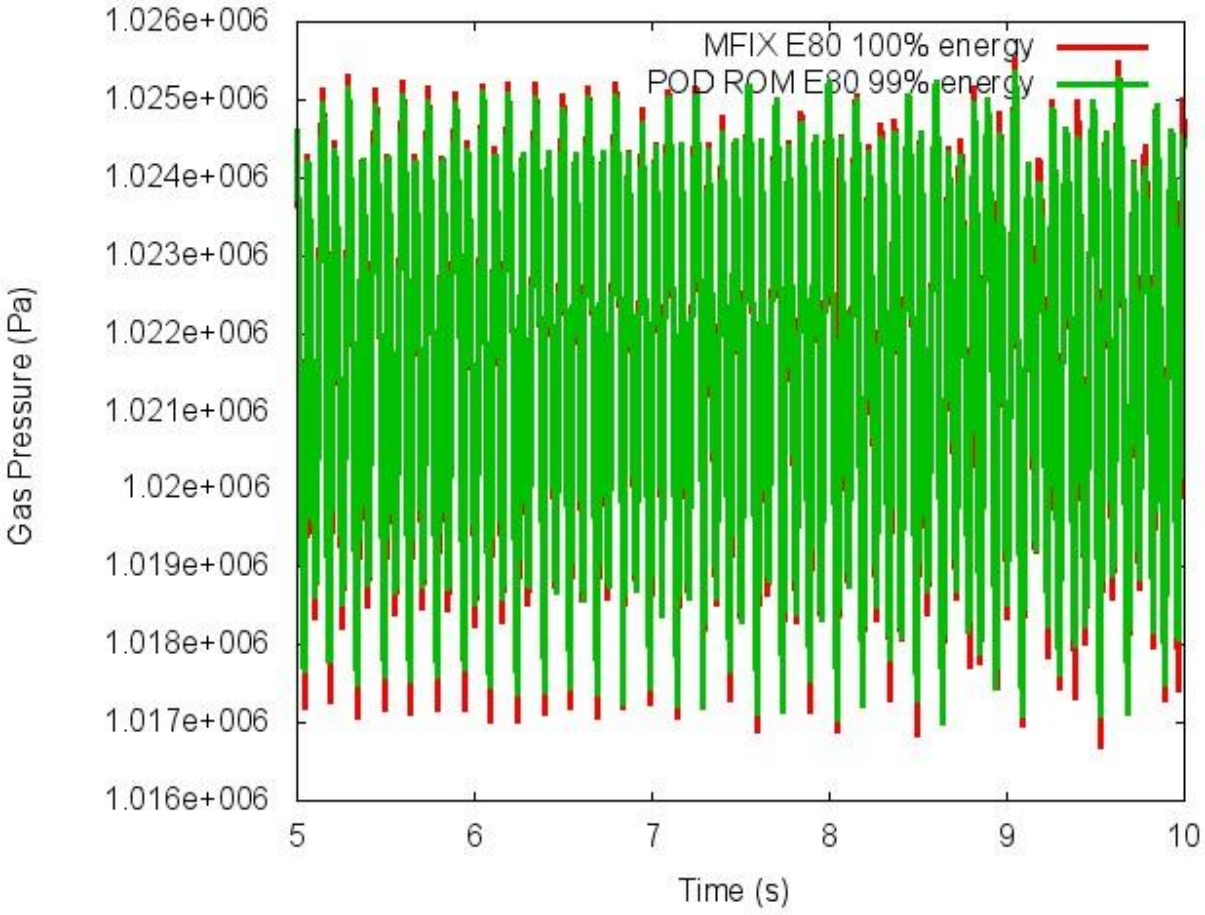
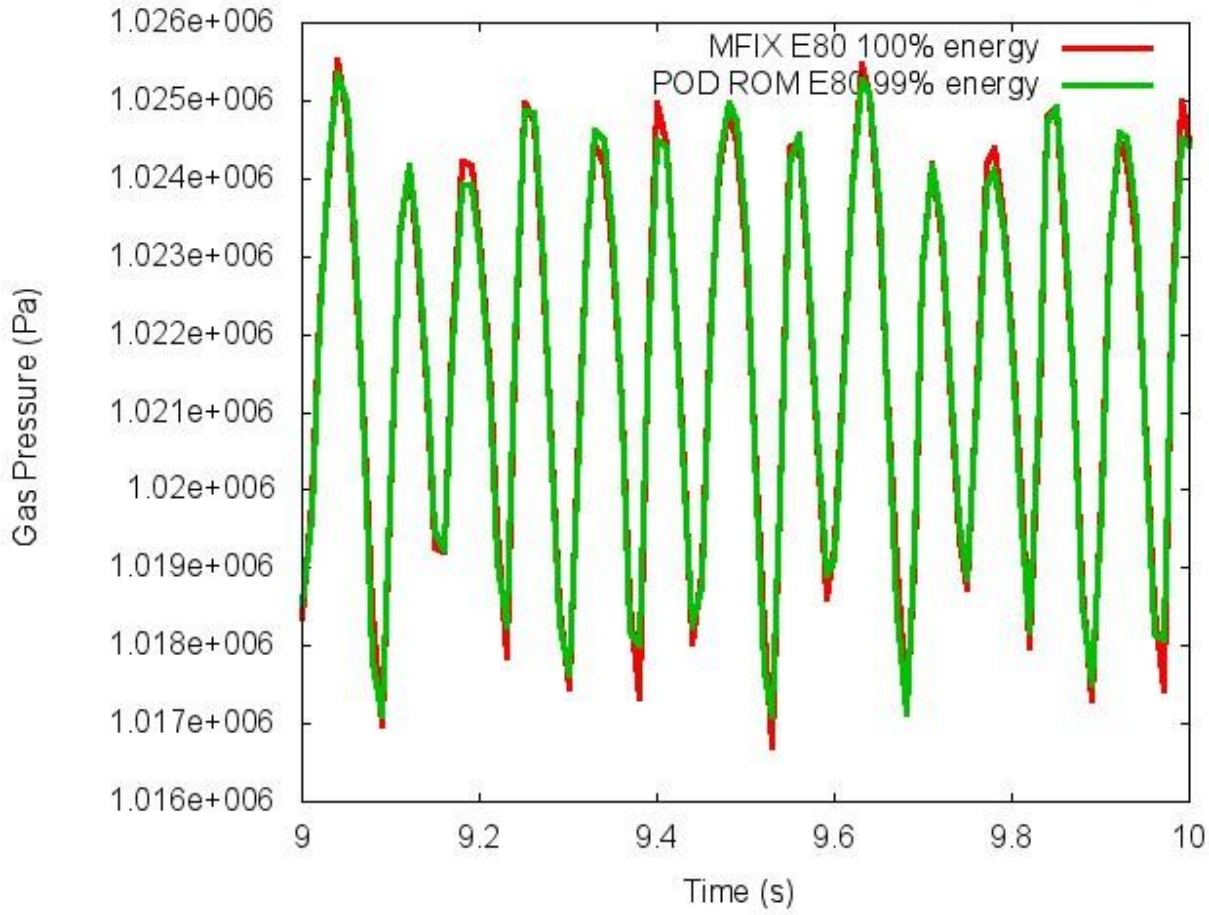


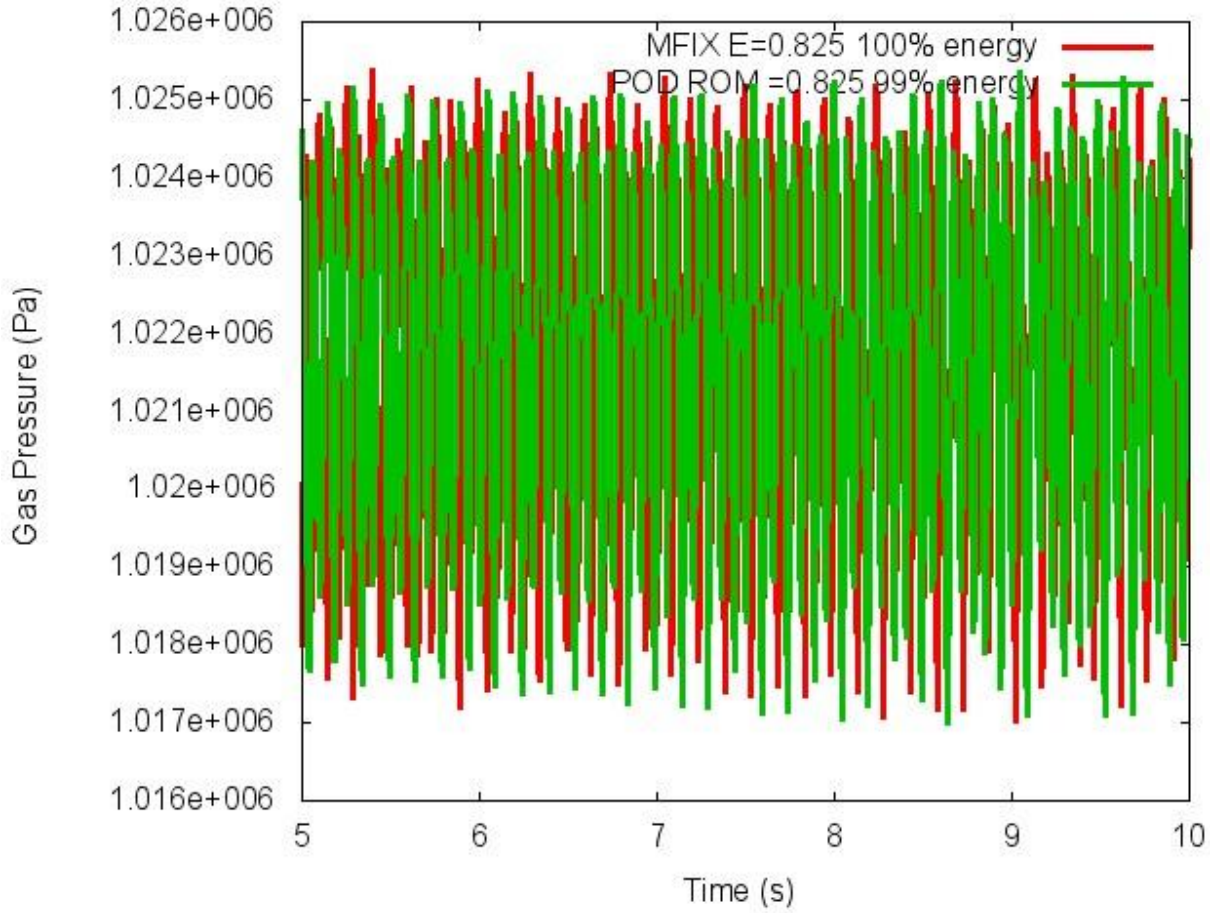
Figure 72: POD ROM Validation of Gas Pressure 9-10 second interval



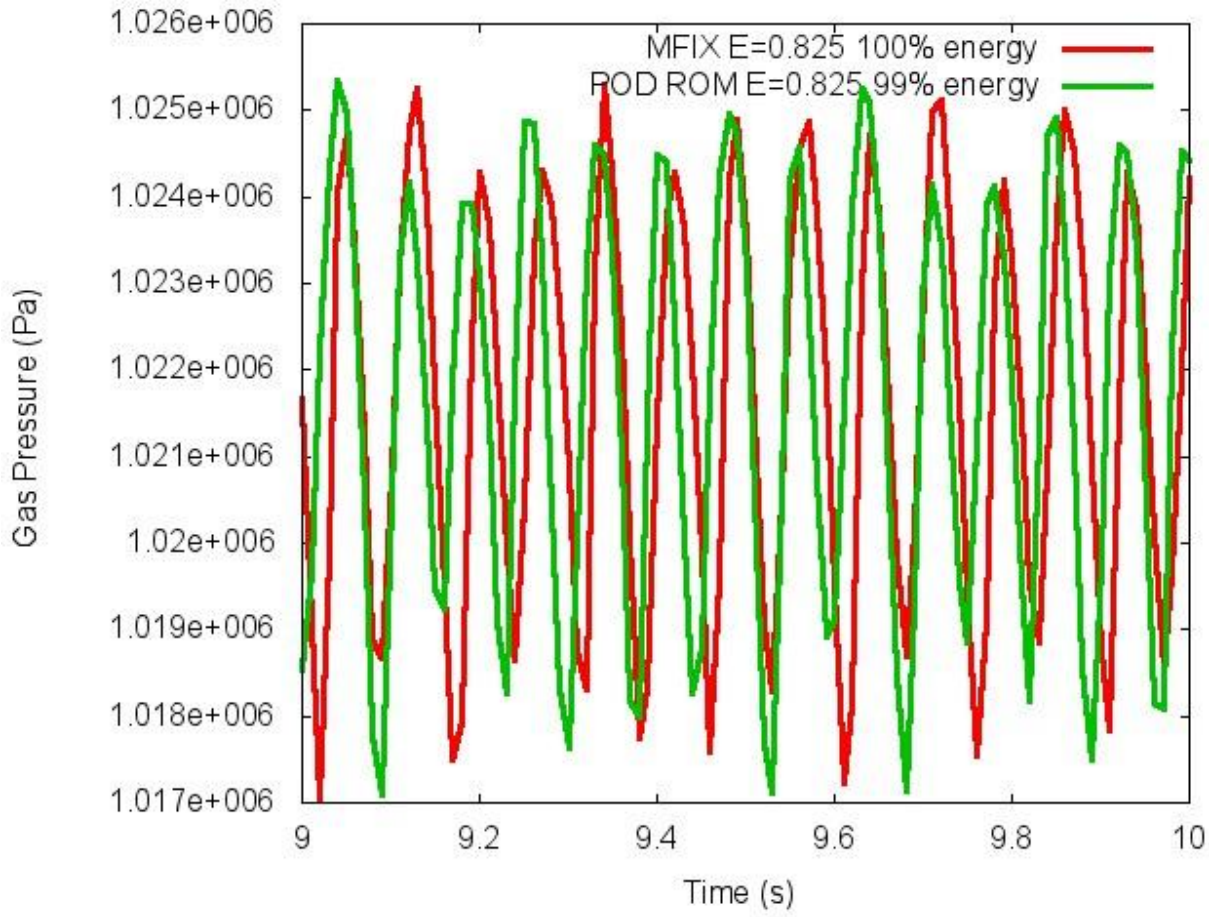
5.5.3 POD ROM Extrapolation

Figures 73 and 74 illustrate the gas pressure behavior at the randomly selected sample data point from the spouting region of both the MFIX validation dataset and the extrapolative POD ROM. In the extrapolation, the MFIX training data has coefficient of restitution equal to 0.80 and the POD ROM creates a new model at coefficient of restitution equal to 0.825. The data comparison is to *another* MFIX validation data set where the coefficient of restitution is set at 0.825. These results are shown from 5 to 10 seconds.

Figure 73: POD ROM Extrapolation of Gas Pressure 5-10 second interval



<u>Figure 74:</u> POD ROM Extrapolation of Gas Pressure 9-10 second interval



As illustrated in Figures 73 and 74, the period of the gas pressure in the spouting region is on average 0.0731 seconds.

Figure 75: POD ROM Extrapolation Gas Void Fraction

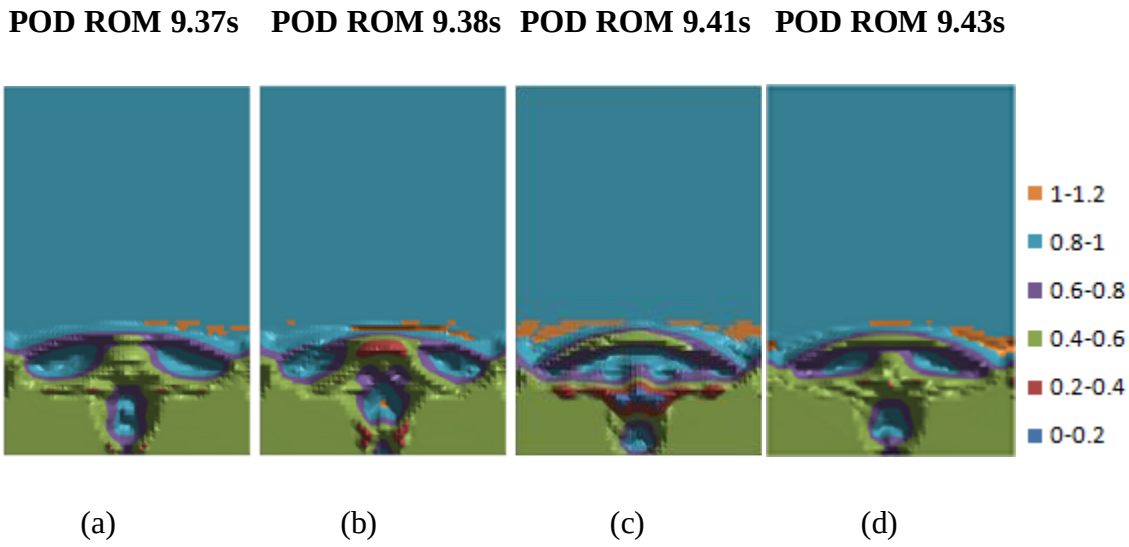


Figure 75 illustrates the gas void fraction during a pulsation of the spouted bed system. Notice the gas void fraction distribution particularly near the central jet, through the annular region of the spout and at the top of the spout. As previously discussed and shown in Figure 57, non physical behavior may result when gas void fraction is not restricted to values less than one. As noted before, a non physical representation of gas void fraction is more likely to occur in areas of rapid transition and along the boundaries of physical characteristics. In particular, the top of the spouted bed bordering the freeboard region exhibits this behavior. Likewise, in image (b), the highly volatile region in the center of the jet exhibits an over prediction of gas void fraction. A similar region is also identified in image (d) which is along the boundary of the central jet and the top of the spouted bed.

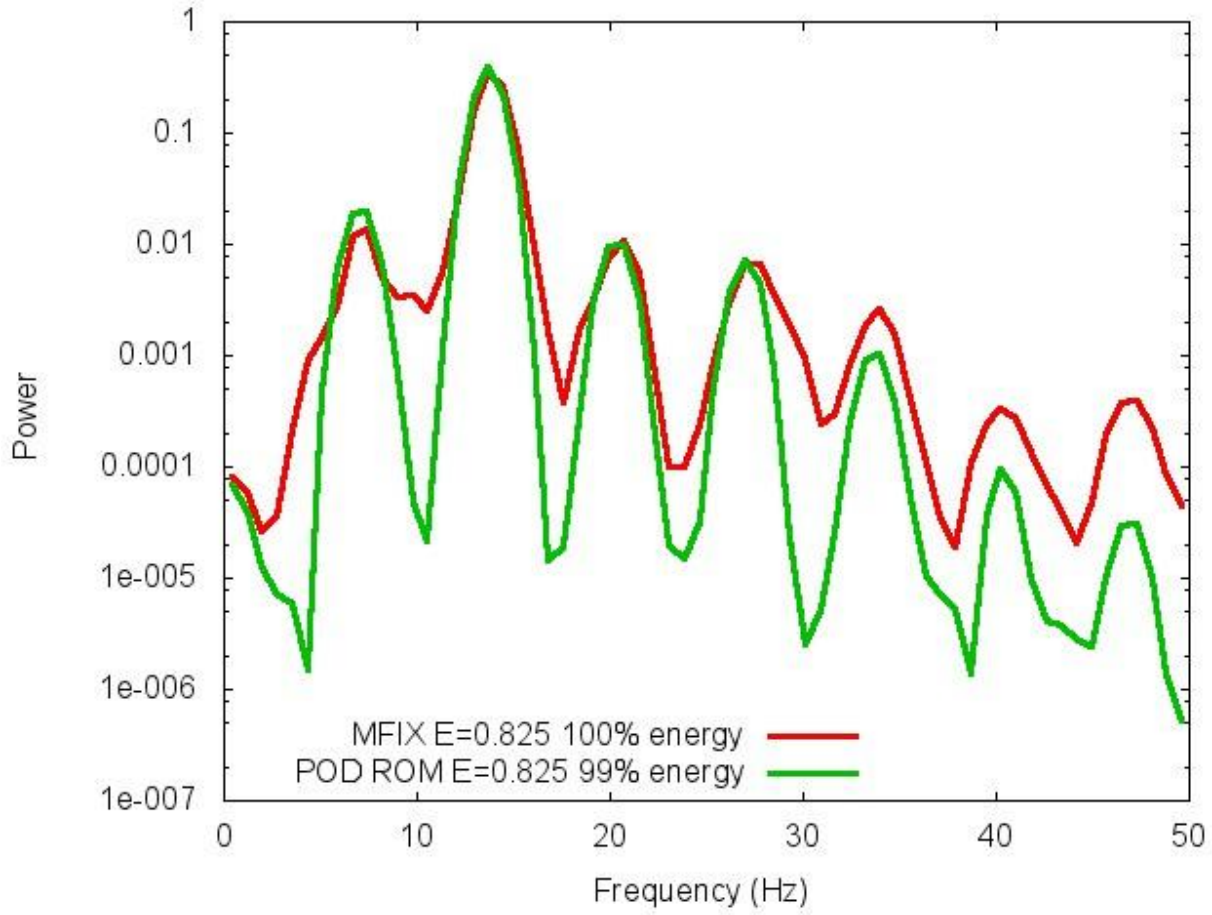
5.5.4 Power Spectral Density

Power spectral density (PSD) measures the *frequency distribution of the statistical variance* called **power** for time series data. Mathematically, PSD is defined as the Fourier Transform of the autocorrelation sequence across a time series [45].

Pannala et. al present PSD in the context of nuclear fuel coaters using spouted beds [16] and show it is a meaningful tool for purposes of validating a numerical model. In this dissertation, comparing the gas pressure power spectrum of the MFIX validation data set to the spectrum associated with the POD ROM extrapolated dataset, provides a more thorough validation of this methodology. PSD calculations were performed utilizing the software package *post-mfix* which is distributed with the MFIX software package [22].

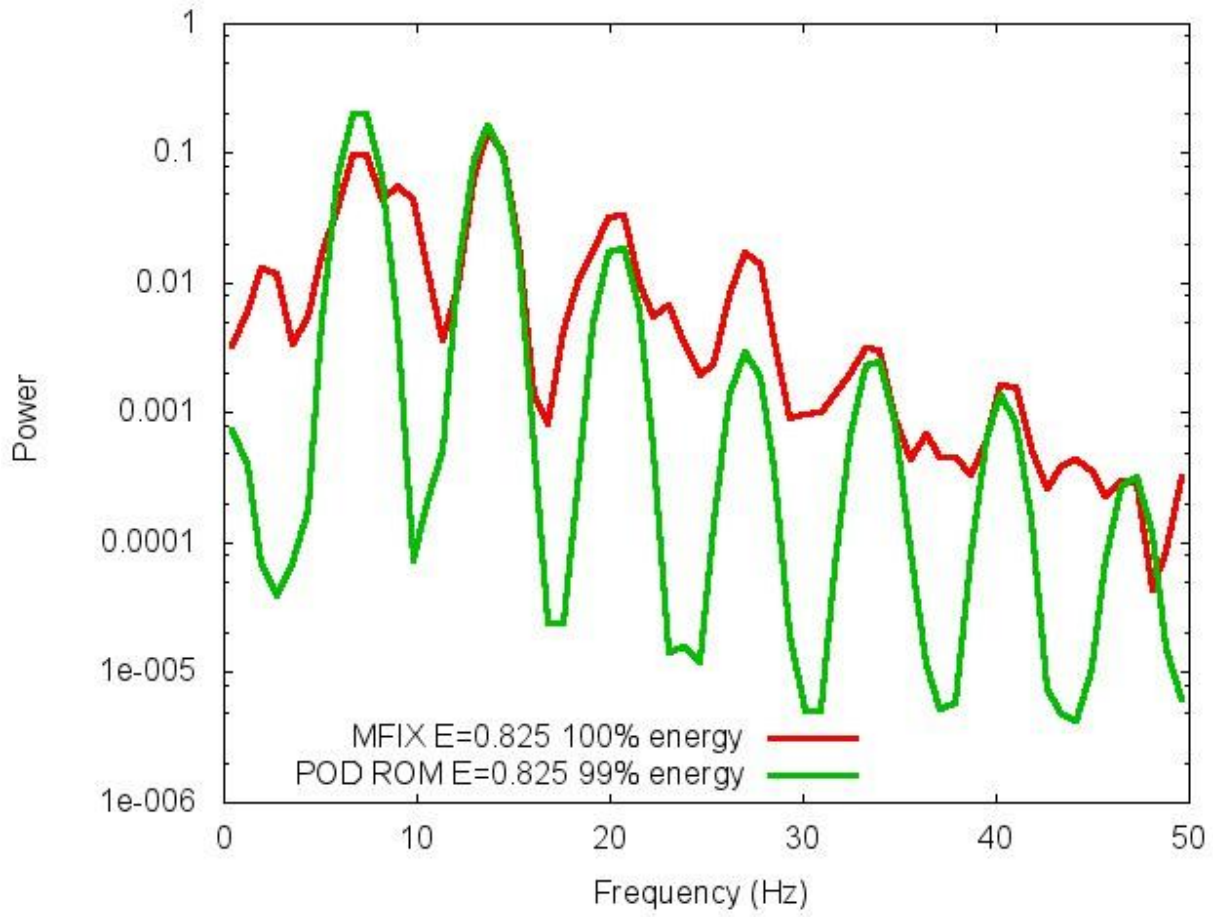
Time series data obtained through examination of the behavior of gas pressure is explored for a sample cell in each of four select locations: the spouting region, the top of the spouted bed, the annular region of the spouted bed and along the wall. The power spectral density functions for these locations are illustrated on a log scale in Figures 76 - 79.

Figure 76: **Gas Pressure PSD in Spouting Region**



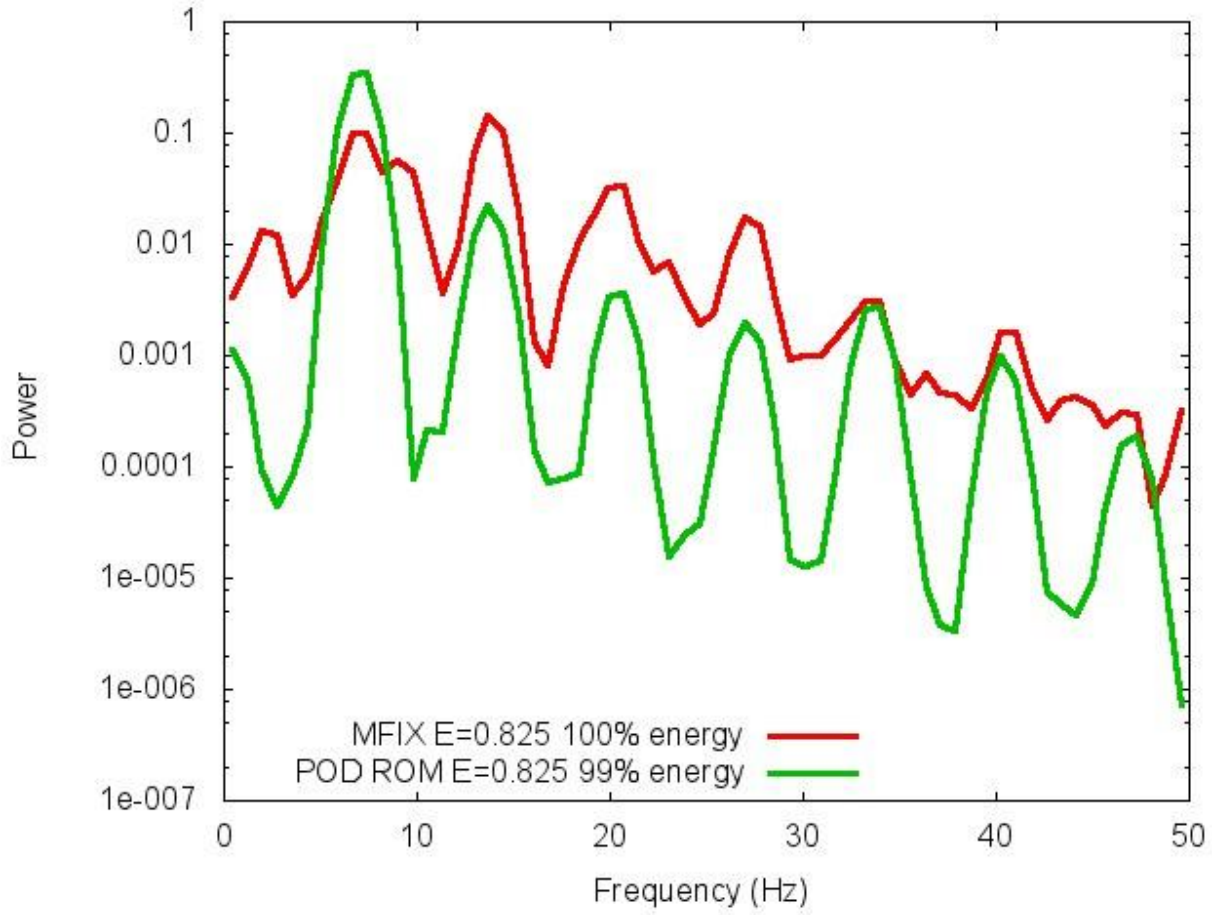
In Figure 76, the gas pressure in the spouting region has a dominant (or peak) frequency of 13.671875Hz for both methodologies with power of 0.3892 for the extrapolative POD ROM and 0.3534 for the MFIX validation data set.

Figure 77: Gas Pressure PSD in Annular Region of the Spout



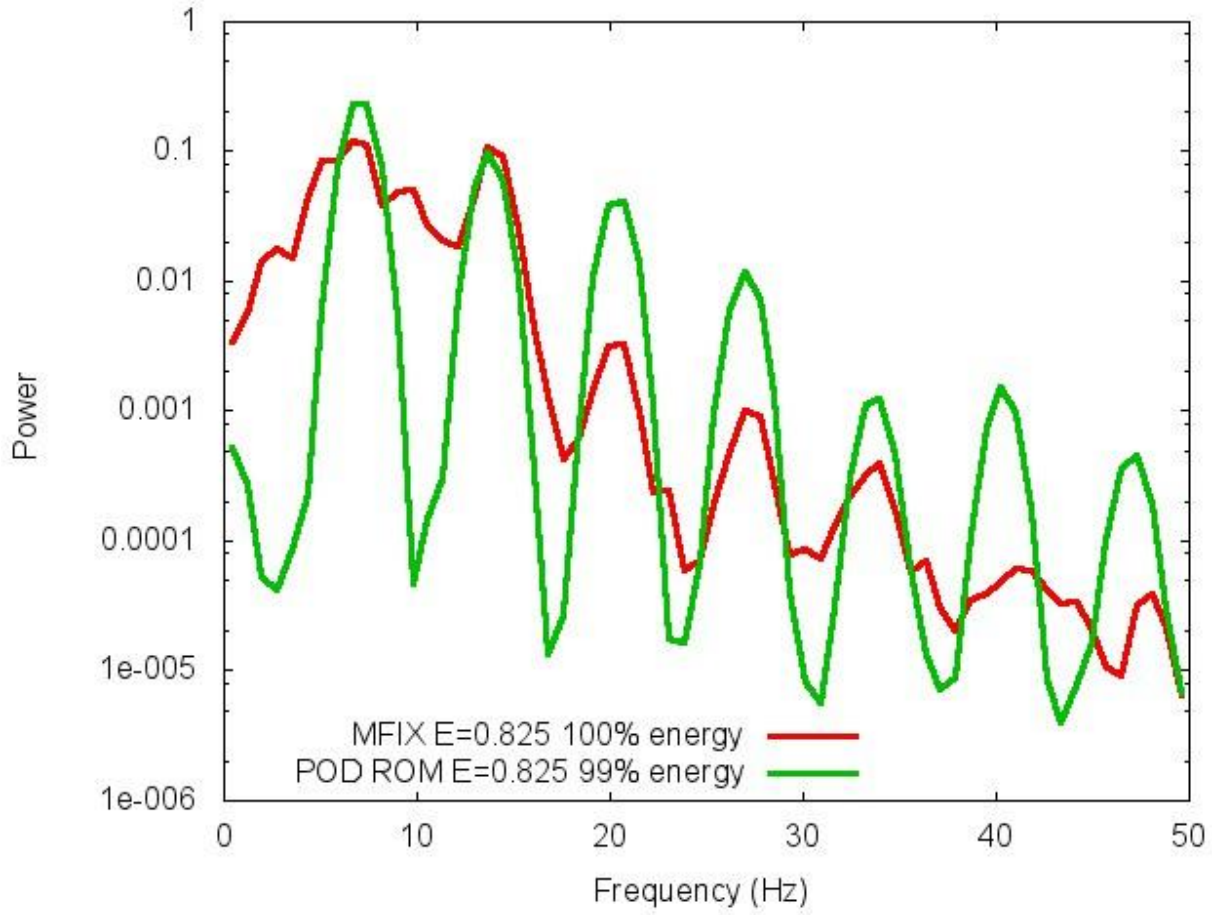
In Figure 77, the gas pressure in the annular region of the spouted bed has a dominant frequency of 13.671875Hz with power of 0.1466 and a secondary frequency at 7.421875Hz with power 0.9902 for the MFIX validation data set. The extrapolative POD ROM has a dominant frequency 7.421875Hz with power 0.2054 and a secondary frequency at 13.671875Hz with power 0.1664. The identified dominant frequencies and subharmonic frequencies of the MFIX validation data set are identified by the extrapolative POD ROM. While the rank of the two most dominant frequencies identified differ between the models, their locations align. Note that as frequency increases the peak locations exhibit good agreement between the methodologies.

Figure 78: Gas Pressure PSD Along the Wall of the Bed



In Figure 78, the gas pressure along the wall has a dominant frequency of 13.671875Hz for the MFIX validation data set and 7.421875Hz for the extrapolative POD ROM with power of 0.1466 and 0.3446, respectively. Notice again that the POD ROM captures the corresponding power spikes at the same frequency locations as the MFIX validation data set.

Figure 79: Gas Pressure PSD on Top of the Spouted Bed



In Figure 79, the gas pressure at the top of the bed has a dominant frequency of 6.640625Hz for the MFIX validation data set and 7.421875Hz for the extrapolative POD ROM with power of 0.1225 and 0.2358, respectively. Again, the POD ROM identifies the same peak frequency locations as the MFIX validation data set.

Overall general agreement among the locations of the dominant frequencies and the sub-harmonic frequencies throughout the spectral analysis of gas pressure within the fluidized bed indicates the results of the presented POD ROM extrapolative methodology is sound. The MFIX validation data set exhibits strong peak frequencies both at 7.421875Hz and 13.671875Hz. The extrapolative POD ROM correctly matches these peak locations as well as most of the subharmonic frequency locations in all four regions.

6.0 CONCLUSION

The reduced order models developed and discussed in this dissertation exhibit similar spatial pattern features and defining characteristics found in the original MFIX data sets. It is easy to identify the strong physical similarities between the reduced order models and the full order models obtained from MFIX. However, it is noted that exact agreement is not present between the models even though a high level of energy capture was enforced on the ROM. Regardless, it is obvious the reduced order data is correlated strongly with its original MFIX data set representation. And, these reduced order models can provide a guide for identification of certain properties exhibited within a full order model although the region identified may cover more cells than the full order model region.

As with any computational model, the results are only as good as the data on which they are based. In fact for ROM construction, this is a governing limitation. The data used for prediction is directly obtained from the original model's data and the equations which govern them. In the event an underlying physical characteristic of the data set is present, but is not depicted in any time stamp from which the ROM is developed, then its presence is not expected in the ROM or any ROM based solver numerical solution. It is for this reason that a sufficient number of uncorrelated snapshots should characterize the construction of a reduced order model.

Due to the increase in accuracy and similarities of results between the POD ROM and the MFIX full order model, interpolative methodology should be used over an extrapolative methodology alone.

The various number of POD basis functions and modes required for energy capture at a variety of levels as depicted in Table 2 illustrates that for predictive calculations, accuracy as well as time of computation, is controlled easily. There is a drastic reduction in the number of POD modes and basis functions required for each variable between 99% and 97.5% energy capture and this slight trade off of accuracy does impact computational time greatly regardless of the number of processors selected.

7.0 FUTURE WORK

The development of reduced order model based solvers can help to provide a computational framework for not only spouted beds, but a variety of fluidized beds with only slight modification. For spouted beds in particular, a significant decrease in the computational time of the development of a ROM could be obtained by reducing the region of interest. For example, a significant portion of the original model's grid contains no data for solid particles. Given this lack of impact on at least one third of the studied domain, focus could be concentrated only on the regions exhibiting particle interaction or those sufficiently near them. This focusing of effort should improve the ability of a ROM to approximate solutions by reducing the regions for which a near approximation is obtained where no significant variable change in the region throughout the time span is considered.

Also, given the obvious restrictions of void fraction of a cell and the current lack of consideration given to it in the presented ROM based solvers, a methodology for restricting or limiting the "overflow" of a fluid ($\varepsilon_g > 1$) or solid ($\varepsilon_g < 0$) within any cell would enhance a reduced order model's accuracy and should be examined. It is possible that weighting the POD modes so that each variable represents a more equitable share of the number of unknown modes at each iteration of the solution process may result in better agreement of the system overall and merits further review.

Furthermore, tolerance and iteration parameters of the Levenberg-Marquardt algorithm were set on a somewhat arbitrary basis and an evaluation of the accuracy/time trade-off should be performed before a recommendation is made as to the appropriateness of the setting of these values. Future work related to this research is needed to investigate the relationship between two-dimensional POD ROMs and the three-dimensional analytical models and data sets that they can accurately mimic. Also, the extension of this work to compare to other computational software models is a natural next step in addition to a comparison against experimental observations.

It is the belief of the author that the most necessary area of future exploration of the methodology presented in this document is a more in depth examination of the influence of

convergence tolerance levels and selection of matrix energy levels and how the selection of these parameters influences the accuracy of the resulting solutions. At present time, the answer to this is unknown.

Finally, conclusions based upon standard linear POD approaches may be very different from the presented nonlinear framework. This is due in large part to the fact that the governing computational factors lie more in the convergence parameters of the nonlinear solvers and the number of desired solution output times as opposed to the number of POD basis functions which are controlled by desired matrix energy levels. One future direction might be to implement a variable specific weighting scheme based upon energy contribution. This could be done by weighting the residual of the variable squared multiplied by the proportion of energy this unknown POD coefficient should contribute. Finally, each variable could be scaled to represent only the respective proportion of their unknown variable to more equally weight the unknown variable's representation.

8.0 BIBLIOGRAPHY

[1] Burkardt J, et. al., "POD and CVT-based Reduced Order Modeling of Navier-Stokes flows," *Comput. Methods Appl. Mech. Engrg.*, vol. 196, 2006.

[2] Sirovich L, "Turbulence and the Dynamics of Coherent Structures Part 1: Coherent Structures," *Quarterly of Applied Mathematics*, vol. 3, 1987.

[3] Finney C, Cizmas P, Daw S, O'Brien T Palacios A, "Experimental Analysis and Visualization of Spatio-Temporal Patterns in Spouted Fluidized Beds," *Chaos*, vol. 14, no. 2, 2004.

[4] McCorkle D, Bryden K Suram S, "Proper Orthogonal Decomposition Based Reduced Order Model of a Hydraulic Mixing Nozzle," in *12th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2007.

[5] Weller J, et. al., "Numerical Methods For Low-Order Modeling of Fluid Flows Based on POD," *INT J NUMER METHOD FLUIDS*, vol. 63, 2010.

[6] Cizmas P, O'Brien T Yuan T, "A Reduced-Order Model for a Bubbling Fluidized Bed Based on Proper Orthogonal Decomposition," *Computers & Chemical Engineering*, vol. 30, 2005.

[7] Zhao Y, et. al., "Numerical Simulation of Two Dimensional Spouted Bed with Draft Plates by Discrete Element Methods," *Front Chemical Engineering*, vol. 1, 2008.

[8] Versteeg W, Malalasekra H, *An Introduction to Computational Fluid Dynamics:The Finite Volume Method*.: Prentice Hall, 2007.

[9] Ortega J, Golub G, *Scientific Computing and Differential Equations: An Introduction to Numerical Methods*. San Diego: Academic Press, 1992.

[10] Pelosi G, "The finite-element method, Part I: R. L. Courant (Historical Corner)," *Antennas and Propagation Magazine, IEEE*, vol. 49, no. 2, 2007.

[11] Wikipedia. [Online]. http://en.wikipedia.org/wiki/Finite_element_method

[12] Bernsdorf J, Zeiser T, Durst F Breuer M, "Accurate Computations of the Laminar Flow Past

a Square Cylinder," *International Journal of Heat and Fluid Flow*, vol. 21, 2000.

[13] Solberg T, Hjertager BH Mathiesen V, "An experimental and computational study of multiphase flow behaviour in a circulating fluidized bed," *Third International Conference on Multiphase Flow*, vol. 26, 2000.

[14] Burkardt J. (2005, April) FSU. [Online]. http://people.sc.fsu.edu/~jburkardt/presentations/fem.ns5.pdf

[15] Lim CJ, Grace JR He YL, "Spouted bed and spout-fluid bed behaviour in a column of diameter 0.91 m," *The Canadian Journal of Chemical Engineering*, vol. 70, no. 5, 1992.

[16] Pannala S, et. al., "Simulating the Dynamics of Spout-Bed Nuclear Fuel Coaters," *Chem. Vap. Deposition*, vol. 13, 2007.

[17] Passos ML, Chattel A LT, Massarani G Franca AS, "MODELING AND SIMULATION OF AIRFLOW IN SPOUTED BED DRYERS," *Drying Technology*, vol. 16, no. 9, 1998.

[18] Barrozo MAS, Duarte CR, Murata VV, "EXPERIMENTAL AND NUMERICAL STUDY OF SPOUTED BED FLUID DYNAMICS," *Brazilian Journal of Chemical Engineering*, vol. 25, no. 1, 2008.

[19] Seungwon Shin and Damir Juric, "Modeling Three-Dimensional Multiphase Flow Using a Level Contour Reconstruction Method for Front Tracking without Connectivity," *Journal of Computational Physics*, vol. 180, 2002.

[20] Symlal M, O'Brien TJ Benyahia S. (2007, July) MFIX. [Online]. https://www.mfix.org/documentation/MFIXEquations2005-4-4.pdf

[21] Department of Physics MIT. (2004, Fall) Department of Physics. [Online]. http://ocw.mit.edu/courses/physics/8-07-electromagnetism-ii-fall-205/readings/indexnotation.pdf

[22] NETL - DOE. [Online]. www.mfix.org

[23] Chatterjee A, "An Introduction to the Proper Orthogonal Decomposition," *Current Science*, vol. 78, 2000.

[24] Fang F, et. al., "A POD Goal-Oriented Error Measure for Mesh Optimization," *International Journal for Numerical Methods in Fluids*, vol. 63, 2010.

[25] Willcox K, "Unsteady Flow Sensing and Estimation via the Gappy Proper Orthogonal Decomposition," *Computers and Fluids*, vol. 35, 2006.

[26] Du Q, Gunzburger M, and Lee HC Burkardt J, "Reduced Order Modeling of Complex Systems," in *Proceedings of NA03*, Dundee, 2003.

[27] Gutman, "The Energy of a Graph," *Ber. Math. Stat. Sekt.*, vol. 103, 1978.

[28] Fang F, et. al., "Reduced-Order Modelling of an Adaptive Mesh Ocean Model," *International Journal for Numerical Methods in Fluids*, vol. 59, 2009.

[29] Fang F, et. al., "A POD Reduced Order Unstructured Mesh Ocean Modeling Method for Moderate Reynolds Number Flows," *Ocean Modelling*, vol. 28, 2009.

[30] Deane A Kalb V, "An Intrinsic Stabilization Scheme for Proper Orthogonal Decomposition Based Low-Dimensional Models," *Physics of Fluids*, vol. 19, 2007.

[31] Brigham JC, Earls CJ, and Sukumar N Aquino W, "Generalized Finite Element Method Using Proper Orthogonal Decomposition," *International Journal for Numerical Methods in Engineering*, vol. 0, 2008.

[32] Cordier L, et. al., "Calibration of POD Reduced-Order Models Using Tikhonov Regularization," *International Journal for Numerical Methods in Fluids*, vol. 63, 2010.

[33] Basdevant C, Sagaut P Couplet M, "Calibrated Reduced-Order POD-Galerkin System for Fluid Flow Modelling," *Journal of Computational Physics*, vol. 20, 2005.

[34] Cizmas P, O'Brien T, and Breault R Brenner T, "Practical Aspects of the Implementation of Proper Orthogonal Decomposition," in *American Institute of Aeronautics and Astronautics*, 2009.

[35] Palacios A, O'Brien T, Symlal M Cizmas P, "Proper Orthogonal Decomposition of Spatio-Temporal Patterns in Fluidized Beds," vol. 58, 2003.

[36] Richardson B, Brenner T, O'Brien T, Breault R Cizmas P, "Acceleration Techniques for Reduced-Order Models Based on Proper Orthogonal Decomposition," *Journal of Computational Physics*, vol. 227, 2008.

[37] Brezinski, *Projection Methods for Systems of Equations*. Amsterdam: Elsevier, 1997.

[38] Broyden CG, "A Class of Methods for Solving Nonlinear Simultaneous Equations,"

Mathematics of Computation, vol. 19, no. 92, 1965.

[39] (2011) Wolfram Mathematica. [Online]. http://reference.wolfram.com/mathematica/tutorial/UnconstrainedOptimizationGaussNewtonMethods.html

[40] (2011) Wolfram Mathematica. [Online]. http://reference.wolfram.com/mathematica/tutorial/UnconstrainedOptimizationTrustRegionMethods.html

[41] Levenberg K, "A Method for the Solution of Certain Non-Linear Problems in Least Squares," *Quarterly of Applied Mathematics*, vol. 2, 1944.

[42] Marquardt D, "An Algorithm for Least-Squares Estimation of Nonlinear Parameters," *Journal for the Society of Industrial Applied Mathematics*, vol. 11, no. 2, 1963.

[43] More JJ, "The Levenberg-Marquardt Algorithm; Implementation and Theory," in *Numerical Analysis; Proceedings of the Biennial Conference*, 1977.

[44] Nielson HB, Tingleff O, Madsen K, "Methods for Non-Linear Least Squares Problems Informatics and Mathematical Modelling 2nd Edition," Technical University of Denmark, Denmark, 2004.

[45] Working Group on Blood Pressure and Heart Rate Variability. European Society of Hypertension. [Online]. http://www.cbi.dongnocchi.it/glossary/PowerSpectralDensity.html

[46] Smith L. (2002) http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf.

[47] Willcox K, "Model Reduction for Large Scale Systems," in *SIAM Conference on Computational Science and Engineering*, 2007.

APPENDIX 1: JACOBIAN FORMULATION

The partial derivative of the gas momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the gas momentum x-component POD approximation is

$$\frac{\partial}{\partial a_{U_{g_j}}}\left(F_{U_{g x_i}}(\vec{a})\right)$$

$$= \rho_g\, \varphi_{U_{g_j}}(x_i)\left[\frac{\partial}{\partial t}\left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\, dx\, dy \qquad \text{(A.1.1)}$$

$$+ 2\rho_g\left(\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\left[\frac{\partial}{\partial x}\left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\varphi_{U_{g_j}}(x_i)\varphi_{U_{g_1}}(x_i)\, dx\, dy$$

$$+\left[\frac{\partial}{\partial x}\left(\varphi_{U_{g_j}}(x_i)\right)\right]\left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\rho_g\varphi_{U_{g_1}}(x_i)\, dx\, dy$$

$$+\rho_g\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\varphi_{U_{g_j}}(x_i)\,\varphi_{U_{g_1}}(x_i)\, dx\, dy$$

$$+\varphi_{U_{g_j}}(x_i)\rho_g\left[\frac{\partial}{\partial y}\left(\mu_{\varepsilon g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{V g}(x_i)\right.$$
$$\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy$$
$$+\rho_g\left[\mu_{\varepsilon g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\left[\mu_{V g}(x_i)\right.$$
$$\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_j}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy$$
$$-\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial y}\left(\mu_{U g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx$$
$$-\mu_{gt}\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_j}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx$$
$$+\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial y}\left(\mu_{U g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_1}}(x_i)\right)\right]\,dx\,dy$$
$$+\mu_{gt}\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_1}}(x_i)\right)\right]\,dx\,dy$$
$$-\frac{4}{3}\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{U g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dy$$
$$-\frac{4}{3}\mu_{gt}\left[\frac{\partial}{\partial x}\left(\varphi_{U_{g_j}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dy$$

$$+\frac{4}{3}\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{g_1}}(x_i)\right)\right]\,dx\,dy$$
$$+\frac{4}{3}\mu_{gt}\left[\frac{\partial}{\partial x}\left(\varphi_{U_{g_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{g_1}}(x_i)\right)\right]\,dx\,dy$$
$$-\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx$$
$$-\mu_{gt}\left[\frac{\partial}{\partial x}\left(\varphi_{U_{g_j}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx$$
$$+\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_1}}(x_i)\right)\right]\,dx\,dy$$
$$+\left[\frac{\partial}{\partial a_{U_{g_j}}}(\beta_{gs})\right]\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right.$$
$$-\left(\mu_{U_g}(x_i)\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy$$
$$-\beta_{gs}\varphi_{U_{g_j}}(x_i)\varphi_{U_{g_1}}(x_i)\,dx\,dy.$$

The partial derivative of the gas momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the gas momentum y-component POD approximation is

$$
\begin{aligned}
&\frac{\partial}{\partial a_{V_{g_j}}}\left(F_{U_{g_{x_i}}}(\vec{a})\right) \\
&= \rho_g \left[\mu_{U_g}(x_i)\right. \\
&\left. + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\mu_{\varepsilon_g}(x_i)\right.\right. \\
&\left.\left. + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right] \varphi_{V_{g_j}}(x_i)\, \varphi_{U_{g_1}}(x_i)\, dx\, dy \\
&+\rho_g \left[\mu_{\varepsilon_g}(x_i)\right. \\
&\left. + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)\right.\right. \\
&\left.\left. + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right] \varphi_{V_{g_j}}(x_i)\, \varphi_{U_{g_1}}(x_i)\, dx\, dy \\
&-\left[\frac{\partial}{\partial a_{V_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)\right.\right. \\
&\qquad\left.\left. + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right] \varphi_{U_{g_1}}(x_i)\, dx \\
&+\left[\frac{\partial}{\partial a_{V_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)\right.\right. \\
&\left.\left. + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right] \left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_1}}(x_i)\right)\right] dx\, dy \\
&-\frac{4}{3}\left[\frac{\partial}{\partial a_{V_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right. \\
&\qquad\left.\left. + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right] \varphi_{U_{g_1}}(x_i)\, dy
\end{aligned}
\qquad \text{(A.1.2)}
$$

$$
\begin{aligned}
&+\frac{4}{3}\left[\frac{\partial}{\partial a_{V_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.\\
&\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{g_1}}(x_i)\right)\right]dx\,dy\\
&-\left[\frac{\partial}{\partial a_{V_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx\\
&+\left[\frac{\partial}{\partial a_{V_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right.\\
&\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_1}}(x_i)\right)\right]\,dx\,dy\\
&+\mu_{gt}\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_1}}(x_i)\right)\right]\,dx\,dy\\
&+\left[\frac{\partial}{\partial a_{V_{g_j}}}(\beta_{gs})\right]\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right.\\
&\left.-\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy.
\end{aligned}
$$

The partial derivative of the gas momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the solid momentum x-component POD approximation is

$$
\begin{aligned}
&\frac{\partial}{\partial a_{U_{s_j}}}(F_{U_{g_{x_i}}}(\vec{a}))\\
&=\left[\frac{\partial}{\partial a_{U_{s_j}}}(\beta_{gs})\right]\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right.\\
&\left.-\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy
\end{aligned}
\qquad (A.1.3)
$$

$$+ \beta_{gs}\varphi_{U_{s_j}}(x_i)\varphi_{U_{g_1}}(x_i)\, dx\, dy.$$

The partial derivative of the gas momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the solid momentum y-component POD approximation is

$$\begin{aligned}
&\frac{\partial}{\partial a_{V_{s_j}}}\left(F_{U_{g_{x_i}}}(\vec{a})\right) \\
&= \left[\frac{\partial}{\partial a_{V_{s_j}}}\left(\beta_{gs}\right)\right]\left[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right. \\
&\left. - \left(\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\, dx\, dy.
\end{aligned} \qquad \text{(A.1.4)}$$

.

The partial derivative of the gas momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the void fraction POD approximation is

$$\begin{aligned}
&\frac{\partial}{\partial a_{\varepsilon_{g_j}}}\left(F_{U_{g_{x_i}}}(\vec{a})\right) \\
&= \rho_g\varphi_{\varepsilon_{g_j}}(x_i)\left[\frac{\partial}{\partial t}\left(\mu_{U_g}(x_i)\right.\right. \\
&\left.\left. + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\, dx\, dy \\
&+\rho_g\left[\frac{\partial}{\partial x}\left(\varphi_{\varepsilon_{g_j}}(x_i)\right)\right]\left[\mu_{U_g}(x_i)\right. \\
&\left. + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]^2\varphi_{U_{g_1}}(x_i)\, dx\, dy
\end{aligned} \qquad \text{(A.1.5)}$$

$$+\rho_g \varphi_{\varepsilon_{g_j}}(x_i)\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy$$

$$+\rho_g\left[\frac{\partial}{\partial y}\left(\varphi_{\varepsilon_{g_j}}(x_i)\right)\right]\left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\left[\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy$$

$$+\rho_g \varphi_{\varepsilon_{g_j}}(x_i)\left[\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy$$

$$+\varphi_{\varepsilon_{g_j}}(x_i)\left[\frac{\partial}{\partial x}\left(\mu_{P_g}(x_i) + \sum_{k=1}^{NGP} a_{P_{g_k}}(t)\varphi_{P_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy$$

$$-\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy$$

$$+\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_1}}(x_i)\right)\right]dx\,dy$$
$$-\frac{4}{3}\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dy$$
$$+\frac{4}{3}\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{g_1}}(x_i)\right)\right]dx\,dy$$
$$-\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx$$
$$+\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_1}}(x_i)\right)\right]dx\,dy$$
$$+\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\beta_{gs})\right]\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right.$$
$$\left.-\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy.$$

The partial derivative of the gas momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the gas pressure POD approximation is

$$\frac{\partial}{\partial a_{P_{g_j}}}(F_{U_{g_{x_i}}}(\vec{a}))$$
$$= \left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{P_{g_j}}(x_i)\right)\right]\varphi_{U_{g_1}}(x_i)\,dx\,dy. \quad \text{(A.1.6)}$$

The partial derivative of the gas momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the solid pressure POD approximation is

$$\frac{\partial}{\partial a_{P_{s_j}}}(F_{U_{g_{x_i}}}(\vec{a})) = 0. \quad \text{(A.1.7)}$$

The partial derivative of the gas momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the solid's temperature POD approximation is

$$\frac{\partial}{\partial a_{T_{s_j}}}(F_{U_{g_{x_i}}}(\vec{a})) = 0. \quad \text{(A.1.8)}$$

The partial derivative of the gas momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the gas momentum x-component POD approximation is

$$\frac{\partial}{\partial a_{U_{g_j}}}(F_{V_{g_{x_i}}}(\vec{a}))$$
$$= \left[\frac{\partial}{\partial x}\left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\rho_g\varphi_{U_{g_j}}(x_i)\left[\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy \quad \text{(A.1.9)}$$

$$+\rho_g \varphi_{U_{g_j}}(x_i)\left[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy$$

$$-\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{V g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx$$

$$-\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{V g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx\,dy$$

$$-\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\mu_{V g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dy$$

$$+\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{V g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx\,dy$$

$$-\frac{4}{3}\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\mu_{V g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx$$

$$+\ \frac{4}{3}\left[\frac{\partial}{\partial a_{U_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_g}(x_i)\right.\right.$$
$$\left.\left.+\ \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]\ dx\ dy$$
$$+\left[\frac{\partial}{\partial a_{U_{g_j}}}(\beta_{gs})\right]\left[\mu_{V_s}(x_i)+\ \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right.$$
$$\left.-\left(\mu_{V_g}(x_i)+\ \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\ dx\ dy.$$

The partial derivative of the gas momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the gas momentum y-component POD approximation is

$$\frac{\partial}{\partial a_{V_{g_j}}}(F_{V_{g_{x_i}}}(\vec{a}))$$
$$=\ \rho_g\left[\frac{\partial}{\partial t}\left(\mu_{\varepsilon_g}(x_i)\right.\right. \qquad \text{(A.1.10)}$$
$$\left.\left.+\ \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\varphi_{V_{g_j}}(x_i)\varphi_{V_{g_1}}(x_i)\ dx\ dy$$
$$+\rho_g\left[\frac{\partial}{\partial x}\left(\mu_{\varepsilon_g}(x_i)+\ \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{U_g}(x_i)\right.$$
$$\left.+\ \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\varphi_{V_{g_j}}(x_i)\varphi_{V_{g_1}}(x_i)\ dx\ dy$$
$$+2\rho_g\left[\frac{\partial}{\partial y}\left(\mu_{\varepsilon_g}(x_i)+\ \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{V_g}(x_i)\right.$$
$$\left.+\ \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\varphi_{V_{g_j}}(x_i)\varphi_{V_{g_1}}(x_i)\ dx\ dy$$

$$+\rho_g\left[\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\left[\mu_{U_g}(x_i)\right.$$

$$\left.+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_j}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy$$

$$-\left[\frac{\partial}{\partial a_{V_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right.$$

$$\left.\left.+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx$$

$$+\mu_{gt}\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_j}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx$$

$$+\left[\frac{\partial}{\partial a_{V_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right.$$

$$\left.\left.+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx\,dy$$

$$+\mu_{gt}\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx\,dy$$

$$-\left[\frac{\partial}{\partial a_{V_{g_j}}}(\mu_{gt})\right]\left[\mu_{V_g}(x_i)\right.$$

$$\left.+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dy$$

$$+\mu_{gt}\varphi_{V_{g_j}}(x_i)\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dy$$

$$+\left[\frac{\partial}{\partial a_{V_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right.$$

$$\left.\left.+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx\,dy$$

$$+\mu_{gt}\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx\,dy$$

$$
\begin{aligned}
&-\frac{4}{3}\left[\frac{\partial}{\partial a_{V_{g_j}}}\left(\mu_{gt}\right)\right]\left[\mu_{V_g}(x_i)\right.\\
&\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx\\
&-\frac{4}{3}\,\mu_{gt}\,\varphi_{V_{g_j}}(x_i)\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx\\
&+\frac{4}{3}\left[\frac{\partial}{\partial a_{V_{g_j}}}\left(\mu_{gt}\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_g}(x_i)\right.\right.\\
&\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx\,dy\\
&+\frac{4}{3}\,\mu_{gt}\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx\,dy\\
&+\left[\frac{\partial}{\partial a_{V_{g_j}}}\left(\beta_{gs}\right)\right]\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right.\\
&\qquad-\left(\mu_{V_g}(x_i)\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy\\
&\qquad-\beta_{gs}\varphi_{V_{g_j}}(x_i)\varphi_{V_{g_1}}(x_i)\,dx\,dy.
\end{aligned}
$$

The partial derivative of the gas momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the solid momentum x-component POD approximation is

$$\frac{\partial}{\partial a_{U_{s_j}}}\left(F_{V_{g x_i}}(\vec{a})\right)$$

$$= \left[\frac{\partial}{\partial a_{U_{s_j}}}\left(\beta_{gs}\right)\right]\left[\mu_{V_s}(x_i) + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right. \qquad \text{(A.1.11)}$$

$$\left. - \left(\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\, dx\, dy.$$

The partial derivative of the gas momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the solid momentum y-component POD approximation is

$$\frac{\partial}{\partial a_{V_{s_j}}}\left(F_{V_{g x_i}}(\vec{a})\right)$$

$$= \left[\frac{\partial}{\partial a_{V_{s_j}}}\left(\beta_{gs}\right)\right]\left[\mu_{V_s}(x_i)\right.$$

$$\left. + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right. \qquad \text{(A.1.12)}$$

$$\left. - \left(\mu_{V_g}(x_i)\right.\right.$$

$$\left.\left. + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\, dx\, dy$$

$$+ \beta_{gs}\varphi_{V_{s_j}}(x_i)\varphi_{V_{g_1}}(x_i)\, dx\, dy.$$

The partial derivative of the gas momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the void fraction POD approximation is

$$\frac{\partial}{\partial a_{\varepsilon_{g_j}}}\left(F_{V_{g_{x_i}}}(\vec{a})\right)$$

$$= \rho_g \varphi_{\varepsilon_{g_j}}(x_i)\left[\frac{\partial}{\partial t}\left(\mu_{V\,g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy \qquad \text{(A.1.13)}$$

$$+\rho_g\left[\frac{\partial}{\partial x}\left(\varphi_{\varepsilon_{g_j}}(x_i)\right)\right]\left[\mu_{U\,g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\left[\mu_{V\,g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy$$

$$+\rho_g\left[\frac{\partial}{\partial y}\left(\varphi_{\varepsilon_{g_j}}(x_i)\right)\right]\left[\mu_{V\,g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]^2\varphi_{V_{g_1}}(x_i)\,dx\,dy$$

$$+\rho_g\varphi_{\varepsilon_{g_j}}(x_i)\left[\mu_{U\,g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V\,g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy$$

$$-\varphi_{\varepsilon_{g_j}}(x_i)\left[\frac{\partial}{\partial y}\left(\mu_{P\,g}(x_i) + \sum_{k=1}^{NGP} a_{P_{g_k}}(t)\varphi_{P_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy$$

$$+ 980.665\rho_g\varphi_{\varepsilon_{g_j}}(x_i)\,\varphi_{V_{g_1}}(x_i)\,dx\,dy$$

$$
\begin{aligned}
&- \left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right. \\
&\qquad \left.\left. + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx \\
&+ \left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right. \\
&+ \left.\left. \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]\,dx\,dy \\
&\quad - \left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\mu_{V_g}(x_i)\right. \\
&\quad + \left. \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dy \\
&+ \left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right. \\
&+ \left.\left. \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_1}}(x_i)\right)\right]dx\,dy \\
&\quad - \frac{4}{3}\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\mu_{V_g}(x_i)\right. \\
&\quad + \left. \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]\,dx \\
&+ \frac{4}{3}\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_{gt})\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_g}(x_i)\right.\right. \\
&+ \left.\left. \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_1}}(x_i)\right)\right]\,dx\,dy
\end{aligned}
$$

$$+\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\beta_{gs})\right]\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right.$$
$$\left.-\left(\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy.$$

The partial derivative of the gas momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the gas pressure POD approximation is

$$\frac{\partial}{\partial a_{P_{g_j}}}(F_{V_{g_{x_i}}}(\vec{a}))$$
$$=-\left[\mu_{\varepsilon_g}(x_i)\right. \qquad \text{(A.1.15)}$$
$$\left.+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{P_{g_j}}(x_i)\right)\right]\varphi_{V_{g_1}}(x_i)\,dx\,dy.$$

The partial derivative of the gas momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the solid pressure POD approximation is

$$\frac{\partial}{\partial a_{P_{s_j}}}(F_{V_{g_{x_i}}}(\vec{a}))=0. \qquad \text{(A.1.16)}$$

The partial derivative of the gas momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the solid's temperature POD approximation is

$$\frac{\partial}{\partial a_{T_{s_j}}}(F_{V_{g_{x_i}}}(\vec{a}))=0. \qquad \text{(A.1.17)}$$

The partial derivative of the solid momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the gas momentum x-component POD approximation is

$$\frac{\partial}{\partial a_{U_{g_j}}}(F_{U_{s_{x_i}}}(\vec{a}))$$

$$= \left[\frac{\partial}{\partial a_{U_{g_j}}}(\beta_{gs})\right]\left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i) - \left(\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy$$

$$+ \beta_{gs}\varphi_{U_{g_j}}(x_i)\varphi_{U_{s_1}}(x_i)\,dx\,dy\,. \qquad \text{(A.1.18)}$$

The partial derivative of the solid momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the gas momentum y-component POD approximation is

$$\frac{\partial}{\partial a_{V_{g_j}}}(F_{U_{s_{x_i}}}(\vec{a}))$$

$$= \left[\frac{\partial}{\partial a_{V_{g_j}}}(\beta_{gs})\right]\left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i) - \left(\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy. \qquad \text{(A.1.18)}$$

The partial derivative of the solid momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the solid momentum x-component POD approximation is

$$\frac{\partial}{\partial a_{U_{s_j}}}\left(F_{U_{s_{x_i}}}(\vec{a})\right)$$

$$= \rho_s \left[-\frac{\partial}{\partial t}\left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i) \right) \right] \varphi_{U_{s_j}}(x_i)\, \varphi_{U_{s_1}}(x_i)\, dx\, dy \qquad \text{(A.1.19)}$$

$$-2\rho_s \left[\frac{\partial}{\partial x}\left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i) \right) \right] \left[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i) \right] \varphi_{U_{s_1}}(x_i)\, dx\, dy$$

$$+\rho_s \left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i) \right) \right] \left[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i) \right] \left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_j}}(x_i) \right) \right] \varphi_{U_{s_1}}(x_i)\, dx\, dy$$

$$+\rho_s \left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i) \right) \right] \left[\frac{\partial}{\partial x}\left(\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i) \right) \right] \varphi_{U_{s_j}}(x_i)\varphi_{U_{s_1}}(x_i)\, dx\, dy$$

$$-\rho_s \left[\frac{\partial}{\partial y}\left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i) \right) \right] \left[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i) \right] \varphi_{U_{s_j}}(x_i)\varphi_{U_{s_1}}(x_i)\, dx\, dy$$

$$
\begin{aligned}
&+\rho_s\Bigg[1 \\
&-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\Bigg]\Bigg[\mu_{V_s}(x_i) \\
&+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\Bigg]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{s_j}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy \\
&-\frac{1}{2}(1+E)\left[\frac{\partial}{\partial x}(\mu_b)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_j}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy \\
&-\frac{1}{2}(1+E)\mu_b\left[\frac{\partial^2}{\partial x^2}\left(\varphi_{U_{s_j}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy \\
&-\mu_s\varphi_{U_{s_j}}(x_i)\left[\frac{\partial}{\partial y}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dx \\
&+\mu_p\left[\frac{\partial}{\partial y}\left(\varphi_{U_{s_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dxdy \\
&-\frac{4}{3}\,\mu_s\varphi_{U_{s_j}}(x_i)\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dy \\
&+\frac{4}{3}\,\mu_s\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dxdy \\
&-\mu_s\varphi_{U_{s_j}}(x_i)\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dx \\
&+\left[\frac{\partial}{\partial a_{U_{s_j}}}\left(\beta_{gs}\right)\right]\Bigg[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i) \\
&-\Bigg(\mu_{U_s}(x_i) \\
&+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\Bigg)\Bigg]\varphi_{U_{s_1}}(x_i)\,dx\,dy \\
&+\beta_{gs}\varphi_{U_{s_j}}(x_i)\varphi_{U_{s_1}}(x_i)\,dx\,dy.
\end{aligned}
$$

The partial derivative of the solid momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the solid momentum y-component POD approximation is

$$\frac{\partial}{\partial a_{V_{s_j}}}(F_{U_{s_{x_i}}}(\vec{a}))$$

$$= -\rho_s\left[\frac{\partial}{\partial y}\left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\varphi_{V_{s_j}}(x_i)\varphi_{U_{s_1}}(x_i)\,dx\,dy \qquad \text{(A.1.20)}$$

$$+ \rho_s\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\varphi_{V_{s_j}}(x_i)\varphi_{U_{s_1}}(x_i)\,dx\,dy$$

$$+ \mu_s\left[\frac{\partial}{\partial x}\left(\varphi_{V_{s_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dxdy$$

$$+ \left[\frac{\partial}{\partial a_{V_{s_j}}}(\beta_{gs})\right]\left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i) - \left(\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy.$$

The partial derivative of the solid momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the void fraction POD approximation is

$$\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(F_{U_{s_{x_i}}}(\vec{a}))$$

$$= -\rho_s\varphi_{\varepsilon_{g_j}}(x_i)\left[\frac{\partial}{\partial t}\left(\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy \qquad \text{(A.1.21)}$$

$$
\begin{aligned}
&- \rho_s \left[\frac{\partial}{\partial x}\left(\varphi_{\varepsilon g_j}(x_i)\right)\right]\left[\mu_{U_S}(x_i)\right. \\
&\qquad \left. + \sum_{k=1}^{NPU} a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)\right]^2 \varphi_{U_{s_1}}(x_i)\, dx\, dy \\
&-\rho_s \varphi_{\varepsilon g_j}(x_i)\left[\frac{\partial}{\partial x}\left(\mu_{U_S}(x_i)\right.\right. \\
&\qquad \left.\left. + \sum_{k=1}^{NPU} a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)\right)\right]\left[\mu_{U_S}(x_i)\right. \\
&\qquad \left. + \sum_{k=1}^{NPU} a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)\right] \varphi_{U_{s_1}}(x_i)\, dx\, dy \\
&- \rho_s \left[\frac{\partial}{\partial y}\left(\varphi_{\varepsilon g_j}(x_i)\right)\right]\left[\mu_{U_S}(x_i)\right. \\
&\qquad \left. + \sum_{k=1}^{NPU} a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)\right]\left[\mu_{V_S}(x_i)\right. \\
&\qquad \left. + \sum_{k=1}^{NPV} a_{V_{sk}}(t)\varphi_{V_{sk}}(x_i)\right] \varphi_{U_{s_1}}(x_i)\, dx\, dy \\
&-\rho_s \varphi_{\varepsilon g_j}(x_i)\left[\mu_{V_S}(x_i)\right. \\
&\qquad \left. + \sum_{k=1}^{NPV} a_{V_{sk}}(t)\varphi_{V_{sk}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_S}(x_i)\right.\right. \\
&\qquad \left.\left. + \sum_{k=1}^{NPU} a_{U_{sk}}(t)\varphi_{U_{sk}}(x_i)\right)\right] \varphi_{U_{s_1}}(x_i)\, dx\, dy \\
&-\varphi_{\varepsilon g_j}(x_i)\left[\frac{\partial}{\partial x}\left(\mu_{P_g}(x_i)\right.\right. \\
&\qquad \left.\left. + \sum_{k=1}^{NGP} a_{P_{g_k}}(t)\varphi_{P_{g_k}}(x_i)\right)\right] \varphi_{U_{s_1}}(x_i)\, dx\, dy
\end{aligned}
$$

$$
\begin{aligned}
&-\frac{1}{2}(1+E)\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}\left(\frac{\partial}{\partial x}(\mu_b)\right)\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_S}(x_i)\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NPU} a_{U_{S_k}}(t)\varphi_{U_{S_k}}(x_i)\right)\right]\varphi_{U_{S_1}}(x_i)\,dx\,dy\\
&-\frac{1}{2}(1+E)\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_b)\right]\left[\frac{\partial^2}{\partial x^2}\left(\mu_{U_S}(x_i)\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NPU} a_{U_{S_k}}(t)\varphi_{U_{S_k}}(x_i)\right)\right]\varphi_{U_{S_1}}(x_i)\,dx\,dy\\
&\quad-\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\left[\mu_{U_S}(x_i)\right.\\
&\quad\left.+\sum_{k=1}^{NPU} a_{U_{S_k}}(t)\varphi_{U_{S_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{S_1}}(x_i)\right)\right]\,dx\\
&\quad+\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\left[\frac{\partial}{\partial y}\left(\mu_{U_S}(x_i)\right.\right.\\
&\quad\left.\left.+\sum_{k=1}^{NPU} a_{U_{S_k}}(t)\varphi_{U_{S_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{S_1}}(x_i)\right)\right]\,dx\,dy\\
&\quad-\frac{4}{3}\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\right]\left[\mu_{U_S}(x_i)\right.\\
&\quad\left.+\sum_{k=1}^{NPU} a_{U_{S_k}}(t)\varphi_{U_{S_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{S_1}}(x_i)\right)\right]\,dy\\
&\quad+\frac{4}{3}\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_S}(x_i)\right.\right.\\
&\quad\left.\left.+\sum_{k=1}^{NPU} a_{U_{S_k}}(t)\varphi_{U_{S_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{S_1}}(x_i)\right)\right]\,dx\,dy
\end{aligned}
$$

$$
\begin{aligned}
&- \frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\left[\mu_{U_s}(x_i)\right. \\
&\left. + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right] dx \\
&+ \frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\left[\frac{\partial}{\partial x}\left(\mu_{V_s}(x_i)\right.\right. \\
&\left.\left. + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right] dx\, dy \\
&+ \left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\beta_{gs})\right]\left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right. \\
&\left. - \left(\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\, dx\, dy.
\end{aligned}
$$

The partial derivative of the solid momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the gas pressure POD approximation is

$$
\begin{aligned}
&\frac{\partial}{\partial a_{P_{g_j}}}(F_{U_{s x_i}}(\vec{a})) \\
&= \left[1\right. \\
&\left. - \left(\mu_{\varepsilon_g}(x_i)\right.\right. \\
&\left.\left. + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{P_{g_j}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\, dx\, dy.
\end{aligned}
\qquad \text{(A.1.22)}
$$

The partial derivative of the solid momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the solid pressure POD approximation is

$$
\frac{\partial}{\partial a_{P_{s_j}}}(F_{U_{s x_i}}(\vec{a})) = \left[\frac{\partial}{\partial x}\left(\varphi_{P_{s_j}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\, dx\, dy. \qquad \text{(A.1.23)}
$$

The partial derivative of the solid momentum's x-component of the *i*th cell with respect to the *j*th POD coefficient of the solid's temperature POD approximation is

$$
\begin{aligned}
&\frac{\partial}{\partial a_{\theta_{s_j}}}\left(F_{U_{s_{x_i}}}(\vec{a})\right) \\
&\qquad = -\frac{1}{2}(1 \\
&\qquad\quad + E)\left[\frac{\partial}{\partial a_{\theta_{s_j}}}\left(\frac{\partial}{\partial x}\left(\mu_b\right)\right)\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_s}(x_i)\right.\right. \\
&\qquad\quad \left.\left. + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy \\
&-\frac{1}{2}(1+E)\left[\frac{\partial}{\partial a_{\theta_{s_j}}}\left(\mu_b\right)\right]\left[\frac{\partial^2}{\partial x^2}\left(\mu_{U_s}(x_i)\right.\right. \\
&\qquad\quad \left.\left. + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy \\
&\quad - \frac{\partial}{\partial a_{\theta_{s_j}}}\left(\mu_s\right)\left[\mu_{U_s}(x_i)\right. \\
&\quad \left. + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dx \\
&+\frac{\partial}{\partial a_{\theta_{s_j}}}\left(\mu_s\right)\left[\frac{\partial}{\partial y}\left(\mu_{U_s}(x_i)\right.\right. \\
&\qquad\quad \left.\left. + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy \\
&\quad - \frac{4}{3}\left[\frac{\partial}{\partial a_{\theta_{s_j}}}\left(\mu_s\right)\right]\left[\mu_{U_s}(x_i)\right. \\
&\quad \left. + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dy
\end{aligned}
\qquad \text{(A.1.24)}
$$

$$+\frac{4}{3}\left[\frac{\partial}{\partial a_{\theta_{s_j}}}(\mu_s)\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_s}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dx\,dy$$
$$-\frac{\partial}{\partial a_{\theta_{s_j}}}(\mu_s)\left[\mu_{U_s}(x_i)\right.$$
$$\left.+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dx$$
$$+\frac{\partial}{\partial a_{\theta_{s_j}}}(\mu_s)\left[\frac{\partial}{\partial x}\left(\mu_{V_s}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{s_1}}(x_i)\right)\right]dx\,dy$$
$$+\left[\frac{\partial}{\partial a_{\theta_{s_j}}}(\beta_{gs})\right]\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right.$$
$$\left.-\left(\mu_{U_s}(x_i)+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right)\right]\varphi_{U_{s_1}}(x_i)\,dx\,dy.$$

The partial derivative of the solid momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the gas momentum x-component POD approximation is

$$\frac{\partial}{\partial a_{U_{g_j}}}\left(F_{V_{s_{x_i}}}(\vec{a})\right)$$

$$= \left[\frac{\partial}{\partial a_{U_{g_j}}}\left(\beta_{gs}\right)\right]\left[\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i) - \left(\mu_{V_s}(x_i) + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy. \quad \text{(A.1.25)}$$

The partial derivative of the solid momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the gas momentum y-component POD approximation is

$$\frac{\partial}{\partial a_{V_{g_j}}}\left(F_{V_{s_{x_i}}}(\vec{a})\right)$$

$$= \left[\frac{\partial}{\partial a_{V_{g_j}}}\left(\beta_{gs}\right)\right]\left[\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i) - \left(\mu_{V_s}(x_i) + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy + \beta_{gs}\varphi_{V_{g_j}}(x_i)\varphi_{V_{s_1}}(x_i)\,dx\,dy \quad \text{(A.1.26)}$$

The partial derivative of the solid momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the solid momentum x-component POD approximation is

$$\frac{\partial}{\partial a_{U_{s_j}}}\left(F_{V_{s_{x_i}}}(\vec{a})\right)$$

$$= \rho_s \left[\frac{\partial}{\partial x}\left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{V_s}(x_i) + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\varphi_{U_{s_j}}(x_i)\varphi_{V_{s_1}}(x_i)\,dx\,dy \qquad \text{(A.1.27)}$$

$$+ \left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_s}(x_i) + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\rho_s\varphi_{U_{s_j}}(x_i)\varphi_{V_{s_1}}(x_i)\,dx\,dy$$

$$+ \left[\frac{\partial}{\partial a_{U_{s_j}}}(\beta_{gs})\right]\left[\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i) - \left(\mu_{V_s}(x_i) + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy.$$

The partial derivative of the solid momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the solid momentum y-component POD approximation is

$$
\begin{aligned}
&\frac{\partial}{\partial a_{V_{s_j}}}\left(F_{V_{s_{x_i}}}(\vec{a})\right) \\
&= \Bigg[1 \\
&- \Bigg(\mu_{\varepsilon_g}(x_i) \\
&+ \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg)\Bigg]\rho_s\varphi_{V_{s_j}}(x_i)\varphi_{V_{s_1}}(x_i)\,dx\,dy \\
&\quad - \rho_s\left[\frac{\partial}{\partial x}\Bigg(\mu_{\varepsilon_g}(x_i)\right. \\
&\quad \left.+ \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg)\right]\varphi_{V_{s_j}}(x_i)\left[\mu_{U_s}(x_i)\right. \\
&\quad \left.+ \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy \\
&-2\rho_s\left[\frac{\partial}{\partial y}\left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{V_s}(x_i)\right. \\
&\qquad\qquad \left.+ \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy \\
&+ \rho_s\Bigg[1 \\
&- \Bigg(\mu_{\varepsilon_g}(x_i) \\
&+ \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg)\Bigg]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_j}}(x_i)\right)\right]\Bigg[\mu_{V_s}(x_i) \\
&+ \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\Bigg]\varphi_{V_{s_1}}(x_i)\,dx\,dy
\end{aligned}
\qquad \text{(A.1.28)}
$$

$$
\begin{aligned}
&+\rho_s\Bigg[1\\
&-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\Bigg]\Bigg[\frac{\partial}{\partial y}\Bigg(\mu_{V_s}(x_i)\\
&+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\Bigg)\Bigg]\varphi_{V_{s_j}}(x_i)\varphi_{V_{s_1}}(x_i)\,dx\,dy\\
&+\rho_s\Bigg[1\\
&-\Bigg(\mu_{\varepsilon_g}(x_i)\\
&+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg)\Bigg]\Bigg[\frac{\partial}{\partial x}\left(\varphi_{V_{s_j}}(x_i)\right)\Bigg]\Bigg[\mu_{U_s}(x_i)\\
&+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\Bigg]\varphi_{V_{s_1}}(x_i)\,dx\,dy\\
&-\frac{1}{2}(1+E)\left[\frac{\partial}{\partial y}(\mu_b)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_j}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy\\
&-\frac{1}{2}(1+E)\mu_b\left[\frac{\partial^2}{\partial x^2}\left(\varphi_{V_{s_j}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy\\
&-\mu_s\left[\frac{\partial}{\partial x}\left(\varphi_{V_{s_j}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy\\
&+\mu_s\left[\frac{\partial}{\partial x}\left(\varphi_{V_{s_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_1}}(x_i)\right)\right]\,dx\,dy\\
&-\mu_s\varphi_{V_{s_j}}(x_i)\left[\frac{\partial}{\partial x}\left(\varphi_{V_{s_1}}(x_i)\right)\right]\,dy\\
&+\mu_s\left[\frac{\partial}{\partial x}\left(\varphi_{V_{s_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{s_1}}(x_i)\right)\right]dx\,dy\\
&-\frac{4}{3}\mu_s\varphi_{V_{s_j}}(x_i)\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_1}}(x_i)\right)\right]\,dx\\
&+\frac{4}{3}\mu_s\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_1}}(x_i)\right)\right]\,dx\,dy
\end{aligned}
$$

$$+\left[\frac{\partial}{\partial a_{V_{s_j}}}(\beta_{gs})\right]\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right.$$
$$\left.-\left(\mu_{V_s}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy.$$

The partial derivative of the solid momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the void fraction POD approximation is

$$\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(F_{V_{s_{x_i}}}(\vec{a}))$$
$$=-\varphi_{\varepsilon_{g_j}}(x_i)\rho_s\left[\mu_{V_s}(x_i)\right. \qquad \text{(A.1.29)}$$
$$\left.+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy$$
$$-\varphi_{\varepsilon_{g_j}}(x_i)\rho_s\left[\frac{\partial}{\partial t}\left(\mu_{V_s}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy$$
$$-\rho_s\left[\frac{\partial}{\partial x}\left(\varphi_{\varepsilon_{g_j}}(x_i)\right)\right]\left[\mu_{V_s}(x_i)\right.$$
$$\left.+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\left[\mu_{U_s}(x_i)\right.$$
$$\left.+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy$$
$$-\rho_s\left[\frac{\partial}{\partial y}\left(\varphi_{\varepsilon_{g_j}}(x_i)\right)\right]$$

$$\left[\mu_{V_S}(x_i) + \sum_{k=1}^{NPV} a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right]^2 \varphi_{V_{S_1}}(x_i)\, dx\, dy$$

$$- \varphi_{\varepsilon_{g_j}}(x_i)\rho_s \left[\frac{\partial}{\partial y}\left(\mu_{V_S}(x_i) + \sum_{k=1}^{NPV} a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right)\right]\left[\mu_{V_S}(x_i) + \sum_{k=1}^{NPV} a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right]\varphi_{V_{S_1}}(x_i)\, dx\, dy$$

$$- \varphi_{\varepsilon_{g_j}}(x_i)\rho_s \left[\mu_{U_S}(x_i) + \sum_{k=1}^{NPU} a_{U_{S_k}}(t)\varphi_{U_{S_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_S}(x_i) + \sum_{k=1}^{NPV} a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right)\right]\varphi_{V_{S_1}}(x_i)\, dx\, dy$$

$$- \frac{1}{2}(1 + E)\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}\left(\frac{\partial}{\partial y}(\mu_b)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_S}(x_i) + \sum_{k=1}^{NPV} a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right)\right]\varphi_{V_{S_1}}(x_i)\, dx\, dy$$

$$- \frac{1}{2}(1 + E)\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_b)\right]\left[\frac{\partial^2}{\partial y^2}\left(\mu_{V_S}(x_i) + \sum_{k=1}^{NPV} a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right)\right]\varphi_{V_{S_1}}(x_i)\, dx\, dy$$

$$- \left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_S}(x_i) + \sum_{k=1}^{NPV} a_{V_{S_k}}(t)\varphi_{V_{S_k}}(x_i)\right)\right]\varphi_{V_{S_1}}(x_i)\, dx$$

$$+\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_s}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_1}}(x_i)\right)\right]\,dx\,dy$$
$$-\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\right]\left[\mu_{V_s}(x_i)\right.$$
$$\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{s_1}}(x_i)\right)\right]\,dy$$
$$+\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_s}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{s_1}}(x_i)\right)\right]\,dx\,dy$$
$$-\frac{4}{3}\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\right]\left[\mu_{V_s}(x_i)\right.$$
$$\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_1}}(x_i)\right)\right]\,dx$$
$$+\frac{4}{3}\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_s)\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_s}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_1}}(x_i)\right)\right]\,dx\,dy$$
$$+\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\beta_{gs})\right]\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right.$$
$$\left.-\left(\mu_{V_s}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy.$$

The partial derivative of the solid momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the gas pressure POD approximation is

$$\begin{aligned}&\frac{\partial}{\partial a_{P_{g_j}}}(F_{V_{s_{x_i}}}(\vec{a}))\\&=\Bigg[1\\&-\Bigg(\mu_{\varepsilon_g}(x_i)\\&+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg)\Bigg]\left[\frac{\partial}{\partial y}\left(\varphi_{P_{g_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy.\end{aligned}\qquad (A.1.30)$$

The partial derivative of the solid momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the solid pressure POD approximation is

$$\frac{\partial}{\partial a_{P_{s_j}}}(F_{V_{s_{x_i}}}(\vec{a}))=\left[\frac{\partial}{\partial y}\left(\varphi_{P_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy.\qquad (A.1.31)$$

The partial derivative of the solid momentum's y-component of the *i*th cell with respect to the *j*th POD coefficient of the solid's temperature POD approximation is

$$\begin{aligned}&\frac{\partial}{\partial a_{\theta_{s_j}}}(F_{V_{s_{x_i}}}(\vec{a}))\\&=-\frac{1}{2}(1\\&+E)\left[\frac{\partial}{\partial a_{\theta_{s_j}}}\left(\frac{\partial}{\partial y}(\mu_b)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_s}(x_i)\right.\right.\\&\left.\left.+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy\end{aligned}\qquad (A.1.32)$$

$$-\frac{1}{2}(1+E)\left[\frac{\partial}{\partial a_{\theta_{s_j}}}(\mu_b)\right]\left[\frac{\partial^2}{\partial y^2}\left(\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx\,dy$$

$$-\left[\frac{\partial}{\partial a_{\theta_{s_j}}}(\mu_s)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\varphi_{V_{s_1}}(x_i)\,dx$$

$$+\left[\frac{\partial}{\partial a_{\theta_{s_j}}}(\mu_s)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_1}}(x_i)\right)\right]dx\,dy$$

$$-\left[\frac{\partial}{\partial a_{\theta_{s_j}}}(\mu_s)\right]\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{s_1}}(x_i)\right)\right]dy$$

$$+\left[\frac{\partial}{\partial a_{\theta_{s_j}}}(\mu_s)\right]\left[\frac{\partial}{\partial x}\left(\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{s_1}}(x_i)\right)\right]dx\,dy$$

$$-\frac{4}{3}\left[\frac{\partial}{\partial a_{\theta_{s_j}}}(\mu_s)\right]\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_1}}(x_i)\right)\right]dx$$

$$+\frac{4}{3}\left[\frac{\partial}{\partial a_{\theta_{s_j}}}\left(\mu_s\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_s}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{s_1}}(x_i)\right)\right]\,dx\,dy.$$

These governing ROM equations require further clarification for the partial derivatives with respect to the unknown POD coefficients for several variables. They are $\beta_{gs}, \mu_{gt}, \mu_b, \mu_s$.

The partial derivatives of μ_{gt} with respect to the unknown POD coefficients are:

$$\frac{\partial}{\partial a_{U_{g_j}}}\left(\mu_{gt}\right) = \rho_g {\iota_s}^2\left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right] \qquad (A.1.33)$$
$$\left[\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]^2\right.$$
$$-2\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]$$
$$+\left[\frac{\partial}{\partial y}\left(\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]^2$$
$$+\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]^2$$

$$+\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]^2$$
$$+\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\Bigg]^{\frac{1}{2}}$$
$$\left[2\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.\right.$$
$$\left.\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{U_{g_j}}(x_i)\right)\right]\right.$$
$$-2\left[\frac{\partial}{\partial x}\left(\varphi_{U_{g_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_j}}(x_i)\right)\right]$$
$$+2\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_j}}(x_i)\right)\right]$$
$$\left.+\left[\frac{\partial}{\partial y}\left(\varphi_{U_{g_j}}(x_i)\right)\right]\right]$$

$$\frac{\partial}{\partial a_{V_{g_j}}}\left(\mu_{gt}\right)=\rho_g {\iota_s}^2\left[\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right] \qquad \text{(A.1.34)}$$
$$\left[\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]^2\right.$$

$$
\begin{aligned}
&-2\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_g}(x_i)\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\\
&+\left[\frac{\partial}{\partial y}\left(\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]^2\\
&+\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]^2\\
&+\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]^2\\
&+\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\Bigg]^{\frac{1}{2}}\\
&\left[-2\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.\right.\\
&\qquad\left.\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\\
&+2\left[\frac{\partial}{\partial y}\left(\mu_{V_g}(x_i)\right.\right.\\
&\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\varphi_{V_{g_j}}(x_i)\right)\right]
\end{aligned}
$$

$$+2\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_j}}(x_i)\right)\right]$$
$$+\left[\frac{\partial}{\partial x}\left(\varphi_{V_{g_j}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\right]$$

$$\frac{\partial}{\partial a_{U_{s_j}}}\left(\mu_{gt}\right)=0 \qquad \text{(A.1.35)}$$

$$\frac{\partial}{\partial a_{V_{s_j}}}\left(\mu_{gt}\right)=0 \qquad \text{(A.1.36)}$$

$$\frac{\partial}{\partial a_{P_{g_j}}}\left(\mu_{gt}\right)=0 \qquad \text{(A.1.37)}$$

$$\frac{\partial}{\partial a_{\varepsilon_{g_j}}}\left(\mu_{gt}\right)=\rho_g \iota_s{}^2 \varphi_{\varepsilon_{g_j}}(x_i) \qquad \text{(A.1.38)}$$

$$\left[\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]^2\right.$$
$$-2\left[\frac{\partial}{\partial x}\left(\mu_{U_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial y}\left(\mu_{V_g}(x_i)\right.\right.$$
$$\left.\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]$$
$$+\left[\frac{\partial}{\partial y}\left(\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]^2$$

$$+\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\right]^2$$

$$+\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]^2$$

$$+\left[\frac{\partial}{\partial x}\left(\mu_{V_g}(x_i)\right.\right.$$

$$\left.+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right)\left]\left[\frac{\partial}{\partial y}\left(\mu_{U_g}(x_i)\right.\right.\right.$$

$$\left.\left.\left.+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right)\right]\right]^{\frac{1}{2}}$$

$$\frac{\partial}{\partial a_{P_{s_j}}}\left(\mu_{gt}\right)=0 \qquad \text{(A.1.39)}$$

$$\frac{\partial}{\partial a_{\theta_{s_j}}}\left(\mu_{gt}\right)=0\,. \qquad \text{(A.1.40)}$$

The partial derivatives of μ_b with respect to the unknown POD coefficients are:

$$\frac{\partial}{\partial a_{U_{g_j}}}\left(\mu_b\right)=0 \qquad \text{(A.1.40)}$$

$$\frac{\partial}{\partial a_{V_{g_j}}}\left(\mu_b\right)=0 \qquad \text{(A.1.41)}$$

$$\frac{\partial}{\partial a_{U_{s_j}}}\left(\mu_b\right)=0 \qquad \text{(A.1.42)}$$

$$\frac{\partial}{\partial a_{V_{s_j}}}\left(\mu_b\right)=0 \qquad \text{(A.1.43)}$$

$$\frac{\partial}{\partial a_{P_{g_j}}}\left(\mu_b\right)=0 \qquad \text{(A.1.44)}$$

$$\frac{\partial}{\partial a_{P_{s_j}}}(\mu_b) = 0 \tag{A.1.45}$$

$$\begin{aligned}\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_b) = -(\frac{16\sqrt{\pi}}{3\pi}\rho_s d_p g_{0m})\varphi_{\varepsilon_{g_j}}(x_i)\Bigg[1 \\ -\Bigg(\mu_{\varepsilon_g}(x_i) \\ +\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg)\Bigg]\Bigg[\mu_{\theta_s}(x_i) \\ +\sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\Bigg]^{\frac{1}{2}}\end{aligned} \tag{A.1.46}$$

$$\begin{aligned}\frac{\partial}{\partial a_{\theta_{s_j}}}(\mu_b) = (\frac{32}{19\sqrt{\pi}}\rho_s d_p g_{0m})\varphi_{\theta_{s_j}}(x_i)\Bigg[1 \\ -\Bigg(\mu_{\varepsilon_g}(x_i) \\ +\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg)\Bigg]^2\Bigg[\mu_{\theta_s}(x_i) \\ +\sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\Bigg]^{-\frac{1}{2}}.\end{aligned} \tag{A.1.47}$$

The partial derivatives of μ_s with respect to the unknown POD coefficients are:

$$\frac{\partial}{\partial a_{U_{g_j}}}(\mu_b) = 0 \tag{A.1.48}$$

$$\frac{\partial}{\partial a_{V_{g_j}}}(\mu_b) = 0 \tag{A.1.49}$$

$$\frac{\partial}{\partial a_{U_{s_j}}}(\mu_b) = 0 \tag{A.1.50}$$

$$\frac{\partial}{\partial a_{V_{s_j}}}(\mu_b) = 0 \qquad \text{(A.1.51)}$$

$$\frac{\partial}{\partial a_{P_{g_j}}}(\mu_b) = 0 \qquad \text{(A.1.52)}$$

$$\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_b) = \left[\left(\left[\frac{5}{96}\sqrt{\pi}d_p g_{0m}\left[-\left(\varphi_{\varepsilon_{g_j}}(x_i)\right)\right]\left[\mu_{\theta_s}(x_i) + \sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{3}{2}}{\rho_s}^2\right]\right) \qquad \text{(A.1.53)}$$

$$\left(\left[\left[\rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\theta_s}(x_i) + \sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right] + \frac{5}{48}\beta d_p\sqrt{\pi}\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]^{-1}\left[\mu_{\theta_s}(x_i) + \sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{1}{2}}\right]\right]\right)$$

$$-\left(-\rho_s\left(\varphi_{\varepsilon_{g_j}}(x_i)\right)\left[\mu_{\theta_s}(x_i) + \sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]\right.$$

$$+\frac{5}{48}\beta\sqrt{\pi}d_p\left[\mu_{\theta_s}(x_i)+\sum_{k=1}^{NPT}a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{0.5}\left[1\right.$$

$$\left.-\left(\mu_{\varepsilon_g}(x_i)\right.\right.$$

$$\left.\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]^{-2}\varphi_{\varepsilon_{g_j}}(x_i)\Bigg)$$

$$\left(\left[\frac{5}{96}\sqrt{\pi}d_p g_{0m}\left[1\right.\right.\right.$$

$$\left.\left.-\left(\mu_{\varepsilon_g}(x_i)\right.\right.\right.$$

$$\left.\left.\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\theta_s}(x_i)\right.\right.$$

$$\left.\left.\left.+\sum_{k=1}^{NPT}a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{3}{2}}{\rho_s}^2\right]\right)\Bigg]$$

$$\left[\left[\left[\left[\rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\right]\left[\mu_{\theta_s}(x_i)\right.\right.\right.\right.$$

$$\left.\left.\left.\left.+\sum_{k=1}^{NPT}a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]\right.\right.\right.$$

$$+\frac{5}{48}\beta d_p\sqrt{\pi}\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]^{-1}\left[\mu_{\theta_s}(x_i) + \sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{1}{2}}\Bigg]\Bigg]^{-1}$$

$$\left[1.2\left(g_{0m}\eta(2-\eta)\right)^{-1}\right]$$

$$\left[1+\frac{8}{5}\eta\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]g_{0m}\right]$$

$$\left[1+\frac{8}{5}\eta(3\eta-2)\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]g_{0m}\right]$$

$$-1.2\left(1 + \frac{8}{5}\eta(3\eta-2)\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]g_{0m}\right)\frac{8}{5}\eta g_{0m}\varphi_{\varepsilon_{g_j}}(x_i)$$

$$\left[\left[\left[\left[\rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\theta_s}(x_i)+\sum_{k=1}^{NPT}a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]\right.\right.\right.\right.$$

$$\left.\left.\left.\left.+\frac{5}{48}\beta d_p\sqrt{\pi}\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]^{-1}\left[\mu_{\theta_s}(x_i)+\sum_{k=1}^{NPT}a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{1}{2}}\right]\right]\right]^{-1}$$

$$\left(\frac{5}{96}\sqrt{\pi}{\rho_s}^2 g_{0m}d_p\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\theta_s}(x_i)+\sum_{k=1}^{NPT}a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{3}{2}}\right)$$

$$[g_{0m}\eta(2-\eta)]^{-1}$$

$$
\begin{aligned}
&-1.2\left(1+\frac{8}{5}\eta\left[1\right.\right.\\
&\qquad\left.-\left(\mu_{\varepsilon_g}(x_i)\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]g_{0m}\Bigg)\frac{8}{5}\eta(3\eta\\
&\qquad-2)\,\varphi_{\varepsilon_{g_j}}(x_i)g_{0m}\\
&\left[\left[\rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\theta_s}(x_i)\right.\right.\right.\\
&\qquad\left.\left.+\sum_{k=1}^{NPT}a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]\right.\\
&\qquad+\frac{5}{48}\beta d_p\sqrt{\pi}\left[1\right.\\
&\qquad-\left(\mu_{\varepsilon_g}(x_i)\right.\\
&\qquad\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\Bigg]^{-1}\left[\mu_{\theta_s}(x_i)\right.\\
&\qquad\left.\left.\left.+\sum_{k=1}^{NPT}a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{1}{2}}\right]\right]^{-1}
\end{aligned}
$$

$$\left(\frac{5}{96}\sqrt{\pi}{\rho_s}^2 g_{0m} d_p \left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\theta_s}(x_i) + \sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{3}{2}}\right)$$

$$[g_{0m}\eta(2-\eta)]^{-1} + \frac{3.6}{5}\eta\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(\mu_b)\right]$$

$$\frac{\partial}{\partial a_{\theta_{s_j}}}(\mu_s) = 1.2\left(1 + \frac{8}{5}\eta g_{0m}\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\right)$$

$$\left(1 + \frac{8}{5}\eta(3\eta-2)g_{0m}\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\right)$$

$$(g_{0m}\eta(2-\eta))^{-1} \qquad \text{(A.1.54)}$$

$$\left(\left[\rho_s{}^2 g_{0m}\frac{5}{96}d_p\sqrt{\pi}\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\right.\right.$$
$$\left.\left.\left(\frac{3}{2}\left[\mu_{\theta_s}(x_i) + \sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{0.5}\varphi_{\theta_{s_j}}(x_i)\right)\right)$$
$$\left[\left[\rho_s\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\theta_s}(x_i) + \sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]\right.\right.$$
$$\left.\left. + \frac{5}{48}\beta d_p\sqrt{\pi}\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]^{-1}\left[\mu_{\theta_s}(x_i) + \sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{1}{2}}\right]\right]$$

$$-\left[\rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\varphi_{\theta_{s_j}}(x_i)\right.$$

$$+\frac{5}{96}\sqrt{\pi}\beta d_p\varphi_{\theta_{s_k}}(x_i)\left[1\right.$$

$$-\left(\mu_{\varepsilon_g}(x_i)\right.$$

$$\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\Bigg]^{-1}\left[\mu_{\theta_s}(x_i)\right.$$

$$\left.\left.+\sum_{k=1}^{NPT}a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{-0.5}\right]$$

$$\left[\frac{5}{96}\sqrt{\pi}d_p g_{0m}\left[1\right.\right.$$

$$-\left(\mu_{\varepsilon_g}(x_i)\right.$$

$$\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\Bigg]\left[\mu_{\theta_s}(x_i)\right.$$

$$\left.\left.+\sum_{k=1}^{NPT}a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{3}{2}}{\rho_s}^2\right]\right)$$

$$\left[\rho_s\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\theta_s}(x_i)\right.\right.$$

$$\left.\left.+\sum_{k=1}^{NPT}a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]\right.$$

$$+\frac{5}{48}\beta d_p\sqrt{\pi}\left[1\right.$$
$$-\left(\mu_{\varepsilon_g}(x_i)\right.$$
$$\left.+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\Bigg]^{-1}\left[\mu_{\theta_s}(x_i)\right.$$
$$\left.+\sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{1}{2}}\Bigg]^{-1}$$
$$+\frac{3.6}{5}\eta\frac{8}{3\sqrt{\pi}}\rho_s d_p g_{0m}$$
$$\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]^2$$
$$\left[\mu_{\theta_s}(x_i)+\sum_{k=1}^{NPT} a_{\theta_{s_k}}(t)\varphi_{\theta_{s_k}}(x_i)\right]^{\frac{1}{2}}.$$

The partial derivatives of β_{gs} with respect to the unknown POD coefficients are:

$$
\begin{aligned}
&\frac{\partial}{\partial a_{U_{g_j}}}\left(\beta_{gs}\right) \\
&= \left[\left(-\frac{9}{4} d_p \rho_g {V_{rm_{POD}}}^{-3}\left[\mu_{\varepsilon_g}(x_i)\right.\right.\right. \\
&\left.\left.\left. + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\right)\left[1\right.\right. \\
&\left.\left. - \left(\mu_{\varepsilon_g}(x_i)\right.\right.\right. \\
&\left.\left.\left. + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial a_{U_{g_j}}}\left(V_{rm_{POD}}\right)\right]\right) \\
&\left(0.63 + 4.8\sqrt{\left(\frac{V_{rm_{POD}}}{R_{\#_{POD}}}\right)}\right)^2 \\
&\left(\left[\left(\left(\left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.\right.\right.\right. \\
&\left.\left.\left.\left. - \left[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2\right.\right. \\
&\left.\left. - \left(\left[\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.\right.\right. \\
&\left.\left.\left. - \left[\mu_{V_s}(x_i)\right.\right.\right. \\
&\left.\left.\left. + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\right)^{\frac{1}{2}}\Bigg]
\end{aligned}
\qquad \text{(A.1.55)}
$$

$$+\left[\left(2\left[0.63\right.\right.\right.$$

$$\left.\left.+\,4.8{V_{rm}}_{POD}^{\frac{1}{2}}{R_{\#}}_{POD}^{-\frac{1}{2}}\right]\left[\left(\frac{1}{2}{V_{rm}}_{POD}^{-\frac{1}{2}}\left[\frac{\partial}{\partial a_{U_{g_j}}}\left(V_{rm_{POD}}\right)\right]\right)\right.\right.$$

$$\left.\left.\left.-\left(\frac{1}{2}{R_{\#}}_{POD}^{-\frac{3}{2}}\left[\frac{\partial}{\partial a_{U_{g_j}}}\left(R_{\#_{POD}}\right)\right]\right)\right]\right)$$

$$\left(\left[\left(\left[\mu_{U\,g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.\right.\right.$$

$$\left.\left.-\left[\mu_{U_S}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2\right.$$

$$\left.-\left(\left[\mu_{V\,g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.\right.$$

$$\left.\left.\left.-\left[\mu_{V_S}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\right)^{\frac{1}{2}}$$

$$\left(\frac{3}{4}\rho_g\left(d_p\right)^{-1}{V_{rm}}_{POD}^{-2}\left[1\right.\right.$$

$$\left.-\left(\mu_{\varepsilon\,g}(x_i)\right.\right.$$

$$\left.\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\varepsilon\,g}(x_i)\right.$$

$$\left.\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\right)\Bigg]$$

$$+\left[\left(\left(\left[\left(\left[\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.\right.\right.\right.\right.$$

$$\left.\left.-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2\right.$$

$$-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.$$

$$\left.-\left[\mu_{V_s}(x_i)\right.\right.$$

$$\left.\left.+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\Bigg]\Bigg)^{-\frac{1}{2}}$$

$$\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.$$

$$\left.-\left[\mu_{U_s}(x_i)\right.\right.$$

$$\left.\left.+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)\varphi_{U_{g_j}}(x_i)\Bigg)$$

$$\left(0.63+4.8\sqrt{\left(\frac{V_{rm_{POD}}}{R_{\#_{POD}}}\right)}\right)^2$$

$$\left(\frac{3}{4}\rho_g(d_p)^{-1}{V_{rm_{POD}}}^{-2}\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\right)\right]$$

$$\frac{\partial}{\partial a_{V_{g_j}}}(\beta_{gs})$$

$$= \left[\left(-\frac{9}{4}d_p\rho_g{V_{rm_{POD}}}^{-3}\left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial a_{V_{g_j}}}(V_{rm_{POD}})\right]\right)\right.$$

$$\left(0.63 + 4.8\sqrt{\left(\frac{V_{rm_{POD}}}{R_{\#_{POD}}}\right)}\right)^2$$

$$\left(\left[\left(\left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right] - \left[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2\right.\right.$$

(A.1.56)

$$-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.$$

$$-\left[\mu_{V_s}(x_i)\right.$$

$$\left.\left.\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\Bigg)^{\frac{1}{2}}\Bigg]$$

$$+\left[\left(2\left[0.63\right.\right.\right.$$

$$\left.\left.\left.+4.8{V_{rm}}_{POD}^{\frac{1}{2}}{R_{\#}}_{POD}^{-\frac{1}{2}}\right]\left[\left(\frac{1}{2}{V_{rm}}_{POD}^{-\frac{1}{2}}\left[\frac{\partial}{\partial a_{V_{g_j}}}\left({V_{rm}}_{POD}\right)\right]\right)\right.\right.$$

$$\left.\left.-\left(\frac{1}{2}{R_{\#}}_{POD}^{-\frac{3}{2}}\left[\frac{\partial}{\partial a_{V_{g_j}}}\left({R_{\#}}_{POD}\right)\right]\right)\right]\right)$$

$$\left(\left[\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.\right.\right.$$

$$\left.\left.\left.-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2\right.\right.$$

$$-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.$$

$$\left.\left.\left.-\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\right)^{\frac{1}{2}}$$

$$\left(\frac{3}{4}\rho_g(d_p)^{-1}{V_{rm}}_{POD}^{-2}\left[1 - \left(\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\right)\right] + \left[\left(\left(\left[\left(\left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right] - \left[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2 - \left(\left[\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right] - \left[\mu_{V_s}(x_i) + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\right)^{-\frac{1}{2}}\left(\left[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right] - \left[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)\varphi_{V_{g_j}}(x_i)\right)$$

$$\left(0.63+4.8\sqrt{\left(\frac{V_{rm_{POD}}}{R_{\#_{POD}}}\right)}\right)^2$$

$$+\left[\left(\frac{3}{4}\rho_g(d_p)^{-1}{V_{rm_{POD}}}^{-2}\left[1\right.\right.\right.$$

$$\left.-\left(\mu_{\varepsilon_g}(x_i)\right.\right.$$

$$\left.\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\varepsilon_g}(x_i)\right.$$

$$\left.\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\right)\Bigg]$$

$$\frac{\partial}{\partial a_{U_{s_j}}}\left(\beta_{gs}\right)=\left[\left(-\frac{9}{4}d_p\rho_g{V_{rm_{POD}}}^{-3}\left[\mu_{\varepsilon_g}(x_i)\right.\right.\right.$$

$$\left.\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\right.$$

$$\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]$$

$$\left.\left[\frac{\partial}{\partial a_{U_{s_j}}}\left(V_{rm_{POD}}\right)\right]\right)$$

(A.1.57)

$$\left(0.63+4.8\sqrt{\left(\frac{V_{rm_{POD}}}{R_{\#_{POD}}}\right)}\right)^2$$

$$\left(\left[\left(\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.\right.\right.\right.$$

$$\left.\left.\left.-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2\right.$$

$$-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.$$

$$-\left[\mu_{V_s}(x_i)\right.$$

$$\left.\left.\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\Bigg)^{\frac{1}{2}}\Bigg]$$

$$+\left[\left(2\left[0.63\right.\right.\right.$$

$$\left.\left.\left.+4.8V_{rm_{POD}}{}^{\frac{1}{2}}R_{\#_{POD}}{}^{-\frac{1}{2}}\right]\left[\left(\frac{1}{2}V_{rm_{POD}}{}^{-\frac{1}{2}}\left[\frac{\partial}{\partial a_{U_{s_j}}}\left(V_{rm_{POD}}\right)\right]\right)\right.\right.\right.$$

$$\left.\left.\left.-\left(\frac{1}{2}R_{\#_{POD}}{}^{-\frac{3}{2}}\left[\frac{\partial}{\partial a_{U_{s_j}}}\left(R_{\#_{POD}}\right)\right]\right)\right]\right)\right]$$

$$\left(\left[\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.\right.\right.$$

$$-\left.\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2$$

$$-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.$$

$$\left.\left.\left.-\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\right)^{\frac{1}{2}}$$

$$\left(\frac{3}{4}\rho_g(d_p)^{-1}{V_{rm}}_{POD}^{-2}\left[1\right.\right.$$
$$-\left(\mu_{\varepsilon_g}(x_i)\right.$$
$$\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\Big]\Big[\mu_{\varepsilon_g}(x_i)$$
$$\left.\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\right)\Big]$$
$$+\left[\left(\left(\left[\left(\left[\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.\right.\right.\right.\right.$$
$$\left.\left.-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2\right.$$
$$-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.$$
$$-\left[\mu_{V_s}(x_i)\right.$$
$$\left.\left.\left.\left.+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\right)^{-\frac{1}{2}}$$
$$\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.$$
$$-\left[\mu_{U_s}(x_i)\right.$$
$$\left.\left.+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)[-\varphi_{U_{s_j}}(x_i)]\Bigg)$$

$$\left(0.63+4.8\sqrt{\left(\frac{V_{rm_{POD}}}{R_{\#_{POD}}}\right)}\right)^{2}$$

$$\left(\frac{3}{4}\rho_g(d_p)^{-1}{V_{rm_{POD}}}^{-2}\right.$$

$$\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]$$

$$\left.\left[\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\right)\Bigg]$$

$$\frac{\partial}{\partial a_{V_{s_j}}}\left(\beta_{gs}\right)=\left[\left(-\frac{9}{4}d_p\rho_g{V_{rm_{POD}}}^{-3}\left[\mu_{\varepsilon_g}(x_i)\right.\right.\right.$$

$$\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]$$

$$\left.\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\frac{\partial}{\partial a_{V_{s_j}}}\left(V_{rm_{POD}}\right)\right]\right)$$

$$\left(0.63+4.8\sqrt{\left(\frac{V_{rm_{POD}}}{R_{\#_{POD}}}\right)}\right)^{2}$$

$$\left(\left[\left(\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.\right.\right.\right.$$

$$\left.\left.\left.\left.-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^{2}\right.\right.\right.$$

$$\left.\left.-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.\right.\right.$$

$$\left.\left.\left.\left.-\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^{2}\right]\right)^{\frac{1}{2}}\right] \qquad \text{(A.1.57)}$$

$$+\left[\left(2\left[0.63+4.8{V_{rm_{POD}}}^{\frac{1}{2}}{R_{\#_{POD}}}^{-\frac{1}{2}}\right]\left[\left(\frac{1}{2}{V_{rm_{POD}}}^{-\frac{1}{2}}\left[\frac{\partial}{\partial a_{V_{s_j}}}\left(V_{rm_{POD}}\right)\right]\right)-\left(\frac{1}{2}{R_{\#_{POD}}}^{-\frac{3}{2}}\left[\frac{\partial}{\partial a_{V_{s_j}}}\left(R_{\#_{POD}}\right)\right]\right)\right]\right)\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]-\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\right)^{\frac{1}{2}}$$

$$\left(\frac{3}{4}\rho_g\left(d_p\right)^{-1}{V_{rm_{POD}}}^{-2}\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\right)\Bigg]$$

$$+\left[\left(\left(\left[\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2\right.\right.\right.\right.$$

$$-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.$$

$$\left.-\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\Bigg]\Bigg)^{-\frac{1}{2}}$$

$$\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.$$

$$\left.-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)$$

$$\left[-\varphi_{V_{s_j}}(x_i)\right]\Big)$$

$$\left(0.63+4.8\sqrt{\left(\frac{V_{rm_{POD}}}{R_{\#_{POD}}}\right)}\right)^2$$

$$\left(\frac{3}{4}\rho_g\left(d_p\right)^{-1}{V_{rm_{POD}}}^{-2}\right.$$

$$\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]$$

$$\left.\left[\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\right)\Bigg]$$

$$\frac{\partial}{\partial a_{\varepsilon_{g_j}}}\left(\beta_{gs}\right)=\left(\frac{3}{4}\rho_g\left(d_p\right)^{-1}\right. \qquad \text{(A.1.58)}$$

$$\left(\left[\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.\right.\right.$$

$$\left.\left.\left.-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2\right.$$

$$-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.$$

$$-\left[\mu_{V_s}(x_i)\right.$$

$$\left.\left.\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\Bigg)^{\frac{1}{2}}\Bigg)$$

$$\left[\left[1-2\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\right.$$

$$\left.\varphi_{\varepsilon_{g_j}}(x_i)\, {V_{rm_{POD}}}^{-2}\left(0.63+4.8\sqrt{\left(\frac{V_{rm_{POD}}}{R_{\#_{POD}}}\right)}\right)^2\right]$$

$$+\Bigg(\frac{3}{4}\rho_g(d_p)^{-1}\Bigg(\Bigg[\Bigg(\Bigg[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\Bigg]$$

$$-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\Bigg)^2$$

$$-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right.$$

$$-\left[\mu_{V_s}(x_i)\right.$$

$$\left.\left.\left.+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\Bigg)^{\frac{1}{2}}\Bigg)$$

$$\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]\left(-2{V_{rm}}_{POD}^{-3}\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(V_{rm})\right]\right)\left(0.63+4.8\sqrt{\left(\frac{{V_{rm}}_{POD}}{R_{\#_{POD}}}\right)}\right)^2+\left(\frac{3}{4}\rho_g(d_p)^{-1}\left(\left[\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU}a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU}a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV}a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]-\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV}a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]\right)^{\frac{1}{2}}\right)$$

$$\left[\left[1-\left(\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right)\right]\left[\mu_{\varepsilon_g}(x_i)\right.\right.$$

$$\left.+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right] {V_{rm_{POD}}}^{-2}\left(4.8\left(0.63\right.\right.$$

$$\left.\left.+4.8\sqrt{\left(\frac{V_{rm_{POD}}}{R_{\#_{POD}}}\right)}\right)\left[V_{rm_{POD}}R_{\#_{POD}}\right]^{-\frac{1}{2}}\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}\left(V_{rm}\right)\right]\right)\Bigg]$$

$$\frac{\partial}{\partial a_{P_{g_j}}}\left(\beta_{gs}\right)=0 \qquad \text{(A.1.59)}$$

$$\frac{\partial}{\partial a_{P_{s_j}}}\left(\beta_{gs}\right)=0 \qquad \text{(A.1.60)}$$

$$\frac{\partial}{\partial a_{\theta_{s_j}}}\left(\beta_{gs}\right)=0 \qquad \text{(A.1.61)}$$

where the POD approximation for $V_{rm}$and $R_{\#}$are given by$V_{rm_{POD}}$and $R_{\#_{POD}}$ respectively

$$R_{\#_{POD}}=d_p\rho_s{\mu_g}^{-1}\left[\left(\left[\mu_{U_g}(x_i)+\sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\right]\right.\right.$$

$$\left.-\left[\mu_{U_s}(x_i)+\sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\right]\right)^2$$

$$-\left(\left[\mu_{V_g}(x_i)+\sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\right]\right. \qquad \text{(A.1.62)}$$

$$\left.\left.-\left[\mu_{V_s}(x_i)+\sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\right]\right)^2\right]^{\frac{1}{2}}$$

$$V_{rm_{POD}} = \frac{1}{2}\Bigg(\Bigg[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\Bigg]^{4.14} - 0.06R_{\#_{POD}} + \Bigg[\left(0.06R_{\#_{POD}}\right)^2 + 0.12R_{\#_{POD}}(2B_{POD} - \Bigg[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\Bigg]^{4.14}) + \Bigg[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\Bigg]^{17.396}\Bigg]^{\frac{1}{2}}\Bigg) \quad \text{(A.1.63)}$$

and B_{POD} is a piecewise function defined by at the previous timestep obtained by

$$B_{POD} = \begin{cases} 0.8\Bigg[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\Bigg]^{1.28}; \\ if\ \Bigg[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\Bigg]_{@(t-\Delta t)} \leq 0.85 \\ \Bigg[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\Bigg]^{2.65}; \\ if\ \Bigg[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\Bigg]_{@(t-\Delta t)} > 0.85 \end{cases} \quad \text{(A.1.64)}$$

The partial derivatives of each with respect to the relevant POD coefficients are listed below.

$$\frac{\partial}{\partial a_{U_{g_j}}}(V_{rm}) = 0.03\left[\frac{\partial}{\partial a_{U_{g_j}}}(R_{\#})\right]$$
$$+\frac{1}{4}\Bigg[(0.06R_{\#})^2 + 0.24B_{POD}R_{\#}$$
$$-0.12\left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]^{4.14} R_{\#} \qquad \text{(A.1.65)}$$
$$+\left(\left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]^{4.14}\right)^2\Bigg]^{-\frac{1}{2}}$$

$$\Bigg[0.12R_{\#}\left[\frac{\partial}{\partial a_{U_{g_j}}}(R_{\#})\right] + 0.24B_{POD}\left[\frac{\partial}{\partial a_{U_{g_j}}}(R_{\#})\right]$$
$$-0.12\Bigg[\mu_{\varepsilon_g}(x_i)$$
$$+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg]^{4.14}\left[\frac{\partial}{\partial a_{U_{g_j}}}(R_{\#})\right]\Bigg]$$

$$\frac{\partial}{\partial a_{U_{g_j}}}(R_{\#}) = \frac{\rho_s d_p}{\mu_g}\Bigg(\Bigg(\Bigg[\Bigg(\Bigg[\Bigg[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\Bigg]$$

$$- \Bigg[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\Bigg]\Bigg)^2$$

$$- \Bigg(\Bigg[\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\Bigg]$$

$$- \Bigg[\mu_{V_s}(x_i)$$

$$+ \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\Bigg]\Bigg)^2\Bigg]\Bigg)^{-\frac{1}{2}}\Bigg(\Bigg[\mu_{U_g}(x_i)$$

$$+ \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\Bigg]$$

$$- \Bigg[\mu_{U_s}(x_i)$$

$$+ \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\Bigg]\Bigg)\varphi_{U_{g_j}}(x_i)\Bigg) \qquad \text{(A.1.66)}$$

$$
\begin{aligned}
&\frac{\partial}{\partial a_{V_{g_j}}}(V_{rm}) \\
&= 0.03\left[\frac{\partial}{\partial a_{V_{g_j}}}(R_{\#})\right] \\
&+\frac{1}{4}\Bigg[(0.06R_{\#})^2 + 0.24B_{POD}R_{\#} \\
&-0.12\left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]^{4.14} R_{\#} \\
&+\Bigg(\Bigg[\mu_{\varepsilon_g}(x_i) \\
&+\sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg]^{4.14}\Bigg)^2\Bigg]^{-\frac{1}{2}}\Bigg[0.12R_{\#}\left[\frac{\partial}{\partial a_{V_{g_j}}}(R_{\#})\right] \\
&+0.24B_{POD}\left[\frac{\partial}{\partial a_{V_{g_j}}}(R_{\#})\right] \\
&-0.12\left[\mu_{\varepsilon_g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]^{4.14}\left[\frac{\partial}{\partial a_{V_{g_j}}}(R_{\#})\right]\Bigg]
\end{aligned}
\qquad \text{(A.1.67)}
$$

$$\frac{\partial}{\partial a_{V_{g_j}}}(R_{\#}) = \frac{\rho_s d_p}{\mu_g}\Bigg(\Bigg(\Bigg[\Bigg(\Bigg[\mu_{U\,g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\Bigg] - \Bigg[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\Bigg]\Bigg)^2 - \Bigg(\Bigg[\mu_{V\,g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\Bigg] - \Bigg[\mu_{V_s}(x_i) + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\Bigg]\Bigg)^2\Bigg]\Bigg)^{-\frac{1}{2}}\Bigg(\Bigg[\mu_{U\,g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\Bigg] - \Bigg[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\Bigg]\Bigg)\varphi_{V_{g_j}}(x_i)\Bigg) \quad \text{(A.1.68)}$$

$$\frac{\partial}{\partial a_{U_{s_j}}}(V_{rm}) = 0.03\left[\frac{\partial}{\partial a_{U_{s_j}}}(R_{\#})\right]$$
$$+\frac{1}{4}\Bigg[(0.06R_{\#})^2 + 0.24B_{POD}R_{\#}$$
$$-0.12\left[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right]^{4.14} R_{\#} \qquad \text{(A.1.69)}$$
$$+\left(\left[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right]^{4.14}\right)^2\Bigg]^{-\frac{1}{2}}$$
$$\Bigg[0.12R_{\#}\left[\frac{\partial}{\partial a_{U_{s_j}}}(R_{\#})\right] + 0.24B_{POD}\left[\frac{\partial}{\partial a_{U_{s_j}}}(R_{\#})\right]$$
$$-0.12\Bigg[\mu_{\varepsilon g}(x_i)$$
$$+\sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\Bigg]^{4.14}\left[\frac{\partial}{\partial a_{U_{s_j}}}(R_{\#})\right]\Bigg]$$

$$\frac{\partial}{\partial a_{U_{s_j}}}(R_{\#}) = \frac{\rho_s d_p}{\mu_g}\Bigg(\Bigg(\Bigg[\Bigg(\Bigg[\mu_{U_g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\Bigg]$$
$$- \Bigg[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\Bigg]\Bigg)^2$$
$$- \Bigg(\Bigg[\mu_{V_g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\Bigg]$$
$$- \Bigg[\mu_{V_s}(x_i)$$
$$+ \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\Bigg]\Bigg)^2\Bigg]\Bigg)^{-\frac{1}{2}}\Bigg(\Bigg[\mu_{U_g}(x_i)$$
$$+ \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\Bigg]$$
$$- \Bigg[\mu_{U_s}(x_i)$$
$$+ \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\Bigg]\Bigg)[-\varphi_{U_{s_j}}(x_i)]\Bigg) \qquad \text{(A.1.70)}$$

$$\frac{\partial}{\partial a_{V_{s_j}}}(V_{rm}) = 0.03\left[\frac{\partial}{\partial a_{V_{s_j}}}(R_{\#})\right] + \frac{1}{4}\Bigg[(0.06R_{\#})^2 + 0.24B_{POD}R_{\#} - 0.12\left[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right]^{4.14} R_{\#} + \left(\left[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right]^{4.14}\right)^2\Bigg]^{-\frac{1}{2}} \left[0.12R_{\#}\left[\frac{\partial}{\partial a_{V_{s_j}}}(R_{\#})\right] + 0.24B_{POD}\left[\frac{\partial}{\partial a_{V_{s_j}}}(R_{\#})\right] - 0.12\Bigg[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\Bigg]^{4.14}\left[\frac{\partial}{\partial a_{V_{s_j}}}(R_{\#})\right]\right] \quad \text{(A.1.71)}$$

$$\frac{\partial}{\partial a_{V_{s_j}}}(R_{\#})$$

$$= \frac{\rho_s d_p}{\mu_g}\Bigg(\Bigg(\Bigg[\Bigg(\Bigg[\mu_{U\,g}(x_i) + \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\Bigg]$$

$$- \Bigg[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\Bigg]\Bigg)^2$$

$$- \Bigg(\Bigg[\mu_{V\,g}(x_i) + \sum_{k=1}^{NGV} a_{V_{g_k}}(t)\varphi_{V_{g_k}}(x_i)\Bigg] \qquad \text{(A.1.72)}$$

$$- \Bigg[\mu_{V_s}(x_i) + \sum_{k=1}^{NPV} a_{V_{s_k}}(t)\varphi_{V_{s_k}}(x_i)\Bigg]\Bigg)^2\Bigg]\Bigg)^{-\frac{1}{2}}\Bigg(\Bigg[\mu_{U\,g}(x_i)$$

$$+ \sum_{k=1}^{NGU} a_{U_{g_k}}(t)\varphi_{U_{g_k}}(x_i)\Bigg]$$

$$- \Bigg[\mu_{U_s}(x_i) + \sum_{k=1}^{NPU} a_{U_{s_k}}(t)\varphi_{U_{s_k}}(x_i)\Bigg]\Bigg)[-\varphi_{V_{s_j}}(x_i)]\Bigg)$$

$$\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(V_{rm}) = \frac{1}{2}\Bigg(4.14\varphi_{\varepsilon_{g_j}}(x_i)\Bigg[\mu_{\varepsilon\,g}(x_i) \qquad \text{(A.1.73)}$$

$$+ \sum_{k=1}^{NGF} a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg]^{3.14}\Bigg)$$

$$+\frac{1}{2}\Bigg[\left(0.06R_{\#_{POD}}\right)^2+0.24B_{POD}R_{\#_{POD}}$$

$$-0.12R_{\#_{POD}}\Bigg[\mu_{\varepsilon_g}(x_i)$$

$$+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg]^{4.14}$$

$$+\left(\left[\mu_{\varepsilon_g}(x_i)+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]^{4.14}\right)^2\Bigg]^{-\frac{1}{2}}$$

$$\left[0.24R_{\#_{POD}}\left[\frac{\partial}{\partial a_{\varepsilon_{g_j}}}(B_{POD})\right]\right.$$

$$-0.12R_{\#_{POD}}\left(4.14\varphi_{\varepsilon_{g_j}}(x_i)\Bigg[\mu_{\varepsilon_g}(x_i)\right.$$

$$\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]^{3.14}\Bigg)$$

$$+2\Bigg[\mu_{\varepsilon_g}(x_i)$$

$$+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\Bigg]^{4.14}\left(4.14\varphi_{\varepsilon_{g_j}}(x_i)\Bigg[\mu_{\varepsilon_g}(x_i)\right.$$

$$\left.+\sum_{k=1}^{NGF}a_{\varepsilon_{g_k}}(t)\varphi_{\varepsilon_{g_k}}(x_i)\right]^{3.14}\Bigg)\Bigg]$$

$$\frac{\partial}{\partial a_{\varepsilon g_j}}(B_{POD})$$

$$= \begin{cases} 1.024\varphi_{\varepsilon g_j}(x_i)\left[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right]^{0.28}; \\ \left[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right]_{@(t-\Delta t)} \leq 0.85 \\ 2.65\varphi_{\varepsilon g_j}(x_i)\left[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right]^{1.65}; \\ \left[\mu_{\varepsilon g}(x_i) + \sum_{k=1}^{NGF} a_{\varepsilon g_k}(t)\varphi_{\varepsilon g_k}(x_i)\right]_{@(t-\Delta t)} > 0.85 \end{cases} \quad \text{(A.1.74)}$$

APPENDIX 2.0 MATRIX ENERGY PROOF

Let $\lambda_1, \lambda_2, \dots, \lambda_n$ denote the eigenvalues of the matrix $K = \frac{1}{n} A^T A$. Let $X_1, X_2, \dots, X_n$ denote the eigenvectors of the matrix K. Because the *n* x *n* matrix K is symmetric, by definition all eigenvalues of K are real. Let ε denote the relative error tolerance of matrix energy for the matrix energy between the full order model of the data set given in K. Let the reduced order representation of A be given by a matrix, R, for any given m < n be obtained through formulation of the largest m eigenvalues given by

$$\varepsilon \geq \frac{|M(K) - M(R)|}{M(K)}. \qquad (A.2.1)$$

This implies that

$$\varepsilon \sum_{k=1}^{n} |\lambda_k| \geq |[\sum_{k=1}^{n} |\lambda_k| - \sum_{k=1}^{m} |\lambda_k|]|. \qquad (A.2.2)$$

Since m < n, the difference on the right side of the equation is positive and thus the outer absolute value operator is unnecessary. Furthermore, dividing individual terms by the matrix energy of K yields

$$\varepsilon \geq 1.0 - \frac{\sum_{k=1}^{m} |\lambda_k|}{\sum_{k=1}^{n} |\lambda_k|}. \qquad (A.2.3)$$

And as a result,

$$\frac{\sum_{k=1}^{m} |\lambda_k|}{\sum_{k=1}^{n} |\lambda_k|} \geq 1.0 - \varepsilon. \qquad (A.2.4)$$

Thus, it suffices to show that the proportion of the sum of the first m largest singular values of A to the total sum of singular values of A is equal to the above eigenvalue proportion. This is done through the realization that Equation (4.2.4) holds. Namely that $\sigma_i^2 = n\lambda_i$ where σ_i^2 denotes the *i*th largest singular value of A and λ_i denotes the *i*th largest eigenvalue of K.

Again, define the singular value decomposition (SVD) of A ($A = U\Sigma \mathrm{V}^{\mathrm{T}}$) as in Equation 4.2.4 where U and V are orthogonal matrices, and Σ is a rectangular pseudo-diagonal matrix. Let σ_i denote the singular values of A where *i* denotes the *i*th largest singular value of A.

From the SVD representation of A

$$K = \frac{1}{n} A^T A = \frac{1}{n} (U\Sigma \mathrm{V}^{\mathrm{T}})^T (U\Sigma \mathrm{V}^{\mathrm{T}}) = \frac{1}{n} (V\Sigma^{\mathrm{T}} \mathrm{U}^{\mathrm{T}})(U\Sigma \mathrm{V}^{\mathrm{T}}). \qquad (\text{A.2.5})$$

Since U is an orthogonal matrix, $\mathrm{U}^{\mathrm{T}}\mathrm{U}$ yields an identity matrix. Because $\frac{1}{n}$ is consistent, $\frac{1}{n}\mathrm{V}=\mathrm{V}\frac{1}{n}$. This allow for the expression of K as

$$K = V \frac{1}{n} \Sigma^{\mathrm{T}} \Sigma \mathrm{V}^{\mathrm{T}}. \qquad (\text{A.2.6})$$

Since V is an orthogonal matrix, this denotes an eigenvalue-eigenvector decomposition of the matrix K where the eigenvalues of K are given by $\frac{1}{n}\sigma_i^{\ 2}$. Making this substitution for each eigenvalue into the left side of the inequality given in (A.2.4), factoring out $\frac{1}{n}$ from the numerator and denominator and subsequently canceling this term yields the desired outcome.

Thus, this ultimately results in the inequality, $\frac{\sum_{\mathrm{k}=1}^{\mathrm{m}} \sigma_{\mathrm{k}}^{\ 2}}{\sum_{\mathrm{k}=1}^{\mathrm{n}} \sigma_{\mathrm{k}}^{\ 2}} = \frac{\sum_{\mathrm{k}=1}^{\mathrm{m}} |\lambda_{\mathrm{k}}|}{\sum_{\mathrm{k}=1}^{\mathrm{n}} |\lambda_{\mathrm{k}}|} \geq 1.0 - \varepsilon$, given by Equation (4.2.6).

APPENDIX 3.0 <u>SOURCE CODE</u>

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//   main.cpp
//   Created by Stephanie R. Beck Roth
//   Last Modified:   07/07/2011
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#include "mpi.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "MfixData.h"
#include "AspenWriter.h"
#include "MFIXReader.h"
#include "PhaseData.h"
#include "PODSolver.h"
#include "NSSolver.h"
#include "VTKWriter.h"

// Assuming units were input as 'cgs'

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

using namespace std;

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int main(int argc, char **argv)
   {

   bool controller(int argc);
   bool Validate(MfixData & D, SimVars & SV, float & f, int i);
   bool GetPOD(SimVars & Data, PODSolver & PODer, int Id, int NIds,
            std::vector< double > & en);
   bool LoadPOD(SimVars & Data, PODSolver & PODer, int Id, int NIds,
             string ln);
   bool ProcessSDataFile(string & sp_name, string & mfix_name,
                    string & dataname,
                    bool & loaddata, bool & project,
```

```
                bool & extrap, double & etime,
                string & exvar, double & exval,
                std::vector<double> & energy, double & exval2,
                double & Lx, double & Rx, string & mfix_name2,
                string & dataname2, bool & interp,
                bool & loaddata2, int & mits, double & toler,
                double & epsilon1, double & epsilon2, int & OF,
                float & starttime, float & endtime);

string mfix_name;
string mfix_name2;
string sp_name;
string strE ("E");
string strVg ("Vg");
int OFormat;  // Flag for output format 0=standard, 1=VTK too

float t, t2; // time step
int Ntimes; // Keeps track of the number of time points stored.
bool loaddata = false; // flag for loading of POD components
bool loaddata2 = false; // flag for loading of POD components
                  // for interpolation purposes
bool project = false;  // project future time steps
bool extrap = false;   // extrapolate
bool interp = false;   // interpolate
bool success = false;  // validity of data input file
std::vector<double> podenergy;
podenergy.clear();
float startpod;
float endpod;

double etime;  // final time step - valid only if project is true
double exval;  // extrapolation variable value - valid only if
          // extrap is true
double exval2; // if extrapolating jet exval contains background
           // flow Vg and exval2 contains central jet y-velocity
double Lx, Rx; // contain the left and right x boundary endpoints of
           // the velocity
string exvar; // variable to extrapolate currently only valid for
          // Ug and E

double tol; // LM tolerance
double eps1; // LM ep1
double eps2; // LM ep2
int maxits;  // LM max iteration #
```

```
std::vector< std::vector< double > > tmpV; //tmp vector for .vtk output

vector< double > bbc;
int i;

string dataname; // data directory for projection, extrap POD data
                 // and first interp dataset
string dataname2; // data directory for interp second POD data

if (!controller(argc))
   {
   cout << "Error in processing your request!  Please resubmit."
        << endl;
   exit(-1);
   }

sp_name =  argv[1];
std::cout << argc << " arguments. \n";
std::cout << sp_name << " is the spoutpod data input file.  \n";

if (argc != 2)
   {
   std::cout << "Error processing your request! \n "
        << "Proper syntax: spoutpod <file directory>spoutpod.dat \n";
   }

success = ProcessSDataFile( sp_name, mfix_name, dataname, loaddata,
                        project, extrap, etime, exvar, exval,
                        podenergy, exval2, Lx, Rx, mfix_name2,
                        dataname2, interp, loaddata2,
                        maxits, tol, eps1, eps2, OFormat,
                        startpod, endpod);

std::cout << "loading " << mfix_name;
std::cout << " file. \n";
std::cout << "Calculating PODs with a minimum of "
        << podenergy.at(0) << " matrix energy for Ug. \n";
std::cout << "Calculating PODs with a minimum of "
        << podenergy.at(1) << " matrix energy for Vg. \n";
std::cout << "Calculating PODs with a minimum of "
        << podenergy.at(2) << " matrix energy for Us. \n";
std::cout << "Calculating PODs with a minimum of "
        << podenergy.at(3) << " matrix energy for Vs. \n";
std::cout << "Calculating PODs with a minimum of "
        << podenergy.at(4) << " matrix energy for Eg. \n";
```

```
    std::cout << "Calculating PODs with a minimum of "
              << podenergy.at(5) << " matrix energy for Pg. \n";
    std::cout << "Calculating PODs with a minimum of "
              << podenergy.at(6) << " matrix energy for Ps. \n";
    std::cout << "Calculating PODs with a minimum of "
              << podenergy.at(7) << " matrix energy for Ts. \n";

  if (!(success))
     {
     std::cout <<  "Error in processing spoutpod input file.  \n";
     std::cout <<  "spoutpod is halting... \n";
     }
  else {

    // For MPI capability {
    int numpids; // Number of Communication Tasks
    int my_id;     // Rank number
    int rc;       // Initialization reason code
    rc = MPI_Init(&argc, &argv);
    if (rc != MPI_SUCCESS)
       {
       std::cout << "Error starting MPI program! \n"
                 << "Terminating. \n";
       MPI_Abort(MPI_COMM_WORLD, rc);
       } // end if rc
    MPI_Comm_size(MPI_COMM_WORLD, &numpids);
    MPI_Comm_rank(MPI_COMM_WORLD, &my_id);
//    my_id = MPI_COMM_WORLD::Get_Rank();
//    numpids = MPI_COMM_WORLD::Get_Size();
    if (my_id == 0)
       std::cout << "Number of processors: " << numpids << "\n";
    std::cout<< "Processor id " << my_id << " reporting for duty! \n";
    // } MPI stuff

    MfixData SimMfixData;
    MfixReader SimReader;
    SimVars Data;
    PODSolver PODer;
    bool good;  // variable which contains the success of the POD model

    SimMfixData.SetName(mfix_name.c_str());  // Written By Phil
    SimMfixData.ReadRes0();                  // Written By Phil
    SimMfixData.CreateVariableNames();       // Written By Phil
    SimMfixData.GetTimes();                  // Written By Phil
```

```
   SimReader.GetBC(mfix_name.c_str(), Data);
//    t = SimReader.GetMinTime();
   Data.Set2DVolVars();

   Ntimes = 0;
   int counter =0;
   while ( counter < SimMfixData.times.size() )
      {
      t = SimMfixData.times[counter];
//       if (my_id == 0)
//          std::cout << "Getting time= " << t << "\n";
      if ((t >= startpod) &&
         (t <= endpod))
         {
//          if (my_id == 0)
//             std::cout << "Getting Data. at " << t << " \n";
         SimReader.GetData(SimMfixData, counter, Data,
                  t, Ntimes, startpod, endpod);
         Ntimes++;
         MPI_Barrier(MPI_COMM_WORLD);
//          if (my_id == 0)
//             std::cout << "Has Data at " << t << " \n";
         }
//       if (my_id == 0)
//          cout << Ntimes << ": At t = "
//               << Data.GetTimept(Ntimes - 1) << endl;
      counter++;
      } // end while

   Data.SetTimes(Ntimes);
   if (my_id == 0)
      {
      cout << Data.GetNt() << "=Ntimes \n";
      cout << "Back to main \n";
      }

   if (my_id == 0)
      std::cout << Ntimes << " is number of timepoints.\n";

   if (my_id == 0)
      cout << "Collected Data \n";
   if (!loaddata)
      {
      if (my_id == 0)
         std::cout << "Generating POD Data... \n";
```

```
   good = GetPOD(Data, PODer, my_id, numpids, podenergy);
   if (good && my_id == 0)
      cout << "Done obtaining POD representation.\n";
   if (!good && my_id == 0)
      cout << "Failure to obtain POD model. \n";
   }
else // Load stored POD basis
   {
   if (my_id == 0)
      std::cout << "Loading POD Data... \n";
   good = LoadPOD(Data, PODer, my_id, numpids, dataname);
   MPI_Barrier(MPI_COMM_WORLD);
   if (my_id == 0)
      {
      if (good)
         cout << "Done loading POD \n";
      if (!good)
         cout << "Failure to load POD basis. \n";
      } // end if my_id is 0
   } // end else

int itp;
int ntp = Data.GetNt();
std::string ftpest = "./TimePts.txt";
std::ofstream ofiletp (ftpest.c_str());
for (itp=0; itp < ntp; itp++)
   ofiletp << Data.GetTimept(itp) << "\n";
ofiletp.close();

// Pass in SimVars, PODData and the type of function to perform
// type = 0 is Projection
// type = 1 is Interpolation
// type = 2 is Extrapolation

PODData POD;
int loadall;
// temporarily forcing Projection
if (project)
   {
   if (my_id == 0)
      std::cout << "Loading PODData.  \n";
   PODer.GetPODData(POD);
   if (my_id == 0)
      std::cout << "Loaded PODData.  \n";
   std::vector < std::vector< double > > Ug;
```

```
        std::vector < std::vector< double > > Vg;
        std::vector < std::vector< double > > Us;
        std::vector < std::vector< double > > Vs;
        std::vector < std::vector< double > > Eg;
        std::vector < std::vector< double > > Pg;
        std::vector < std::vector< double > > Ps;
        std::vector < std::vector< double > > Ts;
        if (!loaddata)
           dataname = "./Output";
        loadall = 1;
        PODer.GetSol(loadall, dataname, loaddata,
                  Ug, Vg, Us, Vs, Eg, Pg, Ps, Ts);
        if (my_id == 0)
           std::cout << "Initialized Solution Variables.  \n";
        Data.ExtractBBC(bbc);
//        if (my_id == 0)
//           {
//           std::cout << "Extracted BBC.  \n";
//           std::cout << bbc.at(0) << " to " << bbc.back()
//                     << " at " << bbc.size() << "\n";
//           }
        NSSolver NSP;
        NSP.InitNS(Data, POD, 0, exvar, exval, bbc,
                  Ug, Vg, Us, Vs, Eg,
                  Pg, Ps, Ts, my_id, numpids,
                  maxits, tol, eps1, eps2);
        if (my_id == 0)
           std::cout << "Initialized NSSolver. \n";
        NSP.ProjectSol(etime);
        MPI_Barrier(MPI_COMM_WORLD);
        if (my_id == 0)
           std::cout << "Solution Projected. \n";

        if (OFormat == 1)
           {
           VTKWriter VTK(NSP.POD.GetNx(), NSP.POD.GetNy(),
                     NSP.POD.Getdx(), NSP.POD.Getdy());
           for (i=0; i < NSP.IndexVec.size(); i++)
              {
              tmpV = NSP.PgSum;
 std::cout << "Index size " << NSP.IndexVec.size() << "\n";
              VTK.WritePg(tmpV,NSP.IndexVec.at(i),
                        NSP.DtVec.at(i),i);
              }
           if (my_id == 0)
```

```
            VTK.WritePvdPg(Data.TimePts);
          std::cout << "VTK Files Written. \n";
          }
       else if (OFormat == 2)
          {
          if (my_id == 0)
             {
             std::cout << "Aspen Files Written. \n";
             AspenWriter Aspen(NSP);
             Aspen.Write();
             }
          }


       }

    if (extrap)
       {
       if (my_id == 0)
          std::cout << "Extrapolating... \n";
       PODer.GetPODData(POD);
       std::vector < std::vector< double > > Ug;
       std::vector < std::vector< double > > Vg;
       std::vector < std::vector< double > > Us;
       std::vector < std::vector< double > > Vs;
       std::vector < std::vector< double > > Eg;
       std::vector < std::vector< double > > Pg;
       std::vector < std::vector< double > > Ps;
       std::vector < std::vector< double > > Ts;
       loadall = 0;
       if (!loaddata)
          dataname = "./Output";
MPI_Barrier(MPI_COMM_WORLD);

       PODer.GetSol(loadall, dataname, loaddata,
                Ug, Vg, Us, Vs, Eg, Pg, Ps, Ts);
       if (exvar == strVg)
          Data.FormBBC(exval, exval2, Lx, Rx, bbc);
       else
          Data.ExtractBBC(bbc);

MPI_Barrier(MPI_COMM_WORLD);
       NSSolver NSE;
       NSE.InitNS(Data, POD, 2, exvar, exval, bbc,
                Ug, Vg, Us, Vs, Eg,
```

```
                Pg, Ps, Ts, my_id, numpids,
                maxits, tol, eps1, eps2);
        if (exvar == strVg)
           NSE.ExtrapSol(exval, true, false);
        else if (exvar == strE)
           NSE.ExtrapSol(exval, false, true);
        else
           {
           if (my_id == 0)
              std::cout << "Invalid Extrapolation variable!";
           exit(-1);
           }
// Temporarily commented for testing...
//       MPI_Barrier(MPI_COMM_WORLD);
        if (my_id == 0)
           std::cout << "Solution Extrapolated. \n";

        if (OFormat == 1)
           {
           VTKWriter VTK(NSE.POD.GetNx(), NSE.POD.GetNy(),
                    NSE.POD.Getdx(), NSE.POD.Getdy());
           for (i=0; i < NSE.IndexVec.size(); i++)
              {

              VTK.WritePg(NSE.PgSum,NSE.IndexVec.at(i),
                       NSE.DtVec.at(i),i);
              VTK.WriteEg(NSE.EgSum,NSE.IndexVec.at(i),
                       NSE.DtVec.at(i),i);
              VTK.WriteUg(NSE.UgSum,NSE.IndexVec.at(i),
                       NSE.DtVec.at(i),i);
              VTK.WriteVg(NSE.VgSum,NSE.IndexVec.at(i),
                       NSE.DtVec.at(i),i);
              VTK.WriteUs(NSE.UsSum,NSE.IndexVec.at(i),
                       NSE.DtVec.at(i),i);
              VTK.WriteVs(NSE.VsSum,NSE.IndexVec.at(i),
                       NSE.DtVec.at(i),i);
              VTK.WritePs(NSE.PsSum,NSE.IndexVec.at(i),
                       NSE.DtVec.at(i),i);
              VTK.WriteTs(NSE.TsSum,NSE.IndexVec.at(i),
                       NSE.DtVec.at(i),i);
              }
           std::cout << my_id << ": VTK Files Written. \n";
           }
        else if (OFormat == 2)
           {
```

```
        if (my_id == 0)
           {
           std::cout << my_id << ": Aspen Files Written. \n";
           AspenWriter Aspen(NSE);
           Aspen.Write();
           }
        }

     } //end extrap

  if (interp)
     {

     if (my_id == 0)
        std::cout << "Interpolating... \n";

     // The following is the second extrapolation part
     MfixData SimMfixData2;
     MfixReader SimReader2;
     SimVars Data2;
     PODSolver PODer2;
     bool good2;  // variable which contains the success
                  // of the POD model

     SimMfixData2.SetName(mfix_name2.c_str());  // Written By Phil
     SimMfixData2.ReadRes0();                   // Written By Phil
     SimMfixData2.CreateVariableNames();        // Written By Phil
     SimMfixData2.GetTimes();                   // Written By Phil

     SimReader2.GetBC(mfix_name2.c_str(), Data2);
     t2 = SimReader2.GetMinTime();
     Data2.Set2DVolVars();




     Ntimes = 0;
     counter = 0;
     while ( counter < SimMfixData2.times.size() )
        {
        t = SimMfixData2.times[counter];
//        std::cout << "Getting Data. at " << t << " \n";
        SimReader2.GetData(SimMfixData2, counter, Data2,
                           t, Ntimes, startpod, endpod);
        if (my_id == 0)
```

```
        cout << Ntimes << ": At t = "
             << Data.GetTimept(Ntimes) << endl;
    counter++;
    } // end while

Data2.SetTimes(Ntimes);
if (my_id == 0)
    cout << "Collected Second Set of Data \n";

PODData POD2;
if (!loaddata2)
    {
    if (my_id == 0)
        std::cout << "Generating Second Set of POD Data... \n";
    good = GetPOD(Data2, PODer2, my_id, numpids, podenergy);
    if (good && my_id == 0)
        cout << "Done obtaining Second POD representation.\n";
    if (!good && my_id == 0)
        cout << "Failure to obtain Second POD model. \n";
    }
else // Load stored second POD basis
    {
    if (my_id == 0)
        std::cout << "Loading Second Set of POD Data... \n";
    good2 = LoadPOD(Data2, PODer2, my_id, numpids, dataname2);
    MPI_Barrier(MPI_COMM_WORLD);
    if (my_id == 0)
        {
        if (good2)
            cout << "Done loading second set of PODs \n";
        if (!good2)
            cout << "Failure to load second set of POD basis. \n";
        } // end if my_id is 0
    } // end else

PODData POD;
if (!loaddata)
    {
    if (my_id == 0)
        std::cout << "Generating First Set of POD Data... \n";
    good = GetPOD(Data, PODer, my_id, numpids, podenergy);
    if (good && my_id == 0)
        cout << "Done obtaining First POD representation.\n";
    if (!good && my_id == 0)
        cout << "Failure to obtain First POD model. \n";
```

```
    }
else // Load stored first POD basis
    {
    if (my_id == 0)
        std::cout << "Loading First Set of POD Data... \n";
    good = LoadPOD(Data, PODer, my_id, numpids, dataname);
    MPI_Barrier(MPI_COMM_WORLD);
    if (my_id == 0)
        {
        if (good)
            cout << "Done loading First set of PODs \n";
        if (!good)
            cout << "Failure to load First set of POD basis. \n";
        } // end if my_id is 0
    } // end else

PODer.GetPODData(POD);
PODer2.GetPODData(POD2);
std::vector < std::vector< double > > Ug;
std::vector < std::vector< double > > Vg;
std::vector < std::vector< double > > Us;
std::vector < std::vector< double > > Vs;
std::vector < std::vector< double > > Eg;
std::vector < std::vector< double > > Pg;
std::vector < std::vector< double > > Ps;
std::vector < std::vector< double > > Ts;
std::vector < std::vector< double > > Ug2;
std::vector < std::vector< double > > Vg2;
std::vector < std::vector< double > > Us2;
std::vector < std::vector< double > > Vs2;
std::vector < std::vector< double > > Eg2;
std::vector < std::vector< double > > Pg2;
std::vector < std::vector< double > > Ps2;
std::vector < std::vector< double > > Ts2;
loadall = 0;
if (!loaddata)
    dataname = "./Output";
PODer.GetSol(loadall, dataname, loaddata,
            Ug, Vg, Us, Vs, Eg, Pg, Ps, Ts);
if (exvar == strVg)
    Data.FormBBC(exval, exval2, Lx, Rx, bbc);
else
    Data.ExtractBBC(bbc);

if (!loaddata2)
```

```
      dataname2 = "./Output";
    PODer2.GetSol(loadall, dataname2, loaddata2,
            Ug2, Vg2, Us2, Vs2, Eg2, Pg2, Ps2, Ts2);
//    PODSolver IPOD;
//    good = IPOD.InterpPOD(Ug2, Vg2, Us2, Vs2, Eg2, Pg2, Ps2, Ts2,
//              PODer2.GetNx(), PODer2.GetNy(), PODer2.GetNt(),
//              my_id, numpids, podenergy, Data2);
//    if (my_id == 0 && good)
//      std::cout << "Generating Second Set of POD Data... \n";
//    good = GetPOD(Data2, PODer2, my_id, numpids, podenergy);
//    if (good && my_id == 0)
//      std::cout << "Solution Interpolated. \n";
//    if (!good && my_id == 0)
//      cout << "Failure to obtain Interpolated Solution. \n";

    NSSolver NSI;
    NSI.InitNS(Data, POD, 1, exvar, exval, bbc,
            Ug, Vg, Us, Vs, Eg,
            Pg, Ps, Ts, my_id, numpids,
            maxits, tol, eps1, eps2);


    if (exvar == strVg)
      NSI.InterpSol(exval, true, false, Data2, POD2,
            Ug2, Vg2, Us2, Vs2, Eg2, Pg2, Ps2, Ts2);
    else if (exvar == strE)
      NSI.InterpSol(exval, false, true, Data2, POD2,
            Ug2, Vg2, Us2, Vs2, Eg2, Pg2, Ps2, Ts2);
//    if (exvar == strVg)
//      NSI.InterpSol2(exval, true, false, Data2, POD2);
//    else if (exvar == strE)
//      NSI.InterpSol2(exval, false, true, Data2, POD2);
    else
      {
      if (my_id == 0)
        std::cout << "Invalid Interpolation variable";
      exit(-1);
      }

//    MPI_Barrier(MPI_COMM_WORLD);
    if (OFormat == 1)
      {
      VTKWriter VTK(NSI.POD.GetNx(), NSI.POD.GetNy(),
            NSI.POD.Getdx(), NSI.POD.Getdy());
      for (i=0; i < NSI.IndexVec.size(); i++)
```

```
            {
            tmpV = NSI.PgSum;
            VTK.WritePg(tmpV,NSI.IndexVec.at(i),
                      NSI.DtVec.at(i),i);
            VTK.WriteEg(NSI.EgSum,NSI.IndexVec.at(i),
                      NSI.DtVec.at(i),i);
            }
         std::cout << my_id << ": VTK Files Written. \n";
         }
      else if (OFormat == 2)
         {
         if (my_id == 0)
            {
            AspenWriter Aspen(NSI);
            Aspen.Write();
            std::cout << my_id << ": Aspen Files Written. \n";
            }
         }

      }// end interp

   MPI_Barrier(MPI_COMM_WORLD);
   // CLOSE MPI
   MPI_Finalize();
   std::cout << "Closing MPI... \n";
  } // end else

   std::cout << "Exiting mfixpod. \n";

   return 0;
   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

   bool controller(int argc)
      {

      //cout << "In controller " << endl;
      if (argc < 2)
         {
         cout << "usage:  mfix_reader.exe RUN_NAME\n";
         return false;
         }
      else
```

```
        return true;

    } // end controller

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

  bool Validate(MfixData & D, SimVars & SV, float & f, int index)

    {

    int di = 0;
    int i, j;

    bool good = true;
    float diff_ff;
    float diff_Ug;
    float diff_Vg;
    float diff_Us;
    float diff_Vs;
    std::ostringstream os;
    ofstream myfileff;
    ofstream myfileug;
    ofstream myfilevg;
    ofstream myfileus;
    ofstream myfilevs;

    D.GetVariableAtTimestep(0, index);
    std::vector<float> EP_G = D.var;

    D.GetVariableAtTimestep(1, index);
    std::vector<float> P_G = D.var;

    D.GetVariableAtTimestep(3, index);
    std::vector<float> U_G = D.var;

    D.GetVariableAtTimestep(4, index);
    std::vector<float> V_G = D.var;

    D.GetVariableAtTimestep(6, index);
    std::vector<float> U_S = D.var;

    D.GetVariableAtTimestep(7, index);
    std::vector<float> V_S = D.var;
```

```
        D.GetVariableAtTimestep(10, index);
        std::vector<float> T_G = D.var;

        D.GetVariableAtTimestep(11, index);
        std::vector<float> T_S = D.var;

        os << "ValidateFF" << index << ".out";
        myfileff.open( os.str().c_str() );
        os.seekp(0);

        os << "ValidateUg" << index << ".out";
        myfileug.open( os.str().c_str() );
        os.seekp(0);

        os << "ValidateVg" << index << ".out";
        myfilevg.open( os.str().c_str() );
        os.seekp(0);

        os << "ValidateUs" << index << ".out";
        myfileus.open( os.str().c_str() );
        os.seekp(0);

        os << "ValidateVs" << index << ".out";
        myfilevs.open( os.str().c_str() );
        os.seekp(0);
        int numX = SV.GetNx();
        int numY = SV.GetNy();
        int tindex = 0;

        for (j=numY+1;  j >= 0; --j)
           {
           for (i=0; i < numX+2; ++i)
              {
//cout << i << "=i and j=" << j << endl;
              if (!((i == 0) || (i >= numX+1) ||
                   (j == 0) || (j >= numY+1)))
                 {
                 di = i + (j*(numX+2));
//cout << i << "=i and j=" << j << " with di=" << di << endl;
                 diff_ff = EP_G.at(di) - SV.Gas.GetEP(tindex,i-1,j-1);
//cout << EP_G.at(di) << "=mfix and spoutpod=" << SV.Gas.GetEP(i-1,j-1) << endl;
                 diff_Ug = U_G.at(di) - SV.Gas.GetU(tindex,i-1,j-1);
                 diff_Vg = V_G.at(di) - SV.Gas.GetV(tindex,i-1,j-1);
                 diff_Us = U_S.at(di) - SV.Particle.GetU(tindex,i-1,j-1);
                 diff_Vs = V_S.at(di) - SV.Particle.GetV(tindex,i-1,j-1);
```

```
                myfileff << "," << diff_ff;
                myfileug << "," << diff_Ug;
                myfilevg << "," << diff_Vg;
                myfileus << "," << diff_Us;
                myfilevs << "," << diff_Vs;
                } // end if

            } // end for i

        myfileff << endl;
        myfileug << endl;
        myfilevg << endl;
        myfileus << endl;
        myfilevs << endl;
        } // end for j

    myfileff.close();
    myfileug.close();
    myfilevg.close();
    myfileus.close();
    myfilevs.close();

    // setting large value for f here
    // need to generate function to return next time
    // comparison value and assign it to f...
    f = 50000;

    return good;
    } // end Validate

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

bool ProcessSDataFile(string & sp_name, string & mfix_name,
                      string & dataname,
                      bool & loaddata, bool & project,
                      bool & extrap, double & etime,
                      string & exvar, double & exval,
                      std::vector<double> & energy, double & exval2,
                      double & Lx, double & Rx, string & mfix_name2,
                      string & dataname2, bool & interp,
                      bool & loaddata2, int & mits, double & toler,
                      double & epsilon1, double & epsilon2, int & OF,
                      float & starttime, float & endtime)
```

```
{

// Anticipated mfixpod data input file is as follows
//
// Line 1:  Full path with prefix of MFix Files.  Note: The
//          corresponding (mfix.dat) is assumed to be present in this
//          directory as well.  MUST follow this by a space!!
// Line 2:  Starting time value in first position.
//          Ending time value in second position.
// Line 3:  The eight variable matrix energy levels (between 0 and 1)
//          to be captured. This is used for when POD values are
//          generated.  They should be listed in the following order:
//          Ug Vg Us Vs Eg Pg Ps Ts
// Line 4:  Load POD data indicator in first position (1=yes, 0=no)
//          NOTE: If no, rest of line is ignored...
//          second position: Directory of POD data files.
//          Do not include trailing "/"
//          Do this to bypass basis reconstruction.
// Line 5:  Output Format 0=Standard & 1=VTK also
// Line 6:  LM Parameters
//          MaxIts Tolerance Epsilon1 Epsilon2
// Line 7:  Time projection indicator in first position (1=yes, 0=no)
//          NOTE: If no, rest of line is ignored...
//          Ending time in second position (should be later than
//          original run to be worthwhile...
// Line 8:  Extrapolation (or Interpolation) data indicator in first
//          position (1=yes, 0=no).
//          NOTE: If no, rest of line is ignored...
//          Second variable is the variable to be extrapolated
//          for the new run (Vg=inlet y gas velocity,
//          E=coefficient of restitution).
// Line 9:  If E, the first variable should contain the value of the
//          variable of E to be extrapolated and the rest of line
//          will be ignored.  (Note: results are only valid if
//          this value is within epsilon to original value) If Vg,
//          the first variable should be the y-velocity surrounding
//          the jet, the second should be the central jet y-velocity
//          and the left endpoint followed by the right endpoint
//          of the central jet velocity.
// Line 10: Interpolation data indicator in first
//          position (1=yes, 0=no).
//          NOTE: If no, rest of line and file is ignored...
//          If yes, then the full path of the second set of MFIX
//          files (along with the mfix.dat) should follow (include
```

```
    //          a trailing space).
    // Line 11: Load POD data indicator in first position (1=yes, 0=no)
    //          for the second set. NOTE: If no, rest of line is ignored.
    //          second position: Directory of POD data files.
    //          Do not include trailing "/"
    //          Do this to bypass basis reconstruction.


    bool good = true;
//    char fname2[256];
    char junk[256];
//    char variable[256];
    int i;

    string temp, temp2, fname2, fname3;
    string strE ("E");
    string strVg ("Vg");
    int getflag;
    float energyval;

    std::ifstream ifile (sp_name.c_str());

    if (! ifile.is_open())
       good = false;
    else
       good = true;

    // Process Line 1
    std::getline(ifile, temp, ' ');
    mfix_name = temp.substr(0, temp.find('\n') - 1);
std::cout << mfix_name << " is the input file. \n";

    // Process Line 3
    ifile >> starttime;
    ifile >> endtime;
    std::cout << starttime << " is the start time bound. \n";
    std::cout << endtime << " is the end time bound. \n";


    // Process Line 3
    energy.clear();
    for (i=0;i <8; i++)
       {
       ifile >> energyval;
       energy.push_back(energyval);
```

```
        }

    // Process Line 4
    ifile >> getflag;

    if (getflag == 0)
        {
        dataname.assign("");
        ifile.getline(junk, 256);
        loaddata = false;
        }
    else
        {
        std::getline(ifile, fname2, ' ');
        std::getline(ifile, fname2, ' ');
//        std::cout << fname2 << " is the input file. \n";
        if (fname2.find('\n'))
            dataname = fname2.substr(0, fname2.find('\n'));
        else
            dataname = fname2;
        if (dataname.find(' '))
            dataname = dataname.substr(0, fname2.find(' '));
        else
            dataname = fname2;
        if (dataname.find('\r'))
            dataname = fname2.substr(0, fname2.find('\r'));
        else
            dataname = fname2;

//        std::cout << dataname << " is the file. \n";
        loaddata = true;
        }

    // Process Line 5
    ifile >> OF;

    // Process Line 6
    ifile >> mits >> toler >> epsilon1 >> epsilon2;


    // Process Line 7
    ifile >> getflag;

    if (getflag == 0)
        {
```

```
        project = false;
        ifile.getline(junk,256);
        }
    else
        {
        project = true;
        ifile >> etime;
        }

    // Process Line 8
    ifile >> getflag;

    if (getflag == 0)
        {
        extrap = false;
        }
    else
        {
        extrap = true;
        std::getline(ifile, temp, '\n');
        if (temp.find(strE) < temp.size() )
            exvar = strE;
        if (temp.find(strVg) < temp.size() )
            exvar = strVg;
        }

//std::cout << "Extrapolating init? ";
if (extrap)
    std::cout << "yes \n";
else
    std::cout << "no \n";

    // Process Line 9
    if (extrap)
        ifile >> exval;
std::cout << "exval is " << exval << "\n";
    if (exvar == strVg)
        {
        ifile >> exval2;
        ifile >> Lx;
        ifile >> Rx;
//std::cout << "exval2 is " << exval2 << "\n";
//std::cout << "Lx is " << Lx << "\n";
//std::cout << "Rx is " << Rx << "\n";
        }
```

```
   // Process Line 10
   ifile >> getflag;
std::cout << "Interp flag is " << getflag << "\n";
   if (getflag == 0)
      {
      interp = false;
      }
   else
      {
      interp = true;
      extrap = false;
std::cout << "Interpolating. \n";

      std::getline(ifile, mfix_name2, '\n');
std::cout << mfix_name2 <<"is mfix file2 \n";
      if (mfix_name2.find(' ') == 0)
         mfix_name2 = mfix_name2.substr(1, mfix_name2.find('\n'));
std::cout << mfix_name2 <<"is mfix file2 after leading trimming \n";

      if (mfix_name2.find(' '))
         mfix_name2 = mfix_name2.substr(0, mfix_name2.find(' '));
      if (mfix_name.find('\r'))
         mfix_name = mfix_name2.substr(0, mfix_name2.find('\r'));
      else
         mfix_name = mfix_name2;
std::cout << mfix_name2 <<"is mfix file2 after trimming \n";


      // Process Line 11
      ifile >> getflag;

      if (getflag == 0)
         {
         dataname2.assign("");
         ifile.getline(junk, 256);
         loaddata2 = false;
         }
      else
         {
         std::getline(ifile, fname3, '\n');
         if (fname3.find(' ') == 0)
            fname3 = fname3.substr(1, fname3.find('\n'));

         if (fname3.find('\n'))
```

```
            dataname2 = fname3.substr(0, fname3.find('\n'));
         else
            dataname2 = fname3;

         if (dataname2.find(' '))
            dataname2 = dataname2.substr(0, dataname2.find(' '));
         else
            dataname2 = dataname2;
         if (dataname2.find('\r'))
            dataname2 = fname3.substr(0, dataname2.find('\r'));
         else
            dataname2 = dataname2;

       std::cout << dataname2 << "is the second POD directory. \n";
         loaddata2 = true;
         }

       }

   ifile.close();

//std::cout << "Extrapolating? ";
//if (extrap)
//    std::cout << "yes \n";
//else
//    std::cout << "no \n";

   return good;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

bool GetPOD(SimVars & Data, PODSolver & PODer, int Id, int NIds,
         std::vector< double > & en)

   {

   bool good;

   PODer.SetNs(Data.GetNt(), Data.GetNx(), Data.GetNy(),
            Data.Getdelx(), Data.Getdely() );
//  This is the Old POD model based upon Badri's work
//    good = PODer.FormPOD(Data, energy, Id, NIds);
```

```
//  This is the POD model based upon Burkardt's paper
    good = PODer.FormSVDPOD(Data, en, Id, NIds);

    return good;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

bool LoadPOD(SimVars & Data, PODSolver & PODer, int Id, int NIds,
            string ln)

    {

    bool good;

    PODer.SetNs(Data.GetNt(), Data.GetNx(), Data.GetNy(),
              Data.Getdelx(), Data.Getdely() );

    good = PODer.LoadPOD(Data, Id, NIds, ln);

    return good;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    MFIXReader.h
//    Created by Stephanie R. Beck Roth
//    Last Modified:   5/17/2010
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#ifndef MFIXREADER_H
#define MFIXREADER_H

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "MfixData.h"
#include "PhaseData.h"

class MfixReader
   {
   private:
   // These need to be set when reading in the
   float MINTIME; // TIME seconds
   float MAXTIME; // TSTOP seconds
   float MINX; // ?? I think these are defaults!
   float MINY; // ?? I think these are defaults!
   float MAXX; // XLENGTH
   float MAXY; // YLENGTH

   public:

      MfixReader();
      ~MfixReader();

      float GetMaxTime();
      float GetMinTime();
      float GetNextTime(float t, int n);
      void GetData(MfixData & Data, int timestep, SimVars & SPV,
               float timept, int & index, float st, float et);
      void GetKeyPts(string line, size_t & start, size_t & end);
      void GetValPts(string line, size_t & start, size_t & end);
      int GetBC(string ifname, SimVars & SPV);
      int AssignBCs(string key, string value, SimVars & SPV);
      };
#endif
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    MFIXReader.cpp
//    Contains member functions for the MFIX Reader
//    Created by Stephanie R. Beck Roth
//    Last Modified:   06/15/2011
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "MFIXReader.h"

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

MfixReader::MfixReader()
   {

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

MfixReader::~MfixReader()
   {

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float MfixReader::GetMaxTime()
   {

   return MAXTIME;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float MfixReader::GetMinTime()
```

```
   {

   return MINTIME;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float MfixReader::GetNextTime(float t, int n)
   {

   float f;

   return f;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void MfixReader::GetData(MfixData & Data, int timestep, SimVars & SPV,
                        float timept, int & index, float st, float et)
   {
//    int k;
//    ofstream mylog("mylog.txt");
   if (index == 0)
       {
       SPV.SetNx( Data.imax2 - 2);
       SPV.SetNy( Data.jmax2 - 2);
       }

//    std::cout << SPV.GetNx() << " is # of X. \n";
//    std::cout << SPV.GetNy() << " is # of Y. \n";

   // Not doing 3D right now so this isn't needed...
   // SPV.numZ = Data.kmax2

//std::cout << "About to GetVariables \n";
   // Initial Void fraction in the gas
   // Note: Solid particle will be 1 - EP_g
   Data.GetVariableAtTimestep(0, timestep);
   std::vector<float> EP_g = Data.var;
//   for (k=0; k < EP_g.size(); k++)
//       mylog << EP_g.at(k) << " is EPg at " << k << endl;
```

```
    // Initial Gas pressure
    Data.GetVariableAtTimestep(1, timestep);
    std::vector<float> P_g = Data.var;

    // Initial Solids pressure
    Data.GetVariableAtTimestep(2, timestep);
    std::vector<float> P_star = Data.var;

    // X component of gas velocity
    Data.GetVariableAtTimestep(3, timestep);
    std::vector<float> U_g = Data.var;
//   for (k=0; k < U_g.size(); k++)
//      mylog << U_g.at(k) << " is Ug at " << k << endl;

    // Y component of gas velocity
    Data.GetVariableAtTimestep(4, timestep);
    std::vector<float> V_g = Data.var;
//   for (k=0; k < V_g.size(); k++)
//      mylog << V_g.at(k) << " is Vg at " << k << endl;

    // Z component of gas velocity
    Data.GetVariableAtTimestep(5, timestep);
    std::vector<float> W_g = Data.var;

    // X component of solids velocity
    Data.GetVariableAtTimestep(6, timestep);
    std::vector<float> U_s = Data.var;
//   for (k=0; k < U_s.size(); k++)
//      mylog << U_s.at(k) << " is Us at " << k << endl;

    // Y component of solids velocity
    Data.GetVariableAtTimestep(7, timestep);
    std::vector<float> V_s = Data.var;
//   for (k=0; k < V_s.size(); k++)
//      mylog << V_s.at(k) << " is Vs at " << k << endl;
//std::cout << "About to store data1.";

    // Z component of solids velocity
    Data.GetVariableAtTimestep(8, timestep);
    std::vector<float> W_s = Data.var;

    // Initial Bulk density of solids
    // Note:  ROP_s = RO_s x EP_s
    Data.GetVariableAtTimestep(9, timestep);
```

```
    std::vector<float> ROP_s = Data.var;

    // Initial Gas Temperature
    Data.GetVariableAtTimestep(10, timestep);
    std::vector<float> T_g = Data.var;
//std::cout << "Size of Temp vector: " << T_g.size() << "\n";
//std::cout << "About to store data2.";
    // Initial Solids Temperature
    Data.GetVariableAtTimestep(11, timestep);
    std::vector<float> T_s = Data.var;

    // Initial Mass fraction of Gas
    Data.GetVariableAtTimestep(12, timestep);
    std::vector<float> X_g = Data.var;

    // Initial Mass fraction of solids
    Data.GetVariableAtTimestep(13, timestep);
    std::vector<float> X_s = Data.var;

    // Initial solids granular temperature
    Data.GetVariableAtTimestep(14, timestep);
    std::vector<float> Theta_m = Data.var;
//   for (k=0; k < Theta_m.size(); k++)
//       mylog << Theta_m.at(k) << " is Theta_m at " << k << endl;


//mylog.close();

//std::cout << timept;
    if ((timept >= st) && (timept <= et))
        {
//std::cout  << ": About to store data3. \n";
//std::cout << EP_g.size() << " is the size of void fraction \n";
        SPV.StoreFluid(EP_g, P_g, U_g, V_g, W_g, T_g, X_g,
                timestep, index, timept);
//std::cout << "About to store data4.\n";
        SPV.StoreSolid(EP_g, P_star, U_s, V_s, W_s, ROP_s, T_s,
                X_s, Theta_m, timestep, index, timept);
//std::cout << "About to store data5. \n";
        SPV.StoreTimeStep(timept, index);
//        index++;
        }
//    else
//        std::cout  << ": Data not stored. \n";
    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void MfixReader::GetKeyPts(string line, size_t & start, size_t & end)
   {
   size_t pos;

   pos = 0;

   while ( ( line.at(pos) == ' ') && ( pos < line.size() - 1 ) )
      {
      pos++;
      } // end while is blank
   if ( line.at(pos) != ' ' )
      {
      start = pos;
      pos++;
      } //end if line's pos is not blank
   while ( ( line.at(pos) != ' ' ) && (pos < line.size() - 1 ) )
      {
      pos++;
      } // end while isn't blank
   // Not sure why I need an extra -1 here...
   end = pos - 1 -1;

   } // end GetKeyPts

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void MfixReader::GetValPts(string line, size_t & start, size_t & end)
   {
   size_t pos;
   bool done;

   // start comes into this routine with the position of the = sign
   pos = start;
   done = false;
   do
      {
      pos++;
      if (pos < line.size() )
        {
        if ( line.at(pos) != ' ' )
```

```
            done = true;
          }
        else
          done = true;
        } // end do while loop
    while (!done);

    start = pos;
    done = false;
    while (pos < line.size() && !done)
        {
        if ( line.at(pos) != ' ' )
           pos++;
        else
           done = true;
        } // end if line's pos isn't blank and
    end = pos;

    } // end GetValPts

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int MfixReader::AssignBCs(string key, string value, SimVars & SPV)
    {
    int found = 1;
//std::cout << "key is " << key << "\n";
//std::cout << "value is " << value << "\n";
    if ( key == "TIME" )
        { // Start time of the run
//std::cout << "Assign min time \n";
        MINTIME = atof( value.c_str() );
        }
    else if ( key == "TSTOP" )
        { // Stop time of the run
        MAXTIME = atof( value.c_str() );
        }
    else if ( key == "DT" )
        { // Starting time step
        SPV.Setdelt( atof( value.c_str() ) );
        }
    else if ( key == "XLENGTH" )
        { // length of X data in experiment
        SPV.Setxdist( atof( value.c_str() ) );
        }
```

```
else if ( key == "YLENGTH" )
   { // length of Y data in experiment
   SPV.Setydist( atof( value.c_str() ) );
   }
else if ( key == "IMAX" )
   { // # of X Data points in interior
     // Recall the border has ghost cells
     // IMAX + 2 is the # of x pts stored
   SPV.SetNx( atoi( value.c_str() ) + 1 );
   }
else if ( key == "JMAX" )
   { // # of Y Data points in interior
     // Recall the border has ghost cells
     // JMAX + 2 is the # of y pts stored
   SPV.SetNy( atoi( value.c_str() ) + 1 );
   }
else if ( key == "drag_c1" )
   { // obtained via umf.xls spreadsheet???
   SPV.SetDragC( atof( value.c_str() ) );
   }
else if ( key == "drag_d1" )
   { // obtained via umf.xls spreadsheet???
   SPV.SetDragD( atof( value.c_str() ) );
   }
else if ( key == "RO_g0" )
   { // Specified constant Gas Density
   SPV.Gas.SetDensity( atof( value.c_str() ) );
   }
else if ( key == "MU_g0" )
   { // Specified constant Gas Viscosity
   SPV.Gas.SetViscosity( atof( value.c_str() ) );
   }
else if ( key == "MU_gmax" )
   { // Specified Maximum Gas Viscosity
   SPV.Gas.SetMaxViscosity( atof( value.c_str() ) );
   }
else if ( key == "IC_L_scale" )
   { // Specified turbulent length
   SPV.Gas.Setlscale( atof( value.c_str() ) );
   }
else if ( key == "MU_s0" )
   { // Specified constant Solids Viscosity
   SPV.Particle.SetViscosity( atof( value.c_str() ) );
   }
else if ( key == "MW_avg" || key == "MW_AVG" )
```

```
    { // average molecular weight of gas
    }
else if ( key == "D_p0" )
    { // Initial particle diameters
    SPV.Particle.SetDiam( atof( value.c_str() ) );
    }
else if ( key == "RO_s" )
    { // Particle Densities
    SPV.Particle.SetDensity( atof( value.c_str() ) );
    }
else if ( key == "e" )
    { // restitution coefficient
    SPV.SetResCoeff( atof( value.c_str() ) );
    }
else if ( key == "Phi" )
    { // angle of internal friction in degrees
    SPV.SetAngIntFric( atof( value.c_str() ) );
    }
else if ( key == "EP_star" )
    { // Packed bed void fraction
    SPV.SetPackedBedEP( atof( value.c_str() ) );
    }
// INITIAL CONDITIONS SECTION
else if ( key == "IC_X_w(1)" )
    { // BED: left X endpoint
    }
else if ( key == "IC_X_e(1)" )
    { // BED: right X endpoint
    }
else if ( key == "IC_Y_s(1)")
    { // BED: bottom Y endpoint

    }
else if ( key == "IC_Y_n(1)" )
    { // BED: top Y endpoint
    }
else if ( key == "IC_EP_g(1)" )
    { // BED: Initial void fractions
    }
else if ( key == "IC_U_g(1)" )
    { // BED: Initial X component of gas velocity
    }
else if ( key == "IC_V_g(1)" )
    { // BED: Initial Y component of gas velocity
    }
```

```
else if ( key == "IC_T_g(1)" )
   { // BED: Initial Temperature of the gas
   }
else if ( key == "IC_U_s(1,1)" )
   { // BED: Initial X component of solids velocity
   }
else if ( key == "IC_V_s(1,1)" )
   { // BED: Initial Y component of solids velocity
   }
else if ( key == "IC_THETA_M(1,1)" || key == "IC_Theta_m(1,1)" )
   { // BED: Initial solids phase-m granular temperature
   }
  // These ICs are for the Freeboard region
  // the top portion above the bed surface
else if ( key == "IC_X_w(2)" )
   { // FreeBoard: left X endpoint
   }
else if ( key == "IC_X_e(2)" )
   { // FreeBoard: right X endpoint
   }
else if ( key == "IC_Y_s(2)" )
   { // FreeBoard: bottom Y endpoint
   }
else if ( key == "IC_Y_n(2)" )
   { // FreeBoard: top Y enpoint
   }
else if ( key == "IC_EP_g(2)" )
   { // FreeBoard: Initial gas void fraction
   }
else if ( key == "IC_U_g(2)" )
   { // FreeBoard: Initial X component of gas velocity
   }
else if ( key == "IC_V_g(2)" )
   { // FreeBoard: Initial Y component of gas velocity
   }
else if ( key == "IC_T_g(2)" )
   { // FreeBoard: Initial gas phase temerature
   }
else if ( key == "IC_U_s(2,1)" )
   { // FreeBoard: Initial X component of solids velocity
   }
else if ( key == "IC_V_s(2,1)" )
   { // FreeBoard: Initial Y component of solids velocity
   }
else if ( key == "IC_THETA_M(2,1)" || key == "IC_Theta_m(2,1)" )
```

```
    { // FreeBoard: Initial solids granular temperature
    }
// BOUNDARY CONDTIONS SECTION
else if ( key == "BC_X_w(1)" )
    { // JET: left X endpoint
    SPV.Gas.slX = atof( value.c_str() );
    }
else if ( key == "BC_X_e(1)" )
    { // JET: right X endpoint
    SPV.Gas.srX = atof( value.c_str() );
    }
else if ( key == "BC_Y_s(1)" )
    { // JET: bottom Y endpoint
    }
else if ( key == "BC_Y_n(1)" )
    { // JET: top Y endpoint
    }
else if ( key == "BC_TYPE(1)" )
    { // JET: Boundary Type of (1) = JET :D
    }
else if ( key == "BC_EP_g(1)" )
    { // JET: Inflow of the gas void fraction
    SPV.Gas.bE = atof( value.c_str() );
    }
else if ( key == "BC_U_g(1)" )
    { // JET: X component of gas velocity
    SPV.Gas.bU = atof( value.c_str() );
    }
else if ( key == "BC_V_g(1)" )
    { // JET: Y component of gas velocity
    SPV.Gas.sbV = atof( value.c_str() );
    }
else if ( key == "BC_P_g(1)" )
    { // JET: inflow gas pressure
    SPV.Gas.bP = atof( value.c_str() );
    }
else if ( key == "BC_T_g(1)" )
    { // JET: inflow gas Temperature
    SPV.Gas.bT = atof( value.c_str() );
    }
  // Uniform Background flow
else if ( key == "BC_X_w(5)" )
    { // BACKGROUND: left X endpoint
    }
else if ( key == "BC_X_e(5)" )
```

```
    { // BACKGROUND: right X endpoint
    }
else if ( key == "BC_Y_s(5)" )
    { // BACKGROUND: bottom Y endpoint
    }
else if ( key == "BC_Y_n(5)" )
    { // BACKGROUND: top Y endpoint
    }
else if ( key == "BC_TYPE(5)" )
    { // BACKGROUND: Type of flow (in)
    }
else if ( key == "BC_EP_g(5)" )
    { // BACKGROUND: gas void fraction
    }
else if ( key == "BC_U_g(5)" )
    { // BACKGROUND: X component flow velocity
    }
else if ( key == "BC_V_g(5)" )
    { // BACKGROUND: Y component flow velocity
    SPV.Gas.bV = atof( value.c_str() );
    }
else if ( key == "BC_P_g(5)" )
    { // BACKGROUND: gas inflow pressure
    }
else if ( key == "BC_T_g(5)" )
    { // BACKGROUND: gas inflow temperature
    }
  // Uniform Background flow
else if ( key == "BC_X_w(6)" )
    { // BACKGROUND: left X endpoint
    }
else if ( key == "BC_X_e(6)" )
    { // BACKGROUND: right X endpoint
    }
else if ( key == "BC_Y_s(6)" )
    { // BACKGROUND: bottom Y endpoint
    }
else if ( key == "BC_Y_n(6)" )
    { // BACKGROUND: top Y endpoint
    }
else if ( key == "BC_TYPE(6)" )
    { // BACKGROUND: Type of flow (in)
    }
else if ( key == "BC_EP_g(6)" )
    { // BACKGROUND: gas void fraction
```

```
    }
else if ( key == "BC_U_g(6)" )
    { // BACKGROUND: X component flow velocity
    }
else if ( key == "BC_V_g(6)" )
    { // BACKGROUND: Y component flow velocity
    }
else if ( key == "BC_P_g(6)" )
    { // BACKGROUND: gas inflow pressure
    }
else if ( key == "BC_T_g(6)" )
    { // BACKGROUND: gas inflow temperature
    }
  // EXIT CONDITIONS
else if ( key == "BC_X_w(2)" )
    { // Exit: left X endpoint
    }
else if ( key == "BC_X_e(2)" )
    { // Exit: right X endpoint
    }
else if ( key == "BC_Y_s(2)" )
    { // Exit: bottom Y endpoint
    }
else if ( key == "BC_Y_n(2)" )
    { // Exit: top Y endpointsams club
    }
else if ( key == "BC_TYPE(2)" )
    { // Exit: constant pressure and Temp outflow
    }
else if ( key == "BC_P_g(2)" )
    { // Exit: value of outflow gas pressure
    SPV.Gas.tP = atof( value.c_str() );
    }
else if ( key == "BC_T_g(2)" )
    { // Exit: value of outflow gas temperature
    SPV.Gas.tT = atof( value.c_str() );
    }
  // LEFT WALL
else if ( key == "BC_X_w(3)" )
    { // Left: left X coordinate
    }
else if ( key == "BC_X_e(3)" )
    { // Left: right X coordinate
    }
else if ( key == "BC_Y_s(3)" )
```

```
    { // Left: bottom Y coordinate
    }
else if ( key == "BC_Y_n(3)" )
    { // Left: top Y coordinate
    }
else if ( key == "BC_TYPE(3)" )
    { // Left: Type of Wall Slip Condition
      //  (NO? (current setting) /Partial?)
    }
else if ( key == "BC_JJ_PS(3)" )
    { // Left: Johnson and Jackson partial
      // slip boundary?? 1 = Y 0 = N
    }
else if ( key == "BC_Uw_s(3,1)" )
    { // Left: Solids phase U for partial
    }
else if ( key == "BC_Vw_s(3,1)" )
    { // Left: Solids phase V for partial
    }
else if ( key == "BC_Thetaw_m(3,1)" )
    { // Left: Solids phace Theta (T sub w)
      //       for partial
    }
  // RIGHT WALL
else if ( key == "BC_X_w(4)" )
    { // Right: left X coordinate
    }
else if ( key == "BC_X_e(4)" )
    { // Right: right X coordinate
    }
else if ( key == "BC_Y_s(4)" )
    { // Right: bottom Y coordinate
    }
else if ( key == "BC_Y_n(4)" )
    { // Right: top Y coordinate
    }
else if ( key == "BC_TYPE(4)" )
    { // Right: Type of Wall Slip Condition
      //  (NO? (current setting) /Partial?)
    }
else if ( key == "BC_JJ_PS(4,1)" )
    { // Right: Johnson and Jackson partial
      // slip boundary?? 1 = Y 0 = N
    }
else if ( key == "BC_Uw_s(4,1)" )
```

```
        { // Right: Solids phase U for partial
        }
    else if ( key == "BC_Vw_s(4,1)" )
        { // Right: Solids phase V for partial
        }
    else if ( key == "BC_Thetaw_m(4,1)" )
        { // Right: Solids phace Theta (T sub w)
          //       for partial
        }
    else if ( key == "RUN_NAME" )
        { // NOT storing at the moment
        }
    else if ( key == "DESCRIPTION" )
        { // NOT storing at the moment
        }
    else if ( key == "RUN_TYPE" )
        { // NOT storing at the moment
        }
    else if ( key == "UNITS" )
        { // NOT storing at the moment
        }
    else
        {
// Way to much stuff in here that isn't being stored...
//       found = 0;
        }

    return found;

    } // end AssignBCs

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int MfixReader::GetBC(string ifname, SimVars & SPV)
    {  // const char * name

    string line;
    string datname;
    string slash;
    size_t slashpos;
    size_t begin;
    size_t end;

    string key;
```

```
    string value;

    int found = 2;
//std::cout << "In BC \n";
    slash = '/';
    slashpos = ifname.rfind(slash);
    if (slashpos != string::npos)
        {
        datname = ifname.substr(0,slashpos + 1);
        datname = datname + "mfix.dat";
        } // end if slash does exist
    else
        datname = "mfix.dat";

    std::ifstream bcfile( datname.c_str() );
    if (! bcfile.is_open())
        {
        std::cout << "Error opening mfix.dat. \n";
        exit (-1);
        }
    size_t equalsignpos;
//std::cout << "Getting line... \n";
    while ( getline(bcfile, line) && ( found > 0 ) )
        {
//std::cout << "Got Line \n";
        equalsignpos = line.find('=');
//std::cout << "= is at " << equalsignpos
//          << " where npos is " << line.npos << "\n";
        if (line.size() > 0)
            { // Isn't an empty line...
            if ((line.at(0) != '!') && (line.at(0) != '#') &&
                (equalsignpos != line.npos))
                {
                GetKeyPts(line, begin, end);
                key = line.substr(begin, end);
//std::cout << "Have Key Pts... " << key << ". \n";
                GetValPts(line, equalsignpos, end);
                // Note: equalsignpos has a good close
                // starting pos and will return with the exact pos
                value = line.substr(equalsignpos, end);
//std::cout << "Have Value... " << value << ". \n";
                found = AssignBCs(key, value, SPV);
                } // end if line isn't a comment
            } // end if line isn't blank...
        } // end while
```

```
    return found;

    } // end GetBC

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    PhaseData.h
//    Created by Stephanie R. Beck Roth
//    Last Modified:   01/12/2010
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#ifndef PHASEDATA_H
#define PHASEDATA_H

#include <iostream>
#include <iomanip>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include <string>
#include <math.h>

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

class Fluid
  {

  private:

      float density;
      float viscosity;
      float Mugmax;
      float lscale;

      int Nx;
      int Ny;
      int Nt;

      std::vector< std::vector< std::vector< float > > > UgEvalues;
      std::vector< std::vector< std::vector< float > > > VgEvalues;

  public:

      Fluid();
      ~Fluid();
```

```
float bT;
float bP;
float bU;
float bE;
float slX;
float srX;
float sbV;
float bV;
float tT;
float tP;

//Vector Data is stored x, y and lastly time
std::vector< std::vector< std::vector< float > > > EP;
std::vector< std::vector< std::vector< float > > > P;
std::vector< std::vector< std::vector< float > > > U;
std::vector< std::vector< std::vector< float > > > V;
std::vector< std::vector< std::vector< float > > > W;
std::vector< std::vector< std::vector< float > > > T;
std::vector< std::vector< std::vector< float > > > X;

void SetNx(int i);
void SetNy(int i);
void SetNt(int i);

float Getlscale();
void Setlscale(float f);

float GetDensity();
void SetDensity(float f);

float GetViscosity();
void SetViscosity(float f);

float GetMaxViscosity();
void SetMaxViscosity(float f);

void UpdateEP(std::vector< std::vector< float > > & N);
void UpdateP(std::vector< std::vector< float > > & N);
void UpdateU(std::vector< std::vector< float > > & tmp);
void UpdateV(std::vector< std::vector< float > > & tmp);
void UpdateT(std::vector< std::vector< float > > & tmp);

float GetEP(int t, int nx, int ny);
float GetV(int t, int nx, int ny);
float GetU(int t, int nx, int ny);
```

```
      float GetP(int t, int nx, int ny);

      float GetBC_EP(int nx, int ny);
      float GetBC_P(int nx, int ny);
      float GetBC_U(int nx, int ny, int mx, int my);
      float GetBC_V(int nx, int ny, int mx, int my);
      float GetBC_T(int nx, int ny);

      void AddUgEval(std::vector< std::vector< float > > & v);
      void AddVgEval(std::vector< std::vector< float > > & v);

      void  Convert2MatrixU(std::vector< std::vector< float > > & In);
      void  Convert2MatrixV(std::vector< std::vector< float > > & In);

   };

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

class Solid
   {
   private:

      float density;
      float viscosity;
      float diam;
      float E;

      int Nx;
      int Ny;
      int Nt;

      std::vector< std::vector< std::vector< float > > > UsEvalues;
      std::vector< std::vector< std::vector< float > > > VsEvalues;

   public:

      Solid();
      ~Solid();

      //Vector Data is stored x, y and lastly time
      std::vector< std::vector< std::vector< float > > > EP;
      std::vector< std::vector< std::vector< float > > > P;
      std::vector< std::vector< std::vector< float > > > U;
```

```
    std::vector< std::vector< std::vector< float > > > V;
    std::vector< std::vector< std::vector< float > > > W;
    std::vector< std::vector< std::vector< float > > > T;
    std::vector< std::vector< std::vector< float > > > X;
    std::vector< std::vector< std::vector< float > > > ROP;
    std::vector< std::vector< std::vector< float > > > Theta_m;

    void SetNx(int i);
    void SetNy(int i);
    void SetNt(int i);

    float GetDensity();
    void SetDensity(float f);

    float GetViscosity();
    void SetViscosity(float f);

    float GetDiam();
    void SetDiam(float f);

    void UpdateEP(std::vector< std::vector< float > > & N);
    void UpdateP(std::vector< std::vector< float > > & N);
    void UpdateU(std::vector< std::vector< float > > & tmp);
    void UpdateV(std::vector< std::vector< float > > & tmp);
    void UpdateT(std::vector< std::vector< float > > & tmp);

    float GetEP(int t, int nx, int ny);
    float GetV(int t, int nx, int ny);
    float GetU(int t, int nx, int ny);
    float GetT(int t, int nx, int ny);
    float GetP(int t, int nx, int ny);
    float GetTheta_m(int t, int nx, int ny);
    float GetTheta(int t, int nx, int ny);

    void AddUsEval(std::vector< std::vector< float > > & v);
    void AddVsEval(std::vector< std::vector< float > > & v);

    void  Convert2MatrixU(std::vector< std::vector< float > > & In);
    void  Convert2MatrixV(std::vector< std::vector< float > > & In);

  };

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
class SimVars
   {
   private:

      float Avagadro;
      float Rydberg;

      float erescoeff;

      float Tol;

      float Vol2DCell;
      float xdist;
      float ydist;
      float delx;
      float dely;
      float delt;
      int Nxcells;
      int Nycells;
      int Ntimes;

      float AngIntFric; // angle of internal friction (Phi)
      float pb_EP;
      float DragC; // Symlal and O'Brien drag_c1 coeff.
      float DragD; // Symlal and O'Brien drag_d1 coeff.

      int NumTimePts;

   public:

      SimVars();
      ~SimVars();

      Fluid Gas;
      Solid Particle;

      std::vector< float > TimePts;

      void Setxdist(float f);
      void Setydist(float f);
      void SetNx(int f);
      void SetNy(int f);
      void SetTimes(int i);

      void SetTol(float t);
```

```
float GetTol();


void SetTimept(float t);
float GetTimept(int i);
float LastTimept();

float Getdelx();
float Getdely();
float Getdelt();
float GetCalcdelt();
void Setdelt(float dt);

float GetX(int i);
float GetY(int i);

int Getxindex(int i);
int Getyindex(int i);

int GetNx();
int GetNy();
int GetNt();

void SetDragC(float f);
void SetDragD(float f);

float GetAvagadro();
float GetRydberg();

void Set2DVolVars();
float Get2DVol();

void StoreFluid(std::vector<float> & EP_g,
                std::vector<float> & P_g,
                std::vector<float> & U_g,
                std::vector<float> & V_g,
                std::vector<float> & W_g,
                std::vector<float> & T_g,
                std::vector<float> & X_g,
                int timestep, int fi, float ts);

void StoreSolid(std::vector<float> & EP_g,
                std::vector<float> & P_star,
                std::vector<float> & U_s,
                std::vector<float> & V_s,
```

```
                std::vector<float> & W_s,
                std::vector<float> & ROP_s,
                std::vector<float> & T_s,
                std::vector<float> & X_s,
                std::vector<float> & Theta_m,
                int timestep, int si, float ts);
    void StoreTimeStep(float timept, int ts);

    float FindEvalue(float a2, float a1, float a0, float oeval);

    float GetGasXVel(int t, int i, int j);
    float GetGasYVel(int t, int i, int j);
    float GetParticleXVel(int t, int i, int j);

    float GetParticleYVel(int t, int i, int j);

    float GetE(); // ResCoeff
    void  SetResCoeff(float val);

    void  SetAngIntFric(float val);
    void  SetPackedBedEP(float val);

    float GetTemp(int t, int nx, int ny);

    void Find0Evalues();

    void ExtractBBC(std::vector< double > & v);
    void FormBBC(double & b, double & j, double & e1,
            double & e2, std::vector< double > & v);

    float GetBLB(float xn, float yn, float x, float y);
    float GetFBLB();
    float GetBLBx(float xn, float yn, float x, float y, float dy);
    float GetBLBy(float xn, float yn, float x, float y, float dx);

  };

#endif
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    PhaseData.cpp
//    Created by Stephanie R. Beck Roth
//    Last Modified:   02/21/2011
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#include <iostream>
#include <iomanip>
#include <fstream>
#include <sstream>
#include <string>
#include <vector>
#include <cmath>
#include <math.h>
#include "PhaseData.h"

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

Fluid::Fluid()
   {
   Mugmax = 0.5;
   lscale = 0.0;
   EP.clear();
   P.clear();
   U.clear();
   V.clear();
   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

Fluid::~Fluid()
   {

   }


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetDensity()
   {
```

```
    return density;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::SetDensity(float f)
    {

    density = f;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::Getlscale()
    {

    return lscale;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::Setlscale(float f)
    {

    lscale = f;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetViscosity()
    {

    return viscosity;

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::SetViscosity(float f)
   {

   viscosity = f;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetMaxViscosity()
   {

   return Mugmax;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::SetMaxViscosity(float f)
   {

   Mugmax = f;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::UpdateEP(std::vector< std::vector< float > > & N)
   {

   EP.push_back(N);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::UpdateP(std::vector< std::vector< float > > & N)
   {
```

```
   P.push_back(N);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::UpdateU(std::vector< std::vector< float > > & tmp)
   {

   U.push_back(tmp);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::UpdateV(std::vector< std::vector< float > > & tmp)
   {

   V.push_back(tmp);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::UpdateT(std::vector< std::vector< float > > & tmp)
   {

   T.push_back(tmp);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetEP(int t, int nx, int ny)
   {

   return EP.at(t).at(nx).at(ny);

   }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetP(int t, int nx, int ny)
   {

   return P.at(t).at(nx).at(ny);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetU(int t, int nx, int ny)
   {

   return U.at(t).at(nx).at(ny);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetV(int t, int nx, int ny)
   {

   return V.at(t).at(nx).at(ny);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetBC_EP(int nx, int ny)
   {

   return bE;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetBC_P(int nx, int ny)
   {
```

```
    float f;

    if (ny == 0)
       f = bP;
    else if (nx == 0)
       f = P.back().at(nx).at(ny);
    else if (nx > 0)
       f = P.back().at(nx).at(ny);
    else if (ny > 0)
       f = tP;

    return f;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetBC_U(int nx, int ny, int mx, int my)
    {

    float f;

    if (ny == 0)
       f = bU;
    else if (nx == 0)
       f = 0.0;
//       f = U.back().at(nx).at(ny);
    else if ((nx+1) == mx)
       f = 0.0;
//       f = U.back().at(nx).at(ny);
    else if ((ny+1) == my)
       f = 0.0;
//       f = U.back().at(nx).at(ny);

    return f;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetBC_V(int nx, int ny, int mx, int my)
    {
```

```
    float f;

    if (ny == 0)
        {
        if (((nx / mx) < slX) ||
            ((nx / mx) > srX))
            f = sbV;
        else
            f = bV;
        }
    else if (nx == 0)
        f = 0.0;
//       f = V.back().at(nx).at(ny);
    else if ((nx+1) == mx)
        f = 0.0;
//       f = V.back().at(nx).at(ny);
    else if ((ny+1) == my)
        f = 0.0;
//       f = V.back().at(nx).at(ny);


    return f;
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Fluid::GetBC_T(int nx, int ny)

    {

    float f;

    if (ny == 0)
        f = bT;
    else if (nx == 0)
        f = bT;
    else if (nx > 0)
        f = tT;
    else if (ny > 0)
        f = tT;

    return f;
```

```
   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::AddUgEval(std::vector< std::vector< float > > & v)
   {

   UgEvalues.push_back(v);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::AddVgEval(std::vector< std::vector< float > > & v)
   {

   VgEvalues.push_back(v);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::Convert2MatrixU(std::vector< std::vector< float > > & In)
   {

   int i;
   int j;
   int t;
   std::vector< float > tmp;

   In.clear();
   for (t=0; t < Nt; t++)
      {
      tmp.clear();
      for (j=0; j < Ny; j++)
         for (i=0; i < Nx; i++)
            {
            tmp.push_back( GetU(t,i,j) );
            } // end i j for
      In.push_back(tmp);
      } // end t for
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::Convert2MatrixV(std::vector< std::vector< float > > & In)
    {

    int i;
    int j;
    int t;
    std::vector< float > tmp;

    In.clear();
    for (t=0; t < Nt; t++)
        {
        tmp.clear();
        for (j=0; j < Ny; j++)
            for (i=0; i < Nx; i++)
                {
                tmp.push_back( GetV(t,i,j) );
                } // end i j for
        In.push_back(tmp);
        } // end t for

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::SetNx(int i)

    {

    Nx = i;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::SetNy(int i)

    {
```

```
    Ny = i;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Fluid::SetNt(int i)

    {

    Nt = i;

    }


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

Solid::Solid()
    {
    EP.clear();
    P.clear();
    U.clear();
    V.clear();
    T.clear();
    Theta_m.clear();
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

Solid::~Solid()
    {

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Solid::GetDensity()
    {

    return density;
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::SetDensity(float f)
    {

    density = f;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Solid::GetViscosity()
    {

    return viscosity;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::SetViscosity(float f)
    {

    viscosity = f;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Solid::GetDiam()
    {

    return diam;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
void Solid::SetDiam(float f)
   {

   diam = f;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::SetNx(int i)

   {

   Nx = i;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::SetNy(int i)

   {

   Ny = i;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::SetNt(int i)

   {

   Nt = i;

   }


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::UpdateEP(std::vector< std::vector< float > > & N)
```

```
   {

   EP.push_back(N);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::UpdateP(std::vector< std::vector< float > > & N)
   {

   P.push_back(N);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::UpdateU(std::vector< std::vector< float > > & tmp)
   {

   U.push_back(tmp);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::UpdateV(std::vector< std::vector< float > > & tmp)
   {

   V.push_back(tmp);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::UpdateT(std::vector< std::vector< float > > & tmp)
   {

   T.push_back(tmp);

   }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Solid::GetEP(int t, int nx, int ny)
   {

   return EP.at(t).at(nx).at(ny);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Solid::GetP(int t, int nx, int ny)
   {

   return P.at(t).at(nx).at(ny);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Solid::GetTheta_m(int t, int nx, int ny)
   {

   return Theta_m.at(t).at(nx).at(ny);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Solid::GetTheta(int t, int nx, int ny)
   {

   return Theta_m.at(t).at(nx).at(ny);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Solid::GetT(int t, int nx, int ny)
```

```
   {

   return T.at(t).at(nx).at(ny);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Solid::GetU(int t, int nx, int ny)
   {

   return U.at(t).at(nx).at(ny);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float Solid::GetV(int t, int nx, int ny)
   {

   return V.at(t).at(nx).at(ny);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::AddUsEval(std::vector< std::vector< float > > & v)
   {

   UsEvalues.push_back(v);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::AddVsEval(std::vector< std::vector< float > > & v)
   {

   VsEvalues.push_back(v);

   }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::Convert2MatrixU(std::vector< std::vector< float > > & In)
  {

  int i;
  int j;
  int t;
  std::vector< float > tmp;

  In.clear();
  for (t=0; t < Nt; t++)
    {
    tmp.clear();
    for (j=0; j < Ny; j++)
      for (i=0; i < Nx; i++)
        {
        tmp.push_back( GetU(t,i,j) );
        } // end i j for
    In.push_back(tmp);
    } // end t for

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void Solid::Convert2MatrixV(std::vector< std::vector< float > > & In)
  {

  int i;
  int j;
  int t;
  std::vector< float > tmp;

  In.clear();
  for (t=0; t < Nt; t++)
    {
    tmp.clear();
    for (j=0; j < Ny; j++)
      for (i=0; i < Nx; i++)
        {
        tmp.push_back( GetV(t,i,j) );
```

```
        } // end i j for
    In.push_back(tmp);
    } // end t for

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

SimVars::SimVars()
  {

  Avagadro = 6.022141793 * pow(10,23);
  Rydberg =  1.097373156 * pow(10,7);

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

SimVars::~SimVars()
  {

  }


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::Set2DVolVars()
  {

  delx = xdist / (Nxcells-1);
//std::cout << xdist << " " << Nxcells << "\n";
  dely = ydist / (Nycells-1);
//std::cout << ydist << " " << Nycells << "\n";
  Vol2DCell = delx * dely;
// Adjusted 2/21/2011 and may effect other calcs...  :(
  Gas.SetNx(Nxcells-1);
  Gas.SetNy(Nycells-1);
  Particle.SetNx(Nxcells-1);
  Particle.SetNy(Nycells-1);
  Ntimes = 0;
  }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::Setxdist(float f)
   {

   xdist = f;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::Setydist(float f)
   {

   ydist = f;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetX(int i)
   {
   // This function returns the midpoint of the cell.

   float x = (i - 0.5) * delx;

   return x;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetY(int i)
   {
   // This function returns the midpoint of the cell.

   float y = (i - 0.5) * dely;

   return y;

   }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::SetNx(int f)
   {

   Nxcells = f;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::SetNy(int f)
   {

   Nycells = f;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::SetTol(float t)
   {

   Tol = t;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetTol()
   {

   return Tol;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int SimVars::GetNx()
   {
```

```
    return Nxcells;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int SimVars::GetNy()
    {

    return Nycells;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int SimVars::GetNt()
    {

    return Ntimes;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetAvagadro()
    {

    return Avagadro;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetRydberg()
    {

    return Rydberg;

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::Get2DVol()
   {

   return Vol2DCell;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::Getdelx()
   {

   return delx;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::Getdely()
   {

   return dely;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::Setdelt(float dt)
   {

   delt = dt;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int SimVars::Getxindex(int i)
   {
```

```
    // This subroutine should return x cell Number based upon global cell
    // index

    int d = i - int(floor(i / GetNx()));

    return d;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int SimVars::Getyindex(int i)
    {
    // This subroutine should return y cell Number based upon global cell
    // index

    int d = int(floor(i/GetNx())) + 1;
    return d;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::SetDragC(float f)
    {

    DragC = f;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::SetDragD(float f)
    {

    DragD = f;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
void SimVars::StoreTimeStep(float timept, int ts)
   {

   TimePts.push_back( timept );
   Ntimes++;
//    std::cout << Ntimes << " time points. \n";
   // this should be at location timestep
   // TimePts.at(timestep) = timept;
   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetCalcdelt()
   {

   int s = TimePts.size();
   float dt = 0.0;
   if (s > 1)
      dt = TimePts.at(s-1) - TimePts.at(s-2);
   else
      std::cout << "Getdelt doesn't have enough time pts... \n";
   return dt;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::Getdelt()
   {

   return delt;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::StoreFluid(std::vector<float> & EP_g,
                     std::vector<float> & P_g,
                     std::vector<float> & U_g,
                     std::vector<float> & V_g,
                     std::vector<float> & W_g,
                     std::vector<float> & T_g,
```

```
                    std::vector<float> & X_g,
                    int timestep, int fi, float ts)
    {

    int i, j; //Don't need k for 3D
    std::vector<float> tmpEP;
    std::vector<float> tmpP;
    std::vector<float> tmpU;
    std::vector<float> tmpV;
    std::vector<float> tmpW;
    std::vector<float> tmpT;
    std::vector<float> tmpX;
    std::vector< std::vector<float> > tEP;
    std::vector< std::vector<float> > tP;
    std::vector< std::vector<float> > tU;
    std::vector< std::vector<float> > tV;
    std::vector< std::vector<float> > tW;
    std::vector< std::vector<float> > tT;
    std::vector< std::vector<float> > tX;
    int numX = GetNx();
    int numY = GetNy();
    float tdata;

    int count = 0;
    int index;
    int index1;
    int index2;
    int maxcount = (numX+2) * (numY+2);

    for (i=0;  i< numX+2; ++i)
       {
       for (j=0; j < numY+2; ++j)
          {
          if ((count < maxcount) &&
            !((i == 0) || (i >= numX+1) ||
            (j == 0) || (j >= numY+1))){

             //For Nodal Values
//             if ((i == 1) || (i == numX+1))
//                tmpEP.push_back(0.0);
//             else if (( j == 1) || (j == numY+1))
//                tmpEP.push_back(1.0);
//             else if ((i > 1) && (i <= numX))
//                {
//                tdata = (EP_g.at( i + (j*(numX+2)) ) +
```

```
//                      EP_g.at( i + ((j-1)*(numX+2)) ) +
//                      EP_g.at( (i-1) + (j*(numX+2)) ) +
//                      EP_g.at( (i-1) + ((j-1)*(numX+2)) ) ) / 4.0;
//                tmpEP.push_back( tdata );
//                }
//             if ((i == 1) || (i == numX+1))
//                tmpP.push_back(0.0);
//             else if (( j == 1) || (j == numY+1))
//                tmpP.push_back(0.0);
//             else if ((i > 1) && (i <= numX))
//                {
//                tdata = ((P_g.at( i + (j*(numX+2)))) +
//                      (P_g.at( i + ((j-1)*(numX+2)))) +
//                      (P_g.at( (i-1) + (j*(numX+2)))) +
//                      (P_g.at( (i-1) + ((j-1)*(numX+2)))) )
//                       / 4.0;
//                tmpP.push_back( tdata );
//                }
//             index1 = i + (j * (numX+2));
//             index2 = i + ((j-1)*(numX+2));
//std::cout << index1 << "=i,j=" << index2 << ": " << U_g.at(index1)
//         << " = Ui & Uj = " << U_g.at(index2) << "\n";
//             tdata = ( U_g.at(index1) + U_g.at(index2) ) / 2.0;
//             tmpU.push_back( tdata );
//             // Gas.U.at(timestep).at(i).at(j) = U_g.at(count);

//             index1 = i + (j * (numX+2));
//             index2 = (i - 1) + (j * (numX+2));
//         << " = Vi & Vj = " << V_g.at(index2) << "\n";
//             tdata = ( V_g.at(index1) + V_g.at(index2) ) / 2.0;
//             tdata = V_g.at(index);
//             tmpV.push_back( tdata );
//             // Gas.V.at(timestep).at(i).at(j) = V_g.at(count);
//             tdata = W_g.at(index);
//             tmpW.push_back( tdata );
//             // Gas.at(timestep).W.at(i).at(j) = W_g.at(count);
//             tdata = T_g.at(index);
//             tmpT.push_back( tdata );
//             // Gas.T.at(timestep).at(i).at(j) = T_g.at(count);
//             tdata = X_g.at(index);
//             tmpX.push_back( tdata );
//             // Gas.X.at(timestep).at(i).at(j) = X_g.at(count);

            // For Center of Cell Values
            index = i + (j*(numX+2));
```

```
//std::cout << "H1 \n";
// Suspect that Eg and Es are switched at t=0, but not neccessarily timestep
            if (ts == 0.0)
               {
               tdata = EP_g.at(index);
               tmpEP.push_back(1.0 - tdata);
               }
            else
               {
               tdata = EP_g.at(index);
               tmpEP.push_back(tdata);
               }
//std::cout << "H2 \n";

            tdata = P_g.at(index);
            tmpP.push_back(tdata);

            tdata = X_g.at(index);
            tmpX.push_back(tdata);

            tdata = T_g.at(index);
            tmpT.push_back(tdata);

            tdata = W_g.at(index);
            tmpW.push_back(tdata);

            index1 = i + (j * (numX+2));
            index2 = (i - 1) + (j * (numX+2));
            tdata = ( U_g.at(index1) + U_g.at(index2) ) / 2.0;
            tmpU.push_back(tdata);
//std::cout << index1 << "=i,j=" << index2 << ": " << U_g.at(index1)
//          << " and " << U_g.at(index2) << "\n";
//std::cout << "H3 \n";

            index1 = i + (j * (numX+2));
            index2 = i + ((j-1)*(numX+2));
            tdata = ( V_g.at(index1) + V_g.at(index2) ) / 2.0;
            tmpV.push_back(tdata);
//std::cout << index1 << "=i,j=" << index2 << ": " << V_g.at(index1)
//          << " and " << V_g.at(index2) << "\n";
//std::cout << index1 << "=i,j=" << index2 << ": " << P_g.at(index)
//          << "\n";
//std::cout << "H4 \n";

            }//end if count < maxcount
```

```
            count++;

            } // end for j
        // they should be the jth columns in the tmp vectors at the ith
        // row
        if (!((i == 0) || (i == numX+1)))
            {
            tEP.push_back(tmpEP);
            tP.push_back(tmpP);
            tU.push_back(tmpU);
            tV.push_back(tmpV);
            tW.push_back(tmpW);
            tT.push_back(tmpT);
            tX.push_back(tmpX);

            tmpEP.clear();
            tmpP.clear();
            tmpU.clear();
            tmpV.clear();
            tmpW.clear();
            tmpT.clear();
            tmpX.clear();
            } // end if ! bottom or top
        } // end for i
//std::cout << "size of tT first" << tT.at(0).size() << "\n";

    Gas.EP.push_back(tEP);
    Gas.P.push_back(tP);
    Gas.U.push_back(tU);
    Gas.V.push_back(tV);
    Gas.W.push_back(tW);
    Gas.T.push_back(tT);
    Gas.X.push_back(tX);
//std::cout << "size of Gas.T " << Gas.T.at(0).at(49).at(74) << "\n";

    }  // end StoreFluid

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::StoreSolid(std::vector<float> & EP_g,
                         std::vector<float> & P_star,
                         std::vector<float> & U_s,
                         std::vector<float> & V_s,
                         std::vector<float> & W_s,
```

```
                    std::vector<float> & ROP_s,
                    std::vector<float> & T_s,
                    std::vector<float> & X_s,
                    std::vector<float> & Theta_m,
                    int timestep, int si, float ts)
{

int i, j; //Don't need k for 3D
float tdata;

std::vector<float> tmpEP;
std::vector<float> tmpP;
std::vector<float> tmpROP;
std::vector<float> tmpTheta_m;
std::vector<float> tmpU;
std::vector<float> tmpV;
std::vector<float> tmpW;
std::vector<float> tmpT;
std::vector<float> tmpX;

std::vector< std::vector<float> > tEP;
std::vector< std::vector<float> > tP;
std::vector< std::vector<float> > tR;
std::vector< std::vector<float> > tM;
std::vector< std::vector<float> > tU;
std::vector< std::vector<float> > tV;
std::vector< std::vector<float> > tW;
std::vector< std::vector<float> > tT;
std::vector< std::vector<float> > tX;

int count = 0;
int index = 0;
int index1;
int index2;
int numX = GetNx();
int numY = GetNy();
int maxcount = (numX+2) * (numY+2);

for (i=0; i < numX+2; ++i)
   {
   for (j=0; j < numY+2; ++j)
      {
      if ((count < maxcount) &&
         (i != 0) && (i != numX +1) &&
         (j != 0) && (j != numY +1))
```

```
            {
//              if ((i == 1) || (i == numX+1))
//                 tmpEP.push_back(0.0);
//              else if (( j == 1) || (j == numY+1))
//                 tmpEP.push_back(0.0);
//              else if ((i > 1) && (i <= numX))
//                 {
//                 tdata = ((1.0 - EP_g.at( i + (j*(numX+2)))) +
//                         (1.0 - EP_g.at( i + ((j-1)*(numX+2)))) +
//                         (1.0 - EP_g.at( (i-1) + (j*(numX+2)))) +
//                         (1.0 - EP_g.at( (i-1) + ((j-1)*(numX+2)))) )
//                          / 4.0;
//                 tmpEP.push_back( tdata );
//                 }
//std::cout << 1.0 - EP_g.at(index) << "=EPs being stored at " << index << "\n";
            // Particle.EP.at(timestep).at(i).at(j) = 1 - EP_g.at(count);

//              if ((i == 1) || (i == numX+1))
//                 tmpP.push_back(0.0);
//              else if (( j == 1) || (j == numY+1))
//                 tmpP.push_back(0.0);
//              else if ((i > 1) && (i <= numX))
//                 {
//                 tdata = ((P_star.at( i + (j*(numX+2)))) +
//                         (P_star.at( i + ((j-1)*(numX+2)))) +
//                         (P_star.at( (i-1) + (j*(numX+2)))) +
//                         (P_star.at( (i-1) + ((j-1)*(numX+2)))) )
//                          / 4.0;
//                 tmpP.push_back( tdata );
//                 }
            // Particle.Theta_m.at(timestep).at(i).at(j) =
            //                             Theta_m.at(count);

//std::cout << U_s.at(index) << "=Us being stored at " << index << "\n";

//              index1 = i + (j * (numX+2));
//              index2 = i + ((j-1)*(numX+2));
//              tdata = ( U_s.at(index1) + U_s.at(index2) ) / 2.0;
//              tmpU.push_back( tdata );
//              // Particle.U.at(timestep).at(i).at(j) = U_s.at(count);
//std::cout << V_s.at(index) << "=Vs being stored at " << index << "\n";

//              index1 = i + (j * (numX+2));
//              index2 = (i - 1) + (j * (numX+2));
//              tdata = ( V_s.at(index1) + V_s.at(index2) ) / 2.0;
```

```
//              tmpV.push_back( tdata );
//              // Particle.V.at(timestep).at(i).at(j) = V_s.at(count);

//std::cout << 1.0 - EP_g.at(index) << " being stored \n";

            // Center of Cell Data
            index = i + (j * (numX+2));

            index1 = i + (j * (numX+2));
            index2 = (i - 1) + (j * (numX+2));
            tdata = ( U_s.at(index1) + U_s.at(index2) ) / 2.0;
            tmpU.push_back(tdata);

            index1 = i + (j * (numX+2));
            index2 = i + ((j-1)*(numX+2));
            tdata = ( V_s.at(index1) + V_s.at(index2) ) / 2.0;
            tmpV.push_back(tdata);

            tdata = W_s.at(index);
            tmpW.push_back( tdata );
            // Particle.W.at(timestep).at(i).at(j) = W_s.at(count);

            tdata = T_s.at(index);
            tmpT.push_back( tdata );
            // Particle.T.at(timestep).at(i).at(j) = T_s.at(count);

            tdata = X_s.at(index);
            tmpX.push_back( tdata );
            // Particle.X.at(timestep).at(i).at(j) = X_s.at(count);

            if (ts != 0.0)
               tdata = 1.0 - EP_g.at(index);
            else
               tdata = EP_g.at(index);
            tmpEP.push_back( tdata );

            tdata = P_star.at(index);
            tmpP.push_back( tdata );
            // Gas.P_star.at(timestep).at(i).at(j) = P_star.at(count);

            tdata = ROP_s.at(index);
            tmpROP.push_back( tdata );
            //Particle.ROP.at(timestep).at(i).at(j) = ROP_s.at(count);

            tdata = Theta_m.at(index);
```

```
            tmpTheta_m.push_back( tdata );
            }
          count++;
//std::cout << "size of tmpT " << tmpT.size() << "\n";
          } // end for j

        // should be the jth columns in the tmp vectors at the ith row
        if ((i != 0) && (i != numX+1))
          {
          tEP.push_back(tmpEP);
          tP.push_back(tmpP);
          tR.push_back(tmpROP);
          tM.push_back(tmpTheta_m);
          tU.push_back(tmpU);
          tV.push_back(tmpV);
          tW.push_back(tmpW);
          tT.push_back(tmpT);
          tX.push_back(tmpX);

//std::cout << "size of tT " << tT.size() << "\n";
          tmpEP.clear();
          tmpP.clear();
          tmpROP.clear();
          tmpTheta_m.clear();
          tmpU.clear();
          tmpV.clear();
          tmpW.clear();
          tmpT.clear();
          tmpX.clear();
          } // end if ! top or bottom
        } // end for i
//std::cout << "size of tT first" << tT.at(0).size() << "\n";

    Particle.EP.push_back(tEP);
    Particle.P.push_back(tP);
    Particle.ROP.push_back(tR);
    Particle.Theta_m.push_back(tM);
    Particle.U.push_back(tU);
    Particle.V.push_back(tV);
    Particle.W.push_back(tW);
    Particle.T.push_back(tT);
    Particle.X.push_back(tX);
//std::cout << "size of Gas.T " << Gas.T.at(0).at(49).at(74) << "\n";

    } // end StoreSolid
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::SetTimes(int i)
   {

   Gas.SetNt(i);
   Particle.SetNt(i);
   Ntimes = i;
   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetTimept(int i)
   {

   return TimePts.at(i);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::LastTimept()
   {

   return TimePts.back();

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetGasXVel(int t, int i, int j)
   {

   return Gas.U.at(t).at(i).at(j);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
float SimVars::GetGasYVel(int t, int i, int j)
   {

   return Gas.V.at(t).at(i).at(j);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetParticleXVel(int t, int i, int j)
   {

   return Particle.U.at(t).at(i).at(j);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetParticleYVel(int t, int i, int j)
   {

   return Particle.V.at(t).at(i).at(j);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::SetResCoeff(float val)
   {

   erescoeff = val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetE()
   {

   return erescoeff;
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::SetAngIntFric(float val)
    {

    AngIntFric = val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::SetPackedBedEP(float val)
    {

    pb_EP = val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetTemp(int t, int nx, int ny)
    {

    return Particle.GetTheta_m(t, nx,ny);

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::FindEvalue(float a2, float a1, float a0, float oeval)
    {

    double r1;
    double r2;
    double radical;
    float evalue;

    radical = pow(a1,2) - (4.0 * a2 * a0);
    if (radical < 0.0)
```

```
        {
        r1 = 0.0;
        r2 = 0.0;
        }
    else
        {
        r1 = ((-1.0 * a1) + sqrt(radical)) / (2.0 * a2);
        r2 = ((-1.0 * a1) - sqrt(radical)) / (2.0 * a2);
        }

    if (fabs(oeval - r1) > fabs(oeval - r2))
        evalue = r2;
    else if (fabs(oeval - r2) > fabs(oeval - r1))
        evalue = r1;
    else if (oeval > 0 && r1 > 0)
        evalue = r1;
    else if (oeval < 0 && r1 < 0)
        evalue = r1;
    else
        evalue = r2;

    return evalue;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::Find0Evalues()
    {

    float basis;
    int i, j;
    std::vector< float > tug;
    std::vector< float > tus;
    std::vector< float > tvg;
    std::vector< float > tvs;
    std::vector< std::vector< float > > ttug;
    std::vector< std::vector< float > > ttus;
    std::vector< std::vector< float > > ttvg;
    std::vector< std::vector< float > > ttvs;


    for (i=0; i < GetNx(); i++)
        {
```

```
        for (j=0; j < GetNy(); j++)
           {
           basis = GetBLB((i+1)*Getdelx(),
                       (j+1)*Getdely(),
                       (i+0.5)*Getdelx(),
                       (j+0.5)*Getdely());
           if (basis == 0)
              {
              tug.push_back(0);
              tus.push_back(0);
              tvg.push_back(0);
              tvs.push_back(0);
              }
           else
              {
              tug.push_back( Gas.GetU(0,i,j) / basis );
//std::cout<< "x=" << i+1 << ", j=" << j+1 << "\n";
//std::cout<< "Ug0=" << (Gas.GetU(i,j) / basis);
              tus.push_back( Particle.GetU(0,i,j) / basis );
              tvg.push_back( Gas.GetV(0,i,j) / basis );
              tvs.push_back( Particle.GetV(0,i,j) / basis );
//std::cout<< ", Vg0=" << (Gas.GetV(i,j) / basis) << "\n";
//std::cout<< "Us0=" << (Particle.GetU(i,j) / basis);
//std::cout<< ", Vs0=" << (Particle.GetV(i,j) / basis) << "\n";
              }
           } // end for j
        ttug.push_back( tug );
        ttus.push_back( tus );
        ttvg.push_back( tvg );
        ttvs.push_back( tvs );
        } // end for i
     Gas.AddUgEval(ttug);
     Gas.AddVgEval(ttvg);
     Particle.AddUsEval(ttus);
     Particle.AddVsEval(ttvs);
//std::cout << UgEvalues.at(0).at(49).at(74) << " are the evals \n";
     }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::ExtractBBC(std::vector< double > & v)
     {

     int e1c = (int) Gas.slX * Nxcells;
```

```
   int e2c = (int) Gas.srX * Nxcells;
   int i;

   v.clear();

   for (i = 0; i < GetNx(); i++)
      {
      if ((i >= e1c) && (i <= e2c))
         v.push_back(Gas.sbV);
      else
         v.push_back(Gas.bV);
      }

   }  //end ExtractBBC

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void SimVars::FormBBC(double & b, double & j, double & e1,
                double & e2, std::vector< double > & v)
   {

   int e1c = (int) e1 * Nxcells;
   int e2c = (int) e2 * Nxcells;
   int i;

   v.clear();

   for (i = 0; i < GetNx(); i++)
      {
      if ((i >= e1c) && (i <= e2c))
         v.push_back(j);
      else
         v.push_back(b);
      }

   }  //end FormBBC

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetBLB(float xn, float yn, float x, float y)
   {

   float val;
```

```
   val = (1.0/8.0) *
         (1.0 + (x/xn)) *
         (1.0 + (y/yn)) *
         (2.0 + ( (x/xn) * (1.0 - (x/xn)) ) +
                ( (y/yn) * (1.0 + (y/yn)) ) );

   //Direct coding - I think....
   val = 0.25;

   return val;

   }  //end GetBiLinearBasis

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetFBLB()
   {

   float val;

   val = 0.25;

   return val;

   }  //end GetFBLB/

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetBLBx(float xn, float yn, float x, float y, float dy)
   {

   float val;

   val = (-1.0 / 8.0) * yn * (1.0 + (x/xn)) * pow((1.0 + (y/yn)), 2) *
                          (1.0 - (y/yn));

   //Direct coding - I think....
   val = ( - dy / 16.0 );

   return val;

   }  //end GetBiLinearBasisx
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float SimVars::GetBLBy(float xn, float yn, float x, float y, float dx)
   {

   float val;

   val = (1.0 / 8.0) * xn * (1.0 + (y/yn)) * pow((1.0 + (x/xn)), 2) *
                        (1.0 - (x/xn));

   //Direct coding - I think....
   val = ( dx / 16.0 );

   return val;

   }  //end GetBiLinearBasisy

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    PODData.h
//    Created by Stephanie R. Beck Roth
//    Last Modified:   01/11/2011
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#ifndef PODDATA_H
#define PODDATA_H

#include <iostream>
#include <fstream>
#include <string>
#include <math.h>
#include <vector>

class PODData
   {
   private:

      int vectordims;

      int NUg;
      int NVg;
      int NUs;
      int NVs;
      int NEg;
      int NPg;
      int NPs;
      int NTs;

      int Nt;
      int Nx;
      int Ny;

      double dx;
      double dy;
      double dt;

   public:

      // These contain the POD coefficients (a_k) which is a function
      // of time.  First component is time step (Nmax = Nt) and the
      // second component is the vector of a_1 to a_m (Nmax = m which
```

```
// the number of POD coefficients in the phase specific model)

std::vector< std::vector< double > > PODsParticleU;
std::vector< std::vector< double > > PODsParticleV;
std::vector< std::vector< double > > PODsGasU;
std::vector< std::vector< double > > PODsGasV;

std::vector< std::vector< double > > PODsParticleP;
std::vector< std::vector< double > > PODsParticleT;
std::vector< std::vector< double > > PODsGasP;
std::vector< std::vector< double > > PODsGasF;

std::vector< std::vector< double > > UgAvgVel;
std::vector< std::vector< double > > VgAvgVel;
std::vector< std::vector< double > > UsAvgVel;
std::vector< std::vector< double > > VsAvgVel;

std::vector< std::vector< double > > FgAvg;
std::vector< std::vector< double > > PgAvg;
std::vector< std::vector< double > > PsAvg;
std::vector< std::vector< double > > TsAvg;

//Phis run over: (y and x) then basis in column vector
std::vector< std::vector< double > > GPhiuPOD;
std::vector< std::vector< double > > GPhivPOD;
std::vector< std::vector< double > > PPhiuPOD;
std::vector< std::vector< double > > PPhivPOD;

std::vector< std::vector< double > > TsPhiPOD;
std::vector< std::vector< double > > PsPhiPOD;
std::vector< std::vector< double > > PgPhiPOD;
std::vector< std::vector< double > > FgPhiPOD;

int GetNUg();
int GetNVg();
int GetNUs();
int GetNVs();
int GetNEg();
int GetNPg();
int GetNPs();
int GetNTs();

int GetNt();
int GetNx();
int GetNy();
```

```
        double Getdx();
        double Getdy();
        double Getdt();

        PODData();
        ~PODData();

        void InitPODData(int x, int y, int t,
                        double delx, double dely, double delt);

        void SetNUG(int n);
        void SetNVG(int n);
        void SetNUS(int n);
        void SetNVS(int n);

        void SetNEG(int n);
        void SetNPG(int n);
        void SetNPS(int n);
        void SetNTS(int n);

    };

#endif
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    PODData.cpp
//    Created by Stephanie R. Beck Roth
//    Last Modified:   10/22/2010
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#include <iostream>
#include <fstream>
#include <string>
#include <math.h>
#include <vector>
#include "PODData.h"

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

PODData::PODData()

   {

   PODsGasU.clear();
   PODsGasV.clear();
   PODsGasP.clear();
   PODsGasF.clear();

   PODsParticleU.clear();
   PODsParticleV.clear();
   PODsParticleP.clear();
   PODsParticleT.clear();

   UgAvgVel.clear();
   VgAvgVel.clear();
   UsAvgVel.clear();
   VsAvgVel.clear();

   PgAvg.clear();
   PsAvg.clear();
   TsAvg.clear();
   FgAvg.clear();

   GPhiuPOD.clear();
   GPhivPOD.clear();
   PPhiuPOD.clear();
```

```
    PPhivPOD.clear();

    TsPhiPOD.clear();
    PsPhiPOD.clear();
    PgPhiPOD.clear();
    FgPhiPOD.clear();

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

PODData::~PODData()

    {

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int PODData::GetNUg()

    {

    return NUg;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int PODData::GetNVg()

    {

    return NVg;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int PODData::GetNUs()
```

```
   {
   return NUs;
   }
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
int PODData::GetNVs()
   {
   return NVs;
   }
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
int PODData::GetNEg()
   {
   return NEg;
   }
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
int PODData::GetNPg()
   {
   return NPg;
   }
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
int PODData::GetNPs()
   {
```

```
    return NPs;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int PODData::GetNTs()

    {

    return NTs;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int PODData::GetNt()

    {

    return Nt;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int PODData::GetNx()

    {

    return Nx;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int PODData::GetNy()

    {
```

```
    return Ny;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double PODData::Getdx()

    {

    return dx;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double PODData::Getdy()

    {

    return dy;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double PODData::Getdt()

    {

    return dt;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODData::InitPODData(int x, int y, int t,
                          double delx, double dely, double delt)

    {
```

```
    Nx = x;
    Ny = y;
    Nt = t;

    dx = delx;
    dy = dely;
    dt = delt;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODData::SetNUG(int n)

    {

    NUg = n;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODData::SetNVG(int n)

    {

    NVg = n;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODData::SetNUS(int n)

    {

    NUs = n;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
void PODData::SetNVS(int n)

   {

   NVs = n;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODData::SetNEG(int n)

   {

   NEg = n;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODData::SetNPG(int n)

   {

   NPg = n;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODData::SetNPS(int n)

   {

   NPs = n;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
void PODData::SetNTS(int n)

    {

    NTs = n;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    PODSolver.h
//    Created by Stephanie R. Beck Roth
//    Last Modified:   12/15/2010
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#ifndef PODSOLVER_H
#define PODSOLVER_H

#include "mpi.h"
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <sstream>
#include "FEMSolver.h"
#include "PODData.h"

class PODSolver
   {

   private:

      PODData POD;

      double epsilon;

      int Nt;
      int Nx;
      int Ny;

      double dx;
      double dy;

      std::vector< double > energy; // energy threshold (1.0 - epsilon model error)

      std::vector< std::vector<
               std::vector< bool > > > UseEVal;
      bool SetUseEval( FEMSolver & F);

      std::vector< std::vector< double > > CovG;
      std::vector< std::vector< double > > CovS;
      std::vector< std::vector< double > > CovFg;
```

```
        std::vector< std::vector< double > > CovPg;
        std::vector< std::vector< double > > CovPs;
        std::vector< std::vector< double > > CovTs;

    //Phis run over: y then x then time comp then velocity dir component
    // The following commented section is NOT valid data

//    std::vector< std::vector< std::vector< std::vector< double > > > > GPhi;
//    std::vector< std::vector< std::vector< std::vector< double > > > > PPhi;

//    std::vector< std::vector< std::vector< double > > > GPhiu;
//    std::vector< std::vector< std::vector< double > > > GPhiv;
//    std::vector< std::vector< std::vector< double > > > PPhiu;
//    std::vector< std::vector< std::vector< double > > > PPhiv;

//    std::vector< std::vector< std::vector< double > > > TsPhi;
//    std::vector< std::vector< std::vector< double > > > PsPhi;
//    std::vector< std::vector< std::vector< double > > > PgPhi;
//    std::vector< std::vector< std::vector< double > > > FgPhi;

        // Approximations to U and V for fluid and solid velocities
        std::vector< std::vector< std::vector< double > > > UgR;
        std::vector< std::vector< std::vector< double > > > VgR;
        std::vector< std::vector< std::vector< double > > > UsR;
        std::vector< std::vector< std::vector< double > > > VsR;

        std::vector< std::vector< std::vector< double > > > PsR;
        std::vector< std::vector< std::vector< double > > > TsR;
        std::vector< std::vector< std::vector< double > > > PgR;
        std::vector< std::vector< std::vector< double > > > FgR;

        // Reconstruction Error for fluid and solid velocities
        // stored as time y and x
        std::vector< std::vector< std::vector< double > > > ErrorUg;
        std::vector< std::vector< std::vector< double > > > ErrorVg;
        std::vector< std::vector< std::vector< double > > > ErrorUs;
        std::vector< std::vector< std::vector< double > > > ErrorVs;

        std::vector< std::vector< std::vector< double > > > ErrorTs;
        std::vector< std::vector< std::vector< double > > > ErrorPs;
        std::vector< std::vector< std::vector< double > > > ErrorPg;
        std::vector< std::vector< std::vector< double > > > ErrorFg;

        std::vector< double > GasEvaluesC;
        std::vector< std::vector< double > > GasEvectorsC;
```

```
std::vector< double > ParticleEvaluesC;
std::vector< std::vector< double > > ParticleEvectorsC;

std::vector< double > GasFEvaluesC;
std::vector< std::vector< double > > GasFEvectorsC;

std::vector< double > GasPEvaluesC;
std::vector< std::vector< double > > GasPEvectorsC;

std::vector< double > ParticlePEvaluesC;
std::vector< std::vector< double > > ParticlePEvectorsC;

std::vector< double > ParticleTEvaluesC;
std::vector< std::vector< double > > ParticleTEvectorsC;

std::vector< double > GasEvalues;
std::vector< std::vector< double > > GasEvectors;

std::vector< double > ParticleEvalues;
std::vector< std::vector< double > > ParticleEvectors;

std::vector< double > GasFEvalues;
std::vector< std::vector< double > > GasFEvectors;

std::vector< double > GasPEvalues;
std::vector< std::vector< double > > GasPEvectors;

std::vector< double > ParticlePEvalues;
std::vector< std::vector< double > > ParticlePEvectors;

std::vector< double > ParticleTEvalues;
std::vector< std::vector< double > > ParticleTEvectors;

void copy2vec (std::vector< std::vector< double > > & M,
        int dim, int i, std::vector< double > & V);

void InputGUBasis(int Nbasis, std::string ln);
void InputGVBasis(int Nbasis, std::string ln);
void InputPUBasis(int Nbasis, std::string ln);
void InputPVBasis(int Nbasis, std::string ln);
void InputGPBasis(int Nbasis, std::string ln);
void InputGFBasis(int Nbasis, std::string ln);
void InputPPBasis(int Nbasis, std::string ln);
void InputPTBasis(int Nbasis, std::string ln);
```

```
    void OutputGUBasis(int Nbasis);
    void OutputGVBasis(int Nbasis);
    void OutputPUBasis(int Nbasis);
    void OutputPVBasis(int Nbasis);
    void OutputGPBasis(int Nbasis);
    void OutputGFBasis(int Nbasis);
    void OutputPPBasis(int Nbasis);
    void OutputPTBasis(int Nbasis);

    void AdjustSingVals(std::vector< double > & S, int Ns);
    void FormGub(std::vector< double > & V, int t, SimVars & Data);
    void FormGvb(std::vector< double > & V, int t, SimVars & Data);
    void FormPub(std::vector< double > & V, int t, SimVars & Data);
    void FormPvb(std::vector< double > & V, int t, SimVars & Data);
    void FormGpb(std::vector< double > & V, int t, SimVars & Data);
    void FormGfb(std::vector< double > & V, int t, SimVars & Data);
    void FormPsb(std::vector< double > & V, int t, SimVars & Data);
    void FormTsb(std::vector< double > & V, int t, SimVars & Data);

    void XChangeBasisMPI(int & NGU, int & NGV, int & NPU,
                         int & NPV, int & NGP, int & NGF,
                         int & NPP, int & NPT, int Id, int NIds);

    void GetMeanPODs(std::vector< std::vector< double > > & Ug,
                     std::vector< std::vector< double > > & Vg,
                     std::vector< std::vector< double > > & Us,
                     std::vector< std::vector< double > > & Vs,
                     std::vector< std::vector< double > > & Eg,
                     std::vector< std::vector< double > > & Pg,
                     std::vector< std::vector< double > > & Ps,
                     std::vector< std::vector< double > > & Ts);


public:

    PODSolver();
    ~PODSolver();

    double Min(double a, double b);
    double Max(double a, double b);
    int GetNx();
    int GetNy();
    int GetNt();
```

```
bool InterpPOD(std::vector< std::vector< double > > & Ug,
               std::vector< std::vector< double > > & Vg,
               std::vector< std::vector< double > > & Us,
               std::vector< std::vector< double > > & Vs,
               std::vector< std::vector< double > > & Eg,
               std::vector< std::vector< double > > & Pg,
               std::vector< std::vector< double > > & Ps,
               std::vector< std::vector< double > > & Ts,
               int Nx, int Ny, int Nt, int Id, int NIds,
               std::vector< double > & e, SimVars & Data);

bool FormSVDPOD(SimVars & Data, std::vector < double > & e,
                int Id, int NIds);
bool LoadPOD(SimVars & Data, int Id, int NId,
             std::string & ln);
bool  PartOfPOD (int x, int y, int t);
bool  FormCorrMat(SimVars & F);
void  SetNs( int t, int x, int y, double idx, double idy);
void RankEm(std::vector< double > & wtr, std::vector< int > & r);
double CalcEnergySVD(std::vector< double > & Sigma, int & N,
                     int index);
double CalcEnergy(std::vector< double > & ev,
                  std::vector< int > & rank,
                  std::vector< int > & use, int & i,
                  int index);
void  GetMeanVars(SimVars & SV);
void  GetCorrMatrices(SimVars & SV);
void  HRed2Tri(std::vector< std::vector< double > > & A,
               std::vector< double > & D,
               std::vector< double > & OD, int n);
double pythag(double a, double b);
void  GetEigenVs(std::vector< double > & D,
                 std::vector< double > & OD,
                 int n, std::vector< double > & EValues,
                 std::vector< std::vector< double > > & EVectors);
void svd(std::vector< std::vector< double > > & A,
         std::vector< std::vector< double > > & U,
         std::vector< std::vector< double > > & V,
         std::vector< double > & Sigma, int m, int n);

void Solver(std::vector< std::vector< double > > & U,
         std::vector< std::vector< double > > & V,
         std::vector< double > & Sigma, int m, int n,
         std::vector< double > & X);
void Solve4coefs(std::vector< std::vector< double > > & U,
```

```
                    std::vector< std::vector< double > > & V,
                    std::vector< double > & W,
                    std::vector< double > & b,
                    int Npts, int mbasis,
                    std::vector< double > & X);

    void FormReconVars(int NbGU, int NbGV, int NbPU, int NbPV,
                       int NbGP, int NbGF, int NbPP, int NbPT,
                       int Id, int NIds);
    void ApproxErrorSplit(SimVars & Data, int NId, int NIds);
    void InputBasis(int NGV, int NPV, int NGP,
                    int NPP, int NPT, int NGF,
                    int Id, int NIds, std::string ln);
    void OutputPODsSplit(int NGU, int NGV, int NPU, int NPV,
                         int NGP, int NGF, int NPP, int NPT,
                         int Id, int NIds);
    void InputPODs( int NGU, int NGV, int NPU, int NPV,
                    int NGP, int NGF, int NPP, int NPT,
                    std::string ln);
    void OutputSingVals(std::vector< double > V, int index);
    void OutputMeans();
    void OutputSolution(int Id, int NIds);
    void OutputMFIX(SimVars & SV);
    void InputMeans(std::string ln);
    void OutputKey(int NGU, int NGV, int NPU, int NPV,
                   int NGP, int NGF, int NPP, int NPT,
                   int Id, int NIds);
    void InputKey(int & NGU, int & NGV, int & NPU, int & NPV,
                  int & NGP, int & NGF, int & NPP, int & NPT,
                  std::string & ln);
    void OutputErrorSplit(int Id, int NIds);
    void GetPODData(PODData & P);
    void GetSol(int & all, std::string & ln, bool load,
                std::vector < std::vector< double > > & Ug,
                std::vector < std::vector< double > > & Vg,
                std::vector < std::vector< double > > & Us,
                std::vector < std::vector< double > > & Vs,
                std::vector < std::vector< double > > & Eg,
                std::vector < std::vector< double > > & Pg,
                std::vector < std::vector< double > > & Ps,
                std::vector < std::vector< double > > & Ts);

};

#endif
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    PODSolver.cpp
//    Created by Stephanie R. Beck Roth
//    Last Modified:   01/09/2011
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#include "mpi.h"
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <cmath>
#include <sstream>
#include "stdio.h"
#include "FEMSolver.h"
#include "PODData.h"
#include "PODSolver.h"

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

PODSolver::PODSolver()
   {
   epsilon = 1.0e-14;
   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

PODSolver::~PODSolver()
   {

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double PODSolver::Max(double a, double b)
   {

   if ( a < b)
      return b;
   else
```

```
        return a;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double PODSolver::Min(double a, double b)
    {

    if ( a > b)
        return b;
    else
        return a;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

bool PODSolver::SetUseEval( FEMSolver & F)
    {

    bool good = false;

    return good;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

bool PODSolver::PartOfPOD (int x, int y, int t)
    {

    bool good = false;

    return good;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

bool PODSolver::FormCorrMat( SimVars & F)
```

```
   {

   bool good = false;

   return good;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::SetNs(int t, int x, int y, double idx, double idy)

   {

   Nt = t;
   Nx = x;
   Ny = y;

   dx = idx;
   dy = idy;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::RankEm(std::vector< double > & wtr,
                 std::vector< int > & r)

   {
   // r returns the index of the elements in wtr
   // which rank lower in order


   int dim = wtr.size();
   std::vector< int > t(dim, dim);
   int rank;
   int i, j;

   for (i = 0; i < dim; i++)
      {
      rank = 0;
      for (j = 0; j < dim; j++)
         {
```

```
        if ( wtr.at(j) > wtr.at(i) )
          rank++;
        //if problems look below at this else if statement
        //checked and then changed to break the tie of two == values
        //re-checked but quickly only
        else if ((wtr.at(j) == wtr.at(i)) && (i > j))
          rank++;
        } // end for j;
      t.at(rank) = i;
      } // end for i
    r = t;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double PODSolver::CalcEnergy(std::vector< double > & ev,
                        std::vector< int > & rank,
                        std::vector< int > & use,
                        int & i, int index)
    {

    double calce = 0.0;
    double d = 0.0;
    double n = 0.0;

    int dim;


    dim = ev.size();
    std::vector< int > t(dim, 0);
    use = t;

    // Get Denominator (sum of ALL evalues)
    for (i=0; i < dim; i++)
      d = d + ev.at(i);

    i = 0;
    n = 0.0;
    while (calce <= energy.at(index))
      {
      n = n + ev.at(rank.at(i));
      use.at(rank.at(i)) = 1;
      calce = n / d;
```

```
        i++;
        }

    return calce;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::GetMeanVars(SimVars & SV)
    {

    double sumUg;
    double sumVg;
    double sumUs;
    double sumVs;

    double sumPs;
    double sumPg;
    double sumFg;
    double sumTs;

//    double BLB;
//    std::cout << Gce << " captures the energy of the Gas phase.\n";


    int i;
    int j;
    int t;

    std::vector< double > tUg;
    std::vector< double > tVg;
    std::vector< double > tUs;
    std::vector< double > tVs;

    std::vector< double > tPs;
    std::vector< double > tTs;
    std::vector< double > tPg;
    std::vector< double > tFg;

    POD.UgAvgVel.clear();
    POD.VgAvgVel.clear();
    POD.UsAvgVel.clear();
    POD.VsAvgVel.clear();
```

```
POD.FgAvg.clear();
POD.PgAvg.clear();
POD.PsAvg.clear();
POD.TsAvg.clear();

// Doing this outside of the positional loop to save time
// Don't need this when using real velocities...
//    BLB = F.GetFBLB();

for (i=0; i < Nx; i++)
   {

   tUg.clear();
   tVg.clear();
   tUs.clear();
   tVs.clear();

   tFg.clear();
   tPg.clear();
   tPs.clear();
   tTs.clear();

   for (j=0; j < Ny; j++)
      {

      sumUg = 0.0;
      sumVg = 0.0;
      sumUs = 0.0;
      sumVs = 0.0;

      sumFg = 0.0;
      sumPg = 0.0;
      sumPs = 0.0;
      sumTs = 0.0;

      for (t=0; t< Nt; t++)
         {
         sumUg = sumUg + SV.Gas.GetU(t,i,j);
         sumVg = sumVg + SV.Gas.GetV(t,i,j);
         sumUs = sumUs + SV.Particle.GetU(t,i,j);
         sumVs = sumVs + SV.Particle.GetV(t,i,j);

         sumFg = sumFg + SV.Gas.GetEP(t,i,j);
         sumPg = sumPg + SV.Gas.GetP(t,i,j);
```

```
            sumPs = sumPs + SV.Particle.GetP(t,i,j);
            sumTs = sumTs + SV.Particle.GetTheta(t,i,j);

            } // end t

         tUg.push_back(sumUg/double(Nt));
         tVg.push_back(sumVg/double(Nt));
         tUs.push_back(sumUs/double(Nt));
         tVs.push_back(sumVs/double(Nt));

         tFg.push_back(sumFg/double(Nt));
         tPg.push_back(sumPg/double(Nt));
         tPs.push_back(sumPs/double(Nt));
         tTs.push_back(sumTs/double(Nt));

         } // end j

      POD.UgAvgVel.push_back(tUg);
      POD.VgAvgVel.push_back(tVg);
      POD.UsAvgVel.push_back(tUs);
      POD.VsAvgVel.push_back(tVs);

      POD.FgAvg.push_back(tFg);
      POD.PgAvg.push_back(tPg);
      POD.PsAvg.push_back(tPs);
      POD.TsAvg.push_back(tTs);

      } // end i

   std::cout << "Size of AvgVel = " << POD.UgAvgVel.size()
          << " and size of x = " << tUg.size() << "\n";

   }


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::GetCorrMatrices(SimVars & SV)
   {

   double Cijs;
   double Cijg;

   double CijFg;
```

```
    double CijPg;
    double CijPs;
    double CijTs;

//    double BLB;

    int t1;
    int t2;
    int i;
    int j;

    std::vector< double > tmpS;
    std::vector< double > tmpG;

    std::vector< double > tmpGF;
    std::vector< double > tmpGP;
    std::vector< double > tmpSP;
    std::vector< double > tmpST;

    // Doing this outside of the positional loop to save time
    // Don't need this when using real velocities...
    //    BLB = F.GetFBLB();

    CovG.clear();
    CovS.clear();

    CovFg.clear();
    CovPg.clear();
    CovPs.clear();
    CovTs.clear();

    // Recognize these matrices are symmetric

    for (t1=0; t1 < Nt; t1++)
       {

       tmpS.clear();
       tmpG.clear();

       tmpGF.clear();
       tmpGP.clear();
       tmpSP.clear();
       tmpST.clear();

       for (t2=0; t2 < Nt; t2++)
```

```
{

if (t1 <= t2)
  {

  Cijg = 0.0;
  Cijs = 0.0;

  CijFg = 0.0;
  CijPg = 0.0;
  CijPs = 0.0;
  CijTs = 0.0;

   // Loop through the positional vectors
   for (j=0; j < Ny; j++)
      for (i=0; i < Nx; i++)
         {
         Cijg = ((SV.Gas.GetU(t1,i,j) -
                  POD.UgAvgVel.at(i).at(j) ) *
                 (SV.Gas.GetU(t2,i,j) -
                  POD.UgAvgVel.at(i).at(j)) ) +
                ((SV.Gas.GetV(t1,i,j) -
                  POD.VgAvgVel.at(i).at(j) ) *
                 (SV.Gas.GetV(t2,i,j) -
                  POD.VgAvgVel.at(i).at(j)) )+
                 Cijg;

         Cijs = ((SV.Particle.GetU(t1,i,j) -
                  POD.UsAvgVel.at(i).at(j) )*
                 (SV.Particle.GetU(t2,i,j) -
                  POD.UsAvgVel.at(i).at(j))) +
                 ((SV.Particle.GetV(t1,i,j) -
                  POD.VsAvgVel.at(i).at(j) ) *
                 (SV.Particle.GetV(t2,i,j) -
                  POD.VsAvgVel.at(i).at(j))) +
                 Cijs;

         CijPg = (SV.Gas.GetEP(t1,i,j) -
                  POD.FgAvg.at(i).at(j) ) *
                 (SV.Gas.GetEP(t2,i,j) -
                  POD.FgAvg.at(i).at(j) );
         CijPg = (SV.Gas.GetP(t1,i,j) -
                  POD.PgAvg.at(i).at(j) ) *
                 (SV.Gas.GetP(t2,i,j) -
                  POD.PgAvg.at(i).at(j) );
```

```
                CijPs = (SV.Particle.GetP(t1,i,j) -
                        POD.PsAvg.at(i).at(j) ) *
                       (SV.Particle.GetP(t2,i,j) -
                        POD.PsAvg.at(i).at(j) );
                CijPg = (SV.Particle.GetTheta(t1,i,j) -
                        POD.TsAvg.at(i).at(j) ) *
                       (SV.Particle.GetTheta(t2,i,j) -
                        POD.TsAvg.at(i).at(j) );
                }  // end for i & j

              // This is for the Covariance matrices
              // we are now using the correlation matrices
              //since uniform mesh can multiply here
//              Cijg = Cijg * SV.Getdelx() * SV.Getdely();
//              Cijs = Cijs * SV.Getdelx() * SV.Getdely();

//              CijPg = CijPg * SV.Getdelx() * SV.Getdely();
//              CijFg = CijFg * SV.Getdelx() * SV.Getdely();
//              CijPs = CijPs * SV.Getdelx() * SV.Getdely();
//              CijTs = CijTs * SV.Getdelx() * SV.Getdely();
              Cijg = Cijg/Nt;
              Cijs = Cijs/Nt;
              CijPg = CijPg/Nt;
              CijFg = CijFg/Nt;
              CijPs = CijPs/Nt;
              CijTs = CijTs/Nt;
              } // end if t1 <= t2
            else
              {
              // t1 > t2 and this matrix is symmetric
              // this cuts cov calcs in half!!!

              Cijg = CovG.at(t2).at(t1);
              Cijs = CovS.at(t2).at(t1);
              CijFg = CovFg.at(t2).at(t1);
              CijPg = CovPg.at(t2).at(t1);
              CijPs = CovPs.at(t2).at(t1);
              CijTs = CovTs.at(t2).at(t1);
              } // end else

            tmpS.push_back(Cijs);
            tmpG.push_back(Cijg);

            tmpSP.push_back(CijPs);
            tmpST.push_back(CijTs);
```

```
            tmpGP.push_back(CijPg);
            tmpGF.push_back(CijFg);

            } // end for t2

        CovG.push_back(tmpG);
        CovS.push_back(tmpS);

        CovFg.push_back(tmpGF);
        CovPg.push_back(tmpGP);
        CovPs.push_back(tmpSP);
        CovTs.push_back(tmpST);

        std::cout << "Computing covariance row number: " << t1 << "\n";
        } // end for t1

    std::cout << "Size of Covs=" << CovG.size() << " and tmpg="
            << tmpG.size() << "\n";
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void  PODSolver::HRed2Tri(std::vector< std::vector< double > > & A,
            std::vector< double > & D,
            std::vector< double > & OD, int n)

    // This subroutine comes from page 479 of the Numerical Recipes in C++
    // second edition.  This is the tred2 subroutine.

    // The input matrix A is known to be symmetric in this case
    // D returns the diagonal elements of the tridiagonal matrix
    // OD returns the off diagonal elements of the tridiagonal matrix
    // Note that since A was diagonal, it's tridiagonal reduction will be
    // as well.  OD has a leading 0 as it's dimension should be one less.
    // The integer n contains the dimensions of the matrix A and of course
    // the returned dimensions of D and OD to form the tri-diagonal matrix

    {

    int l, k, j, i;
    double scale, hh, h, g, f;

    std::vector< double > tOD (n, 0.0);
    std::vector< double > tD (n, 0.0);
```

```
std::cout << "size of tOD=" << tOD.size()
        << " size of tC=" << tD.size() << "\n";
for (i=(n-1); i > 0; i--)
   {
   l = i-1;
   scale = 0.0;
   h = 0.0;
   if (l > 0)
      {
      for (k=0; k < (l+1); k++)
         scale = scale + fabs( A.at(i).at(k) );
      if (scale == 0.0)
         tOD.at(i) = A.at(i).at(l);
      else
         {
         for (k = 0; k < (l+1); k++)
            {
            A.at(i).at(k) = A.at(i).at(k) / scale;
            h = h + A.at(i).at(k) * A.at(i).at(k);
            }
         f = A.at(i).at(l);
         g = (f >= 0.0 ? -sqrt(h) : sqrt(h));
         tOD.at(i) = scale * g;
         h = h - f * g;
         A.at(i).at(l) = f - g;
         f = 0.0;
         for (j=0; j < (l+1); j++)
            {
            A.at(j).at(i) = A.at(i).at(j) / h;
            g = 0.0;
            for (k=0; k < (j+1); k++)
               g = g + A.at(j).at(k) * A.at(i).at(k);
            for (k=(j+1); k < (l+1); k++)
               g = g + A.at(k).at(j) * A.at(i).at(k);
            tOD.at(j) = g / h;
            f = f + tOD.at(j) * A.at(i).at(j);
            }
         hh = f / (h + h);
         for (j = 0; j < (l+1); j ++)
            {
            f = A.at(i).at(j);
            g = tOD.at(j) - hh * f;
            tOD.at(j) = g;
            for (k=0; k < (j+1); k++)
```

```
                    A.at(j).at(k) = A.at(j).at(k) -
                        (f * tOD.at(k) + g * A.at(k).at(k) );
                }
            }
        }
        else
            tOD.at(i) = A.at(i).at(l);
        tD.at(i) = h;
        }

    tD.at(0) = 0.0;
    tOD.at(0) = 0.0;
    for (i=0; i < n; i++)
        {
        l = i;
        if (tD.at(i) != 0.0)
            {
            for (j=0; j < l; j++)
                {
                g = 0.0;
                for (k=0; k < l; k++)
                    g = g + A.at(i).at(k) * A.at(k).at(j);
                for (k=0; k < l; k++)
                    A.at(k).at(j) = A.at(k).at(j) - g * A.at(k).at(i);
                }
            }
        tD.at(i) = A.at(i).at(i);
        A.at(i).at(i) = 1.0;
        for (j=0; j < l; j++)
            {
            A.at(j).at(i) = 0.0;
            A.at(i).at(j) = 0.0;
            }
        }

    // Returns Diagonal and Off-Diagonal elements...
    D = tD;
    OD = tOD;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double PODSolver::pythag(double a, double b)
```

```
    {

    double c;

    c = sqrt((a*a) + (b*b));

    return c;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void  PODSolver::GetEigenVs(std::vector< double > & D,
                  std::vector< double > & OD,
                  int n, std::vector< double > & EValues,
                  std::vector< std::vector< double > > & EVectors)

    // This subroutine is from Numerical Recipes in C++ second edition
    // It can be found on page 485 called tqli.
    // This subroutine accepts D - the diagonal elements of a
    // tri-diagonal matrix and OD as the symmetric off-diagonal
    // components.  The EVectors matrix should be populated
    // initially with the output matrix from A in HRed2Tri.
    // EValues and EVectors will return the evalues with EVectors in each
    // column of the matrix.

    {

    int m, l, iter, i, k;
    double s, r, p, g, f, dd, c, b;
    double pf;
    std::vector< double > tE;
//    std::vector< std::vector< double > > EVectors;
//    std::vector< double > tE;
//    std::vector< std::vector< double > > EVectors;


    EValues = D;
    tE = OD;
//    for (i=0; i <n; i++)
//        {
//        for (j=0; j<n; j++)
//            EVectors.push_back(double(oEVectors.at(i).at(j)));
```

```
//       tE.push_back(double(OD.at(i)));
//       }

    for (i=1; i < n; i++)
       tE.at(i-1) = tE.at(i);
    tE.at(n-1) = 0.0;
    for (l=0; l < n; l++)
       {
       iter=0;
       do
          {
          for (m=l; m < (n-1); m++)
             {
             dd = fabs(EValues.at(m)) + fabs(EValues.at(m+1));
             if (fabs(tE.at(m)) + dd == dd)
                break;
             } // end for
          if (m != l)
             {
             if (iter++ == 100)
                std::cerr << "Too many iterations in GetEigenVs...\n";
             g = ( EValues.at(l+1) - EValues.at(l) ) /
                (2.0 * tE.at(l));
             r = pythag(g, 1.0);
             if (g >= 0)
                pf = 1.0;
             else
                pf = -1.0;
             g = EValues.at(m) - EValues.at(l) + tE.at(l) /
                (g + (r * pf));
             s = 1.0;
             c = 1.0;
             p = 0.0;
             for (i=(m-1); i >= l; i--)
                {
                f = s * tE.at(i);
                b = c * tE.at(i);
                tE.at(i+1) = (r = pythag(f,g) );
                if (r == 0.0)
                   {
                   EValues.at(i+1) = EValues.at(i+1) - p;
                   tE.at(m) = 0.0;
                   break;
                   }
                s = f / r;
```

```
                c = g / r;
                g = EValues.at(i+1) - p;
                r = (EValues.at(i) - g) * s + 2.0 * c*b;
                EValues.at(i+1) = g + (p = s*r);
                g = c*r - b;
                for (k=0; k < n; k++)
                   {
                   f = EVectors.at(k).at(i+1);
                   EVectors.at(k).at(i+1) = s *
                         EVectors.at(k).at(i) + c*f;
                   EVectors.at(k).at(i) = c*EVectors.at(k).at(i)
                                  - s*f;
                   }
                }
             if (r == 0.0 && i >= l)
                continue;
             EValues.at(l) = EValues.at(l) - p;
             tE.at(l) = g;
             tE.at(m) = 0.0;
             }
          }
       while (m != l);
       }
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::svd(std::vector< std::vector< double > > & A,
                std::vector< std::vector< double > > & U,
                std::vector< std::vector< double > > & V,
                std::vector< double > & Sigma, int m, int n)

    {
    // This routine is a direct translation from the c routine
    // svdcmp found in Numerical Recipes in C++ 2003 edition

    bool flag;
    int i;
    int its;
    int j;
    int jj;
    int k;
    int l;
    int nm;
```

```
    int tmp;

    double anorm;
    double c;
    double f;
    double g;
    double h;
    double s;
    double scale;
    double x;
    double y;
    double z;
    int Nits = 100;

    std::vector< double > rv1(n, 0.0);
    std::vector< double > t1(n, 0.0);

//    std::cout << "size of t1=" << t1.size() << "\n";
    // Need to initialize V
    V.clear();
    for (i=0; i < n; i++)
       V.push_back(t1);
//    std::cout << "size of V=" << V.size() << "\n";

    Sigma = t1;
    U = A;
    g = 0.0;
    scale = 0.0;
    anorm = 0.0;

    for (i=0; i < n; i++)
       {
       l = i + 2;
       rv1.at(i) = scale * g;
       g = 0.0;
       s = 0.0;
       scale = 0.0;
       if (i < m)
          {
          for (k=i; k < m; k++)
             {
             scale = scale + fabs(U.at(k).at(i));
             }
          if (scale != 0.0)
             {
```

```
        for (k = i; k < m; k++)
           {
           U.at(k).at(i) = U.at(k).at(i) / scale;
           s = s + pow(U.at(k).at(i), 2);
           }
        f = U.at(i).at(i);
        if (f >= 0)
           g = -1.0 * fabs(sqrt(s));
        else
           g = fabs(sqrt(s));
        h = f * g - s;
        U.at(i).at(i) = f - g;
        for (j=l-1; j < n; j++)
           {
           for (s=0.0, k = i; k < m; k++)
              s = s + U.at(k).at(i) * U.at(k).at(j);
           f = s / h;
           for (k=i; k < m; k++)
              U.at(k).at(j) = U.at(k).at(j) + f * U.at(k).at(i);
           }
        for (k=i; k < m; k++)
           U.at(k).at(i) = U.at(k).at(i) * scale;
        } // end if scale not 0.0
     } // end if i < m
  Sigma.at(i) = scale * g;
  g = 0.0;
  scale = 0.0;
  s = 0.0;
  if ((i + 1 <= m ) && (i + 1 != n))
     {
     for (k= l-1; k < n; k++)
        scale = scale + fabs(U.at(i).at(k));
    if (scale != 0.0)
        {
        for (k = l-1; k < n; k++)
           {
           U.at(i).at(k) = U.at(i).at(k) / scale;
           s = s + U.at(i).at(k) * U.at(i).at(k);
           }
        f = U.at(i).at(l-1);
        if (f >= 0)
           g = -1.0 * fabs(sqrt(s));
        else
           g = fabs(sqrt(s));
        h = f * g - s;
```

```
            U.at(i).at(l-1) = f-g;
            for (k = l-1; k < n; k++)
               rv1.at(k) = U.at(i).at(k) / h;
            for (j=l-1; j < m; j++)
               {
               for (s = 0.0, k=l-1; k < n; k++)
                  s = s + U.at(j).at(k) * U.at(i).at(k);
               for (k = l-1; k < n; k++)
                  U.at(j).at(k) = U.at(j).at(k) + s * (rv1.at(k));
               }
            for (k=l-1; k < n; k++)
               U.at(i).at(k) = U.at(i).at(k) * scale;
            } // end if scale not 0.0
         } // end if i's
      anorm = Max(anorm, (fabs(Sigma.at(i)) + fabs(rv1.at(i))));
      } // end for
   for (i=n-1; i >= 0; i--)
      {
      if (i < n-1)
         {
         if (g != 0.0)
            {
            for (j=l; j < n; j++)
               V.at(j).at(i) = (U.at(i).at(j) / U.at(i).at(l)) / g;
            for (j = l; j < n; j++)
               {
               for (s= 0.0, k = l; k < n; k++)
                  s = s + U.at(i).at(k) * V.at(k).at(j);
               for (k = l; k <n; k++)
                  V.at(k).at(j) = V.at(k).at(j) + s*V.at(k).at(i);
               } // end for j
            } // end if
         for (j=l; j < n; j++)
            {
            V.at(i).at(j) = 0.0;
            V.at(j).at(i) = 0.0;
            } // end for j
         } // end if i
      V.at(i).at(i) = 1.0;
      g = rv1.at(i);
      l = i;
      } // end for
   if (m < n)
      tmp = m;
   else
```

```
   tmp = n;
for (i = tmp - 1; i >= 0; i--)
   {
   l = i + 1;
   g = Sigma.at(i);
   for (j=l; j < n; j++)
      U.at(i).at(j) = 0.0;
   if (g != 0.0)
      {
      g = 1.0 / g;
      for (j=l; j < n; j++)
         {
         for (s=0.0, k=l; k < m; k++)
             s = s + U.at(k).at(i) * U.at(k).at(j);
         f = (s / U.at(i).at(i) ) * g;
         for (k=i; k < m; k++)
            U.at(k).at(j) = U.at(k).at(j) + f *U.at(k).at(i);
         } // end j
      for (j=i; j < m; j++)
         U.at(j).at(i) = U.at(j).at(i) * g;
      } // end if g
   else
      for (j=i; j <m; j++)
         U.at(j).at(i) = 0.0;
   U.at(i).at(i) = U.at(i).at(i) + 1.0;
   } // end for
for (k = n-1; k >= 0; k--)
   {
   for (its=0; its < Nits; its++)
      {
      flag = true;
      for (l=k; l >= 0; l--)
         {
         nm = l - 1;
         if (fabs(rv1.at(l) ) + anorm == anorm)
            {
            flag = false;
            break;
            } // end if
         if (fabs(Sigma.at(nm)) + anorm == anorm)
            break;
         } // end for l
      if (flag)
         {
         c = 0.0;
```

```
    s = 1.0;
    for (i = l; i < k+1; i++)
        {
        f = s * rv1.at(i);
        rv1.at(i) = c * rv1.at(i);
        if (fabs(f) + anorm == anorm)
            break;
        g = Sigma.at(i);
        h = pythag(f,g);
        Sigma.at(i) = h;
        h = 1.0 / h;
        c = g * h;
        s = -f * h;
        for (j=0; j < m; j++)
            {
            y = U.at(j).at(nm);
            z = U.at(j).at(i);
            U.at(j).at(nm) = y * c + z *s;
            U.at(j).at(i) = z * c - y*s;
            } // end for j
        } // end for i
    } // end if flag
z = Sigma.at(k);
if (l == k)
    {
    if (z < 0.0)
        {
        Sigma.at(k) = -z;
        for (j=0; j <n; j++)
            V.at(j).at(k) = - V.at(j).at(k);
        } // end if z
    break;
    } // end if l
if (its == Nits-1)
    std::cout << "No Convergence in 30 iterations. \n";
x = Sigma.at(l);
nm = k - 1;
y = Sigma.at(nm);
g = rv1.at(nm);
h = rv1.at(k);
f = ((y - z) * (y + z) + (g-h)*(g+h)) / (2.0*h*y);
g = pythag(f, 1.0);
if (f >= 0)
    f = ((x-z) * (x+z) + h*((y/(f+fabs(g)))-h)) / x;
else
```

```
    f = ((x-z) * (x+z) + h*((y/(f-fabs(g)))-h)) / x;
c = 1.0;
s = 1.0;
for (j=l; j <= nm; j++)
    {
    i = j +1;
    g = rv1.at(i);
    y = Sigma.at(i);
    h = s * g;
    g = c * g;
    z = pythag(f,h);
    rv1.at(j) = z;
    c = f / z;
    s = h / z;
    f = x*c + g*s;
    g = g*c - x*s;
    h = y*s;
    y = y*c;
    for (jj=0; jj < n; jj++)
        {
        x = V.at(jj).at(j);
        z = V.at(jj).at(i);
        V.at(jj).at(j) = x*c+z*s;
        V.at(jj).at(i) = z*c-x*s;
        } // end for jj
    z = pythag(f,h);
    Sigma.at(j) = z;
    if (z)
        {
        z = 1.0 / z;
        c = f * z;
        s = h *z;
        } // end if z
    f = c *g + s*y;
    x = c*y - s*g;
    for (jj=0; jj < m; jj++)
        {
        y = U.at(jj).at(j);
        z = U.at(jj).at(i);
        U.at(jj).at(j) = y * c + z * s;
        U.at(jj).at(i) = z * c - y*s;
        } // end for jj
    } // end for j
rv1.at(l) = 0.0;
rv1.at(k) = f;
```

```
            Sigma.at(k) = x;
            } // end for its
        } // end for k

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::Solver(std::vector< std::vector< double > > & U,
                    std::vector< std::vector< double > > & V,
                    std::vector< double > & Sigma, int m, int n,
                    std::vector< double > & X)

    { // mostly derived from svbksb in Numerical recipes for C: 2nd ed.
      // Solves the system Ax = b where a is mxn
      // m = 2 * NX * NY and n = # modes

    std::vector< double > b;

    int jj;
    int j;
    int i;

    double s;
    std::vector< double > tmp(n,0.0);

    b = X;

    for (j=0; j < n; j++)
        {
        s = 0.0;
        if (Sigma.at(j) )
            {
            for (i = 0; i < m; i++)
                s = s + U.at(i).at(j)*b.at(i);
            s = s / Sigma.at(j);
            } // end if Sigma j
        tmp.at(j) = s;
        } // end for j

    for (j= 0; j < n; j++)
        {
        s = 0.0;
        for (jj = 0; jj < n; jj++)
```

```
        s = s + V.at(i).at(jj) * tmp.at(jj);
      X.at(j) = s;
      } // end for j

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int PODSolver::GetNx()

   {

   return Nx;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int PODSolver::GetNy()

   {

   return Ny;

   }


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

int PODSolver::GetNt()

   {

   return Nt;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::copy2vec(std::vector< std::vector< double > > & M,
                 int dim, int i, std::vector< double > & V)
```

```
    {

    int j;

    V.clear();
    for (j=0; j < dim; j++)
       V.push_back(M.at(i).at(j));

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

bool PODSolver::LoadPOD(SimVars & Data, int Id, int NIds,
                  std::string & ln)

    {

    bool good = true;
    int i;

    std::cout << ln;
    std::cout << "\n is the directory. \n";

    int GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis;

    POD.InitPODData(Data.GetNx(), Data.GetNy(), Data.GetNt(),
               Data.Getdelx(), Data.Getdely(),
               Data.Getdelt() );

    MPI_Barrier(MPI_COMM_WORLD);

    InputMeans(ln);

    std::cout << Id << " out of ";
    std::cout << NIds << " has input means. \n";

    MPI_Barrier(MPI_COMM_WORLD);

    for (i = 0; i < NIds; i++)
       {
       if (Id == i)
          {
```

```
      std::cout << Id << " is getting key. \n";
      InputKey(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,
            ln);
      }
   MPI_Barrier(MPI_COMM_WORLD);
   }

std::cout << GUNBasis << "\n" << GVNBasis << "\n" << PUNBasis
        << "\n" << PVNBasis << "\n" << GPNBasis << "\n" << GFNBasis
        << "\n" << PPNBasis << "\n" << TSNBasis << " are basis #s. \n";


std::cout << Id << " has input key. \n";

POD.SetNUG(GUNBasis);
POD.SetNVG(GVNBasis);
POD.SetNUS(PUNBasis);
POD.SetNVS(PVNBasis);
POD.SetNEG(GFNBasis);
POD.SetNPG(GPNBasis);
POD.SetNPS(PPNBasis);
POD.SetNTS(TSNBasis);

std::cout << Id << " has stored basis numbers. \n";

InputBasis(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
         GPNBasis, GFNBasis, PPNBasis, TSNBasis, ln);

std::cout << Id << " has input basis. \n";

InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
        GPNBasis, GFNBasis, PPNBasis, TSNBasis, ln);

std::cout << Id << " has input POD coefficients. \n";

MPI_Barrier(MPI_COMM_WORLD);

std::cout << "Data has been loaded.  \n";

return good;

}
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::FormGub(std::vector< double > & V, int t, SimVars & Data)

   {

   int i;
   int j;

   V.clear();
   for (j = 0; j < Ny; j++)
      for (i = 0; i < Nx; i++)
         V.push_back(Data.Gas.GetU(t,i,j) -
                  POD.UgAvgVel.at(i).at(j));
//    std::cout << "Have Gub \n";

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::FormGvb(std::vector< double > & V, int t, SimVars & Data)

   {

   int i;
   int j;

   V.clear();
   for (j = 0; j < Ny; j++)
      for (i = 0; i < Nx; i++)
         V.push_back(Data.Gas.GetV(t,i,j) -
                  POD.VgAvgVel.at(i).at(j));
//    std::cout << "Have Gvb \n";

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::FormPub(std::vector< double > & V, int t, SimVars & Data)

   {
```

```
    int i;
    int j;

    V.clear();
    for (j = 0; j < Ny; j++)
        for (i = 0; i < Nx; i++)
            V.push_back(Data.Particle.GetU(t,i,j) -
                        POD.UsAvgVel.at(i).at(j));
//    std::cout << "Have Pub \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::FormPvb(std::vector< double > & V, int t, SimVars & Data)

    {

    int i;
    int j;

    V.clear();
    for (j = 0; j < Ny; j++)
        for (i = 0; i < Nx; i++)
            V.push_back(Data.Particle.GetV(t,i,j) -
                        POD.VsAvgVel.at(i).at(j));
//    std::cout << "Have Pvb \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::FormGpb(std::vector< double > & V, int t, SimVars & Data)

    {

    int i;
    int j;

    V.clear();
    for (j = 0; j < Ny; j++)
        for (i = 0; i < Nx; i++)
            V.push_back(Data.Gas.GetP(t,i,j) -
```

```
                    POD.PgAvg.at(i).at(j));
//    std::cout << "Have Gpb \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::FormGfb(std::vector< double > & V, int t, SimVars & Data)

    {

    int i;
    int j;

    V.clear();
    for (j = 0; j < Ny; j++)
        for (i = 0; i < Nx; i++)
            V.push_back(Data.Gas.GetEP(t,i,j) -
                    POD.FgAvg.at(i).at(j));
//    std::cout << "Have Gfb \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::FormPsb(std::vector< double > & V, int t, SimVars & Data)

    {

    int i;
    int j;

    V.clear();
    for (j = 0; j < Ny; j++)
        for (i = 0; i < Nx; i++)
            V.push_back(Data.Particle.GetP(t,i,j) -
                    POD.PsAvg.at(i).at(j));
//    std::cout << "Have Psb \n";


    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::FormTsb(std::vector< double > & V, int t, SimVars & Data)

   {

   int i;
   int j;

   V.clear();
   for (j = 0; j < Ny; j++)
      for (i = 0; i < Nx; i++)
         V.push_back(Data.Particle.GetTheta(t,i,j) -
                  POD.TsAvg.at(i).at(j));
//   std::cout << "Have Tsb \n";

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::FormReconVars(int NbGU, int NbGV, int NbPU, int NbPV,
                     int NbGP, int NbGF, int NbPP, int NbPT,
                     int Id, int NIds)

   {
   int t;
   int x;
   int y;
   int k;

   double valu0;
   double valv1;
   double valu2;
   double valv3;

   double valfg;
   double valpg;
   double valps;
   double valts;

   std::vector< double > tUg;
   std::vector< double > tVg;
   std::vector< double > tUs;
```

```
    std::vector< double > tVs;

    std::vector< double > tFg;
    std::vector< double > tPg;
    std::vector< double > tPs;
    std::vector< double > tTs;

    std::vector< std::vector< double > > txUg;
    std::vector< std::vector< double > > txVg;
    std::vector< std::vector< double > > txUs;
    std::vector< std::vector< double > > txVs;

    std::vector< std::vector< double > > txFg;
    std::vector< std::vector< double > > txPg;
    std::vector< std::vector< double > > txPs;
    std::vector< std::vector< double > > txTs;

    for (t=0; t < Nt; t++)
       {
       txUg.clear();
       txVg.clear();
       txUs.clear();
       txVs.clear();

       txFg.clear();
       txPg.clear();
       txPs.clear();
       txTs.clear();

       for (y=0; y < Ny; y++)
          {
          tUg.clear();
          tVg.clear();
          tUs.clear();
          tVs.clear();

          tPg.clear();
          tFg.clear();
          tPs.clear();
          tTs.clear();

          for (x=0; x < Nx; x++)
             {

//             if (Id == (0 % NIds))
```

```
//                {
                  valu0 = POD.UgAvgVel.at(x).at(y);

                  for (k=0; k < NbGU; k++)
                     valu0 = valu0 + (POD.GPhiuPOD.at(x+(Nx*y)).at(k) *
                              POD.PODsGasU.at(t).at(k));
                  if (isnan(valu0))
                     std::cout << x << "=x and y=" << y
                           << " Ug NaN! \n";
                  tUg.push_back(valu0);
//                } // end 0

//              if (Id == (1 % NIds))
//                {
                  valv1 = POD.VgAvgVel.at(x).at(y);

                  for (k=0; k < NbGV; k++)
                      valv1 = valv1 + (POD.GPhivPOD.at(x+(Nx*y)).at(k) *
                               POD.PODsGasV.at(t).at(k));

                  if (isnan(valv1))
                     std::cout << x << "=x and y=" << y
                           << " Vg NaN! \n";
                  tVg.push_back(valv1);
//                } // end 1

//              if (Id == (2 % NIds))
//                {
                  valu2 = POD.UsAvgVel.at(x).at(y);

                  for (k=0; k < NbPU; k++)
                     valu2 = valu2 + (POD.PPhiuPOD.at(x+(Nx*y)).at(k) *
                              POD.PODsParticleU.at(t).at(k));

                  if (isnan(valu2))
                     std::cout << x << "=x and y=" << y
                           << " Us NaN! \n";
                  tUs.push_back(valu2);
//                } // end 2

//              if (Id == (3 % NIds))
//                {
                  valv3 = POD.VsAvgVel.at(x).at(y);

                  for (k=0; k < NbPV; k++)
```

```
                valv3 = valv3 + (POD.PPhivPOD.at(x+(Nx*y)).at(k) *
                        POD.PODsParticleV.at(t).at(k));
              if (isnan(valv3))
                 std::cout << x << "=x and y=" << y
                        << " Vs NaN! \n";
              tVs.push_back(valv3);
//              } // end 3

//            if (Id == (5 % NIds))
//              {
              valfg = POD.FgAvg.at(x).at(y);
              for (k=0; k < NbGF; k++)
                 valfg = valfg + (POD.FgPhiPOD.at(x+(Nx*y)).at(k) *
                        POD.PODsGasF.at(t).at(k));
              if (isnan(valfg))
                 std::cout << x << "=x and y=" << y
                        << " Fg NaN! \n";
              tFg.push_back(valfg);
//              } // end 5

//            if (Id == (4 % NIds))
//              {
              valpg = POD.PgAvg.at(x).at(y);
              for (k=0; k < NbGP; k++)
                 valpg = valpg + (POD.PgPhiPOD.at(x+(Nx*y)).at(k) *
                        POD.PODsGasP.at(t).at(k));
              if (isnan(valpg))
                 std::cout << x << "=x and y=" << y
                        << " Pg NaN! \n";
              tPg.push_back(valpg);
//              } // end 4

//            if (Id == (6 % NIds))
//              {
              valps = POD.PsAvg.at(x).at(y);
              for (k=0; k < NbPP; k++)
                 valps = valps + (POD.PsPhiPOD.at(x+(Nx*y)).at(k) *
                        POD.PODsParticleP.at(t).at(k));
              if (isnan(valps))
                 std::cout << x << "=x and y=" << y
                        << " Ps NaN! \n";
              tPs.push_back(valps);
//              } // end 6

//            if (Id == (7 % NIds))
```

```
//                  {
                  valts = POD.TsAvg.at(x).at(y);
                  for (k=0; k < NbPT; k++)
                      valts = valts + (POD.TsPhiPOD.at(x+(Nx*y)).at(k) *
                                  POD.PODsParticleT.at(t).at(k));
                   if (isnan(valts))
                      std::cout << x << "=x and y=" << y
                              << " Ts NaN! \n";
                  tTs.push_back(valts);
//                  } // end 7

               } // end for x

            txUg.push_back(tUg);
            txVg.push_back(tVg);
            txUs.push_back(tUs);
            txVs.push_back(tVs);

            txPg.push_back(tPg);
            txFg.push_back(tFg);
            txPs.push_back(tPs);
            txTs.push_back(tTs);
            } // end for y
        UgR.push_back(txUg);
        VgR.push_back(txVg);
        UsR.push_back(txUs);
        VsR.push_back(txVs);

        FgR.push_back(txFg);
        PgR.push_back(txPg);
        PsR.push_back(txPs);
        TsR.push_back(txTs);
        } // end for t

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::ApproxErrorSplit(SimVars & Data, int Id, int NIds)

    {

    int x;
    int y;
```

```
int t;

double val;

std::vector< double > tegu;
std::vector< double > tegv;
std::vector< double > tesu;
std::vector< double > tesv;

std::vector< double > tegf;
std::vector< double > tegp;
std::vector< double > tesp;
std::vector< double > test;

std::vector< std::vector< double > > txegu;
std::vector< std::vector< double > > txegv;
std::vector< std::vector< double > > txesu;
std::vector< std::vector< double > > txesv;

std::vector< std::vector< double > > txegf;
std::vector< std::vector< double > > txegp;
std::vector< std::vector< double > > txesp;
std::vector< std::vector< double > > txest;

std::cout << Id << "Approx Error \n";
for (t=0; t < Nt; t++)
   {
   txegu.clear();
   txegv.clear();
   txesu.clear();
   txesv.clear();

   txegf.clear();
   txegp.clear();
   txesp.clear();
   txest.clear();

   for (y=0; y < Ny; y++)
      {
      tegu.clear();
      tegv.clear();
      tesu.clear();
      tesv.clear();

      tegf.clear();
```

```
        tegp.clear();
        tesp.clear();
        test.clear();
        for (x=0; x < Nx; x++)
           {
//            if (Id == (0 % NIds))
//               {
              if (fabs(UgR.at(t).at(y).at(x)) > epsilon)
                 val = fabs(Data.Gas.GetU(t,x,y) -
                          UgR.at(t).at(y).at(x)) /
                      fabs(UgR.at(t).at(y).at(x));
              else
                 val = 0.0;
              tegu.push_back(val);
//std::cout << Data.Gas.GetU(t,x,y) << " =O & N= "
//        << UgR.at(t).at(y).at(x) << "\n";
//               }

//            if (Id == (1 % NIds))
//               {
              if (fabs(VgR.at(t).at(y).at(x)) > epsilon)
                 val = fabs(Data.Gas.GetV(t,x,y) -
                          VgR.at(t).at(y).at(x)) /
                      fabs(VgR.at(t).at(y).at(x));
              else
                 val = 0.0;
//std::cout << Data.Gas.GetV(t,x,y) << " =O & N= "
//        << VgR.at(t).at(y).at(x) << "\n";
              tegv.push_back(val);
//               }

//            if (Id == (2 % NIds))
//               {
              if (fabs(UsR.at(t).at(y).at(x)) > epsilon)
                 val = fabs(Data.Particle.GetU(t,x,y) -
                          UsR.at(t).at(y).at(x)) /
                      fabs(UsR.at(t).at(y).at(x));
              else
                 val = 0.0;
//std::cout << Data.Particle.GetU(t,x,y) << " =O & N= "
//        << UsR.at(t).at(y).at(x) << "\n";
              tesu.push_back(val);
//               } // end 2

//            if (Id == (3 % NIds))
```

```
//                {
                if (fabs(VsR.at(t).at(y).at(x)) > epsilon)
                   val = fabs(Data.Particle.GetV(t,x,y) -
                         VsR.at(t).at(y).at(x)) /
                      fabs(VsR.at(t).at(y).at(x));
                else
                   val = 0.0;
//std::cout << Data.Particle.GetV(t,x,y) << " =O & N= "
//          << VsR.at(t).at(y).at(x) << "\n";
                tesv.push_back(val);
//                } // end 3

//              if (Id == (5 % NIds))
//                {
                if (fabs(FgR.at(t).at(y).at(x)) > epsilon)
                   val = fabs(Data.Gas.GetEP(t,x,y) -
                         FgR.at(t).at(y).at(x)) /
                      fabs(FgR.at(t).at(y).at(x));
                else
                   val = 0.0;
//std::cout << Data.Gas.GetEP(t,x,y) << " =O & N= "
//          << FgR.at(t).at(y).at(x) << "\n";
                tegf.push_back(val);
//                } // end 5

//              if (Id == (4 % NIds))
//                {
                if (fabs(PgR.at(t).at(y).at(x)) > epsilon)
                   val = fabs(Data.Gas.GetP(t,x,y) -
                         PgR.at(t).at(y).at(x)) /
                      fabs(PgR.at(t).at(y).at(x));
                else
                   val = 0.0;
//std::cout << Data.Gas.GetP(t,x,y) << " =O & N= "
//          << PgR.at(t).at(y).at(x) << "\n";
                tegp.push_back(val);
//                } // end 4

//              if (Id == (6 % NIds))
//                {
                if (fabs(PsR.at(t).at(y).at(x)) > epsilon)
                   val = fabs(Data.Particle.GetP(t,x,y) -
                         PsR.at(t).at(y).at(x)) /
                      fabs(PsR.at(t).at(y).at(x));
                else
```

```
                    val = 0.0;
//std::cout << Data.Particle.GetP(t,x,y) << " =O & N= "
//          << PsR.at(t).at(y).at(x) << "\n";
                tesp.push_back(val);
//                } // end 6

//              if (Id == (7 % NIds))
//                {
                if (fabs(TsR.at(t).at(y).at(x)) > epsilon)
                    val = fabs(Data.Particle.GetTheta(t,x,y) -
                            TsR.at(t).at(y).at(x)) /
                        fabs(TsR.at(t).at(y).at(x));
                else
                    val = 0.0;
//std::cout << Data.Particle.GetTheta(t,x,y) << " =O & N= "
//          << TsR.at(t).at(y).at(x) << "\n";
                test.push_back(val);
//                }

             } // end for x
//           std::cout << Id << " pushing back x vectors "
//                     << y << "\n";
          txegu.push_back(tegu);
          txegv.push_back(tegv);
          txesu.push_back(tesu);
          txesv.push_back(tesv);

          txegf.push_back(tegf);
          txegp.push_back(tegp);
          txesp.push_back(tesp);
          txest.push_back(test);
          } // end for y
       ErrorUg.push_back(txegu);
       ErrorUs.push_back(txesu);
       ErrorVg.push_back(txegv);
       ErrorVs.push_back(txesv);

       ErrorFg.push_back(txegf);
       ErrorPg.push_back(txegp);
       ErrorPs.push_back(txesp);
       ErrorTs.push_back(txest);
       } // end for t
    std::cout << Id << " exiting Approx error split... \n";

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::AdjustSingVals(std::vector< double > & S, int Ns)

  {

  float Smax = 0.0;
  float TOL = 1.0e-13;
  float thresh;
  int j;

  for (j=0; j < Ns; j++)
    if (S.at(j) > Smax)
      Smax = S.at(j);
  thresh = TOL * Smax;
  for (j=0; j < Ns; j++)
    if (S.at(j) < thresh)
      S.at(j) = 0.0;

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::Solve4coefs(std::vector< std::vector< double > > & U,
              std::vector< std::vector< double > > & V,
              std::vector< double > & W,
              std::vector< double > & b,
              int Npts, int mbasis,
              std::vector< double > & X)

  {

  std::vector< double > a (mbasis, 0.0);

  int i;
  int index;
  int ai;
  float temp;

  for (i= 0; i < mbasis; i++)
    {
    temp = 0.0;
```

```
        for (index = 0; index < Npts; index++)
            {
//            std::cout << index << "=index \n";
            temp = temp + (U.at(index).at(i) * b.at(index));
            }
        temp = temp / W.at(i);
//        std::cout << "Entering solver \n";
        for (ai=0; ai < mbasis; ai++)
            {
//            std::cout << ai << "=ai \n";
            a.at(ai) = a.at(ai) + (temp * V.at(i).at(ai));
            }
        } // end for i

    X = a;

    std::cout << "Have all coefficients...  \n";
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

std::string intToString(int t)

    {

    std::string ch;

    std::ostringstream outs;

    outs << t; // Convert value into a string.

    ch = outs.str();

    return ch;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputMeans(std::string ln)

    {
```

```
char c = '.';

if (ln.size() == 0)
   ln = c;

std::string fegu = ln + "/AvgGasVelocityX.txt";
std::string fegv = ln + "/AvgGasVelocityY.txt";
std::string fesu = ln + "/AvgParticleVelocityX.txt";
std::string fesv = ln + "/AvgParticleVelocityY.txt";
std::string fegf = ln + "/AvgGasFF.txt";
std::string fegp = ln + "/AvgGasPressure.txt";
std::string fesp = ln + "/AvgParticlePressure.txt";
std::string fest = ln + "/AvgParticleTemperature.txt";

std::ifstream ifilegu (fegu.c_str());
std::ifstream ifilegv (fegv.c_str());
std::ifstream ifilesu (fesu.c_str());
std::ifstream ifilesv (fesv.c_str());

std::ifstream ifilegf (fegf.c_str());
std::ifstream ifilegp (fegp.c_str());
std::ifstream ifilesp (fesp.c_str());
std::ifstream ifilest (fest.c_str());

int y;
int x;

std::vector< double > tmp0 (Ny, 0.0);
std::vector< std::vector< double > > t0(Nx, tmp0);
std::vector< std::vector< double > > t1(Nx, tmp0);
std::vector< std::vector< double > > t2(Nx, tmp0);
std::vector< std::vector< double > > t3(Nx, tmp0);
std::vector< std::vector< double > > t5(Nx, tmp0);
std::vector< std::vector< double > > t4(Nx, tmp0);
std::vector< std::vector< double > > t6(Nx, tmp0);
std::vector< std::vector< double > > t7(Nx, tmp0);

for (y=Ny-1; y >=0; y--)
   {
   for (x=Nx-1; x >= 0; x--)
      {
      ifilegu >> t0.at(x).at(y) ;
      ifilegv >> t1.at(x).at(y) ;
      ifilesu >> t2.at(x).at(y) ;
      ifilesv >> t3.at(x).at(y) ;
```

```
         ifilegf >> t5.at(x).at(y) ;
         ifilegp >> t4.at(x).at(y) ;
         ifilesp >> t6.at(x).at(y) ;
         ifilest >> t7.at(x).at(y) ;
         } // end for x
      } // end for y

   ifilegu.close();
   ifilegu.close();
   ifilesv.close();
   ifilesv.close();

   ifilegf.close();
   ifilegp.close();
   ifilesp.close();
   ifilest.close();

   POD.UgAvgVel = t0;
   POD.VgAvgVel = t1;
   POD.UsAvgVel = t2;
   POD.VsAvgVel = t3;
   POD.FgAvg = t5;
   POD.PgAvg = t4;
   POD.PsAvg = t6;
   POD.TsAvg = t7;

   std::cout << t0.size() << " is the size of t0. \n";
   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputSingVals(std::vector< double > V, int index)

   {

   int z = V.size();
   int t;
   std::string fegu;

   if (index == 0)
      fegu = "./Output/SingValsGasVelocityX.txt";
   else if (index == 1)
      fegu = "./Output/SingValsGasVelocityY.txt";
```

```
   else if (index == 2)
      fegu = "./Output/SingValsParticleVelocityX.txt";
   else if (index == 3)
      fegu = "./Output/SingValsParticleVelocityY.txt";
   else if (index == 4)
      fegu = "./Output/SingValsGasPressure.txt";
   else if (index == 5)
      fegu = "./Output/SingValsGasVoidFraction.txt";
   else if (index == 6)
      fegu = "./Output/SingValsParticlePressure.txt";
   else if (index == 7)
      fegu = "./Output/SingValsParticleTemp.txt";

   std::ofstream ofilegu (fegu.c_str());

   for (t=0; t < z; t++)
      ofilegu << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << V.at(t) << "\n";

   ofilegu.close();

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputMeans()

   {

   std::string fegu = "./Output/AvgGasVelocityX.txt";
   std::string fegv = "./Output/AvgGasVelocityY.txt";
   std::string fesu = "./Output/AvgParticleVelocityX.txt";
   std::string fesv = "./Output/AvgParticleVelocityY.txt";
   std::string fegf = "./Output/AvgGasFF.txt";
   std::string fegp = "./Output/AvgGasPressure.txt";
   std::string fesp = "./Output/AvgParticlePressure.txt";
   std::string fest = "./Output/AvgParticleTemperature.txt";

   std::ofstream ofilegu (fegu.c_str());
   std::ofstream ofilegv (fegv.c_str());
   std::ofstream ofilesu (fesu.c_str());
   std::ofstream ofilesv (fesv.c_str());
```

```
std::ofstream ofilegf (fegf.c_str());
std::ofstream ofilegp (fegp.c_str());
std::ofstream ofilesp (fesp.c_str());
std::ofstream ofilest (fest.c_str());

int y;
int x;


for (y=(Ny-1); y >=0; y--)
   {
   for (x=(Nx-1); x > 0; x--)
      {
      ofilegu << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << POD.UgAvgVel.at(x).at(y) << " ";
      ofilegv << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << POD.VgAvgVel.at(x).at(y) << " ";
      ofilesu << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << POD.UsAvgVel.at(x).at(y) << " ";
      ofilesv << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << POD.VsAvgVel.at(x).at(y) << " ";

      ofilegf << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << POD.FgAvg.at(x).at(y) << " ";
      ofilegp << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << POD.PgAvg.at(x).at(y) << " ";
      ofilesp << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << POD.PsAvg.at(x).at(y) << " ";
      ofilest << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << POD.TsAvg.at(x).at(y) << " ";
      } // end for x
   ofilegu << std::setiosflags(std::ios::fixed)
         << std::setprecision(15)
         << POD.UgAvgVel.at(0).at(y) << "\n";
   ofilegv << std::setiosflags(std::ios::fixed)
         << std::setprecision(15)
         << POD.VgAvgVel.at(0).at(y) << "\n";
```

```
        ofilesu << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << POD.UsAvgVel.at(0).at(y) << "\n";
        ofilesv << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << POD.VsAvgVel.at(0).at(y) << "\n";
        ofilegf << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << POD.FgAvg.at(0).at(y) << "\n";
        ofilegp << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << POD.PgAvg.at(0).at(y) << "\n";
        ofilesp << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << POD.PsAvg.at(0).at(y) << "\n";
        ofilest << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << POD.TsAvg.at(0).at(y) << "\n";
        } // end for y

    ofilegu.close();
    ofilegu.close();
    ofilesv.close();
    ofilesv.close();

    ofilegf.close();
    ofilegp.close();
    ofilesp.close();
    ofilest.close();
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputGUBasis(int Nbasis, std::string ln)

    {

    int x;
    int y;
    int i;

    char c = '.';

    std::vector< double > tmp(Nbasis, 0.0);
```

```
    std::vector< std::vector< double > > GPuPOD(Ny*Nx, tmp);

    if (ln.size() == 0)
        ln = c;

    for (i=0; i < Nbasis; i++)
        {
        std::string fnx = ln +
         "/GasVelocityBasisX" + intToString(i) + ".txt";

        std::ifstream ifilex (fnx.c_str());

        for (y=Ny-1; y >=0; y--)
            for (x=Nx-1; x >= 0; x--)
                ifilex >> GPuPOD.at((y*Nx)+x).at(i);
        ifilex.close();
        } // end for i

    POD.GPhiuPOD = GPuPOD;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputGUBasis(int Nbasis)

    {

    int x;
    int y;
    int i;

    for (i=0; i < Nbasis; i++)
        {
        std::string fnx =
         "./Output/GasVelocityBasisX" + intToString(i) + ".txt";

        std::ofstream ofilex (fnx.c_str());

        for (y=Ny-1; y >=0; y--)
            {
            for (x=Nx-1; x > 0; x--)
                {
                ofilex << std::setiosflags(std::ios::fixed)
```

```
                  << std::setprecision(15)
                  << POD.GPhiuPOD.at((y*Nx)+x).at(i) << " ";
              } // end for x
            ofilex << std::setiosflags(std::ios::fixed)
                 << std::setprecision(15)
                 << POD.GPhiuPOD.at(y*Nx).at(i) << "\n";
            } // end for y

        ofilex.close();

        } // end for i

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputGVBasis(int Nbasis, std::string ln)

    {

    int x;
    int y;
    int i;

    char c = '.';

//    std::cout << "In Gv Basis... " << Nbasis << "\n";

    std::vector< double > tmp(Nbasis, 0.0);
    std::vector< std::vector< double > > GPvPOD(Ny*Nx, tmp);

    if (ln.size() == 0)
        ln = c;

//    std::cout << "In Gv Basis... \n";

    for (i=0; i < Nbasis; i++)
        {

        std::string fny = ln +
         "/GasVelocityBasisY" + intToString(i) + ".txt";

        std::ifstream ifiley (fny.c_str());
```

```
    for (y=Ny-1; y >=0; y--)
      for (x=Nx-1; x >= 0; x--)
        ifiley >> GPvPOD.at((y*Nx)+x).at(i) ;

    ifiley.close();
    } // end for i

  POD.GPhivPOD = GPvPOD;

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputGVBasis(int Nbasis)

  {

  int x;
  int y;
  int i;

  for (i=0; i < Nbasis; i++)
    {
    std::string fny =
     "./Output/GasVelocityBasisY" + intToString(i) + ".txt";

    std::ofstream ofiley (fny.c_str());

    for (y=Ny-1; y >=0; y--)
      {
      for (x=Nx-1; x > 0; x--)
        {
        ofiley << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << POD.GPhivPOD.at((y*Nx)+x).at(i) << " ";
        } // end for
      ofiley << std::setiosflags(std::ios::fixed)
             << std::setprecision(15)
             << POD.GPhivPOD.at(y*Nx).at(i) << "\n";
      }  // end for y

    ofiley.close();
    } // end for i
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputPUBasis(int Nbasis, std::string ln)

    {

    int x;
    int y;
    int i;

    char c = '.';

    std::vector< double > tmp(Nbasis, 0.0);
    std::vector< std::vector< double > > PPuPOD(Ny*Nx, tmp);

    if (ln.size() == 0)
        ln = c;

    for (i=0; i < Nbasis; i++)
        {

        std::string fnx = ln +
         "/ParticleVelocityBasisX" + intToString(i) + ".txt";

        std::ifstream ifilex (fnx.c_str());

        for (y=Ny-1; y >=0; y--)
            for (x=Nx-1; x >= 0; x--)
                ifilex >> PPuPOD.at((y*Nx)+x).at(i) ;

        ifilex.close();
        } // end for i

    POD.PPhiuPOD = PPuPOD;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputPUBasis(int Nbasis)
```

```
    {

    int x;
    int y;
    int i;

    for (i=0; i < Nbasis; i++)
       {
       std::string fnx =
        "./Output/ParticleVelocityBasisX" + intToString(i) + ".txt";

       std::ofstream ofilex (fnx.c_str());

       for (y=Ny-1; y >=0; y--)
          {
          for (x=Nx-1; x > 0; x--)
             {
             ofilex << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << POD.PPhiuPOD.at((y*Nx)+x).at(i) << " ";
             } // end for
          ofilex << std::setiosflags(std::ios::fixed)
                 << std::setprecision(15)
                 << POD.PPhiuPOD.at(y*Nx).at(i) << "\n";
          } // end for y

       ofilex.close();
       } // end for i

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputPVBasis(int Nbasis, std::string ln)

    {

    int x;
    int y;
    int i;

    char c = '.';

    std::vector< double > tmp(Nbasis, 0.0);
```

```
    std::vector< std::vector< double > > PPvPOD(Ny*Nx, tmp);

    if (ln.size() == 0)
        ln = c;

    for (i=0; i < Nbasis; i++)
        {

        std::string fny = ln +
         "/ParticleVelocityBasisY" + intToString(i) + ".txt";

        std::ifstream ifiley (fny.c_str());

        for (y=Ny-1; y >=0; y--)
            for (x=Nx-1; x >= 0; x--)
                ifiley >> PPvPOD.at((y*Nx)+x).at(i);

        ifiley.close();
        } // end for i

    POD.PPhivPOD = PPvPOD;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputPVBasis(int Nbasis)

    {

    int x;
    int y;
    int i;

    for (i=0; i < Nbasis; i++)
        {

        std::string fny =
         "./Output/ParticleVelocityBasisY" + intToString(i) + ".txt";

        std::ofstream ofiley (fny.c_str());

        for (y=Ny-1; y >=0; y--)
            {
```

```
            for (x=Nx-1; x > 0; x--)
                {
                ofiley << std::setiosflags(std::ios::fixed)
                       << std::setprecision(15)
                       << POD.PPhivPOD.at((y*Nx)+x).at(i) << " ";
                } // end for
            ofiley << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << POD.PPhivPOD.at(y*Nx).at(i) << "\n";
            } // end for y

        ofiley.close();
        } // end for i

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputGFBasis(int Nbasis, std::string ln)

    {

    int x;
    int y;
    int i;

    char c = '.';

    std::vector< double > tmp(Nbasis, 0.0);
    std::vector< std::vector< double > > FgPOD(Ny*Nx, tmp);

    if (ln.size() == 0)
        ln = c;

    for (i=0; i < Nbasis; i++)
        {

        std::string fn = ln +
         "/GasFFBasis" + intToString(i) + ".txt";
        std::ifstream ifile (fn.c_str());

        for (y=Ny-1; y >=0; y--)
            for (x=Nx-1; x >= 0; x--)
                ifile >> FgPOD.at((y*Nx)+x).at(i) ;
```

```
        ifile.close();
        }

    POD.FgPhiPOD = FgPOD;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputGFBasis(int Nbasis)

    {

    int x;
    int y;
    int i;

    for (i = 0; i < Nbasis; i++)
        {
        std::string fn =
         "./Output/GasFFBasis" + intToString(i) + ".txt";
        std::ofstream ofile (fn.c_str());

        for (y=Ny-1; y >=0; y--)
            {
            for (x=Nx-1; x > 0; x--)
                {
//std::cout<< "I'm writing GF" << (y*Nx)+x << " = i and j = "
// << i << "\n";
                ofile << std::setiosflags(std::ios::fixed)
                      << std::setprecision(15)
                      << POD.FgPhiPOD.at((y*Nx)+x).at(i) << " ";
                } // end for x
//std::cout<< "I'm writing GF" << (y*Nx)+x << " = i and j = "
// << i << "\n";
            ofile << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << POD.FgPhiPOD.at((y*Nx)).at(i) << "\n";
            } // end for y

        ofile.close();
        } // end for i
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputGPBasis(int Nbasis, std::string ln)

    {

    int x;
    int y;
    int i;

    char c = '.';

    std::vector< double > tmp(Nbasis, 0.0);
    std::vector< std::vector< double > > PgPOD(Ny*Nx, tmp);

    if (ln.size() == 0)
        ln = c;

    for (i=0; i < Nbasis; i++)
        {

        std::string fn = ln +
         "/GasPressureBasis" + intToString(i) + ".txt";
        std::ifstream ifile (fn.c_str());

        for (y=Ny-1; y >=0; y--)
            for (x=Nx-1; x >= 0; x--)
                ifile >> PgPOD.at((y*Nx)+x).at(i);

        ifile.close();

        } // end for i

    POD.PgPhiPOD = PgPOD;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputGPBasis(int Nbasis)
```

```
    {

    int x;
    int y;
    int i;

    for (i=0; i < Nbasis; i++)
        {

        std::string fn =
         "./Output/GasPressureBasis" + intToString(i) + ".txt";
        std::ofstream ofile (fn.c_str());

        for (y=(Ny-1); y >=0; y--)
            {
            for (x=(Nx-1); x > 0; x--)
                {
//std::cout<< "I'm writing Gp " << (y*Nx)+x << " = i and j = "
//<< i << "\n";
                ofile << std::setiosflags(std::ios::fixed)
                      << std::setprecision(15)
                      << POD.PgPhiPOD.at((y*Nx)+x).at(i) << " ";
                } // end for x
//std::cout<< "I'm writing Gp " << (y*Nx)+x << " = i and j = "
//<< i << "\n";
            ofile << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << POD.PgPhiPOD.at(y*Nx).at(i) << "\n";
            } // end for y

        ofile.close();
        } // end for i;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputPPBasis(int Nbasis, std::string ln)

    {

    int x;
    int y;
    int i;
```

```
    char c = '.';

    std::vector< double > tmp(Nbasis, 0.0);
    std::vector< std::vector< double > > PsPOD(Ny*Nx, tmp);

    if (ln.size() == 0)
       ln = c;

    for (i=0; i < Nbasis; i++)
       {

       std::string fn = ln +
        "/ParticlePressureBasis" + intToString(i) + ".txt";
       std::ifstream ifile (fn.c_str());

       for (y=Ny-1; y >=0; y--)
          for (x=Nx-1; x >= 0; x--)
             ifile >> PsPOD.at((y*Nx)+x).at(i);

       ifile.close();
       } // end for i

    POD.PsPhiPOD = PsPOD;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputPPBasis(int Nbasis)

    {

    int x;
    int y;
    int i;

    for (i=0; i < Nbasis; i++)
       {

       std::string fn =
        "./Output/ParticlePressureBasis" + intToString(i) + ".txt";
       std::ofstream ofile (fn.c_str());
```

```
        for (y=(Ny-1); y >=0; y--)
          {
          for (x=(Nx-1); x > 0; x--)
             {
//std::cout<< "I'm writing Pp" << (y*Nx)+x << " = i and j = "
//<< i  << "\n";
          ofile << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
              << POD.PsPhiPOD.at((y*Nx)+x).at(i) << " ";
             } // end for x
//std::cout<< "I'm writing Pp" << (y*Nx) << " = i and j = "
//<< i << "\n";
          ofile << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
              << POD.PsPhiPOD.at(y*Nx).at(i) << "\n";
          } // end for y

       ofile.close();
       } // end for i

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputPTBasis(int Nbasis, std::string ln)

   {

   int x;
   int y;
   int i;

   char c = '.';

   std::vector< double > tmp(Nbasis, 0.0);
   std::vector< std::vector< double > > TsPOD(Nx*Ny, tmp);

   if (ln.size() == 0)
      ln = c;
   for (i=0; i < Nbasis; i++)
      {

      std::string fn = ln +
       "/ParticleTemperatureBasis" + intToString(i) + ".txt";
```

```
        std::ifstream ifile (fn.c_str());

        for (y=Ny-1; y >=0; y--)
            for (x=Nx-1; x >= 0; x--)
                ifile >> TsPOD.at((y*Nx)+x).at(i) ;

        ifile.close();
        } // end for i

    POD.TsPhiPOD = TsPOD;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputPTBasis(int Nbasis)

    {

    int x;
    int y;
    int i;

    for (i=0; i < Nbasis; i++)
        {

        std::string fn =
         "./Output/ParticleTemperatureBasis" + intToString(i) + ".txt";
        std::ofstream ofile (fn.c_str());

        for (y=Ny-1; y >=0; y--)
            {
            for (x=Nx-1; x > 0; x--)
                {
                ofile << std::setiosflags(std::ios::fixed)
                      << std::setprecision(15)
                      << POD.TsPhiPOD.at((y*Nx)+x).at(i) << " ";
                } // end for x
            ofile << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << POD.TsPhiPOD.at(y*Nx).at(i) << "\n";
            } // end for y

        ofile.close();
```

```
        } // end for i

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputBasis(int NGU, int NGV, int NPU, int NPV, int NGP,
                    int NGF, int NPP, int NPS, std::string ln)

    {

//    if (Id == (0 % NIds))
        InputGUBasis(NGU, ln);

//      std::cout << "Read Ug Basis \n";

//    if (Id == (1 % NIds))
        InputGVBasis(NGV, ln);

//      std::cout << "Read Vg Basis \n";

//    if (Id == (2 % NIds))
        InputPUBasis(NPU, ln);

//      std::cout << "Read Us Basis \n";

//    if (Id == (3 % NIds))
        InputPVBasis(NPV, ln);

//      std::cout << "Read Vs Basis \n";

//    if (Id == (4 % NIds))
        InputGPBasis(NGP, ln);

//      std::cout << "Read Pg Basis \n";

//    if (Id == (5 % NIds))
        InputGFBasis(NGF, ln);

//      std::cout << "Read Eg Basis \n";

//    if (Id == (6 % NIds))
        InputPPBasis(NPP, ln);
```

```
//      std::cout << "Read Ps Basis \n";

//    if (Id == (7 % NIds))
        InputPTBasis(NPS, ln);

      std::cout << "Read Basis \n";

    }


/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputPODs(int NGU, int NGV, int NPU, int NPV,
                         int NGP, int NGF, int NPP, int NPT,
                         std::string ln)

    {


    int k;
    int t;
    int NT = GetNt();
    char c = '.';
    std::vector< double > tmp0 (NGU, 0.0);
    std::vector< double > tmp1 (NGV, 0.0);
    std::vector< double > tmp2 (NPU, 0.0);
    std::vector< double > tmp3 (NPV, 0.0);
    std::vector< double > tmp5 (NGF, 0.0);
    std::vector< double > tmp4 (NGP, 0.0);
    std::vector< double > tmp6 (NPP, 0.0);
    std::vector< double > tmp7 (NPT, 0.0);

    if (ln.size() == 0)
        ln = c;

    std::string fegx = ln + "/PODsGasVelocityX.txt";
    std::ifstream ifilegx (fegx.c_str());
    std::vector< std::vector< double > > PODsGasU (NT, tmp0);
    for (k=0; k < NGU; k++)
        for (t=0; t < NT; t++)
            ifilegx >> PODsGasU.at(t).at(k);
    ifilegx.close();
    POD.PODsGasU = PODsGasU;
    std::cout << "Input GasU PODs \n";
```

```
std::string fegy = ln + "/PODsGasVelocityY.txt";
std::ifstream ifilegy (fegy.c_str());
std::vector< std::vector< double > > PODsGasV (NT, tmp1);
for (k=0; k < NGV; k++)
   for (t=0; t < NT; t++)
         ifilegy >> PODsGasV.at(t).at(k);
ifilegy.close();
POD.PODsGasV = PODsGasV;
std::cout << "Input GasV PODs \n";

std::string fesx = ln + "/PODsParticleVelocityX.txt";
std::ifstream ifilesx (fesx.c_str());
std::vector< std::vector< double > > PODsParticleU (NT, tmp2);
for (k=0; k < NPU; k++)
   for (t=0; t < NT; t++)
         ifilesx >> PODsParticleU.at(t).at(k);
ifilesx.close();
POD.PODsParticleU = PODsParticleU;
std::cout << "Input ParticleU PODs \n";


std::string fesy = ln + "/PODsParticleVelocityY.txt";
std::ifstream ifilesy (fesy.c_str());
std::vector< std::vector< double > > PODsParticleV (NT, tmp3);
for (k=0; k < NPV; k++)
   for (t=0; t < NT; t++)
      ifilesy >> PODsParticleV.at(t).at(k);
ifilesy.close();
POD.PODsParticleV = PODsParticleV;
std::cout << "Input ParticleV PODs \n";

std::string fegf = ln + "/PODsGasFF.txt";
std::ifstream ifilegf (fegf.c_str());
std::vector< std::vector< double > > PODsGasF (NT, tmp5);
for (k=0; k < NGF; k++)
   for (t=0; t < NT; t++)
      ifilegf >> PODsGasF.at(t).at(k);
ifilegf.close();
POD.PODsGasF = PODsGasF;
std::cout << "Input GasF PODs \n";

std::string fegp = ln + "/PODsGasPressure.txt";
std::ifstream ifilegp (fegp.c_str());
std::vector< std::vector< double > > PODsGasP (NT, tmp4);
```

```
    for (k=0; k < NGP; k++)
      for (t=0; t < NT; t++)
          ifilegp >> PODsGasP.at(t).at(k);
    ifilegp.close();
    POD.PODsGasP = PODsGasP;
    std::cout << "Input GasP PODs \n";

    std::string fesp = ln + "/PODsParticlePressure.txt";
    std::ifstream ifilesp (fesp.c_str());
    std::vector< std::vector< double > > PODsParticleP (NT, tmp6);
    for (k=0; k < NPP; k++)
      for (t=0; t < NT; t++)
          ifilesp >> PODsParticleP.at(t).at(k);
    ifilesp.close();
    POD.PODsParticleP = PODsParticleP;
    std::cout << "Input ParticleP PODs \n";

    std::string fest = ln + "/PODsParticleTemperature.txt";
    std::ifstream ifilest (fest.c_str());
    std::vector< std::vector< double > > PODsParticleT (NT, tmp7);
    for (k=0; k < NPT; k++)
      for (t=0; t < NT; t++)
          ifilest >> PODsParticleT.at(t).at(k) ;
    ifilest.close();
    POD.PODsParticleT = PODsParticleT;
    std::cout << "Input ParticleT PODs \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputPODsSplit(int NGU, int NGV, int NPU, int NPV,
                          int NGP, int NGF, int NPP, int NPT,
                          int Id, int NIds)

    {



    int k;
    int t;

    if (Id == (0 % NIds))
      {
      std::string feg = "./Output/PODsGasVelocityX.txt";
```

```
    std::ofstream ofileg (feg.c_str());
    for (k=0; k < NGU; k++)
        {
        for (t=0; t < GetNt() - 1; t++)
            ofileg << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << POD.PODsGasU.at(t).at(k) << " ";
        ofileg << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << POD.PODsGasU.at(t).at(k) << " \n";
        }
    ofileg.close();
    std::cout << Id << " wrote PODs \n";
    } // end ID

 if (Id == (1 % NIds))
    {
    std::string feg = "./Output/PODsGasVelocityY.txt";
    std::ofstream ofileg (feg.c_str());
    for (k=0; k < NGV; k++)
        {
        for (t=0; t < GetNt() - 1; t++)
            ofileg << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << POD.PODsGasV.at(t).at(k) << " ";
        ofileg << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << POD.PODsGasV.at(t).at(k) << " \n";
        }
    ofileg.close();
    std::cout << Id << " wrote PODs \n";
    } // end ID

  if (Id == (2 % NIds))
    {
    std::string fes = "./Output/PODsParticleVelocityX.txt";
    std::ofstream ofiles (fes.c_str());
    for (k=0; k < NPU; k++)
        {
        for (t=0; t < GetNt() - 1; t++)
            ofiles << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << POD.PODsParticleU.at(t).at(k) << " ";
        ofiles << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
```

```
            << POD.PODsParticleU.at(t).at(k) << "\n";
        }
    ofiles.close();
    std::cout << Id << " wrote PODs \n";
    } // end ID

 if (Id == (3 % NIds))
    {
    std::string fes = "./Output/PODsParticleVelocityY.txt";
    std::ofstream ofiles (fes.c_str());
    for (k=0; k < NPV; k++)
        {
        for (t=0; t < GetNt() - 1; t++)
            ofiles << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << POD.PODsParticleV.at(t).at(k) << " ";
        ofiles << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << POD.PODsParticleV.at(t).at(k) << "\n";
        }
    ofiles.close();
    std::cout << Id << " wrote PODs \n";
    } // end ID

if (Id == (5 % NIds))
    {
    std::string fegf = "./Output/PODsGasFF.txt";
    std::ofstream ofilegf (fegf.c_str());
    for (k=0; k < NGF; k++)
        {
        for (t=0; t < GetNt() - 1; t++)
            ofilegf << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << POD.PODsGasF.at(t).at(k) << " ";
        ofilegf << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << POD.PODsGasF.at(t).at(k) << "\n";
        }
    ofilegf.close();
    std::cout << Id << " wrote PODs \n";
    } // enD 5

if (Id == (4 % NIds))
    {
    std::string fegp = "./Output/PODsGasPressure.txt";
```

```
    std::ofstream ofilegp (fegp.c_str());
    for (k=0; k < NGP; k++)
       {
       for (t=0; t < GetNt()  - 1; t++)
          ofilegp << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << POD.PODsGasP.at(t).at(k) << " ";
       ofilegp << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << POD.PODsGasP.at(t).at(k) << "\n";
       }
    ofilegp.close();
    std::cout << Id << " wrote PODs \n";
    } // END 4

 if (Id == (6 % NIds))
    {
    std::string fesp = "./Output/PODsParticlePressure.txt";
    std::ofstream ofilesp (fesp.c_str());
    for (k=0; k < NPP; k++)
       {
       for (t=0; t < GetNt()  - 1; t++)
          ofilesp << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << POD.PODsParticleP.at(t).at(k) << " ";
       ofilesp << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << POD.PODsParticleP.at(t).at(k) << "\n";
       }
    ofilesp.close();
    std::cout << Id << " wrote PODs \n";
    } // end 6

 if (Id == (7 % NIds))
    {
    std::string fest = "./Output/PODsParticleTemperature.txt";
    std::ofstream ofilest (fest.c_str());
    for (k=0; k < NPT; k++)
       {
       for (t=0; t < GetNt() - 1; t++)
          ofilest << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << POD.PODsParticleT.at(t).at(k) << " ";
       ofilest << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
```

```
            << POD.PODsParticleT.at(t).at(k) << "\n";
        }
      ofilest.close();
      std::cout << Id << " wrote PODs \n";
      } // end 7

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputErrorSplit(int Id, int NIds)

   {

   int x;
   int y;
   int t;

   double minUg, maxUg, avgUg;
   double minVg, maxVg, avgVg;
   double minUs, maxUs, avgUs;
   double minVs, maxVs, avgVs;

   double minEg, maxEg, avgEg;
   double minPg, maxPg, avgPg;
   double minPs, maxPs, avgPs;
   double minTs, maxTs, avgTs;

   if (Id == (0 % NIds))
      {
      std::string efUgs = "./Output/UgErrorSummary.txt";
      std::ofstream efUg (efUgs.c_str());
      // Create a text file - one per timestep
      //writing file
      for (t=0; t < Nt; t++)
         {
         minUg = ErrorUg.at(t).at(Ny-1).at(Nx-1);
         maxUg = ErrorUg.at(t).at(Ny-1).at(Nx-1);
         avgUg = 0.0;
         std::string feg =
         "./Output/GasVelocityXError" + intToString(t) + ".txt";
         std::ofstream ofileg (feg.c_str());
         for (y=Ny-1; y >=0; y--)
            {
```

```
            for (x=Nx-1; x > 0; x--)
               {
               ofileg << std::setiosflags(std::ios::fixed)
                      << std::setprecision(15)
                      << ErrorUg.at(t).at(y).at(x) << " ";
               if (ErrorUg.at(t).at(y).at(x) > maxUg)
                  maxUg = ErrorUg.at(t).at(y).at(x);
               if (ErrorUg.at(t).at(y).at(x) < minUg)
                  minUg = ErrorUg.at(t).at(y).at(x);
               avgUg = avgUg + ErrorUg.at(t).at(y).at(x);
               }
            ofileg << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << ErrorUg.at(t).at(y).at(0) << "\n";
            if (ErrorUg.at(t).at(y).at(0) > maxUg)
               maxUg = ErrorUg.at(t).at(y).at(0);
            if (ErrorUg.at(t).at(y).at(0) < minUg)
               minUg = ErrorUg.at(t).at(y).at(0);
            avgUg = avgUg + ErrorUg.at(t).at(y).at(0);
            }
         ofileg.close();
         avgUg = avgUg / (((double)Nx) * ((double)Ny));
         efUg << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
              << minUg << " " << avgUg << " " << maxUg << "\n";
         } // end for t
      efUg.close();
      } // end 0
   if (Id == (1 % NIds))
      {
      std::string efVgs = "./Output/VgErrorSummary.txt";
      std::ofstream efVg (efVgs.c_str());
      for (t=0; t < Nt; t++)
         {
         minVg = ErrorVg.at(t).at(Ny-1).at(Nx-1);
         maxVg = ErrorVg.at(t).at(Ny-1).at(Nx-1);
         avgVg = 0.0;
         std::string feg =
         "./Output/GasVelocityYError" + intToString(t) + ".txt";
         std::ofstream ofileg (feg.c_str());
         for (y=Ny-1; y >=0; y--)
            {
            for (x=Nx-1; x > 0; x--)
               {
               ofileg << std::setiosflags(std::ios::fixed)
```

```
                    << std::setprecision(15)
                    << ErrorVg.at(t).at(y).at(x) << " ";
                if (ErrorVg.at(t).at(y).at(x) > maxVg)
                    maxVg = ErrorVg.at(t).at(y).at(x);
                if (ErrorVg.at(t).at(y).at(x) < minVg)
                    minVg = ErrorVg.at(t).at(y).at(x);
                avgVg = avgVg + ErrorVg.at(t).at(y).at(x);
                }
            ofileg << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << ErrorVg.at(t).at(y).at(0) << "\n";
            if (ErrorVg.at(t).at(y).at(0) > maxVg)
                maxVg = ErrorVg.at(t).at(y).at(0);
            if (ErrorVg.at(t).at(y).at(0) < minVg)
                minVg = ErrorVg.at(t).at(y).at(0);
            avgVg = avgVg + ErrorVg.at(t).at(y).at(0);
            }
        ofileg.close();
        avgVg = avgVg / (((double)Nx) * ((double)Ny));
        efVg << std::setiosflags(std::ios::fixed)
             << std::setprecision(15)
             << minVg << " " << avgVg << " " << maxVg << "\n";
        } // end for t
    efVg.close();
    } // end 1
if (Id == (2 % NIds))
    {
    std::string efUss = "./Output/UsErrorSummary.txt";
    std::ofstream efUs (efUss.c_str());
    for (t=0; t < Nt; t++)
        {
        minUs = ErrorUs.at(t).at(Ny-1).at(Nx-1);
        maxUs = ErrorUs.at(t).at(Ny-1).at(Nx-1);
        avgUs = 0.0;
        std::string fes =
        "./Output/ParticleVelocityXError" + intToString(t) + ".txt";
        std::ofstream ofiles (fes.c_str());
        for (y=Ny-1; y >=0; y--)
            {
            for (x=Nx-1; x > 0; x--)
                {
                ofiles << std::setiosflags(std::ios::fixed)
                       << std::setprecision(15)
                       << ErrorUs.at(t).at(y).at(x) << " ";
                if (ErrorUs.at(t).at(y).at(x) > maxUs)
```

```
                maxUs = ErrorUs.at(t).at(y).at(x);
              if (ErrorUs.at(t).at(y).at(x) < minUs)
                minUs = ErrorUs.at(t).at(y).at(x);
              avgUs = avgUs + ErrorUs.at(t).at(y).at(x);
              }
            ofiles << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << ErrorUs.at(t).at(y).at(0) << "\n";
            if (ErrorUs.at(t).at(y).at(0) > maxUs)
              maxUs = ErrorUs.at(t).at(y).at(0);
            if (ErrorUs.at(t).at(y).at(0) < minUs)
              minUs = ErrorUs.at(t).at(y).at(0);
            avgUs = avgUs + ErrorUs.at(t).at(y).at(0);
            }
          ofiles.close();
          avgUs = avgUs / (((double)Nx) * ((double)Ny));
          efUs << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
              << minUs << " " << avgUs << " " << maxUs << "\n";
          } // end for t
        efUs.close();
        } // end for 2
    if (Id == (3 % NIds))
        {
        std::string efVss = "./Output/VsErrorSummary.txt";
        std::ofstream efVs (efVss.c_str());
        for (t=0; t < Nt; t++)
          {
          minVs = ErrorVs.at(t).at(Ny-1).at(Nx-1);
          maxVs = ErrorVs.at(t).at(Ny-1).at(Nx-1);
          avgVs = 0.0;
          std::string fes =
          "./Output/ParticleVelocityYError" + intToString(t) + ".txt";
          std::ofstream ofiles (fes.c_str());
          for (y=Ny-1; y >=0; y--)
            {
            for (x=Nx-1; x > 0; x--)
              {
              ofiles << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << ErrorVs.at(t).at(y).at(x) << " ";
              if (ErrorVs.at(t).at(y).at(x) > maxVs)
                maxVs = ErrorVs.at(t).at(y).at(x);
              if (ErrorVs.at(t).at(y).at(x) < minVs)
                minVs = ErrorVs.at(t).at(y).at(x);
```

```
            avgVs = avgVs + ErrorVs.at(t).at(y).at(x);
            }
          ofiles << std::setiosflags(std::ios::fixed)
                 << std::setprecision(15)
                 << ErrorVs.at(t).at(y).at(0) << "\n";
          if (ErrorVs.at(t).at(y).at(0) > maxVs)
            maxVs = ErrorVs.at(t).at(y).at(0);
          if (ErrorVs.at(t).at(y).at(0) < minVs)
            minVs = ErrorVs.at(t).at(y).at(0);
          avgVs = ErrorVs.at(t).at(y).at(0);
          }
        ofiles.close();
        avgVs = avgVs / (((double)Nx) * ((double)Ny));
        efVs << std::setiosflags(std::ios::fixed)
             << std::setprecision(15)
             << minVs << " " << avgVs << " " << maxVs << "\n";
        } // end for t
      efVs.close();
      } // end for 3
  if (Id == (4 % NIds))
      {
      std::string efPgs = "./Output/PgErrorSummary.txt";
      std::ofstream efPg (efPgs.c_str());
      for (t=0; t < Nt; t++)
          {
          minPg = ErrorPg.at(t).at(Ny-1).at(Nx-1);
          maxPg = ErrorPg.at(t).at(Ny-1).at(Nx-1);
          avgPg = 0.0;
          std::string fegp =
          "./Output/GasPressureError" + intToString(t) + ".txt";
          std::ofstream ofilegp (fegp.c_str());
          for (y=Ny-1; y >=0; y--)
              {
              for (x=Nx-1; x > 0; x--)
                  {
                  ofilegp << std::setiosflags(std::ios::fixed)
                          << std::setprecision(15)
                          << ErrorPg.at(t).at(y).at(x) << " ";
                  if (ErrorPg.at(t).at(y).at(x) > maxPg)
                      maxPg = ErrorPg.at(t).at(y).at(x);
                  if (ErrorPg.at(t).at(y).at(x) < minPg)
                      minPg = ErrorPg.at(t).at(y).at(x);
                  avgPg = avgPg + ErrorPg.at(t).at(y).at(x);
                  }
              ofilegp << std::setiosflags(std::ios::fixed)
```

```
                << std::setprecision(15)
                << ErrorPg.at(t).at(y).at(0) << "\n";
            if (ErrorPg.at(t).at(y).at(0) > maxPg)
               maxPg = ErrorPg.at(t).at(y).at(0);
            if (ErrorPg.at(t).at(y).at(0) < minPg)
               minPg = ErrorPg.at(t).at(y).at(0);
            avgPg = avgPg + ErrorPg.at(t).at(y).at(0);
            }
         ofilegp.close();
         avgPg = avgPg / (((double)Nx) * ((double)Ny));
         efPg << std::setiosflags(std::ios::fixed)
             << std::setprecision(15)
             << minPg << " " << avgPg << " " << maxPg << "\n";
         } // end for t
      efPg.close();
      } // end 4
   if (Id == (5 % NIds))
      {
      std::string efEgs = "./Output/EgErrorSummary.txt";
      std::ofstream efEg (efEgs.c_str());
      for (t=0; t < Nt; t++)
         {
         minEg = ErrorFg.at(t).at(Ny-1).at(Nx-1);
         maxEg = ErrorFg.at(t).at(Ny-1).at(Nx-1);
         avgEg = 0.0;
         std::string fegf =
         "./Output/GasFFError" + intToString(t) + ".txt";
         std::ofstream ofilegf (fegf.c_str());
         for (y=Ny-1; y >=0; y--)
            {
            for (x=Nx-1; x > 0; x--)
               {
               ofilegf << std::setiosflags(std::ios::fixed)
                     << std::setprecision(15)
                     << ErrorFg.at(t).at(y).at(x) << " ";
               if (ErrorFg.at(t).at(y).at(x) > maxEg)
                  maxEg = ErrorFg.at(t).at(y).at(x);
               if (ErrorFg.at(t).at(y).at(x) < minEg)
                  minEg = ErrorFg.at(t).at(y).at(x);
               avgEg = avgEg + ErrorFg.at(t).at(y).at(x);
               }
            ofilegf << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << ErrorFg.at(t).at(y).at(0) << "\n";
            if (ErrorFg.at(t).at(y).at(0) > maxEg)
```

```
            maxEg = ErrorFg.at(t).at(y).at(0);
          if (ErrorFg.at(t).at(y).at(0) < minEg)
            minEg = ErrorFg.at(t).at(y).at(0);
          avgEg = avgEg + ErrorFg.at(t).at(y).at(0);
          }
        ofilegf.close();
        avgEg = avgEg / (((double)Nx) * ((double)Ny));
        efEg << std::setiosflags(std::ios::fixed)
             << std::setprecision(15)
             << minEg << " " << avgEg << " " << maxEg << "\n";
        } // end for t
      efEg.close();
      } // end 5
  if (Id == (6 % NIds))
      {
      std::string efPss = "./Output/PsErrorSummary.txt";
      std::ofstream efPs (efPss.c_str());
      for (t=0; t < Nt; t++)
        {
        minPs = ErrorPs.at(t).at(Ny-1).at(Nx-1);
        maxPs = ErrorPs.at(t).at(Ny-1).at(Nx-1);
        avgPs = 0.0;
        std::string fesp =
        "./Output/ParticlePressureError" + intToString(t) + ".txt";
        std::ofstream ofilesp (fesp.c_str());
        for (y=Ny-1; y >=0; y--)
          {
          for (x=Nx-1; x > 0; x--)
            {
            ofilesp << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << ErrorPs.at(t).at(y).at(x) << " ";
            if (ErrorPs.at(t).at(y).at(x) > maxPs)
              maxPs = ErrorPs.at(t).at(y).at(x);
            if (ErrorPs.at(t).at(y).at(x) < minPs)
              minPs = ErrorPs.at(t).at(y).at(x);
            avgPs = avgPs + ErrorPs.at(t).at(y).at(x);
            }
          ofilesp << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << ErrorPs.at(t).at(y).at(0) << "\n";
          if (ErrorPs.at(t).at(y).at(0) > maxPs)
            maxPs = ErrorPs.at(t).at(y).at(0);
          if (ErrorPs.at(t).at(y).at(0) < minPs)
            minPs = ErrorPs.at(t).at(y).at(0);
```

```
            avgPs = avgPs + ErrorPs.at(t).at(y).at(0);
            }
         ofilesp.close();
         avgPs = avgPs / (((double)Nx) * ((double)Ny));
         efPs << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
              << minPs << " " << avgPs << " " << maxPs << "\n";
         } // end for t
      efPs.close();
      } // end 6
   if (Id == (7 % NIds))
      {
      std::string efTss = "./Output/TsErrorSummary.txt";
      std::ofstream efTs (efTss.c_str());
      for (t=0; t < Nt; t++)
         {
         maxTs = ErrorTs.at(t).at(Ny-1).at(Nx-1);
         minTs = ErrorTs.at(t).at(Ny-1).at(Nx-1);
         avgTs = 0.0;
         std::string fest =
         "./Output/ParticleTemperatureError" + intToString(t) + ".txt";
         std::ofstream ofilest (fest.c_str());
         for (y=Ny-1; y >=0; y--)
            {
            for (x=Nx-1; x > 0; x--)
               {
               ofilest << std::setiosflags(std::ios::fixed)
                       << std::setprecision(15)
                       << ErrorTs.at(t).at(y).at(x) << " ";
               if (ErrorTs.at(t).at(y).at(x) > maxTs)
                  maxTs = ErrorTs.at(t).at(y).at(x);
               if (ErrorTs.at(t).at(y).at(x) < minTs)
                  minTs = ErrorTs.at(t).at(y).at(x);
               avgTs = avgTs + ErrorTs.at(t).at(y).at(x);
               }
            ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << ErrorTs.at(t).at(y).at(0) << "\n";
            if (ErrorTs.at(t).at(y).at(0) > maxTs)
               maxTs = ErrorTs.at(t).at(y).at(0);
            if (ErrorTs.at(t).at(y).at(0) < minTs)
               minTs = ErrorTs.at(t).at(y).at(0);
            avgTs = avgTs + ErrorTs.at(t).at(y).at(0);
            }
         ofilest.close();
```

```
        avgTs = avgTs / (((double)Nx) * ((double)Ny));
        efTs << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << minTs << " " << avgTs << " " << maxTs << "\n";
        } // end for t
      efTs.close();
      } // end 7

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double PODSolver::CalcEnergySVD(std::vector< double > & Sigma, int & N,
                         int index)

   {

   int i;
   int NSigma;
   double e;
   double num;
   double SumSVs;
   SumSVs = 0.0;
   num = 0.0;

   NSigma = Sigma.size();

   for (i=0; i < NSigma; i++)
      SumSVs = SumSVs + (Sigma.at(i) * Sigma.at(i));
   i = 0;
   do
     {
     num = num + (Sigma.at(i) * Sigma.at(i));
     e = num / SumSVs;
     i++;
     } while ((energy.at(index) > e) && (i < NSigma));

   N = i;
   return e;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
bool PODSolver::FormSVDPOD(SimVars & Data, std::vector< double > & e,
                           int Id, int NIds)

    {

    bool good = true;

    std::vector< double > Dg;
    std::vector< double > ODg;
    std::vector< double > Ds;
    std::vector< double > ODs;

    std::vector< double > DFg;
    std::vector< double > ODFg;
    std::vector< double > DPg;
    std::vector< double > ODPg;
    std::vector< double > DPs;
    std::vector< double > ODPs;
    std::vector< double > DTs;
    std::vector< double > ODTs;

    std::vector< int > GRank;
    std::vector< int > PRank;

    std::vector< int > TsRank;
    std::vector< int > PsRank;
    std::vector< int > PgRank;
    std::vector< int > FgRank;

    std::vector< int > GPODevs;
    std::vector< int > PPODevs;

    std::vector< int > PSODevs;
    std::vector< int > PGODevs;
    std::vector< int > FGODevs;
    std::vector< int > TSODevs;

    energy = e;

    int GUNBasis;
    int GVNBasis;
    int PUNBasis;
    int PVNBasis;
    int GFNBasis;
```

```
int GPNBasis;
int PPNBasis;
int TSNBasis;

int i;
int j;
int t;

double Gceu;
double Gcev;
double Pceu;
double Pcev;
double Fgce;
double Pgce;
double Psce;
double Tsce;

std::vector< std::vector< double > > MU;
std::vector< std::vector< double > > MV0;
std::vector< std::vector< double > > MV1;
std::vector< std::vector< double > > MV2;
std::vector< std::vector< double > > MV3;
std::vector< std::vector< double > > MV4;
std::vector< std::vector< double > > MV5;
std::vector< std::vector< double > > MV6;
std::vector< std::vector< double > > MV7;

std::vector< double > SigmaUg;
std::vector< double > SigmaVg;
std::vector< double > SigmaUs;
std::vector< double > SigmaVs;
std::vector< double > SigmaPg;
std::vector< double > SigmaPs;
std::vector< double > SigmaFg;
std::vector< double > SigmaTs;
std::vector< double > SigmaG;
std::vector< double > SigmaP;

std::vector<double> tmp;

std::vector< std::vector< double > > GAu;
std::vector< std::vector< double > > GAv;
std::vector< std::vector< double > > PAu;
std::vector< std::vector< double > > PAv;
```

```
    std::vector< std::vector< double > > GA;
    std::vector< std::vector< double > > PA;

    std::vector< std::vector< double > > GAP;
    std::vector< std::vector< double > > GAF;
    std::vector< std::vector< double > > PAP;
    std::vector< std::vector< double > > PAT;

    std::vector< double > tGAu;
    std::vector< double > tGAv;
    std::vector< double > tPAu;
    std::vector< double > tPAv;

    std::vector< double > tGA;
    std::vector< double > tPA;

    std::vector< double > tGAP;
    std::vector< double > tGAF;
    std::vector< double > tPAP;
    std::vector< double > tPAT;

    std::vector<double> Gub;
    std::vector<double> Gvb;
    std::vector<double> Pub;
    std::vector<double> Pvb;

    std::vector<double> Gpb;
    std::vector<double> Gfb;
    std::vector<double> Psb;
    std::vector<double> Tsb;

    std::vector< std::vector< double > > U;
    std::vector< std::vector< double > > V;
    std::vector< double > W;
    std::vector< double > X;

//    int NumCVars = 6;
    int k;

    POD.InitPODData(Data.GetNx(), Data.GetNy(), Data.GetNt(),
                Data.Getdelx(), Data.Getdely(), Data.Getdelt());

    // Calculates mean cell velocities for Gas and Solids
    // As well as Temperature and Pressure
    GetMeanVars(Data);
```

```
    if (Id == 0)
       {
       std::cout << "Going to output the means.  \n";
       OutputMeans();
//       OutputMFIX(Data);
       }

    std::cout<< "Have Mean Velocities... "
             << Id << "\n";

    // Forms covariance matrices for both Gas and Solids & Combo
//    GetCorrMatrices(Data);
    for (j=0; j < Ny; j++)
       for (i=0; i < Nx; i++)
          {
          for (t=0;t < Nt; t++)
             {
             if (Id == (0 % NIds))
                tGAu.push_back(Data.Gas.GetU(t, i, j) -
                POD.UgAvgVel.at(i).at(j));
             if (Id == (1 % NIds))
                tGAv.push_back(Data.Gas.GetV(t, i, j) -
                POD.VgAvgVel.at(i).at(j));
             if (Id == (2 % NIds))
                tPAu.push_back(Data.Particle.GetU(t, i, j) -
                POD.UsAvgVel.at(i).at(j));
             if (Id == (3 % NIds))
                tPAv.push_back(Data.Particle.GetV(t, i, j) -
                POD.VsAvgVel.at(i).at(j));
             if (Id == (4 % NIds))
                tGAP.push_back(Data.Gas.GetP(t, i, j) -
                POD.PgAvg.at(i).at(j));
             if (Id == (5 % NIds))
                tGAF.push_back(Data.Gas.GetEP(t, i, j) -
                POD.FgAvg.at(i).at(j));
             if (Id == (6 % NIds))
                tPAP.push_back(Data.Particle.GetP(t, i, j) -
                POD.PsAvg.at(i).at(j));
             if (Id == (7 % NIds))
                tPAT.push_back(Data.Particle.GetTheta(t, i, j) -
                POD.TsAvg.at(i).at(j));
             } // end for t
          if (Id == (1 % NIds))
             {
```

```
            GAv.push_back(tGAv);
            tGAv.clear();
            }
        if (Id == (0 % NIds))
            {
            GAu.push_back(tGAu);
            tGAu.clear();
            }
        if (Id == (2 % NIds))
            {
            PAu.push_back(tPAu);
            tPAu.clear();
            }
        if (Id == (3 % NIds))
            {
            PAv.push_back(tPAv);
            tPAv.clear();
            }
        if (Id == (6 % NIds))
            {
            PAP.push_back(tPAP);
            tPAP.clear();
            }
        if (Id == (7 % NIds))
            {
            PAT.push_back(tPAT);
            tPAT.clear();
            }
        if (Id == (4 % NIds))
            {
            GAP.push_back(tGAP);
            tGAP.clear();
            }
        if (Id == (5 % NIds))
            {
            GAF.push_back(tGAF);
            tGAF.clear();
            }
        } // end for i and j

    std::cout<< "Getting POD basis using SVD Decomposition "
             << Id << "\n";

std::cout << Nt << " is the size of the matrix... \n";
    if (Id == (0 % NIds))
```

```
        {
        // POD Basis values are returned in the first
        // GUNBasis vectors of U = GPhiuPOD
        svd(GAu,POD.GPhiuPOD, MV0, SigmaUg, Nx*Ny, Nt);
        AdjustSingVals(SigmaUg, Nt);
        Gceu = CalcEnergySVD(SigmaUg, GUNBasis,0);
        OutputSingVals(SigmaUg, 0);
        POD.SetNUG(GUNBasis);
        std::cout << Gceu
                <<" captured energy of the Gas x velocity in "
             << GUNBasis << " basis functions.\n";
        // Keeping only the first m to form the basis
//        for (p=Nt; p > GUNBasis; p--)
//           GPhiuPOD.pop_back();
        std::cout << GUNBasis << " is how many I should have. \n "
                << POD.GPhiuPOD.size() << " is how many I have. \n"
                << "Singular value spread is "
                << SigmaUg.at(0) << " to " << SigmaUg.at(GUNBasis-1)
                << " . \n";
        OutputGUBasis(GUNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        // Solving for the coefficients
        POD.PODsGasU.clear();
//        svd(POD.GPhiuPOD, U, V, W, Nx*Ny, GUNBasis);
//        AdjustSingVals(W, GUNBasis);
        X.clear();
        for (t = 0; t < Nt; t++)
           { //Assuming svd outputs V transpose and not V
//           std::cout << t << " POD coefficient calculation \n";
//           FormGub(Gub, t, Data);
//           std::cout << Id << " formed solution \n";
//           std::cout << Id << " survived coeff svd \n";
//           std::cout << Id << " adjusting SVs again \n";
//           Solve4coefs(U, V, W, Gub, Nx*Ny, GUNBasis, X);
//           std::cout << Id << "Have POD coefficients \n";
           for (k=0; k < GUNBasis; k++)
              X.push_back(SigmaUg.at(k) * MV0.at(t).at(k));
           POD.PODsGasU.push_back(X);
           X.clear();
           } // end t
        }
     if (Id == (1 % NIds))
        {
        svd(GAv,POD.GPhivPOD, MV1, SigmaVg, Nx*Ny, Nt);
        AdjustSingVals(SigmaVg, Nt);
```

```
        Gcev = CalcEnergySVD(SigmaVg, GVNBasis, 1);
        OutputSingVals(SigmaVg, 1);
        POD.SetNVG(GVNBasis);
        std::cout << Gcev
                <<" captured energy of the Gas y velocity in "
             << GVNBasis << " basis functions.\n";
//        for (p=Nt; p > GVNBasis; p--)
//           GPhivPOD.pop_back();
        OutputGVBasis(GVNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << GVNBasis << " is how many I should have and "
                << POD.GPhivPOD.size() << " is how many I have with a "
                << SigmaVg.at(0) << " to " << SigmaVg.at(GVNBasis-1)
                << " spread.";
        POD.PODsGasV.clear();
//        svd(POD.GPhivPOD, U, V, W, Nx*Ny, GVNBasis);
//        AdjustSingVals(W, GVNBasis);
        X.clear();
        for (t = 0; t < Nt; t++)
           {
//           FormGvb(Gvb, t, Data);
//           Solve4coefs(U, V, W, Gvb, Nx*Ny, GVNBasis, X);
           for (k=0; k < GVNBasis; k++)
              X.push_back(SigmaVg.at(k) * MV1.at(t).at(k));
           POD.PODsGasV.push_back(X);
           X.clear();
           } // end t
        }

     if (Id == (2 % NIds))
        {
        svd(PAu,POD.PPhiuPOD, MV2, SigmaUs, Nx*Ny, Nt);
        AdjustSingVals(SigmaUs, Nt);
        Pceu = CalcEnergySVD(SigmaUs, PUNBasis, 2);
        OutputSingVals(SigmaUs, 2);
        POD.SetNUS(PUNBasis);
        std::cout << Pceu
                <<" captured energy of the Particle x velocity in "
             << PUNBasis << " basis functions.\n";
//        for (p=Nt; p > PUNBasis; p--)
//           PPhiuPOD.pop_back();
        OutputPUBasis(PUNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << PUNBasis << " is how many I should have and "
                << POD.PPhiuPOD.size() << " is how many I have with a "
```

```
            << SigmaUs.at(0) << " to " << SigmaUs.at(PUNBasis-1)
            << " spread.";
      POD.PODsParticleU.clear();
//      svd(POD.PPhiuPOD, U, V, W, Nx*Ny, PUNBasis);
//      AdjustSingVals(W, PUNBasis);
      X.clear();
      for (t=0; t < Nt; t++)
         {
//         FormPub(Pub, t, Data);
//         Solve4coefs(U, V, W, Pub, Nx*Ny, PUNBasis, X);
         for (k=0; k < PUNBasis; k++)
            X.push_back(SigmaUs.at(k) * MV2.at(t).at(k));
         POD.PODsParticleU.push_back(X);
         X.clear();
         } // end t
      }

   if (Id == (3 % NIds))
      {
      svd(PAv,POD.PPhivPOD, MV3, SigmaVs, Nx*Ny, Nt);
      AdjustSingVals(SigmaVs, Nt);
      Pcev = CalcEnergySVD(SigmaVs, PVNBasis,3);
      OutputSingVals(SigmaVs, 3);
      POD.SetNVS(PVNBasis);
      std::cout << Pcev
            <<" captured energy of the Particle y velocity in "
         << PVNBasis << " basis functions.\n";
//      for (p=Nt; p > PVNBasis; p--)
//         PPhivPOD.pop_back();
      OutputPVBasis(PVNBasis);
      std::cout << "Output basis vectors via " << Id << " \n";
      std::cout << PVNBasis << " is how many I should have and "
            << POD.PPhivPOD.size() << " is how many I have with a "
            << SigmaVs.at(0) << " to " << SigmaVs.at(PVNBasis-1)
            << " spread.";
      POD.PODsParticleV.clear();
//      svd(POD.PPhivPOD, U, V, W, Nx*Ny, PVNBasis);
//      AdjustSingVals(W, PVNBasis);
      X.clear();
      for (t=0; t < Nt; t++)
         {
//         FormPvb(Pvb, t, Data);
//         Solve4coefs(U, V, W, Pvb, Nx*Ny, PVNBasis, X);
         for (k=0; k < PVNBasis; k++)
            X.push_back(SigmaVs.at(k) * MV3.at(t).at(k));
```

```
            POD.PODsParticleV.push_back(X);
            X.clear();
            } // end t
        }

    if (Id == (4 % NIds))
        {
//std::cout << "Starting Gas Pressure... \n";
        svd(GAP,POD.PgPhiPOD, MV4, SigmaPg, Nx*Ny, Nt);
        AdjustSingVals(SigmaPg, Nt);
        Pgce = CalcEnergySVD(SigmaPg, GPNBasis, 5);
        OutputSingVals(SigmaPg, 4);
        POD.SetNPG(GPNBasis);
        std::cout << Pgce
                <<" captured energy of the Gas pressure in "
              << GPNBasis << " basis functions.\n";
//        for (p=Nt; p > GPNBasis; p--)
//            PgPhiPOD.pop_back();
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << GPNBasis << " is how many I should have and "
                << POD.PgPhiPOD.size() << " is how many I have with a "
                << SigmaPg.at(0) << " to " << SigmaPg.at(GPNBasis-1)
                << " spread.";
        OutputGPBasis(GPNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << GPNBasis << " is how many I should have and "
                << POD.PgPhiPOD.size() << " is how many I have with a "
                << SigmaPg.at(0) << " to " << SigmaPg.at(GPNBasis-1)
                << " spread.";
        POD.PODsGasP.clear();
//        svd(POD.PgPhiPOD, U, V, W, Nx*Ny, GPNBasis);
//        AdjustSingVals(W, GPNBasis);
        X.clear();
        for (t=0; t < Nt; t++)
            {
//            FormGpb(Gpb, t, Data);
//            Solve4coefs(U, V, W, Gpb, Nx*Ny, GPNBasis, X);
            for (k=0; k < GPNBasis; k++)
                X.push_back(SigmaPg.at(k) * MV4.at(t).at(k));
            POD.PODsGasP.push_back(X);
            X.clear();
            } // end t
        }

    if (Id == (5 % NIds))
```

```
        {
        svd(GAF,POD.FgPhiPOD, MV5, SigmaFg, Nx*Ny, Nt);
        AdjustSingVals(SigmaFg, Nt);
        Fgce = CalcEnergySVD(SigmaFg, GFNBasis, 4);
        OutputSingVals(SigmaFg, 5);
        POD.SetNEG(GFNBasis);
        std::cout << Fgce
                <<" captured energy of the Gas fluid fraction in "
             << GFNBasis << " basis functions.\n";
//       for (p=Nt; p > GFNBasis; p--)
//          FgPhiPOD.pop_back();
        std::cout << GFNBasis << " is how many I should have and "
                << POD.FgPhiPOD.size() << " is how many I have with a "
                << SigmaFg.at(0) << " to " << SigmaFg.at(GFNBasis-1)
                << " spread.";
        OutputGFBasis(GFNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << GFNBasis << " is how many I should have and "
                << POD.FgPhiPOD.size() << " is how many I have with a "
                << SigmaFg.at(0) << " to " << SigmaFg.at(GFNBasis-1)
                << " spread.";
        POD.PODsGasF.clear();
//       svd(POD.FgPhiPOD, U, V, W, Nx*Ny, GFNBasis);
//       AdjustSingVals(W, GFNBasis);
        X.clear();
        for (t=0; t < Nt; t++)
           {
//          FormGfb(Gfb, t, Data);
//          Solve4coefs(U, V, W, Gfb, Nx*Ny, GFNBasis, X);
           for (k=0; k < GFNBasis; k++)
              X.push_back(SigmaFg.at(k) * MV5.at(t).at(k));
           POD.PODsGasF.push_back(X);
           X.clear();
           } // end t
        }

     if (Id == (6 % NIds))
        {
        svd(PAP,POD.PsPhiPOD, MV6, SigmaPs, Nx*Ny, Nt);
        AdjustSingVals(SigmaPs, Nt);
        Psce = CalcEnergySVD(SigmaPs, PPNBasis,6);
        OutputSingVals(SigmaPs, 6);
        POD.SetNPS(PPNBasis);
        std::cout << Psce
                <<" captured energy of the Particle pressure in "
```

```
            << PPNBasis << " basis functions.\n";
//        for (p=Nt; p > PPNBasis; p--)
//            PsPhiPOD.pop_back();
        OutputPPBasis(PPNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << PPNBasis << " is how many I should have and "
                << POD.PsPhiPOD.size() << " is how many I have with a "
                << SigmaPs.at(0) << " to " << SigmaPs.at(PPNBasis-1)
                << " spread.";
        POD.PODsParticleP.clear();
//        svd(POD.PsPhiPOD, U, V, W, Nx*Ny, PPNBasis);
//        AdjustSingVals(W, PPNBasis);
        X.clear();
        for (t=0; t < Nt; t++)
            {
//            FormPsb(Psb, t, Data);
//            Solve4coefs(U, V, W, Psb, Nx*Ny, PPNBasis, X);
            for (k=0; k < PPNBasis; k++)
                X.push_back(SigmaPs.at(k) * MV6.at(t).at(k));
            POD.PODsParticleP.push_back(X);
            X.clear();
            } // end t
        }

    if (Id == (7 % NIds))
        {
        svd(PAT,POD.TsPhiPOD, MV7, SigmaTs, Nx*Ny, Nt);
        AdjustSingVals(SigmaTs, Nt);
        Tsce = CalcEnergySVD(SigmaTs, TSNBasis, 7);
        OutputSingVals(SigmaTs, 7);
        POD.SetNTS(TSNBasis);
        std::cout << Tsce
                <<" captured energy of the Gas temperature in "
            << TSNBasis << " basis functions.\n";
//        for (p=Nt; p > TSNBasis; p--)
//            TsPhiPOD.pop_back();
        OutputPTBasis(TSNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << TSNBasis << " is how many I should have and "
                << POD.TsPhiPOD.size() << " is how many I have with a "
                << SigmaTs.at(0) << " to " << SigmaTs.at(TSNBasis-1)
                << " spread.";
        POD.PODsParticleT.clear();
//        svd(POD.TsPhiPOD, U, V, W, Nx*Ny, TSNBasis);
//        AdjustSingVals(W, TSNBasis);
```

```
        X.clear();
        for (t=0; t < Nt; t++)
           {
//           FormTsb(Tsb, t, Data);
//           Solve4coefs(U, V, W, Tsb, Nx*Ny, TSNBasis, X);
           for (k=0; k < TSNBasis; k++)
              X.push_back(SigmaTs.at(k) * MV7.at(t).at(k));
           POD.PODsParticleT.push_back(X);
           X.clear();
           } // end t
        }

     MPI_Barrier(MPI_COMM_WORLD);
     OutputPODsSplit(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
                  GPNBasis, GFNBasis, PPNBasis, TSNBasis,
                  Id, NIds);
     std::cout << "Output PODs Coefficients \n";

     MPI_Barrier(MPI_COMM_WORLD);

     OutputKey(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,
            Id, NIds);

     MPI_Barrier(MPI_COMM_WORLD);

     if (Id == (0 % NIds))
        std::cout << "Wrote key! \n";

     if (Id == 0)
        std::cout << "Have POD basis via SVD and "
                  << "output them to files. Done!\n";
     MPI_Barrier(MPI_COMM_WORLD);

//   std::ifstream ifilekey;
//   int ikc;
//   for (ikc = 0; ikc < NIds; ikc++)
//      if (ikc == Id)
//       {
       std::string ln = "./Output";
//       ifilekey.open("./Output/Key.txt");
        InputKey(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
              GPNBasis, GFNBasis, PPNBasis, TSNBasis,
              ln);
//       ifilekey.close();
```

```
//        }

    MPI_Barrier(MPI_COMM_WORLD);

    if (Id == (0 % NIds))
       std::cout << "Read key! \n";

    MPI_Barrier(MPI_COMM_WORLD);

    if (Id == (0 % NIds))
       {
       std::string ln = "./Output";
       InputGVBasis(GVNBasis,ln);
       InputPUBasis(PUNBasis,ln);
       InputPVBasis(PVNBasis,ln);
       InputGFBasis(GFNBasis,ln);
       InputGPBasis(GPNBasis,ln);
       InputPPBasis(PPNBasis,ln);
       InputPTBasis(TSNBasis,ln);
       InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
              GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
       }
    if (Id == (1 % NIds))
       {
       std::string ln = "./Output";
       InputGUBasis(GUNBasis,ln);
       InputPUBasis(PUNBasis,ln);
       InputPVBasis(PVNBasis,ln);
       InputGFBasis(GFNBasis,ln);
       InputGPBasis(GPNBasis,ln);
       InputPPBasis(PPNBasis,ln);
       InputPTBasis(TSNBasis,ln);
       InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
              GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
       }
    if (Id == (2 % NIds))
       {
       std::string ln = "./Output";
       InputGUBasis(GUNBasis,ln);
       InputGVBasis(GVNBasis,ln);
       InputPVBasis(PVNBasis,ln);
       InputGFBasis(GFNBasis,ln);
       InputGPBasis(GPNBasis,ln);
       InputPPBasis(PPNBasis,ln);
       InputPTBasis(TSNBasis,ln);
```

```
    InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
    }
  if (Id == (3 % NIds))
    {
    std::string ln = "./Output";
    InputGUBasis(GUNBasis,ln);
    InputGVBasis(GVNBasis,ln);
    InputPUBasis(PUNBasis,ln);
    InputGFBasis(GFNBasis,ln);
    InputGPBasis(GPNBasis,ln);
    InputPPBasis(PPNBasis,ln);
    InputPTBasis(TSNBasis,ln);
    InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
    }
  if (Id == (4 % NIds))
    {
    std::string ln = "./Output";
    InputGUBasis(GUNBasis,ln);
    InputGVBasis(GVNBasis,ln);
    InputPUBasis(PUNBasis,ln);
    InputPVBasis(PVNBasis,ln);
    InputGFBasis(GFNBasis,ln);
    InputPPBasis(PPNBasis,ln);
    InputPTBasis(TSNBasis,ln);
    InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
    }
  if (Id == (5 % NIds))
    {
    std::string ln = "./Output";
    InputGUBasis(GUNBasis,ln);
    InputGVBasis(GVNBasis,ln);
    InputPUBasis(PUNBasis,ln);
    InputPVBasis(PVNBasis,ln);
    InputGPBasis(GPNBasis,ln);
    InputPPBasis(PPNBasis,ln);
    InputPTBasis(TSNBasis,ln);
    InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
    }
  if (Id == (6 % NIds))
    {
    std::string ln = "./Output";
```

```
      InputGUBasis(GUNBasis,ln);
      InputGVBasis(GVNBasis,ln);
      InputPUBasis(PUNBasis,ln);
      InputPVBasis(PVNBasis,ln);
      InputGFBasis(GFNBasis,ln);
      InputGPBasis(GPNBasis,ln);
      InputPTBasis(TSNBasis,ln);
      InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
      }
   if (Id == (7 % NIds))
      {
      std::string ln = "./Output";
      InputGUBasis(GUNBasis,ln);
      InputGVBasis(GVNBasis,ln);
      InputPUBasis(PUNBasis,ln);
      InputPVBasis(PVNBasis,ln);
      InputGFBasis(GFNBasis,ln);
      InputGPBasis(GPNBasis,ln);
      InputPPBasis(PPNBasis,ln);
      InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
      }

   MPI_Barrier(MPI_COMM_WORLD);

   if (Id == 0)
      std::cout << "Shared Results!\n";

   FormReconVars(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,
            Id, NIds);
   std::cout << "Formed Reconstructed Solution \n";
//   MPI_Barrier(MPI_COMM_WORLD);
   ApproxErrorSplit(Data, Id, NIds);
   std::cout << "Approximated Error \n";

//   MPI_Barrier(MPI_COMM_WORLD);
   OutputSolution(Id, NIds);
   std::cout << "Output Solutions \n";

//   OutputErrorSplit(Id, NIds);

   MPI_Barrier(MPI_COMM_WORLD);
```

```
//    std::cout << "Output Error \n";


   return good;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputKey(int NGU, int NGV, int NPU, int NPV,
                    int NGP, int NGF, int NPP, int NPT,
                    int Id, int NIds)

   {

   std::ofstream ofilekey;
   std::ifstream ifilekey;

   int at, ax, ay, aug, avg, aus, avs, aeg, apg, aps;

   if (Id == (0 % NIds))
      {
      ofilekey.open("./Output/Key.txt");
      ofilekey << Nt << "\n";
      ofilekey << Nx << "\n";
      ofilekey << Ny << "\n";
      ofilekey << NGU << "\n";
      ofilekey.close();
      }

   MPI_Barrier(MPI_COMM_WORLD);
   if (Id == (1 % NIds))
      {
      ifilekey.open("./Output/Key.txt");
      ifilekey >> at >> ax >> ay >> aug;
      ifilekey.close();

      ofilekey.open("./Output/Key.txt");
      ofilekey << Nt << "\n";
      ofilekey << Nx << "\n";
      ofilekey << Ny << "\n";
      ofilekey << aug << "\n";
      ofilekey << NGV << "\n";
      ofilekey.close();
```

```
    }

MPI_Barrier(MPI_COMM_WORLD);
if (Id == (2 % NIds))
    {
    ifilekey.open("./Output/Key.txt");
    ifilekey >> at >> ax >> ay >> aug >> avg;
    ifilekey.close();

    ofilekey.open("./Output/Key.txt");
    ofilekey << Nt << "\n";
    ofilekey << Nx << "\n";
    ofilekey << Ny << "\n";
    ofilekey << aug << "\n";
    ofilekey << avg << "\n";
    ofilekey << NPU << "\n";
    ofilekey.close();
    }

MPI_Barrier(MPI_COMM_WORLD);
if (Id == (3 % NIds))
    {
    ifilekey.open("./Output/Key.txt");
    ifilekey >> at >> ax >> ay >> aug >> avg >> aus;
    ifilekey.close();

    ofilekey.open("./Output/Key.txt");
    ofilekey << Nt << "\n";
    ofilekey << Nx << "\n";
    ofilekey << Ny << "\n";
    ofilekey << aug << "\n";
    ofilekey << avg << "\n";
    ofilekey << aus << "\n";
    ofilekey << NPV << "\n";
    ofilekey.close();
    }

MPI_Barrier(MPI_COMM_WORLD);
if (Id == (4 % NIds))
    {
    ifilekey.open("./Output/Key.txt");
    ifilekey >> at >> ax >> ay >> aug >> avg >> aus >> avs;
    ifilekey.close();

    ofilekey.open("./Output/Key.txt");
```

```
    ofilekey << Nt << "\n";
    ofilekey << Nx << "\n";
    ofilekey << Ny << "\n";
    ofilekey << aug << "\n";
    ofilekey << avg << "\n";
    ofilekey << aus << "\n";
    ofilekey << avs << "\n";
    ofilekey << NGP << "\n";
    ofilekey.close();
    }

MPI_Barrier(MPI_COMM_WORLD);
if (Id == (5 % NIds))
    {
    ifilekey.open("./Output/Key.txt");
    ifilekey >> at >> ax >> ay >> aug >> avg >> aus >> avs >> apg;
    ifilekey.close();

    ofilekey.open("./Output/Key.txt");
    ofilekey << Nt << "\n";
    ofilekey << Nx << "\n";
    ofilekey << Ny << "\n";
    ofilekey << aug << "\n";
    ofilekey << avg << "\n";
    ofilekey << aus << "\n";
    ofilekey << avs << "\n";
    ofilekey << apg << "\n";
    ofilekey << NGF << "\n";
    ofilekey.close();
    }

MPI_Barrier(MPI_COMM_WORLD);
if (Id == (6 % NIds))
    {
    ifilekey.open("./Output/Key.txt");
    ifilekey >> at >> ax >> ay >> aug >> avg >> aus >> avs
            >> apg >> aeg;
    ifilekey.close();

    ofilekey.open("./Output/Key.txt");
    ofilekey << Nt << "\n";
    ofilekey << Nx << "\n";
    ofilekey << Ny << "\n";
    ofilekey << aug << "\n";
    ofilekey << avg << "\n";
```

```
    ofilekey << aus << "\n";
    ofilekey << avs << "\n";
    ofilekey << apg << "\n";
    ofilekey << aeg << "\n";
    ofilekey << NPP << "\n";
    ofilekey.close();
    }

  MPI_Barrier(MPI_COMM_WORLD);
  if (Id == (7 % NIds))
    {
    ifilekey.open("./Output/Key.txt");
    ifilekey >> at >> ax >> ay >> aug >> avg >> aus >> avs
          >> apg >> aeg >> aps;
    ifilekey.close();

    ofilekey.open("./Output/Key.txt");
    ofilekey << Nt << "\n";
    ofilekey << Nx << "\n";
    ofilekey << Ny << "\n";
    ofilekey << aug << "\n";
    ofilekey << avg << "\n";
    ofilekey << aus << "\n";
    ofilekey << avs << "\n";
    ofilekey << apg << "\n";
    ofilekey << aeg << "\n";
    ofilekey << aps << "\n";
    ofilekey << NPT << "\n";
    ofilekey.close();
    }

  MPI_Barrier(MPI_COMM_WORLD);

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::InputKey(int & NGU, int & NGV, int & NPU, int & NPV,
                         int & NGP, int & NGF, int & NPP, int & NPT,
                         std::string & ln)

  {

  int Ntz, Nxz, Nyz;
```

```
    int a0, a1, a2, a3, a4, a5, a6, a7;

    std::string c = "/Key.txt";

    std::cout << "In InputKey \n";

    std::cout << ln << "\n is the directory in IK.  \n";

    std::cout << ln+c << "\n is the file in IK.  \n";

    std::ifstream ifile;
//    ifile.open(
//    "/home/sbeck/DOE/Programs/MfixPOD2/V1/Testing/Run01/Output/Key.txt" );
    ifile.open( (ln+c).c_str() );
//    ifile.open( "../Run01/Output/Key.txt");

    ifile >> Ntz >> Nxz >> Nyz;
    ifile >> a0 >> a1 >> a2 >> a3;
    ifile >> a4 >> a5 >> a6 >> a7;
    ifile.close();

    NGU = a0;
    NGV = a1;
    NPU = a2;
    NPV = a3;
    NGP = a4;
    NGF = a5;
    NPP = a6;
    NPT = a7;

    std::cout << NGU << "\n" << NGU << "\n" << NPU << "\n"
            << NPV << "\n" << NGP << "\n" << NGF << "\n"
            << NPP << "\n" << NPT << "\n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputMFIX(SimVars & SV)

    {

    std::string fegu0 = "./Output/MFIXGasVelocityX0.txt";
    std::string fegv0 = "./Output/MFIXGasVelocityY0.txt";
```

```
std::string fesu0 = "./Output/MFIXParticleVelocityX0.txt";
std::string fesv0 = "./Output/MFIXParticleVelocityY0.txt";
std::string fegf0 = "./Output/MFIXGasFF0.txt";
std::string fegp0 = "./Output/MFIXGasPressure0.txt";
std::string fesp0 = "./Output/MFIXParticlePressure0.txt";
std::string fest0 = "./Output/MFIXParticleTemperature0.txt";

std::ofstream ofilegu0 (fegu0.c_str());
std::ofstream ofilegv0 (fegv0.c_str());
std::ofstream ofilesu0 (fesu0.c_str());
std::ofstream ofilesv0 (fesv0.c_str());

std::ofstream ofilegf0 (fegf0.c_str());
std::ofstream ofilegp0 (fegp0.c_str());
std::ofstream ofilesp0 (fesp0.c_str());
std::ofstream ofilest0 (fest0.c_str());

std::string fegu1 = "./Output/MFIXGasVelocityX1.txt";
std::string fegv1 = "./Output/MFIXGasVelocityY1.txt";
std::string fesu1 = "./Output/MFIXParticleVelocityX1.txt";
std::string fesv1 = "./Output/MFIXParticleVelocityY1.txt";
std::string fegf1 = "./Output/MFIXGasFF1.txt";
std::string fegp1 = "./Output/MFIXGasPressure1.txt";
std::string fesp1 = "./Output/MFIXParticlePressure1.txt";
std::string fest1 = "./Output/MFIXParticleTemperature1.txt";

std::ofstream ofilegu1 (fegu1.c_str());
std::ofstream ofilegv1 (fegv1.c_str());
std::ofstream ofilesu1 (fesu1.c_str());
std::ofstream ofilesv1 (fesv1.c_str());

std::ofstream ofilegf1 (fegf1.c_str());
std::ofstream ofilegp1 (fegp1.c_str());
std::ofstream ofilesp1 (fesp1.c_str());
std::ofstream ofilest1 (fest1.c_str());

std::string fegu2 = "./Output/MFIXGasVelocityX2.txt";
std::string fegv2 = "./Output/MFIXGasVelocityY2.txt";
std::string fesu2 = "./Output/MFIXParticleVelocityX2.txt";
std::string fesv2 = "./Output/MFIXParticleVelocityY2.txt";
std::string fegf2 = "./Output/MFIXGasFF2.txt";
std::string fegp2 = "./Output/MFIXGasPressure2.txt";
std::string fesp2 = "./Output/MFIXParticlePressure2.txt";
std::string fest2 = "./Output/MFIXParticleTemperature2.txt";
```

```
std::ofstream ofilegu2 (fegu2.c_str());
std::ofstream ofilegv2 (fegv2.c_str());
std::ofstream ofilesu2 (fesu2.c_str());
std::ofstream ofilesv2 (fesv2.c_str());

std::ofstream ofilegf2 (fegf2.c_str());
std::ofstream ofilegp2 (fegp2.c_str());
std::ofstream ofilesp2 (fesp2.c_str());
std::ofstream ofilest2 (fest2.c_str());

int y;
int x;

std::cout << "Output MFIX data.  \n";
for (y=(Ny-1); y >=0; y--)
   {
   for (x=(Nx-1); x > 0; x--)
      {
      ofilegu0 << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << SV.Gas.GetU(0,x,y) << " ";
      ofilegv0 << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << SV.Gas.GetV(0,x,y) << " ";
      ofilesu0 << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << SV.Particle.GetU(0,x,y) << " ";
      ofilesv0 << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << SV.Particle.GetV(0,x,y) << " ";

      ofilegf0 << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << SV.Particle.GetEP(0,x,y) << " ";
      ofilegp0 << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << SV.Gas.GetP(0,x,y) << " ";
      ofilesp0 << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << SV.Particle.GetP(0,x,y) << " ";
      ofilest0 << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << SV.Particle.GetTheta(0,x,y) << " ";

      ofilegu1 << std::setiosflags(std::ios::fixed)
```

```
        << std::setprecision(15)
        << SV.Gas.GetU(500,x,y) << " ";
ofilegv1 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetV(500,x,y) << " ";
ofilesu1 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetU(500,x,y) << " ";
ofilesv1 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetV(500,x,y) << " ";

ofilegf1 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetEP(500,x,y) << " ";
ofilegp1 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetP(500,x,y) << " ";
ofilesp1 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetP(500,x,y) << " ";
ofilest1 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetTheta(500,x,y) << " ";

ofilegu2 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetU(1000,x,y) << " ";
ofilegv2 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetV(1000,x,y) << " ";
ofilesu2 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetU(1000,x,y) << " ";
ofilesv2 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetV(1000,x,y) << " ";

ofilegf2 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetEP(1000,x,y) << " ";
ofilegp2 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetP(1000,x,y) << " ";
ofilesp2 << std::setiosflags(std::ios::fixed)
```

```
            << std::setprecision(15)
            << SV.Particle.GetP(1000,x,y) << " ";
    ofilest2 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Particle.GetTheta(1000,x,y) << " ";

    } // end for x
ofilegu0 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetU(0,0,y) << "\n";
ofilegv0 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetV(0,0,y) << "\n";
ofilesu0 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetU(0,0,y) << "\n";
ofilesv0 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetV(0,0,y) << "\n";
ofilegf0 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetEP(0,0,y) << "\n";
ofilegp0 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetP(0,0,y) << "\n";
ofilesp0 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetP(0,0,y) << "\n";
ofilest0 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetTheta(0,0,y) << "\n";

ofilegu1 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetU(500,0,y) << "\n";
ofilegv1 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Gas.GetV(500,0,y) << "\n";
ofilesu1 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetU(500,0,y) << "\n";
ofilesv1 << std::setiosflags(std::ios::fixed)
        << std::setprecision(15)
        << SV.Particle.GetV(500,0,y) << "\n";
ofilegf1 << std::setiosflags(std::ios::fixed)
```

```
            << std::setprecision(15)
            << SV.Gas.GetEP(500,0,y) << "\n";
    ofilegp1 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Gas.GetP(500,0,y) << "\n";
    ofilesp1 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Particle.GetP(500,0,y) << "\n";
    ofilest1 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Particle.GetTheta(500,0,y) << "\n";

    ofilegu2 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Gas.GetU(1000,0,y) << "\n";
    ofilegv2 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Gas.GetV(1000,0,y) << "\n";
    ofilesu2 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Particle.GetU(1000,0,y) << "\n";
    ofilesv2 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Particle.GetV(1000,0,y) << "\n";
    ofilegf2 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Gas.GetEP(1000,0,y) << "\n";
    ofilegp2 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Gas.GetP(1000,0,y) << "\n";
    ofilesp2 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Particle.GetP(1000,0,y) << "\n";
    ofilest2 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << SV.Particle.GetTheta(1000,0,y) << "\n";
    } // end for y

ofilegu0.close();
ofilegv0.close();
ofilesu0.close();
ofilesv0.close();

ofilegf0.close();
ofilegp0.close();
```

```
    ofilesp0.close();
    ofilest0.close();

    ofilegu1.close();
    ofilegv1.close();
    ofilesu1.close();
    ofilesv1.close();

    ofilegf1.close();
    ofilegp1.close();
    ofilesp1.close();
    ofilest1.close();

    ofilegu2.close();
    ofilegv2.close();
    ofilesu2.close();
    ofilesv2.close();

    ofilegf2.close();
    ofilegp2.close();
    ofilesp2.close();
    ofilest2.close();

    std::cout << "Stored MFIX data for 0, 500, 1000. \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::OutputSolution(int Id, int NIds)

    {

    int x;
    int y;
    int t;

    if (Id == (0 % NIds))
        {
        // Create a text file - one per timestep
        //writing file
        for (t=0; t < Nt; t++)
            {
            std::string feg =
```

```
        "./Output/GasVelocityX" + intToString(t) + ".txt";
        std::ofstream ofileg (feg.c_str());
        for (y=Ny-1; y >=0; y--)
           {
           for (x=Nx-1; x > 0; x--)
              {
              ofileg << std::setiosflags(std::ios::fixed)
                     << std::setprecision(15)
                     << UgR.at(t).at(y).at(x) << " ";
              }
           ofileg << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << UgR.at(t).at(y).at(0) << "\n";
           }
        ofileg.close();
        } // end for t
     } // end 0
  if (Id == (1 % NIds))
     {
     for (t=0; t < Nt; t++)
        {
        std::string feg =
        "./Output/GasVelocityY" + intToString(t) + ".txt";
        std::ofstream ofileg (feg.c_str());
        for (y=Ny-1; y >=0; y--)
           {
           for (x=Nx-1; x > 0; x--)
              {
              ofileg << std::setiosflags(std::ios::fixed)
                     << std::setprecision(15)
                     << VgR.at(t).at(y).at(x) << " ";
              }
           ofileg << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << VgR.at(t).at(y).at(0) << "\n";
           }
        ofileg.close();
        } // end for t
     } // end 1
  if (Id == (2 % NIds))
     {
     for (t=0; t < Nt; t++)
        {
        std::string fes =
        "./Output/ParticleVelocityX" + intToString(t) + ".txt";
```

```
        std::ofstream ofiles (fes.c_str());
        for (y=Ny-1; y >=0; y--)
           {
           for (x=Nx-1; x > 0; x--)
              {
              ofiles << std::setiosflags(std::ios::fixed)
                     << std::setprecision(15)
                     << UsR.at(t).at(y).at(x) << " ";
              }
           ofiles << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << UsR.at(t).at(y).at(0) << "\n";
           }
        ofiles.close();
        } // end for t
     } // end for 2
  if (Id == (3 % NIds))
     {
     for (t=0; t < Nt; t++)
        {
        std::string fes =
        "./Output/ParticleVelocityY" + intToString(t) + ".txt";
        std::ofstream ofiles (fes.c_str());
        for (y=Ny-1; y >=0; y--)
           {
           for (x=Nx-1; x > 0; x--)
              {
              ofiles << std::setiosflags(std::ios::fixed)
                     << std::setprecision(15)
                     << VsR.at(t).at(y).at(x) << " ";
              }
           ofiles << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << VsR.at(t).at(y).at(0) << "\n";
           }
        ofiles.close();
        } // end for t
     } // end for 3
  if (Id == (4 % NIds))
     {
     for (t=0; t < Nt; t++)
        {
        std::string fegp =
        "./Output/GasPressure" + intToString(t) + ".txt";
        std::ofstream ofilegp (fegp.c_str());
```

```
        for (y=Ny-1; y >=0; y--)
            {
            for (x=Nx-1; x > 0; x--)
                {
                ofilegp << std::setiosflags(std::ios::fixed)
                        << std::setprecision(15)
                        << PgR.at(t).at(y).at(x) << " ";
                }
            ofilegp << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << PgR.at(t).at(y).at(0) << "\n";
            }
        ofilegp.close();
        } // end for t
    } // end 4
if (Id == (5 % NIds))
    {
    for (t=0; t < Nt; t++)
        {
        std::string fegf =
        "./Output/GasFF" + intToString(t) + ".txt";
        std::ofstream ofilegf (fegf.c_str());
        for (y=Ny-1; y >=0; y--)
            {
            for (x=Nx-1; x > 0; x--)
                {
                ofilegf << std::setiosflags(std::ios::fixed)
                        << std::setprecision(15)
                        << FgR.at(t).at(y).at(x) << " ";
                }
            ofilegf << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << FgR.at(t).at(y).at(0) << "\n";
            }
        ofilegf.close();
        } // end for t
    } // end 5
if (Id == (6 % NIds))
    {
    for (t=0; t < Nt; t++)
        {
        std::string fesp =
        "./Output/ParticlePressure" + intToString(t) + ".txt";
        std::ofstream ofilesp (fesp.c_str());
        for (y=Ny-1; y >=0; y--)
```

```
            {
            for (x=Nx-1; x > 0; x--)
                {
                ofilesp << std::setiosflags(std::ios::fixed)
                        << std::setprecision(15)
                        << PsR.at(t).at(y).at(x) << " ";
                }
            ofilesp << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << PsR.at(t).at(y).at(0) << "\n";
            }
        ofilesp.close();
        } // end for t
    } // end 6
  if (Id == (7 % NIds))
    {
    for (t=0; t < Nt; t++)
        {
        std::string fest =
        "./Output/ParticleTemperature" + intToString(t) + ".txt";
        std::ofstream ofilest (fest.c_str());
        for (y=Ny-1; y >=0; y--)
            {
            for (x=Nx-1; x > 0; x--)
                {
                ofilest << std::setiosflags(std::ios::fixed)
                        << std::setprecision(15)
                        << TsR.at(t).at(y).at(x) << " ";
                }
            ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << TsR.at(t).at(y).at(0) << "\n";
            }
        ofilest.close();
        } // end for t
    } // end 7

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::GetPODData(PODData & P)

  {
```

```
    P = POD;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::GetSol(int & all, std::string & ln, bool load,
                    std::vector < std::vector< double > > & Ug,
                    std::vector < std::vector< double > > & Vg,
                    std::vector < std::vector< double > > & Us,
                    std::vector < std::vector< double > > & Vs,
                    std::vector < std::vector< double > > & Eg,
                    std::vector < std::vector< double > > & Pg,
                    std::vector < std::vector< double > > & Ps,
                    std::vector < std::vector< double > > & Ts)
    {

    int y;
    int x;
    int t;
    int maxt;

   if (load)
      {
      std::vector< double > tUg (GetNx() * GetNy(), 0.0);
      std::vector< double > tVg (GetNx() * GetNy(), 0.0);
      std::vector< double > tUs (GetNx() * GetNy(), 0.0);
      std::vector< double > tVs (GetNx() * GetNy(), 0.0);
      std::vector< double > tEg (GetNx() * GetNy(), 0.0);
      std::vector< double > tPg (GetNx() * GetNy(), 0.0);
      std::vector< double > tPs (GetNx() * GetNy(), 0.0);
      std::vector< double > tTs (GetNx() * GetNy(), 0.0);
      if (all == 1)
         maxt = GetNt();
      else
         maxt = 1;
      for (t=0; t < maxt; t++)
         {
         std::string UgS =
          "./Output/GasVelocityX" + intToString(t)
                                + ".txt";
         std::string VgS =
          "./Output/GasVelocityY" + intToString(t)
```

```
                        + ".txt";
std::string UsS =
 "./Output/ParticleVelocityX" + intToString(t)
                        + ".txt";
std::string VsS =
 "./Output/ParticleVelocityY" + intToString(t)
                        + ".txt";
std::string EgS =
 "./Output/GasFF" + intToString(t)
                        + ".txt";
std::string PgS =
 "./Output/GasPressure" + intToString(t)
                        + ".txt";
std::string PsS =
 "./Output/ParticlePressure" + intToString(t)
                        + ".txt";
std::string TsS =
 "./Output/ParticleTemperature" + intToString(t)
                        + ".txt";
std::ifstream ifileUg (UgS.c_str());
std::ifstream ifileVg (VgS.c_str());
std::ifstream ifileUs (UsS.c_str());
std::ifstream ifileVs (VsS.c_str());
std::ifstream ifileEg (EgS.c_str());
std::ifstream ifilePg (PgS.c_str());
std::ifstream ifilePs (PsS.c_str());
std::ifstream ifileTs (TsS.c_str());
for (y=GetNy()-1; y >=0; y--)
   for (x=GetNx()-1; x >= 0; x--)
      {
      ifileTs >> tTs.at(x+(y * GetNx()));
      ifilePs >> tPs.at(x+(y * GetNx()));
      ifilePg >> tPg.at(x+(y * GetNx()));
      ifileEg >> tEg.at(x+(y * GetNx()));
      ifileVg >> tVg.at(x+(y * GetNx()));
      ifileUg >> tUg.at(x+(y * GetNx()));
      ifileUs >> tUs.at(x+(y * GetNx()));
      ifileVs >> tVs.at(x+(y * GetNx()));
      }
ifileUg.close();
ifileVg.close();
ifileUs.close();
ifileVs.close();
ifileEg.close();
ifilePg.close();
```

```
        ifilePs.close();
        ifileTs.close();

        Ug.push_back( tUg );
        Vg.push_back( tVg );
        Us.push_back( tUs );
        Vs.push_back( tVs );
        Eg.push_back( tEg );
        Pg.push_back( tPg );
        Ps.push_back( tPs );
        Ts.push_back( tTs );
        }// end for maxt

    } // if load

else
 {
 std::vector< double > tUg;
 std::vector< double > tVg;
 std::vector< double > tUs;
 std::vector< double > tVs;
 std::vector< double > tEg;
 std::vector< double > tPg;
 std::vector< double > tPs;
 std::vector< double > tTs;

 tUg.clear();
 tVg.clear();
 tUs.clear();
 tVs.clear();
 tEg.clear();
 tPg.clear();
 tPs.clear();
 tTs.clear();

 Ug.clear();
 Vg.clear();
 Us.clear();
 Vs.clear();
 Eg.clear();
 Pg.clear();
 Ps.clear();
 Ts.clear();

 for (y = 0; y < GetNy(); y++)
```

```
    for (x = 0; x < GetNx(); x++)
        {
        tUg.push_back( UgR.at(0).at(y).at(x) );
        tVg.push_back( VgR.at(0).at(y).at(x) );
        tUs.push_back( UsR.at(0).at(y).at(x) );
        tVs.push_back( VsR.at(0).at(y).at(x) );
        tEg.push_back( FgR.at(0).at(y).at(x) );
        tPg.push_back( PgR.at(0).at(y).at(x) );
        tPs.push_back( PsR.at(0).at(y).at(x) );
        tTs.push_back( TsR.at(0).at(y).at(x) );
        }

Ug.push_back( tUg );
Vg.push_back( tVg );
Us.push_back( tUs );
Vs.push_back( tVs );
Eg.push_back( tEg );
Pg.push_back( tPg );
Ps.push_back( tPs );
Ts.push_back( tTs );

if (all == 1)
   {
   for (t = 1; t < GetNt(); t++)
      {
      tUg.clear();
      tVg.clear();
      tUs.clear();
      tVs.clear();
      tEg.clear();
      tPg.clear();
      tPs.clear();
      tTs.clear();

      for (y = 0; y < GetNy(); y++)
         for (x = 0; x < GetNx(); x++)
            {
            tUg.push_back( UgR.at(t).at(y).at(x) );
            tVg.push_back( VgR.at(t).at(y).at(x) );
            tUs.push_back( UsR.at(t).at(y).at(x) );
            tVs.push_back( VsR.at(t).at(y).at(x) );
            tEg.push_back( FgR.at(t).at(y).at(x) );
            tPg.push_back( PgR.at(t).at(y).at(x) );
            tPs.push_back( PsR.at(t).at(y).at(x) );
            tTs.push_back( TsR.at(t).at(y).at(x) );
```

```
            }
        Ug.push_back( tUg );
        Vg.push_back( tVg );
        Us.push_back( tUs );
        Vs.push_back( tVs );
        Eg.push_back( tEg );
        Pg.push_back( tPg );
        Ps.push_back( tPs );
        Ts.push_back( tTs );
        } // end for t
      }  // end for all == 1

    }// end else (!load)

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::XChangeBasisMPI(int & NGU, int & NGV, int & NPU,
                      int & NPV, int & NGP, int & NGF,
                      int & NPP, int & NPT, int Id, int NIds)

  {

  MPI_Barrier(MPI_COMM_WORLD);
  MPI_Status status;

  if (Id == (0 % NIds))
     {
     MPI_Send( &NGU, 1, MPI_INT, 1, 0, MPI_COMM_WORLD );
     MPI_Send( &NGU, 1, MPI_INT, 2, 0, MPI_COMM_WORLD );
     MPI_Send( &NGU, 1, MPI_INT, 3, 0, MPI_COMM_WORLD );
     MPI_Send( &NGU, 1, MPI_INT, 4, 0, MPI_COMM_WORLD );
     MPI_Send( &NGU, 1, MPI_INT, 5, 0, MPI_COMM_WORLD );
     MPI_Send( &NGU, 1, MPI_INT, 6, 0, MPI_COMM_WORLD );
     MPI_Send( &NGU, 1, MPI_INT, 7, 0, MPI_COMM_WORLD );

     MPI_Recv( &NGV, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &status );
     MPI_Recv( &NPU, 1, MPI_INT, 2, 2, MPI_COMM_WORLD, &status );
     MPI_Recv( &NPV, 1, MPI_INT, 3, 3, MPI_COMM_WORLD, &status );
     MPI_Recv( &NGP, 1, MPI_INT, 4, 4, MPI_COMM_WORLD, &status );
     MPI_Recv( &NGF, 1, MPI_INT, 5, 5, MPI_COMM_WORLD, &status );
     MPI_Recv( &NPP, 1, MPI_INT, 6, 6, MPI_COMM_WORLD, &status );
     MPI_Recv( &NPT, 1, MPI_INT, 7, 7, MPI_COMM_WORLD, &status );
```

```
    }

if (Id == (1 % NIds))
    {
    MPI_Send( &NGV, 1, MPI_INT, 0, 1, MPI_COMM_WORLD );
    MPI_Send( &NGV, 1, MPI_INT, 2, 1, MPI_COMM_WORLD );
    MPI_Send( &NGV, 1, MPI_INT, 3, 1, MPI_COMM_WORLD );
    MPI_Send( &NGV, 1, MPI_INT, 4, 1, MPI_COMM_WORLD );
    MPI_Send( &NGV, 1, MPI_INT, 5, 1, MPI_COMM_WORLD );
    MPI_Send( &NGV, 1, MPI_INT, 6, 1, MPI_COMM_WORLD );
    MPI_Send( &NGV, 1, MPI_INT, 7, 1, MPI_COMM_WORLD );

    MPI_Recv( &NGU, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPU, 1, MPI_INT, 2, 2, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPV, 1, MPI_INT, 3, 3, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGP, 1, MPI_INT, 4, 4, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGF, 1, MPI_INT, 5, 5, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPP, 1, MPI_INT, 6, 6, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPT, 1, MPI_INT, 7, 7, MPI_COMM_WORLD, &status );
    }

if (Id == (2 % NIds))
    {
    MPI_Send( &NPU, 1, MPI_INT, 0, 2, MPI_COMM_WORLD );
    MPI_Send( &NPU, 1, MPI_INT, 1, 2, MPI_COMM_WORLD );
    MPI_Send( &NPU, 1, MPI_INT, 3, 2, MPI_COMM_WORLD );
    MPI_Send( &NPU, 1, MPI_INT, 4, 2, MPI_COMM_WORLD );
    MPI_Send( &NPU, 1, MPI_INT, 5, 2, MPI_COMM_WORLD );
    MPI_Send( &NPU, 1, MPI_INT, 6, 2, MPI_COMM_WORLD );
    MPI_Send( &NPU, 1, MPI_INT, 7, 2, MPI_COMM_WORLD );

    MPI_Recv( &NGU, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGV, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPV, 1, MPI_INT, 3, 3, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGP, 1, MPI_INT, 4, 4, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGF, 1, MPI_INT, 5, 5, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPP, 1, MPI_INT, 6, 6, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPT, 1, MPI_INT, 7, 7, MPI_COMM_WORLD, &status );
    }

if (Id == (3 % NIds))
    {
    MPI_Send( &NPV, 1, MPI_INT, 0, 3, MPI_COMM_WORLD );
    MPI_Send( &NPV, 1, MPI_INT, 1, 3, MPI_COMM_WORLD );
    MPI_Send( &NPV, 1, MPI_INT, 2, 3, MPI_COMM_WORLD );
```

```
    MPI_Send( &NPV, 1, MPI_INT, 4, 3, MPI_COMM_WORLD );
    MPI_Send( &NPV, 1, MPI_INT, 5, 3, MPI_COMM_WORLD );
    MPI_Send( &NPV, 1, MPI_INT, 6, 3, MPI_COMM_WORLD );
    MPI_Send( &NPV, 1, MPI_INT, 7, 3, MPI_COMM_WORLD );

    MPI_Recv( &NGU, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGV, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPU, 1, MPI_INT, 2, 2, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGP, 1, MPI_INT, 4, 4, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGF, 1, MPI_INT, 5, 5, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPP, 1, MPI_INT, 6, 6, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPT, 1, MPI_INT, 7, 7, MPI_COMM_WORLD, &status );
    }

 if (Id == (4 % NIds))
    {
    MPI_Send( &NGP, 1, MPI_INT, 0, 4, MPI_COMM_WORLD );
    MPI_Send( &NGP, 1, MPI_INT, 1, 4, MPI_COMM_WORLD );
    MPI_Send( &NGP, 1, MPI_INT, 2, 4, MPI_COMM_WORLD );
    MPI_Send( &NGP, 1, MPI_INT, 3, 4, MPI_COMM_WORLD );
    MPI_Send( &NGP, 1, MPI_INT, 5, 4, MPI_COMM_WORLD );
    MPI_Send( &NGP, 1, MPI_INT, 6, 4, MPI_COMM_WORLD );
    MPI_Send( &NGP, 1, MPI_INT, 7, 4, MPI_COMM_WORLD );

    MPI_Recv( &NGU, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGV, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPU, 1, MPI_INT, 2, 2, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPV, 1, MPI_INT, 3, 3, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGF, 1, MPI_INT, 5, 5, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPP, 1, MPI_INT, 6, 6, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPT, 1, MPI_INT, 7, 7, MPI_COMM_WORLD, &status );
    }

 if (Id == (5 % NIds))
    {
    MPI_Send( &NGF, 1, MPI_INT, 0, 5, MPI_COMM_WORLD );
    MPI_Send( &NGF, 1, MPI_INT, 1, 5, MPI_COMM_WORLD );
    MPI_Send( &NGF, 1, MPI_INT, 2, 5, MPI_COMM_WORLD );
    MPI_Send( &NGF, 1, MPI_INT, 3, 5, MPI_COMM_WORLD );
    MPI_Send( &NGF, 1, MPI_INT, 4, 5, MPI_COMM_WORLD );
    MPI_Send( &NGF, 1, MPI_INT, 6, 5, MPI_COMM_WORLD );
    MPI_Send( &NGF, 1, MPI_INT, 7, 5, MPI_COMM_WORLD );

    MPI_Recv( &NGU, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGV, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &status );
```

```
    MPI_Recv( &NPU, 1, MPI_INT, 2, 2, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPV, 1, MPI_INT, 3, 3, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGP, 1, MPI_INT, 4, 4, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPP, 1, MPI_INT, 6, 6, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPT, 1, MPI_INT, 7, 7, MPI_COMM_WORLD, &status );
    }

if (Id == (6 % NIds))
    {
    MPI_Send( &NPP, 1, MPI_INT, 0, 6, MPI_COMM_WORLD );
    MPI_Send( &NPP, 1, MPI_INT, 1, 6, MPI_COMM_WORLD );
    MPI_Send( &NPP, 1, MPI_INT, 2, 6, MPI_COMM_WORLD );
    MPI_Send( &NPP, 1, MPI_INT, 3, 6, MPI_COMM_WORLD );
    MPI_Send( &NPP, 1, MPI_INT, 4, 6, MPI_COMM_WORLD );
    MPI_Send( &NPP, 1, MPI_INT, 5, 6, MPI_COMM_WORLD );
    MPI_Send( &NPP, 1, MPI_INT, 7, 6, MPI_COMM_WORLD );

    MPI_Recv( &NGU, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGV, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPU, 1, MPI_INT, 2, 2, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPV, 1, MPI_INT, 3, 3, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGP, 1, MPI_INT, 4, 4, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGF, 1, MPI_INT, 5, 5, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPT, 1, MPI_INT, 7, 7, MPI_COMM_WORLD, &status );
    }

if (Id == (7 % NIds))
    {
    MPI_Send( &NPT, 1, MPI_INT, 0, 7, MPI_COMM_WORLD );
    MPI_Send( &NPT, 1, MPI_INT, 1, 7, MPI_COMM_WORLD );
    MPI_Send( &NPT, 1, MPI_INT, 2, 7, MPI_COMM_WORLD );
    MPI_Send( &NPT, 1, MPI_INT, 3, 7, MPI_COMM_WORLD );
    MPI_Send( &NPT, 1, MPI_INT, 4, 7, MPI_COMM_WORLD );
    MPI_Send( &NPT, 1, MPI_INT, 5, 7, MPI_COMM_WORLD );
    MPI_Send( &NPT, 1, MPI_INT, 6, 7, MPI_COMM_WORLD );

    MPI_Recv( &NGU, 1, MPI_INT, 0, 0, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGV, 1, MPI_INT, 1, 1, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPU, 1, MPI_INT, 2, 2, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPV, 1, MPI_INT, 3, 3, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGP, 1, MPI_INT, 4, 4, MPI_COMM_WORLD, &status );
    MPI_Recv( &NGF, 1, MPI_INT, 5, 5, MPI_COMM_WORLD, &status );
    MPI_Recv( &NPP, 1, MPI_INT, 6, 6, MPI_COMM_WORLD, &status );
    }
```

```
    MPI_Barrier(MPI_COMM_WORLD);

    std::cout << "Finished exchanging basis... \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

bool PODSolver::InterpPOD(std::vector< std::vector< double > > & Ug,
                          std::vector< std::vector< double > > & Vg,
                          std::vector< std::vector< double > > & Us,
                          std::vector< std::vector< double > > & Vs,
                          std::vector< std::vector< double > > & Eg,
                          std::vector< std::vector< double > > & Pg,
                          std::vector< std::vector< double > > & Ps,
                          std::vector< std::vector< double > > & Ts,
                          int Nx, int Ny, int Nt, int Id, int NIds,
                          std::vector< double > & e, SimVars & Data)

    {

    bool good = true;

    std::vector< double > Dg;
    std::vector< double > ODg;
    std::vector< double > Ds;
    std::vector< double > ODs;

    std::vector< double > DFg;
    std::vector< double > ODFg;
    std::vector< double > DPg;
    std::vector< double > ODPg;
    std::vector< double > DPs;
    std::vector< double > ODPs;
    std::vector< double > DTs;
    std::vector< double > ODTs;

    std::vector< int > GRank;
    std::vector< int > PRank;

    std::vector< int > TsRank;
    std::vector< int > PsRank;
    std::vector< int > PgRank;
    std::vector< int > FgRank;
```

```
std::vector< int > GPODevs;
std::vector< int > PPODevs;

std::vector< int > PSODevs;
std::vector< int > PGODevs;
std::vector< int > FGODevs;
std::vector< int > TSODevs;

energy = e;

int GUNBasis;
int GVNBasis;
int PUNBasis;
int PVNBasis;
int GFNBasis;
int GPNBasis;
int PPNBasis;
int TSNBasis;

int i;
int j;
int t;

double Gceu;
double Gcev;
double Pceu;
double Pcev;
double Fgce;
double Pgce;
double Psce;
double Tsce;

std::vector< std::vector< double > > MU;
std::vector< std::vector< double > > MV;

std::vector< double > SigmaUg;
std::vector< double > SigmaVg;
std::vector< double > SigmaUs;
std::vector< double > SigmaVs;
std::vector< double > SigmaPg;
std::vector< double > SigmaPs;
std::vector< double > SigmaFg;
std::vector< double > SigmaTs;
std::vector< double > SigmaG;
```

```
std::vector< double > SigmaP;

std::vector<double> tmp;

std::vector< std::vector< double > > GAu;
std::vector< std::vector< double > > GAv;
std::vector< std::vector< double > > PAu;
std::vector< std::vector< double > > PAv;

std::vector< std::vector< double > > GA;
std::vector< std::vector< double > > PA;

std::vector< std::vector< double > > GAP;
std::vector< std::vector< double > > GAF;
std::vector< std::vector< double > > PAP;
std::vector< std::vector< double > > PAT;

std::vector< double > tGAu;
std::vector< double > tGAv;
std::vector< double > tPAu;
std::vector< double > tPAv;

std::vector< double > tGA;
std::vector< double > tPA;

std::vector< double > tGAP;
std::vector< double > tGAF;
std::vector< double > tPAP;
std::vector< double > tPAT;

std::vector<double> Gub;
std::vector<double> Gvb;
std::vector<double> Pub;
std::vector<double> Pvb;

std::vector<double> Gpb;
std::vector<double> Gfb;
std::vector<double> Psb;
std::vector<double> Tsb;

std::vector< std::vector< double > > U;
std::vector< std::vector< double > > V;
std::vector< double > W;
std::vector< double > X;
```

```
//    int NumCVars = 6;
    int k;

    POD.InitPODData(Nx, Ny, Nt, 0, 0, 0);

    // Calculates mean cell velocities for Gas and Solids
    // As well as Temperature and Pressure
    GetMeanPODs(Ug, Vg, Us, Vs, Eg, Pg, Ps, Ts);
    if (Id == 0)
        {
        std::cout << "Going to output the means.  \n";
        OutputMeans();
//        OutputMFIX(Data);
        }

    std::cout<< "Have Mean Velocities... "
             << Id << "\n";

    // Forms covariance matrices for both Gas and Solids & Combo
//    GetCorrMatrices(Data);
    for (j=0; j < Ny; j++)
        for (i=0; i < Nx; i++)
            {
            for (t=0;t < Nt; t++)
                {
                if (Id == (0 % NIds))
                    tGAu.push_back(Data.Gas.GetU(t, i, j) -
                    POD.UgAvgVel.at(i).at(j));
                if (Id == (1 % NIds))
                    tGAv.push_back(Data.Gas.GetV(t, i, j) -
                    POD.VgAvgVel.at(i).at(j));
                if (Id == (2 % NIds))
                    tPAu.push_back(Data.Particle.GetU(t, i, j) -
                    POD.UsAvgVel.at(i).at(j));
                if (Id == (3 % NIds))
                    tPAv.push_back(Data.Particle.GetV(t, i, j) -
                    POD.VsAvgVel.at(i).at(j));
                if (Id == (4 % NIds))
                    tGAP.push_back(Data.Gas.GetP(t, i, j) -
                    POD.PgAvg.at(i).at(j));
                if (Id == (5 % NIds))
                    tGAF.push_back(Data.Gas.GetEP(t, i, j) -
                    POD.FgAvg.at(i).at(j));
                if (Id == (6 % NIds))
                    tPAP.push_back(Data.Particle.GetP(t, i, j) -
```

```
        POD.PsAvg.at(i).at(j));
    if (Id == (7 % NIds))
        tPAT.push_back(Data.Particle.GetTheta(t, i, j) -
        POD.TsAvg.at(i).at(j));
    } // end for t
if (Id == (1 % NIds))
    {

    GAv.push_back(tGAv);
    tGAv.clear();
    }
if (Id == (0 % NIds))
    {
    GAu.push_back(tGAu);
    tGAu.clear();
    }
if (Id == (2 % NIds))
    {
    PAu.push_back(tPAu);
    tPAu.clear();
    }
if (Id == (3 % NIds))
    {
    PAv.push_back(tPAv);
    tPAv.clear();
    }
if (Id == (6 % NIds))
    {
    PAP.push_back(tPAP);
    tPAP.clear();
    }
if (Id == (7 % NIds))
    {
    PAT.push_back(tPAT);
    tPAT.clear();
    }
if (Id == (4 % NIds))
    {
    GAP.push_back(tGAP);
    tGAP.clear();
    }
if (Id == (5 % NIds))
    {
    GAF.push_back(tGAF);
    tGAF.clear();
```

```
            }
          } // end for i and j

    std::cout<< "Getting POD basis using SVD Decomposition "
             << Id << "\n";

    if (Id == (0 % NIds))
       {
       // POD Basis values are returned in the first
       // GUNBasis vectors of U = GPhiuPOD
       svd(GAu,POD.GPhiuPOD, MV, SigmaUg, Nx*Ny, Nt);
       AdjustSingVals(SigmaUg, Nt);
       Gceu = CalcEnergySVD(SigmaUg, GUNBasis,0);
       OutputSingVals(SigmaUg, 0);
       POD.SetNUG(GUNBasis);
       std::cout << Gceu
                 <<" captured energy of the Gas x velocity in "
              << GUNBasis << " basis functions.\n";
       // Keeping only the first m to form the basis
//       for (p=Nt; p > GUNBasis; p--)
//          GPhiuPOD.pop_back();
       std::cout << GUNBasis << " is how many I should have. \n "
                 << POD.GPhiuPOD.size() << " is how many I have. \n"
                 << "Singular value spread is "
                 << SigmaUg.at(0) << " to " << SigmaUg.at(GUNBasis-1)
                 << " . \n";
       OutputGUBasis(GUNBasis);
       std::cout << "Output basis vectors via " << Id << " \n";
       // Solving for the coefficients
       POD.PODsGasU.clear();
//       svd(POD.GPhiuPOD, U, V, W, Nx*Ny, GUNBasis);
//       AdjustSingVals(W, GUNBasis);
       X.clear();
       for (t = 0; t < Nt; t++)
          { //Assuming svd outputs V transpose and not V
//          std::cout << t << " POD coefficient calculation \n";
//          FormGub(Gub, t, Data);
//          std::cout << Id << " formed solution \n";
//          std::cout << Id << " survived coeff svd \n";
//          std::cout << Id << " adjusting SVs again \n";
//          Solve4coefs(U, V, W, Gub, Nx*Ny, GUNBasis, X);
//          std::cout << Id << "Have POD coefficients \n";
          for (k=0; k < GUNBasis; k++)
             X.push_back(SigmaUg.at(k) * MV.at(k).at(t));
          POD.PODsGasU.push_back(X);
```

```
            X.clear();
            } // end t
        }
    if (Id == (1 % NIds))
        {
        svd(GAv,POD.GPhivPOD, MV, SigmaVg, Nx*Ny, Nt);
        AdjustSingVals(SigmaVg, Nt);
        Gcev = CalcEnergySVD(SigmaVg, GVNBasis, 1);
        OutputSingVals(SigmaVg, 1);
        POD.SetNVG(GVNBasis);
        std::cout << Gcev
                <<" captured energy of the Gas y velocity in "
            << GVNBasis << " basis functions.\n";
//       for (p=Nt; p > GVNBasis; p--)
//           GPhivPOD.pop_back();
        OutputGVBasis(GVNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << GVNBasis << " is how many I should have and "
                << POD.GPhivPOD.size() << " is how many I have with a "
                << SigmaVg.at(0) << " to " << SigmaVg.at(GVNBasis-1)
                << " spread.";
        POD.PODsGasV.clear();
//       svd(POD.GPhivPOD, U, V, W, Nx*Ny, GVNBasis);
//       AdjustSingVals(W, GVNBasis);
        X.clear();
        for (t = 0; t < Nt; t++)
            {
//           FormGvb(Gvb, t, Data);
//           Solve4coefs(U, V, W, Gvb, Nx*Ny, GVNBasis, X);
            for (k=0; k < GVNBasis; k++)
               X.push_back(SigmaVg.at(k) * MV.at(k).at(t));
            POD.PODsGasV.push_back(X);
            X.clear();
            } // end t
        }

    if (Id == (2 % NIds))
        {
        svd(PAu,POD.PPhiuPOD, MV, SigmaUs, Nx*Ny, Nt);
        AdjustSingVals(SigmaUs, Nt);
        Pceu = CalcEnergySVD(SigmaUs, PUNBasis, 2);
        OutputSingVals(SigmaUs, 2);
        POD.SetNUS(PUNBasis);
        std::cout << Pceu
                <<" captured energy of the Particle x velocity in "
```

```
            << PUNBasis << " basis functions.\n";
//        for (p=Nt; p > PUNBasis; p--)
//            PPhiuPOD.pop_back();
        OutputPUBasis(PUNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << PUNBasis << " is how many I should have and "
                << POD.PPhiuPOD.size() << " is how many I have with a "
                << SigmaUs.at(0) << " to " << SigmaUs.at(PUNBasis-1)
                << " spread.";
        POD.PODsParticleU.clear();
//        svd(POD.PPhiuPOD, U, V, W, Nx*Ny, PUNBasis);
//        AdjustSingVals(W, PUNBasis);
        X.clear();
        for (t=0; t < Nt; t++)
            {
//            FormPub(Pub, t, Data);
//            Solve4coefs(U, V, W, Pub, Nx*Ny, PUNBasis, X);
            for (k=0; k < PUNBasis; k++)
                X.push_back(SigmaUs.at(k) * MV.at(k).at(t));
            POD.PODsParticleU.push_back(X);
            X.clear();
            } // end t
        }

    if (Id == (3 % NIds))
        {
        svd(PAv,POD.PPhivPOD, MV, SigmaVs, Nx*Ny, Nt);
        AdjustSingVals(SigmaVs, Nt);
        Pcev = CalcEnergySVD(SigmaVs, PVNBasis, 3);
        OutputSingVals(SigmaVs, 3);
        POD.SetNVS(PVNBasis);
        std::cout << Pcev
                <<" captured energy of the Particle y velocity in "
            << PVNBasis << " basis functions.\n";
//        for (p=Nt; p > PVNBasis; p--)
//            PPhivPOD.pop_back();
        OutputPVBasis(PVNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << PVNBasis << " is how many I should have and "
                << POD.PPhivPOD.size() << " is how many I have with a "
                << SigmaVs.at(0) << " to " << SigmaVs.at(PVNBasis-1)
                << " spread.";
        POD.PODsParticleV.clear();
//        svd(POD.PPhivPOD, U, V, W, Nx*Ny, PVNBasis);
//        AdjustSingVals(W, PVNBasis);
```

```
        X.clear();
        for (t=0; t < Nt; t++)
           {
//           FormPvb(Pvb, t, Data);
//           Solve4coefs(U, V, W, Pvb, Nx*Ny, PVNBasis, X);
           for (k=0; k < PVNBasis; k++)
              X.push_back(SigmaVs.at(k) * MV.at(k).at(t));
           POD.PODsParticleV.push_back(X);
           X.clear();
           } // end t
        }

     if (Id == (4 % NIds))
        {
//std::cout << "Starting Gas Pressure... \n";
        svd(GAP,POD.PgPhiPOD, MV, SigmaPg, Nx*Ny, Nt);
        AdjustSingVals(SigmaPg, Nt);
        Pgce = CalcEnergySVD(SigmaPg, GPNBasis, 5);
        OutputSingVals(SigmaPg, 4);
        POD.SetNPG(GPNBasis);
        std::cout << Pgce
                <<" captured energy of the Gas pressure in "
             << GPNBasis << " basis functions.\n";
//       for (p=Nt; p > GPNBasis; p--)
//          PgPhiPOD.pop_back();
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << GPNBasis << " is how many I should have and "
                << POD.PgPhiPOD.size() << " is how many I have with a "
                << SigmaPg.at(0) << " to " << SigmaPg.at(GPNBasis-1)
                << " spread.";
        OutputGPBasis(GPNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << GPNBasis << " is how many I should have and "
                << POD.PgPhiPOD.size() << " is how many I have with a "
                << SigmaPg.at(0) << " to " << SigmaPg.at(GPNBasis-1)
                << " spread.";
        POD.PODsGasP.clear();
//       svd(POD.PgPhiPOD, U, V, W, Nx*Ny, GPNBasis);
//       AdjustSingVals(W, GPNBasis);
        X.clear();
        for (t=0; t < Nt; t++)
           {
//           FormGpb(Gpb, t, Data);
//           Solve4coefs(U, V, W, Gpb, Nx*Ny, GPNBasis, X);
           for (k=0; k < GPNBasis; k++)
```

```
            X.push_back(SigmaPg.at(k) * MV.at(k).at(t));
          POD.PODsGasP.push_back(X);
          X.clear();
          } // end t
        }

    if (Id == (5 % NIds))
        {
        svd(GAF,POD.FgPhiPOD, MV, SigmaFg, Nx*Ny, Nt);
        AdjustSingVals(SigmaFg, Nt);
        Fgce = CalcEnergySVD(SigmaFg, GFNBasis,4);
        OutputSingVals(SigmaFg, 5);
        POD.SetNEG(GFNBasis);
        std::cout << Fgce
                <<" captured energy of the Gas fluid fraction in "
             << GFNBasis << " basis functions.\n";
//        for (p=Nt; p > GFNBasis; p--)
//           FgPhiPOD.pop_back();
        std::cout << GFNBasis << " is how many I should have and "
                << POD.FgPhiPOD.size() << " is how many I have with a "
                << SigmaFg.at(0) << " to " << SigmaFg.at(GFNBasis-1)
                << " spread.";
        OutputGFBasis(GFNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << GFNBasis << " is how many I should have and "
                << POD.FgPhiPOD.size() << " is how many I have with a "
                << SigmaFg.at(0) << " to " << SigmaFg.at(GFNBasis-1)
                << " spread.";
        POD.PODsGasF.clear();
//        svd(POD.FgPhiPOD, U, V, W, Nx*Ny, GFNBasis);
//        AdjustSingVals(W, GFNBasis);
        X.clear();
        for (t=0; t < Nt; t++)
           {
//           FormGfb(Gfb, t, Data);
//           Solve4coefs(U, V, W, Gfb, Nx*Ny, GFNBasis, X);
           for (k=0; k < GFNBasis; k++)
              X.push_back(SigmaFg.at(k) * MV.at(k).at(t));
           POD.PODsGasF.push_back(X);
           X.clear();
           } // end t
        }

    if (Id == (6 % NIds))
        {
```

```
        svd(PAP,POD.PsPhiPOD, MV, SigmaPs, Nx*Ny, Nt);
        AdjustSingVals(SigmaPs, Nt);
        Psce = CalcEnergySVD(SigmaPs, PPNBasis,6);
        OutputSingVals(SigmaPs, 6);
        POD.SetNPS(PPNBasis);
        std::cout << Psce
                <<" captured energy of the Particle pressure in "
             << PPNBasis << " basis functions.\n";
//      for (p=Nt; p > PPNBasis; p--)
//          PsPhiPOD.pop_back();
        OutputPPBasis(PPNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
        std::cout << PPNBasis << " is how many I should have and "
                << POD.PsPhiPOD.size() << " is how many I have with a "
                << SigmaPs.at(0) << " to " << SigmaPs.at(PPNBasis-1)
                << " spread.";
        POD.PODsParticleP.clear();
//      svd(POD.PsPhiPOD, U, V, W, Nx*Ny, PPNBasis);
//      AdjustSingVals(W, PPNBasis);
        X.clear();
        for (t=0; t < Nt; t++)
           {
//          FormPsb(Psb, t, Data);
//          Solve4coefs(U, V, W, Psb, Nx*Ny, PPNBasis, X);
           for (k=0; k < PPNBasis; k++)
              X.push_back(SigmaPs.at(k) * MV.at(k).at(t));
           POD.PODsParticleP.push_back(X);
           X.clear();
           } // end t
        }

     if (Id == (7 % NIds))
        {
        svd(PAT,POD.TsPhiPOD, MV, SigmaTs, Nx*Ny, Nt);
        AdjustSingVals(SigmaTs, Nt);
        Tsce = CalcEnergySVD(SigmaTs, TSNBasis, 7);
        OutputSingVals(SigmaTs, 7);
        POD.SetNTS(TSNBasis);
        std::cout << Tsce
                <<" captured energy of the Gas temperature in "
             << TSNBasis << " basis functions.\n";
//      for (p=Nt; p > TSNBasis; p--)
//          TsPhiPOD.pop_back();
        OutputPTBasis(TSNBasis);
        std::cout << "Output basis vectors via " << Id << " \n";
```

```
      std::cout << TSNBasis << " is how many I should have and "
                << POD.TsPhiPOD.size() << " is how many I have with a "
                << SigmaTs.at(0) << " to " << SigmaTs.at(TSNBasis-1)
                << " spread.";
      POD.PODsParticleT.clear();
//      svd(POD.TsPhiPOD, U, V, W, Nx*Ny, TSNBasis);
//      AdjustSingVals(W, TSNBasis);
      X.clear();
      for (t=0; t < Nt; t++)
         {
//         FormTsb(Tsb, t, Data);
//         Solve4coefs(U, V, W, Tsb, Nx*Ny, TSNBasis, X);
         for (k=0; k < TSNBasis; k++)
            X.push_back(SigmaTs.at(k) * MV.at(k).at(t));
         POD.PODsParticleT.push_back(X);
         X.clear();
         } // end t
      }

   MPI_Barrier(MPI_COMM_WORLD);
   OutputPODsSplit(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
                   GPNBasis, GFNBasis, PPNBasis, TSNBasis,
                   Id, NIds);
   std::cout << "Output PODs Coefficients \n";

   MPI_Barrier(MPI_COMM_WORLD);

   OutputKey(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
             GPNBasis, GFNBasis, PPNBasis, TSNBasis,
             Id, NIds);

   MPI_Barrier(MPI_COMM_WORLD);

   if (Id == (0 % NIds))
      std::cout << "Wrote key! \n";

   if (Id == 0)
      std::cout << "Have POD basis via SVD and "
                << "output them to files. Done!\n";
   MPI_Barrier(MPI_COMM_WORLD);

//   std::ifstream ifilekey;
//   int ikc;
//   for (ikc = 0; ikc < NIds; ikc++)
//      if (ikc == Id)
```

```
//        {
        std::string ln = "./Output";
//        ifilekey.open("./Output/Key.txt");
         InputKey(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
                GPNBasis, GFNBasis, PPNBasis, TSNBasis,
                ln);
//        ifilekey.close();
//        }

    MPI_Barrier(MPI_COMM_WORLD);

    if (Id == (0 % NIds))
        std::cout << "Read key! \n";

    MPI_Barrier(MPI_COMM_WORLD);

    if (Id == (0 % NIds))
        {
        std::string ln = "./Output";
        InputGVBasis(GVNBasis,ln);
        InputPUBasis(PUNBasis,ln);
        InputPVBasis(PVNBasis,ln);
        InputGFBasis(GFNBasis,ln);
        InputGPBasis(GPNBasis,ln);
        InputPPBasis(PPNBasis,ln);
        InputPTBasis(TSNBasis,ln);
        InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
                GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
        }
    if (Id == (1 % NIds))
        {
        std::string ln = "./Output";
        InputGUBasis(GUNBasis,ln);
        InputPUBasis(PUNBasis,ln);
        InputPVBasis(PVNBasis,ln);
        InputGFBasis(GFNBasis,ln);
        InputGPBasis(GPNBasis,ln);
        InputPPBasis(PPNBasis,ln);
        InputPTBasis(TSNBasis,ln);
        InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
                GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
        }
    if (Id == (2 % NIds))
        {
        std::string ln = "./Output";
```

```
    InputGUBasis(GUNBasis,ln);
    InputGVBasis(GVNBasis,ln);
    InputPVBasis(PVNBasis,ln);
    InputGFBasis(GFNBasis,ln);
    InputGPBasis(GPNBasis,ln);
    InputPPBasis(PPNBasis,ln);
    InputPTBasis(TSNBasis,ln);
    InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
    }
  if (Id == (3 % NIds))
    {
    std::string ln = "./Output";
    InputGUBasis(GUNBasis,ln);
    InputGVBasis(GVNBasis,ln);
    InputPUBasis(PUNBasis,ln);
    InputGFBasis(GFNBasis,ln);
    InputGPBasis(GPNBasis,ln);
    InputPPBasis(PPNBasis,ln);
    InputPTBasis(TSNBasis,ln);
    InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
    }
  if (Id == (4 % NIds))
    {
    std::string ln = "./Output";
    InputGUBasis(GUNBasis,ln);
    InputGVBasis(GVNBasis,ln);
    InputPUBasis(PUNBasis,ln);
    InputPVBasis(PVNBasis,ln);
    InputGFBasis(GFNBasis,ln);
    InputPPBasis(PPNBasis,ln);
    InputPTBasis(TSNBasis,ln);
    InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
            GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
    }
  if (Id == (5 % NIds))
    {
    std::string ln = "./Output";
    InputGUBasis(GUNBasis,ln);
    InputGVBasis(GVNBasis,ln);
    InputPUBasis(PUNBasis,ln);
    InputPVBasis(PVNBasis,ln);
    InputGPBasis(GPNBasis,ln);
    InputPPBasis(PPNBasis,ln);
```

```
        InputPTBasis(TSNBasis,ln);
        InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
                GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
        }
    if (Id == (6 % NIds))
        {
        std::string ln = "./Output";
        InputGUBasis(GUNBasis,ln);
        InputGVBasis(GVNBasis,ln);
        InputPUBasis(PUNBasis,ln);
        InputPVBasis(PVNBasis,ln);
        InputGFBasis(GFNBasis,ln);
        InputGPBasis(GPNBasis,ln);
        InputPTBasis(TSNBasis,ln);
        InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
                GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
        }
    if (Id == (7 % NIds))
        {
        std::string ln = "./Output";
        InputGUBasis(GUNBasis,ln);
        InputGVBasis(GVNBasis,ln);
        InputPUBasis(PUNBasis,ln);
        InputPVBasis(PVNBasis,ln);
        InputGFBasis(GFNBasis,ln);
        InputGPBasis(GPNBasis,ln);
        InputPPBasis(PPNBasis,ln);
        InputPODs(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
                GPNBasis, GFNBasis, PPNBasis, TSNBasis,ln);
        }

    MPI_Barrier(MPI_COMM_WORLD);

    if (Id == 0)
        std::cout << "Shared Results!\n";

    FormReconVars(GUNBasis, GVNBasis, PUNBasis, PVNBasis,
                GPNBasis, GFNBasis, PPNBasis, TSNBasis,
                Id, NIds);
    std::cout << "Formed Reconstructed Solution \n";
//    MPI_Barrier(MPI_COMM_WORLD);
    ApproxErrorSplit(Data, Id, NIds);
    std::cout << "Approximated Error \n";

//    MPI_Barrier(MPI_COMM_WORLD);
```

```
    OutputSolution(Id, NIds);
    std::cout << "Output Solutions \n";

    OutputErrorSplit(Id, NIds);

    MPI_Barrier(MPI_COMM_WORLD);

    std::cout << "Output Error \n";


    return good;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void PODSolver::GetMeanPODs(std::vector< std::vector< double > > & Ug,
                std::vector< std::vector< double > > & Vg,
                std::vector< std::vector< double > > & Us,
                std::vector< std::vector< double > > & Vs,
                std::vector< std::vector< double > > & Eg,
                std::vector< std::vector< double > > & Pg,
                std::vector< std::vector< double > > & Ps,
                std::vector< std::vector< double > > & Ts)
    {

    double sumUg;
    double sumVg;
    double sumUs;
    double sumVs;

    double sumPs;
    double sumPg;
    double sumFg;
    double sumTs;

//  double BLB;
//  std::cout << Gce << " captures the energy of the Gas phase.\n";


    int i;
    int j;
    int t;
```

```
std::vector< double > tUg;
std::vector< double > tVg;
std::vector< double > tUs;
std::vector< double > tVs;

std::vector< double > tPs;
std::vector< double > tTs;
std::vector< double > tPg;
std::vector< double > tFg;

POD.UgAvgVel.clear();
POD.VgAvgVel.clear();
POD.UsAvgVel.clear();
POD.VsAvgVel.clear();

POD.FgAvg.clear();
POD.PgAvg.clear();
POD.PsAvg.clear();
POD.TsAvg.clear();

// Doing this outside of the positional loop to save time
// Don't need this when using real velocities...
//    BLB = F.GetFBLB();

for (i=0; i < Nx; i++)
   {

   tUg.clear();
   tVg.clear();
   tUs.clear();
   tVs.clear();

   tFg.clear();
   tPg.clear();
   tPs.clear();
   tTs.clear();

   for (j=0; j < Ny; j++)
      {

      sumUg = 0.0;
      sumVg = 0.0;
      sumUs = 0.0;
      sumVs = 0.0;
```

```
            sumFg = 0.0;
            sumPg = 0.0;
            sumPs = 0.0;
            sumTs = 0.0;

            for (t=0; t< Nt; t++)
               {
               sumUg = sumUg + Ug.at(t).at((j*Nx) +i);
               sumVg = sumVg + Vg.at(t).at((j*Nx) +i);
               sumUs = sumUs + Us.at(t).at((j*Nx) +i);
               sumVs = sumVs + Vs.at(t).at((j*Nx) +i);

               sumFg = sumFg + Eg.at(t).at((j*Nx) +i);
               sumPg = sumPg + Pg.at(t).at((j*Nx) +i);
               sumPs = sumPs + Ps.at(t).at((j*Nx) +i);
               sumTs = sumTs + Ts.at(t).at((j*Nx) +i);

               } // end t

            tUg.push_back(sumUg/double(Nt));
            tVg.push_back(sumVg/double(Nt));
            tUs.push_back(sumUs/double(Nt));
            tVs.push_back(sumVs/double(Nt));

            tFg.push_back(sumFg/double(Nt));
            tPg.push_back(sumPg/double(Nt));
            tPs.push_back(sumPs/double(Nt));
            tTs.push_back(sumTs/double(Nt));

            } // end j
         POD.UgAvgVel.push_back(tUg);
         POD.VgAvgVel.push_back(tVg);
         POD.UsAvgVel.push_back(tUs);
         POD.VsAvgVel.push_back(tVs);
         POD.FgAvg.push_back(tFg);
         POD.PgAvg.push_back(tPg);
         POD.PsAvg.push_back(tPs);
         POD.TsAvg.push_back(tTs);
         } // end i

      std::cout << "Size of AvgVel = " << POD.UgAvgVel.size()
             << " and size of x = " << tUg.size() << "\n";
      }
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    NSSolver.h
//    Created by Stephanie R. Beck Roth
//    Last Modified:   05/19/2011
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#ifndef NSSOLVER_H
#define NSSOLVER_H

#include "mpi.h"
#include <iostream>
#include <fstream>
#include <string>
#include <math.h>
#include <vector>
#include "PhaseData.h"
#include "PODData.h"
#include "PODSolver.h"

class NSSolver
   {
   private:

   int MaxIts;
   double Tol;
   double Epsilon1;
   double Epsilon2;


   float Dt;

   bool ProjTime;
   bool Interp;
   bool Extrap;

   bool VarE;

   int vectordims;

   int NEQs;
   int NUVs;

   int NIds;
```

```
int Id;

int CurTS;

double CoR;
std::vector< double > lbc;

double GetPDUg4Ug(std::vector< double > & V, int i, int j);
double GetPDUg4Vg(std::vector< double > & V, int i, int j);
double GetPDUg4Us(std::vector< double > & V, int i, int j);
double GetPDUg4Vs(std::vector< double > & V, int i, int j);
double GetPDUg4Eg(std::vector< double > & V, int i, int j);
double GetPDUg4Pg(std::vector< double > & V, int i, int j);
double GetPDUg4Ps(std::vector< double > & V, int i, int j);
double GetPDUg4Ts(std::vector< double > & V, int i, int j);
double GetPDVg4Ug(std::vector< double > & V, int i, int j);
double GetPDVg4Vg(std::vector< double > & V, int i, int j);
double GetPDVg4Us(std::vector< double > & V, int i, int j);
double GetPDVg4Vs(std::vector< double > & V, int i, int j);
double GetPDVg4Eg(std::vector< double > & V, int i, int j);
double GetPDVg4Pg(std::vector< double > & V, int i, int j);
double GetPDVg4Ps(std::vector< double > & V, int i, int j);
double GetPDVg4Ts(std::vector< double > & V, int i, int j);
double GetPDUs4Ug(std::vector< double > & V, int i, int j);
double GetPDUs4Vg(std::vector< double > & V, int i, int j);
double GetPDUs4Us(std::vector< double > & V, int i, int j);
double GetPDUs4Vs(std::vector< double > & V, int i, int j);
double GetPDUs4Eg(std::vector< double > & V, int i, int j);
double GetPDUs4Pg(std::vector< double > & V, int i, int j);
double GetPDUs4Ps(std::vector< double > & V, int i, int j);
double GetPDUs4Ts(std::vector< double > & V, int i, int j);
double GetPDVs4Ug(std::vector< double > & V, int i, int j);
double GetPDVs4Vg(std::vector< double > & V, int i, int j);
double GetPDVs4Us(std::vector< double > & V, int i, int j);
double GetPDVs4Vs(std::vector< double > & V, int i, int j);
double GetPDVs4Eg(std::vector< double > & V, int i, int j);
double GetPDVs4Pg(std::vector< double > & V, int i, int j);
double GetPDVs4Ps(std::vector< double > & V, int i, int j);
double GetPDVs4Ts(std::vector< double > & V, int i, int j);

double DxUg(int x, int y);
double DyUg(int x, int y);
double DtUg(int x, int y);
double DxVg(int x, int y);
double DyVg(int x, int y);
```

```
double DtVg(int x, int y);
double DxUs(int x, int y);
double DyUs(int x, int y);
double DtUs(int x, int y);
double DxVs(int x, int y);
double DyVs(int x, int y);
double DtVs(int x, int y);
double DxEg(int x, int y);
double DyEg(int x, int y);
double DtEg(int x, int y);
double DxPg(int x, int y);
double DyPg(int x, int y);
double DtPg(int x, int y);
double DxPs(int x, int y);
double DyPs(int x, int y);
double DtPs(int x, int y);
double DxTs(int x, int y);
double DyTs(int x, int y);
double DtTs(int x, int y);

double DxUgInit(int x, int y);
double DyVgInit(int x, int y);

double DxUgi(int x, int y, int i);
double DyUgi(int x, int y, int i);
double DtUgi(int x, int y, int i);
double DxVgi(int x, int y, int i);
double DyVgi(int x, int y, int i);
double DtVgi(int x, int y, int i);
double DxUsi(int x, int y, int i);
double DyUsi(int x, int y, int i);
double DtUsi(int x, int y, int i);
double DxVsi(int x, int y, int i);
double DyVsi(int x, int y, int i);
double DtVsi(int x, int y, int i);
double DxEgi(int x, int y, int i);
double DyEgi(int x, int y, int i);
double DtEgi(int x, int y, int i);
double DxPgi(int x, int y, int i);
double DyPgi(int x, int y, int i);
double DtPgi(int x, int y, int i);
double DxPsi(int x, int y, int i);
double DyPsi(int x, int y, int i);
double DtPsi(int x, int y, int i);
double DxTsi(int x, int y, int i);
```

```
double DyTsi(int x, int y, int i);
double DtTsi(int x, int y, int i);

double I2DG(int x, int y);
double GetGom(int i);
double GetVrm(int i);
double GetRem(int i);
double DaiUgI2DG(int x, int y, int i);
double DaiVgI2DG(int x, int y, int i);
double DaiUgVrm(int x, int y, int i);
double DaiUgRem(int x, int y, int i);
double DaiVgVrm(int x, int y, int i);
double DaiVgRem(int x, int y, int i);
double DaiUsVrm(int x, int y, int i);
double DaiUsRem(int x, int y, int i);
double DaiVsVrm(int x, int y, int i);
double DaiVsRem(int x, int y, int i);
double DaiEgVrm(int x, int y, int i);
double DaiEgRem(int x, int y, int i);
double DaiUgMugt(int x, int y, int i);
double DaiUgBeta(int x, int y, int i);
double DaiVgMugt(int x, int y, int i);
double DaiVgBeta(int x, int y, int i);
double DaiUsBeta(int x, int y, int i);
double DaiVsBeta(int x, int y, int i);
double DaiEgMugt(int x, int y, int i);
double DaiEgBeta(int x, int y, int i);
double DaiEgMup(int x, int y, int i);
double DaiEgMums(int x, int y, int i);
double DaiTsMums(int x, int y, int i);

double DaiEgDxMub(int x, int y, int i);
double DaiTsDxMub(int x, int y, int i);
double DaiEgDyMub(int x, int y, int i);
double DaiTsDyMub(int x, int y, int i);

double DaiEgMub(int x, int y, int i);
double DaiTsMub(int x, int y, int i);
double DaiTsMup(int x, int y, int i);
double DaiTsBeta(int x, int y, int i);

double GetMugt(int x, int y);
double GetBeta(int x, int y);
double GetMub(int x, int y);
double GetMup(int x, int y);
```

```
double GetMums(int x, int y);
double GetE();

double DxMub(int x, int y);
double DyMub(int x, int y);

double DxxUs(int x, int y);
double DxxUsi(int x, int y, int i);
double DxxVsi(int x, int y, int i);
double DyyVs(int x, int y);

std::vector< double > SEG;
std::vector< double > SUG;
std::vector< double > SVG;
std::vector< double > SUS;
std::vector< double > SVS;
std::vector< double > SPG;
std::vector< double > SPS;
std::vector< double > STS;

// Note: There should be 1 more in the array than what is needed.
// This is due to the fact that Coeffs.at(0).at(*) contains
// the initialization solution.
std::vector< std::vector< double > > Coeffs;

float  GetDelt(float time);  // Checks the courant condition
void   Findf(std::vector< double > & x, std::vector< double > & y);
void   FormJac(std::vector< double > & x);
void   SetJA(std::vector<double> & x);
void   SetJA_NP(std::vector<double> & x);
void   JacTf(std::vector<double> & x, std::vector<double> & y);
double GetAmax(std::vector< std::vector< double >  > & A,
            int m, int n);
void   GetH(double mu, std::vector< double > & g,
          std::vector<double> & h);
double GetDenom(std::vector< double > & h,
             std::vector<double> & g, double mu);
double infnorm(std::vector< double > & x);
double norm(std::vector< double > & x);
double GetF(std::vector< double > & x);
void   VAdd(std::vector< double > & x1,
          std::vector< double > & x2,
          std::vector< double > & y);
void   LMM(std::vector< double > & x, double tol, int maxits,
         double ep1, double ep2);
```

```
    void   LMM_NP(std::vector< double > & x, double tol, int maxits,
                double ep1, double ep2);

    void   FormLUDecomp(std::vector< int > & index, double & d);
    void   DoLUBackSolve(std::vector< int > & index,
                    std::vector< double > & x);

    double GetUgSum(std::vector< double > & x, int i, int j);
    double GetVgSum(std::vector< double > & x, int i, int j);
    double GetUsSum(std::vector< double > & x, int i, int j);
    double GetVsSum(std::vector< double > & x, int i, int j);
    double GetEgSum(std::vector< double > & x, int i, int j);
    double GetPgSum(std::vector< double > & x, int i, int j);
    double GetPsSum(std::vector< double > & x, int i, int j);
    double GetTsSum(std::vector< double > & x, int i, int j);


//    float Jac2[];
    std::vector< std::vector< double > > Jac;
    std::vector< std::vector< double > > JA;

    double JAmaxval; //set in FormJA

    void FormTSums(std::vector< double > & x);

    void GetOldSol(std::vector< double > & V, int t);
    void FormInitOld(std::vector< double > & V, int ind);
    void WriteCoeffs(int NTPts);
    void WriteSol(int Nt);
    void WriteIndSol(int t);
    void WriteIndSol_NP(int t, int nt);
    void StoreSums(std::vector< double > & x);

    public:

       NSSolver();

       SimVars PHD;
       PODData POD;

    // For sums, access in format: .at(timepoint).at( x+(y*NX) )
    // For Projection the "timepoint" starts at the initial mfix
    //    start and continues through the projection time
    // For Extrapolation the "timepoint" runs through the mfix
    //    timepoints (similarly for interpolation)
```

```
    std::vector< std::vector< double > > UgSum;
    std::vector< std::vector< double > > VgSum;
    std::vector< std::vector< double > > UsSum;
    std::vector< std::vector< double > > VsSum;
    std::vector< std::vector< double > > EgSum;
    std::vector< std::vector< double > > PgSum;
    std::vector< std::vector< double > > PsSum;
    std::vector< std::vector< double > > TsSum;

      std::vector< double > DtVec;
      std::vector< int > IndexVec;

      void InitNS(SimVars & SV, PODData & PD, int type,
            std::string & evar,
            double & E, std::vector< double > & BC_L,
            std::vector< std::vector< double > > & Ug,
            std::vector< std::vector< double > > & Vg,
            std::vector< std::vector< double > > & Us,
            std::vector< std::vector< double > > & Vs,
            std::vector< std::vector< double > > & Eg,
            std::vector< std::vector< double > > & Pg,
            std::vector< std::vector< double > > & Ps,
            std::vector< std::vector< double > > & Ts,
            int my_id, int numpids, int maxits,
            double tol, double eps1, double eps2);

      ~NSSolver();

      void CallLMM(std::vector< double > & x,
                std::vector< double > & y);
      void CallLMM_NP(std::vector< double > & x,
                  std::vector< double > & y);

      void ExtrapSol(double value, bool vg, bool e);
      void ExtrapSol_NP(double value, bool vg, bool e);
      void ExtrapSolVg(double value);
//      void InterpSol2(double value, bool vg, bool e,
//                      SimVars & PHD2, PODData & POD2);
      void InterpSol(double value, bool vg, bool e,
                     SimVars & PHD2, PODData & POD2,
                     std::vector< std::vector< double > > & Ug2,
                     std::vector< std::vector< double > > & Vg2,
                     std::vector< std::vector< double > > & Us2,
                     std::vector< std::vector< double > > & Vs2,
                     std::vector< std::vector< double > > & Eg2,
```

```
                std::vector< std::vector< double > > & Pg2,
                std::vector< std::vector< double > > & Ps2,
                std::vector< std::vector< double > > & Ts2);
    void ProjectSol(double endtime);

    double EvalUg(int i, std::vector< double > & V);
    double EvalVg(int i, std::vector< double > & V);
    double EvalUs(int i, std::vector< double > & V);
    double EvalVs(int i, std::vector< double > & V);

    double PDUg(std::vector< double > & V, int i, int j);
    double PDVg(std::vector< double > & V, int i, int j);
    double PDUs(std::vector< double > & V, int i, int j);
    double PDVs(std::vector< double > & V, int i, int j);

    double AccessUgSum(int t, int i, int j);
    double AccessVgSum(int t, int i, int j);
    double AccessUsSum(int t, int i, int j);
    double AccessVsSum(int t, int i, int j);
    double AccessPgSum(int t, int i, int j);
    double AccessPsSum(int t, int i, int j);
    double AccessEgSum(int t, int i, int j);
    double AccessTsSum(int t, int i, int j);

    std::string int2String(int t);

};

#endif
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    NSSolver.cpp
//    Created by Stephanie R. Beck Roth
//    Last Modified:   05/19/2011
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#include "mpi.h"
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <math.h>
#include <cmath>
#include <vector>
#include "PhaseData.h"
#include "NSSolver.h"
#include "PODData.h"

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

NSSolver::NSSolver()

{

}

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::InitNS(SimVars & SV, PODData & PD, int type,
            std::string & evar,
            double & E, std::vector< double > & BC_L,
            std::vector< std::vector< double > > & Ug,
            std::vector< std::vector< double > > & Vg,
            std::vector< std::vector< double > > & Us,
            std::vector< std::vector< double > > & Vs,
            std::vector< std::vector< double > > & Eg,
            std::vector< std::vector< double > > & Pg,
            std::vector< std::vector< double > > & Ps,
            std::vector< std::vector< double > > & Ts,
            int my_id, int numpids, int maxits,
            double tol, double eps1, double eps2)
```

```
{

// x and y only - no Z direction

// remove this eventually...
    type = 0;
if (type == 0)
   {
   ProjTime = true;
   Interp = false;
   Extrap = false;
   }
else if (type == 1)
   {
   ProjTime = false;
   Interp = true;
   Extrap = false;
   }
else
   {
   if (type == 2)
      {
      ProjTime = false;
      Interp = false;
      Extrap = true;
      }
   }

PHD = SV;

POD = PD;

MaxIts = maxits;
Tol = tol;
Epsilon1 = eps1;
Epsilon2 = eps2;


if ((Extrap || Interp) && (evar == "E"))
   {
   VarE = true;
   CoR = E;
   }
else
```

```
        {
        VarE = false;
        CoR = PHD.GetE();
        }

        lbc = BC_L;

    vectordims = 2;

    // Note:  The 2 in the below equation corresponds to the
    //        number of phases...
    NEQs = PHD.GetNx() * PHD.GetNy() * vectordims * 2;

    NUVs =   POD.GetNUg() + POD.GetNVg() + POD.GetNUs() + POD.GetNVs()
          + POD.GetNEg() + POD.GetNPg() + POD.GetNPs() + POD.GetNTs();

    Id = my_id;
    NIds = numpids;

    UgSum = Ug;
    VgSum = Vg;
    UsSum = Us;
    VsSum = Vs;
    EgSum = Eg;
    PgSum = Pg;
    PsSum = Ps;
    TsSum = Ts;

    std::vector< double > t  (NUVs, 0.0);
    std::vector< std::vector< double > > td (NUVs, t);
    JA = td;

    CurTS = 0;

    td.clear();

// Make this a parameter in mfixpod data file...
//    Dt = PHD.Getdelt();
    Dt = .01;
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

NSSolver::~NSSolver()
```

```
    {

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxUgInit(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int lc;
    int rc;

    if (x == 0)
        lc = -1;
    else
        lc = (x-1) + ( POD.GetNx() * y);

    if (x == (POD.GetNx() - 1))
        rc = -1;
    else
        rc = (x+1) + ( POD.GetNx() * y);

//std::cout << "Accessing " << PHD.GetNt()-1 << "\n";

    if (lc == -1)
        val = (PHD.Gas.GetU(PHD.GetNt()-1,x+1,y) -
              PHD.Gas.GetU(PHD.GetNt()-1,x,y)) /
              PHD.Getdelx();
    else if (rc == -1)
        val = (PHD.Gas.GetU(PHD.GetNt()-1,x,y) -
              PHD.Gas.GetU(PHD.GetNt()-1,x-1,y)) /
              PHD.Getdelx();
    else
        val = (PHD.Gas.GetU(PHD.GetNt()-1,x+1,y) -
              PHD.Gas.GetU(PHD.GetNt()-1,x-1,y)) /
              (2.0 * PHD.Getdelx() );

    return val;
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyVgInit(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int bc;
    int uc;

    if (y == 0)
       bc = -1;
    else
       bc = x + (POD.GetNx() * (y - 1));

    if (y == (POD.GetNy() - 1))
       uc = -1;
    else
       uc = x + (POD.GetNx() * (y + 1));

    if (bc == -1)
       val = (PHD.Gas.GetV(PHD.GetNt()-1,x,y+1) -
            PHD.Gas.GetV(PHD.GetNt()-1,x,y)) /
            PHD.Getdely();
    else if (uc == -1)
       val = (PHD.Gas.GetV(PHD.GetNt()-1,x,y) -
            PHD.Gas.GetV(PHD.GetNt()-1,x,y-1)) /
            PHD.Getdely();
    else
       val = (PHD.Gas.GetV(PHD.GetNt()-1,x,y+1) -
            PHD.Gas.GetV(PHD.GetNt()-1,x,y-1)) /
            (2.0 * PHD.Getdely() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
double NSSolver::DxUg(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int lc;
    int rc;

    if (x == 0)
        lc = -1;
    else
        lc = (x-1) + ( POD.GetNx() * y);

    if (x == (POD.GetNx() - 1))
        rc = -1;
    else
        rc = (x+1) + ( POD.GetNx() * y);

    if (lc == -1)
        val = (SUG.at(rc) - SUG.at(cell) ) /
              PHD.Getdelx();
    else if (rc == -1)
        val = (SUG.at(cell) - SUG.at(lc) ) /
              PHD.Getdelx();
    else
        val = (SUG.at(rc) - SUG.at(lc)) /
               (2.0 * PHD.Getdelx() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyUg(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int bc;
```

```
    int uc;

    if (y == 0)
       bc = -1;
    else
       bc = x + (POD.GetNx() * (y - 1));

    if (y == (POD.GetNy() - 1))
       uc = -1;
    else
       uc = x + (POD.GetNx() * (y + 1));

    if (bc == -1)
       val = (SUG.at(uc) - SUG.at(cell) ) /
            PHD.Getdely();
    else if (uc == -1)
       val = (SUG.at(cell) - SUG.at(bc) ) /
            PHD.Getdely();
    else
       val = (SUG.at(uc) - SUG.at(bc)) /
             (2.0 * PHD.Getdely() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtUg(int x, int y)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;
//    int sz = UgSum.size();

//   ts == 0 is the check for the initial timestep
//   realization is that the initial timestep is ALWAYS the initial
//   condition alleviates this...

//    if (sz < 2)
//        val = 0.0;
//    else
```

```
    if (VarE)
        val = (PHD.Gas.GetU(CurTS,x,y) -
               PHD.Gas.GetU(CurTS-1,x,y) ) /
               Dt;
    else
        val = ( SUG.at(cell) - UgSum.back().at(cell) ) /
            Dt;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxVg(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int lc;
    int rc;

    if (x == 0)
        lc = -1;
    else
        lc = (x-1) + ( POD.GetNx() * y);

    if (x == (POD.GetNx() - 1))
        rc = -1;
    else
        rc = (x+1) + ( POD.GetNx() * y);

    if (lc == -1)
        val = (SVG.at(rc) - SVG.at(cell) ) /
              PHD.Getdelx();
    else if (rc == -1)
        val = (SVG.at(cell) - SVG.at(lc) ) /
              PHD.Getdelx();
    else
        val = (SVG.at(rc) - SVG.at(lc)) /
               (2.0 * PHD.Getdelx() );
```

```
    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyVg(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int bc;
    int uc;

    if (y == 0)
       bc = -1;
    else
       bc = x + (POD.GetNx() * (y - 1));

    if (y == (POD.GetNy() - 1))
       uc = -1;
    else
       uc = x + (POD.GetNx() * (y + 1));

    if (bc == -1)
       val = (SVG.at(uc) - lbc.at(cell) ) /
            (2.0 * PHD.Getdely() );
//        val = (VgSum.back().at(uc) - VgSum.back().at(cell) ) /
//              PHD.Getdely();
    else if (uc == -1)
       val = (SVG.at(cell) - SVG.at(bc) ) /
            PHD.Getdely();
    else
       val = (SVG.at(uc) - SVG.at(bc)) /
            (2.0 * PHD.Getdely() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
double NSSolver::DtVg(int x, int y)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;
//    int sz = VgSum.size();

//    if (sz < 2)
//        val = 0.0;
//    else

    if (VarE)
        val = (PHD.Gas.GetV(CurTS,x,y) -
              PHD.Gas.GetV(CurTS-1,x,y) ) /
              Dt;
    else
        val = ( SVG.at(cell) - VgSum.back().at(cell) ) /
          Dt;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxUs(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int lc;
    int rc;

    if (x == 0)
        lc = -1;
    else
        lc = (x-1) + ( POD.GetNx() * y);

    if (x == (POD.GetNx() - 1))
        rc = -1;
```

```
    else
        rc = (x+1) + ( POD.GetNx() * y);

    if (lc == -1)
        val = (SUS.at(rc) - SUS.at(cell) ) /
                PHD.Getdelx();
    else if (rc == -1)
        val = (SUS.at(cell) - SUS.at(lc) ) /
                PHD.Getdelx();
    else
        val = (SUS.at(rc) - SUS.at(lc)) /
                (2.0 * PHD.Getdelx() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyUs(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int bc;
    int uc;

    if (y == 0)
        bc = -1;
    else
        bc = x + (POD.GetNx() * (y - 1));

    if (y == (POD.GetNy() - 1))
        uc = -1;
    else
        uc = x + (POD.GetNx() * (y + 1));

    if (bc == -1)
        val = (SUS.at(uc) - SUS.at(cell) ) /
                PHD.Getdely();
    else if (uc == -1)
        val = (SUS.at(cell) - SUS.at(bc) ) /
```

```
            PHD.Getdely();
    else
        val = (SUS.at(uc) - SUS.at(bc)) /
            (2.0 * PHD.Getdely() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtUs(int x, int y)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;
//    int sz = UsSum.size();

//    if (sz < 2)
//        val = 0.0;
//    else

    if (VarE)
        val = (PHD.Particle.GetU(CurTS,x,y) -
            PHD.Particle.GetU(CurTS-1,x,y) ) /
            Dt;
    else
        val = ( SUS.at(cell) - UsSum.back().at(cell) ) /
          Dt;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxVs(int x, int y)

    {

    double val;
```

```
    int cell = (y * POD.GetNx() ) + x;
    int lc;
    int rc;

    if (x == 0)
       lc = -1;
    else
       lc = (x-1) + ( POD.GetNx() * y);

    if (x == (POD.GetNx() - 1))
       rc = -1;
    else
       rc = (x+1) + ( POD.GetNx() * y);

    if (lc == -1)
       val = (SVS.at(rc) - SVS.at(cell) ) /
            PHD.Getdelx();
    else if (rc == -1)
       val = (SVS.at(cell) - SVS.at(lc) ) /
            PHD.Getdelx();
    else
       val = (SVS.at(rc) - SVS.at(lc)) /
            (2.0 * PHD.Getdelx() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyVs(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int bc;
    int uc;

    if (y == 0)
       bc = -1;
    else
       bc = x + (POD.GetNx() * (y - 1));
```

```
    if (y == (POD.GetNy() - 1))
        uc = -1;
    else
        uc = x + (POD.GetNx() * (y + 1));

    if (bc == -1)
        val = (SVS.at(uc) - SVS.at(cell) ) /
              PHD.Getdely();
    else if (uc == -1)
        val = (SVS.at(cell) - SVS.at(bc) ) /
              PHD.Getdely();
    else
        val = (SVS.at(uc) - SVS.at(bc)) /
              (2.0 * PHD.Getdely() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtVs(int x, int y)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;
//  int sz = VsSum.size();

//  if (sz < 2)
//      val = 0.0;
//  else

    if (VarE)
        val = (PHD.Particle.GetV(CurTS,x,y) -
              PHD.Particle.GetV(CurTS-1,x,y) ) /
              Dt;
    else
        val = ( SVS.at(cell) - VsSum.back().at(cell) ) /
            Dt;

    return val;
```

```
   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxEg(int x, int y)

   {

   double val;

   int cell = (y * POD.GetNx() ) + x;
   int lc;
   int rc;

   if (x == 0)
      lc = -1;
   else
      lc = (x-1) + ( POD.GetNx() * y);

   if (x == (POD.GetNx() - 1))
      rc = -1;
   else
      rc = (x+1) + ( POD.GetNx() * y);

   if (lc == -1)
      val = (SEG.at(rc) - SEG.at(cell) ) /
           PHD.Getdelx();
   else if (rc == -1)
      val = (SEG.at(cell) - SEG.at(lc) ) /
           PHD.Getdelx();
   else
      val = (SEG.at(rc) - SEG.at(lc)) /
           (2.0 * PHD.Getdelx() );

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyEg(int x, int y)

   {
```

```
    double val;

    int cell = (y * POD.GetNx() ) + x;
    int bc;
    int uc;

    if (y == 0)
       bc = -1;
    else
       bc = x + (POD.GetNx() * (y - 1));

    if (y == (POD.GetNy() - 1))
       uc = -1;
    else
       uc = x + (POD.GetNx() * (y + 1));

    if (bc == -1)
       val = (SEG.at(uc) - SEG.at(cell) ) /
             PHD.Getdely();
    else if (uc == -1)
       val = (SEG.at(cell) - SEG.at(bc) ) /
             PHD.Getdely();
    else
       val = (SEG.at(uc) - SEG.at(bc)) /
             (2.0 * PHD.Getdely() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtEg(int x, int y)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;
//    int sz = EgSum.size();

//    if (sz < 2)
//       val = 0.0;
//    else
```

```
    if (VarE)
       val = (PHD.Gas.GetEP(CurTS,x,y) -
             PHD.Gas.GetEP(CurTS-1,x,y) ) /
             Dt;
    else
       val = ( SEG.at(cell) - EgSum.back().at(cell) ) /
          Dt;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxPg(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int lc;
    int rc;

    if (x == 0)
       lc = -1;
    else
       lc = (x-1) + ( POD.GetNx() * y);

    if (x == (POD.GetNx() - 1))
       rc = -1;
    else
       rc = (x+1) + ( POD.GetNx() * y);

    if (lc == -1)
       val = (SPG.at(rc) - SPG.at(cell) ) /
             PHD.Getdelx();
    else if (rc == -1)
       val = (SPG.at(cell) - SPG.at(lc) ) /
             PHD.Getdelx();
    else
       val = (SPG.at(rc) - SPG.at(lc)) /
             (2.0 * PHD.Getdelx() );
```

```
    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyPg(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int bc;
    int uc;

    if (y == 0)
        bc = -1;
    else
        bc = x + (POD.GetNx() * (y - 1));

    if (y == (POD.GetNy() - 1))
        uc = -1;
    else
        uc = x + (POD.GetNx() * (y + 1));

    if (bc == -1)
        val = (SPG.at(uc) - SPG.at(cell) ) /
              PHD.Getdely();
    else if (uc == -1)
        val = (SPG.at(cell) - SPG.at(bc) ) /
              PHD.Getdely();
    else
        val = (SPG.at(uc) - SPG.at(bc)) /
              (2.0 * PHD.Getdely() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
double NSSolver::DtPg(int x, int y)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;
//    int sz = PgSum.size();

//    if (sz < 2)
//        val = 0.0;
//    else

    if (VarE)
        val = (PHD.Gas.GetP(CurTS,x,y) -
              PHD.Gas.GetP(CurTS-1,x,y) ) /
              Dt;
    else
          val = ( SPG.at(cell) - PgSum.back().at(cell) ) /
                Dt;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxPs(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int lc;
    int rc;

    if (x == 0)
        lc = -1;
    else
        lc = (x-1) + ( POD.GetNx() * y);

    if (x == (POD.GetNx() - 1))
        rc = -1;
    else
```

```
        rc = (x+1) + ( POD.GetNx() * y);

    if (lc == -1)
        val = (SPS.at(rc) - SPS.at(cell) ) /
                PHD.Getdelx();
    else if (rc == -1)
        val = (SPS.at(cell) - SPS.at(lc) ) /
                PHD.Getdelx();
    else
        val = (SPS.at(rc) - SPS.at(lc)) /
                (2.0 * PHD.Getdelx() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyPs(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int bc;
    int uc;

    if (y == 0)
        bc = -1;
    else
        bc = x + (POD.GetNx() * (y - 1));

    if (y == (POD.GetNy() - 1))
        uc = -1;
    else
        uc = x + (POD.GetNx() * (y + 1));

    if (bc == -1)
        val = (SPS.at(uc) - SPS.at(cell) ) /
                PHD.Getdely();
    else if (uc == -1)
        val = (SPS.at(cell) - SPS.at(bc) ) /
                PHD.Getdely();
```

```
    else
        val = (SPS.at(uc) - SPS.at(bc)) /
                (2.0 * PHD.Getdely() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtPs(int x, int y)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;
//    int sz = PsSum.size();

//    if (sz < 2)
//        val = 0.0;
//    else


    if (VarE)
        val = (PHD.Particle.GetP(CurTS,x,y) -
                PHD.Particle.GetP(CurTS-1,x,y) ) /
                Dt;
    else
        val = ( SPS.at(cell) - PsSum.back().at(cell) ) /
                 Dt;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxTs(int x, int y)

    {

    double val;
```

```
    int cell = (y * POD.GetNx() ) + x;
    int lc;
    int rc;

    if (x == 0)
       lc = -1;
    else
       lc = (x-1) + ( POD.GetNx() * y);

    if (x == (POD.GetNx() - 1))
       rc = -1;
    else
       rc = (x+1) + ( POD.GetNx() * y);

    if (lc == -1)
       val = (STS.at(rc) - STS.at(cell) ) /
             PHD.Getdelx();
    else if (rc == -1)
       val = (STS.at(cell) - STS.at(lc) ) /
             PHD.Getdelx();
    else
       val = (STS.at(rc) - STS.at(lc)) /
             (2.0 * PHD.Getdelx() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyTs(int x, int y)

    {

    double val;

    int cell = (y * POD.GetNx() ) + x;
    int bc;
    int uc;

    if (y == 0)
       bc = -1;
    else
       bc = x + (POD.GetNx() * (y - 1));
```

```
    if (y == (POD.GetNy() - 1))
        uc = -1;
    else
        uc = x + (POD.GetNx() * (y + 1));

    if (bc == -1)
        val = (STS.at(uc) - STS.at(cell) ) /
              PHD.Getdely();
    else if (uc == -1)
        val = (STS.at(cell) - STS.at(bc) ) /
              PHD.Getdely();
    else
        val = (STS.at(uc) - STS.at(bc)) /
              (2.0 * PHD.Getdely() );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtTs(int x, int y)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;
//    int sz = TsSum.size();

//    if (sz < 2)
//        val = 0.0;
//    else

    if (VarE)
        val = (PHD.Particle.GetTheta_m(CurTS,x,y) -
              PHD.Particle.GetTheta_m(CurTS-1,x,y) ) /
              Dt;
    else
        val = ( STS.at(cell) - TsSum.back().at(cell) ) /
            Dt;

    return val;
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxUgi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

//std::cout << i << " is time & "
//          << POD.GPhiuPOD.size() << " is the size. \n";
//std::cout << i << " " << x << "," << y << "\n";

    if (x == 0)
       val = (POD.GPhiuPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
             POD.GPhiuPOD.at( cell ).at(i) ) / PHD.Getdelx();
    else if (x == (POD.GetNx() - 1))
       val = (POD.GPhiuPOD.at( cell ).at(i) -
             POD.GPhiuPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
             PHD.Getdelx();
    else
       val = (POD.GPhiuPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
             POD.GPhiuPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
            ( PHD.Getdelx() * 2.0 );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyUgi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (y == 0)
       val = (POD.GPhiuPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
             POD.GPhiuPOD.at( cell ).at(i) ) / PHD.Getdely();
```

```
    else if (y == (POD.GetNy() - 1))
       val = (POD.GPhiuPOD.at( cell ).at(i) -
            POD.GPhiuPOD.at( ((y-2) * POD.GetNx() ) + x ).at(i) ) /
            PHD.Getdely();
    else
       val = (POD.GPhiuPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
            POD.GPhiuPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
           ( PHD.Getdely() * 2.0 );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtUgi(int x, int y, int i)

    {

    double val;

    val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxVgi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (x == 0)
       val = (POD.GPhivPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
            POD.GPhivPOD.at( cell ).at(i) ) / PHD.Getdelx();
    else if (x == (POD.GetNx() - 1))
       val = (POD.GPhivPOD.at( cell ).at(i) -
            POD.GPhivPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
            PHD.Getdelx();
```

```
    else
       val = (POD.GPhivPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
            POD.GPhivPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
           ( PHD.Getdelx() * 2.0 );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyVgi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (y == 0)
       val = (POD.GPhivPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
            POD.GPhivPOD.at( cell ).at(i) ) / PHD.Getdely();
    else if (y == (POD.GetNy() - 1))
       val = (POD.GPhivPOD.at( cell ).at(i) -
            POD.GPhivPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
            PHD.Getdely();
    else
       val = (POD.GPhivPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
            POD.GPhivPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
           ( PHD.Getdely() * 2.0 );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtVgi(int x, int y, int i)

    {

    double val;

    val = 0.0;
```

```
    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxUsi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (x == 0)
       val = (POD.PPhiuPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
             POD.PPhiuPOD.at( cell ).at(i) ) / PHD.Getdelx();
    else if (x == (POD.GetNx() - 1))
       val = (POD.PPhiuPOD.at( cell ).at(i) -
             POD.PPhiuPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
             PHD.Getdelx();
    else
       val = (POD.PPhiuPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
             POD.PPhiuPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
            ( PHD.Getdelx() * 2.0 );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyUsi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (y == 0)
       val = (POD.PPhiuPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
             POD.PPhiuPOD.at( cell ).at(i) ) / PHD.Getdely();
    else if (y == (POD.GetNy() - 1))
```

```
        val = (POD.PPhiuPOD.at( cell ).at(i) -
             POD.PPhiuPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
             PHD.Getdely();
    else
        val = (POD.PPhiuPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
             POD.PPhiuPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
            ( PHD.Getdely() * 2.0 );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtUsi(int x, int y, int i)

    {

    double val;

    val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxVsi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (x == 0)
        val = (POD.PPhivPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
             POD.PPhivPOD.at( cell ).at(i) ) / PHD.Getdelx();
    else if (x == (POD.GetNx() - 1))
        val = (POD.PPhivPOD.at( cell ).at(i) -
             POD.PPhivPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
             PHD.Getdelx();
    else
```

```
      val = (POD.PPhivPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
            POD.PPhivPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
           ( PHD.Getdelx() * 2.0 );

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyVsi(int x, int y, int i)

   {

   double val;
   int cell = (y * POD.GetNx() ) + x;

   if (y == 0)
      val = (POD.PPhivPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
            POD.PPhivPOD.at( cell ).at(i) ) / PHD.Getdely();
   else if (y == (POD.GetNy() - 1))
      val = (POD.PPhivPOD.at( cell ).at(i) -
            POD.PPhivPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
            PHD.Getdely();
   else
      val = (POD.PPhivPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
            POD.PPhivPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
           ( PHD.Getdely() * 2.0 );

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtVsi(int x, int y, int i)

   {

   double val;

   val = 0.0;
```

```
    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxEgi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (x == 0)
        val = (POD.FgPhiPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
              POD.FgPhiPOD.at( cell ).at(i) ) / PHD.Getdelx();
    else if (x == (POD.GetNx() - 1))
        val = (POD.FgPhiPOD.at( cell ).at(i) -
              POD.FgPhiPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
              PHD.Getdelx();
    else
        val = (POD.FgPhiPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
              POD.FgPhiPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
             ( PHD.Getdelx() * 2.0 );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyEgi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (y == 0)
        val = (POD.FgPhiPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
              POD.FgPhiPOD.at( cell ).at(i) ) / PHD.Getdely();
    else if (y == (POD.GetNy() - 1))
        val = (POD.FgPhiPOD.at( cell ).at(i) -
```

```
            POD.FgPhiPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
            PHD.Getdely();
    else
        val = (POD.FgPhiPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
            POD.FgPhiPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
           ( PHD.Getdely() * 2.0 );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtEgi(int x, int y, int i)

    {

    double val;

    val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxPgi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (x == 0)
        val = (POD.PgPhiPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
            POD.PgPhiPOD.at( cell ).at(i) ) / PHD.Getdelx();
    else if (x == (POD.GetNx() - 1))
        val = (POD.PgPhiPOD.at( cell ).at(i) -
            POD.PgPhiPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
            PHD.Getdelx();
    else
        val = (POD.PgPhiPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
```

```
            POD.PgPhiPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
           ( PHD.Getdelx() * 2.0 );

   return val;

   }
```

/*%%*/

```
double NSSolver::DyPgi(int x, int y, int i)

   {

   double val;
   int cell = (y * POD.GetNx() ) + x;

   if (y == 0)
      val = (POD.PgPhiPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
            POD.PgPhiPOD.at( cell ).at(i) ) / PHD.Getdely();
   else if (y == (POD.GetNy() - 1))
      val = (POD.PgPhiPOD.at( cell ).at(i) -
            POD.PgPhiPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
            PHD.Getdely();
   else
      val = (POD.PgPhiPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
            POD.PgPhiPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
           ( PHD.Getdely() * 2.0 );

   return val;

   }
```

/*%%*/

```
double NSSolver::DtPgi(int x, int y, int i)

   {

   double val;

   val = 0.0;

   return val;
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxPsi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (x == 0)
        val = (POD.PsPhiPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
            POD.PsPhiPOD.at( cell ).at(i) ) / PHD.Getdelx();
    else if (x == (POD.GetNx() - 1))
        val = (POD.PsPhiPOD.at( cell ).at(i) -
            POD.PsPhiPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
            PHD.Getdelx();
    else
        val = (POD.PsPhiPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
            POD.PsPhiPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
           ( PHD.Getdelx() * 2.0 );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyPsi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (y == 0)
        val = (POD.PsPhiPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
            POD.PsPhiPOD.at( cell ).at(i) ) / PHD.Getdely();
    else if (y == (POD.GetNy() - 1))
        val = (POD.PsPhiPOD.at( cell ).at(i) -
            POD.PsPhiPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
```

```
            PHD.Getdely();
    else
       val = (POD.PsPhiPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
            POD.PsPhiPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
           ( PHD.Getdely() * 2.0 );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtPsi(int x, int y, int i)

    {

    double val;

    val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxTsi(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx() ) + x;

    if (x == 0)
       val = (POD.TsPhiPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
            POD.TsPhiPOD.at( cell ).at(i) ) / PHD.Getdelx();
    else if (x == (POD.GetNx() - 1))
       val = (POD.TsPhiPOD.at( cell ).at(i) -
            POD.TsPhiPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
            PHD.Getdelx();
    else
       val = (POD.TsPhiPOD.at( (y * POD.GetNx() ) + x+1 ).at(i) -
            POD.TsPhiPOD.at( (y * POD.GetNx() ) + x-1 ).at(i) ) /
```

```
        ( PHD.Getdelx() * 2.0 );

  return val;

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyTsi(int x, int y, int i)

  {

  double val;
  int cell = (y * POD.GetNx() ) + x;

  if (y == 0)
    val = (POD.TsPhiPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
          POD.TsPhiPOD.at( cell ).at(i) ) / PHD.Getdely();
  else if (y == (POD.GetNy() - 1))
    val = (POD.TsPhiPOD.at( cell ).at(i) -
          POD.TsPhiPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
          PHD.Getdely();
  else
    val = (POD.TsPhiPOD.at( ((y+1) * POD.GetNx() ) + x ).at(i) -
          POD.TsPhiPOD.at( ((y-1) * POD.GetNx() ) + x ).at(i) ) /
         ( PHD.Getdely() * 2.0 );

  return val;

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DtTsi(int x, int y, int i)

  {

  double val;

  val = 0.0;

  return val;
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiUgI2DG(int x, int y, int i)

    {

    double val;

//std::cout << DxUg(x,y) << " is DxUg \n";
//std::cout << DxUgi(x,y,i) << " is DxUgi \n";
//std::cout << DyUg(x,y) << " is DyUg \n";
//std::cout << DyUgi(x,y,i) << " is DyUgi \n";
//std::cout << DxVg(x,y) << " is DxVg \n";
//std::cout << DyVgi(x,y,i) << " is DyVgi \n";


    val = (4.0 * DxUg(x,y) * DxUgi(x,y,i)) - (2.0 * DxUgi(x,y,i)
          * DyVg(x,y)) + (2.0 * DyUg(x,y) * DyUgi(x,y,i))
          + (DyUgi(x,y,i) * DxVg(x,y));

//    val = (2.0 * DxUg(x,y) * DxUgi(x,y,i)) - (2.0 * DxUgi(x,y,i)
//          * DyVgi(x,y,i)) + (2.0 * DyUg(x,y) * DyUgi(x,y,i))
//          + (DyUgi(x,y,i) * DxVg(x,y));

    if ( isnan(val) )
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiUgMugt(int x, int y, int i)

    {

    double val;
    int cell = (y * POD.GetNx()) + x;

//std::cout << "In DaiUgMugt \n";
```

```
//std::cout << I2DG(x,y) << " is I2DG \n";
//std::cout << DaiUgI2DG(x,y,i) << " is DaiUgI2DG \n";

   val = pow(PHD.Gas.Getlscale(), 2.0) * SEG.at(cell)
        * PHD.Gas.GetDensity()
        * DaiUgI2DG(x,y,i) / sqrt(I2DG(x,y));

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiUgBeta(int x, int y, int i)

   {

   double val;
   int cell = (y * POD.GetNx()) + x;

   val = (( (-9.0/4.0) * (1.0 - SEG.at(cell)) * SEG.at(cell)
         * PHD.Gas.GetDensity() /
        PHD.Particle.GetDiam() * pow(GetVrm(cell), 3.0)
        * DaiUgVrm(x,y,i))
        * pow( 0.63 + (4.8 * sqrt(GetVrm(cell)/GetRem(cell)) ), 2.0) *
        sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0)
        + pow(SVG.at(cell) - SVS.at(cell), 2.0) )) +
       ((( (3.0 * (1.0 - SEG.at(cell)) * SEG.at(cell)
          * PHD.Gas.GetDensity() ) /
         (4.0 * PHD.Particle.GetDiam()) * pow(GetVrm(cell), 2.0))) *
        ( 2.0 * (0.63 + (4.8 * sqrt(GetVrm(cell)) /
            sqrt(GetRem(cell)) ))
         * ( (0.5 * DaiUgVrm(x,y,i) / sqrt(GetVrm(cell))) -
             (0.5 * pow(GetRem(cell), -3.0/2.0) * DaiUgRem(x,y,i))))*
        sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0) + pow(SVG.at(cell)
             - SVS.at(cell), 2.0) )) +
       (( (SUG.at(cell) - SUS.at(cell)) * POD.GPhiuPOD.at(cell).at(i) /
        ( sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0) + pow(SVG.at(cell)
            - SVS.at(cell), 2.0)) )) *
        pow( 0.63 + (4.8 * sqrt(GetVrm(cell)/GetRem(cell)) ), 2.0) *
        ( (3.0 * (1.0 - SEG.at(cell)) * SEG.at(cell)
```

```
            * PHD.Gas.GetDensity() ) /
          (4.0 * PHD.Particle.GetDiam()) * pow(GetVrm(cell), 2.0)));

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiVgI2DG(int x, int y, int i)

   {

   double val;

   val = (-2.0 * DxUg(x,y) * DyVgi(x,y,i)) + (4.0 * DyVg(x,y)
        * DyVgi(x,y,i)) + (2.0 * DxVg(x,y) * DxVgi(x,y,i))
        + (DxVgi(x,y,i) * DyUg(x,y));

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiVgMugt(int x, int y, int i)

   {

   double val;
   int cell = (y * POD.GetNx() ) + x;

   val = pow(PHD.Gas.Getlscale(), 2.0) * SEG.at(cell)
        * PHD.Gas.GetDensity()
        * DaiVgI2DG(x,y,i) / sqrt(I2DG(x,y));

   if ( isnan(val) )
      val = 0.0;

   return val;
```

```
   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiVgBeta(int x, int y, int i)

   {

   double val;
   int cell = (y * POD.GetNx()) + x;

   val = (( (-9.0/4.0) * (1.0 - SEG.at(cell)) * SEG.at(cell)
          * PHD.Gas.GetDensity() /
         PHD.Particle.GetDiam() * pow(GetVrm(cell), 3.0)
         * DaiVgVrm(x,y,i))
         * pow( 0.63 + (4.8 * sqrt(GetVrm(cell)/GetRem(cell)) ), 2.0) *
         sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0)
         + pow(SVG.at(cell) - SVS.at(cell), 2.0) )) +
        ((( (3.0 * (1.0 - SEG.at(cell)) * SEG.at(cell)
            * PHD.Gas.GetDensity() ) /
          (4.0 * PHD.Particle.GetDiam()) * pow(GetVrm(cell), 2.0))) *
         ( 2.0 * (0.63 + (4.8 * sqrt(GetVrm(cell)) /
              sqrt(GetRem(cell)) ))
          * ( (0.5 * DaiVgVrm(x,y,i) / sqrt(GetVrm(cell))) -
               (0.5 * pow(GetRem(cell), -3.0/2.0) * DaiVgRem(x,y,i))))*
         sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0) + pow(SVG.at(cell)
               - SVS.at(cell), 2.0) )) +
        (( (SUG.at(cell) - SUS.at(cell)) * POD.GPhivPOD.at(cell).at(i) /
         ( sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0) + pow(SVG.at(cell)
              - SVS.at(cell), 2.0)) )) *
         pow( 0.63 + (4.8 * sqrt(GetVrm(cell)/GetRem(cell)) ), 2.0) *
         ( (3.0 * (1.0 - SEG.at(cell)) * SEG.at(cell)
             * PHD.Gas.GetDensity() ) /
          (4.0 * PHD.Particle.GetDiam()) * pow(GetVrm(cell), 2.0)));

   if ( isnan(val) )
      val = 0.0;

   return val;

   }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiUsBeta(int x, int y, int i)

   {

   double val;
   int cell = (y * POD.GetNx()) + x;

   val = (( (-9.0/4.0) * (1.0 - SEG.at(cell)) * SEG.at(cell)
        * PHD.Gas.GetDensity() /
        PHD.Particle.GetDiam() * pow(GetVrm(cell), 3.0)
        * DaiUsVrm(x,y,i)) *
        pow( 0.63 + (4.8 * sqrt(GetVrm(cell)/GetRem(cell)) ), 2.0) *
        sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0)
          + pow(SVG.at(cell) - SVS.at(cell), 2.0) )) +
        ((( (3.0 * (1.0 - SEG.at(cell))
              * SEG.at(cell) * PHD.Gas.GetDensity() ) /
          (4.0 * PHD.Particle.GetDiam()) * pow(GetVrm(cell), 2.0))) *
        ( 2.0 * (0.63 + (4.8 * sqrt(GetVrm(cell)) /
          sqrt(GetRem(cell)) ))
          * ( (0.5 * DaiUsVrm(x,y,i) / sqrt(GetVrm(cell))) -
            (0.5 * pow(GetRem(cell), -3.0/2.0)
          * DaiUsRem(x,y,i)) ) ) *
        sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0)
          + pow(SVG.at(cell) - SVS.at(cell), 2.0) )) +
        (( (SUG.at(cell) - SUS.at(cell))
          * -1.0 * POD.PPhiuPOD.at(cell).at(i) /
        ( sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0)
            + pow(SVG.at(cell) - SVS.at(cell), 2.0)) )) *
        pow( 0.63 + (4.8 * sqrt(GetVrm(cell)/GetRem(cell)) ), 2.0) *
        ( (3.0 * (1.0 - SEG.at(cell)) * SEG.at(cell)
            * PHD.Gas.GetDensity() ) /
        (4.0 * PHD.Particle.GetDiam()) * pow(GetVrm(cell), 2.0)));

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
double NSSolver::DaiVsBeta(int x, int y, int i)

   {

   double val;
   int cell = (y * POD.GetNx()) + x;

   val =(( (-9.0/4.0) * (1.0 - SEG.at(cell)) * SEG.at(cell)
        * PHD.Gas.GetDensity() /
        PHD.Particle.GetDiam() * pow(GetVrm(cell), 3.0)
        * DaiVsVrm(x,y,i)  ) *
        pow( 0.63 + (4.8 * sqrt(GetVrm(cell)/GetRem(cell)) ), 2.0) *
        sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0)
          + pow(SVG.at(cell) - SVS.at(cell), 2.0) )) +
       ((( (3.0 * (1.0 - SEG.at(cell)) * SEG.at(cell)
             * PHD.Gas.GetDensity() ) /
         (4.0 * PHD.Particle.GetDiam()) * pow(GetVrm(cell), 2.0))) *
        ( 2.0 * (0.63 + (4.8 * sqrt(GetVrm(cell)) /
             sqrt(GetRem(cell)) ))
         * ( (0.5 * DaiVsVrm(x,y,i) / sqrt(GetVrm(cell))) -
            (0.5 * pow(GetRem(cell), -3.0/2.0)
               * DaiVsRem(x,y,i)) ) ) *
        sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0)
          + pow(SVG.at(cell) - SVS.at(cell), 2.0) )) +
       (( (SUG.at(cell) - SUS.at(cell)) * -1.0
              * POD.PPhivPOD.at(cell).at(i) /
        ( sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0)
           + pow(SVG.at(cell) - SVS.at(cell), 2.0)) )) *
        pow( 0.63 + (4.8 * sqrt(GetVrm(cell)/GetRem(cell)) ), 2.0) *
        ( (3.0 * (1.0 - SEG.at(cell)) * SEG.at(cell)
             * PHD.Gas.GetDensity() ) /
         (4.0 * PHD.Particle.GetDiam()) * pow(GetVrm(cell), 2.0)) );

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::I2DG(int x, int y)
```

```
    {

    double val;

    val = (2.0 * pow(DxUg(x,y), 2.0))
          - (2.0 * DxUg(x,y) * DyVg(x,y))
          + (2.0 * pow(DyVg(x,y), 2.0))
          + pow(DxVg(x,y), 2.0)  + pow(DyUg(x,y), 2.0)
           + (DxVg(x,y) * DyUg(x,y));

//    val = pow(DxUg(x,y), 2.0) - (2.0 * DxUg(x,y) * DyVg(x,y))
//           + pow(DyVg(x,y), 2.0) + pow(DxVg(x,y), 2.0)
//           + pow(DyUg(x,y), 2.0) + (DxVg(x,y) * DyUg(x,y));

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiEgMugt(int x, int y, int i)

    {

    double val;

    int cell = (POD.GetNx() * y) + x;

    val = 2.0 * pow(PHD.Gas.Getlscale(), 2.0)
            * POD.FgPhiPOD.at(cell).at(i)
            * PHD.Gas.GetDensity() * sqrt(I2DG(x,y));

    if ( isnan(val) )
       val = 0.0;

    return val;

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiEgBeta(int x, int y, int i)

   {

   double val;
   int cell = (POD.GetNx() * y) + x;

   val = ((3.0 * PHD.Gas.GetDensity() *
        sqrt(pow(SUG.at(cell) - SUS.at(cell),2.0) -
            pow(SVG.at(cell) - SVS.at(cell), 2.0)) /
        (4.0 * PHD.Particle.GetDiam() )) *
        (POD.FgPhiPOD.at(cell).at(i) *( 1.0 - (2.0 * SEG.at(cell))))
        * (pow(GetVrm(cell), -2.0))
        * (pow(0.63 + (4.8 * sqrt(GetVrm(cell)) / sqrt(GetRem(cell))),
             2.0)) )
        +
        ( (SEG.at(cell) - pow(SEG.at(cell), 2.0))
        * (-2.0 * pow(GetVrm(cell), -3.0) * DaiEgVrm(x,y,i))
        * (pow(0.63 + (4.8 * sqrt(GetVrm(cell)) / sqrt(GetRem(cell))),
             2.0)) )
        +
        ( (SEG.at(cell) - pow(SEG.at(cell), 2.0))
        * (pow(GetVrm(cell), -2.0))
        * (4.8 * (0.63 + (4.8 * sqrt(GetVrm(cell) /
          sqrt(GetRem(cell))))
        * pow(GetVrm(cell) * GetRem(cell), - 0.5) * DaiEgVrm(x,y,i)) ));

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiEgVrm(int x, int y, int i)

   {

   int cell = (POD.GetNx() * y) + x;
```

```
    double val;
    double A = pow(SEG.at(cell), 4.14);
    double B;
    double DA = pow(SEG.at(cell), 3.14) * 4.14;
    double DB;
    if (SEG.at(cell) > 0.85)
       {
       B = pow(SEG.at(cell), 2.65);
       DB = 2.65 * pow(SEG.at(cell), 1.65) * POD.FgPhiPOD.at(cell).at(i);
       }
    else
       {
       B = 0.8 * pow(SEG.at(cell), 1.28);
       DB = 0.8 * 1.28 * pow(SEG.at(cell), 0.28)
              * POD.FgPhiPOD.at(cell).at(i);
       }

    val = (0.5 * DA) + (0.25 * (pow (pow(0.06 * GetRem(cell), 2.0) +
          (0.24 * GetRem(cell) * B) - (0.12 * GetRem(cell) * A)
           + pow(A, 2.0)
          ,-0.5) * ((0.24 * GetRem(cell) * DB)
          - (0.12 * GetRem(cell) * DA)
          + (2.0 * A * DA)) ));

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiUgVrm(int x, int y, int i)

    {

    int cell = (y * POD.GetNx() ) + x;

    double val;
    double B;
    double A = pow(SEG.at(cell), 4.14);
```

```
    if (SEG.at(cell) > 0.85)
       B = pow(SEG.at(cell), 2.65);
    else
       B = 0.8 * pow(SEG.at(cell), 1.28);

    val = (0.03 * DaiUgRem(x,y,i)) + (0.25 *
          pow(pow(0.06 * GetRem(cell),2.0) + (0.24 * GetRem(cell) * B)
              - (0.12 * GetRem(cell) * A) + pow(A, 2.0),-0.5)
          * ((0.12 * GetRem(cell) * DaiUgRem(x,y,i))
            + (0.24 * B * DaiUgRem(x,y,i))
            - (0.12 * A * DaiUgRem(x,y,i))));

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiVgVrm(int x, int y, int i)

    {

    int cell = (y * POD.GetNx() ) + x;

    double val;
    double B;
    double A = pow(SEG.at(cell), 4.14);

    if (SEG.at(cell) > 0.85)
       B = pow(SEG.at(cell), 2.65);
    else
       B = 0.8 * pow(SEG.at(cell), 1.28);

    val = (0.03 * DaiVgRem(x,y,i)) + (0.25 *
          pow(pow(0.06 * GetRem(cell),2.0) + (0.24 * GetRem(cell) * B)
              - (0.12 * GetRem(cell) * A) + pow(A, 2.0),-0.5)
          * ((0.12 * GetRem(cell) * DaiVgRem(x,y,i))
            + (0.24 * B * DaiVgRem(x,y,i))
            - (0.12 * A * DaiVgRem(x,y,i))));

    if ( isnan(val) )
```

```
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiUsVrm(int x, int y, int i)

    {

    int cell = (y * POD.GetNx() ) + x;

    double val;
    double B;
    double A = pow(SEG.at(cell), 4.14);

    if (SEG.at(cell) > 0.85)
        B = pow(SEG.at(cell), 2.65);
    else
        B = 0.8 * pow(SEG.at(cell), 1.28);

    val = (0.03 * DaiUsRem(x,y,i)) + (0.25 *
          pow(pow(0.06 * GetRem(cell),2.0) + (0.24 * GetRem(cell) * B)
              - (0.12 * GetRem(cell) * A) + pow(A, 2.0),-0.5)
          * ((0.12 * GetRem(cell) * DaiUsRem(x,y,i))
            + (0.24 * B * DaiUsRem(x,y,i))
            - (0.12 * A * DaiUsRem(x,y,i))));

    if ( isnan(val) )
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiVsVrm(int x, int y, int i)

    {
```

```
    int cell = (y * POD.GetNx() ) + x;

    double val;
    double B;
    double A = pow(SEG.at(cell), 4.14);

    if (SEG.at(cell) > 0.85)
       B = pow(SEG.at(cell), 2.65);
    else
       B = 0.8 * pow(SEG.at(cell), 1.28);

    val = (0.03 * DaiVsRem(x,y,i)) + (0.25 *
          pow(pow(0.06 * GetRem(cell),2.0) + (0.24 * GetRem(cell) * B)
              - (0.12 * GetRem(cell) * A) + pow(A, 2.0),-0.5)
          * ((0.12 * GetRem(cell) * DaiVsRem(x,y,i))
            + (0.24 * B * DaiVsRem(x,y,i))
            - (0.12 * A * DaiVsRem(x,y,i))));

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiUgRem(int x, int y, int i)

    {

    double val;
    int cell = (POD.GetNx() * y) + x;

    val = pow( pow(SUG.at(cell) - SUS.at(cell), 2.0)
            + pow(SVG.at(cell) - SVS.at(cell), 2.0) , -0.5) *
            (SUG.at(cell) - SUS.at(cell)) * POD.GPhiuPOD.at(cell).at(i);

    if ( isnan(val) )
       val = 0.0;

    return val;

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiVgRem(int x, int y, int i)

   {

   double val;
   int cell = (POD.GetNx() * y) + x;

   val = pow( pow(SUG.at(cell) - SUS.at(cell), 2.0)
          + pow(SVG.at(cell) - SVS.at(cell), 2.0) , -0.5) *
          (SUG.at(cell) - SUS.at(cell)) * POD.GPhivPOD.at(cell).at(i);

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiUsRem(int x, int y, int i)

   {

   double val;
   int cell = (POD.GetNx() * y) + x;

   val = pow( pow(SUG.at(cell) - SUS.at(cell), 2.0)
          + pow(SVG.at(cell) - SVS.at(cell), 2.0) , -0.5) *
          (SUG.at(cell) - SUS.at(cell))
          * (-1.0 * POD.PPhiuPOD.at(cell).at(i) );

   if ( isnan(val) )
      val = 0.0;

   return val;

   }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiVsRem(int x, int y, int i)

   {

   double val;
   int cell = (POD.GetNx() * y) + x;

   val = pow( pow(SUG.at(cell) - SUS.at(cell), 2.0)
          + pow(SVG.at(cell) - SVS.at(cell), 2.0) , -0.5) *
           (SUG.at(cell) - SUS.at(cell))
          * (-1.0 * POD.PPhivPOD.at(cell).at(i) );

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiEgDxMub(int x, int y, int i)

   {

   double val;
   int cell = (POD.GetNx() * y) + x;

   val = (-256.0 / (48.0 * M_PI)) * PHD.Gas.GetDensity()
        * PHD.Particle.GetDiam() * GetGom(cell)
        * ( (sqrt(M_PI * DxTs(x,y)) * (1.0 - SEG.at(cell))
                  * (-POD.FgPhiPOD.at(cell).at(i)) )
        + (sqrt(M_PI * STS.at(cell)) * ( -DxEg(x,y) )
          * (-POD.FgPhiPOD.at(cell).at(i)) )
        + (sqrt(M_PI * STS.at(cell)) * (1.0 - SEG.at(cell))
                              * DxEgi(x,y,i)));

   if ( isnan(val) )
      val = 0.0;

   return val;
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiTsDxMub(int x, int y, int i)

    {

    double val;
    int cell = (POD.GetNx() * y) + x;

    val = (-128.0 / (96.0 * M_PI)) * M_PI * PHD.Gas.GetDensity()
          * PHD.Particle.GetDiam() * GetGom(cell)
          * ( (2.0 * (1.0 - SEG.at(cell)) * (-DxEg(x,y)) *
             (1.0 / sqrt(M_PI * STS.at(cell)))
               * POD.TsPhiPOD.at(cell).at(i))
           +  (pow(1.0 - SEG.at(cell), 2.0)
            * (-0.5 * pow(M_PI * STS.at(cell), -1.5) *
              DxTs(x,y)) * POD.TsPhiPOD.at(cell).at(i))
           +  (pow(1.0 - SEG.at(cell), 2.0)
              * (1.0 / sqrt(M_PI * STS.at(cell)))
              * DxTsi(x,y,i)) );

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiEgDyMub(int x, int y, int i)

    {

    double val;
    int cell = (POD.GetNx() * y) + x;

    val = (-256.0 / (48.0 * M_PI)) * PHD.Gas.GetDensity()
          * PHD.Particle.GetDiam() * GetGom(cell)
          * ( (sqrt(M_PI * DyTs(x,y)) * (1.0 - SEG.at(cell))
```

```
                    * (-POD.FgPhiPOD.at(cell).at(i)) )
          + (sqrt(M_PI * STS.at(cell)) * ( -DyEg(x,y) )
             * (-POD.FgPhiPOD.at(cell).at(i)) )
          + (sqrt(M_PI * STS.at(cell)) * (1.0 - SEG.at(cell))
             * DyEgi(x,y,i)));

    if ( isnan(val) )
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiTsDyMub(int x, int y, int i)

    {

    double val;
    int cell = (POD.GetNx() * y) + x;

    val = (-128.0 / (96.0 * M_PI)) * M_PI * PHD.Gas.GetDensity()
          * PHD.Particle.GetDiam() * GetGom(cell)
          * ( (2.0 * (1.0 - SEG.at(cell)) * (-DyEg(x,y)) *
              (1.0 / sqrt(M_PI * STS.at(cell)))
             * POD.TsPhiPOD.at(cell).at(i))
           +  (pow(1.0 - SEG.at(cell), 2.0)
             * (-0.5 * pow(M_PI * STS.at(cell), -1.5) *
               DyTs(x,y)) * POD.TsPhiPOD.at(cell).at(i))
           +  (pow(1.0 - SEG.at(cell), 2.0)
             * (1.0 / sqrt(M_PI * STS.at(cell)))
               * DyTsi(x,y,i)) );

    if ( isnan(val) )
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
double NSSolver::DaiEgMub(int x, int y, int i)

   {

   double val;
   int cell = (POD.GetNx() * y) + x;

   val = (-256.0 / (48.0 * M_PI)) * PHD.Gas.GetDensity()
         * PHD.Particle.GetDiam() * GetGom(cell)
         *  (sqrt(M_PI * STS.at(cell)) * (1.0 - SEG.at(cell))
                   * (-POD.FgPhiPOD.at(cell).at(i)) );

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiEgMums(int x, int y, int i)

   {

   double val;
   int cell = (POD.GetNx() * y) + x;

   val = (
         ((pow(PHD.Particle.GetDensity(), 2.0) * GetGom(cell)
          * pow(POD.TsPhiPOD.at(cell).at(i), 1.5) * (-5.0 / 96.0)
          * PHD.Particle.GetDiam() * sqrt(M_PI)
          * POD.FgPhiPOD.at(cell).at(i))
         * ( ((PHD.Particle.GetDensity() * (1.0 - SEG.at(cell))
         * STS.at(cell)) +
        ((2.0 * GetBeta(x,y) * (5.0/96.0) * PHD.Particle.GetDiam()
          * sqrt(M_PI) * sqrt(STS.at(cell))) / (1.0 - SEG.at(cell)))
            ))) - (
          ((-PHD.Particle.GetDensity() * STS.at(cell)
            * POD.FgPhiPOD.at(cell).at(i)) +
         (2.0 * GetBeta(x,y) * (5.0/96.0) * PHD.Particle.GetDiam()
          * sqrt(M_PI) * sqrt(STS.at(cell))
            * (pow(1.0 - SEG.at(cell), -2.0) *
            POD.FgPhiPOD.at(cell).at(i) )))
```

```
         * (( pow(PHD.Particle.GetDensity(), 2.0) * GetGom(cell)
         * (1.0 - SEG.at(cell)) * (5.0 / 96.0)
         * PHD.Particle.GetDiam() * sqrt(M_PI)
         * pow(STS.at(cell), 1.5))
        ))
        )
        /
        ((PHD.Particle.GetDensity() * (1.0 - SEG.at(cell))
         * STS.at(cell)) +
        ((2.0 * GetBeta(x,y) * (5.0/96.0) * PHD.Particle.GetDiam()
          * sqrt(M_PI) * sqrt(STS.at(cell))) / (1.0 - SEG.at(cell))));

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiEgMup(int x, int y, int i)

   {

   double val;
   double nu = (1.0 + CoR ) / 2.0;
   int cell = x + (POD.GetNx() * y);

   val = (DaiEgMums(x,y,i) * (1.2 / (GetGom(cell) * nu * (2.0 - nu)))
        * (1.0 + ( (8.0/5.0) * nu * (1.0 - SEG.at(cell))
                * GetGom(cell)) )
        * (1.0 + ( (8.0/5.0) * nu * (1.0 - SEG.at(cell))
                * GetGom(cell)) * ((3.0 * nu) - 2.0 ) ))
        + (1.2 * (GetMums(x,y) / (GetGom(cell) * nu * (2.0 - nu)) )
              * (1.0 + ( (8.0/5.0) * nu * (1.0 - SEG.at(cell)
                * GetGom(cell)) * ((3.0 * nu) - 2.0 ) ))
          * ((-8.0/5.0) * nu * GetGom(cell)
          * POD.FgPhiPOD.at(cell).at(i) ))
        + (1.2 * (GetMums(x,y) / (GetGom(cell) * nu * (2.0 - nu)) )
        * (1.0 + ( (8.0/5.0) * nu * (1.0 - SEG.at(cell))
                * GetGom(cell)) ) *
          ((-1.6 * nu * ((3.0 * nu) - 2.0) * GetGom(cell)
           * POD.FgPhiPOD.at(cell).at(i))))
```

```
      + (0.72 * nu * DaiEgMub(x,y,i));

  if ( isnan(val) )
    val = 0.0;

  return val;

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiTsMub(int x, int y, int i)

  {

  double val;
  int cell = x + (POD.GetNx() * y);

  val = (-128.0 / (96.0 * M_PI)) * M_PI * PHD.Gas.GetDensity()
       * PHD.Particle.GetDiam() * GetGom(cell)
       * (pow(1.0 - SEG.at(cell), 2.0) *
         (1.0 / sqrt(M_PI * STS.at(cell)))
       * POD.TsPhiPOD.at(cell).at(i));

  if ( isnan(val) )
    val = 0.0;

  return val;

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiTsMums(int x, int y, int i)

  {

  double val;
  int cell = x + (POD.GetNx() * y);

  val = ((
      (pow(PHD.Particle.GetDensity(), 2.0) * (1.0 - SEG.at(cell))
       * GetGom(cell) * (5.0/ 96.0) * PHD.Particle.GetDiam()
```

```
         * sqrt(M_PI) * POD.TsPhiPOD.at(cell).at(i)
         * (1.5 / sqrt(STS.at(cell))))
         *((PHD.Particle.GetDensity() * (1.0 - SEG.at(cell))
         * STS.at(cell)) +
        ((2.0 * GetBeta(x,y) * (5.0/96.0) * PHD.Particle.GetDiam()
          * sqrt(M_PI) * sqrt(STS.at(cell))) / (1.0 - SEG.at(cell))))
         ) -
        ( ( (PHD.Particle.GetDensity() * (1.0 - SEG.at(cell))
             * POD.TsPhiPOD.at(cell).at(i)) +
        ( ((5.0/48.0) * GetBeta(x,y) * PHD.Particle.GetDiam()
            * sqrt(M_PI) * POD.TsPhiPOD.at(cell).at(i))
        / ((1.0 - SEG.at(cell)) * 2.0
            * sqrt(POD.TsPhiPOD.at(cell).at(i)) ) ) )
        * (( pow(PHD.Particle.GetDensity(), 2.0) * GetGom(cell)
          * (1.0 - SEG.at(cell)) * (5.0 / 96.0)
          * PHD.Particle.GetDiam() * sqrt(M_PI)
          * pow(STS.at(cell), 1.5))
          ))
         )
         /
        ((PHD.Particle.GetDensity() * (1.0 - SEG.at(cell))
         * STS.at(cell)) +
        ((2.0 * GetBeta(x,y) * (5.0/96.0) * PHD.Particle.GetDiam()
          * sqrt(M_PI) * sqrt(STS.at(cell))) / (1.0 - SEG.at(cell))) );

    if ( isnan(val) )
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiTsMup(int x, int y, int i)

    {

    double val;
    int cell = x + (POD.GetNx() * y);
    double nu = (1.0 + CoR ) / 2.0;

    val = (1.2 * (1.0 / (GetGom(cell) * nu * (2.0 - nu)) )
          * DaiTsMums(x,y,i)
```

```
        * (1.0 + ( (8.0/5.0) * nu * (1.0 - SEG.at(cell))
                * GetGom(cell)) )
        * (1.0 + ( (8.0/5.0) * nu * (1.0 - SEG.at(cell) )
                * GetGom(cell)) * ((3.0 * nu) - 2.0 ) ))
        + (0.72 * nu * GetMub(x,y));

    if ( isnan(val) )
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DaiTsBeta(int x, int y, int i)

    {

    double val;

    val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetMugt(int x, int y)

    {

    int cell = x + (POD.GetNx() * y);
    double val;

    val = 2.0 * pow(PHD.Gas.Getlscale(), 2.0) * SEG.at(cell) *
        PHD.Gas.GetDensity() * sqrt(I2DG(x,y));

    if ( isnan(val) )
        val = 0.0;

    return val;
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetBeta(int x, int y)

    {

    int cell = x + (POD.GetNx() * y);
    double val;

    val = ( (3.0 * (1.0 - SEG.at(cell)) * SEG.at(cell)
                * PHD.Gas.GetDensity() ) /
            (4.0 * PHD.Particle.GetDiam()) * pow(GetVrm(cell), 2.0)) *
          pow( 0.63 + (4.8 * sqrt(GetVrm(cell)/GetRem(cell)) ), 2.0) *
          sqrt(pow(SUG.at(cell) - SUS.at(cell), 2.0)
             + pow(SVG.at(cell) - SVS.at(cell), 2.0) );

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetMub(int x, int y)

    {

    int cell = x + (POD.GetNx() * y);
    double val;

    val = (256.0 / (96.0 * M_PI)) * PHD.Particle.GetDensity()
          * PHD.Particle.GetDiam() * GetGom(cell)
          * sqrt(M_PI * STS.at(cell))
          * pow(1.0 - SEG.at(cell), 2.0);

    if ( isnan(val) )
       val = 0.0;
```

```
    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetMums(int x, int y)

    {

    int cell = x + (POD.GetNx() * y);
    double val;

    val = ( pow(PHD.Particle.GetDensity(), 2.0) * GetGom(cell)
           * (1.0 - SEG.at(cell)) * (5.0 / 96.0)
           * PHD.Particle.GetDiam() * sqrt(M_PI)
           * pow(STS.at(cell), 1.5))
          /
         ((PHD.Particle.GetDensity() * (1.0 - SEG.at(cell))
          * STS.at(cell)) +
         ((2.0 * GetBeta(x,y) * (5.0/96.0) * PHD.Particle.GetDiam()
           * sqrt(M_PI) * sqrt(STS.at(cell))) / (1.0 - SEG.at(cell))));

    if ( isnan(val) )
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetMup(int x, int y)

    {

    int cell = x + (POD.GetNx() * y);
    double val;
    double nu = (1.0 + CoR ) / 2.0;

    val = (1.2 * (GetMums(x,y) / (GetGom(cell) * nu * (2.0 - nu)) )
         * (1.0 + ( (8.0/5.0) * nu * (1.0 - SEG.at(cell))
                  * GetGom(cell)) )
```

```
        * (1.0 + ( (8.0/5.0) * nu * (1.0 - SEG.at(cell))
                * GetGom(cell)) * ((3.0 * nu) - 2.0 ) ))
        + (0.72 * nu * GetMub(x,y));

    if ( isnan(val) )
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxMub(int x, int y)

    {

    double val;
    int lc;
    int rc;

    if (x == 0)
        lc = -1;
    else
        lc = (x-1) + ( POD.GetNx() * y);

    if (x == (POD.GetNx() - 1))
        rc = -1;
    else
        rc = (x+1) + ( POD.GetNx() * y);

    if (lc == -1)
        val = (GetMub(x+1,y) - GetMub(x,y)) / PHD.Getdelx();
    else if (rc == -1)
        val = (GetMub(x,y) - GetMub(x-1,y)) / PHD.Getdelx();
    else
        val = (GetMub(x+1,y) - GetMub(x-1,y)) / PHD.Getdelx();

    if ( isnan(val) )
        val = 0.0;

    return val;

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyMub(int x, int y)

   {

   double val;
   int lc;
   int uc;

   if (y == 0)
      lc = -1;
   else
      lc = (x) + ( POD.GetNx() * (y-1));

   if (y == (POD.GetNy() - 1))
      uc = -1;
   else
      uc = (x) + ( POD.GetNx() * (y+1));

   if (lc == -1)
      val = (GetMub(x,y+1) - GetMub(x,y)) / PHD.Getdelx();
   else if (uc == -1)
      val = (GetMub(x,y) - GetMub(x,y-1)) / PHD.Getdelx();
   else
      val = (GetMub(x,y+1) - GetMub(x,y-1)) / PHD.Getdelx();

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxxUs(int x, int y)

   {

   double val;
   int cell = x + (POD.GetNx() * y);
```

```
int lc;
int rc;

int lc2 = 0;
int rc2 = 0;

if (x == 0)
   lc = -1;
else
   lc = (x-1) + ( POD.GetNx() * y);

if (x == (POD.GetNx() - 1))
   rc = -1;
else
   rc = (x+1) + ( POD.GetNx() * y);

if (lc == -1)
   lc2 = (x-2) + ( POD.GetNx() * y);

if (rc == -1)
   rc2 = (x+2) + ( POD.GetNx() * y);

if (lc == -1)
   // on the left wall...
   val = (UsSum.back().at(rc2)
         - (2.0 * UsSum.back().at(rc))
         + UsSum.back().at(cell)) /
         pow( PHD.Getdelx() , 2.0);
else if (rc == -1)
   // on the right wall ...
   val = (UsSum.back().at(lc2)
         - (2.0 * UsSum.back().at(lc))
         + UsSum.back().at(cell))
         / pow( PHD.Getdelx() , 2.0);
else
   val = (UsSum.back().at(rc)
         - (2.0 * UsSum.back().at(cell))
         + UsSum.back().at(lc)) /
         (pow( PHD.Getdelx() , 2.0) );

if ( isnan(val) )
   val = 0.0;

return val;
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxxUsi(int x, int y, int i)

    {

    double val;
    int cell = x + (POD.GetNx() * y);
    int lc;
    int rc;

    int lc2 = 0;
    int rc2 = 0;

    if (x == 0)
        lc = -1;
    else
        lc = (x-1) + ( POD.GetNx() * y);

    if (x == (POD.GetNx() - 1))
        rc = -1;
    else
        rc = (x+1) + ( POD.GetNx() * y);

    if (lc == -1)
        lc2 = (x-2) + ( POD.GetNx() * y);

    if (rc == -1)
        rc2 = (x+2) + ( POD.GetNx() * y);

    if (lc == -1)
        // on the left wall...
        val = (POD.PPhiuPOD.at(rc2).at(i)
                - (2.0 * POD.PPhiuPOD.at(rc).at(i))
                + POD.PPhiuPOD.at(cell).at(i)) /
                pow( PHD.Getdelx() , 2.0);
    else if (rc == -1)
        // on the right wall ...
        val = (POD.PPhiuPOD.at(lc2).at(i)
                - (2.0 * POD.PPhiuPOD.at(lc).at(i))
                + POD.PPhiuPOD.at(cell).at(i))
                / pow( PHD.Getdelx() , 2.0);
```

```
else
   val = (POD.PPhiuPOD.at(rc).at(i)
        - (2.0 * POD.PPhiuPOD.at(cell).at(i))
        + POD.PPhiuPOD.at(lc).at(i))
        / pow( PHD.Getdelx() , 2.0);

return val;

}

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DxxVsi(int x, int y, int i)

{

double val;
int cell = x + (POD.GetNx() * y);
int lc;
int rc;

int lc2 = 0;
int rc2 = 0;

if (x == 0)
   lc = -1;
else
   lc = (x-1) + ( POD.GetNx() * y);

if (x == (POD.GetNx() - 1))
   rc = -1;
else
   rc = (x+1) + ( POD.GetNx() * y);

if (lc == -1)
   lc2 = (x-2) + ( POD.GetNx() * y);

if (rc == -1)
   rc2 = (x+2) + ( POD.GetNx() * y);

if (lc == -1)
   // on the left wall...
   val = (POD.PPhivPOD.at(rc2).at(i)
        - (2.0 * POD.PPhivPOD.at(rc).at(i))
```

```
            + POD.PPhivPOD.at(cell).at(i)) /
            pow( PHD.Getdelx() , 2.0);
    else if (rc == -1)
        // on the right wall ...
        val = (POD.PPhivPOD.at(lc2).at(i)
            - (2.0 * POD.PPhivPOD.at(lc).at(i))
            + POD.PPhivPOD.at(cell).at(i))
            / pow( PHD.Getdelx() , 2.0);
    else
        val = (POD.PPhivPOD.at(rc).at(i)
            - (2.0 * POD.PPhivPOD.at(cell).at(i))
            + POD.PPhivPOD.at(lc).at(i))
            / pow( PHD.Getdelx() , 2.0);

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::DyyVs(int x, int y)

    {

    double val;
    int cell = (x * POD.GetNy() ) + y;
    int lc;
    int uc;

    int lc2 = 0;
    int uc2 = 0;

    if (y == 0)
        lc = -1;
    else
        lc = (x) + ( POD.GetNx() * (y-1));

    if (y == (POD.GetNy() - 1))
        uc = -1;
    else
        uc = (x) + ( POD.GetNx() * (y+1));

    if (lc == -1)
        lc2 = (x) + ( POD.GetNx() * (y-2));
```

```
    if (uc == -1)
        uc2 = (x) + ( POD.GetNx() * (y+2));

    if (lc == -1)
        // on the left wall...
        val = (VsSum.back().at(uc2)
              - (2.0 * VsSum.back().at(uc))
              + VsSum.back().at(cell)) /
              pow( PHD.Getdely() , 2.0);
    else if (uc == -1)
        // on the right wall ...
        val = (VsSum.back().at(lc2)
              - (2.0 * VsSum.back().at(lc))
              + VsSum.back().at(cell))
              / pow( PHD.Getdely() , 2.0);
    else
        val = (VsSum.back().at(uc)
              - (2.0 * VsSum.back().at(cell))
              + VsSum.back().at(lc)) /
              (pow( PHD.Getdely() , 2.0) );


    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::EvalUg(int i, std::vector< double > & V)

    {// The Ug evaluation occurs for some specific cell
     // this allows precalculation of key "sums" prior to solving.  :)

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = i - (y * POD.GetNx());

        // T1P1
    val = (PHD.Gas.GetDensity() * SUG.at(i) * DtEg(x, y)
        * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +
```

```
 // T1P2
(SEG.at(i) * PHD.Gas.GetDensity() * DtUg(x, y)
* POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // T2P1
(DxEg(x, y) * PHD.Gas.GetDensity() * pow(SUG.at(i),2)
* POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // T2P2
(DxUg(x, y) * SUG.at(i) * SEG.at(i) * PHD.Gas.GetDensity()
 * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // T2P3
(SUG.at(i) * DyEg(x, y) * PHD.Gas.GetDensity() * SVG.at(i)
 * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // T2P4
(SEG.at(i) * PHD.Gas.GetDensity() * SVG.at(i) * DyVg(x, y)
 * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // R1P1                          //R2P2
(SEG.at(i) * DxPg(x, y)* POD.GPhiuPOD.at(i).at(0)
 * POD.Getdx() * POD.Getdy()) - 0.0 +

 // R3P1
(GetMugt(x,y) * DyUg(x, y) * POD.GPhiuPOD.at(i).at(0)
 * POD.Getdx()) +

 // R3P2
(GetMugt(x,y) * DyUg(x, y) * DyUgi(x,y,0) * POD.Getdx()
 * POD.Getdy()) +

 // R3P3
((-4.0/3.0) * GetMugt(x,y) * DxUg(x, y)
 * POD.GPhiuPOD.at(i).at(0) * POD.Getdy()) +

 // R3P4
((4.0/3.0) * GetMugt(x,y) * DxUg(x, y) * DxUgi(x,y,0)
 * POD.Getdx() * POD.Getdy()) +

 // R3P5
(-GetMugt(x,y) * DxUg(x, y) * POD.GPhiuPOD.at(i).at(0)
 * POD.Getdx()) +
```

```
        // R3P6
       (GetMugt(x,y) * DxUg(x, y) * DyUgi(x,y,0) * POD.Getdx()
        * POD.Getdy()) +

        // R4
       (GetBeta(x,y) * (SUS.at(i) - SUG.at(i))
        * POD.GPhiuPOD.at(i).at(0)
        * POD.Getdx() * POD.Getdy());

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::EvalVg(int i, std::vector< double > & V)

   {

   double val;

   int x, y;

   y = (int) floor(i / POD.GetNx());
   x = (int) i - (y * POD.GetNx());

        // T1P1
   val = (DtEg(x, y) * PHD.Gas.GetDensity() * SUG.at(i)
        * POD.GPhivPOD.at(i).at(0)
        * POD.Getdx() * POD.Getdy()) +

        // T1P2
       (SEG.at(i) * PHD.Gas.GetDensity() * DtVg(x, y)
        * POD.GPhivPOD.at(i).at(0)
        * POD.Getdx() * POD.Getdy()) +

        // T2P1
       (DxEg(x, y) * PHD.Gas.GetDensity() * SUG.at(i) * SVG.at(i)
        * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P2
```

```
        (DyEg(x, y) * PHD.Gas.GetDensity() * pow(SVG.at(i),2)
         * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // T2P3
        (SEG.at(i) * PHD.Gas.GetDensity() * DxVg(x, y)
         * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // R1
        (-SEG.at(i) * DyPg(x,y) * POD.GPhivPOD.at(i).at(0)
         * POD.Getdx() * POD.Getdy()) +

         // R2
        (SEG.at(i) * PHD.Gas.GetDensity() * 980.665
         * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // R3P1
        (-GetMugt(x,y) * DxVg(x,y) * POD.GPhivPOD.at(i).at(0)
         * POD.Getdx() * POD.Getdy()) +

         // R3P2
        (GetMugt(x,y) * DxVg(x,y) * DyVgi(x,y,0) * POD.Getdx()
         * POD.Getdy()) +

         // R3P3
        (-GetMugt(x,y) * SVG.at(i) * DxVg(x,y) * POD.Getdy()) +

         // R3P4
        (GetMugt(x,y) * DxVg(x,y) * DxVgi(x,y,0) * POD.Getdx()
         * POD.Getdy()) +

         // R3P5
        ((-4.0 / 3.0) * GetMugt(x,y) * SVG.at(i) * DyVgi(x,y,0)
         * POD.Getdx() * POD.Getdy()) +

         // R3P6
        ((4.0 / 3.0) * GetMugt(x,y) * DyVg(x,y) * DyVgi(x,y,0)
         * POD.Getdx() * POD.Getdy()) +

         // R4
        (GetBeta(x, y) * (SVS.at(i) - SVG.at(i))
         * POD.GPhivPOD.at(i).at(0)
         * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
       val = 0.0;
```

```
    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::EvalUs(int i, std::vector< double > & V)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

        // T1P1
    val = (-DtEg(x, y) * PHD.Particle.GetDensity() * SUS.at(i)
        * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T1P2
        ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * DtUs(x,y)
        * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P1
        (-DxEg(x,y) * PHD.Particle.GetDensity() * pow(SUS.at(i),2)
        * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P2
        ((1.0-SEG.at(i)) * PHD.Particle.GetDensity() * DxUs(x,y)
        * SUS.at(i)
        * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P3
        (-DyEg(x,y) * PHD.Particle.GetDensity() * SUS.at(i) * SVS.at(i)
        * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P4
        ((1.0 -SEG.at(i)) * PHD.Particle.GetDensity() * SVS.at(i)
        * DyUs(x,y)
        * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +
```

```
        // R1 - R2
        ((1.0 - SEG.at(i)) * DxPg(x,y) * POD.PPhiuPOD.at(i).at(0)
         * POD.Getdx()
         * POD.Getdy()) - 0.0 +

         // R3P1
        (DxPs(x,y) * POD.PPhiuPOD.at(i).at(0) * POD.Getdx()
         * POD.Getdy()) +

         // R3P2
        (-0.5 * (1.0 + CoR) * DxMub(x,y) * DxUs(x,y)
         * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // R3P3
        (-0.5 * (1.0 + CoR) * GetMub(x,y) * DxxUs(x,y)
         * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // R3P4
        (-GetMup(x,y) * SUS.at(i) * DyUsi(x,y,0) * POD.Getdx()) +

         // R3P5
        (GetMup(x,y) * DyUs(x,y) * DyUsi(x,y,0) * POD.Getdx()
         * POD.Getdy()) +

         // R3P6
        ((-4.0/3.0) * GetMup(x,y) * SUS.at(i) * DxUsi(x,y,0)
         * POD.Getdy()) +

         // R3P7
        ((4.0/3.0) * GetMup(x,y) * DxUs(x,y) * DxUsi(x,y,0)
         * POD.Getdx() * POD.Getdy()) +

         // R3P8
        (-GetMup(x,y) * DxUs(x,y) * DxUsi(x,y,0) * POD.Getdx()) +

         // R3P9
        (GetMup(x,y) * DxVs(x,y) * DxUsi(x,y,0) * POD.Getdx()
         * POD.Getdy()) +

         // R4
        (GetBeta(x,y) * (SUG.at(i) - SUS.at(i))
         * POD.PPhiuPOD.at(i).at(0)
         * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
```

```
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::EvalVs(int i, std::vector< double > & V)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

         // T1P1
    val = ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * SVS.at(i)
         * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // T1P2
        ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * DtVs(x,y)
         * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // T2P1
        (-DxEg(x,y) * PHD.Particle.GetDensity() * SVS.at(i) * SUS.at(i)
         * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // T2P2
        (-DyEg(x,y) * PHD.Particle.GetDensity() * pow(SVS.at(i),2)
         * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // T2P3
        ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * DyVs(x,y)
         * SVS.at(i)
         * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // T2P4
        ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * SUS.at(i)
         * DxVs(x,y)
         * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +
```

```
 // R1
((1.0 - SEG.at(i)) * DyPg(x,y) * POD.PPhivPOD.at(i).at(0)
 * POD.Getdx() * POD.Getdy()) +

 // R2
(980.665 * (1.0 - SEG.at(i)) * PHD.Particle.GetDensity()
 * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // R3P1
(DyPs(x,y) * POD.PPhivPOD.at(i).at(0) * POD.Getdx()
 * POD.Getdy()) +

 // R3P2
(-0.5 * (1.0 + CoR) * DyMub(x,y) * DyVs(x,y)
 * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // R3P3
(-0.5 * (1.0 + CoR) * GetMub(x,y) * DyyVs(x,y)
 * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // R3P4
(-GetMup(x,y) * DxVs(x,y) * POD.PPhivPOD.at(i).at(0)
 * POD.Getdx()) +

 // R3P5
(GetMup(x,y) * DxVs(x,y) * DyVsi(x,y,0) * POD.Getdx()
 * POD.Getdy()) +

 // R3P6
(-GetMup(x,y) * SVS.at(i) * DxVsi(x,y,0) * POD.Getdy()) +

 // R3P7
(GetMup(x,y) * DxVs(x,y) * DxVsi(x,y,0) * POD.Getdx()
 * POD.Getdy()) +

 // R3P8
((-4.0/3.0) * GetMup(x,y) * SVS.at(i) * DyVsi(x,y,0)
 * POD.Getdx()) +

 // R3P9
((4.0/3.0) * GetMup(x,y) * DyVs(x,y) * DyVsi(x,y,0)
 * POD.Getdx() * POD.Getdy()) +

 // R4
```

```
        (GetBeta(x,y) * (SVG.at(i) - SVS.at(i))
         * POD.PPhivPOD.at(i).at(0)
         * POD.Getdx() * POD.Getdy());

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUg4Ug(std::vector< double > & V, int i, int j)

   {// NOTE: i = equation index which runs over cells
    //       j = coefficient in question

   double val;

   int x, y;

   y = (int) floor(i / POD.GetNx());
   x = (int) i - (y * POD.GetNx());

//   std::cout << "In GetPDUg4Ug \n";

//   std::cout << DxEg(x,y) << " is DxEg \n";
//   std::cout << DaiUgMugt(x,y,j) << " is DaiUgMugt for j \n";
//   std::cout << SUG.at(i) << " is SUG \n";

         // T1P1 + T1P2
   val = (PHD.Gas.GetDensity() * DtEg(x,y) * POD.GPhiuPOD.at(i).at(j)
         * POD.GPhiuPOD.at(i).at(0) * POD.Getdx()
         * POD.Getdy()) + 0.0 +

         // T2P1
         (2.0 * SUG.at(i) * POD.GPhiuPOD.at(i).at(j) * DxEg(x,y)
         * PHD.Gas.GetDensity() * POD.GPhiuPOD.at(i).at(0)
         * POD.Getdx() * POD.Getdy()) +
//std::cout << "Made it this far a... \n";

         // T2P2
         (((DxUgi(x,y,j) * SUG.at(i) * SEG.at(i)) + (DxUg(x,y)
```

```
        * SEG.at(i)
        * POD.GPhiuPOD.at(i).at(j) )) * POD.GPhiuPOD.at(i).at(0)
        * PHD.Gas.GetDensity() * POD.Getdx() * POD.Getdy()) +
// std::cout << "Made it this far b... \n";

        // T2P3
       (pow(POD.GPhiuPOD.at(i).at(0),2) * DyEg(x,y) * SVG.at(i)
        * PHD.Gas.GetDensity() * POD.Getdx() * POD.Getdy())  +
//std::cout << "Made it this far c... \n";

        // T2P4 + R1 + R2
       (SEG.at(i) * SVG.at(i) * PHD.Gas.GetDensity() * DyUgi(x,y,j)
        * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy())
        + 0.0 + 0.0 +
//std::cout << "Made it this far 1... \n";

        // R3P1
       (((-DaiUgMugt(x,y,j) * DyUg(x,y)) + (-GetMugt(x,y)
        * DyUgi(x,y,j))) * POD.GPhiuPOD.at(i).at(0) * POD.Getdx()) +

//std::cout << "Made it this far 2... \n";

        // R3P2
       (((DaiUgMugt(x,y,j) * DyUg(x,y)) + (GetMugt(x,y) * DyUgi(x,y,j)))
        * DyUgi(x,y,0) * POD.Getdx() * POD.Getdy()) +

//std::cout << "Made it this far 3... \n";

        // R3P4
       (((DaiUgMugt(x,y,j) * DxUg(x,y)) + (GetMugt(x,y) * DxUgi(x,y,j)))
        * (4.0/3.0) * DxUgi(x,y,0) * POD.Getdx() * POD.Getdy()) +

//std::cout << "Made it this far 4... \n";

        // R3P5
       (((-DaiUgMugt(x,y,j) * DxUg(x,y)) - (GetMugt(x,y)
        * DxUgi(x,y,j))) * POD.GPhiuPOD.at(i).at(0) * POD.Getdx()) +

        // R3P6
       (DaiUgMugt(x,y,j) * DxVg(x,y) * DyVgi(x,y,0) * POD.Getdx()
        * POD.Getdy()) +

        // R4
       ( (    ( DaiUgBeta(x,y,j) * (SUS.at(i) - SUG.at(i)) )
           - (GetBeta(x,y) * POD.GPhiuPOD.at(i).at(j) )     )
```

```
        * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy() );

//    std::cout << "Leaving GetPDUg4Ug \n";

    if ( isnan(val) )
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVg4Ug(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

        // T1P1 + T1P2
    val = (0.0) +

        // T2P1
        (DxEg(x,y) * PHD.Gas.GetDensity() * POD.GPhiuPOD.at(i).at(j)
        * SVG.at(i) * POD.GPhivPOD.at(i).at(0) * POD.Getdx()
        * POD.Getdy()) +

        // T2P2
        (0.0) +

        // T2P3
        (DxVg(x,y) * PHD.Gas.GetDensity() * POD.GPhiuPOD.at(i).at(j)
        * SEG.at(i) * POD.GPhivPOD.at(i).at(0) * POD.Getdx()
        * POD.Getdy()) +

        // R1 + R2
        (0.0) +

        // R3P1
```

```
        (-DaiUgMugt(x,y,j) * DxVg(x,y) * POD.GPhivPOD.at(i).at(0)
         * POD.Getdx()) +

         // R3P2
        (-DaiUgMugt(x,y,j) * DxVg(x,y) * DyVgi(x,y,0) * POD.Getdx()
         * POD.Getdy()) +

         // R3P3
        (-DaiUgMugt(x,y,j) * SVG.at(i) * DxVgi(x,y,0) * POD.Getdy()) +

         // R3P4
        (DaiUgMugt(x,y,j) * DxVg(x,y) * DxVgi(x,y,0) * POD.Getdx()
         * POD.Getdy()) +

         // R3P5
        ((-4.0/3.0) * DaiUgMugt(x,y,j) * SVG.at(i) * DyVgi(x,y,0)
         * POD.Getdx()) +

         // R3P6
        ((4.0/3.0) * DaiUgMugt(x,y,j) * DyVg(x,y) * DyVgi(x,y,0)
         * POD.Getdx() * POD.Getdy()) +

         // R4
        (DaiUgBeta(x,y,j) * (SVS.at(i) - SVG.at(i))
         * POD.PPhivPOD.at(i).at(0)
         * POD.Getdx() * POD.Getdy());

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUs4Ug(std::vector< double > & V, int i, int j)

   {

   double val;

   int x, y;
```

```
   y = (int) floor(i / POD.GetNx());
   x = (int) i - (y * POD.GetNx());

   // NOTE:  all terms are 0 except RT4

   val = (((DaiUgBeta(x,y,j) * (SUG.at(i) - SUS.at(i))) + (GetBeta(x,y)
        * POD.GPhiuPOD.at(i).at(j)))
        * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVs4Ug(std::vector< double > & V, int i, int j)

   {

   double val;

   int x, y;

   y = (int) floor(i / POD.GetNx());
   x = (int) i - (y * POD.GetNx());

   // NOTE:  all terms are 0 except RT4

   val = (DaiUgBeta(x,y,j) * (SVG.at(i) - SVS.at(i))
        * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
double NSSolver::GetPDUg4Vg(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

          //(T1P1 -> T2P2)
    val = (0.0) +

          // T2P3
         (SUG.at(i) * DyEg(x,y) * PHD.Gas.GetDensity()
          * POD.GPhivPOD.at(i).at(j)
          * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

          // T2P4
         (SEG.at(i) * PHD.Gas.GetDensity() * POD.GPhivPOD.at(i).at(j)
          * DyUg(x,y) * POD.GPhiuPOD.at(i).at(0) * POD.Getdx()
          * POD.Getdy()) +

          // R1 + R2
         (0.0) +

          // R3P1
         (DaiVgMugt(x,y,j) * DyUg(x,y) * POD.GPhiuPOD.at(i).at(0)
          * POD.Getdx()) +

          // R3P2
         (DaiVgMugt(x,y,j) * DyUg(x,y) * DyUgi(x,y,0) * POD.Getdx()
          * POD.Getdy()) +

          // R3P3
         (DaiVgMugt(x,y,j) * DxUg(x,y) * (-4.0/3.0)
          * POD.GPhiuPOD.at(i).at(0) * POD.Getdy()) +

          // R3P4
         (DaiVgMugt(x,y,j) * DxUg(x,y) * (4.0/3.0)
          * DxUgi(x,y,0) * POD.Getdx() * POD.Getdy()) +

          // R3P5
         (-DaiVgMugt(x,y,j) * DxUg(x,y) * POD.GPhiuPOD.at(i).at(0)
```

```
        * POD.Getdx()) +

        // R3P6
       (((DaiVgMugt(x,y,j) * DxVg(x,y)) + (GetMugt(x,y)
        * DxVgi(x,y,j))) * DyUgi(x,y,0) * POD.Getdx() * POD.Getdy()) +

        // R4
       (DaiVgBeta(x,y,j) * (SUS.at(i) - SUG.at(i))
        * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVg4Vg(std::vector< double > & V, int i, int j)

   {

   double val;

   int x, y;

   y = (int) floor(i / POD.GetNx());
   x = (int) i - (y * POD.GetNx());

         // T1P1
   val =  (DtEg(x,y) * PHD.Gas.GetDensity()
         * POD.GPhivPOD.at(i).at(j) * POD.Getdx()
         * POD.GPhivPOD.at(i).at(0) * POD.Getdy()) +

         // T1P2
        (0.0) +

         // T2P1
        (DxEg(x,y) * PHD.Gas.GetDensity() * SUG.at(i)
         * POD.GPhivPOD.at(i).at(j) * POD.Getdx()
         * POD.GPhivPOD.at(i).at(0) * POD.Getdy()) +

         // T2P2
```

```
        (DyEg(x,y) * PHD.Gas.GetDensity() * 2.0 * SVG.at(i)
         * POD.GPhivPOD.at(i).at(j) * POD.Getdx()
         * POD.GPhivPOD.at(i).at(0) * POD.Getdy()) +

         // T2P3
        (SEG.at(i) * PHD.Gas.GetDensity() * SUG.at(i) * DxVgi(x,y,j)
         * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // R1 + R2
        (0.0) +

         // R3P1
        (((-DaiVgMugt(x,y,j) * DxVg(x,y) )
           + (GetMugt(x,y) * DxVgi(x,y,j)))
         * POD.GPhivPOD.at(i).at(0) * POD.Getdx()) +

         // R3P2
        (((DaiVgMugt(x,y,j) * DxVg(x,y) ) + (GetMugt(x,y)
         * DxVgi(x,y,j))) * DyVgi(x,y,0) * POD.Getdx() * POD.Getdy()) +

         // R3P3
        (((-DaiVgMugt(x,y,j) * SVG.at(i)) + (GetMugt(x,y)
         * POD.GPhivPOD.at(i).at(j) )) * DxVgi(x,y,0) * POD.Getdy()) +

         // R3P4
        (((-DaiVgMugt(x,y,j) * DxVg(x,y)) + (GetMugt(x,y)
         * DxVgi(x,y,j))) * DxVgi(x,y,0) * POD.Getdx() * POD.Getdy()) +

         // R3P5
        ((-4.0/3.0) * ((DaiVgMugt(x,y,j) * SVG.at(i)) + (GetMugt(x,y)
         * POD.GPhivPOD.at(i).at(j) )) * DyVgi(x,y,0) * POD.Getdx()) +

         // R3P6
        (((DaiVgMugt(x,y,j) * DyVg(x,y)) + (GetMugt(x,y)
         * DyVgi(x,y,j))) * (4.0/3.0) * DyVgi(x,y,0) * POD.Getdx()
         * POD.Getdy()) +

         // R4
        (((DaiVgBeta(x,y,j) * (SVS.at(i) - SVG.at(i))) - (GetBeta(x,y)
         * POD.GPhivPOD.at(i).at(j))) * POD.GPhivPOD.at(i).at(0)
         * POD.Getdx() * POD.Getdy() );

    if ( isnan(val) )
       val = 0.0;
```

```
    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUs4Vg(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    // NOTE:  All are 0.0 except R4
    val = DaiVgBeta(x,y,j) * (SUG.at(i) - SUS.at(i))
         * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy();

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVs4Vg(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    // NOTE:  all terms are 0 except RT4
```

```
    val = (((DaiVgBeta(x,y,j) * (SVG.at(i) - SVS.at(i))) + (GetBeta(x,y)
          * POD.GPhivPOD.at(i).at(j)))
          * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUg4Us(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    // NOTE:  all terms are 0 except RT4

    val = (((DaiUsBeta(x,y,j) * (SUS.at(i) - SUG.at(i))) + (GetBeta(x,y)
          * POD.PPhiuPOD.at(i).at(j)))
          * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVg4Us(std::vector< double > & V, int i, int j)

    {
```

```
    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    // NOTE:  all terms are 0 except RT4

    val = (DaiUsBeta(x,y,j) * (SVS.at(i) - SVG.at(i))
          * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUs4Us(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

          // T1P1 + T1P2
    val = (-DtEg(x,y) * PHD.Particle.GetDensity() * POD.Getdx()
          * POD.Getdy() *POD.PPhiuPOD.at(i).at(j)
          * POD.PPhiuPOD.at(i).at(0)) + 0.0 +

          // T2P1
         (-DxEg(x,y) * PHD.Particle.GetDensity() * pow(SUS.at(i), 2)
          * 2.0 * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

          // T2P2
         ((((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * DxUsi(x,y,j)
          * SUS.at(i))
```

```
 + ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * DxUs(x,y)
 * POD.PPhiuPOD.at(i).at(j)))
 * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // T2P3
(-DyEg(x,y) * PHD.Particle.GetDensity() * SVS.at(i)
 * POD.PPhiuPOD.at(i).at(j) * POD.Getdx()
 * POD.PPhiuPOD.at(i).at(0) * POD.Getdy()) +

 // T2P4 + (R1 + R2 + R3P1)
((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * SVS.at(i)
 * DyUsi(x,y,j)
 * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy())
 + (0.0) +

 // R3P2
(-0.5 * (1.0 + CoR) * DxMub(x,y) * DxUsi(x,y,j)
 * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // R3P3
(-0.5 * (1.0 + CoR) * DxxUsi(x,y,j)
 * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // R3P4
(-GetMup(x,y) * POD.PPhiuPOD.at(i).at(j) * DyUsi(x,y,0)
 * POD.Getdx()) +

 // R3P5
(GetMup(x,y) * DyUsi(x,y,j) * POD.Getdx()
 * DyUsi(x,y,0) * POD.Getdy()) +

 // R3P6
((-4.0/3.0) * GetMup(x,y) * POD.PPhiuPOD.at(i).at(j)
 * DxUsi(x,y,0) * POD.Getdy()) +

 // R3P7
((4.0/3.0) * GetMup(x,y) * DxUsi(x,y,j) * POD.Getdx()
 * DxUsi(x,y,0) * POD.Getdy()) +

 // R3P8 + R3P9
(-GetMup(x,y) * POD.PPhiuPOD.at(i).at(j) * DxUsi(x,y,0)
 * POD.Getdx()) + (0.0) +

 // R4
(((DaiUsBeta(x,y,j) * (SUG.at(i) - SUS.at(i))) + (GetBeta(x,y)
```

```
        * POD.PPhiuPOD.at(i).at(j) ))
        * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy() );

    if ( isnan(val) )
      val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVs4Us(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    // NOTE: There are all 0.0 except the following 3!

        // T2P1
    val = (-DxEg(x,y) * PHD.Particle.GetDensity() * SVS.at(i)
        * POD.PPhiuPOD.at(i).at(j)
        * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P4
       ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * DxVs(x,y)
        * POD.PPhiuPOD.at(i).at(j)
        * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // R4
       (DaiUsBeta(x,y,j) * (SVG.at(i) - SVS.at(i))
        * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
      val = 0.0;

    return val;
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUg4Vs(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    // NOTE:  all terms are 0 except RT4

    val = (((DaiVsBeta(x,y,j) * (SUS.at(i) - SUG.at(i))) + GetBeta(x,y) )
          * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVg4Vs(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    // NOTE:  all terms are 0 except RT4

    val = (((DaiVsBeta(x,y,j) * (SVS.at(i) - SVG.at(i))) + (GetBeta(x,y)
```

```
        * POD.PPhivPOD.at(i).at(j)))
        * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
        val = 0.0;

    return val;

    }
```

/*%%%*/

```
double NSSolver::GetPDUs4Vs(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    //NOTE:  All are zero except the following!

        // T2P3
    val = (-DyEg(x,y) * PHD.Particle.GetDensity() * SUS.at(i)
        * POD.PPhiuPOD.at(i).at(j)
        * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P4
       ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * DyUs(x,y)
        * POD.PPhiuPOD.at(i).at(0)
        * POD.PPhivPOD.at(i).at(j) * POD.Getdx() * POD.Getdy()) +

        // R3P9
       (GetMup(x,y) * DxVsi(x,y,j) * DxUsi(x,y,0) * POD.Getdx()
        * POD.Getdy()) +

        // R4
       (DaiVsBeta(x,y,j) * (SUG.at(i) - SUS.at(i))
        * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
```

```
        val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVs4Vs(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

        // T1P1 + T1P2
    val = ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity()
        * POD.PPhivPOD.at(i).at(j) * POD.Getdx()
        * POD.PPhivPOD.at(i).at(0 )* POD.Getdy()) + 0.0 +

        // T2P1
       (-DxEg(x,y) * PHD.Particle.GetDensity() * SUS.at(i)
        * POD.PPhivPOD.at(i).at(j) * POD.Getdx()
        * POD.PPhivPOD.at(i).at(0) * POD.Getdy()) +

        // T2P2
       (-DyEg(x,y) * PHD.Particle.GetDensity() * 2.0 * SVS.at(i)
        * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P3
       ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * DyVsi(x,y,j)
        * SVS.at(i)
        * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +
       ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * DyVs(x,y)
        * POD.PPhivPOD.at(i).at(j) * POD.PPhivPOD.at(i).at(0)
        * POD.Getdx() * POD.Getdy()) +

        // T2P4 + (R1 + R2 + R3P1)
       ((1.0 - SEG.at(i)) * PHD.Particle.GetDensity() * SUS.at(i)
        * DxVsi(x,y,j) * POD.PPhivPOD.at(i).at(0) * POD.Getdx()
```

```
      * POD.Getdy()) + 0.0 +

      // R3P2
      (-0.5 * (1.0 + CoR) * DyMub(x,y) * DyVsi(x,y,j)
      * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

      // R3P3
      (-0.5 * (1.0 + CoR) * GetMub(x,y) * DxxVsi(x,y,j)
      * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

      // R3P4
      (-GetMup(x,y) * DxVsi(x,y,j) * POD.PPhivPOD.at(i).at(0)
      * POD.Getdx()) +

      // R3P5
      (GetMup(x,y) * DxVsi(x,y,j) * DyVsi(x,y,0) * POD.Getdx()
      * POD.Getdy()) +

      // R3P6
      (-GetMup(x,y) * POD.PPhivPOD.at(i).at(j) * DxVsi(x,y,0)
      * POD.Getdy()) +

      // R3P7
      (GetMup(x,y) * DxVsi(x,y,j) * DxVsi(x,y,0)
      * POD.Getdx() * POD.Getdy()) +

      // R3P8
      ((-4.0/3.0) * GetMup(x,y) * DyVsi(x,y,0)
      * POD.PPhivPOD.at(i).at(j) * POD.Getdx()) +

      // R3P9
      ((4.0/3.0) * GetMup(x,y) * DyVsi(x,y,j) * POD.Getdx()
      * DyVsi(x,y,0) * POD.Getdy()) +

      // R4
      (((DaiVsBeta(x,y,j) * (SVG.at(i) - SVS.at(i))) + (GetBeta(x,y)
      * POD.PPhivPOD.at(i).at(j)))
      * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

if ( isnan(val) )
   val = 0.0;

return val;

}
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUg4Eg(std::vector< double > & V, int i, int j)

   {

   double val;

   int x, y;

   y = (int) floor(i / POD.GetNx());
   x = (int) i - (y * POD.GetNx());

        // T1P1
   val = (0.0) +

        // T1P2
       (POD.FgPhiPOD.at(i).at(j) * PHD.Gas.GetDensity() * DtUg(x,y)
        * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P1
       (DxEgi(x,y,j) * PHD.Gas.GetDensity() * pow(SUG.at(i), 2)
        * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P2
       (DxUg(x,y) * SUG.at(i) * POD.FgPhiPOD.at(i).at(j)
        * PHD.Gas.GetDensity()
        * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P3
       (SUG.at(i) * DyEgi(x,y,j) * PHD.Gas.GetDensity() * SVG.at(i)
        * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T2P4
       (POD.FgPhiPOD.at(i).at(j) * PHD.Gas.GetDensity() * SVG.at(i)
        * DyUg(x,y) * POD.GPhiuPOD.at(i).at(0) * POD.Getdx()
        * POD.Getdy()) +

        // R1 + R2
       (POD.FgPhiPOD.at(i).at(j) * DxPg(x,y)
        * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy())
        + 0.0 +
```

```
        // R3P1
        (-DaiEgMugt(x,y,j) * DyUg(x,y)
         * POD.GPhiuPOD.at(i).at(0) * POD.Getdx()) +

        // R3P2
        (DaiEgMugt(x,y,j) * DyUg(x,y) * DyUgi(x,y,0) * POD.Getdx()
         * POD.Getdy()) +

        // R3P3
        ((-4.0/3.0) * DaiEgMugt(x,y,j) * DxUg(x,y)
         * POD.GPhiuPOD.at(i).at(0) * POD.Getdy()) +

        // R3P4
        ((4.0/3.0) * DaiEgMugt(x,y,j) * DxUg(x,y) * DxUgi(x,y,0)
         * POD.Getdx() * POD.Getdy()) +

        // R3P5
        (-DaiEgMugt(x,y,j) * DxUg(x,y) * POD.GPhiuPOD.at(i).at(0)
         * POD.Getdx()) +

        // R3P6
        (-DaiEgMugt(x,y,j) * DxVg(x,y) * DyUgi(x,y,0) * POD.Getdx()
         * POD.Getdy()) +

        // R4
        (DaiEgBeta(x,y,j) * (SUS.at(i) - SUG.at(i))
         * POD.GPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVg4Eg(std::vector< double > & V, int i, int j)

   {

   double val;

   int x, y;
```

```
y = (int) floor(i / POD.GetNx());
x = (int) i - (y * POD.GetNx());

      // T1P1
val = (0.0) +

      // T1P2
     (POD.FgPhiPOD.at(i).at(j) * PHD.Gas.GetDensity() * DtVg(x,y)
      * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

      // T2P1
     (DxEgi(x,y,j) * PHD.Gas.GetDensity() * SUG.at(i) * SVG.at(i)
      * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

      // T2P2
     (DyEgi(x,y,j) * PHD.Gas.GetDensity() * pow(SVG.at(i), 2)
      * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

      // T2P3
     (POD.FgPhiPOD.at(i).at(j) * PHD.Gas.GetDensity() * SUG.at(i)
      * DxVg(x,y) * POD.GPhivPOD.at(i).at(0) * POD.Getdx()
      * POD.Getdy()) +

      // R1
     (-POD.FgPhiPOD.at(i).at(j) * DyPg(x,y)
      * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

      // R2
     (POD.FgPhiPOD.at(i).at(j) * PHD.Gas.GetDensity() * 980.665
      * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

      // R3P1
     (-DaiEgMugt(x,y,j) * DxVg(x,y) * POD.GPhivPOD.at(i).at(0)
      * POD.Getdx()) +

      // R3P2
     (DaiEgMugt(x,y,j) * DxVg(x,y) * DyVgi(x,y,0) * POD.Getdx()
      * POD.Getdy()) +

      // R3P3
     (-DaiEgMugt(x,y,j) * SVG.at(i) * DxVgi(x,y,0) * POD.Getdy()) +

      // R3P4
     (DaiEgMugt(x,y,j) * DxVg(x,y) * DxVgi(x,y,0) * POD.Getdx()
```

```
        * POD.Getdy()) +

        // R3P5
       ((-4.0/3.0) * DaiEgMugt(x,y,j) * SVG.at(i) * DyVgi(x,y,0)
        * POD.Getdx()) +

        // R3P6
       (DaiEgMugt(x,y,j) * DyVg(x,y) * (4.0/3.0) * DyVgi(x,y,0)
        * POD.Getdx() * POD.Getdy()) +

        // R4
       (DaiEgBeta(x,y,j) * (SVS.at(i) - SVG.at(i))
        * POD.GPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUs4Eg(std::vector< double > & V, int i, int j)

   {

   double val;

   int x, y;

   y = (int) floor(i / POD.GetNx());
   x = (int) i - (y * POD.GetNx());

        // T1P1
   val = (0.0) +

        // T1P2
       (-POD.FgPhiPOD.at(i).at(j) * PHD.Particle.GetDensity()
        * DtUs(x,y) * POD.PPhiuPOD.at(i).at(0) * POD.Getdx()
        * POD.Getdy()) +

        // T2P1
       (-DxEgi(x,y,j) * PHD.Particle.GetDensity() * pow(SUS.at(i), 2)
```

```
 * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // T2P2
(-POD.FgPhiPOD.at(i).at(j) * PHD.Particle.GetDensity()
 * DxUs(x,y) * SUS.at(i)
 * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // T2P3
(-DyEgi(x,y,j) * PHD.Particle.GetDensity() * SUS.at(i)
 * SVS.at(i)
 * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // T2P4
(-POD.FgPhiPOD.at(i).at(j) * PHD.Particle.GetDensity()
 * SVS.at(i)
 * DyUs(x,y) * POD.PPhiuPOD.at(i).at(0) * POD.Getdx()
 * POD.Getdy()) +

 // R1 + (R2 + R3P1)
(-POD.FgPhiPOD.at(i).at(j) * DxPg(x,y)
 * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy())
 + (0.0) +

 // R3P2
(-0.5 * (1.0 + CoR) * DaiEgDxMub(x,y,j) * DxUs(x,y)
 * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // R3P3
(-0.5 * (1.0 + CoR) * DaiEgMub(x,y,j) * DxxUs(x,y)
 * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // R3P4
(-DaiEgMup(x,y,j) * SUS.at(i) * DyUsi(x,y,0) * POD.Getdx()) +

 // R3P5
(DaiEgMup(x,y,j) * DyUs(x,y) * DyUsi(x,y,0) * POD.Getdx()
 * POD.Getdy()) +

 // R3P6
((-4.0/3.0) * DaiEgMup(x,y,j) * SUS.at(i) * DxUsi(x,y,0)
 * POD.Getdy()) +

 // R3P7
((4.0/3.0) * DaiEgMup(x,y,j) * DxUs(x,y) * DxUsi(x,y,0)
 * POD.Getdx() * POD.Getdy()) +
```

```
        // R3P8
       (-DaiEgMup(x,y,j) * SUS.at(i) * DxUsi(x,y,0) * POD.Getdx()) +

        // R3P9
       (DaiEgMup(x,y,j) * DxVs(x,y) * DxUsi(x,y,0) * POD.Getdx()
        * POD.Getdy()) +

        // R4
       (DaiEgBeta(x,y,j) * (SUG.at(i) - SUS.at(i))
        * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) ;

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVs4Eg(std::vector< double > & V, int i, int j)

   {

   double val;

   int x, y;

   y = (int) floor(i / POD.GetNx());
   x = (int) i - (y * POD.GetNx());

        // T1P1
   val = (-POD.FgPhiPOD.at(i).at(j) * PHD.Particle.GetDensity()
        * SVS.at(i)
        * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // T1P2
       (POD.FgPhiPOD.at(i).at(j) * PHD.Particle.GetDensity()
        * DtVs(x,y) * POD.PPhivPOD.at(i).at(0) * POD.Getdx()
        * POD.Getdy()) +

        // T2P1
       (-DxEgi(x,y,j) * PHD.Particle.GetDensity() * SVS.at(i)
```

```
 * SUS.at(i)
 * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // T2P2
(-DyEgi(x,y,j) * PHD.Particle.GetDensity() * pow(SVS.at(i), 2)
 * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // T2P3
(-POD.FgPhiPOD.at(i).at(j) * PHD.Particle.GetDensity()
 * DyVs(x,y) * SVS.at(i) * POD.PPhivPOD.at(i).at(0)
 * POD.Getdx() * POD.Getdy()) +

 // T2P4 + (R1 + R2 + R3P1)
(-POD.FgPhiPOD.at(i).at(j) * PHD.Particle.GetDensity()
 * DyVs(x,y) * SVS.at(i) *POD.PPhivPOD.at(i).at(0)
 * POD.Getdx() * POD.Getdy()) + (0.0) +

 // R3P2
(-0.5 * (1.0 + CoR) * DaiEgDyMub(x,y,j) * DyVs(x,y)
 * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // R3P3
(-0.5 * (1.0 + CoR) * DaiEgMub(x,y,j) * DyyVs(x,y)
 * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

 // R3P4
(DaiEgMup(x,y,j) * DxVs(x,y)
 * POD.PPhivPOD.at(i).at(0) * POD.Getdx()) +

 // R3P5
(DaiEgMup(x,y,j) * DxVs(x,y) * DyVsi(x,y,0) * POD.Getdx()
 * POD.Getdy()) +

 // R3P6
(-DaiEgMup(x,y,j) * SVS.at(i) * DxVsi(x,y,0) * POD.Getdy()) +

 // R3P7
(DaiEgMup(x,y,j) * DxVs(x,y) * DxVsi(x,y,0) * POD.Getdx()
 * POD.Getdy()) +

 // R3P8
((-4.0/3.0) * DaiEgMup(x,y,j) * SVS.at(i) * DyVsi(x,y,0)
 * POD.Getdx()) +

 // R3P9
```

```
        ((4.0/3.0) * DaiEgMup(x,y,j) * DyVs(x,y) * DyVsi(x,y,0)
         * POD.Getdx() * POD.Getdy()) +

         // R4
        (DaiEgBeta(x,y,j) * (SVG.at(i) - SVS.at(i))
         * POD.PPhivPOD.at(i).at(0)
         * POD.Getdx() * POD.Getdy());

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUg4Pg(std::vector< double > & V, int i, int j)

   {

   double val;

   int x, y;

   y = (int) floor(i / POD.GetNx());
   x = (int) i - (y * POD.GetNx());

   // NOTE:  All terms except RT1 are 0

   val = (SEG.at(i) * DxPgi(x,y,j) * POD.GPhiuPOD.at(i).at(0)
        * POD.Getdx() * POD.Getdy());

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVg4Pg(std::vector< double > & V, int i, int j)
```

```
    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    // NOTE:  All terms except RT1 are 0

    val = (-SEG.at(i) * DyPgi(x,y,j) * POD.GPhivPOD.at(i).at(0)
          * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUs4Pg(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    // NOTE:  All terms except RT1 are 0

    val = ((1.0 - SEG.at(i)) * DxPgi(x,y,j) * POD.PPhivPOD.at(i).at(0)
          * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
       val = 0.0;

    return val;
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVs4Pg(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    // NOTE:  All terms except RT1 are 0

    val = ((1.0 - SEG.at(i)) * DyPgi(x,y,j) * POD.PPhivPOD.at(i).at(0)
          * POD.Getdx() * POD.Getdy());

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUg4Ps(std::vector< double > & V, int i, int j)

    {

    double val;    // NOTE:  All terms except RT1 are 0


    val = 0.0;

    return val;

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVg4Ps(std::vector< double > & V, int i, int j)

   {

   double val;

   // NOTE:  ALL are 0.0
   val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUs4Ps(std::vector< double > & V, int i, int j)

   {

   double val;

   int x, y;

   y = (int) floor(i / POD.GetNx());
   x = (int) i - (y * POD.GetNx());

   // NOTE:  All are 0.0 except R3P1
   val = DxPsi(x,y,j) * POD.PPhiuPOD.at(i).at(0) * POD.Getdx()
                * POD.Getdy();

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVs4Ps(std::vector< double > & V, int i, int j)
```

```
    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    // NOTE:  All are 0.0 except R3P1
    val = DyPsi(x,y,j) * POD.PPhivPOD.at(i).at(0) * POD.Getdx()
                  * POD.Getdy();

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUg4Ts(std::vector< double > & V, int i, int j)

    {

    double val;

    // NOTE:  ALL are 0.0
    val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVg4Ts(std::vector< double > & V, int i, int j)

    {

    double val;
```

```
    // NOTE:  ALL are 0.0
    val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDUs4Ts(std::vector< double > & V, int i, int j)

    {

    double val;

    int x, y;

    y = (int) floor(i / POD.GetNx());
    x = (int) i - (y * POD.GetNx());

    //NOTE:  all previous ones are 0.0

         // R3P2
    val = (-0.5 * (1.0 + CoR) * DaiTsDxMub(x,y,j) * DxUs(x,y)
         * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // R3P3
        (-0.5 * (1.0 + CoR) * DaiTsMub(x,y,j) * DxxUs(x,y)
         * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // R3P4
        (-DaiTsMup(x,y,j) * SUS.at(i) * DyUsi(x,y,0) * POD.Getdx()) +

         // R3P5
        (-DaiTsMup(x,y,j) * DyUs(x,y)
         * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

         // R3P6
        ((-4.0/3.0) * DaiTsMup(x,y,j) * SUS.at(i) * DxUsi(x,y,0)
         * POD.Getdy()) +

         // R3P7
        ((4.0/3.0) * DaiTsMup(x,y,j) * DxUs(x,y) * DxUsi(x,y,0)
```

```
        * POD.Getdx() * POD.Getdy()) +

        // R3P8
       (-DaiTsMup(x,y,j) * SUS.at(i) * DxUsi(x,y,0) * POD.Getdx()) +

        // R3P9
       (DaiTsMup(x,y,j) * DxVs(x,y) * DxUsi(x,y,0) * POD.Getdx()
        * POD.Getdy()) +

        // R4
       (DaiTsBeta(x,y,j) * (SUG.at(i) - SUS.at(i))
        * POD.PPhiuPOD.at(i).at(0) * POD.Getdx() * POD.Getdy());

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPDVs4Ts(std::vector< double > & V, int i, int j)

   {

   double val;

   int x, y;

   y = (int) floor(i / POD.GetNx());
   x = (int) i - (y * POD.GetNx());

   //NOTE:  All previous ones are 0.0

        // R3P2
   val = (-0.5 * (1.0 + CoR) * DaiTsDyMub(x,y,j) * DyVs(x,y)
        * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // R3P3
       (-0.5 * (1.0 + CoR) * DaiTsMub(x,y,j) * DyyVs(x,y)
        * POD.PPhivPOD.at(i).at(0) * POD.Getdx() * POD.Getdy()) +

        // R3P4
```

```
        (DaiTsMup(x,y,j) * DxVs(x,y)
         * POD.PPhivPOD.at(i).at(0) * POD.Getdx()) +

         // R3P5
        (DaiTsMup(x,y,j) * DxVs(x,y) * DyVsi(x,y,0) * POD.Getdx()
         * POD.Getdy()) +

         // R3P6
        (-DaiTsMup(x,y,j) * SVS.at(i) * DxVsi(x,y,0) * POD.Getdy()) +

         // R3P7
        (DaiTsMup(x,y,j) * DxVs(x,y) * DxVsi(x,y,0) * POD.Getdx()
         * POD.Getdy()) +

         // R3P8
        ((-4.0/3.0) * DaiTsMup(x,y,j) * DyVs(x,y) * DyVsi(x,y,0)
         * POD.Getdx() * POD.Getdy()) +

         // R3P9 + R4
        ((4.0/3.0) * DaiTsMup(x,y,j) * DyVs(x,y) * DyVsi(x,y,0)
         * POD.Getdx() * POD.Getdy()) + 0.0;

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetGom(int i)

    {

    // This seems to be what is happening in MFIX a single
    // granular phase.  During literature searches I discovered
    // the Carnahan & Starling equation solution for a
    // 2D disc (3D I believe is what has been implemented)
    // was unknown as of 1998 and I couldn't find one since.
    // In G_0 it is set to G_0CS which is coded as follows...

    // Found this: Carnahan–Starling approximation g(x)=(1-x/2)/(1-x)^3
    //  where x is I believe the packing fraction, but couldn't tell if
```

```
    //  this should work for 2D...
    // Garzó, V., and Dufty, J. W., Phys. Rev. E 59, 5895–5911 (1999)

    float G_0cs;
    float ff = 1.0 - SEG.at(i);
//    float ff = Particle.EP.at(nx).at(ny).back();

    G_0cs = (1.0 / (1.0 - ff ) )
          + (1.5 * ff * pow((1.0 / (1.0 - ff)),2))
          + (0.5 * pow(ff,2) * pow(1.0 / (1.0 - ff), 3));

    if ( isnan(G_0cs) )
       G_0cs = 0.0;

    return G_0cs;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetVrm(int i)

    {

    double val;

    double A = pow(SEG.at(i), 4.14);
    double B;

    if (SEG.at(i) > 0.85)
       B = pow(SEG.at(i), 2.65);
    else
       B = pow(SEG.at(i), 1.28) * 0.8;

    val = 0.5 * (A - (0.06 * GetRem(i))
           + sqrt(pow(0.06 * GetRem(i), 2.0) + (0.12 * GetRem(i)
                        * ((2.0 *B) - A)) + pow(A, 2.0)));

    if ( isnan(val) )
       val = 0.0;

    return val;

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetRem(int i)

   {

   double val;

   val = PHD.Particle.GetDiam() * PHD.Gas.GetDensity()
        / PHD.Gas.GetViscosity();
   val = val * sqrt(pow(SUG.at(i) - SUS.at(i), 2.0)
              + pow(SVG.at(i) - SVS.at(i), 2.0));

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::PDUg(std::vector< double > & x, int i, int j)

   {

    // NOTE:  x is the vector, i is the equation index #
    //        and j is the index of the a_i th coefficient in which we
    //        are taking the partial derivative with respect to it.
   double val;

   int NUg = POD.GetNUg();
   int NVg = POD.GetNVg();
   int NUs = POD.GetNUs();
   int NVs = POD.GetNVs();
   int NEg = POD.GetNEg();
   int NPg = POD.GetNPg();
   int NPs = POD.GetNPs();
   int NTs = POD.GetNTs();

//    std::cout << "In PDUg \n";
```

```
    if (j < NUg)
       val = GetPDUg4Ug(x, i, j);
    else if (j < (NVg + NUg))
       val = GetPDUg4Vg(x, i, j - NUg);
    else if (j < (NUs + NVg + NUg))
       val = GetPDUg4Us(x, i, j - NUg - NVg);
    else if (j < (NVs + NUs + NVg + NUg))
       val = GetPDUg4Vs(x, i, j - NUg - NVg - NUs);
    else if (j < (NEg + NVs + NUs + NVg + NUg))
       val = GetPDUg4Eg(x, i, j - NUg - NVg - NUs- NVs);
    else if (j < (NPg + NEg + NVs + NUs + NVg + NUg))
       val = GetPDUg4Pg(x, i, j - NUg - NVg - NUs - NVs - NEg);
    else if (j < (NPs + NPg + NEg + NVs + NUs + NVg + NUg))
       val = GetPDUg4Ps(x, i, j - NUg - NVg - NUs - NVs - NEg - NPg);
    else if (j < (NTs + NPs + NPg + NEg + NVs + NUs + NVg + NUg))
       val = GetPDUg4Ts(x, i, j - NUg - NVg - NUs - NVs - NEg - NPg
                        - NPs);
//    std::cout << "Leaving PDUg \n";

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::PDVg(std::vector< double > & x, int i, int j)

    {

     // NOTE:  x is the vector, i is the equation index #
     //        and j is the index of the a_i th coefficient in which we
     //        are taking the partial derivative with respect to it.
    double val;

    int NUg = POD.GetNUg();
    int NVg = POD.GetNVg();
    int NUs = POD.GetNUs();
    int NVs = POD.GetNVs();
    int NEg = POD.GetNEg();
    int NPg = POD.GetNPg();
    int NPs = POD.GetNPs();
```

```
   int NTs = POD.GetNTs();

   if (j < NUg)
      val = GetPDVg4Ug(x, i, j);
   else if (j < (NVg + NUg))
      val = GetPDVg4Vg(x, i, j - NUg);
   else if (j < (NUs + NVg + NUg))
      val = GetPDVg4Us(x, i, j - NUg - NVg);
   else if (j < (NVs + NUs + NVg + NUg))
      val = GetPDVg4Vs(x, i, j - NUg - NVg - NUs);
   else if (j < (NEg + NVs + NUs + NVg + NUg))
      val = GetPDVg4Eg(x, i, j - NUg - NVg - NUs - NVs);
   else if (j < (NPg + NEg + NVs + NUs + NVg + NUg))
      val = GetPDVg4Pg(x, i, j - NUg - NVg - NUs - NVs - NEg);
   else if (j < (NPs + NPg + NEg + NVs + NUs + NVg + NUg))
      val = GetPDVg4Ps(x, i, j - NUg - NVg - NUs - NVs - NEg - NPg);
   else if (j < (NTs + NPs + NPg + NEg + NVs + NUs + NVg + NUg))
      val = GetPDVg4Ts(x, i, j - NUg - NVg - NUs - NVs - NEg - NPg
                          - NPs);

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::PDUs(std::vector< double > & x, int i, int j)

   {

    // NOTE:  x is the vector, i is the equation index #
    //       and j is the index of the a_i th coefficient in which we
    //       are taking the partial derivative with respect to it.
   double val;

   int NUg = POD.GetNUg();
   int NVg = POD.GetNVg();
   int NUs = POD.GetNUs();
   int NVs = POD.GetNVs();
   int NEg = POD.GetNEg();
   int NPg = POD.GetNPg();
```

```
   int NPs = POD.GetNPs();
   int NTs = POD.GetNTs();

   if (j < NUg)
      val = GetPDUs4Ug(x, i, j);
   else if (j < (NVg + NUg))
      val = GetPDUs4Vg(x, i, j - NUg);
   else if (j < (NUs + NVg + NUg))
      val = GetPDUs4Us(x, i, j - NUg - NVg);
   else if (j < (NVs + NUs + NVg + NUg))
      val = GetPDUs4Vs(x, i, j - NUg - NVg - NUs);
   else if (j < (NEg + NVs + NUs + NVg + NUg))
      val = GetPDUs4Eg(x, i, j - NUg - NVg - NUs - NVs);
   else if (j < (NPg + NEg + NVs + NUs + NVg + NUg))
      val = GetPDUs4Pg(x, i, j - NUg - NVg - NUs - NVs - NEg);
   else if (j < (NPs + NPg + NEg + NVs + NUs + NVg + NUg))
      val = GetPDUs4Ps(x, i, j - NUg - NVg - NUs - NVs - NEg - NPg);
   else if (j < (NTs + NPs + NPg + NEg + NVs + NUs + NVg + NUg))
      val = GetPDUs4Ts(x, i, j - NUg - NVg - NUs - NVs - NEg - NPg
                        - NPs);

   if ( isnan(val) )
      val = 0.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::PDVs(std::vector< double > & x, int i, int j)

   {

    // NOTE:  x is the vector, i is the equation index #
    //       and j is the index of the a_i th coefficient in which we
    //       are taking the partial derivative with respect to it.
   double val;

   int NUg = POD.GetNUg();
   int NVg = POD.GetNVg();
   int NUs = POD.GetNUs();
   int NVs = POD.GetNVs();
   int NEg = POD.GetNEg();
```

```
    int NPg = POD.GetNPg();
    int NPs = POD.GetNPs();
    int NTs = POD.GetNTs();

    if (j < NUg)
       val = GetPDVs4Ug(x, i, j);
    else if (j < (NVg + NUg))
       val = GetPDVs4Vg(x, i, j - NUg);
    else if (j < (NUs + NVg + NUg))
       val = GetPDVs4Us(x, i, j - NUg - NVg);
    else if (j < (NVs + NUs + NVg + NUg))
       val = GetPDVs4Vs(x, i, j - NUg - NVg - NUs);
    else if (j < (NEg + NVs + NUs + NVg + NUg))
       val = GetPDVs4Eg(x, i, j - NUg - NVg - NUs - NVs);
    else if (j < (NPg + NEg + NVs + NUs + NVg + NUg))
       val = GetPDVs4Pg(x, i, j - NUg - NVg - NUs - NVs - NEg);
    else if (j < (NPs + NPg + NEg + NVs + NUs + NVg + NUg))
       val = GetPDVs4Ps(x, i, j - NUg - NVg - NUs - NVs - NEg - NPg);
    else if (j < (NTs + NPs + NPg + NEg + NVs + NUs + NVg + NUg))
       val = GetPDVs4Ts(x, i, j - NUg - NVg - NUs - NVs - NEg - NPg
                           - NPs);

    if ( isnan(val) )
       val = 0.0;

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::FormTSums(std::vector< double > & x)

    { // This Routine sets SUG, SEG, etc. based upon
      // the current x.

    int i;
    int j;

    SUG.clear();
    SVG.clear();
```

```
    SEG.clear();
    SPG.clear();
    SUS.clear();
    SVS.clear();
    STS.clear();
    SPS.clear();

    for (j = 0; j < POD.GetNy(); j++)
        for (i = 0; i < POD.GetNx(); i++)
            {
            SUG.push_back(GetUgSum(x, i, j));
            SVG.push_back(GetVgSum(x, i, j));
            SUS.push_back(GetUsSum(x, i, j));
            SVS.push_back(GetVsSum(x, i, j));
            SEG.push_back(GetEgSum(x, i, j));
            SPG.push_back(GetPgSum(x, i, j));
            SPS.push_back(GetPsSum(x, i, j));
            STS.push_back(GetTsSum(x, i, j));
            }

//    std::cout<< "Just formed sums... \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::AccessUgSum(int t, int i, int j)

    {

    return UgSum.at(t).at(i+(j*POD.GetNx()));

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::AccessVgSum(int t, int i, int j)

    {

    return VgSum.at(t).at(i+(j*POD.GetNx()));

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::AccessUsSum(int t, int i, int j)

   {

   return UsSum.at(t).at(i+(j*POD.GetNx()));

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::AccessVsSum(int t, int i, int j)

   {

   return VsSum.at(t).at(i+(j*POD.GetNx()));

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::AccessPgSum(int t, int i, int j)

   {

   return PgSum.at(t).at(i+(j*POD.GetNx()));

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::AccessPsSum(int t, int i, int j)

   {

   return PsSum.at(t).at(i+(j*POD.GetNx()));

   }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::AccessEgSum(int t, int i, int j)

   {

   return EgSum.at(t).at(i+(j*POD.GetNx()));

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::AccessTsSum(int t, int i, int j)

   {

   return TsSum.at(t).at(i+(j*POD.GetNx()));

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetUgSum(std::vector< double > & x, int i, int j)

   {

   int N = 0;
   int MN = POD.GetNUg();
   int index;
   double val = POD.UgAvgVel.at(i).at(j);

//    std::cout << "Forming UgSum \n";

   for (index = N; index < MN; index ++)
      val = val + (x.at(index) *
               POD.GPhiuPOD.at(i + (j* POD.GetNx() ) ).at(index) );

//    std::cout << "Leaving UgSum \n";

   return val;

   }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetVgSum(std::vector< double > & x, int i, int j)

   {

   int N = POD.GetNUg();
   int MN = POD.GetNUg() + POD.GetNVg();
   int index;
   double val = POD.VgAvgVel.at(i).at(j);

//    std::cout << "Forming VgSum \n";

   for (index = N; index < MN; index ++)
      val = val + (x.at(index) *
               POD.GPhivPOD.at(i + (j* POD.GetNx() ) ).at(index - N) );

//    std::cout << "Leaving VgSum \n";

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetUsSum(std::vector< double > & x, int i, int j)

   {

   int N = POD.GetNUg() + POD.GetNVg();
   int MN = POD.GetNUg() + POD.GetNVg() + POD.GetNUs();
   int index;
   double val = POD.UsAvgVel.at(i).at(j);

//    std::cout << "Forming UsSum \n";

   for (index = N; index < MN; index ++)
      val = val + (x.at(index) *
          POD.PPhiuPOD.at(i + (j* POD.GetNx() ) ).at(index - N) );
   return val;

   }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetVsSum(std::vector< double > & x, int i, int j)

   {

   int N = POD.GetNUg() + POD.GetNVg() + POD.GetNUs();
   int MN = POD.GetNUg() + POD.GetNVg() + POD.GetNUs() + POD.GetNVs();
   int index;
   double val = POD.VsAvgVel.at(i).at(j);

//    std::cout << "Forming VsSum \n";

   for (index = N; index < MN; index ++)
      val = val + (x.at(index) *
           POD.PPhivPOD.at(i + (j* POD.GetNx() ) ).at(index - N) );

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetEgSum(std::vector< double > & x, int i, int j)

   {

   int N = POD.GetNUg() + POD.GetNVg() + POD.GetNUs() + POD.GetNVs();
   int MN = POD.GetNUg() + POD.GetNVg() + POD.GetNUs() + POD.GetNVs()
                    + POD.GetNEg();
   int index;
   double val = POD.FgAvg.at(i).at(j);

//    std::cout << "Forming EgSum \n";

   for (index = N; index < MN; index ++)
      val = val + (x.at(index) *
           POD.FgPhiPOD.at(i + (j* POD.GetNx() ) ).at(index - N) );

   return val;

   }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPgSum(std::vector< double > & x, int i, int j)

   {

   int N = POD.GetNUg() + POD.GetNVg() + POD.GetNUs() + POD.GetNVs()
                  + POD.GetNEg();
   int MN = POD.GetNUg() + POD.GetNVg() + POD.GetNUs() + POD.GetNVs()
                  + POD.GetNEg() + POD.GetNPg();
   int index;
   double val = POD.PgAvg.at(i).at(j);

//   std::cout << "Forming PgSum \n";

   for (index = N; index < MN; index ++)
      val = val + (x.at(index) *
           POD.PgPhiPOD.at(i + (j* POD.GetNx() ) ).at(index - N) );

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetPsSum(std::vector< double > & x, int i, int j)

   {

   int N = POD.GetNUg() + POD.GetNVg() + POD.GetNUs() + POD.GetNVs()
                  + POD.GetNEg() + POD.GetNPg();
   int MN = POD.GetNUg() + POD.GetNVg() + POD.GetNUs() + POD.GetNVs()
                  + POD.GetNEg() + POD.GetNPg() + POD.GetNPs();
   int index;
   double val = POD.PsAvg.at(i).at(j);

//   std::cout << "Forming PsSum \n";

   for (index = N; index < MN; index ++)
      val = val + (x.at(index) *
           POD.PsPhiPOD.at(i + (j* POD.GetNx() ) ).at(index - N) );
```

```
    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetTsSum(std::vector< double > & x, int i, int j)

    {

    int N = POD.GetNUg() + POD.GetNVg() + POD.GetNUs() + POD.GetNVs()
                    + POD.GetNEg() + POD.GetNPg() + POD.GetNPs();
    int MN = POD.GetNUg() + POD.GetNVg() + POD.GetNUs() + POD.GetNVs()
                     + POD.GetNEg() + POD.GetNPg() + POD.GetNPs()
                     + POD.GetNTs();
    int index;
    double val = POD.TsAvg.at(i).at(j);

//    std::cout << "Forming TsSum \n";

    for (index = N; index < MN; index ++)
        val = val + (x.at(index) *
             POD.TsPhiPOD.at(i + (j* POD.GetNx() ) ).at(index - N) );


//    std::cout << "Leaving TsSum \n";

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::Findf(std::vector< double > & x,
                  std::vector< double > & y)

    {

    // Number of equations of each type...
    int Neach = NEQs / (vectordims * 2);
    int i;

    for (i=0; i < NEQs; i++)
```

```
        { // Hard Coding specific to two dimensions...
        if (i >= (3*Neach))
           y.push_back( EvalVs(i - (3*Neach), x));
        else if (i >= (2 * Neach))
           y.push_back( EvalUs(i - (2*Neach), x));
        else if (i >= Neach)
           y.push_back( EvalVg(i - Neach, x));
        else
           y.push_back( EvalUg(i, x));
        } // end for

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::FormJac(std::vector< double > & x)

    {

    int i, j;
    int Neach = NEQs / (2 * vectordims);

    std::vector< double > row;

//    std::cout << "Entering jacobian... \n";

    Jac.clear();

    for (i=0; i < NEQs; i++)
        {
        row.clear();
        for (j=0; j < NUVs; j++)
            {
//            std::cout << i << " = i and j = " << j << "\n";
            if (i < Neach)
               row.push_back(PDUg(x, i, j));
            else if (i < (Neach * 2))
               row.push_back(PDVg(x, i - Neach, j));
            else if (i < (Neach * 3))
               row.push_back(PDUs(x, i - (2*Neach), j));
            else
               row.push_back(PDVs(x, i - (3*Neach), j));
//if (Id == 0)
//  std::cout << row.back() << " at " << i << " & " << j << "\n";
```

```
          } // end for j
        Jac.push_back(row);
        } // end for i

//    std::cout << "Leaving Jacobian... \n";


    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::SetJA(std::vector<double> & x)

    {// input is x and output is stored globally as JA

    int i, j, ii;

    MPI_Status status;

    int NumXRows = NUVs % NIds;
    int NumRowsEach = (NUVs - NumXRows) / NIds;

    if (Id == 0)
        {
//        std::cout << NumRowsEach
//                  << " will be calculated per processor. \n";
//        std::cout << NumXRows << " extra going on ID=0. \n";
        }
    else
        JA.clear();

    double val;
    std::vector< double > tmp; // (m,0.0);
    double tmpV[NumRowsEach*NUVs];

    FormJac(x);
//    std::cout << "Stored Jacobian... \n";

    JAmaxval = 0.0;
    // sum of squares will always be >= 0.0

    if (Id == 0)
        {
        for (i=0; i < (NumRowsEach + NumXRows); i++)
```

```
            {
            for (j=0; j < NUVs; j++)
               {
               if (j < i)
                  val = JA.at(j).at(i);
               else
                  {
                  val = 0.0;
                  for (ii = 0; ii < NEQs; ii++)
                     val = val + (Jac.at(ii).at(i)
                           * Jac.at(ii).at(j));
                  if (val > JAmaxval)
                     JAmaxval = val;
                  JA.at(i).at(j) = val;
                  }
               } //end for j
            } // end for i
         } // end ID == 0
      else
         {
         int c = 0;
         for (i = (NumXRows + (NumRowsEach * Id));
                i <(NumXRows + (NumRowsEach * (Id + 1))); i++)
            {
            for (j = 0; j < NUVs; j++)
               {
                  val = 0.0;
                  for (ii = 0; ii < NEQs; ii++)
                     val = val + (Jac.at(ii).at(i) * Jac.at(ii).at(j));
                  if (val > JAmaxval)
                     JAmaxval = val;
               tmpV[c] = val;
               c++;
               }
            } // end for i
         } // end else

//std::cout << Id << " finished matrix calculations... \n";

      MPI_Barrier(MPI_COMM_WORLD);


      if (Id > 0)
         {
//       std::cout << Id << " is sending mv " << JAmaxval << "\n";
```

```
        MPI_Send(&JAmaxval, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD);
        }
    else if ((NIds > 1) && (Id == 0))
        {
        double tmpmv;
        for (i = 1; i < NIds; i++)
            {
            MPI_Recv(&tmpmv, 1, MPI_DOUBLE, i, 1,
                                MPI_COMM_WORLD, &status);
//              std::cout << i << " sent " << tmpmv << "\n";
            if (tmpmv > JAmaxval)
                JAmaxval = tmpmv;
            }
        } // end else if

//std::cout << Id << " finished max value transfer... \n";

    int numpass = NumRowsEach * NUVs;
    if (Id > 0)
        MPI_Send(&tmpV, numpass, MPI_DOUBLE, 0, 2, MPI_COMM_WORLD);
    else if ((NIds > 1) && (Id == 0))
        for (i = 1; i < NIds; i++)
            {
            MPI_Recv(&tmpV[0], numpass,
                     MPI_DOUBLE, i, 2, MPI_COMM_WORLD, &status);
            int cindex = 0;
            int k, m;
            for (k=0; k < NumRowsEach; k++)
                for (m=0; m < NUVs; m++)
                    {
                    JA.at(k + (NumRowsEach * i) + NumXRows).at(m)
                          = tmpV[cindex];
                    cindex++;
                    }
            } //end for i receipt

    MPI_Barrier(MPI_COMM_WORLD);

//    std::cout << "Exiting JA mpi splitter... \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
void NSSolver::JacTf(std::vector<double> & x, std::vector<double> & y)

    {// input is x and output is y

    int i, j;
    double val;

    std::vector< double > f;

    y.clear();

    Findf(x, f);
//    std::cout << "Have f with size " << f.size() << " \n";

    for (i=0; i < NUVs; i++)
        {
        val = 0.0;
        for (j=0; j < NEQs; j++)
            {
//            std::cout << j << " is calculating. \n";
//if (Id == 0)
//  std::cout << Jac.at(j).at(i) * f.at(j)
//            << " at " << i << " & " << j << " for JacTf "
//            << Jac.at(j).at(i) << " & "
//            << f.at(j) << " are the components.\n";
            val = val + (Jac.at(j).at(i) * f.at(j));
            }
        y.push_back(val);
//if (Id == 0)
//  std::cout << val << " was pushed.  5083  \n";
        }

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetAmax(std::vector< std::vector< double >  > & A,
                         int m, int n)

    {

    int i,j;

    double val = A.at(0).at(0);
```

```
    for (i= 0; i < m; i++)
        for (j=0; j < n; j++)
            if (A.at(i).at(j) > val)
                val = A.at(i).at(j);

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::FormLUDecomp(std::vector< int > & index, double & d)

    {

    // This subroutine is based upon the ludcmp routine from the book
    // Numerical Recipes 2nd edition on page 49.

    double tiny = 1.0e-20;
    int i, imax, j, k;
    double big, dum, sum, temp;

    std::vector< double > vv(NUVs); // stores the scaling for each row

    d = 1.0;

    // get scaling info
    for (i = 0; i < NUVs; i++)
        {
        big = 0.0;
        for (j=0; j < NUVs; j++)
            if ((temp = fabs(JA.at(i).at(j))) > big)
                big = temp;
        if (big == 0.0)
            {
            std::cout << "Singular matrix in NSSolver::"
                    << "FormLUDecomp.  \n";
            exit (-1);
            }
        else
            vv.at(i) = 1.0 / big;
        }
```

```
// Loop over columns of Crout's Method
for (j = 0; j<NUVs; j++)
   {
   for (i = 0; i < j; i++)
      {
      sum = JA.at(i).at(j);
      for (k=0; k<i; k++)
         sum = sum - (JA.at(i).at(k) * JA.at(k).at(j));
      JA.at(i).at(j) = sum;
      }
   big = 0.0; // search for largest pivot element
   for (i=j; i<NUVs; i++)
      {
      sum = JA.at(i).at(j);
      for (k = 0; k < j; k++)
         sum = sum - (JA.at(i).at(k) * JA.at(k).at(j));
      JA.at(i).at(j) = sum;
      if ((dum = vv.at(i) * fabs(sum)) >= big)
         {// better pivot
         big = dum;
         imax = i;
         }
      }
   if (j != imax)
      { // interchange rows
      for (k = 0; k < NUVs; k++)
         {
         dum = JA.at(imax).at(k);
         JA.at(imax).at(k) = JA.at(j).at(k);
         JA.at(j).at(k) = dum;
         }
      d = -d;
      vv.at(imax) = vv.at(j); // change the scale factor
      }
   index.at(j) = imax;
   if (JA.at(j).at(j) == 0.0)
      JA.at(j).at(j) = tiny;
   //NOTE: if the pivot element is 0 the matrix is singular
   // a desirable substitution  of tiny is made.
   if (j != (NUVs-1))
      { // divide by the pivot element
      dum = 1.0 / JA.at(j).at(j);
      for (i = j + 1; i < NUVs; i++)
          JA.at(i).at(j) = dum * JA.at(i).at(j);
      } // end if j not NUVs-1
```

```
        }

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::DoLUBackSolve(std::vector< int > & index,
                        std::vector< double > & x)

    {

    std::vector< double > b;
    b = x;

    int i, ip, j;
    int ii = 0;
    double sum;

    for (i= 0; i <NUVs; i++)
        {
        ip = index.at(i);
        sum = b.at(ip);
        b.at(ip) = b.at(i);
        if (ii != 0)
            for (j = ii - 1; j < i; j++)
                sum = sum - (JA.at(i).at(j) * b.at(j));
        else if (sum != 0.0)
            ii = i+1;  // do sums above
        b.at(i) = sum;
        }

    for (i = NUVs-1; i >= 0; i--)
        {// perform back substitution
        sum = b.at(i);
        for (j = i+1; j < NUVs; j++)
            sum = sum - (JA.at(i).at(j) * b.at(j));
        b.at(i) = sum / JA.at(i).at(i);
        }// done :)
    x = b;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
void NSSolver::GetH(double mu, std::vector< double > & g,
                    std::vector<double> & h)

    {// mu, g and output is h

    int i;
    double d;

    std::vector< int > index (NUVs, 0);

//if (Id == 0)
//    std::cout << mu << " = mu \n";

    h.clear();

    for (i=0; i < NUVs; i++)
        {
        JA.at(i).at(i) = JA.at(i).at(i) + mu;
        h.push_back( -1.0 * g.at(i) );
        }

    // This now reduces to solving Ah = g for the vector h where A is JA

    FormLUDecomp(index, d);
    DoLUBackSolve(index, h);

    for (i=0; i < NUVs; i++)
        JA.at(i).at(i) = JA.at(i).at(i) - mu;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetDenom(std::vector< double > & h,
                          std::vector< double > & g, double mu)

    {// input is mu, g and h and output is a double

    int i;
    int imax = h.size();
    double val = 0.0;

    for (i=0; i < imax; i++)
```

```
        val = val + ( (0.5 * h.at(i)) *
                    ( (mu * h.at(i)) - g.at(i) ) );

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::infnorm(std::vector< double > & x)

    {// input is x and output is a double

    int i;
    int imax = x.size();
    double val = 0.0;

    for (i=0; i < imax; i++)
        if ( fabs(x.at(i)) > val)
            val = fabs(x.at(i));

    return val;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::norm(std::vector< double > & x)

    {// input is x and output is a double

    int i;
    int imax = x.size();
    double val = 0.0;

    for (i=0; i < imax; i++)
        val = val + pow(x.at(i), 2.0);

    val = sqrt(val);

    return val;

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

double NSSolver::GetF(std::vector< double > & x)

   {// input is x and output is a double

   double val = 0.0;
   int i;
   int m = x.size();

   for (i=0; i < m; i++)
      val = val + pow(x.at(i), 2.0);
   val = val / 2.0;

   return val;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::VAdd(std::vector< double > & x1,
                std::vector< double > & x2,
                std::vector< double > & y)

   {// input is x1 and x2 and output is y

   int mi = x1.size();
   int i;
   y.clear();

   for (i = 0; i < mi; i++)
      {
      y.push_back(x1.at(i) + x2.at(i));
//if ((Id == 0) && (i > 600))
//    std::cout << x2.at(i) << " is added to "
//            << x1.at(i) << " the " << i
//            << " component of x yielding "
//            << y.at(i) << "\n";
      }

   }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::LMM(std::vector< double > & x, double tol, int maxits,
                double ep1, double ep2)

   {

   int k;
   double v;
   double step;
   double mu;
   double mval;
   bool found;

   std::vector< double > g;
   std::vector< double > h;
   std::vector< double > y;
   std::vector< double > ynew;
   std::vector< double > xnew;

   MPI_Status status;

   double gA[NUVs];
   double hA[NUVs];
   int i;

   k = 0;  // iteration number count
   v = 2.0;

//    std::cout << "In LMM. \n";
//    std::cout << x.size() << " is the size of x. \n";
   FormTSums(x);
//    std::cout << Id << ": Formed sums \n";

   MPI_Barrier(MPI_COMM_WORLD);

   SetJA(x);      // Jac is formed and stored in this routine
                  // JA is obtained via matrix transpose the matrix
                  // this operation ALWAYS yields a square matrix!!
//    std::cout << "In LMM post transpose. \n";
   JacTf(x, g);
//    std::cout << "Have JacTf \n";

   MPI_Barrier(MPI_COMM_WORLD);
```

```
    if (infnorm(g) <= ep1)
       found = true;
    else
       found = false;
    if (Id == 0)
       std::cout << "Checking Norms (pre): "
                 << std::setiosflags(std::ios::fixed)
                 << std::setprecision(15)
                 << infnorm(g) << " & "
                 <<  ep1 << "\n";

    mu = tol * JAmaxval;

//    std::cout << "Entering loop... \n";
    while ((!found) && (k < maxits))
       {
       k++;
       if (k % 20 == 0)
          std::cout << "At " << k << " iteration. \n";
       MPI_Barrier(MPI_COMM_WORLD);
       if (Id == 0)
          {
          GetH(mu, g, h);
          for (i=0; i < NUVs; i++)
             {
             gA[i] = g.at(i);
             hA[i] = h.at(i);
             }
          for (i = 1; i < NIds; i++)
             {
             MPI_Send(&gA, NUVs, MPI_DOUBLE, i, 1, MPI_COMM_WORLD);
             MPI_Send(&hA, NUVs, MPI_DOUBLE, i, 2, MPI_COMM_WORLD);
             }
          } // end if Id == 0
       else
          {
          MPI_Recv(&gA[0], NUVs, MPI_DOUBLE, 0, 1,
                        MPI_COMM_WORLD, &status);
          MPI_Recv(&hA[0], NUVs, MPI_DOUBLE, 0, 2,
                        MPI_COMM_WORLD, &status);
             g.clear();
             h.clear();
             for (i=0; i < NUVs; i++)
                {
```

```
                g.push_back(gA[i]);
                h.push_back(hA[i]);
                }
            } // end else
        MPI_Barrier(MPI_COMM_WORLD);
        if (Id == 0)
            std::cout << "Checking Norms (loop): " << norm(h) << " & "
                    <<  (ep2 * (norm(x) + ep2)) << "\n";

        if (norm(h) <= (ep2 * (norm(x) + ep2)))
            found = true;
        else
            {
            VAdd(x, h, xnew);
            step = (GetF(x) - GetF(xnew)) / GetDenom(h, g, mu);
if (Id == 0)
    std::cout << step <<  " is step. \n";
            if (step > 0.0)
                {
                x = xnew;
                MPI_Barrier(MPI_COMM_WORLD);
                FormTSums(x);
                SetJA(x);
                JacTf(x, g);
                MPI_Barrier(MPI_COMM_WORLD);
                if (infnorm(g) <= ep1)
                    found = true;
                else
                    found = false;
                if ((1.0/3.0) > (1.0 - pow((2.0 * step) - 1.0, 3.0)))
                    mval = 1.0/3.0;
                else
                    mval = 1.0 - pow((2.0 * step) - 1.0, 3.0);
                mu = mu * mval;
                v = 2.0;
                } // end if step > 0
            else
                {
                mu = mu * v;
                v = 2.0 * v;
                } // end step <= 0
            } // end else  Norm is >
        } // end while
    if (Id == 0)
        std::cout << k << " iterations for converging solution. \n";
```

```
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::SetJA_NP(std::vector<double> & x)

    {// input is x and output is stored globally as JA

    int i, j, ii;
    double val;
    std::vector< double > tmp; // (m,0.0);
    float Perc;

    FormJac(x);
//    std::cout << "Stored Jacobian... \n";

    JAmaxval = 0.0;
    // sum of squares will always be >= 0.0

    JA.clear();
    for (i=0; i < NUVs; i++)
        {
        for (j=0; j < NUVs; j++)
            {
            if (j < i)
                val = JA.at(j).at(i);
            else
                {
                val = 0.0;
                for (ii = 0; ii < NEQs; ii++)
                    val = val + (Jac.at(ii).at(i) * Jac.at(ii).at(j));
                if (val > JAmaxval)
                    JAmaxval = val;
                }
            tmp.push_back(val);
            }
        JA.push_back(tmp);
        tmp.clear();
        Perc = i/NUVs;
//        std::cout << "Row " << i << " complete at " << Perc << "% \n";
        } // end for i

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::LMM_NP(std::vector< double > & x, double tol, int maxits,
                double ep1, double ep2)

   {

   int k;
   double v;
   double step;
   double mu;
   double mval;
   bool found;

   std::vector< double > g;
   std::vector< double > h;
   std::vector< double > y;
   std::vector< double > ynew;
   std::vector< double > xnew;

   k = 0;  // iteration number count
   v = 2.0;

//   std::cout << "In LMM. \n";
//   std::cout << x.size() << " is the size of x. \n";
   FormTSums(x);
   SetJA_NP(x);      // Jac is formed and stored in this routine
                 // JA is obtained via matrix transpose the matrix
                 // this operation ALWAYS yields a square matrix!!
//   std::cout << "In LMM post transpose. \n";
   JacTf(x, g);
//   std::cout << "Have JacTf \n";

   if (infnorm(g) <= ep1)
      found = true;
   else
      found = false;
   mu = tol * JAmaxval;

//   std::cout << "Entering loop... \n";
   while ((!found) && (k < maxits))
      {
      k++;
```

```
        GetH(mu, g, h);
//        std::cout << "Checking Norms... \n";
        if (norm(h) <= (ep2 * (norm(x) + ep2)))
           found = true;
        else
           {
           VAdd(x, h, xnew);
           step = (GetF(x) - GetF(xnew)) / GetDenom(h, g, mu);

           if (step > 0.0)
              {
              x = xnew;
              FormTSums(x);
              SetJA_NP(x);
              JacTf(x, g);
              if (infnorm(g) <= ep1)
                 found = true;
              else
                 found = false;
              if ((1.0/3.0) > (1.0 - pow((2.0 * step) - 1.0, 3)))
                 mval = 1.0/3.0;
              else
                 mval = 1.0 - pow((2.0 * step) - 1.0, 3);
              mu = mu * mval;
              v = 2.0;
              } // end if step > 0
           else
              {
              mu = mu * v;
              v = 2 * v;
              } // end step <= 0
           } // end else  Norm is >
        } // end while

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::CallLMM_NP(std::vector< double > & x,
                          std::vector< double > & y)
    {

    // Nx * Ny * 4 (2 for gas momentum equations and 2 for solids)
    //    int Neqs = Nx * Ny * 4;
```

```
    // NbasisSum = sum of number of basis of variables in the
    // calculations which I believe should be
    //  = NGU + NGV + NPU + NPV + NGF + NGP + NPP
    //  if Theta is in the calculations, then + NPT too

    int maxits = MaxIts;
    double tol = Tol;
    double ep1 = Epsilon1;
    double ep2 = Epsilon2;

//    tol = 1.0e-3;
//    maxits = 50;
//    ep1 = 1.0e-15;
//    ep2 = 1.0e-6;
    y = x;

    //  y goes in as the "best guess" and comes out as the solution
    LMM_NP(y, tol, maxits, ep1, ep2);

//    std::cout << "Have new solution. \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::CallLMM(std::vector< double > & x,
                    std::vector< double > & y)

    {

    // Nx * Ny * 4 (2 for gas momentum equations and 2 for solids)
    //    int Neqs = Nx * Ny * 4;
    // NbasisSum = sum of number of basis of variables in the
    // calculations which I believe should be
    //  = NGU + NGV + NPU + NPV + NGF + NGP + NPP
    //  if Theta is in the calculations, then + NPT too

    int maxits = MaxIts;
    double tol = Tol;
    double ep1 = Epsilon1;
    double ep2 = Epsilon2;

//    tol = 1.0e-3;
//    maxits = 10000;
```

```
//    ep1 = 1.0e-15;
//    ep2 = 1.0e-6;
    y = x;

    //  y goes in as the "best guess" and comes out as the solution
    LMM(y, tol, maxits, ep1, ep2);

//    std::cout << "Have new solution. \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::FormInitOld(std::vector< double > & V, int ind)

    {

    int i;

    V.clear();
    if (ind == 1)
        {
        for (i=0; i < POD.GetNUg(); i++)
            V.push_back(POD.PODsGasU.back().at(i));
        for (i=0; i < POD.GetNVg(); i++)
            V.push_back(POD.PODsGasV.back().at(i));
        for (i=0; i < POD.GetNUs(); i++)
            V.push_back(POD.PODsParticleU.back().at(i));
        for (i=0; i < POD.GetNVs(); i++)
            V.push_back(POD.PODsParticleV.back().at(i));
        for (i=0; i < POD.GetNEg(); i++)
            V.push_back(POD.PODsGasF.back().at(i));
        for (i=0; i < POD.GetNPg(); i++)
            V.push_back(POD.PODsGasP.back().at(i));
        for (i=0; i < POD.GetNPs(); i++)
            V.push_back(POD.PODsParticleP.back().at(i));
        for (i=0; i < POD.GetNTs(); i++)
            V.push_back(POD.PODsParticleT.back().at(i));
        }
    else if (ind == 0)
        {
        for (i=0; i < POD.GetNUg(); i++)
            V.push_back(POD.PODsGasU.at(0).at(i));
        for (i=0; i < POD.GetNVg(); i++)
```

```
            V.push_back(POD.PODsGasV.at(0).at(i));
        for (i=0; i < POD.GetNUs(); i++)
            V.push_back(POD.PODsParticleU.at(0).at(i));
        for (i=0; i < POD.GetNVs(); i++)
            V.push_back(POD.PODsParticleV.at(0).at(i));
        for (i=0; i < POD.GetNEg(); i++)
            V.push_back(POD.PODsGasF.at(0).at(i));
        for (i=0; i < POD.GetNPg(); i++)
            V.push_back(POD.PODsGasP.at(0).at(i));
        for (i=0; i < POD.GetNPs(); i++)
            V.push_back(POD.PODsParticleP.at(0).at(i));
        for (i=0; i < POD.GetNTs(); i++)
            V.push_back(POD.PODsParticleT.at(0).at(i));
        }

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::GetOldSol(std::vector< double > & V, int t)

    {

    int i;

//    std::cout << "Getting Old Solution for " << t << "\n";
    V.clear();
    for (i=0; i < POD.GetNUg(); i++)
        V.push_back(POD.PODsGasU.at(t).at(i));
    for (i=0; i < POD.GetNVg(); i++)
        V.push_back(POD.PODsGasV.at(t).at(i));
    for (i=0; i < POD.GetNUs(); i++)
        V.push_back(POD.PODsParticleU.at(t).at(i));
    for (i=0; i < POD.GetNVs(); i++)
        V.push_back(POD.PODsParticleV.at(t).at(i));
    for (i=0; i < POD.GetNEg(); i++)
        V.push_back(POD.PODsGasF.at(t).at(i));
    for (i=0; i < POD.GetNPg(); i++)
        V.push_back(POD.PODsGasP.at(t).at(i));
    for (i=0; i < POD.GetNPs(); i++)
        V.push_back(POD.PODsParticleP.at(t).at(i));
    for (i=0; i < POD.GetNTs(); i++)
        V.push_back(POD.PODsParticleT.at(t).at(i));
    if (Id == 0)
```

```
        std::cout << "Stored Old Solution for " << t << "\n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::ProjectSol(double endtime)

    {

    std::vector< double > OldSol;
    std::vector< double > NewSol;

    float t;
    float time = PHD.LastTimept();

    FormInitOld(OldSol, 1);
    Coeffs.push_back(OldSol);
    CurTS = POD.GetNt();

    float delt = GetDelt(time);
    t = time + delt;

//    std::cout << "Beginning Loop... \n";
    while (t <= endtime)
        { // 3rd term = #vars, 4th term = #eqns
        if (Id == 0)
            std::cout << t << " = t and endime = "
                      << endtime << "... \n";
        CallLMM(OldSol, NewSol);
//        std::cout << "Returned from CallLMM... \n";
        OldSol = NewSol;
        CurTS++;
        Coeffs.push_back(NewSol);
        StoreSums(NewSol);
        delt = GetDelt(t);
        t = t + delt;
        }
    if (Id == 0)
        {
        std::cout << "Writing Coefficients... \n";
        WriteSol(CurTS);
        std::cout << "Writing Solution... \n";
        WriteCoeffs(CurTS - POD.GetNt());
```

```
        }
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

float NSSolver::GetDelt(float time)

    {

    float dt;
    float maxt=0;
    int i, j, size;
    double dxug;
    double dyvg;

//std::cout << "In GetDelt() \n";
//size = lbc.size();
//for(i=0; i < size; i++)
//    std::cout << DyVg(0,i) / PHD.Getdely() << " ";

//    std::cout << " \n";

    if ((time == PHD.LastTimept()) ||
        (time == PHD.GetTimept(0)))
        {
        dxug = DxUgInit(i,j);
        dyvg = DyVgInit(i,j);
        }
    else
        {
        dxug = DxUg(i,j);
        dyvg = DyVg(i,j);
        }

    for (i=0; i < PHD.GetNx(); i++)
         for (j=0; j < PHD.GetNy(); j++)
              if ( fabs( ( dxug / PHD.Getdelx() ) +
                   ( dyvg / PHD.Getdely() ) ) > maxt)
                 maxt = fabs(( dxug / PHD.Getdelx() ) +
                   ( dyvg / PHD.Getdely() ));


    dt = 1.0 / maxt;
```

```
    std::cout << maxt << " is maxt. " << dt << "  \n";

    return dt;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::ExtrapSolVg(double value)

    {

    float MaxTime = PHD.LastTimept();
    int CurTS;
    float t;

    float delt = Dt;
    float time = PHD.GetTimept(0); // initial time

    std::vector< double > OldSol;
    std::vector< double > NewSol;

    delt = GetDelt(time);
    t = time + delt;

    CurTS = 1;
    int index = 0;
    FormInitOld(OldSol, 0);
    FormTSums(OldSol);
    Coeffs.push_back(OldSol);

    while (t <= MaxTime)
        {
        index++;
        if (Id == 0)
            std::cout << t << " = t and endime = "
                    << MaxTime << "... \n";
// For Debugging here...
//       std::cout << delt << " is delt. \n";
        GetOldSol(OldSol,CurTS);
        std::cout << "Solution Stored. \n";
        CallLMM(OldSol, NewSol);
        std::cout << "Have new sums. \n";
        OldSol = NewSol;
```

```
        FormTSums(NewSol);
        CurTS++;
        CurTS = CurTS + 10;
        Coeffs.push_back(NewSol);
        StoreSums(NewSol);
        delt = GetDelt(t);
        t = t + delt;
        if (Id == 0)
           WriteIndSol(index);
        }
    if (Id == 0)
        {
        std::cout << "Writing Solution... \n";
        WriteCoeffs(index-1);
//        WriteSol(CurTS);
        }
std::cout << "Done with Extrapolation! \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::ExtrapSol(double value, bool vg, bool e)

    {

    if (vg)
        ExtrapSolVg(value);
    else
    {
// Debugging here...
    float MaxTime = PHD.LastTimept();

    float delt = Dt;
    float t = PHD.GetTimept(0); // initial time

    int NumTPts = PHD.GetNt();
//    int NumTPts = floor((MaxTime - t)/delt) + 1;
//    std::cout << MaxTime << "=mt;"<< t
//              << "=t;delt=" << delt << "&" << NumTPts << " points to calculate. \n";

    std::vector< double > OldSol;
    std::vector< double > NewSol;
```

```
    int index = 0;
    int i = 0;
    DtVec.clear();
    IndexVec.clear();
//    FormInitOld(OldSol, 0);
//    Coeffs.push_back(OldSol);
//    IndexVec.push_back(i);
//    DtVec.push_back(delt*i);

//    if (Id == 0)
//        WriteIndSol(index);

    for (i=0; i < NumTPts; i++)
        {
        if (Id == (i % NIds))
            {
            std::cout << "Processor " << Id << " at "
                    << i << "\n";
            index++;
            GetOldSol(OldSol,i);
            FormTSums(OldSol);

            CallLMM_NP(OldSol, NewSol);
            Coeffs.push_back(NewSol);
            StoreSums(NewSol);
            IndexVec.push_back(i);
            DtVec.push_back(delt*i);
            WriteIndSol_NP(index,i);
            std::cout << index << " Writing Solution... \n";
            } // end if
        } // end for

//    if (Id == 0)
//        {
//        std::cout << "Writing Solution... \n";
//        WriteCoeffs(index-1);
//        WriteSol(CurTS);
//        }

    std::cout << Id << "Done with Extrapolation! \n";
//    MPI_Barrier(MPI_COMM_WORLD);
    }

    }
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::ExtrapSol_NP(double value, bool vg, bool e)

   {

   float MaxTime = PHD.LastTimept();
   int CurTS;

   float delt = Dt;
   float t = PHD.GetTimept(0); // initial time

   std::vector< double > OldSol;
   std::vector< double > NewSol;

   CurTS = 1;
   int index = 0;
   FormInitOld(OldSol, 0);
   FormTSums(OldSol);
   Coeffs.push_back(OldSol);

   while (t <= MaxTime)
      {
      index++;
      if (Id == 0)
         std::cout << t << " = t and endime = "
                 << MaxTime << "... \n";
// For Debugging here...
//       delt = GetDelt();
//       std::cout << delt << " is delt. \n";
      t = t + delt;
      GetOldSol(OldSol,CurTS);
      std::cout << "Solution Stored. \n";
      CallLMM(OldSol, NewSol);
      std::cout << "Have new sums. \n";
      OldSol = NewSol;
//       FormTSums(NewSol);
//       CurTS++;
      CurTS = CurTS + 10;
      Coeffs.push_back(NewSol);
      StoreSums(NewSol);
//       if ((Id == 0) && ((CurTS - 1) % 10 == 0))
      if (Id == 0)
         WriteIndSol(index);
```

```
        }
    if (Id == 0)
        {
        std::cout << "Writing Solution... \n";
        WriteCoeffs(index-1);
//        WriteSol(CurTS);
        }
//std::cout << "Done with Extrapolation! \n";
    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::InterpSol(double value, bool vg, bool e,
                         SimVars & PHD2, PODData & POD2,
                         std::vector< std::vector< double > > & Ug2,
                         std::vector< std::vector< double > > & Vg2,
                         std::vector< std::vector< double > > & Us2,
                         std::vector< std::vector< double > > & Vs2,
                         std::vector< std::vector< double > > & Eg2,
                         std::vector< std::vector< double > > & Pg2,
                         std::vector< std::vector< double > > & Ps2,
                         std::vector< std::vector< double > > & Ts2)

    {

// Solutions via "2" coding are the second set to Extrapolate PODs.
// Interpolated solution leaves via "2" coding.

// There are two ways this can be done.
// Only one of these two methodologies are technically correct...
// The first option: Express the solution in terms of a true
//     combination of the basis functions and solve for modes.
//     e.g. NUV1 + NUV2 = # basis functions NUV for interp model
//     This however violates the "orthogonal" part by definition...
//     Not to mention double the number of equations (which is VERY
//     expensive) and roughly the amount of unknowns. NOT very efficient!
// The second option: Solve for "solution" of each model separately via
//     extrapolation.  Average the solution, and reconstruct the POD
//     basis and modes on the averaged solution just like what was done
//     initially for the MFIX data to obtain those basis and modes...
//     Guaranteed to have faster run time, cost in accuracy would be
//     minimal and in all likelihood will yield better results (Hypoth.).
//     For the reasons stated above, this option is implemented here.
```

```
    std::vector< std::vector< double > > Coeffs1;
    std::vector< std::vector< double > > Coeffs2;

    std::vector< std::vector< double > > Ugs;
    std::vector< std::vector< double > > Vgs;
    std::vector< std::vector< double > > Uss;
    std::vector< std::vector< double > > Vss;
    std::vector< std::vector< double > > Pss;
    std::vector< std::vector< double > > Tss;
    std::vector< std::vector< double > > Pgs;
    std::vector< std::vector< double > > Egs;


//  Extrapolation 1:

    float MaxTime = PHD.LastTimept();
    float delt = Dt;
    float t = PHD.GetTimept(0); // initial time

    std::vector< double > OldSol1;
    std::vector< double > NewSol1;

    CurTS = 1;

    FormInitOld(OldSol1, 0);
    Coeffs.push_back(OldSol1);
    while (t < MaxTime)
       {
       if (Id == 0)
          std::cout << t << " = t and endime = "
                << MaxTime << "... \n";
       t = t + delt;
       CallLMM(OldSol1, NewSol1);
       OldSol1 = NewSol1;
       Coeffs.push_back(NewSol1);
       CurTS++;
       StoreSums(NewSol1);
       }

    Ugs = UgSum;
    Vgs = VgSum;
    Uss = UsSum;
    Vss = VsSum;
    Egs = EgSum;
```

```
    Pgs = PgSum;
    Pss = PsSum;
    Tss = TsSum;

std::cout << "Have made it this far in Interp. \n";
//  Extrapolation 2:

    NUVs =   POD2.GetNUg() + POD2.GetNVg()
          + POD2.GetNUs() + POD2.GetNVs()
          + POD2.GetNEg() + POD2.GetNPg()
          + POD2.GetNPs() + POD2.GetNTs();

    UgSum = Ug2;
    VgSum = Vg2;
    UsSum = Us2;
    VsSum = Vs2;
    EgSum = Eg2;
    PgSum = Pg2;
    PsSum = Ps2;
    TsSum = Ts2;

    std::vector< double > tds  (NUVs, 0.0);
    std::vector< std::vector< double > > td (NUVs, tds);
    JA = td;
    td.clear();


    MaxTime = PHD2.LastTimept();
    delt = PHD2.Getdelt();
    t = PHD2.GetTimept(0); // initial time

    std::vector< double > OldSol2;
    std::vector< double > NewSol2;

    CurTS = 1;

    Coeffs1 = Coeffs;
    Coeffs.clear();

    FormInitOld(OldSol2, 0);
    Coeffs.push_back(OldSol2);
    while (t <= MaxTime)
       {
       if (Id == 0)
          std::cout << t << " = t and endime = "
```

```
                << MaxTime << "... \n";
        t = t + delt;
        CallLMM(OldSol2, NewSol2);
        OldSol2 = NewSol2;
        Coeffs.push_back(NewSol2);
        CurTS++;
        StoreSums(NewSol2);
        }
    Coeffs2 = Coeffs;

//  Formulate solution averages

    Ug2.clear();
    Vg2.clear();
    Us2.clear();
    Vs2.clear();
    Eg2.clear();
    Pg2.clear();
    Ps2.clear();
    Ts2.clear();

    std::vector< double > tUg;
    std::vector< double > tVg;
    std::vector< double > tUs;
    std::vector< double > tVs;
    std::vector< double > tEg;
    std::vector< double > tPg;
    std::vector< double > tPs;
    std::vector< double > tTs;

    int i,j;

    for (j=0; j < CurTS; j++)
        {
        for (i = 0; i < (POD.GetNx() * POD.GetNy()); i++)
            {

            tUg.push_back( (Ugs.at(j).at(i) + UgSum.at(j).at(i)) / 2.0 );
            tVg.push_back( (Vgs.at(j).at(i) + VgSum.at(j).at(i)) / 2.0 );
            tUs.push_back( (Uss.at(j).at(i) + UsSum.at(j).at(i)) / 2.0 );
            tVs.push_back( (Vss.at(j).at(i) + VsSum.at(j).at(i)) / 2.0 );
            tEg.push_back( (Egs.at(j).at(i) + EgSum.at(j).at(i)) / 2.0 );
            tPg.push_back( (Pgs.at(j).at(i) + PgSum.at(j).at(i)) / 2.0 );
            tPs.push_back( (Pss.at(j).at(i) + PsSum.at(j).at(i)) / 2.0 );
            tTs.push_back( (Tss.at(j).at(i) + TsSum.at(j).at(i)) / 2.0 );
```

```
          }
        Ug2.push_back(tUg);
        Vg2.push_back(tVg);
        Us2.push_back(tUs);
        Vs2.push_back(tVs);
        Eg2.push_back(tEg);
        Pg2.push_back(tPg);
        Ps2.push_back(tPs);
        Ts2.push_back(tTs);

        tUg.clear();
        tVg.clear();
        tUs.clear();
        tVs.clear();
        tEg.clear();
        tPg.clear();
        tPs.clear();
        tTs.clear();

        }


    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

std::string NSSolver::int2String(int t)

    {

    std::string ch;

    std::ostringstream outs;

    outs << t; // Convert value into a string.

    ch = outs.str();

    return ch;

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
void NSSolver::WriteCoeffs(int NTPts)

    {

    int k;
    int t;

//    if (Id == (0 % NIds))
//        {
        std::string feg0 = "./Output/PODsGasVelocityX.txt";
        std::ofstream ofile0 (feg0.c_str());
        for (k=0; k < POD.GetNUg(); k++)
            {
            for (t=0; t < NTPts - 1; t++)
                ofile0 << std::setiosflags(std::ios::fixed)
                       << std::setprecision(15)
                       << Coeffs.at(t+1).at(k) << " ";
            ofile0 << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << Coeffs.at(NTPts).at(k) << " \n";
            }
        ofile0.close();
//        std::cout << Id << " wrote PODs \n";
//        } // end ID

//    if (Id == (1 % NIds))
//        {
        std::string feg1 = "./Output/PODsGasVelocityY.txt";
        std::ofstream ofile1 (feg1.c_str());
        for (k=POD.GetNUg(); k < (POD.GetNUg() + POD.GetNVg() ); k++)
            {
            for (t=0; t < NTPts - 1; t++)
                ofile1 << std::setiosflags(std::ios::fixed)
                       << std::setprecision(15)
                       << Coeffs.at(t+1).at(k) << " ";
            ofile1 << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << Coeffs.at(NTPts).at(k) << " \n";
            }
        ofile1.close();
//        std::cout << Id << " wrote PODs \n";
//        } // end ID

//    if (Id == (2 % NIds))
```

```
//       {
        std::string fes2 = "./Output/PODsParticleVelocityX.txt";
        std::ofstream ofile2 (fes2.c_str());
        for (k=(POD.GetNUg() + POD.GetNVg() );
             k < (POD.GetNUg() + POD.GetNVg() + POD.GetNUs() ); k++)
            {
            for (t=0; t < NTPts - 1; t++)
               ofile2 << std::setiosflags(std::ios::fixed)
                      << std::setprecision(15)
                      << Coeffs.at(t+1).at(k) << " ";
            ofile2 << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << Coeffs.at(NTPts).at(k) << "\n";
            }
        ofile2.close();
//        std::cout << Id << " wrote PODs \n";
//       } // end ID

//     if (Id == (3 % NIds))
//       {
        std::string fes3 = "./Output/PODsParticleVelocityY.txt";
        std::ofstream ofile3 (fes3.c_str());
        for (k=(POD.GetNUg() + POD.GetNVg() + POD.GetNUs() );
             k < (POD.GetNUg() + POD.GetNVg() + POD.GetNUs()
               + POD.GetNVs() ); k++)
            {
            for (t=0; t < NTPts - 1; t++)
               ofile3 << std::setiosflags(std::ios::fixed)
                      << std::setprecision(15)
                      << Coeffs.at(t+1).at(k) << " ";
            ofile3 << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << Coeffs.at(NTPts).at(k) << "\n";
            }
        ofile3.close();
//        std::cout << Id << " wrote PODs \n";
//       } // end ID

//     if (Id == (5 % NIds))
//       {
        std::string fegf = "./Output/PODsGasFF.txt";
        std::ofstream ofilegf (fegf.c_str());
        for (k=(POD.GetNUg() + POD.GetNVg() + POD.GetNUs()
              + POD.GetNVs() );
             k < (POD.GetNUg() + POD.GetNVg() + POD.GetNUs()
```

```
            + POD.GetNVs() + POD.GetNEg() ); k++)
          {
          for (t=0; t < NTPts - 1; t++)
             ofilegf << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << Coeffs.at(t+1).at(k) << " ";
          ofilegf << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << Coeffs.at(NTPts).at(k) << "\n";
          }
       ofilegf.close();
//       std::cout << Id << " wrote PODs \n";
//       } // enD 5

//    if (Id == (4 % NIds))
//       {
       std::string fegp = "./Output/PODsGasPressure.txt";
       std::ofstream ofilegp (fegp.c_str());
       for (k=(POD.GetNUg() + POD.GetNVg() + POD.GetNUs()
             + POD.GetNVs() + POD.GetNEg() );
            k < (POD.GetNUg() + POD.GetNVg() + POD.GetNUs()
             + POD.GetNVs() + POD.GetNEg() + POD.GetNPg() ); k++)
          {
          for (t=0; t < NTPts  - 1; t++)
             ofilegp << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << Coeffs.at(t+1).at(k) << " ";
          ofilegp << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << Coeffs.at(NTPts).at(k) << "\n";
          }
       ofilegp.close();
//       std::cout << Id << " wrote PODs \n";
//       } // END 4

//    if (Id == (6 % NIds))
//       {
       std::string fesp = "./Output/PODsParticlePressure.txt";
       std::ofstream ofilesp (fesp.c_str());
       for (k=(POD.GetNUg() + POD.GetNVg() + POD.GetNUs()
             + POD.GetNVs() + POD.GetNEg() + POD.GetNPg() );
            k < (POD.GetNUg() + POD.GetNVg() + POD.GetNUs()
             + POD.GetNVs() + POD.GetNEg() + POD.GetNPg()
             + POD.GetNPs() ); k++)
          {
```

```
          for (t=0; t < NTPts  - 1; t++)
             ofilesp << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << Coeffs.at(t+1).at(k) << " ";
          ofilesp << std::setiosflags(std::ios::fixed)
                 << std::setprecision(15)
                 << Coeffs.at(NTPts).at(k) << "\n";
          }
       ofilesp.close();
//       std::cout << Id << " wrote PODs \n";
//       } // end 6

//    if (Id == (7 % NIds))
//       {
       std::string fest = "./Output/PODsParticleTemperature.txt";
       std::ofstream ofilest (fest.c_str());
       for (k=(POD.GetNUg() + POD.GetNVg() + POD.GetNUs()
             + POD.GetNVs() + POD.GetNEg() + POD.GetNPg()
             + POD.GetNPs() );
           k < (POD.GetNUg() + POD.GetNVg() + POD.GetNUs()
             + POD.GetNVs() + POD.GetNEg() + POD.GetNPg()
             + POD.GetNPs() + POD.GetNTs() ); k++)
          {
          for (t=0; t < NTPts - 1; t++)
             ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << Coeffs.at(t+1).at(k) << " ";
          ofilest << std::setiosflags(std::ios::fixed)
                 << std::setprecision(15)
                 << Coeffs.at(NTPts).at(k) << "\n";
          }
       ofilest.close();
//       std::cout << Id << " wrote PODs \n";
//       } // end 7
    std::cout << "MFIX POD wrote PODs \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::WriteSol(int Nt)

    {
```

```
    int x;
    int y;
    int t;

//   Note: MPI stuff is commented out.
//    if (Id == (0 % NIds))
//       {
       // Create a text file - one per timestep
       //writing file
       for (t=0; t < Nt; t++)
          {
          std::string feg0 =
          "./Output/GasVelocityX" + int2String(t) + ".txt";
          std::ofstream ofileg (feg0.c_str());
          for (y=POD.GetNy()-1; y >=0; y--)
             {
             for (x=POD.GetNx()-1; x > 0; x--)
                {
                ofileg << std::setiosflags(std::ios::fixed)
                       << std::setprecision(15)
                       << UgSum.at(t).at( (y * POD.GetNx() ) + x)
                       << " ";
                }
             ofileg << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << UgSum.at(t).at(y * POD.GetNx() ) << "\n";
             }
          ofileg.close();
          } // end for t
//       } // end 0
//    if (Id == (1 % NIds))
//       {
       for (t=0; t < Nt; t++)
          {
          std::string feg1 =
          "./Output/GasVelocityY" + int2String(t) + ".txt";
          std::ofstream ofileg (feg1.c_str());
          for (y=POD.GetNy()-1; y >=0; y--)
             {
             for (x=POD.GetNx()-1; x > 0; x--)
                {
                ofileg << std::setiosflags(std::ios::fixed)
                       << std::setprecision(15)
                       << VgSum.at(t).at( (y * POD.GetNx() ) + x)
                       << " ";
```

```
                }
            ofileg << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << VgSum.at(t).at( y * POD.GetNx() ) << "\n";
            }
          ofileg.close();
          } // end for t
//        } // end 1
//    if (Id == (2 % NIds))l
//        {
        for (t=0; t < Nt; t++)
          {
          std::string fes2 =
          "./Output/ParticleVelocityX" + int2String(t) + ".txt";
          std::ofstream ofiles (fes2.c_str());
          for (y=POD.GetNy()-1; y >=0; y--)
             {
             for (x=POD.GetNx()-1; x > 0; x--)
                {
                ofiles << std::setiosflags(std::ios::fixed)
                       << std::setprecision(15)
                       << UsSum.at(t).at( (y * POD.GetNx() ) + x)
                       << " ";
                }
             ofiles << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << UsSum.at(t).at( y * POD.GetNx() ) << "\n";
             }
          ofiles.close();
          } // end for t
//        } // end for 2
//    if (Id == (3 % NIds))
//        {
        for (t=0; t < Nt; t++)
          {
          std::string fes3 =
          "./Output/ParticleVelocityY" + int2String(t) + ".txt";
          std::ofstream ofiles (fes3.c_str());
          for (y=POD.GetNy()-1; y >=0; y--)
             {
             for (x=POD.GetNx()-1; x > 0; x--)
                {
                ofiles << std::setiosflags(std::ios::fixed)
                       << std::setprecision(15)
                       << VsSum.at(t).at( (y * POD.GetNx() ) + x)
```

```
                    << " ";
                }
            ofiles << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << VsSum.at(t).at( y * POD.GetNx() ) << "\n";
            }
          ofiles.close();
          } // end for t
//       } // end for 3
//    if (Id == (4 % NIds))
//       {
       for (t=0; t < Nt; t++)
          {
          std::string fegp =
          "./Output/GasPressure" + int2String(t) + ".txt";
          std::ofstream ofilegp (fegp.c_str());
          for (y=POD.GetNy()-1; y >=0; y--)
             {
             for (x=POD.GetNx()-1; x > 0; x--)
                {
                ofilegp << std::setiosflags(std::ios::fixed)
                        << std::setprecision(15)
                        << PgSum.at(t).at( (y * POD.GetNx() ) + x)
                        << " ";
                }
             ofilegp << std::setiosflags(std::ios::fixed)
                     << std::setprecision(15)
                     << PgSum.at(t).at( y * POD.GetNx() ) << "\n";
             }
          ofilegp.close();
          } // end for t
//       } // end 4
//    if (Id == (5 % NIds))
//       {
       for (t=0; t < Nt; t++)
          {
          std::string fegf =
          "./Output/GasFF" + int2String(t) + ".txt";
          std::ofstream ofilegf (fegf.c_str());
          for (y=POD.GetNy()-1; y >=0; y--)
             {
             for (x=POD.GetNx()-1; x > 0; x--)
                {
                ofilegf << std::setiosflags(std::ios::fixed)
                        << std::setprecision(15)
```

```
                    << EgSum.at(t).at( (y * POD.GetNx() ) + x)
                    << " ";
                }
            ofilegf << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << EgSum.at(t).at( y * POD.GetNx() ) << "\n";
            }
          ofilegf.close();
          } // end for t
//        } // end 5
//    if (Id == (6 % NIds))
//        {
       for (t=0; t < Nt; t++)
          {
          std::string fesp =
          "./Output/ParticlePressure" + int2String(t) + ".txt";
          std::ofstream ofilesp (fesp.c_str());
          for (y=POD.GetNy()-1; y >=0; y--)
             {
             for (x=POD.GetNx()-1; x > 0; x--)
                {
                ofilesp << std::setiosflags(std::ios::fixed)
                        << std::setprecision(15)
                        << PsSum.at(t).at( (y * POD.GetNx() ) + x)
                        << " ";
                }
             ofilesp << std::setiosflags(std::ios::fixed)
                     << std::setprecision(15)
                     << PsSum.at(t).at( y * POD.GetNx() ) << "\n";
             }
          ofilesp.close();
          } // end for t
//        } // end 6
//    if (Id == (7 % NIds))
//        {
       for (t=0; t < Nt; t++)
          {
          std::string fest =
          "./Output/ParticleTemperature" + int2String(t) + ".txt";
          std::ofstream ofilest (fest.c_str());
          for (y=POD.GetNy()-1; y >=0; y--)
             {
             for (x=POD.GetNx()-1; x > 0; x--)
                {
                ofilest << std::setiosflags(std::ios::fixed)
```

```
                        << std::setprecision(15)
                        << TsSum.at(t).at( (y * POD.GetNx() ) + x)
                        << " ";
                    }
                ofilest << std::setiosflags(std::ios::fixed)
                        << std::setprecision(15)
                        << TsSum.at(t).at( y * POD.GetNx() ) << "\n";
                }
            ofilest.close();
            } // end for t
//        } // end 7

std::cout << "Wrote solution. \n";

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::WriteIndSol(int t)

    {

    int x;
    int y;

            std::string feg0 =
            "./Output/GasVelocityX" + int2String(t) + ".txt";
            std::ofstream ofileg (feg0.c_str());
            for (y=POD.GetNy()-1; y >=0; y--)
                {
                for (x=POD.GetNx()-1; x > 0; x--)
                    {
                    ofileg << std::setiosflags(std::ios::fixed)
                           << std::setprecision(15)
                           << UgSum.at(t).at( (y * POD.GetNx() ) + x)
                           << " ";
                    }
                ofileg << std::setiosflags(std::ios::fixed)
                       << std::setprecision(15)
                       << UgSum.at(t).at(y * POD.GetNx() ) << "\n";
                }
            ofileg.close();

            std::string feg1 =
```

```
"./Output/GasVelocityY" + int2String(t) + ".txt";
std::ofstream ofileg1 (feg1.c_str());
for (y=POD.GetNy()-1; y >=0; y--)
   {
   for (x=POD.GetNx()-1; x > 0; x--)
      {
      ofileg1 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << VgSum.at(t).at( (y * POD.GetNx() ) + x)
            << " ";
      }
   ofileg1 << std::setiosflags(std::ios::fixed)
         << std::setprecision(15)
         << VgSum.at(t).at( y * POD.GetNx() ) << "\n";
   }
ofileg1.close();

std::string fes2 =
"./Output/ParticleVelocityX" + int2String(t) + ".txt";
std::ofstream ofiles (fes2.c_str());
for (y=POD.GetNy()-1; y >=0; y--)
   {
   for (x=POD.GetNx()-1; x > 0; x--)
      {
      ofiles << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << UsSum.at(t).at( (y * POD.GetNx() ) + x)
            << " ";
      }
   ofiles << std::setiosflags(std::ios::fixed)
         << std::setprecision(15)
         << UsSum.at(t).at( y * POD.GetNx() ) << "\n";
   }
ofiles.close();

std::string fes3 =
"./Output/ParticleVelocityY" + int2String(t) + ".txt";
std::ofstream ofiles1 (fes3.c_str());
for (y=POD.GetNy()-1; y >=0; y--)
   {
   for (x=POD.GetNx()-1; x > 0; x--)
      {
      ofiles1 << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << VsSum.at(t).at( (y * POD.GetNx() ) + x)
```

```
            << " ";
        }
    ofiles1 << std::setiosflags(std::ios::fixed)
          << std::setprecision(15)
          << VsSum.at(t).at( y * POD.GetNx() ) << "\n";
    }
ofiles1.close();

std::string fegp =
"./Output/GasPressure" + int2String(t) + ".txt";
std::ofstream ofilegp (fegp.c_str());
for (y=POD.GetNy()-1; y >=0; y--)
    {
    for (x=POD.GetNx()-1; x > 0; x--)
        {
        ofilegp << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
              << PgSum.at(t).at( (y * POD.GetNx() ) + x)
              << " ";
        }
    ofilegp << std::setiosflags(std::ios::fixed)
          << std::setprecision(15)
          << PgSum.at(t).at( y * POD.GetNx() ) << "\n";
    }
ofilegp.close();

std::string fegf =
"./Output/GasFF" + int2String(t) + ".txt";
std::ofstream ofilegf (fegf.c_str());
for (y=POD.GetNy()-1; y >=0; y--)
    {
    for (x=POD.GetNx()-1; x > 0; x--)
        {
        ofilegf << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
              << EgSum.at(t).at( (y * POD.GetNx() ) + x)
              << " ";
        }
    ofilegf << std::setiosflags(std::ios::fixed)
          << std::setprecision(15)
          << EgSum.at(t).at( y * POD.GetNx() ) << "\n";
    }
ofilegf.close();

std::string fesp =
```

```
        "./Output/ParticlePressure" + int2String(t) + ".txt";
        std::ofstream ofilesp (fesp.c_str());
        for (y=POD.GetNy()-1; y >=0; y--)
          {
          for (x=POD.GetNx()-1; x > 0; x--)
            {
            ofilesp << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << PsSum.at(t).at( (y * POD.GetNx() ) + x)
                    << " ";
            }
          ofilesp << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << PsSum.at(t).at( y * POD.GetNx() ) << "\n";
          }
        ofilesp.close();

        std::string fest =
        "./Output/ParticleTemperature" + int2String(t) + ".txt";
        std::ofstream ofilest (fest.c_str());
        for (y=POD.GetNy()-1; y >=0; y--)
          {
          for (x=POD.GetNx()-1; x > 0; x--)
            {
            ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << TsSum.at(t).at( (y * POD.GetNx() ) + x)
                    << " ";
            }
          ofilest << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << TsSum.at(t).at( y * POD.GetNx() ) << "\n";
          }
        ofilest.close();

std::cout << "Wrote solution. \n";

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::WriteIndSol_NP(int t, int nt)

  {
```

```
int x;
int y;

        std::string feg0 =
        "./Output/GasVelocityX" + int2String(nt) + ".txt";
        std::ofstream ofileg (feg0.c_str());
        for (y=POD.GetNy()-1; y >=0; y--)
           {
           for (x=POD.GetNx()-1; x > 0; x--)
              {
              ofileg << std::setiosflags(std::ios::fixed)
                     << std::setprecision(15)
                     << UgSum.at(t).at( (y * POD.GetNx() ) + x)
                     << " ";
              }
           ofileg << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << UgSum.at(t).at(y * POD.GetNx() ) << "\n";
           }
        ofileg.close();

        std::string feg1 =
        "./Output/GasVelocityY" + int2String(nt) + ".txt";
        std::ofstream ofileg1 (feg1.c_str());
        for (y=POD.GetNy()-1; y >=0; y--)
           {
           for (x=POD.GetNx()-1; x > 0; x--)
              {
              ofileg1 << std::setiosflags(std::ios::fixed)
                      << std::setprecision(15)
                      << VgSum.at(t).at( (y * POD.GetNx() ) + x)
                      << " ";
              }
           ofileg1 << std::setiosflags(std::ios::fixed)
                   << std::setprecision(15)
                   << VgSum.at(t).at( y * POD.GetNx() ) << "\n";
           }
        ofileg1.close();

        std::string fes2 =
        "./Output/ParticleVelocityX" + int2String(nt) + ".txt";
        std::ofstream ofiles (fes2.c_str());
        for (y=POD.GetNy()-1; y >=0; y--)
           {
```

```
    for (x=POD.GetNx()-1; x > 0; x--)
       {
       ofiles << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
              << UsSum.at(t).at( (y * POD.GetNx() ) + x)
              << " ";
       }
    ofiles << std::setiosflags(std::ios::fixed)
           << std::setprecision(15)
           << UsSum.at(t).at( y * POD.GetNx() ) << "\n";
    }
ofiles.close();

std::string fes3 =
"./Output/ParticleVelocityY" + int2String(nt) + ".txt";
std::ofstream ofiles1 (fes3.c_str());
for (y=POD.GetNy()-1; y >=0; y--)
    {
    for (x=POD.GetNx()-1; x > 0; x--)
       {
       ofiles1 << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
              << VsSum.at(t).at( (y * POD.GetNx() ) + x)
              << " ";
       }
    ofiles1 << std::setiosflags(std::ios::fixed)
           << std::setprecision(15)
           << VsSum.at(t).at( y * POD.GetNx() ) << "\n";
    }
ofiles1.close();

std::string fegp =
"./Output/GasPressure" + int2String(nt) + ".txt";
std::ofstream ofilegp (fegp.c_str());
for (y=POD.GetNy()-1; y >=0; y--)
    {
    for (x=POD.GetNx()-1; x > 0; x--)
       {
       ofilegp << std::setiosflags(std::ios::fixed)
               << std::setprecision(15)
               << PgSum.at(t).at( (y * POD.GetNx() ) + x)
               << " ";
       }
    ofilegp << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
```

```
            << PgSum.at(t).at( y * POD.GetNx() ) << "\n";
    }
ofilegp.close();

std::string fegf =
"./Output/GasFF" + int2String(nt) + ".txt";
std::ofstream ofilegf (fegf.c_str());
for (y=POD.GetNy()-1; y >=0; y--)
    {
    for (x=POD.GetNx()-1; x > 0; x--)
        {
        ofilegf << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << EgSum.at(t).at( (y * POD.GetNx() ) + x)
                << " ";
        }
    ofilegf << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << EgSum.at(t).at( y * POD.GetNx() ) << "\n";
    }
ofilegf.close();

std::string fesp =
"./Output/ParticlePressure" + int2String(nt) + ".txt";
std::ofstream ofilesp (fesp.c_str());
for (y=POD.GetNy()-1; y >=0; y--)
    {
    for (x=POD.GetNx()-1; x > 0; x--)
        {
        ofilesp << std::setiosflags(std::ios::fixed)
                << std::setprecision(15)
                << PsSum.at(t).at( (y * POD.GetNx() ) + x)
                << " ";
        }
    ofilesp << std::setiosflags(std::ios::fixed)
            << std::setprecision(15)
            << PsSum.at(t).at( y * POD.GetNx() ) << "\n";
    }
ofilesp.close();

std::string fest =
"./Output/ParticleTemperature" + int2String(nt) + ".txt";
std::ofstream ofilest (fest.c_str());
for (y=POD.GetNy()-1; y >=0; y--)
    {
```

```
            for (x=POD.GetNx()-1; x > 0; x--)
               {
               ofilest << std::setiosflags(std::ios::fixed)
                     << std::setprecision(15)
                     << TsSum.at(t).at( (y * POD.GetNx() ) + x)
                     << " ";
               }
            ofilest << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << TsSum.at(t).at( y * POD.GetNx() ) << "\n";
            }
         ofilest.close();

std::cout << "Wrote solution. \n";

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void NSSolver::StoreSums(std::vector< double > & x)

   {

   UgSum.push_back(SUG);
   VgSum.push_back(SVG);
   UsSum.push_back(SUS);
   VsSum.push_back(SVS);
   EgSum.push_back(SEG);
   PgSum.push_back(SPG);
   PsSum.push_back(SPS);
   TsSum.push_back(STS);

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    VTKWriter.h
//    Created by Stephanie R. Beck Roth
//    Last Modified:   5/24/2011
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#ifndef VTKWRITER_H
#define VTKWRITER_H

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include "NSSolver.h"

class VTKWriter
   {
   private:

      int Nx;
      int Ny;
      int NCells;
      double Dx;
      double Dy;

//        void WriteUg();
//        void WriteVg();
//        void WriteUs();
//        void WriteVs();
//        void WritePg();
//        void WritePs();
//        void WriteEg();
//        void WriteTs();

   public:

      VTKWriter(int x, int y, double ddx, double ddy);
      ~VTKWriter();

      void WriteUg(std::vector< std::vector< double > > & V, int t,
                 double time, int index);
      void WriteVg(std::vector< std::vector< double > > & V, int t,
                 double time, int index);
```

```
        void WriteUs(std::vector< std::vector< double > > & V, int t,
                double time, int index);
        void WriteVs(std::vector< std::vector< double > > & V, int t,
                double time, int index);
        void WritePg(std::vector< std::vector< double > > & V, int t,
                double time, int index);
        void WritePs(std::vector< std::vector< double > > & V, int t,
                double time, int index);
        void WriteEg(std::vector< std::vector< double > > & V, int t,
                double time, int index);
        void WriteTs(std::vector< std::vector< double > > & V, int t,
                double time, int index);
        void WritePvdPg(std::vector<float> & tps);

    };

#endif
```

```
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
//    VTKWriter.cpp
//    Created by Stephanie R. Beck Roth
//    Last Modified:   7/07/2011
/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <sstream>
#include "stdio.h"
#include "NSSolver.h"
#include "VTKWriter.h"

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

VTKWriter::VTKWriter(int x, int y, double ddx, double ddy)

   {

   Nx = x;
   Ny = y;
   NCells = x * y;
   Dx = ddx;
   Dy = ddy;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

std::string int2String(int t)

   {

   std::string ch;

   std::ostringstream outs;

   outs << t; // Convert value into a string.
```

```
   ch = outs.str();

   return ch;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

std::string f2String(float t)

   {

   std::string ch;

   std::ostringstream outs;

   outs << t; // Convert value into a string.

   ch = outs.str();

   return ch;

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void VTKWriter::WriteUg(std::vector<std::vector< double > >& V, int t,
                  double time, int index)

   {

   int x;
   int y;

         std::string fest =
         "./Output/VTK/GasVelX"
         + int2String(t) + ".vtk";
         std::ofstream ofilest (fest.c_str());

         ofilest << "# vtk DataFile Version 1.0 \n"
               << "2D Unstructured Grid of VTK_QUAD \n"
               << "ASCII \n \n";
```

```
ofilest << "DATASET UNSTRUCTURED_GRID \n"
        << "POINTS "
        << (Nx + 1) * (Ny + 1)
        << " float \n";

float col3 = 0.0;
for (y=Ny; y >=0; y--)
   for (x=0; x <= Nx;  x++)
      ofilest << std::setiosflags(std::ios::fixed)
              << std::setprecision(2)
              << "\t" << x*Dx << "\t" << y*Dy
              << "\t" << col3 << "\n";

ofilest << "\n" << "CELLS "
        << NCells << " " << NCells * 5 << "\n";

int col1 = 4;
for (x=0; x < Nx;  x++)
   for (y=Ny-1; y >=0; y--)
      ofilest << std::setiosflags(std::ios::fixed)
              << std::setprecision(2) << "\t" << col1
              << "\t" << (x) + (y* (Nx+1))
              << "\t" << (x+1) + (y* (Nx+1))
              << "\t" << (x+1) + ((y+1) * (Nx+1))
              << "\t" << (x) + ((y+1)* (Nx+1))
              << "\n";

ofilest << "\n" << "CELL_TYPES " << NCells << "\n";

col1 = 9;
for (y=0; y < NCells; y++)
   ofilest << std::setiosflags(std::ios::fixed)
           << std::setprecision(2)<< "\t" << col1 << "\n";

ofilest << "\n" << "CELL_DATA " << NCells << "\n"
        << "SCALARS GasXVelocity" << time << " float \n"
        << "LOOKUP_TABLE default \n";

for (x=0; x < Nx; x++)
   for (y=0; y < Ny; y++)
      ofilest << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
              << "\t" << V.at(index).at((y * Nx) + x)
              << "\n";
```

```
        ofilest.close();

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void VTKWriter::WriteVg(std::vector< std::vector< double > > & V, int t,
                    double time, int index)

    {

    int x;
    int y;

        std::string fest =
        "./Output/VTK/GasVelY"
        + int2String(t) + ".vtk";
        std::ofstream ofilest (fest.c_str());

        ofilest << "# vtk DataFile Version 1.0 \n"
              << "2D Unstructured Grid of VTK_QUAD \n"
              << "ASCII \n \n";

        ofilest << "DATASET UNSTRUCTURED_GRID \n"
              << "POINTS "
              << (Nx + 1) * (Ny + 1)
              << " float \n";

        float col3 = 0.0;
        for (y=Ny; y >=0; y--)
           for (x=0; x <= Nx;  x++)
              ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(2)
                    << "\t" << x*Dx << "\t" << y*Dy
                    << "\t" << col3 << "\n";

        ofilest << "\n" << "CELLS "
              << NCells << " " << NCells * 5 << "\n";

        int col1 = 4;
        for (x=0; x < Nx;  x++)
           for (y=Ny-1; y >=0; y--)
              ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(2) << "\t" << col1
```

```
                    << "\t" << (x) + (y* (Nx+1))
                    << "\t" << (x+1) + (y* (Nx+1))
                    << "\t" << (x+1) + ((y+1) * (Nx+1))
                    << "\t" << (x) + ((y+1) * (Nx+1))
                    << "\n";

        ofilest << "\n" << "CELL_TYPES " << NCells << "\n";

        col1 = 9;
        for (y=0; y < NCells; y++)
           ofilest << std::setiosflags(std::ios::fixed)
                  << std::setprecision(2)<< "\t" << col1 << "\n";

        ofilest << "\n" << "CELL_DATA " << NCells << "\n"
              << "SCALARS GasYVelocity" << time << " float \n"
              << "LOOKUP_TABLE default \n";

        for (x=0; x < Nx; x++)
           for (y=0; y < Ny; y++)
              ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << "\t" << V.at(index).at((y * Nx) + x)
                    << "\n";

        ofilest.close();

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void VTKWriter::WriteUs(std::vector< std::vector< double > > & V, int t,
                  double time, int index)

   {

   int x;
   int y;
        std::string fest =
        "./Output/VTK/SolidsVelX"
        + int2String(t) + ".vtk";
        std::ofstream ofilest (fest.c_str());

        ofilest << "# vtk DataFile Version 1.0 \n"
              << "2D Unstructured Grid of VTK_QUAD \n"
```

```
        << "ASCII \n \n";

ofilest << "DATASET UNSTRUCTURED_GRID \n"
        << "POINTS "
        << (Nx + 1) * (Ny + 1)
        << " float \n";

float col3 = 0.0;
for (y=Ny; y >=0; y--)
   for (x=0; x <= Nx;  x++)
      ofilest << std::setiosflags(std::ios::fixed)
              << std::setprecision(2)
              << "\t" << x*Dx << "\t" << y*Dy
              << "\t" << col3 << "\n";

ofilest << "\n" << "CELLS "
        << NCells << " " << NCells * 5 << "\n";

int col1 = 4;
for (x=0; x < Nx;  x++)
   for (y=Ny-1; y >=0; y--)
      ofilest << std::setiosflags(std::ios::fixed)
              << std::setprecision(2) << "\t" << col1
              << "\t" << (x) + (y* (Nx+1))
              << "\t" << (x+1) + (y* (Nx+1))
              << "\t" << (x+1) + ((y+1) * (Nx+1))
              << "\t" << (x) + ((y+1)* (Nx+1))
              << "\n";

ofilest << "\n" << "CELL_TYPES " << NCells << "\n";

col1 = 9;
for (y=0; y < NCells; y++)
   ofilest << std::setiosflags(std::ios::fixed)
           << std::setprecision(2)<< "\t" << col1 << "\n";

ofilest << "\n" << "CELL_DATA " << NCells << "\n"
        << "SCALARS SolidsXVelocity" << time << " float \n"
        << "LOOKUP_TABLE default \n";

for (x=0; x < Nx; x++)
   for (y=0; y < Ny; y++)
      ofilest << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
              << "\t" << V.at(index).at((y * Nx) + x)
```

```
                    << "\n";

        ofilest.close();

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void VTKWriter::WriteVs(std::vector< std::vector< double > > & V, int t,
                  double time, int index)

    {

    int x;
    int y;
        std::string fest =
        "./Output/VTK/SolidsVelY"
        + int2String(t) + ".vtk";
        std::ofstream ofilest (fest.c_str());

        ofilest << "# vtk DataFile Version 1.0 \n"
              << "2D Unstructured Grid of VTK_QUAD \n"
              << "ASCII \n \n";

        ofilest << "DATASET UNSTRUCTURED_GRID \n"
              << "POINTS "
              << (Nx + 1) * (Ny + 1)
              << " float \n";

        float col3 = 0.0;
        for (y=Ny; y >=0; y--)
           for (x=0; x <= Nx;  x++)
              ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(2)
                    << "\t" << x*Dx << "\t" << y*Dy
                    << "\t" << col3 << "\n";

        ofilest << "\n" << "CELLS "
              << NCells << " " << NCells * 5 << "\n";

        int col1 = 4;
        for (x=0; x < Nx;  x++)
           for (y=Ny-1; y >=0; y--)
              ofilest << std::setiosflags(std::ios::fixed)
```

```
                    << std::setprecision(2) << "\t" << col1
                    << "\t" << (x) + (y* (Nx+1))
                    << "\t" << (x+1) + (y* (Nx+1))
                    << "\t" << (x+1) + ((y+1) * (Nx+1))
                    << "\t" << (x) + ((y+1)* (Nx+1))
                    << "\n";

        ofilest << "\n" << "CELL_TYPES " << NCells << "\n";

        col1 = 9;
        for (y=0; y < NCells; y++)
          ofilest << std::setiosflags(std::ios::fixed)
                << std::setprecision(2)<< "\t" << col1 << "\n";

        ofilest << "\n" << "CELL_DATA " << NCells << "\n"
              << "SCALARS SolidsYVelocity" << time << " float \n"
              << "LOOKUP_TABLE default \n";

        for (x=0; x < Nx; x++)
          for (y=0; y < Ny; y++)
            ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << "\t" << V.at(index).at((y * Nx) + x)
                    << "\n";

        ofilest.close();

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void VTKWriter::WritePg(std::vector< std::vector< double > > & V, int t,
                double time, int index)

    {

    int x;
    int y;
        std::string fest =
        "./Output/VTK/GasPressure"
        + int2String(t) + ".vtk";
        std::ofstream ofilest (fest.c_str());

        ofilest << "# vtk DataFile Version 1.0 \n"
```

```
        << "2D Unstructured Grid of VTK_QUAD \n"
        << "ASCII \n \n";

ofilest << "DATASET UNSTRUCTURED_GRID \n"
        << "POINTS "
        << (Nx + 1) * (Ny + 1)
        << " float \n";

float col3 = 0.0;
for (y=Ny; y >=0; y--)
   for (x=0; x <= Nx;  x++)
      ofilest << std::setiosflags(std::ios::fixed)
              << std::setprecision(2)
              << "\t" << x*Dx << "\t" << y*Dy
              << "\t" << col3 << "\n";

ofilest << "\n" << "CELLS "
        << NCells << " " << NCells * 5 << "\n";

int col1 = 4;
for (x=0; x < Nx;  x++)
   for (y=Ny-1; y >=0; y--)
      ofilest << std::setiosflags(std::ios::fixed)
              << std::setprecision(2) << "\t" << col1
              << "\t" << (x) + ((y)* (Nx+1))
              << "\t" << (x+1) + ((y)* (Nx+1))
              << "\t" << (x+1) + ((y+1) * (Nx+1))
              << "\t" << (x) + ((y+1)* (Nx+1))
              << "\n";

ofilest << "\n" << "CELL_TYPES " << NCells << "\n";

col1 = 9;
for (y=0; y < NCells; y++)
   ofilest << std::setiosflags(std::ios::fixed)
           << std::setprecision(2)<< "\t" << col1 << "\n";

ofilest << "\n" << "CELL_DATA " << NCells << "\n"
        << "SCALARS GasPressure float \n"
        << "LOOKUP_TABLE default \n";

for (x=0; x < Nx; x++)
   for (y=0; y < Ny; y++)
      ofilest << std::setiosflags(std::ios::fixed)
              << std::setprecision(15)
```

```
                    << "\t" << V.at(index).at((y * Nx) + x)
                    << "\t \n";

        ofilest.close();

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void VTKWriter::WritePs(std::vector< std::vector< double > > & V, int t,
            double time, int index)

    {

    int x;
    int y;

        std::string fest =
        "./Output/VTK/SolidsPressure"
        + int2String(t) + ".vtk";
        std::ofstream ofilest (fest.c_str());

        ofilest << "# vtk DataFile Version 1.0 \n"
                << "2D Unstructured Grid of VTK_QUAD \n"
                << "ASCII \n \n";

        ofilest << "DATASET UNSTRUCTURED_GRID \n"
                << "POINTS "
                << (Nx + 1) * (Ny + 1)
                << " float \n";

        float col3 = 0.0;
        for (y=Ny; y >=0; y--)
           for (x=0; x <= Nx;  x++)
              ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(2)
                    << "\t" << x*Dx << "\t" << y*Dy
                    << "\t" << col3 << "\n";

        ofilest << "\n" << "CELLS "
                << NCells << " " << NCells * 5 << "\n";

        int col1 = 4;
        for (x=0; x < Nx;  x++)
```

```
        for (y=Ny-1; y >=0; y--)
           ofilest << std::setiosflags(std::ios::fixed)
                 << std::setprecision(2) << "\t" << col1
                 << "\t" << (x) + (y* (Nx+1))
                 << "\t" << (x+1) + (y* (Nx+1))
                 << "\t" << (x+1) + ((y+1) * (Nx+1))
                 << "\t" << (x) + ((y+1)* (Nx+1))
                 << "\n";

     ofilest << "\n" << "CELL_TYPES " << NCells << "\n";

     col1 = 9;
     for (y=0; y < NCells; y++)
        ofilest << std::setiosflags(std::ios::fixed)
              << std::setprecision(2)<< "\t" << col1 << "\n";

     ofilest << "\n" << "CELL_DATA " << NCells << "\n"
           << "SCALARS SolidPressure" << time << " float \n"
           << "LOOKUP_TABLE default \n";

     for (x=0; x < Nx; x++)
        for (y=0; y < Ny; y++)
           ofilest << std::setiosflags(std::ios::fixed)
                 << std::setprecision(15)
                 << "\t" << V.at(index).at((y * Nx) + x)
                 << "\n";

     ofilest.close();

  }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void VTKWriter::WriteEg(std::vector< std::vector< double > > & V, int t,
                 double time, int index)

  {

  int x;
  int y;
     std::string fest =
     "./Output/VTK/GasVoidFrac"
     + int2String(t) + ".vtk";
     std::ofstream ofilest (fest.c_str());
```

```
ofilest << "# vtk DataFile Version 1.0 \n"
       << "2D Unstructured Grid of VTK_QUAD \n"
       << "ASCII \n \n";

ofilest << "DATASET UNSTRUCTURED_GRID \n"
       << "POINTS "
       << (Nx + 1) * (Ny + 1)
       << " float \n";

float col3 = 0.0;
for (y=Ny; y >=0; y--)
   for (x=0; x <= Nx;  x++)
      ofilest << std::setiosflags(std::ios::fixed)
            << std::setprecision(2)
            << "\t" << x*Dx << "\t" << y*Dy
            << "\t" << col3 << "\n";

ofilest << "\n" << "CELLS "
       << NCells << " " << NCells * 5 << "\n";

int col1 = 4;
for (x=0; x < Nx;  x++)
   for (y=Ny-1; y >=0; y--)
      ofilest << std::setiosflags(std::ios::fixed)
            << std::setprecision(2) << "\t" << col1
            << "\t" << (x) + (y* (Nx+1))
            << "\t" << (x+1) + (y* (Nx+1))
            << "\t" << (x+1) + ((y+1) * (Nx+1))
            << "\t" << (x) + ((y+1)* (Nx+1))
            << "\n";

ofilest << "\n" << "CELL_TYPES " << NCells << "\n";

col1 = 9;
for (y=0; y < NCells; y++)
   ofilest << std::setiosflags(std::ios::fixed)
         << std::setprecision(2)<< "\t" << col1 << "\n";

ofilest << "\n" << "CELL_DATA " << NCells << "\n"
       << "SCALARS GasVoidFrac" << time << " float \n"
       << "LOOKUP_TABLE default \n";

for (x=0; x < Nx; x++)
   for (y=0; y < Ny; y++)
```

```
            ofilest << std::setiosflags(std::ios::fixed)
                  << std::setprecision(15)
                  << "\t" << V.at(index).at((y * Nx) + x)
                  << "\n";

        ofilest.close();

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void VTKWriter::WriteTs(std::vector< std::vector< double > > & V, int t,
                  double time, int index)

    {

    int x;
    int y;

        std::string fest =
        "./Output/VTK/GranTemp"
        + int2String(t) + ".vtk";
        std::ofstream ofilest (fest.c_str());

        ofilest << "# vtk DataFile Version 1.0 \n"
            << "2D Unstructured Grid of VTK_QUAD \n"
            << "ASCII \n \n";

        ofilest << "DATASET UNSTRUCTURED_GRID \n"
            << "POINTS "
            << (Nx + 1) * (Ny + 1)
            << " float \n";

        float col3 = 0.0;
        for (y=Ny; y >=0; y--)
          for (x=0; x <= Nx;  x++)
            ofilest << std::setiosflags(std::ios::fixed)
                  << std::setprecision(2)
                  << "\t" << x*Dx << "\t" << y*Dy
                  << "\t" << col3 << "\n";

        ofilest << "\n" << "CELLS "
            << NCells << " " << NCells * 5 << "\n";
```

```
        int col1 = 4;
        for (x=0; x < Nx;  x++)
           for (y=Ny-1; y >=0; y--)
              ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(2) << "\t" << col1
                    << "\t" << (x) + (y * (Nx+1))
                    << "\t" << (x+1) + (y* (Nx+1))
                    << "\t" << (x+1) + ((y+1) * (Nx+1))
                    << "\t" << (x) + ((y+1)* (Nx+1))
                    << "\n";

        ofilest << "\n" << "CELL_TYPES " << NCells << "\n";

        col1 = 9;
        for (y=0; y < NCells; y++)
           ofilest << std::setiosflags(std::ios::fixed)
                 << std::setprecision(2)<< "\t" << col1 << "\n";

        ofilest << "\n" << "CELL_DATA " << NCells << "\n"
              << "SCALARS GranTemp" << time << " float \n"
              << "LOOKUP_TABLE default \n";

        for (x=0; x < Nx; x++)
           for (y=0; y < Ny; y++)
              ofilest << std::setiosflags(std::ios::fixed)
                    << std::setprecision(15)
                    << "\t" << V.at(index).at((y * Nx) + x)
                    << "\n";

        ofilest.close();

   }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

void VTKWriter::WritePvdPg(std::vector<float> & tps)

   {

   int i;

   std::string fest = "./Output/VTK/VTK.pvd";
   std::ofstream ofile (fest.c_str());
```

```
    ofile << "<?xml version=\"1.0\"?> \n";
    ofile << "<VTKFile type=\"Collection\" version"
          << "=\"0.1\" byte_order=\"LittleEndian\"> <Collection> \n";

// "<DataSet timestep="00000.000000" group="" part="0" file=".vtk"/> ";

    for (i=0; i < tps.size(); i++)
        {

        ofile << "<DataSet timestep=\"f2String(tps.at(i))\" group=\"\" part=\"0\" "
              << "file=\"GasPressure "<< int2String(i) << ".vtk\"/> \n";

        }

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/

VTKWriter::~VTKWriter()

    {

    }

/*%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%*/
```