2019

A Study Of Approximation Error In Eulerian Hydrocodes

Parth Yogeshbhai Patel
University of Alabama at Birmingham

A STUDY OF APPROXIMATION ERROR IN EULERIAN HYDROCODES

by

PARTH YOGESHBHAI PATEL

DAVID L. LITTLEFIELD, CHAIR
DEAN SICKING
LEE MORADI

A THESIS

Submitted to the graduate faculty of The University of Alabama at Birmingham,
in partial fulfillment of the requirements for the degree of
Master of Science

BIRMINGHAM, ALABAMA

2019

A STUDY OF APPROXIMATION ERROR IN EULERIAN HYDROCODES

PARTH YOGESHBHAI PATEL

MECHANICAL ENGINEERING

ABSTRACT

In this study, we examine a number of approximations in the formulation of hydrocodes. These approximations were borne out of an original requirement for the code to run as fast as possible i.e. with accuracy being secondary to speed. Many of these approximations originated from the 1970's when computers were slow and memory was at a premium. Although speed and memory are not as much of an issue today, these approximations are still used to formulate the hydrocodes. In this study, the effect of these approximations is examined systematically.

The lumped mass approximation is a simplification to the consistent mass formulation and is routinely used in hydrocodes. While this approximation is computationally efficient, the consistent mass formulation is the most accurate (and computationally expensive) option. There are other levels of approximation between these two extremes that trade off computational efficiency for accuracy. As is shown in this work, some of these result in tridiagonal systems which are very computationally efficient to solve. We introduce these algorithms in this work and refer to them as the *reduced consistent mass method*.

Linear finite elements are also used pervasively in hydrocodes. Like the lumped mass approximation, the use of linear elements was borne out of the requirement for computational efficiency and not accuracy. Surprisingly, linear elements are still used routinely today, despite their numerous accuracy issues such as realistic representation of

geometry and the need for hourglass stabilization. In this work higher order finite elements, including quadratic and cubic elements, are examined. Special attention is placed on quadrature order used in integration and its effect on overall accuracy.

The 2D version of ALEAS (Arbitrary Lagrangian-Eulerian Adaptive Solver), an in-house ALE (Arbitrary Lagrangian-Eulerian) research code, is used in this work. Some simple benchmark problems are used to assess and quantify the effect of higher order approximations in Eulerian hydrocodes.

Keywords: ALEAS, Hydrocodes, Lumped mass, Consistent mass, Tridiagonal systems

ACKNOWLEDGEMENTS

I would like to express my gratitude to Dr. David Littlefield, my research advisor, for his guidance, and patience, which have helped me to become an independent researcher. It was his ideas that motivated this work, and without his expertise none of this would have been possible. Also, I would like to express my gratitude especially to Dr. Kenneth Cline Walls, III, for his constant guidance in coding.

I am also grateful to Dr. Dean Sicking, and Dr. Lee Moradi for serving on my committee and providing valuable feedback on this work.

I would also like to thank Mechanical Engineering Department at the University of Alabama at Birmingham. During my time as a student in this department, they have provided tremendous support. I am very delighted to have worked with all of them.

Lastly, I would like to thank my family and my guru for the support and encouragement they have provided during this process. My father, Y. N. Patel, taught me the value of hard work, and patience and for that I am forever grateful. I would like to especially thank my guru, Mahant Swami Maharaj, who has taught me values of unity, positivity, and loyalty, my mother, Varsha, and sister, Hima for believing in me.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF ABBREVIATIONS

ALE	Arbitrary Lagrangian-Eulerian
ALEAS	Arbitrary Lagrangian-Eulerian Adaptive Solver

CHAPTER 1

INTRODUCTION

Hydrocode Modeling

Hydrocodes are computer software packages that can be used for the numerical solution of mathematical models. This includes ability to accurately approximate the mathematical models. Hydrocodes also include capabilities to solve structural dynamics models. This makes them useful for modelling vehicular collisions, biomechanical injury analysis, planetary impacts, explosions, hypervelocity impacts and penetrations, fluid structure interactions, and many more.

Hydrocodes formulate dynamical structural models and compute approximate solutions, which can involve various types of materials with very different properties. Thus, the user is responsible for constructing the most accurate model and also for specifying equations that govern how the materials will behave under specific conditions. Development of this code is still used in today's generation of software such as EPIC and ALEAS. The ALEAS hydrocode is used in this work.

Hydrocodes are mainly categorized by the frame of reference of the computational mesh. Each frame of reference has his own advantages and disadvantages that the user must have to account for before modelling with the code.

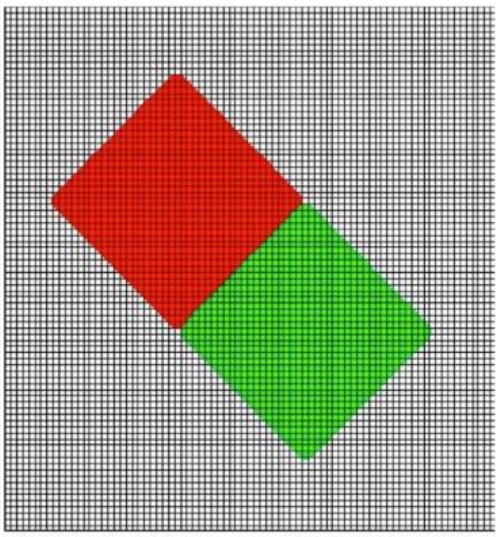

Figure 1.1: Representations of the Eulerian computational mesh

The Lagrangian formulation uses a computational mesh that is fixed in the material domain and no material passes between elements. Eulerian formulations use a mesh that is fixed in space and material flows through the mesh, as show in Figure 1.1. In this work, Eulerian computational mesh is used because it allows large stresses and deformations as the mesh is fixed in space.

History of the Finite Element Method

Courant appears to have been the first to propose the finite element method in history. In a 1941 mathematics lecture, which was published in 1943, he used the principle of stationary potential energy and piecewise polynomial interpolation over triangular subregions to study the Saint-Venant torsion problem [1]. Courant's work was ignored until engineers had independently developed the torsion problem.

None of the ongoing work was of much practical value at that time because there were no computers available to generate and solve large sets of algebraic equations. As major advances in digital computers and programming languages, the development of finite element coincided with it. By 1953, engineers were able to write stiffness equations in matrix form and solve with the digital computers [2]. Most of this took place at the Boeing Airplane Company. At the time, a large problem was one with 100 degrees of freedom. Turner suggested that triangular plane stress elements be used to model the skin of a delta wing [3]. Much of this publication was unrecognized because of company policies against publication [4].

The name “finite element method” was given by Clough in 1960 [5]. Because of the practical value of this method, new elements for stress analysis applications were developed. The finite element method was regarded as the solution of a variational problem by minimization of a functional. Thus the finite element method was seen as applicable to all kinds of problems that are in variational form.

Large general purpose finite element computer programs were developed during late 1960s and early 1970s like ANSYS, ASKA, and NASTRAN. Each of this programs can do different types of analysis such as static analysis, dynamic and heat transfer analysis. Today there are hundreds of finite element solvers that are available for different specialized purposes.

Background

In this section, we will consider several works that have had a significant influence on this thesis. This work is an extension of the work done by Littlefield [6], [7], [8], and Kenneth Walls, III, [9], and it provided valuable background information. A new mass lumping scheme, referred to as the *reduced consistent mass method*, is introduced in this work. As such, there are only a few literature resources related to this new method.

Much of this early work led the way for the development of the Eulerian and ALE finite element formulations that we use in this thesis. Many of these approximations like lumped mass approximation and consistent mass approximations, originated from the 1970's when the computers were slow and memory was at premium. Although speed and memory are not as much of an issue today, these approximations are still used to formulate the hydrocodes. As a part of this research, the effect of these approximations are examined systematically for the lumped and consistent mass methods.

Thesis Organization

The layout of this thesis is as follows:

In Chapter 1 we have presented the research motivation for this work as well as an introduction to the topics to be covered.

In Chapter 2, a brief introduction to the mathematical background and notations necessary to develop the ALE form of the conservation equations developed for this work is presented.

In Chapter 3, formulations for the conservation equations used in this work will be developed. This section will also present the finite element formulations of the conservation of mass, momentum, and energy equations that are used in ALEAS.

Chapter 4 introduces the *reduced consistent mass method* and will provide some intermediate options to create a tridiagonal system, and a description of the Thomas algorithm to solve the tridiagonal system.

Chapter 5 will provide an overview of the research code ALEAS, which will be used in this work and was developed by Littlefield in two-dimensions [10]. Then, it is shown how to set up the tridiagonal system in the ALEAS code.

Chapter 6 presents the results of a simple four quadratic element problem. By implementing the new reduced consistent mass method developed in Chapter 4, it was possible to gain a significant improvement in a simple four-element problem.

Chapter 7 concludes this thesis and makes future recommendations of research with regards to higher order elements and linear finite elements.

CHAPTER 2

MATHEMATICAL FOUNDATIONS

Mathematical Background for Continuum Mechanics

Notation

Physical quantities are expressed by mathematical representations in the form of tensors and vectors.

Vectors are made up of unit vectors and scalar coefficients, and they are represented by magnitude and a direction. The velocity vector in a three dimensional Cartesian coordinate system can be expressed by:

$$\mathbf{v} = u\boldsymbol{i} + v\boldsymbol{j} = \sum_{i=1}^{n} v_i \widehat{e_i} \tag{2.1}$$

where i and j are unit vectors, and u and v are scalar quantities for vector notation and $\widehat{e_i}$ is the unit vector and v_i is velocity component in indicial notation where *n* is the number of dimensions of the problem (*n* = 2 for the case hand).

The gradient operator ∇ is frequently used in the conservation equations developed in this work. It is defined as:

$$\nabla = \boldsymbol{i}\frac{\partial}{\partial x} + \boldsymbol{j}\frac{\partial}{\partial y} = \widehat{e_i}\frac{\partial}{\partial x_i} \tag{2.2}$$

The product of ∇ and a scalar quantity ϕ, results in a vector defined as:

$$\nabla\phi = \boldsymbol{i}\frac{\partial \phi}{\partial x} + \boldsymbol{j}\frac{\partial \phi}{\partial y} = \widehat{e_i}\frac{\partial \phi}{\partial x_i} \tag{2.3}$$

The product of ∇ and a vector **v**, is known as the divergence and is defined as:

$$\nabla \cdot \mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = \frac{\partial v_i}{\partial x_i} \tag{2.4}$$

The Cauchy stress, which is the only stress used in this thesis is denoted by σ, the density is ρ, the specific internal energy is e, the traction is $\boldsymbol{t}$, and the body force per unit mass is $\boldsymbol{f}$.

The velocity gradient $\nabla \mathbf{v}$, which can also be denoted as L_{ij}, is divided into a symmetric component, D_{ij}, which is the deformation rate, and a skew component, W_{ij}, which is the spin. This is given by following equations:

$$\begin{aligned} L_{ij} &= \frac{\partial v_i}{\partial x_i} \\ D_{ij} &= \frac{1}{2}\left(L_{ij} + L_{ji}\right) \\ W_{ij} &= \frac{1}{2}\left(L_{ij} - L_{ji}\right) \end{aligned} \tag{2.5}$$

The deformation rate, D_{ij}, is referred to the strain rate, $\dot{\varepsilon_{ij}}$, as is done in Chapter 3.

Conservation Equations

This section contains a brief introduction of the governing equations for momentum, mass, and energy in Eulerian reference frame.

The conservation of mass equation in the Lagrangian computational reference frame is written as:

$$\frac{D\rho}{Dt} + \rho \frac{\partial v_i}{\partial x_i} = 0 \tag{2.6}$$

where ρ is the density.

The conservation of momentum equation in the Lagrangian computational reference frame is given by:

$$\rho \frac{Dv_i}{Dt} = \frac{\partial}{\partial x_j}\sigma_{ji} + \rho f_i \quad \textbf{(2.7)}$$

where σ_{ji} is the Cauchy stress and f_i is the body force per unit mass.

The conservation of energy equation in the Lagrangian computational reference frame is:

$$\rho \frac{De}{Dt} = \sigma_{ij}\dot{\varepsilon_{ij}} + \rho f_i v_i \quad \textbf{(2.8)}$$

where e is the internal energy and $\dot{\varepsilon_{ij}}$ is the strain rate.

Here the *D/Dt* term is known as the material derivative. This is the time rate of change associated with the material. It is defined as:

$$\frac{D}{Dt}( ) = \frac{\partial}{\partial t}( ) + v_i \frac{\partial}{\partial x_i}( ) \quad \textbf{(2.9)}$$

where the first term on the right-hand side is the local change and the second term is the convective change.

The conservation of mass equation in the Eulerian computational reference frame is given by:

$$\frac{\partial}{\partial x_i}(\rho v_i) + \frac{\partial \rho}{\partial t} = 0 \quad \textbf{(2.10)}$$

The conservation of momentum equation in the Eulerian computational reference frame is given by:

$$\frac{\partial}{\partial t}(\rho v_i) + \frac{\partial}{\partial x_j}(\rho v_i v_j) = \frac{\partial}{\partial x_j}\sigma_{ji} + \rho f_i \quad \textbf{(2.11)}$$

The conservation of energy in the Eulerian computational reference frame is given by:

$$\frac{\partial}{\partial t}(\rho e) + \frac{\partial}{\partial x_j}(\rho e v_j) = \sigma_{ji}\dot{\varepsilon_{ij}} + \rho f_i v_i \quad \textbf{(2.12)}$$

CHAPTER 3

FORMULATIONS

Development of the Eulerian Conservation Equations and Finite Element Formulation

Let X denote the current, coordinate system of a given volume, *V*. We must have to define two additional coordinate systems: the ALE coordinates where **y = y(X,*t*)**, and the Eulerian coordinates, where **x = x(X,*t*).** The ALE and Eulerian coordinates describe the deformation of the volume *V* and deformation of a body of interest Ω respectively as shown in Figure 3.1.

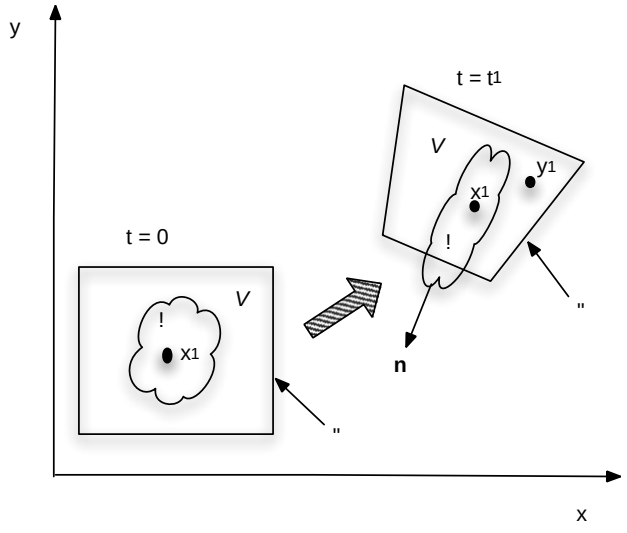

Figure 3.1: Lagrangian, Eulerian, and ALE coordinate systems

The Lagrangian description can be recovered by setting the advection velocity **c** to zero from the ALE equations and the Eulerian description is recovered by setting the advection velocity **c** equal to **v** and the Jacobian, J, equal to one.

In following development section, we use Kenneth Walls III [9] dissertation as a reference to create finite element formulations.

Conservation of Momentum in ALE Coordinates

The conservation of momentum equation is solved by performing one dimensional sweeps in each direction.

The initial momentum, $\mathbf{M}_0$, of the body Ω at t = 0 is given by:

$$\boldsymbol{M}_0 = \int_{\Omega \cap \mathrm{V}} \rho_0 \mathbf{v}_0 \, dX \tag{3.1}$$

Likewise, at some later time $t = t_1$, the momentum of the body, $\mathbf{M}_1$, is given by:

$$\boldsymbol{M}_1 = \int_{\Omega \cap \mathrm{V}} \rho_0 \mathbf{v}_0 \, dX + \int_0^t \int_\Gamma \rho \mathbf{v} (\mathbf{c} \cdot n) \, ds \tag{3.2}$$

where n is the outward unit normal vector to V along its boundary Γ and c is the advection velocity, given by:

$$\mathbf{c} = \mathbf{v} \qquad \text{[Eulerian Description]}$$

The change in momentum is given by $\mathbf{M_1} - \mathbf{M_0}$, and the rate of change is the time derivative, which can be written as:

$$\frac{\partial \boldsymbol{M}}{\partial t} = \frac{\partial \boldsymbol{M_1}}{\partial t} = \frac{\partial}{\partial t} \int_{\Omega \cap \mathrm{V}} \rho \mathbf{v} \, dx + \int_\Gamma \rho \mathbf{v} \, (\mathbf{v} \cdot n) \, ds \tag{3.3}$$

Now, let

$$F_y = \frac{dy}{dX} \tag{3.4}$$

be the second-order tensor describing the deformation of the Eulerian coordinate system. Now, by using this we can transform the first term on the right-hand side of Equation 3.3 into the original coordinate system as follows:

$$\frac{\partial}{\partial t}\int_{\Omega\cap \mathrm{V}} \rho\mathbf{v}\, dx = \frac{\partial}{\partial t}\int_{\Omega\cap \mathrm{V}} \rho\mathbf{v}\det(F_y)\, dX = \frac{\partial}{\partial t}\int_{\Omega\cap \mathrm{V}} \rho\mathbf{v}\, dX \tag{3.5}$$

where J = det($\mathbf{F}_y$), which is equal to one for Eulerian description and is the Jacobian of the reference frame. Then applying Gauss' theorem to the second integral which gives:

$$\int_{\Gamma} \rho\mathbf{v}\,(\mathbf{v}\cdot n)\, ds = \int_{\Omega\cap \mathrm{V}} \nabla\cdot(\rho\mathbf{v}\mathbf{v})\, dx \tag{3.6}$$

Using Cauchy's law the forces acting on the body are given by:

$$\boldsymbol{F} = \int_{\Gamma} \boldsymbol{n}\cdot\sigma\, ds + \int_{\Omega\cap \mathrm{V}} \boldsymbol{f}\, dx \tag{3.7}$$

where **f** is an externally applied force per unit volume and σ is the Cauchy stress tensor.

Using Gauss' Theorem on the first integral in Equation 3.7 gives:

$$\boldsymbol{F} = \int_{\Gamma} \boldsymbol{n}\cdot\sigma\, ds + \int_{\Omega\cap \mathrm{V}} \boldsymbol{f}\, dx = \int_{\Omega\cap \mathrm{V}} [\nabla\cdot\sigma + \boldsymbol{f}]\, dx \tag{3.8}$$

Changing the reference frame of Equation 3.8 gives:

$$\int_{\Omega\cap \mathrm{V}} [\nabla\cdot\sigma + \boldsymbol{f}]\, dy = \int_{\Omega\cap \mathrm{V}} [\nabla\cdot\sigma + \boldsymbol{f}]\, dx \tag{3.9}$$

So, the conservation of momentum equation now becomes:

$$\int_{\Omega\cap \mathrm{V}} \left[\frac{\partial}{\partial t}(\rho\mathbf{v}) + \nabla\cdot(\rho\mathbf{v}\mathbf{v})\right] dx = \int_{\Omega\cap \mathrm{V}} [\nabla\cdot\sigma + \boldsymbol{f}]\, dx \tag{3.10}$$

This must be valid for any choice of control volume *V*, so therefore the integrand must be zero.

$$\left[\frac{\partial}{\partial t}(\rho\mathbf{v}) + \nabla\cdot(\rho\mathbf{v}\mathbf{v})\right] = [\nabla\cdot\sigma + \boldsymbol{f}] \tag{3.11}$$

The first term on the left-hand side can be rewritten as:

$$\frac{\partial}{\partial t}(\rho\mathbf{v}) = \rho\frac{\partial\mathbf{v}}{\partial t} + \mathbf{v}\,\frac{\partial\rho}{\partial t} \tag{3.12}$$

The second term on the left-hand side of Equation 3.11 can be rewritten as:

$$\nabla \cdot (\rho \mathbf{v}\mathbf{v}) = \rho \mathbf{v} \nabla \cdot \mathbf{v} + \mathbf{v} \cdot \nabla(\rho \mathbf{v}) = \rho \mathbf{v} \nabla \cdot \mathbf{v} + \rho \mathbf{v} \nabla \cdot \mathbf{v} + \mathbf{v}\mathbf{v} \cdot \nabla \rho \quad \textbf{(3.13)}$$

Using equations 3.12 and 3.13, Equation 3.11 becomes

$$\rho \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \frac{\partial \rho}{\partial t} + \rho \mathbf{v} \nabla \cdot \mathbf{v} + \rho \mathbf{v} \nabla \cdot \mathbf{v} + \mathbf{v}\mathbf{v} \cdot \nabla \rho = \nabla \cdot \sigma + \boldsymbol{f} \quad \textbf{(3.14)}$$

From the conservation of mass equation shown by Kenneth Walls III [9] that it has $\partial\rho/\partial t + \rho\nabla \cdot \mathbf{v} + \mathbf{v} \cdot \nabla\rho = 0$. So, equation 3.14 becomes

$$\rho \frac{\partial \mathbf{v}}{\partial t} + \rho \mathbf{v} \nabla \cdot \mathbf{v} = \nabla \cdot \sigma + \boldsymbol{f} \quad \textbf{(3.15)}$$

This is the conversation of momentum expressed in the Eulerian coordinate system.

We must express the momentum equation in the weak form. This is done by multiplying by a test function *w* and integrating over the volume *V* which will give:

$$\int_V w \left[\rho \frac{\partial \mathbf{v}}{\partial t} + \rho \mathbf{v} \nabla \cdot \mathbf{v} \right] dx = \int_V w \left[\nabla \cdot \sigma + \boldsymbol{f} \right] dx \quad \textbf{(3.16)}$$

Using Gauss's theorem, the second term on the left-hand side can be integrated by parts to obtain:

$$\int_V w \rho \mathbf{v} \nabla \cdot \mathbf{v}\, dx = -\int_V \nabla w \cdot (\rho \mathbf{v}\mathbf{v})\, dx + \int_\Gamma w \mathbf{v} \rho \mathbf{v} \cdot \mathrm{n}\, \mathrm{ds} \quad \textbf{(3.17)}$$

Likewise, the stress term on the right-hand side can be integrated using Gauss' theorem and Cauchy's Law to give:

$$\int_V w \nabla \cdot \sigma\, dx = -\int_V \nabla w \cdot \sigma\, dx + \int_\Gamma w \mathbf{t}\, \mathrm{ds} \quad \textbf{(3.18)}$$

where **t** is the traction.

So, the weak form of the conservation of momentum equation can be written in Eulerian form as:

$$\int_V \left[\rho \frac{\partial \mathbf{v}}{\partial t} - \nabla w \cdot (\rho \mathbf{v}\mathbf{v}) \right] dx + \int_\Gamma w \mathbf{v} \rho \mathbf{v} \cdot \mathrm{n}\, \mathrm{ds} = \int_V \left[wf - \nabla w \cdot \sigma \right] \mathrm{dx} + \int_\Gamma w \mathbf{t}\, \mathrm{ds} \quad \textbf{(3.19)}$$

For the first operator split this gives us as follows:

$$\int_V \left[\rho \frac{\partial \mathbf{v}}{\partial t} - \nabla w \cdot (\rho \mathbf{v}\mathbf{v})\right] dx + \int_\Gamma w\mathbf{v}\rho\mathbf{v} \cdot n \, ds$$

$$= \int_V [wf - \nabla w \cdot \sigma] \, dx \quad \textbf{(3.20)}$$

Performing the first operator split which gives:

Lagrangian step (no momentum advection)

$$\int_V \left[\rho \frac{\partial v_i^{lag}}{\partial t}\right] dx = \int_V [wf - \nabla w \cdot \sigma] \, dx \quad \textbf{(3.21)}$$

Remap step

$$\int_V \left[w\rho \frac{\partial v_i^{remap}}{\partial t} - \nabla w \cdot \left(\rho v_i^{lag} v\right)\right] dx + \int_\Gamma w v_j \rho v_i^{lag} \cdot n \, ds = 0 \quad \textbf{(3.22)}$$

Finite element approximation of the conservation of momentum equation

The finite element approximation can be developed by replacing the test function w with the shape function N_l in the Lagrangian step and using a piecewise constant function N_k^α which has a value of one for the element. We will now introduce the subscript *m* to indicate the material of interest. The material velocity, **v,** is replaced with following approximations:

$$\mathbf{v}_m = \sum_{j=1}^{n_n} N_j \, \mathbf{v}_{m,j} \quad \textbf{(3.23)}$$

where n_n is the total number of nodes and the subscript m indicates the material of interest. The Cauchy stress, σ, can be decomposed into its components as:

$$\sigma_{ij} = -p^* \delta_{ij} + s_{ij} \quad \textbf{(3.24)}$$

Where δ_{ij} is the Kronecker delta function, p^* is the pressure, and s_{ij} is the deviatoric stress, and the indices i and j have values 1, …, n where n is the number of dimensions of the problem. The pressure term, p^*, is the sum of the pressure, p, determined by the

equation of state, and the artificial viscosity, q, while the six deviatoric stress terms that arise in three-dimensions are found through the constitutive laws. Each of these must be written in a form suitable for the finite element method and are given by:

$$\sigma_m = \sum_{k=1}^{n_e} N_k^{\alpha}\, \sigma_{m,k} \tag{3.25}$$

$$p_m = \sum_{k=1}^{n_e} N_k^{\alpha}\, p_{m,k} \tag{3.26}$$

$$q_m = \sum_{k=1}^{n_e} N_k^{\alpha}\, q_{m,k} \tag{3.27}$$

$$s_m = \sum_{k=1}^{n_e} N_k^{\alpha}\, s_{m,k} \tag{3.28}$$

where n_e is the number of elements. Furthermore, the density term,ρ can be approximated by:

$$\rho_m = \sum_{k=1}^{n_e} N_k^{\alpha}\, \rho_{m,k} \tag{3.29}$$

So for the Lagrangian step the finite element approximation for the conservation of momentum equation is given by:

$$\sum_{k=1}^{n_e} \int_{\Omega_k} N_l\, N_k^{\alpha}\, \rho_{m,k} \left(\sum_{j=1}^{n_n} N_l\, \frac{\partial \mathbf{v}_{m,j}^{lag}}{\partial t} \right) \emptyset_{m,k}\, dx$$

$$= \sum_{k=1}^{n_e} \left(N_l f - \nabla N_l \cdot N_k^{\alpha} \sigma_{m,k} \right) \emptyset_{m,k}\; dx \tag{3.30}$$

$$l = 1,2, \ldots, n_n$$

Here, assumption is made that N_k^{α} is a piecewise constant function to eliminate the sums for σ and ρ in Equations 3.25 and 3.29.

The left-hand side term $\sum_{k=1}^{n_e} \int_{\Omega_k} N_l\, N_k^{\alpha}\, \rho_{m,k}\; \emptyset_{m,k}\, dx$ is known as the consistent mass matrix. It is not diagonalized and results in a high computational cost. Therefore, to

simplify the solution we can diagonalize Equation 3.30 by using the lumped mass, $M_{m,l}^{lag}$ at node *l*, which is defined as:

$$M_{m,l}^{lag} = \sum_{j=1}^{n_n} \sum_{k=1}^{n_e} \int_{\Omega_k} N_l \, N_j \, \rho_{m,k} \;\; \emptyset_{m,k} \, dx \qquad \textbf{(3.31)}$$

Equation 3.31 is usually referred to as the nodal mass.

With the assumption that N_k^{α} is piecewise constant i.e. $N_k^{\alpha} = 1$ on Ω_k and $\emptyset_k = 1$, then the left hand side of equation 3.30 can be rewritten as:

$$\sum_{j=1}^{n_n} \sum_{k=1}^{n_e} \int_{\Omega_k} N_l \, N_j \, \rho_k \;\; \frac{\partial \mathbf{v}_{k,j}^{lag}}{\partial t} \, dx \qquad \textbf{(3.32)}$$

$$l = 1,2, \ldots, n_n$$

Conservation of Mass in ALE Coordinates

We also need to express the conservation of mass in the Eulerian coordinate system using the same description as the conservation of momentum. The conservation of mass equation solves for the mass and, by extension, the density, ρ. At $t = 0$, the mass of the body is given by:

$$\boldsymbol{m}_0 = \int_{\Omega \cap \mathrm{V}} \rho_0 \, dX \qquad \textbf{(3.33)}$$

Likewise, the mass at some later time, $t = t_1$, can be written as:

$$\boldsymbol{m}_1 = \int_{\Omega \cap \mathrm{V}} \rho_0 \, dX + \int_0^t \int_{\Gamma} \rho \mathbf{c} \cdot \boldsymbol{n} \, dsdt \qquad \textbf{(3.34)}$$

where n is the outward unit normal vector to V along its boundary Γ and **c** is the advection velocity, given by:

$$\mathbf{c} = \mathbf{v} \qquad \text{[Eulerian Description]}$$

We can now express the conservation of mass as:

$$\int_V \rho_0 \, dX = \int_V \rho \, dX + \int_0^t \int_\Gamma \rho \mathbf{v} \cdot \mathbf{n} \, ds dt \tag{3.35}$$

Here we have replaced $\Omega \cap V$ with V by setting $\rho = 0$ in V - $\Omega \cap V$. Substituting this, changing reference frames, and differentiating with respect to time, we get:

$$\int_V \left[\frac{\partial}{\partial t}(\rho)\right] dx + \int_\Gamma \rho \mathbf{v} \cdot \mathbf{n} \, ds = 0 \tag{3.36}$$

Now using Gauss's theorem, we find the second integral to be:

$$\int_\Gamma \rho \mathbf{v} \cdot \mathbf{n} \, ds = \int_V \nabla \cdot (\rho \mathbf{v}) \, dy = \int_V \nabla \cdot (\rho \mathbf{v}) \, dx \tag{3.37}$$

So, by substituting this, the conservation of mass equation can now be rewritten as:

$$\int_V \left[\frac{\partial}{\partial t}(\rho) + \nabla \cdot (\rho \mathbf{v})\right] dx = 0 \tag{3.38}$$

Since this must apply for any choice of control volume V, we can conclude that the integrand must be zero, so using this and dividing through by J which is equal to one (Eulerian description) gives:

$$\frac{\partial}{\partial t}(\rho) + \nabla \cdot (\rho \mathbf{v}) = 0 \tag{3.39}$$

Using the product rule, the first term on the left-hand side gives:

$$\frac{\partial}{\partial t}(\rho) = \left[\frac{\partial \rho}{\partial t} + \rho \frac{\partial J}{\partial t}\right] = \frac{\partial \rho}{\partial t} + \rho \frac{\partial J}{\partial t} \tag{3.40}$$

Equation 3.40 can be rewritten:

$$\frac{\partial \rho}{\partial t} + \rho \frac{\partial J}{\partial t} = \frac{\partial \rho}{\partial t} + \rho \nabla \cdot \dot{y} \tag{3.41}$$

The second term on the left-hand side of Equation 3.39 can be rewritten as:

$$\nabla \cdot (\rho \mathbf{v}) = \rho \nabla \cdot \mathbf{v} + \mathbf{v} \cdot \nabla \rho = \rho \nabla \cdot (\mathbf{v} - \dot{y}) + \mathbf{v} \cdot \nabla \rho \tag{3.42}$$

Substituting Equations 3.41 and 3.42 into Equation 3.39 and canceling terms gives:

$$\frac{\partial \rho}{\partial t} + \rho \nabla \cdot \mathbf{v} + \mathbf{v} \cdot \nabla \rho = 0 \tag{3.43}$$

This is mass conservation expressed in the Eulerian coordinate system.

We must now express the conservation of mass equation in a weak form in order to develop the finite element formulation. This is accomplished by multiplying the governing differential equation by a test function, *w*, and integrating over the volume, *V*, to get:

$$\int_{\mathrm{V}} w \left[\frac{\partial \rho}{\partial t} + \rho \nabla \cdot \mathbf{v} + \mathbf{v} \cdot \nabla \rho\right] dx = 0 \qquad \textbf{(3.44)}$$

Using the Reynolds transport theorem for the first term on the left-hand side gives:

$$\int_{\mathrm{V}} w \frac{\partial \rho}{\partial t} dx = \frac{\partial}{\partial t} \int_{\mathrm{V}} w\rho \, dx - \int_{\Gamma} w\rho \dot{y} \cdot n \, ds \qquad \textbf{(3.45)}$$

The second term can be integrated by parts to obtain:

$$\begin{aligned} \int_{\mathrm{V}} w\rho \nabla \cdot \mathbf{v} \, dx &= -\int_{\mathrm{V}} \nabla w \cdot (\rho \mathbf{v}) dx + \int_{V} \nabla \rho \cdot (\mathrm{w}\mathbf{v}) \, dx \\ &= -\int_{\mathrm{V}} \nabla w \cdot (\rho \mathbf{v}) dx + \int_{\Gamma} \mathrm{w}\rho \mathbf{v} \cdot \mathbf{n} \, ds \\ &= -\int_{\mathrm{V}} \nabla w \cdot (\rho \mathbf{v}) dx + \int_{\Gamma} \mathrm{w}\rho(\mathbf{v} + \dot{y}) \cdot n \, ds \end{aligned} \qquad \textbf{(3.46)}$$

So, after canceling terms the statement of the weak form of the conservation of mass equation in the Eulerian coordinate system becomes:

$$\frac{\partial}{\partial t} \int_{\mathrm{V}} w\rho \, dx + \int_{\mathrm{V}} [w\mathbf{v} \cdot \nabla \rho - \nabla w \cdot (\rho \mathbf{v})] \, dx + \int_{\Gamma} \mathrm{w}\rho \mathbf{v} \cdot \mathbf{n} \, ds = 0 \qquad \textbf{(3.47)}$$

Performing the first operator split we arrive at the following:

Lagrangian step

$$\frac{\partial}{\partial t} \int_{\mathrm{V}} w\rho^{lag} \, dx = 0 \qquad \textbf{(3.48)}$$

Remap step

$$\frac{\partial}{\partial t} \int_{\mathrm{V}} w\rho^{remap} \, dx + \int_{\mathrm{V}} [w\mathbf{v} \cdot \nabla \rho^{lag} - \nabla w \cdot (\rho^{lag} \mathbf{v})] \, dx + \int_{\Gamma} \mathrm{w}\rho^{lag} \mathbf{v} \cdot \mathbf{n} \, ds = 0$$

(3.49)

Finite element approximation of the conservation of mass equation

In order to develop the finite element approximation of the conservation of mass equation we replace the test function *w* with a piecewise constant N_k^α which is equal to one for element *k*. Likewise, the density, and velocity are approximated using Equations 3.29, and 3.23, respectively.

For the Lagrangian step, the finite element formulation for the conservation of mass equation is given by:

$$\frac{\partial}{\partial t}\int_{\Omega_k} N_k^\alpha \left(N_k^\alpha \rho_{m,k}^{lag}\right) \emptyset_{m,k}\, dx = 0 \qquad \textbf{(3.50)}$$

$$k = 1, 2, \ldots, n_e$$

Since $N_k^\alpha = 1$ for element *k* Equation 3.50 becomes:

$$\frac{\partial}{\partial t}\int_{\Omega_k} \left(\rho_{m,k}^{lag}\right) \emptyset_{m,k}\, dx = 0 \qquad \textbf{(3.51)}$$

$$k = 1, 2, \ldots, n_e$$

The integral $\int_{\Omega_k} \left(\rho_{m,k}^{lag}\right) \emptyset_{m,k}\, dx$ is equal to the element mass $m_{m,k}^{lag}$, and can be written as:

$$\frac{\partial}{\partial t} m_{m,k}^{lag} = 0 \qquad \textbf{(3.52)}$$

$$k = 1, 2, \ldots, n_e$$

This shows that the mass is constant in element *k* during the Lagrangian step, therefore it is not necessary to carry out the finite element approximation for conservation of mass in the Lagrangian step.

Conservation of Energy in ALE Coordinates

The initial total energy, E_0, of a body Ω at $t = 0$ is given by:

$$\boldsymbol{E}_0 = \int_{\Omega \cap V} \rho_0 \, E_0 \, dX \tag{3.53}$$

Likewise, at some later time $t = t_1$, the total energy of the body, E_1, is given by:

$$\boldsymbol{E}_1 = \int_{\Omega \cap V} \rho E \, dX + \int_0^t \int_\Gamma \rho \boldsymbol{E} \, (\mathbf{v} \cdot \boldsymbol{n}) \, ds \tag{3.54}$$

The change in total energy is given by $E_1 - E_0$, and the rate of change is the time derivative, which can be written as:

$$\frac{\partial \boldsymbol{E}}{\partial t} = \frac{\partial \boldsymbol{E_1}}{\partial t} = \frac{\partial}{\partial t} \int_{\Omega \cap V} \rho E \, dx + \int_\Gamma \rho E \, (\mathbf{v} \cdot \boldsymbol{n}) \, ds \tag{3.55}$$

Using Equation 3.4 we can convert the first term on the left-hand side of Equation 3.55 to the original coordinate system as:

$$\frac{\partial}{\partial t} \int_{\Omega \cap V} \rho E \, dx = \frac{\partial}{\partial t} \int_{\Omega \cap V} (\rho E) \, dX = \int_{\Omega \cap V} \frac{\partial}{\partial t} (\rho E) \, dX \tag{3.56}$$

Applying Gauss' theorem to the second term in Equation 3.77 gives:

$$\int_\Gamma \rho E \, (\mathbf{v} \cdot \boldsymbol{n}) \, ds = \int_{\Omega \cap V} \nabla \cdot (\rho E \mathbf{v}) \, dy = \int_{\Omega \cap V} \nabla \cdot (\rho E \mathbf{v}) \, dX \tag{3.57}$$

The total energy is the sum of the work done by the body, where $\dot{W}$ is the rate of mechanical work and $\dot{Q}$ is the rate of energy supplied by heat transfer or energy sources. In this work we do not consider the rate of energy supplied by heat transfer or energy sources, so $\dot{Q}$ is assumed to be zero.

The rate of mechanical work is the sum of the work done by external forces and body forces given by:

$$\dot{W} = \int_{\Omega \cap V} \boldsymbol{f} \cdot \mathbf{v} \, dx + \int_\Gamma (\boldsymbol{t} \cdot \mathbf{v}) \, ds \tag{3.58}$$

Using Cauchy's Law on the first integral in Equation 3.58 becomes:

$$\int_{\Omega \cap V} \boldsymbol{f} \cdot \mathbf{v} \, dx + \int_\Gamma (\boldsymbol{t} \cdot \mathbf{v}) \, ds = \int_{\Omega \cap V} \boldsymbol{f} \cdot \mathbf{v} \, dx + \int_\Gamma (\sigma \cdot \boldsymbol{n}) \cdot \mathbf{v} \, ds \tag{3.59}$$

Applying Gauss' theorem to the first integral on the right-hand side of Equation 3.59 gives:

$$\int_{\Omega\cap V} \boldsymbol{f}\cdot\mathbf{v}\,dx + \int_{\Gamma}(\sigma\cdot\boldsymbol{n})\cdot\mathbf{v}\,ds = \int_{\Omega\cap V}\boldsymbol{f}\cdot\mathbf{v}\,dx + \int_{\Omega\cap V}\nabla\cdot(\mathbf{v}\cdot\sigma)\,ds \qquad \textbf{(3.60)}$$

So, the conservation of total energy equation now becomes:

$$\int_{\Omega\cap V}\left[\frac{\partial}{\partial t}(\rho E) + \nabla\cdot(\rho\mathbf{v}E)\right]\,dX = \int_{\Omega\cap V}[\nabla\cdot(\mathbf{v}\cdot\sigma) + \boldsymbol{f}\cdot\mathbf{v}]\,dX \qquad \textbf{(3.61)}$$

This must be valid for any choice of control volume *V*, so therefore the integrand must be zero. Using this and dividing through by *J* which is equal to one for Eulerian description gives:

$$\frac{\partial}{\partial t}(\rho E) + \nabla\cdot(\rho\mathbf{v}E) = \nabla\cdot(\mathbf{v}\cdot\sigma) + \boldsymbol{f}\cdot\mathbf{v} \qquad \textbf{(3.62)}$$

Using the product rule for the first term on the left-hand side gives:

$$\frac{\partial}{\partial t}(\rho E) = \left[\frac{\partial}{\partial t}(\rho E) + \rho E\frac{\partial J}{\partial t}\right] = \frac{\partial}{\partial t}(\rho E) + \rho E\nabla\cdot\dot{y} \qquad \textbf{(3.63)}$$

Substituting Equations 3.63 into Equation 3.62 gives:

$$\frac{\partial}{\partial t}(\rho E) + \rho E\nabla\cdot\dot{y} + \nabla\cdot(\rho E\mathbf{v}) = \nabla\cdot(\mathbf{v}\cdot\sigma) + \boldsymbol{f}\cdot\mathbf{v} \qquad \textbf{(3.64)}$$

The first term on the left-hand side can be rewritten as:

$$\frac{\partial}{\partial t}(\rho E) = \rho\frac{\partial E}{\partial t} + E\frac{\partial\rho}{\partial t} \qquad \textbf{(3.65)}$$

The third term on the left-hand side of Equation 3.64 can be rewritten as:

$$\begin{aligned}\nabla\cdot(\rho E\mathbf{v}) &= \rho E\nabla\cdot\mathbf{v} + \mathbf{v}\cdot\nabla(\rho E)\\ &= \rho E\nabla\cdot(\mathbf{v}-\dot{y}) + \rho\mathbf{v}\nabla\cdot E + \mathbf{v}E\cdot\nabla\rho \qquad \textbf{(3.66)}\end{aligned}$$

Using Equations 3.65 and 3.66 and canceling terms, Equation 3.64 becomes:

$$\rho\frac{\partial E}{\partial t} + E\frac{\partial\rho}{\partial t} + \rho E\nabla\cdot\mathbf{v} + \rho\mathbf{v}\nabla\cdot E + \mathbf{v}E\cdot\nabla\rho = \nabla\cdot(\mathbf{v}\cdot\sigma) + \boldsymbol{f}\cdot\mathbf{v} \qquad \textbf{(3.67)}$$

Using the conservation of mass equation shown in Equation 3.43 we can simplify Equation 3.67 as:

$$\rho \frac{\partial E}{\partial t} + \rho \mathbf{v} \nabla \cdot E = \nabla \cdot (\mathbf{v} \cdot \sigma) + \boldsymbol{f} \cdot \mathbf{v} \quad \textbf{(3.68)}$$

This is one form of the conservation of energy in the Eulerian coordinate system expressed in terms of total energy E. But the total energy E is the sum of the internal energy e and the kinetic energy k, where the kinetic energy is given by:

$$k = \frac{\mathbf{v} \cdot \mathbf{v}}{2}$$

So the total energy is given by:

$$E = e + \frac{\mathbf{v} \cdot \mathbf{v}}{2}$$

Substituting this into Equation 3.68 gives:

$$\rho \frac{\partial}{\partial t}\left(e + \frac{\mathbf{v} \cdot \mathbf{v}}{2}\right) + \rho \mathbf{v} \nabla \cdot \left(e + \frac{\mathbf{v} \cdot \mathbf{v}}{2}\right) = \nabla \cdot (\mathbf{v} \cdot \sigma) + \boldsymbol{f} \cdot \mathbf{v} \quad \textbf{(3.69)}$$

The right-hand side can be rewritten as:

$$\nabla \cdot (\mathbf{v} \cdot \sigma) + \boldsymbol{f} \cdot \mathbf{v} = \sigma : (\nabla \mathbf{v}) + \mathbf{v} \cdot (\nabla \cdot \sigma) + \boldsymbol{f} \cdot \mathbf{v} \quad \textbf{(3.70)}$$

Substituting this and using the product rule on the kinetic energy terms gives:

$$\rho \frac{\partial e}{\partial t} + \rho \mathbf{v} \nabla \cdot e + \rho \mathbf{v} \frac{\partial \mathbf{v}}{\partial t} + \rho \mathbf{v} \mathbf{v} \nabla \cdot \mathbf{v} = \sigma : (\nabla \mathbf{v}) + \mathbf{v} \cdot (\nabla \cdot \sigma) + \boldsymbol{f} \cdot \mathbf{v} \quad \textbf{(3.71)}$$

Using the conservation of momentum equation given in Equation 3.43, this equation simplifies to:

$$\rho \frac{\partial e}{\partial t} + \rho \mathbf{v} \nabla \cdot e = \sigma : (\nabla \mathbf{v}) \quad \textbf{(3.72)}$$

This is another form of the conservation on energy equation, expressed in terms of internal energy.

Taking the weak form of Equation 3.72 gives:

$$\int_V w \left[\rho \frac{\partial e}{\partial t} + \rho \mathbf{v} \nabla \cdot e\right] dx = \int_V w \left[\sigma : (\nabla \mathbf{v})\right] dx \quad \textbf{(3.73)}$$

Using Gauss's theorem, the second term on the left-hand side can be integrated by parts to obtain:

$$\int_V [w\,\rho\mathbf{v}\nabla\cdot e]\,dx = -\int_V \nabla w\cdot(\rho e\mathbf{v})\,dx + \int_\Gamma we\rho\mathbf{v}\cdot\boldsymbol{n}\,ds \qquad \textbf{(3.74)}$$

So, the weak form of the conservation of energy equation can be written in ALE form as:

$$\int_V \left[w\rho\frac{\partial e}{\partial t} - \nabla w\cdot(\rho e\mathbf{v})\right] dx + \int_\Gamma we\rho\mathbf{v}\cdot\boldsymbol{n}\,ds = \int_V w\,[\sigma:(\nabla\mathbf{v})]\,dx \qquad \textbf{(3.75)}$$

This is the form of the energy equation used in ALEAS. It should be noted that the traction is implied in the rate of work, and thus does not appear in this equation. Performing the first operator split we arrive at the following:

Lagrangian step

$$\int_V \left[w\rho\frac{\partial}{\partial t}e^{lag}\right] dx = \int_V w\,[\sigma:(\nabla\mathbf{v})]\,dx \qquad \textbf{(3.76)}$$

Remap step

$$\int_V \left[w\rho\frac{\partial}{\partial t}e^{remap} - \nabla w\cdot(\rho\mathbf{v}e^{lag})\right] dx + \int_\Gamma we^{lag}\rho\mathbf{v}\cdot\boldsymbol{n}\,ds = 0 \qquad \textbf{(3.77)}$$

Finite element approximation of the conservation of energy equation

The test function w is replaced by the shape function, a piecewise constant function N_k^α, which has a value of one for element *k*. The approximations for **v,** ρ and σ are the same as those given in the derivation of the mass and momentum equations. The specific internal energy term is given by:

$$e_m = \sum_{k=1}^{n_e} N_k^{\alpha}\, e_{m,k} \tag{3.78}$$

Using this in the Lagrangian step of the conservation of energy equation we have:

$$\sum_{k=1}^{n_e}\left[\int_{\Omega_m} N_k^{\alpha}\, \rho_{m,k}\, \frac{\partial}{\partial t}\left(e_{m,k}^{lag}\right)\emptyset_{m,k}\, dx\right]$$

$$= \sum_{k=1}^{n_e}\left[\int_{\Omega_m} N_k^{\alpha}\, \left[\sigma_{m,k} : \nabla\left(\sum_{j=1}^{n_n} N_j\, \mathbf{v}_{m,j}\right)\right]\emptyset_{m,k}\, dx\right] \tag{3.79}$$

where we have made use of the fact that N_k^{α} is a piecewise constant function to eliminate the sums for e, ρ, and σ. Since $N_k^{\alpha} = 1$ and $\nabla\mathbf{v} = \partial v_i / \partial x_j = D_{ij} + W_{ij}$ and also $\sigma_{ij}W_{ij} = 0$.Therefore, we can rewrite Equation 3.79 as:

$$\sum_{k=1}^{n_e}\left[\int_{\Omega_k} \rho_{m,k}\, \frac{\partial}{\partial t}\left(e_{m,k}^{lag}\right)\emptyset_{m,k}\, dx\right]$$

$$= \sum_{k=1}^{n_e}\left[\int_{\Omega_k} \left[\sigma_{m,k} : \dot{\varepsilon_{m,k}}\right]\emptyset_{m,k}\, dx\right] \tag{3.80}$$

The volume integral on the left-hand side, $\sum_{k=1}^{n_e}\int_{\Omega_k} N_l\, N_k^{\alpha}\, \rho_{m,k}\ \emptyset_{m,k}\, dx$ is the element mass $m_{m,k}^{lag}$, so we can rewrite the Lagrangian step of the conservation of energy equation as:

$$m_{m,k}^{lag}\ \frac{\partial}{\partial t}\left(e_{m,k}^{lag}\right) = \int_{\Omega_k} \left[\sigma_{m,k} : \dot{\varepsilon_{m,k}}\right]\emptyset_{m,k}\, dx \tag{3.81}$$

$$k = 1, 2, \ldots, n_e$$

The equations solved for Lagrangian step are summarized below.

Lagrangian Step:

Conservation of Momentum

$$\sum_{k=1}^{n_e}\int_{\Omega_k} N_l\, N_k^{\alpha}\, \rho_{m,k}\, \left(\sum_{j=1}^{n_n} N_l\, \frac{\partial \mathbf{v}_{m,j}^{lag}}{\partial t}\right)\emptyset_{m,k}\, dx$$

$$= \sum_{k=1}^{n_e}\left(N_l f - \nabla N_l \cdot N_k^{\alpha}\sigma_{m,k}\right)\emptyset_{m,k}\ dx \tag{3.82}$$

$$l = 1,2, \ldots, n_n$$

Conservation of Mass

$$\frac{\partial}{\partial t}\int_{\Omega_k} N_k^{\alpha}\left(N_k^{\alpha}\rho_{m,k}^{lag}\right)\,\emptyset_{m,k}\,dx = 0 \qquad \textbf{(3.83)}$$

$$k = 1, 2, \ldots, n_e$$

Conservation of Energy

$$m_{m,k}^{lag}\;\frac{\partial}{\partial t}\left(e_{m,k}^{lag}\right) \;=\; \int_{\Omega_k}\left[\sigma_{m,k} : \dot{\varepsilon_{m,k}}\right]\emptyset_{m,k}\,dx \qquad \textbf{(3.84)}$$

$$k = 1, 2, \ldots, n_e$$

CHAPTER 4

REDUCED CONSISTENT MASS METHOD

Description of Lumped Mass Equation

In this section, we will review the consistent mass approach, lumped mass approach, and some intermediate options to create a tridiagonal and pentadiagonal system, introduced for the first time in this thesis and referred to as the *reduced consistent mass method*. Also we will see a description of the Thomas algorithm to solve the tridiagonal system.

The equation of lumped mass derived in section 3 for the conservation of momentum is defined as follows:

$$\sum_{j=1}^{n_n} \sum_{k=1}^{n_e} \int_{\Omega_k} N_l \, N_j \, \rho_k \, \frac{\partial \mathbf{v}_{k,j}^{lag}}{\partial t} \, dx$$

$$l = 1,2, \ldots, n_n$$

We consider four elements attached to a node 5 and let $j = 5$. Note that $N_5 = 0$, for all elements except those attached to node 5, as shown in figure 4.1.

$$\sum_{k=1}^{4} \sum_{j=1}^{9} \int_{\Omega_k} \left[N_5 \, N_j \, \rho_k \, dx \right] \frac{\partial \mathbf{v}_{k,j}^{lag}}{\partial t} \qquad \textbf{(4.1)}$$

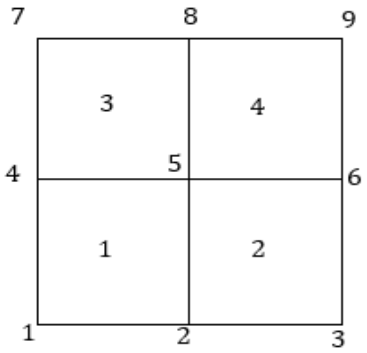

Figure 4.1: Quadratic element

Let $a_j = \frac{\partial \mathbf{v}_{k,j}^{lag}}{\partial t}$, then if we were to write out the terms in the sums (excluding those that are zero) we get

$$\int_{\Omega_1} [N_5 N_1 \rho_1 dx] a_1 + \int_{\Omega_1} [N_5 N_2 \rho_1 dx] a_2 + \int_{\Omega_1} [N_5 N_4 \rho_1 dx] a_4 + \int_{\Omega_1} [N_5 N_5 \rho_1 dx] a_5$$
$$+ \int_{\Omega_2} [N_5 N_2 \rho_2 dx] a_2 + \int_{\Omega_2} [N_5 N_3 \rho_2 dx] a_3 + \int_{\Omega_2} [N_5 N_5 \rho_2 dx] a_5 + \int_{\Omega_2} [N_5 N_6 \rho_2 dx] a_6$$
$$+ \int_{\Omega_3} [N_5 N_4 \rho_3 dx] a_4 + \int_{\Omega_3} [N_5 N_5 \rho_3 dx] a_5 + \int_{\Omega_3} [N_5 N_7 \rho_3 dx] a_7 + \int_{\Omega_3} [N_5 N_8 \rho_3 dx] a_8$$
$$+ \int_{\Omega_4} [N_5 N_5 \rho_4 dx] a_5 + \int_{\Omega_4} [N_5 N_6 \rho_4 dx] a_6 + \int_{\Omega_4} [N_5 N_8 \rho_4 dx] a_8 + \int_{\Omega_4} [N_5 N_9 \rho_4 dx] a_9$$

(4.2)

Collecting terms in equation 4.2, we get as follows:

$$\int_{\Omega_1} [N_5 N_1 \rho_1 dx] a_1 + \int_{\Omega_{1+2}} [N_5 N_2 \rho_{1+2} dx] a_2 + \int_{\Omega_2} [N_5 N_3 \rho_2 dx] a_3 + \int_{\Omega_{1+3}} [N_5 N_4 \rho_{1+3} dx] a_4$$
$$+ \int_{\Omega_{1+2+3+4}} [N_5 N_5 \rho_{1+2+3+4} dx] a_5 + \int_{\Omega_{2+4}} [N_5 N_6 \rho_{2+4} dx] a_6 + \int_{\Omega_3} [N_5 N_7 \rho_3 dx] a_7$$
$$+ \int_{\Omega_{3+4}} [N_5 N_8 \rho_{3+4} dx] a_8 + \int_{\Omega_4} [N_5 N_9 \rho_4 dx] a_9$$

(4.3)

It can be noticed from the above equation that it will produce 9 unknowns (a_1– a_9) for each component but only a single equation for each component.

Letting l = 1, 2, 3,… n successively then it recovers an algebraic system to solve for the unknowns (a_1– a_9). This is the consistent mass approach. Another approach is the so-called *lumped mass* approximation as discussed in the above section. In the example above, we approximate all a_j's with a_5.

$$\left(\int_{\Omega_1} N_5\,(N_1 + N_2 + N_4 + N_5)\,\rho_1 dx + \int_{\Omega_2} N_5\,(N_2 + N_3 + N_5 + N_6)\,\rho_2 dx\right.$$

(=1) (=1)

$$\left. + \int_{\Omega_3} N_5\,(N_4 + N_5 + N_7 + N_8)\,\rho_3 dx + \int_{\Omega_4} N_5\,(N_5 + N_6 + N_8 + N_9)\,\rho_4 dx\right)\, a_5$$

(=1) (=1)

$$= \int_{\Omega_{1+2+3+4}} [\, N_5\ \rho_{1+2+3+4}\, dx\,]\, a_5 \qquad \textbf{(4.4)}$$

The “lumped mass” (simply lumping the masses on the diagonal) is then

$$M_l^{lumped} = \sum_{k=1}^{n_e} N_l\, \rho_k\, dx \qquad \textbf{(4.5)}$$

$$l = 1,2, \ldots, n_n$$

The Reduced Consistent Mass Method

Some intermediate options to create a tridiagonal system are described below.

Tridiagonal 1

In this system, the lumping of masses is done as 1,4,7; 2,5,8, and 3,6,9. This will give an easy-to-solve tridiagonal system.

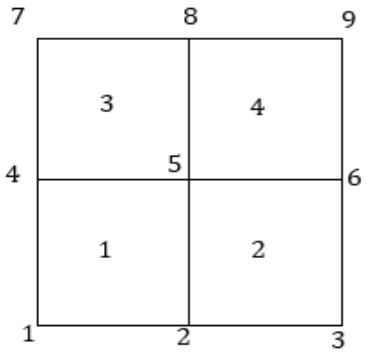

Figure 4.2: Quadratic element with node numbering in x-direction

The equation of lumped mass is written as follows:

$$\left(\int_{\Omega_1} N_5\, N_1\, \rho_1 dx + \int_{\Omega_{1+3}} N_5\, N_4\, \rho_{1+3} dx + \int_{\Omega_3} N_5\, N_7\, \rho_3 dx\right)\, a_4$$
$$+ \left(\int_{\Omega_{1+2}} N_5\, N_2\, \rho_{1+2} dx + \int_{\Omega_{1+2+3+4}} N_5\, N_5\, \rho_{1+2+3+4} dx + \int_{\Omega_{3+4}} N_5\, N_8\, \rho_{3+4} dx\right)\, a_5$$
$$+ \left(\int_{\Omega_2} N_5\, N_3\, \rho_2 dx + \int_{\Omega_{2+4}} N_5\, N_6\, \rho_{2+4} dx + \int_{\Omega_4} N_5\, N_9\, \rho_4 dx\right)\, a_6 \qquad \textbf{(4.6)}$$

Tridiagonal 2

This system is the same as tridiagonal 1, except lumping of masses 1,2,3; 4,5,6 and 7,8,9. Now, if the nodes were re-ordered so that 2, 5 & 8 were numbered sequentially, then this would also be tridiagonal as shown in figure 4.3.

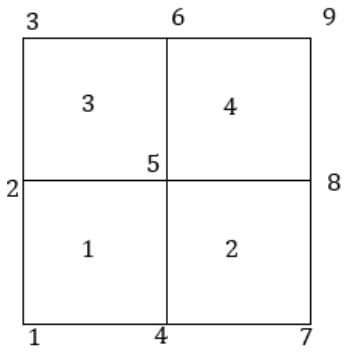

Figure 4.3: Quadratic element with node numbering in y-direction

Pentadiagonal

In this system, lumping of masses is done by lumping 1 with 2 & 4; 3 with 2 & 6; 7 with 4 & 8; 9 with 6 & 8 as shown in figure 4.4. This system is not as easy to solve, but it is faster than the consistent mass approach.

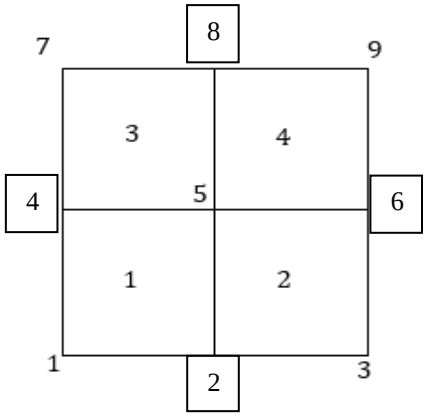

Figure 4.4: Quadratic element used for Pentadiagonal mass lumping

Higher order system:

When elements are higher order, there are many possibilities between lumped mass and consistent mass. It might be possible to lump to 10,11 & 12 then to 7,11 & 15. The next level may be to lump to 9,10,11,12,13 and then to lump to 3,7,11,15 &19 as shown in figure 4.5.

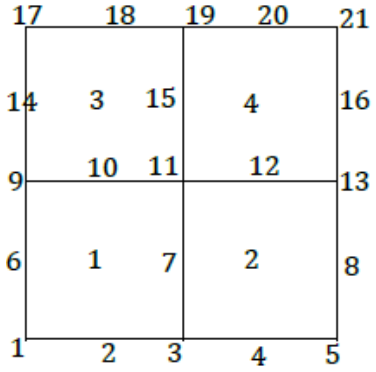

Figure 4.5: Four elements of serendipity family for higher order mass lumping

Thomas algorithm to solve tridiagonal system

As described previously, one intermediate step between lumped mass and consistent mass is to form a tridiagonal system by lumping nodes which are not adjacent (i.e. not sequential in numbering) to the current node. In above example, we lump **1,4,7; 2,5,8**; & **3,6,9**. This is the Tridiagonal 1 case. In this case, we get a matrix of the following form:

$$\begin{pmatrix} b_1 & c_1 & & & \\ a_2 & b_2 & c_2 & & \\ & \cdot & & & \\ & & \cdot & & \\ & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_n & b_n \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ v_{n-1} \\ v_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \cdot \\ \cdot \\ f_{n-1} \\ f_n \end{pmatrix}$$

Forward elimination gives:

$b_2 \longleftarrow b_2 - (c_1/\, b_1)\, a_2$ & $f_2 \longleftarrow f_2 - (f_1/\, b_1)\, a_2$

$b_3 \longleftarrow b_3 - (c_2/\, b_2)\, a_3$ & $f_3 \longleftarrow f_3 - (f_2/\, b_2)\, a_3$ etc.

After the forward elimination step, the system looks like as follows:

$$\begin{pmatrix} b_1 & c_1 & & & \\ & b_2 & c_2 & & \\ & & \cdot & & \\ & & & b_{n-1} & c_{n-1} \\ & & & b_n & c_n \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ v_{n-1} \\ v_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \cdot \\ \cdot \\ f_{n-1} \\ f_n \end{pmatrix}$$

This can be solved by back substitution as follows:

$v_n = f_n / b_n$;

$v_{n-1} = 1/ (v_{n-1}) [f_{n-1} - c_{n-1}\, v_n]$ etc.

A modification to this procedure is needed for the Tridiagonal 2 case (In the previous example, lumping 1,2,3; 4,5,6; & 7,8,9). In the Tridiagonal 1 case, the diagonals are separated by m-1 diagonals of zeros. Here, m would be the number of nodes in the *X*-direction, assuming a logical *ij* mesh. The system looks like as follows:

$$\begin{pmatrix} \overbrace{b_1}^{\text{m-1 zero's}} & c_1 & & & \\ & b_2 & c_2 & & \\ & & \cdot & & \\ & & & b_{n-1} & c_{n-1} \\ & & & \underbrace{b_n}_{\text{m-1 zero's}} & c_n \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \\ \cdot \\ \cdot \\ v_{n-1} \\ v_n \end{pmatrix} = \begin{pmatrix} f_1 \\ f_2 \\ \cdot \\ \cdot \\ f_{n-1} \\ f_n \end{pmatrix}$$

Let's consider the case, m = 3 with 3 × 3 mesh to see if column and row interchanges can produce a tridiagonal system.

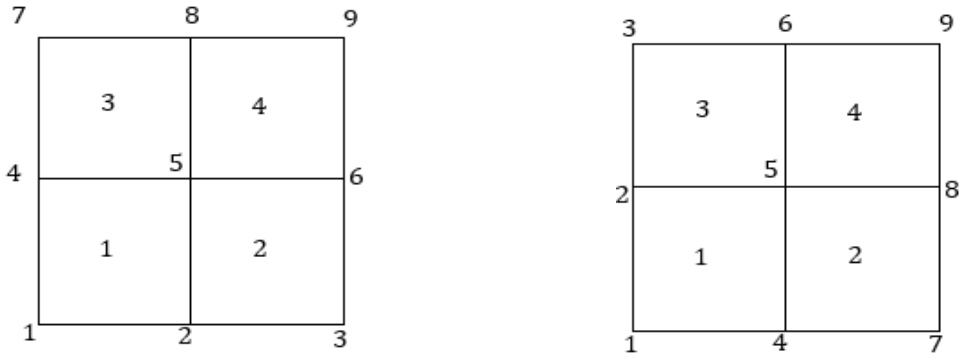

Figure 4.6: Quadratic element (a) Existing ordering system (b) Desire ordering system

So, the system will look like as follows:

$$\begin{pmatrix} b1 & 0 & 0 & c1 & 0 & 0 & 0 & 0 & 0 \\ 0 & b2 & 0 & 0 & c2 & 0 & 0 & 0 & 0 \\ 0 & 0 & b3 & 0 & 0 & c3 & 0 & 0 & 0 \\ a4 & 0 & 0 & b4 & 0 & 0 & c4 & 0 & 0 \\ 0 & a5 & 0 & 0 & b5 & 0 & 0 & c5 & 0 \\ 0 & 0 & a6 & 0 & 0 & b6 & 0 & 0 & c6 \\ 0 & 0 & 0 & a7 & 0 & 0 & b7 & 0 & 0 \\ 0 & 0 & 0 & 0 & a8 & 0 & 0 & b8 & 0 \\ 0 & 0 & 0 & 0 & 0 & a9 & 0 & 0 & b9 \end{pmatrix} \begin{pmatrix} v1 \\ v2 \\ v3 \\ v4 \\ v5 \\ v6 \\ v7 \\ v8 \\ v9 \end{pmatrix} = \begin{pmatrix} f1 \\ f2 \\ f3 \\ f4 \\ f5 \\ f6 \\ f7 \\ f8 \\ f9 \end{pmatrix}$$

Now, interchange both columns and rows 2 & 4:

Changing columns 2 & 4:

$$\begin{pmatrix} b1 & c1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & b2 & c2 & 0 & 0 & 0 & 0 \\ 0 & 0 & b3 & 0 & 0 & c3 & 0 & 0 & 0 \\ a4 & b4 & 0 & 0 & 0 & 0 & c4 & 0 & 0 \\ 0 & 0 & 0 & a5 & b5 & 0 & 0 & c5 & 0 \\ 0 & 0 & a6 & 0 & 0 & b6 & 0 & 0 & c6 \\ 0 & a7 & 0 & 0 & 0 & 0 & b7 & 0 & 0 \\ 0 & 0 & 0 & 0 & a8 & 0 & 0 & b8 & 0 \\ 0 & 0 & 0 & 0 & 0 & a9 & 0 & 0 & b9 \end{pmatrix}$$

Changing rows 2 & 4:

$$\begin{pmatrix} b1 & c1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a4 & b4 & 0 & 0 & 0 & 0 & c4 & 0 & 0 \\ 0 & 0 & b3 & 0 & 0 & c3 & 0 & 0 & 0 \\ 0 & 0 & 0 & b2 & c2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a5 & b5 & 0 & 0 & c5 & 0 \\ 0 & 0 & a6 & 0 & 0 & b6 & 0 & 0 & c6 \\ 0 & a7 & 0 & 0 & 0 & 0 & b7 & 0 & 0 \\ 0 & 0 & 0 & 0 & a8 & 0 & 0 & b8 & 0 \\ 0 & 0 & 0 & 0 & 0 & a9 & 0 & 0 & b9 \end{pmatrix}$$

Now, interchange both columns and rows 3 & 7:

Changing columns 3 & 7:

$$\begin{pmatrix} b1 & c1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a4 & b4 & c4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & c3 & b3 & 0 & 0 \\ 0 & 0 & 0 & b2 & c2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a5 & b5 & 0 & 0 & c5 & 0 \\ 0 & 0 & 0 & 0 & 0 & b6 & a6 & 0 & c6 \\ 0 & a7 & b7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a8 & 0 & 0 & b8 & 0 \\ 0 & 0 & 0 & 0 & 0 & a9 & 0 & 0 & b9 \end{pmatrix}$$

Changing rows 3 & 7:

$$\begin{pmatrix} b1 & c1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a4 & b4 & c4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a7 & b7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & b2 & c2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a5 & b5 & 0 & 0 & c5 & 0 \\ 0 & 0 & 0 & 0 & 0 & b6 & a6 & 0 & c6 \\ 0 & 0 & 0 & 0 & 0 & c3 & b3 & 0 & 0 \\ 0 & 0 & 0 & 0 & a8 & 0 & 0 & b8 & 0 \\ 0 & 0 & 0 & 0 & 0 & a9 & 0 & 0 & b9 \end{pmatrix}$$

Now, interchange both columns and rows 6 & 8:

Changing columns 6 & 8:

$$\begin{pmatrix} b1 & c1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a4 & b4 & c4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a7 & b7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & b2 & c2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a5 & b5 & c5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a6 & b6 & c6 \\ 0 & 0 & 0 & 0 & 0 & 0 & b3 & c3 & 0 \\ 0 & 0 & 0 & 0 & a8 & b8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a9 & b9 \end{pmatrix}$$

Changing rows 6 & 8:

$$\begin{pmatrix} b1 & c1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ a4 & b4 & c4 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a7 & b7 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & b2 & c2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a5 & b5 & c5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & a8 & b8 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & b3 & c3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & a6 & b6 & c6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a9 & b9 \end{pmatrix}$$

Here, the final system we get is tridiagonal. So, in the Thomas algorithm if we first define an integer array giving the needed row/column swaps - i.e. in the example above ipvt (1) = 1; ipvt (2) = 4; ipvt (4) = 2; ipvt (5) = 5; ipvt (6) = 8; ipvt (7) = 3; ipvt (8) = 6; ipvt (9) = 9. The indexes on a_n refer to row *i* and column *i-1*. The example above suggests that we simply replace *i* with ipvt (*i*). Same is true for *c* & *f*. The solution we get for v will be pivoted as well i.e.

$v_{1,solved} = v_1$, $v_{2,solved} = v_4$ $v_{3,solved} = v_7$ etc. In general $v_i = v_{ipvt(i),\text{solved}}$.

Summarizing, once the pivot array is defined, the Thomas algorithm becomes as follows:

Forward elimination

$$b_{ipvt(i)} \longleftarrow b_{ipvt(i)} - \left[\left(c_{ipvt(i-1)} / b_{ipvt(i-1)}\right) a_{ipvt(i)}\right]$$

$$f_{ipvt(i)} \longleftarrow f_{ipvt(i)} - \left[\left(f_{ipvt(i-1)} / b_{ipvt(i-1)}\right) a_{ipvt(i)}\right]$$

where, $i = 2, 3, \ldots, n$

Back substitution

$$v_{ipvt(n)} = f_{ipvt(n)} / b_{ipvt(n)}$$

$$v_{ipvt(i)} = \left(1 / v_{ipvt(i)}\right) \left[f_{ipvt(i)} - c_{ipvt(i)} v_{ipvt(i+1)}\right]$$

where, $i = n\text{-}1, n\text{-}2, \ldots, 2, 1$.

CHAPTER 5

SOFTWARE AND SETUP

ALEAS

ALEAS (Arbitrary Lagrangian-Eulerian Adaptive Solver) is a finite element research code. It was developed in 2D by Dr. David Littlefield [10], and it is used for this work to develop the *Reduced Consistent Mass Method*. ALEAS is capable of performing multi-material calculations in Lagrangian, ALE, Eulerian, and multi-material ALE frameworks. The conservation equations presented in chapter 3 are implemented in ALEAS. Note that in the ALEAS input file, we are only using ijob = 0, which is for the Lagrangian step only, in this work and according to that, all subroutines are edited.

Setup for virtual node numbering

We have to create a new node numbering which we refer to as “Virtual Node Numbering” for the reduced consistent mass method. The reason behind the need to develop virtual node numbering is so that we can easily define locations of neighboring nodes, which help us to create a tridiagonal matrix. In this section, we will see how to develop virtual node numbering in ALEAS. To develop virtual node numbering, first, we have to edit the easy_mesh and mesh_setup subroutines in the mesh source directory of ALEAS.

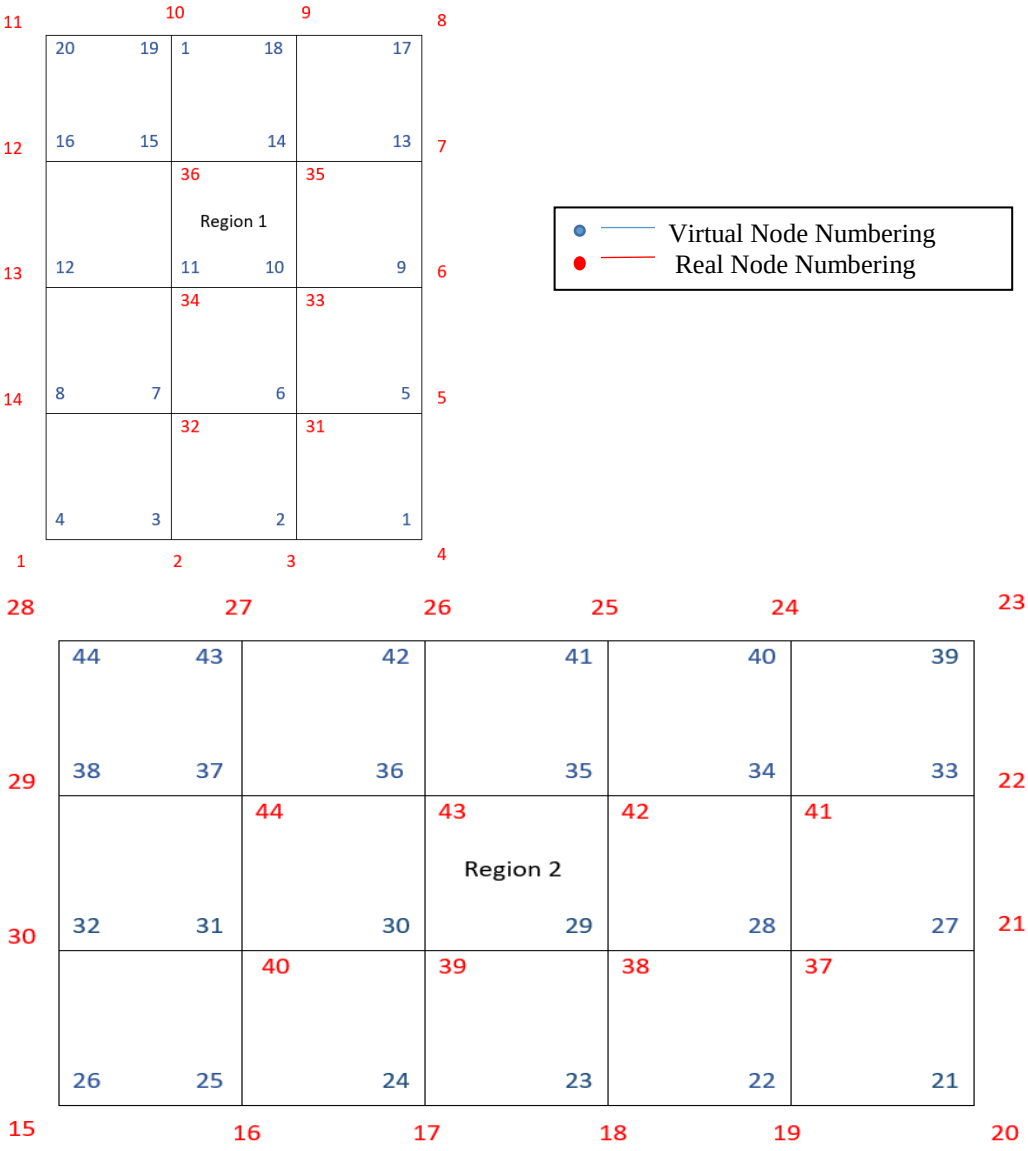

Figure 5.1: Two different regions with virtual and real node numbering

The real node numbering and virtual node numbering are shown in figure 5.1. The blue node numbering is the virtual node numbering, and the red node numbering is the real node numbering. Note that once we define virtual node numbering in a few subroutines, we must also reassign this numbering back to real node numbering in others.

Easy_mesh subroutine

Easy_mesh subroutine inserts a uniform mesh in region ireg. After this routine develops real node numbering, we defined the ipvt array to develop virtual node numbering, as shown in Appendix A. Here, offset2 is used to calculate the offset when virtual node numbering is required for a second material. So, the structure of the ipvt array is defined as follows:

ipvt (ireg, real node) = virtual node

Where ireg = number of regions

Mesh_setup subroutine

This subroutine generates a mesh, region by region, using the paving technique of Blacker and Stephenson. In this subroutine, we develop two single arrays, ipvt4 and ipvt5, as shown in Appendix B. The structure of both arrays are defined as follows:

ipvt4 (real node) = virtual node

ipvt5 (virtual node) = real node

So, ipvt4 returns the virtual node number given the real node number, and ipvt5 returns the real node number given the virtual node number. We can use these arrays in any subroutine to get virtual or real node numbering.

Setup for reduced consistent mass matrix

In this section, we will see how to diagonalize the consistent mass matrix according to the new reduced consistent mass method, and which subroutines need to be edited before stiff_explicit is called in main aleas2dc file.

Zeroed subroutine

This subroutine zeroes out various values. So, before we start updating values in reduced consistent mass matrix i.e., in the stiff_explicit subroutine, we must zero out all those values. So, this array is edited, as shown in Appendix C.

a(1,i) = 0

a(2,i) = 0

a(3,i) = 0

Bnd_explicit subroutine

This routine implements the boundary conditions into the stiffness matrix and right-hand-side vector. So, before we call the stiff_explicit and rhs_explicit routines, icol needs to refer to virtual node numbering, as shown in Appendix D.

inode = ipvt4(ixv(i)), inode = ipvt4(iyv(i))

inode = ipvt4(iemv(i)), inode = ipvt4(iemz(i))

Stiff_explicit subroutine

This subroutine generates the reduced consistent mass matrix by looping through the elements and update the values of the lumped mass matrix. First, to develop reduced consistent mass matrix, we have to define the value of i1max for each region. i1max is the number of nodes on the boundary in the x-direction. To get the value of i1max for each region, we defined an array before we call x and y locations, as shown in Appendix E. So, i1max is defined as follows:

i1max = nftbndry(x-direction)

According to the new reduced consistent mass scheme, we have to split the integrand in three sections, referred to here as Left integrand, Center integrand, and Right integrand, to make a tridiagonal matrix. For that, we have to define the location of neighboring nodes. Considering the center node, the neighboring nodes will automatically fall under Right or Left integrand. Now, there are two cycles, icycle = 0 and icycle = 1 as shown in figure 5.2 and figure 5.3.

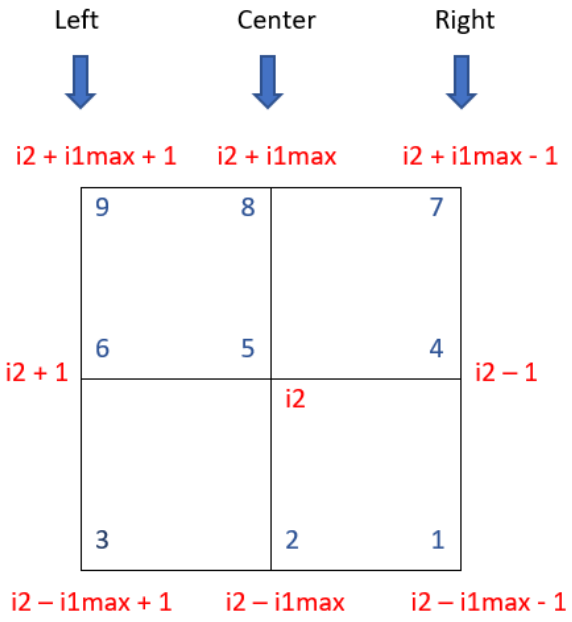

Figure 5.2: Quadratic element when icycle equals to zero

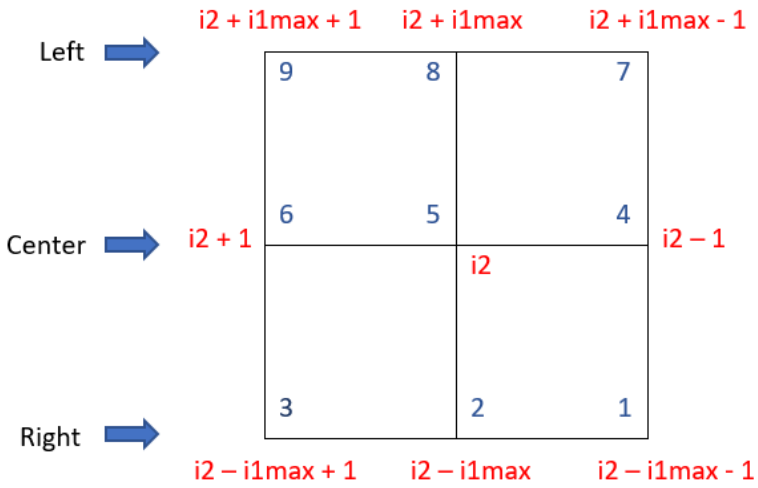

Figure 5.3: Quadratic element when icycle equals to one

To understand more in detail, consider an element when icycle equals to zero, as shown in figure 5.2. As described in the above section, the value of i1max = 3 and the center node, i.e. i2, which is equal to 5. So, according to new reduced consistent mass scheme, nodes 2,5,8 will fall under Center integrand, nodes 1,4,7 will fall under Right integrand, and nodes 3,6,9 will fall under Left integrand. This is the same case for icycle equals to one; the only difference is lumping of masses is done in a different direction, as shown in figure 5.3. After assigning all integrands to the nodes, we sum up all integrands for momentum, mass, and energy, as shown in Appendix D.

Rhs_explicit subroutine

This subroutine constructs the right-hand side for explicit calculations. Since we are using virtual node numbering, the irow in a right-hand side also needs to refer to virtual node numbering. So, instead of using real node in the irow equation, we must use virtual node numbering as shown in Appendix F.

$$irow = imax * mmat * (ipvt4(jnode) - 1) + imax * (imat-1) + 1$$

Here, ipvt4 refers to the virtual node numbering as described in the above section.

Solve_explicit subroutine

This subroutine solves the equations to get the full iteration step. We use the Thomas algorithm to solve the system of equations given the matrix a and vector b, as shown in Appendix G. In this subroutine, we also setup an array which reverses icol from virtual node numbering to real node numbering. Here, we see the ipvt3 array which is already set up in the mesh_setup subroutine and gives virtual node numbering. Instead of using the ipvt3 array, the ipvt4 array could also be used, but ipvt3 is used here for convenience.

Contact routines

This subroutine checks for node penetration and makes the appropriate corrections for sliding surfaces. There are mainly two subroutines, slide4, and update_energy. Those subroutines require editing for both nodal mass and virtual node numbering.

Slide4 subroutine

This routine is a diagonalized version wherein velocities of adjacent master nodes are set equal. As we divided the lumped mass into three different sections, they need to be summed up in this routine with referring to virtual node numbering, as shown in Appendix H.

Mass b = a(1,imax ∗ ipvt4(node) -1) + a(2,imax ∗ ipvt4(node) -1)

+ a(3,imax ∗ ipvt4(node) -1)

Update_energy subroutine

This subroutine updates the energy as a result of contact forces. So, the icol needs to refer to virtual node numbering and also the lumped mass needs to be summed up, as shown in Appendix I.

icol = ioff ∗ ipvt4(node)

fv = fv + (vxc(1,node) ∗ vx(1,node) + vyc(1,node) ∗ vy(1,node))

* mass(0,i) / (ne ∗ (a(1,icol) + a(2,icol) + a(3,icol)))

CHAPTER 6

DISCUSSION AND RESULTS

Results of Simple Quadratic Four Elements

In this section, results from a simple four quadratic element test problem are presented and discussed. The reason behind presenting this simple problem is that some additional debugging in the ALEAS code will be required to run more realistic examples. Considering this simple quadratic four element problem, the codes gives correct results up to the 35th cycle, and after that, it gives values five times larger than expected. This is the same case for every test and verification we did after modifying the code.

We discovered a work-around for this error by setting pres(imat,i) equal to zero, as it should be for this problem, and is as shown in Appendix F. With that change, the code will give all the correct values for each cycle, but this is not the correct way to specify the pressure. We are forcefully defining a value of pressure equals to zero in rhs_explicit, and that is why we are getting correct values for each cycle, but in actual conditions, the pressure in each cycle should be set equal to the value returned from the equation of state.

The results of a simple four quadratic elements with a constant velocity of 100 m/s in the y-direction and zero velocity in the x-direction and with no contact with another material are presented below.

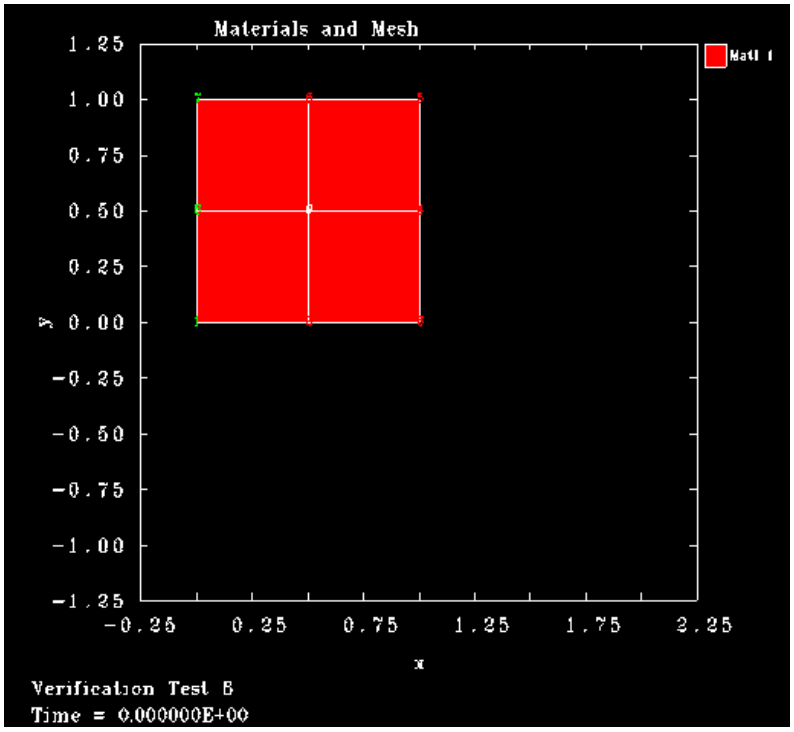

Figure 6.1: Representation of mesh and material of four quadratic element at t = 0

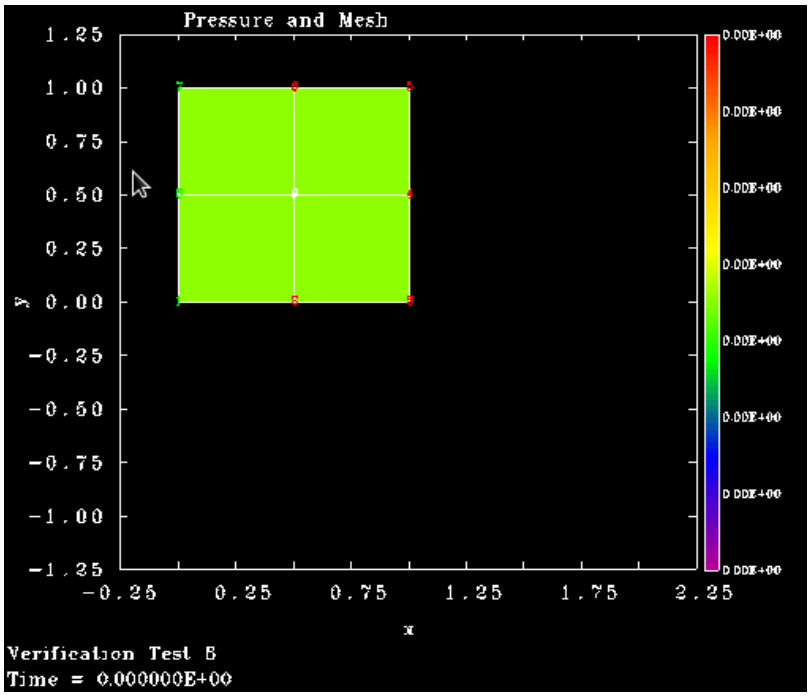

Figure 6.2: Representation of pressure and mesh of four quadratic element at t = 0

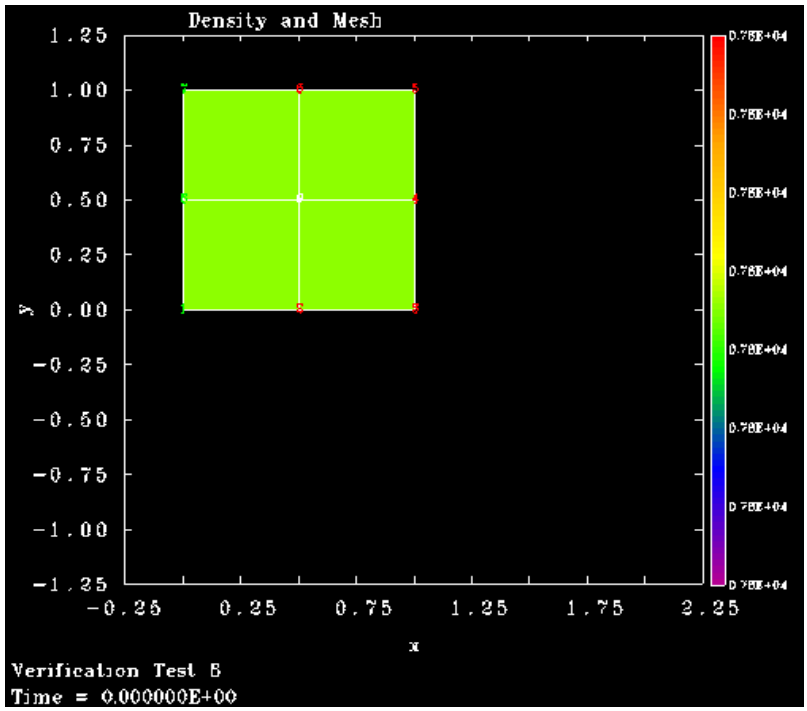

Figure 6.3: Representation of density and mesh of four quadratic element at t = 0

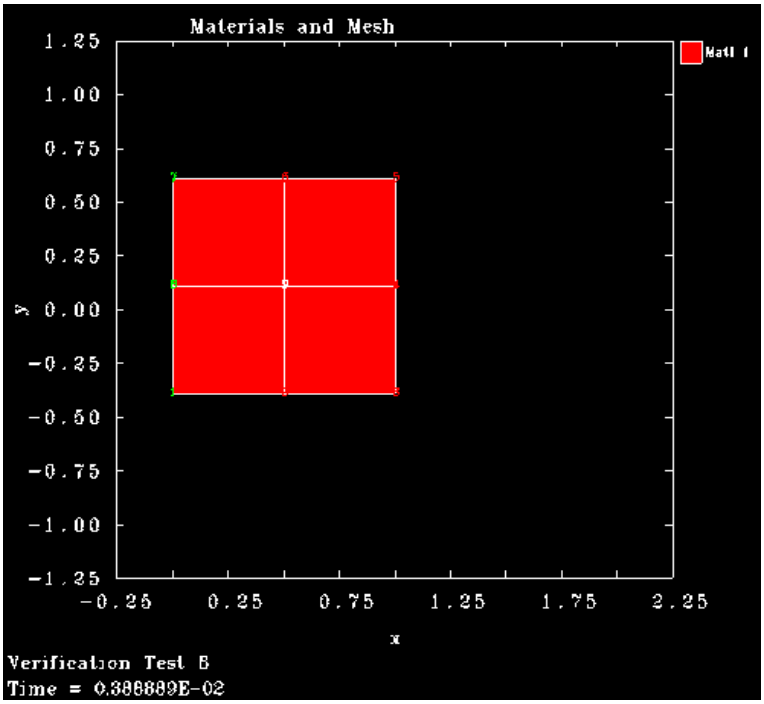

Figure 6.4: Representation of mesh and material of four quadratic element at t = 0.38E-02

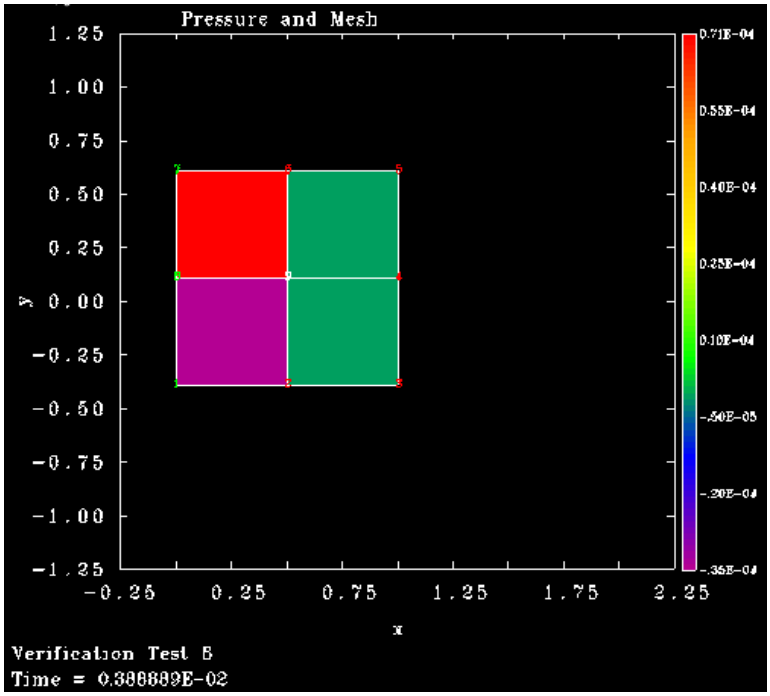

Figure 6.5: Representation of pressure and mesh of four quadratic element at t = 0.38E-02

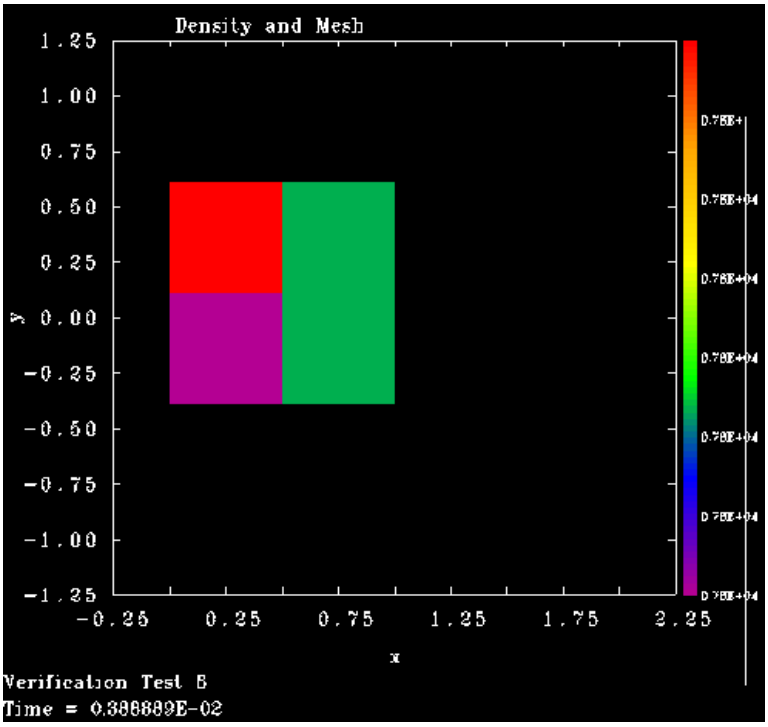

Figure 6.6: Representation of density and mesh of four quadratic elements at t= 0.38E-02

The results obtained for a simple four quadratic element problem are not as accurate as we expected from this new reduced consistent mass method. It does not mean that the new reduced consistent mass method is flawed. The tests are inconclusive at this point but suggest the method has promise.

CHAPTER 7

CONCLUSIONS

In this work we have presented a new computational method for improving the accuracy of the lumped mass approximation, referred to as the *reduced consistent mass method*. The Eulerian form of the conservation equations have been solved by using Finite Element Approximations. The new reduced consistent mass method was successfully implemented in the ALEAS code. There are several levels on which the lumped mass approximation was generalized, including the Tridiagonal 1 and Tridiagonal 2 approximation as described in Section 4, the pentadiagonal approximation, culminating with the consistent mass formulation where no approximation were used. Those are studied theoretically, but it requires more detailed study to implement into ALEAS. While a work-around was developed to run a simple test case of the new method, more work remains to fully debug the code. Because of that, we did not get the results as expected from the new method. It does not mean that the new method is wrong; it requires more study in future work.

Future Work

This research intended to demonstrate a tremendous improvement in accuracy when using a new reduced consistent mass method, but it still requires more work to fix the code. Some possible areas for future work include the following list.

- ALEAS needs to be debugged in more detail to find a small bug which will give accurate results as expected from theoretical calculations.
- Linear finite elements have been traditionally used in hydrocodes. The limitations in accuracy for these elements are well known. For example, linear elements cannot accurately represent curvilinear surfaces and result in inaccurate approximation to the volume of geometries. Linear elements also admit spurious zero-energy modes of deformation (called hourglass modes) and require artificial stabilization methods to be used successfully. Higher-order elements do not suffer from many of these deficiencies. So in the future, it will be great to quantify the improvements in accuracy afforded with the use of higher-order finite elements.
- To investigate the effect of element order on accuracy in Eulerian hydrocodes, linear, serendipity quadratic, Lagrange biquadratic, serendipity cubic, and Lagrange bicubic elements should be compared.
- As we developed a tridiagonal matrix for the new reduced consistent mass method, in the same way, it is possible to develop a new pentadiagonal matrix to further enhance the accuracy of Eulerian hydrocodes.

REFERENCES

[1] R. Courant, "Variational Methods for the Solution of Problems of Equilibrium and Vibrations," *Bulletin of the American Mathematical Society,* Vol. 49, 1943, pp.1-23.

[2] S. Levy, "Structural Analysis and Influence Coefficient for Delta Wings," *J. Aero. Sci., Vol.* 20 7, 1953, pp. 449-454.

[3] R. W. Clough, "The Finite Element Method After Twenty-Five Years: Personal View," *Computers & Structures,* Vol. 12, No. 4, 1980, pp. 361-370.

[4] J. Robinson, Early FEM Pioneers, Robinson & Associates, Dorest, England, 1985.

[5] Clough, R. (1960). The finite element method in plane stress analysis. In *2nd Conference in Electronic Computation*, pages 345-378. American Society of Civil Engineers, Pittsburg, PA.

[6] Littlefield, D. (2002). A Method for Treatment of Dynamic Contact-Impact in Multi-material Frameworks. *Fifth World Congress on Computational Mechanics*, Vienna, Austria, July 7-12, 2002.

[7] Littlefield, D. (2006). Modeling Contact in Multi-material Frameworks. *Seventh World Congress on Computational Mechanics*. Los Angeles, CA, July 16-22, 2006.

[8] Littlefield, D. (2007). Improvements to Multi-material Advection Algorithms in MACH. Final report from Air Force Research Laboratory Contract #FA9451-05-M-0225, Submitted August 2007.

[9] Kenneth C. Walls (2017). An Improved Contact Method for Multi-Material Eulerian Hydrocodes.

[10] Littlefield, D. (2001). The Use of r-adaptivity with Local, Intermittent Remesh for Modeling Hypervelocity Impact and Penetration. *International Journal of Impact Engineering*, 26 (1), 433-442.

APPENDIX A

Easy_mesh.f File

∗ In this file, the only edited part is shown

∗ ipvt array

```
do i=1,maxnodes
  ipvt(ireg,i)=0
enddo
```

∗ Right side boundary nodes

```
do i=1,i2max
 ipvt(ireg,(i-1)*i1max+1)=offset2+i1max-1+i
enddo
```

∗ Bottom side boundary nodes

```
do i=2,i1max
ipvt(ireg,i)=offset2+i1max-i+1
enddo
```

∗ Top side boundary nodes

```
do i=2,i1max-1
ipvt(ireg,(i2max-1)*i1max+i)=offset2+i1max+i2max+i-2
enddo
```

∗ Left side boundary nodes

```
do i=2,i2max
ipvt(ireg,i1max*i)=offset2+2*(i1max+i2max-1)-i
enddo
```

∗ Interior boundary nodes

```
      do iy=2,i2max-1
       do ix=2,i1max-1
         ipvt(ireg,i1max*(iy-1)+ix)=nnodes0+1
         nnodes0=nnodes0+1
       enddo
     enddo
```

```
offset2 = 2*(i1max-1)+2*(i2max-1)+offset2
```

APPENDIX B

Mesh_setup.f File

∗ In this file, the only edited part is shown

∗ ipvt3 array which gives virtual node numbering

```
      offset2=0
      offset5=0
            do ireg=1,nreg
               do j=1,maxnodes
                  if(jnode.eq.0)then
                  offset2=offset2+j-1
                    if(ireg.gt.1)then
                    offset5(ireg)=offset5(ireg-1)+j-1
                    endif
                    goto 999
                  endif
                  ipvt3(ireg,jnode)=j+offset2
               enddo
999            continue
            enddo
```

∗ ipvt4 and ipvt5 array's which gives virtual and real node numbering respectively

```
            do ireg=1,nreg
               nmax=nnreg(ireg)
               do iv=1,nmax
                  nodeR=ipvt(ireg,iv)
                  nodeV=ipvt3(ireg,nodeR)
                  ipvt5(nodeV)=nodeR
                  ipvt4(nodeR)=nodeV
               enddo
            enddo
```

APPENDIX C

Zeroed.f File

∗ In this file, the only edited part is shown

∗ This subroutine zeroes out various quantities

```
      if(jobtype.eq.0)then
        if(lmm)then
        imax=(neql-neqe)*nummat*nnodes+neqe*nummat*nel
        else
        imax=(neql-neqe)*nnodes+neqe*nummat*nel
        endif
        do i=1,imax
            a(1,i)=pzero
            a(2,i)=pzero
            a(3,i)=pzero
            b(i)=pzero
        enddo
      else
        imax=2*nnodes
        jmax=ml+md
       do i=1,imax
            do j=1,jmax
                a(j,i)=pzero
            enddo
            b(i)=pzero
       enddo
```

APPENDIX D

Bnd_explicit.f File

∗ In this file, the only edited part is shown

∗ This subroutine implements the boundary conditions into the stiffness matrix and rhs - vector

```
      if(nxvel.ne.0)then
        do i=1,nxvel
              inode=ipvt4(ixv(i))
              icol=imax*inode-2
              a(1,icol)=pzero
              a(2,icol)=pone
              a(3,icol)=pzero
              b(icol)=xvel(i)
       enddo
      endif

      if(nyvel.ne.0)then
        do i=1,nyvel
              inode=ipvt4(iyv(i))
              icol=imax*inode-1
              a(1,icol)=pzero
              a(2,icol)=pone
              a(3,icol)=pzero
              b(icol)=yvel(i)
        enddo
      endif
      if(nemv.ne.0)then
        do i=1,ntract-1
              x1=ttract(i)
              x2=ttract(i+1)
              if(x2.gt.ptime)then
              y1=ptract(i)
```

```
            y2=ptract(i+1)
            current=y1+(y2-y1)*(ptime-x1)/(x2-x1)
            goto 10
            endif
          enddo

10      fact=mu0*current/ptwo/pi

          do i=1,nemv
            inode=ipvt4(iemv(i))
            icol=imax*inode
            a(1,icol)=pzero
            a(2,icol)=pone
            a(3,icol)=pzero
            b(icol)=fact/xloc(inode)
          enddo
        endif

        if(nemz.ne.0)then
          do i=1,nemz
            inode=ipvt4(iemz(i))
            icol=imax*inode
            a(1,icol)=pzero
            a(2,icol)=pone
            a(3,icol)=pzero
            b(icol)=pzero
          enddo
        endif
```

APPENDIX E

Stiff_explicit.f File

∗ In this file, the only edited part is shown

∗ This subroutine loop through the elements, and update the values of the lumped mass matrix

∗ An array to calculate i1max for each region

```
      ishift=0
        do ibs=1,4
            iseg=nsbnd(ibs+ishift,ibndry)
            ismax=ndfltseg(iseg)
          do is=1,ismax
            node1=iabs(nfltseg(is,iseg))
            nfloat(is,ibs)=node1
          enddo
            isum=ismax
            nftbndry(ibs)=isum
            i1max=nftbndry(1)
        enddo
```

∗ An array to calculate left, center, and right integrands for both icycles

```
      do i1=1,ne1
            i2=ipvt4(jnode)
            i1node=econ(i1,iel)
```

callfcom1(xg(ig,ng),xg(jg,ng),ne,norder,igeom,rtmp,ztmp,rtmp,ztmp,i1,ni,dum,dum,dum,dum,dum,dum,dum,jacob,r,z,1)

```
if(mod(icycle,2).eq.0)then
  if(igeom.eq.1)then
        if(i1node.eq.ipvt5(i2))then
        integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
        integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
        endif

  if(i2+i1max.le.maxnodes)then
        if(i1node.eq.ipvt5(i2+i1max))then
        integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
        integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
        endif
  endif

  if((i2-i1max).gt.0)then
        if(i1node.eq.ipvt5(i2-i1max))then
        integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
        integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*Jacob
        endif
  endif

  if((i2+1).le.maxnodes)then
        if(i1node.eq.ipvt5(i2+1))then
        integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
        integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
        endif
  endif

  if((i2+1+i1max).le.maxnodes)then
        if(i1node.eq.ipvt5(i2+i1max+1))then
        integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
        integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
        endif
  endif

  if((i2+1-i1max).gt.0)then
        if(i1node.eq.ipvt5(i2+1-i1max))then
        integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
        integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
        endif
```

```
        endif

        if((i2-1).gt.0)then
            if(i1node.eq.ipvt5(i2-1))then
            integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
            integranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
            endif
        endif

        if((i2+i1max-1).le.maxnodes)then
            if(i1node.eq.ipvt5(i2+i1max-1))then
            integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
            integranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
            endif
        endif

        if((i2-1-i1max).gt.0)then
            if(i1node.eq.ipvt5(i2-1-i1max))then
            integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
            integranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
            endif
        endif
    else
        if(i1node.eq.ipvt5(i2))then
            integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
            integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
        endif

        if((i2+i1max).le.maxnodes)then
            if(i1node.eq.ipvt5(i2+i1max))then
            integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
            integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
            endif
        endif

        if((i2-i1max).gt.0)then
            if(i1node.eq.ipvt5(i2-i1max))then
            integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
            integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
            endif
```

```
endif

if((i2+1).le.maxnodes)then
      if(i1node.eq.ipvt5(i2+1))then
      integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
      integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
      endif
endif

if((i2+i1max+1).le.maxnodes)then
      if(i1node.eq.ipvt5(i2+i1max+1))then
      integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
      integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
      endif
endif

if((i2+1-i1max).gt.0)then
      if(i1node.eq.ipvt5(i2+1-i1max))then
      integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
      integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
      endif
endif

if((i2-1).gt.0)then
      if(i1node.eq.ipvt5(i2-1))then
      integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
      integranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
      endif
endif

if((i2+i1max-1).le.maxnodes)then
      if(i1node.eq.ipvt5(i2+i1max-1))then
      integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
      integranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
      endif
endif

if((i2-1-i1max).gt.0)then
      if(i1node.eq.ipvt5(i2-1-i1max))then
      integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
```

```
                ntegranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
                endif
            endif
          endif
        endif

if(mod(icycle,2).eq.1)then
        if(igeom.eq.1)then

                if(i1node.eq.ipvt5(i2))then
                integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
                integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
                endif

            if((i2+1).le.maxnodes)then
                if(i1node.eq.ipvt5(i2+1))then
                integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
                integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
                endif
            endif

            if((i2-1).gt.0)then
                if(i1node.eq.ipvt5(i2-1))then
                integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
                integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
                endif
            endif

            if((i2+i1max).le.maxnodes)then
                if(i1node.eq.ipvt5(i2+i1max))then
                integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
                integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
                endif
            endif

            if((i2+i1max+1).le.maxnodes)then
                if(i1node.eq.ipvt5(i2+i1max+1))then
                integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
                integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
                endif
```

```
        endif

        if((i2-1+i1max).le.maxnodes)then
            if(i1node.eq.ipvt5(i2-1+i1max))then
            integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
            integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
            endif
        endif

        if((i2-i1max).gt.0)then
            if(i1node.eq.ipvt5(i2-i1max))then
            integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
            integranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
            endif
        endif

        if((i2-i1max+1).gt.0)then
            if(i1node.eq.ipvt5(i2-i1max+1))then
            integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
            integranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
            endif
        endif

        if((i2-1-i1max).gt.0)then
            if(i1node.eq.ipvt5(i2-1-i1max))then
            integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*r*jacob
            integranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*r*jacob
            endif
        endif
else

            if(i1node.eq.ipvt5(i2))then
            integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
            integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
            endif

        if((i2+1).le.maxnodes)then
            if(i1node.eq.ipvt5(i2+1))then
            integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
            integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*Jacob
```

```
        endif
endif

if((i2-1).gt.0)then
        if(i1node.eq.ipvt5(i2-1))then
        integranC=integranC+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
        integranC1=integranC1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
        endif
endif

if((i2+i1max).le.maxnodes)then
        if(i1node.eq.ipvt5(i2+i1max))then
        integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
        integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
        endif
endif

if((i2+i1max+1).le.maxnodes)then
        if(i1node.eq.ipvt5(i2+i1max+1))then
        integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
        integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
        endif
endif

if((i2-1+i1max).le.maxnodes)then
        if(i1node.eq.ipvt5(i2-1+i1max))then
        integranL=integranL+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
        integranL1=integranL1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
        endif
endif

if((i2-i1max).gt.0)then
        if(i1node.eq.ipvt5(i2-i1max))then
        integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
        integranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
        endif
endif

if((i2-i1max+1).gt.0)then
        if(i1node.eq.ipvt5(i2-i1max+1))then
```

```
                integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
                integranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
                endif
            endif

            if((i2-1-i1max).gt.0)then
                if(i1node.eq.ipvt5(i2-1-i1max))then
                integranR=integranR+wg(ig,ng)*wg(jg,ng)*rho1*ni*nj*jacob
                integranR1=integranR1+wg(ig,ng)*wg(jg,ng)*ni*nj*jacob
                endif
            endif
          endif
        endif

          icol=imax*mmat*(ipvt4(jnode)-1)+imax*(imat-1)+1

          a(1,icol)=a(1,icol)+integranL
          a(2,icol)=a(2,icol)+integranC
          a(3,icol)=a(3,icol)+integranR

          icol=icol+1

          a(1,icol)=a(1,icol)+integranL
          a(2,icol)=a(2,icol)+integranC
          a(3,icol)=a(3,icol)+integranR

          icol=icol+1

          a(1,icol)=a(1,icol)+(pone-phiv(iel))*integranL1
          a(2,icol)=a(2,icol)+(pone-phiv(iel))*integranC1
          a(3,icol)=a(3,icol)+(pone-phiv(iel))*integranR1

          enddo
        enddo
      enddo
    enddo !

          do i=1,nel
                do imat=1,nummat
                icol=ieoff+(i-1)*nummat+imat
```

```
            a(1,icol)=pzero
            a(2,icol)=pone
            a(3,icol)=pzero
            enddo
      enddo
```

APPENDIX F

Rhs_explicit.f File

∗ In this file, the only edited part is shown

∗ This subroutine constructs the right hand side for explicit calculations

```
irow=imax*mmat*(ipvt4(jnode)-1)+imax*(imat-1)+1
b(irow)=b(irow)+vxint
irow=irow+1
b(irow)=b(irow)+vyint
irow=irow+1
b(irow)=b(irow)+bint
```

APPENDIX G

Solve_explicit.f File

∗ In this file, the only edited part is shown

∗ This subroutine solves the equations to get the full iteration step

∗ Thomas Algorithm to solve equations

```
      do i=1,imax
            a1(i)=a(1,i)
            a2(i)=a(2,i)
            a3(i)=a(3,i)
      enddo

      do i=2,imax
         if(a2(i).ne.pzero)then
           factor=a1(i)/a2(i-1)
           a2(i)=a2(i)-(factor*a3(i-1))
           b(i)=b(i)-(factor*b(i-1))
         else
           b(i)=pzero
           a2(i)=1.0d0
         endif
      enddo

x(imax)=b(imax)/a2(imax)

      do i=imax-1,1,-1
            x(i)=(b(i)-(a3(i)*x(i+1)))/a2(i)
      enddo

      do i=1,imax
            b(i)=x(i)
      enddo
```

∗ An array to reverse from virtual node numbering back to real node numbering

```
i2max=neql-neqe
      do ireg=1,nreg
         nmax=nnreg(ireg)
          do iv=1,nmax
           do imat=1,nummat
             inode=ipvt(ireg,iv)
             ivnode=ipvt3(ireg,inode)
             icolR=i2max*nummat*(inode-1)+i2max*(imat-1)+1
             icolV=i2max*nummat*(ivnode-1)+i2max*(imat-1)+1
             breal(icolR)=b(icolV)
             breal(icolR+1)=b(icolV+1)
             breal(icolR+2)=b(icolV+2)
           enddo
         enddo
      enddo

i3max=i2max*nummat*nnodes

do i=1,i3max
   b(i)=breal(i)
enddo
```

APPENDIX H

Slide4.f File

∗ In this file, the only edited part is shown

∗ This subroutine checks for node penetration and makes the appropriate corrections for sliding surfaces

```
node=nfltbnd(j1,jb)
nodem(3)=node
massb=a(1,imax*ipvt4(node)-1)+a(3,imax*ipvt4(node)-1)
      +a(2,imax*ipvt4(node)-1)
bcol(j1)=massb
vxb=vx(1,node)
vyb=vy(1,node)

vni=(vxi-vxf)*nx+(vyi-vyf)*ny
      if(vni.gt.pzero)goto 20
      mass1=a(1,imax*ipvt4(inode)-1)+a(2,imax*ipvt4(inode)-1)
            +a(3,imax*ipvt4(inode)-1)

vn1=vxi*nx+vyi*ny
      if(min(vn1,vn2).gt.pzero)goto 20
      mass1=a(1,imax*ipvt4(inode)-1)+a(2,imax*ipvt4(inode)-1)
            +a(3,imax*ipvt4(inode)-1)
vn1=vxi*nx+vyi*ny
```

APPENDIX I

Update_energy.f File

∗ In this file, the only edited part is shown

∗ This subroutine updates the energy as a result of contact forces

```
      ioff=neql-neqe
            do 10 i=1,nel
               fv=pzero
                 do 15 j=1,ne
                    node=econ(j,i)
                    icol=ioff*ipvt4(node)
                    fv=fv+(vxc(1,node)*vx(1,node)+vyc(1,node)*vy(1,node))*
                        mass(0,i)/(ne*(a(1,icol)+a(2,icol)+a(3,icol)))
15                  continue
                    energy(1,i)=energy(1,i)+fv*dt/mass(1,i)
10          continue
            return
            end
```

APPENDIX J

Input File

```
* Example input file for running MAAP2D
* The first line is the job title, but this can be anywhere in the input file
* Jobtitle = 'Verification Test 8'
*
* Control Records
*
Control
ttype = explicit              !default is explicit
mtype = lagrangian            !default is lagrangian
start = yes                   !this is the default and this line is not necessary
restart = no                  !this is the default and this line is not necessary
remesh = no                   !set equal to yes to allow auto-remeshing (default is no)
glen = 0.001                  !length scale for boundary reseeding during remesh
amr = no                      !set equal to yes to allow AMR (default is no)
nlevel = 4                    !number of levels for an AMR calculation
stest = no                    !default is no for seed test
mtest = no                    !default is no for mesh test
tmin = 0.                     !required input
tmax =1000.                   !required input
tstep = 0.005                 !required input for implicit
cycle_max = 50                !default is 50000
gauss = 2                     !default is 1
weight = volume               !default is volume weighting
file_number = 0               !required input ONLY if restart = yes or remesh = yes
coordinate_system = cyl       !the default is cylindrical
debug=no                      !default is no
end_control
*
* Plot and edit dump records
*
plot
pcycle=0 pdc = 100
```

```
ecycle=0 edc = 100
pvoid=yes                    !default is no
evoid=yes                    !default is no
end_plot
*
* Region definition records
*
regions
        region=1 mesh = uniform material = 1 model = 1
        boundary = 1 btype = exterior
        xvel = 0. yvel = -100.
*
        geometry = line bcond = free
        start = 0. 0.
        finish= 1. 0.
        nseeds = 2
        prop = 1
*
        geometry = line bcond = free
        start = 1. 0.
        finish = 1. 1.
        nseeds = 2
        prop = 1.
*
        geometry = line bcond = free
        start = 1. 1.
        finish = 0. 1.
        nseeds = 2
        prop = 1.
*
        geometry = line bcond = symmetric
        start = 0. 1.
        finish = 0. 0.
        nseeds = 2
        prop = 1.
* end regions
*
* End of input file
*
```

APPENDIX K

Plot Input File

```
* This is an example keyword-based input file for plotting
* The first line here is the job title, but this line could be anywhere in the file
*
* jobtitle = 'Verification test 8 plots'  !this is a comment in a line
evaluate_seeds = no                       !default is no
number = 0                                !this is the number appended to the file seeddmp
                                           that will be read (default is 0)
seed_numbering=yes                        ! the default is no
read_neutral_file=no                      !default is no
neutral_file=rempat                       !neutral file name
* trailer=R                               !trailer for plot dump file (default is none)
*
* Now the input for plot dump files
*
* begin 0 end 10 skip 5
  begin 0 end 10
*
* Now the input telling what to plot
*
        type=mat dim=two
        nodes=yes
        elements=no
        range=-0.2 2.2 -1.2 1.2
*
        type=mat dim=two
        nodes=yes
        elements=no
        range=-2 6 -2 6
*
        type=band dim=two var=pressure
        nodes=yes
        elements=no
```

```
        range=-0.2 2.2 -1.2 1.2
*
        type=band dim=two var=density
        nodes=yes
        elements=no
        range=-0.2 2.2 -1.2 1.2
*
* end of plot input file
```