LA-UR-18-30390

An introduction to finite element methods for Lagrangian solid dynamics

Nathaniel Morgan

X-Computational Physics

Acknowledgements:

V. Chiravalle, S. Tokareva, D. Burton, Steve Beissel, Xiaodong Liu, Konstantin Lipnikov, and Jacob Moore

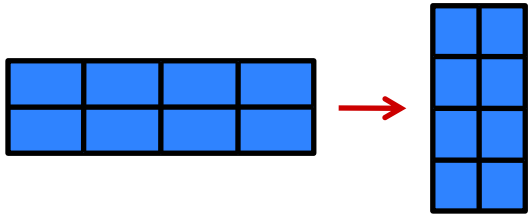
I gratefully acknowledge the support of the ASC program at Los Alamos National Laboratory for supporting the algorithm research

Overview

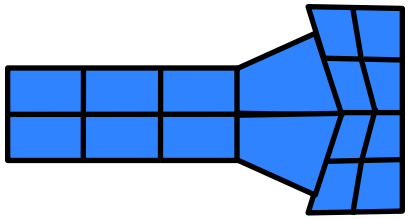
- Geometry and maps
- Mass
- Momentum
- Energy
- Stability on shock problems
- Time integration
- Mesh stability
- Examples
- A 1D C++ lumped mass FE Lagrangian code

What is the difference between Lagrangian and Eulerian methods?

- **Lagrangian: the mesh moves with the flow**

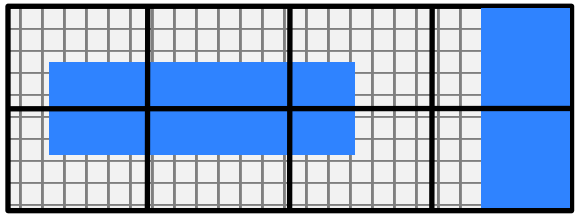


Time=0

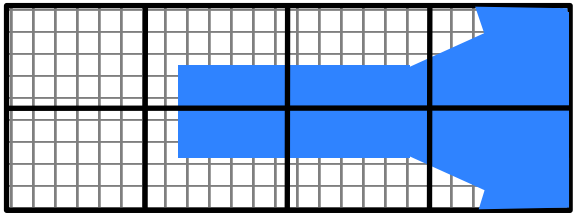


Time=n

- **Eulerian: the mesh is fixed in space so material flows through the mesh**



Time=0



Time=n

- **Arbitrary Lagrangian Eulerian (ALE): Combines both approaches so some portions of the calculation will be Lagrangian and some portions will be Eulerian**

A short review of tensors and indicial notation is helpful for understanding the governing equations and derivations in this talk

stress

$$\boldsymbol{\sigma} = \begin{bmatrix} \sigma_{11} & \sigma_{12} & \sigma_{13} \\ \sigma_{21} & \sigma_{22} & \sigma_{23} \\ \sigma_{31} & \sigma_{32} & \sigma_{33} \end{bmatrix} = \sigma_{ij} \hat{\mathbf{e}}_i \otimes \hat{\mathbf{e}}_j$$

velocity gradient

$$\nabla \otimes \mathbf{v} = \begin{bmatrix} \frac{\partial v_1}{\partial x_1} & \frac{\partial v_1}{\partial x_2} & \frac{\partial v_1}{\partial x_3} \\ \frac{\partial v_2}{\partial x_1} & \frac{\partial v_2}{\partial x_2} & \frac{\partial v_3}{\partial x_3} \\ \frac{\partial v_3}{\partial x_1} & \frac{\partial v_3}{\partial x_2} & \frac{\partial v_3}{\partial x_3} \end{bmatrix} = \frac{\partial v_i}{\partial x_j} \hat{\mathbf{e}}_i \otimes \hat{\mathbf{e}}_j$$

pressure gradient

$$\nabla p = \begin{bmatrix} \frac{\partial h}{\partial x_1} & \frac{\partial h}{\partial x_2} & \frac{\partial h}{\partial x_3} \end{bmatrix} = \frac{\partial p}{\partial x_i} \hat{\mathbf{e}}_i$$

Inner product

$$\mathbf{a} \cdot \mathbf{b} = a_i b_i$$

Outer product

$$\mathbf{a} \otimes \mathbf{b} = a_i b_j \hat{\mathbf{e}}_i \otimes \hat{\mathbf{e}}_j$$

Double concatenation

$$\mathbf{A} : \mathbf{B} = a_{ij} b_{ji}$$

Overview of governing analytic Lagrangian equations that we seek to solve

Differential form

Volume: $$\frac{1}{w}\frac{dw}{dt} - \nabla \cdot \mathbf{v} = 0$$

Specific volume: $$\rho \frac{d\gamma}{dt} - \nabla \cdot \mathbf{v} = 0$$

Momentum: $$\rho \frac{d\mathbf{v}}{dt} - \nabla \cdot \boldsymbol{\sigma} = 0$$

Internal Energy: $$\rho \frac{de}{dt} - \boldsymbol{\sigma} : \nabla \otimes \mathbf{v} = 0$$

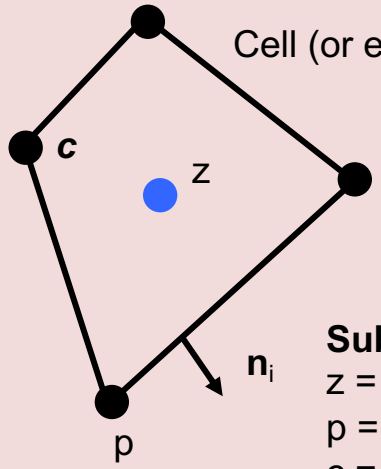
$$\rho \frac{de}{dt} + p\left(\nabla \cdot \mathbf{v}\right) = 0$$

Total Energy: $$\rho \frac{d\tau}{dt} - \nabla \cdot \left(\boldsymbol{\sigma} \cdot \mathbf{v}\right) = 0$$

Velocity: $$\frac{d\mathbf{x}}{dt} = \mathbf{v}$$

Nomenclature

w = Volume
γ = Specific volume, inverse of density
$\mathbf{x}$ = position
$\boldsymbol{\sigma}$ = Stress
$\mathbf{v}$ = Velocity
τ = Specific total energy = e+k
ρ = Density
e = Specific internal energy



Subscripts:
z = cell centered quantity
p = vertex of cell
c = cell corner
i = facet of cell boundary

Geometry and maps

The position, displacement, and velocity in the cell (also called zone or element) are approximated with interpolation polynomials

Position in the cell

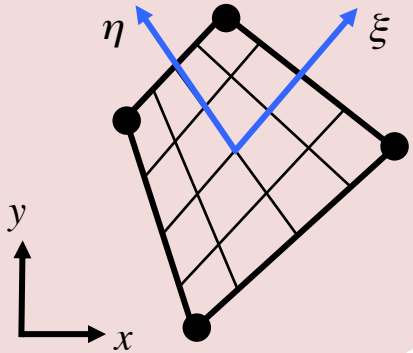
$$\mathbf{x} = \sum_{p \in h} \mathbf{x}_p \phi_p(\xi, \eta, \zeta)$$

Displacement in the cell

$$\mathbf{u} = \sum_{p \in h} \mathbf{u}_p \phi_p(\xi, \eta, \zeta)$$

Velocity in the cell

$$\mathbf{v} = \sum_{p \in h} \mathbf{v}_p \phi_p(\xi, \eta, \zeta)$$



Basis functions sum to 1 at any point in the cell, called partition of unity

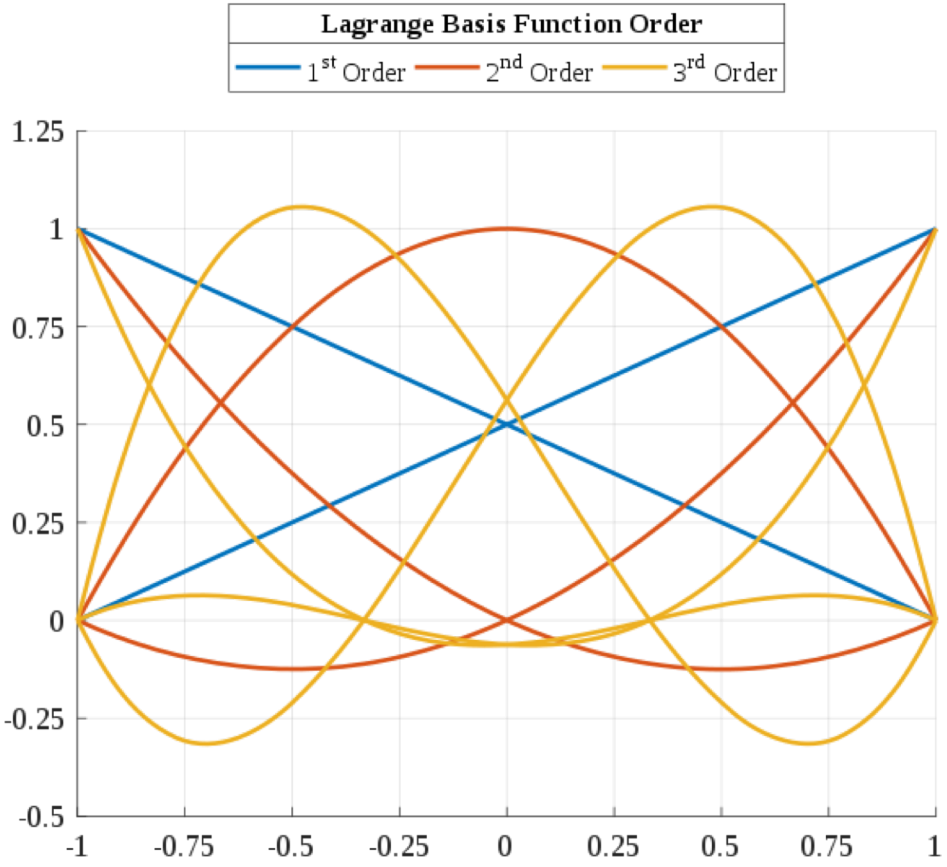
$$\begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \\ y_1 & y_2 & \cdots & y_N \\ z_1 & z_2 & \cdots & z_N \end{bmatrix} \begin{bmatrix} \phi_1(\xi) \\ \phi_2(\xi) \\ \vdots \\ \phi_N(\xi) \end{bmatrix}$$

Basis functions for a vertex (also called shape functions)

The basis functions for each point $\phi_p(\xi, \eta, \zeta)$ are expressed in terms of a reference coordinates (ξ, η, ζ) that typically range from -1 to 1

The Lagrange interpolating functions are commonly used in finite element methods

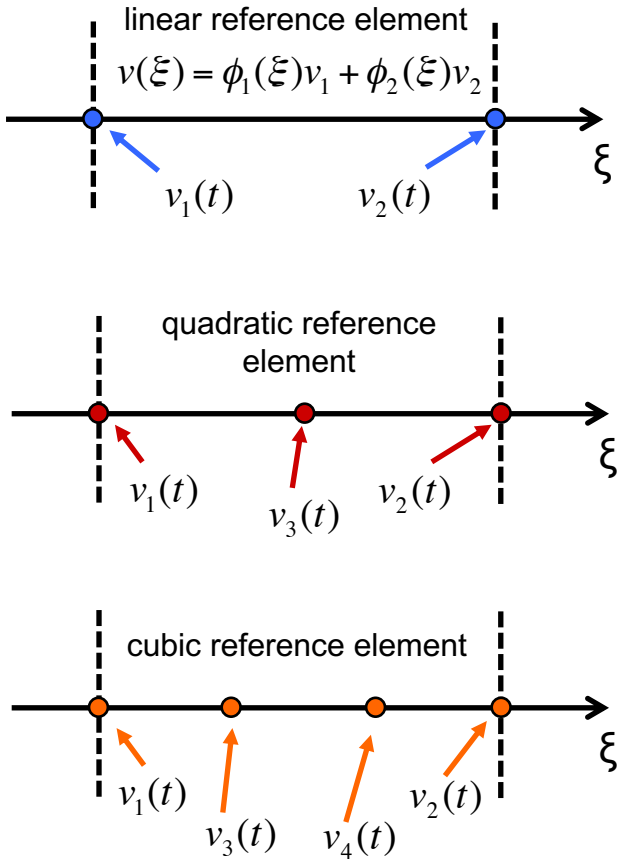
The Lagrange polynomials are commonly used with finite element (FE) methods



1. Plot from Wikipedia, https://en.wikipedia.org/wiki/Lagrange_polynomial

Velocity in the cell h

$$v(\xi,t) = \sum_{p\in h} \phi_p(\xi) v_p(t)$$



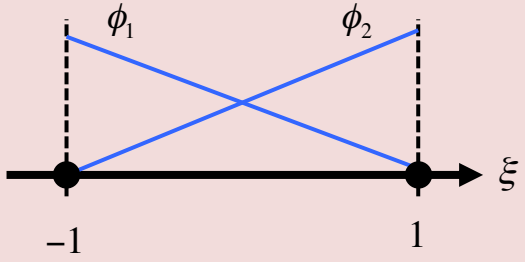
Linear Lagrange polynomial examples

$$\phi_1(\xi) = \frac{1}{2}(1-\xi)$$

$$\phi_2(\xi) = \frac{1}{2}(1+\xi)$$

or

$$\phi_p(\xi) = \frac{1}{2}(1+\xi_p\xi)$$



The interpolation polynomial for a linear line segment

Fig: A 1D example

$$\phi_p = \frac{1}{8}(1+\xi\xi_p)(1+\eta\eta_p)(1+\zeta\zeta_p)$$

The interpolation polynomial for a linear hexahedron

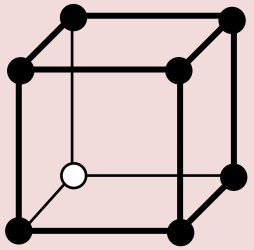


Fig: A 3D example

Shape functions exist for higher-order elements

- Serendipity 3D elements only have degrees of freedom on the cell exterior up to cubic elements

Hexahedron with 8 vertices

$$\phi_p = \frac{1}{8}(1+\xi_0)(1+\eta_0)(1+\zeta_0)$$

$$\xi_0 = \xi\xi_p \qquad \eta_0 = \eta\eta_p \qquad \zeta_0 = \zeta\zeta_p$$

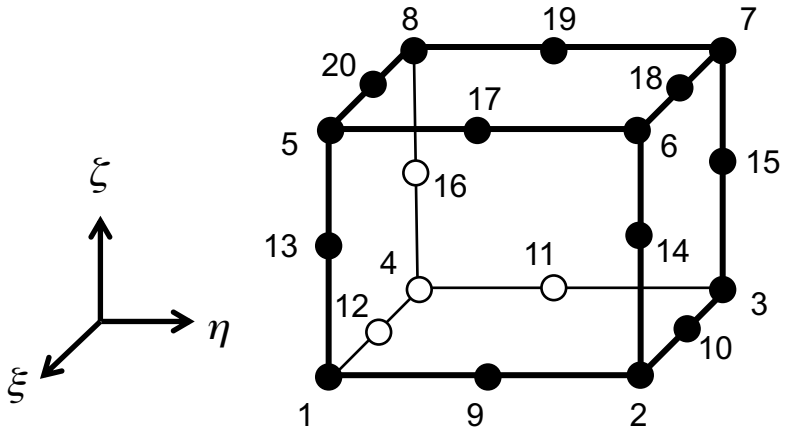
Hexahedron with 20 vertices

$$\phi_p = \frac{1}{8}(1+\xi_0)(1+\eta_0)(1+\zeta_0)(\xi_0+\eta_0+\zeta_0-2) \qquad \text{p=1-8}$$

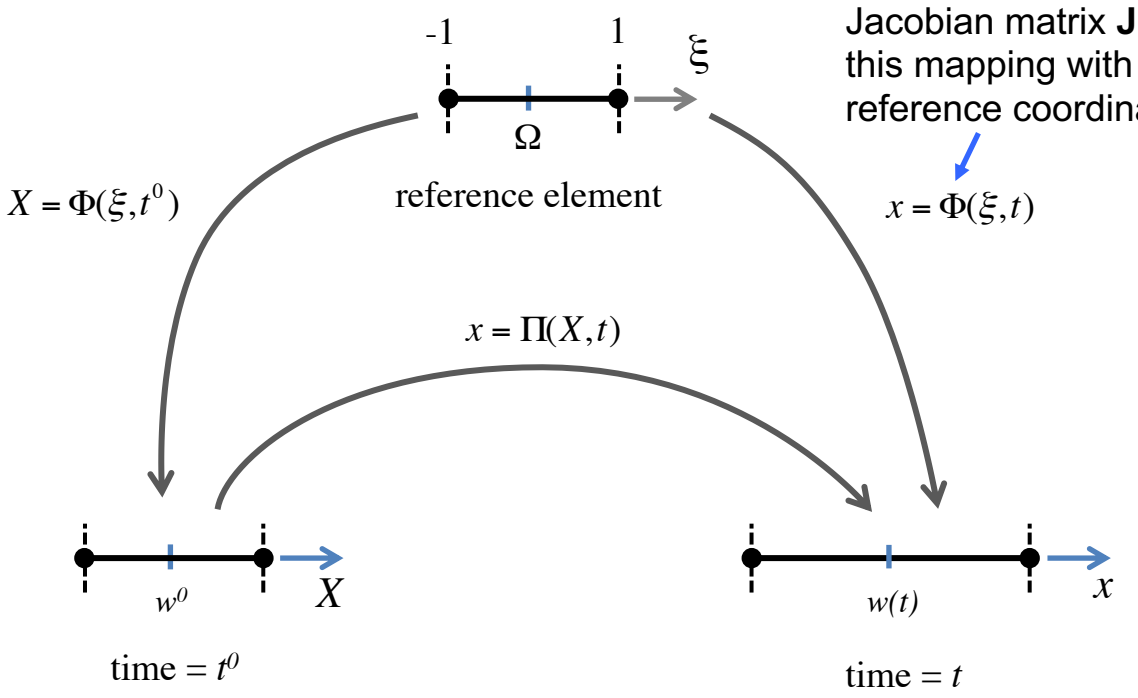
$$\phi_p = \frac{1}{4}(1-\xi^2)(1+\eta_0)(1+\zeta_0) \qquad \text{p= 9, 11, 17, 19}$$

$$\phi_p = \frac{1}{4}(1-\eta^2)(1+\xi_0)(1+\zeta_0) \qquad \text{p= 10, 12, 18, 20}$$

$$\phi_p = \frac{1}{4}(1-\zeta^2)(1+\xi_0)(1+\eta_0) \qquad \text{p= 13, 14, 15, 16}$$

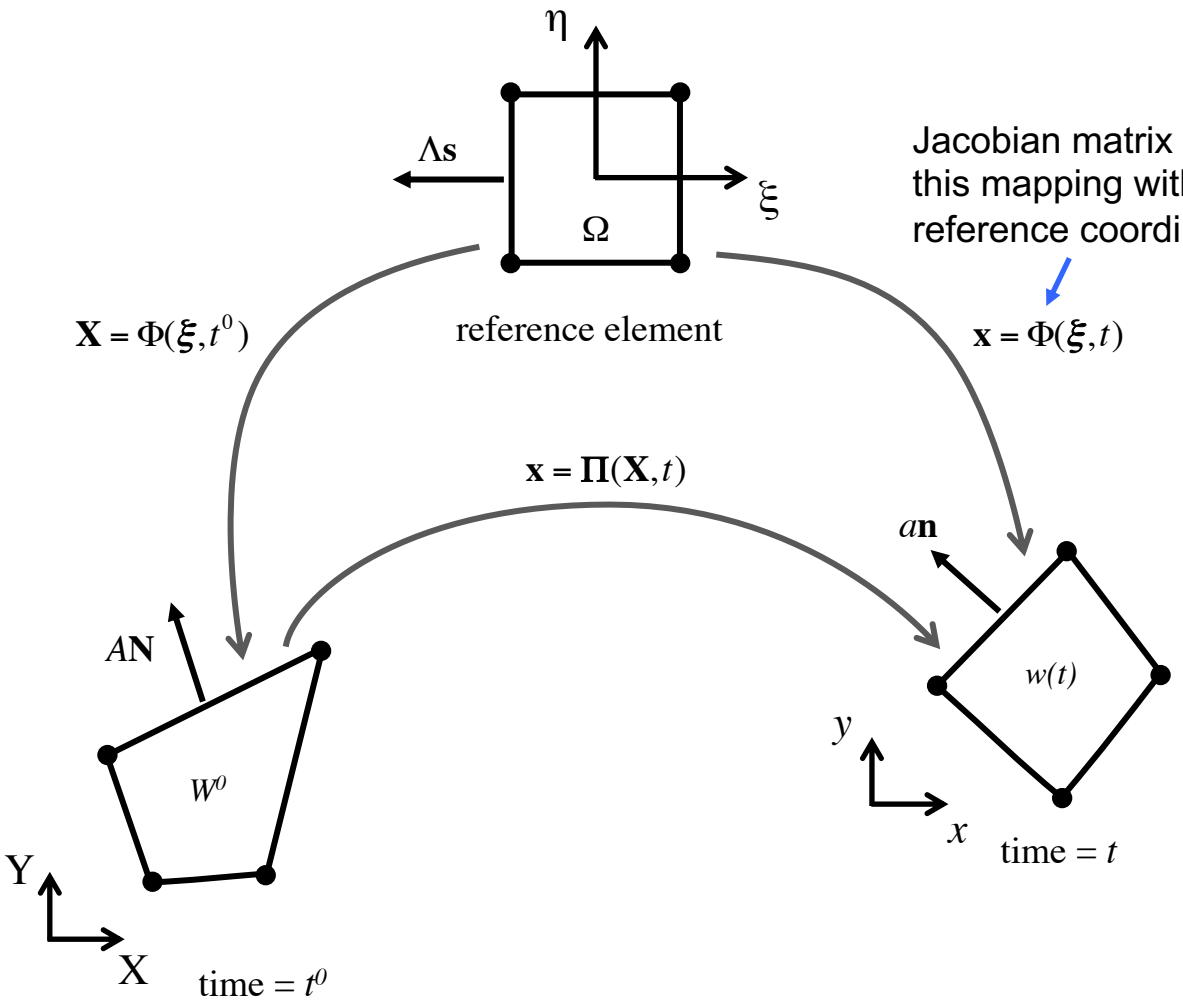


1D reference element



Jacobian matrix **J** is the gradient of this mapping with respect to the reference coordinates

2D reference element



Interpolating polynomials describe the deformation of the cell

- **The mapping for a position is**

$$\mathbf{x} = \boldsymbol{\Phi}(\boldsymbol{\xi}, t) = \sum_{p \in h} \mathbf{x}_p \phi_p(\xi, \eta, \zeta)$$

$$\begin{bmatrix} 1 \\ x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \\ y_1 & y_2 & \cdots & y_N \\ z_1 & z_2 & \cdots & z_N \end{bmatrix} \begin{bmatrix} \phi_1(\boldsymbol{\xi}) \\ \phi_2(\boldsymbol{\xi}) \\ \vdots \\ \phi_N(\boldsymbol{\xi}) \end{bmatrix}$$

- **The Jacobi matrix is:**

$$\mathbf{J}(\boldsymbol{\xi}, t) = \nabla_{\xi} \otimes \boldsymbol{\Phi} = \frac{\partial \mathbf{x}}{\partial \boldsymbol{\xi}} = \frac{\partial x_i}{\partial \xi_j} \hat{\mathbf{e}}_i \otimes \hat{\mathbf{e}}_j$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial x}{\partial \eta} & \frac{\partial x}{\partial \zeta} \\ \frac{\partial y}{\partial \xi} & \frac{\partial y}{\partial \eta} & \frac{\partial y}{\partial \zeta} \\ \frac{\partial z}{\partial \xi} & \frac{\partial z}{\partial \eta} & \frac{\partial z}{\partial \zeta} \end{bmatrix}$$

$$\frac{\partial x}{\partial \xi} = \sum_{p \in h} \frac{\partial \phi_p}{\partial \xi} x_p \quad \frac{\partial x}{\partial \eta} = \sum_{p \in h} \frac{\partial \phi_p}{\partial \eta} x_p \quad \frac{\partial x}{\partial \zeta} = \sum_{p \in h} \frac{\partial \phi_p}{\partial \zeta} x_p$$

$$\frac{\partial y}{\partial \xi} = \sum_{p \in h} \frac{\partial \phi_p}{\partial \xi} y_p \quad \frac{\partial y}{\partial \eta} = \sum_{p \in h} \frac{\partial \phi_p}{\partial \eta} y_p \quad \frac{\partial y}{\partial \zeta} = \sum_{p \in h} \frac{\partial \phi_p}{\partial \zeta} y_p$$

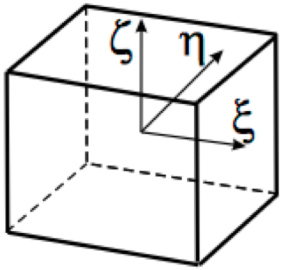
$$\frac{\partial z}{\partial \xi} = \sum_{p \in h} \frac{\partial \phi_p}{\partial \xi} z_p \quad \frac{\partial z}{\partial \eta} = \sum_{p \in h} \frac{\partial \phi_p}{\partial \eta} z_p \quad \frac{\partial z}{\partial \zeta} = \sum_{p \in h} \frac{\partial \phi_p}{\partial \zeta} z_p$$

$$\mathbf{J}(\boldsymbol{\xi}, t) = \sum_{p \in h} \nabla_{\xi} \phi_p(\boldsymbol{\xi}) \otimes \mathbf{x}_p(t)$$

Interpolating polynomials describe the deformation of the cell (cont.)

- **The integral over the volume**

$$\int_w h(x,y,z)\,dx\,dy\,dz = \int_\Omega f(\xi,\eta,\zeta)\ \det(\mathbf{J})\,d\xi\,d\eta\,d\zeta$$



- **The inverse of the Jacobi matrix:**

$$\mathbf{J}^{-1}(\boldsymbol{\xi},t) = \frac{\partial \xi_i}{\partial x_j}\hat{\mathbf{e}}_i \otimes \hat{\mathbf{e}}_j$$

$$\mathbf{J}^{-1} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \xi}{\partial y} & \frac{\partial \xi}{\partial z} \\ \frac{\partial \eta}{\partial x} & \frac{\partial \eta}{\partial y} & \frac{\partial \eta}{\partial z} \\ \frac{\partial \zeta}{\partial x} & \frac{\partial \zeta}{\partial y} & \frac{\partial \zeta}{\partial z} \end{bmatrix}$$

and the transpose is

$$\mathbf{J}^{-T} = \begin{bmatrix} \frac{\partial \xi}{\partial x} & \frac{\partial \eta}{\partial x} & \frac{\partial \zeta}{\partial x} \\ \frac{\partial \xi}{\partial y} & \frac{\partial \eta}{\partial y} & \frac{\partial \zeta}{\partial y} \\ \frac{\partial \xi}{\partial z} & \frac{\partial \eta}{\partial z} & \frac{\partial \zeta}{\partial z} \end{bmatrix}$$

- **The determinant of Jacobi matrix:**

$$j = \det(\mathbf{J})$$

Finite element methods typically solve the governing equations on a reference element

- **The spatial gradient in the reference element is**

$$\nabla_x h(\boldsymbol{\xi}) = \frac{\partial h}{\partial x_j}\hat{\mathbf{e}}_j = \left(\frac{\partial h}{\partial \xi_i}\right)\left(\frac{\partial \xi_i}{\partial x_j}\right)\hat{\mathbf{e}}_j = \left(\frac{\partial h}{\partial \xi_i}\right)J^{-1}_{ij}\hat{\mathbf{e}}_j = J^{-T}_{ji}\left(\frac{\partial h}{\partial \xi_i}\right)\hat{\mathbf{e}}_j = \mathbf{J}^{-T}\cdot\nabla_\xi h$$

Example:
$$M_p\frac{d\mathbf{v}_p}{dt} = -\sum_{h\in p}\left(\int_{w(t)}(\nabla\phi_p)\cdot\boldsymbol{\sigma}\,dw\right) = -\sum_{h\in p}\left(\int_{\Omega}\mathbf{J}^{-T}\cdot(\nabla_\xi\phi_p)\cdot\boldsymbol{\sigma}\,j\,d\Omega\right)$$

$h =$ Arbitrary scalar field
$\mathbf{h} =$ Arbitrary vector field
$\hat{\mathbf{e}}_j =$ A basis vector

Fig: Nomenclature

- **The divergence and gradient in the reference element are**

$$\nabla_x\cdot\mathbf{h}(\boldsymbol{\xi}) = \frac{\partial h_i}{\partial x_i} = \left(\frac{\partial h_i}{\partial \xi_j}\right)\left(\frac{\partial \xi_j}{\partial x_i}\right) = \left(\frac{\partial h_i}{\partial \xi_j}\right)J^{-1}_{ji} = J^{-T}_{ij}\frac{\partial h_i}{\partial \xi_j} = \mathbf{J}^{-T}:\nabla_\xi\otimes\mathbf{h}$$

$$\nabla_x\otimes\mathbf{h}(\boldsymbol{\xi}) = \frac{\partial h_i}{\partial x_j}\hat{\mathbf{e}}_i\otimes\hat{\mathbf{e}}_j = \left(\frac{\partial h_i}{\partial \xi_k}\right)\left(\frac{\partial \xi_k}{\partial x_j}\right)\hat{\mathbf{e}}_i\otimes\hat{\mathbf{e}}_j = \frac{\partial h_i}{\partial \xi_k}J^{-1}_{kj}\hat{\mathbf{e}}_i\otimes\hat{\mathbf{e}}_j = J^{-T}_{jk}\frac{\partial h_i}{\partial \xi_k}\hat{\mathbf{e}}_j\otimes\hat{\mathbf{e}}_i = \mathbf{J}^{-T}\cdot\nabla_\xi\otimes\mathbf{h}$$

Example:
$$M_z\frac{de_z}{dt} = \int_{w(t)}\boldsymbol{\sigma}:(\nabla\otimes\mathbf{v})\,dw = \int_{\Omega}\boldsymbol{\sigma}:(\mathbf{J}^{-T}\cdot\nabla_\xi\otimes\mathbf{v})\,j\,d\Omega$$

Mass

The strong-form mass conservation approach enforces conservation at every quadrature point in the element

- **Mass in a cell is constant for a Lagrangian calculation**

$$\frac{d}{dt}M = \frac{d}{dt}\int_{w(t)} \rho\, dw = \frac{d}{dt}\int_{\Omega} \rho j\, d\Omega = 0$$

- **Volume integrals are calculated using Gauss quadrature**

$$\int_{\Omega} f(\xi,\eta,\zeta)\, d\Omega = \sum_{g} f(\xi_g,\eta_g,\zeta_g)\, \varpi_g$$

weight

Number of points	Points, ξ_g	Weights, ω_g
1	0	2
2	$\pm\frac{1}{\sqrt{3}}$	1
3	0	$\frac{8}{9}$
	$\pm\sqrt{\frac{3}{5}}$	$\frac{5}{9}$

Quadrature rules for multiple dimensions can be calculated by using a tensor product of 1D quadrature rules

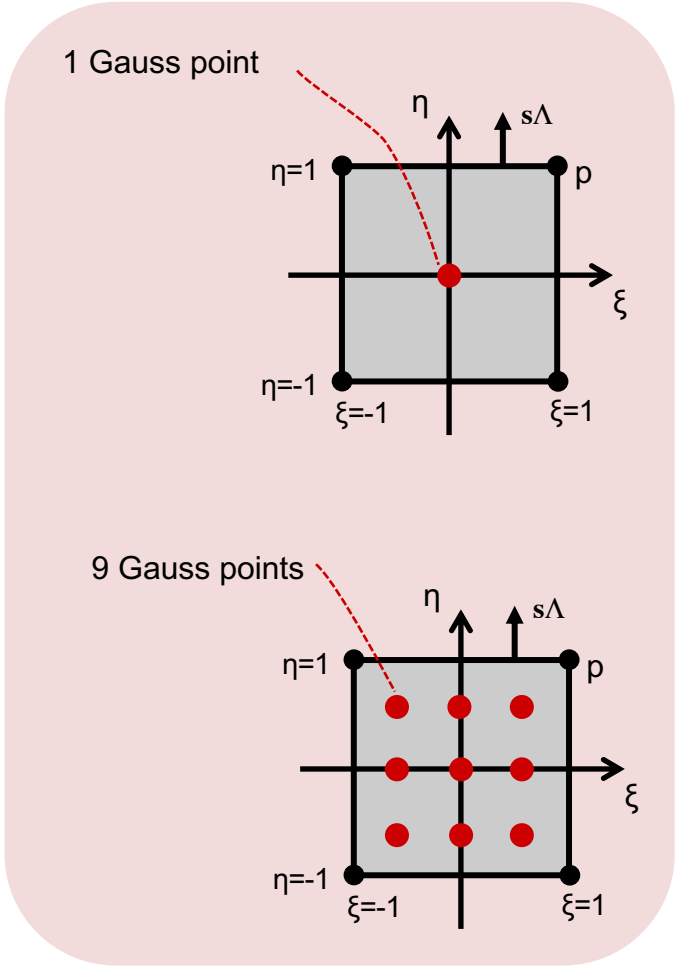


Fig: A 2D example

The strong-form mass conservation approach enforces conservation at every quadrature point in the element (cont.)

- **Mass conservation:**

$$\frac{d}{dt}M = \frac{d}{dt}\int_{w(t)} \rho\, dw = \frac{d}{dt}\int_{\Omega} \rho\, j\, d\Omega = 0$$ Analytic equation

$$M = \int_{\Omega} \rho(t^0,\xi)\, j(t^0,\xi)\, d\Omega = \int_{\Omega} \rho(t,\xi)\, j(t,\xi)\, d\Omega$$

$$\rho(t^0,\xi)\, j(t^0,\xi) = \rho(t,\xi)\, j(t,\xi)$$

Initial density at the quadrature point

$$M = \sum_g \rho_g^0 j_g^0 \varpi_g = \sum_g \rho_g j_g \varpi_g$$ Conservation is enforced for any mesh resolution and quadrature set

$$\rho_g^0 j_g^0 = \rho_g j_g$$

$$\rho_g = \rho_g^0 \frac{j_g^0}{j_g}$$

Weak-forms enforce conservation (or some other property) when integrated over the element; whereas, strong-forms enforce conservation (or some other property) at every point

Momentum

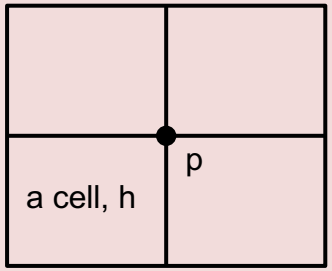
The finite element method for solving the momentum equation

- **Discretization:**

$$\rho\frac{d\mathbf{v}}{dt}-\nabla\cdot\boldsymbol{\sigma}=0$$

$$\sum_{h\in q}\int_{w(t)}\phi_q\left(\rho\frac{d\mathbf{v}}{dt}-\nabla\cdot\boldsymbol{\sigma}\right)dw=0$$

Multiply by the basis for the velocity (a.k.a. test function)



The cells used to calculate the velocity at p

$$\sum_{h\in q}\int_{w(t)}\rho\phi_q\frac{d\mathbf{v}}{dt}dw=\sum_{h\in q}\int_{w(t)}\nabla\cdot\left(\phi_q\boldsymbol{\sigma}\right)dw-\sum_{h\in q}\int_{w(t)}\left(\nabla\phi_q\right)\cdot\boldsymbol{\sigma}\,dw$$

Sum over all the cells that have test function q

$$\sum_{h\in q}\int_{w(t)}\rho\phi_q\frac{d\mathbf{v}}{dt}dw=\sum_{h\in q}\oint_{\partial w(t)}da\mathbf{n}\cdot\left(\phi_q\boldsymbol{\sigma}^*\right)-\sum_{h\in q}\int_{w(t)}\left(\nabla\phi_q\right)\cdot\boldsymbol{\sigma}\,dw$$

$$\mathbf{v}=\sum_{p\in h}\mathbf{v}_p\phi_p(\xi,\eta,\zeta)$$

The velocity field in a cell

The velocity at node p

$$\sum_{q\in h}\sum_{p\in h}\int_{w(t)}\rho\phi_q\phi_p\,dw\frac{d\mathbf{v}_p}{dt}=\sum_{h\in q}\oint_{\partial w(t)}da\mathbf{n}\cdot\left(\phi_q\boldsymbol{\sigma}^*\right)-\sum_{h\in q}\int_{w(t)}\left(\nabla\phi_q\right)\cdot\boldsymbol{\sigma}\,dw$$

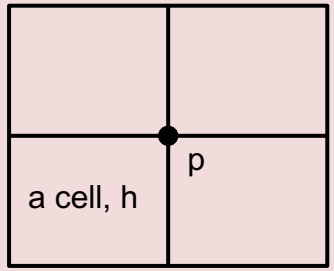
Sparse, but global matrix

=0 because it is continuous

One can lump the mass to the nodes to obviate the need to invert a sparse global mass matrix

- **Lumped mass formulation**

$$M_p \frac{d\mathbf{v}_p}{dt} = -\sum_{h \in p} \left(\int_{w(t)} \left(\nabla \phi_p \right) \cdot \boldsymbol{\sigma} \, dw \right) = \sum_{c \in p} \mathbf{F}_c$$



The cells used to calculate the velocity at p

- **Many options for lumping the mass**

  Option 1) Use a lumped mass approximation [1] and assign a portion of the cell mass to the node

  Option 2) Sum over all the columns in the row

1. V. Chiravalle and N. Morgan, *A 3D finite element ALE method using an approximate Riemann solution*, IJNMF, 2016; 83:642-663

The finite element method for solving momentum equation (cont.)

- **Calculating the forces on the nodes**

$$M_p \frac{d\mathbf{v}_p}{dt} = -\sum_{h \in p} \left(\int_{\Omega} \mathbf{J}^{-T} \cdot \left(\nabla_{\xi} \phi_p \right) \cdot \sigma j \, d\Omega \right)$$

Volume integrals are calculated using Gauss quadrature

$$\int_{\Omega} f(\xi, \eta, \zeta) d\Omega = \sum_{g} f(\xi_g, \eta_g, \zeta_g) \, \varpi_g$$

ϖ_g: weight

Number of points	Points, ξ_g	Weights, ω_g
1	0	2
2	$\pm\frac{1}{\sqrt{3}}$	1
3	0	$\frac{8}{9}$
	$\pm\sqrt{\frac{3}{5}}$	$\frac{5}{9}$

Quadrature rules for multiple dimensions can be calculated by using a tensor product of 1D quadrature rules

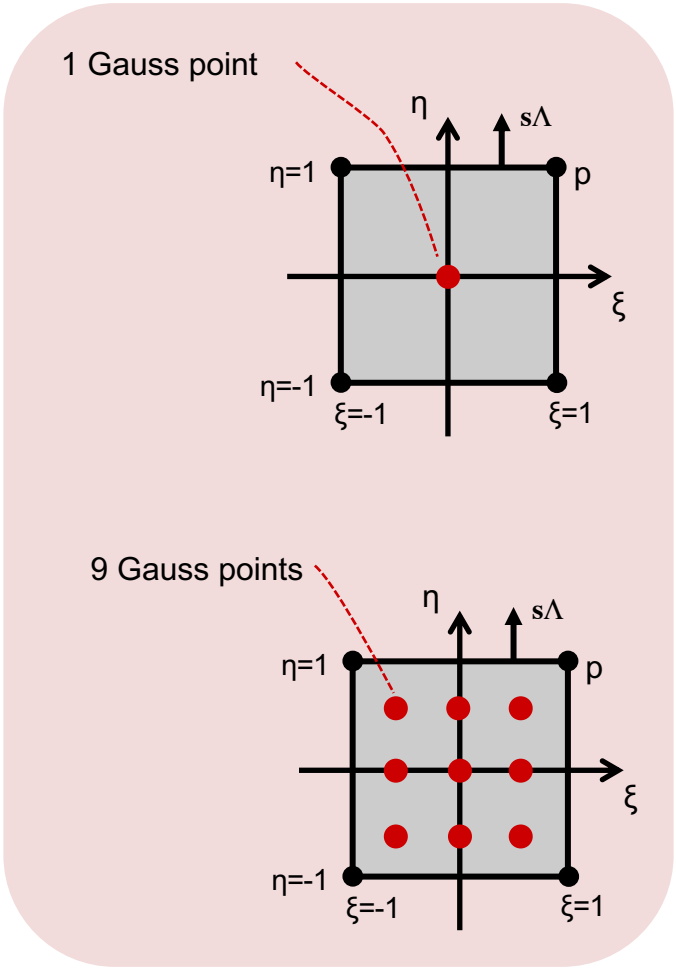


Fig: A 2D example

The finite element method for solving momentum equation (cont.)

- **The discrete momentum equation with quadrature**

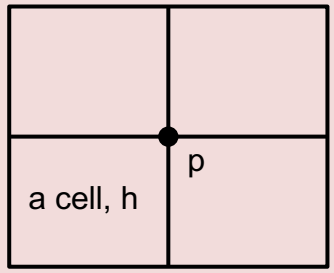
$$M_p \frac{d\mathbf{v}_p}{dt} = -\sum_{h\in p}\left(\sum_{g\in h}\left(\mathbf{J}^{-T}\cdot\left(\nabla_\xi \phi_p\right)\cdot\boldsymbol{\sigma}\,\varpi\, j\right)\right)$$

evaluate each term in the integral at the quadrature points $\boldsymbol{\xi}_g = (\xi_g, \eta_g, \zeta_g)$

$$M_p \frac{d\mathbf{v}_p}{dt} = -\sum_{h\in p}\left(\sum_{g\in h}\left(\mathbf{J}^{-T}(\boldsymbol{\xi}_g)\cdot\left(\nabla_\xi \phi_p(\boldsymbol{\xi}_g)\right)\cdot\boldsymbol{\sigma}_g\varpi_g j(\boldsymbol{\xi}_g)\right)\right) = \sum_{c\in p}\left(\mathbf{F}_c\right)$$

- **A classical approach [1] is to use a single quadrature point in a linear element**

$$M_p \frac{d\mathbf{v}_p}{dt} = -\sum_{h\in p}\left(\mathbf{J}^{-T}(\boldsymbol{\xi}_z)\cdot\left(\nabla_\xi \phi_p(\boldsymbol{\xi}_z)\right)\cdot\boldsymbol{\sigma}_z\varpi_z j(\boldsymbol{\xi}_z)\right) = \sum_{c\in p}\left(\mathbf{F}_c\right)$$



The cells used to calculate the velocity at p

1. D. Flanagan and T. Belytchko, *A uniform strain hexahedron and quadrilateral with orthogonal hourglass control*, IJNME, 1981; 17:679-706

Energy

Total energy can be conserved in finite element formulations

- **The internal energy equation must be compatible with the change of kinetic energy and momentum, and the time integration method [1]**

Define kinetic energy change for a vertex

$$K_p^n = \frac{1}{2} M_P \mathbf{v}_p^n \cdot \mathbf{v}_p^n$$

$$K_p^{n+1} = \frac{1}{2} M_P \mathbf{v}_p^{n+1} \cdot \mathbf{v}_p^{n+1}$$

$$M_P = \sum_{c \in p} M_c$$

$$K_p^{n+1} - K_p^n = M_P \left(\mathbf{v}_p^{n+1} - \mathbf{v}_p^n \right) \cdot \left(\frac{\mathbf{v}_p^{n+1} + \mathbf{v}_p^n}{2} \right)$$

$$= \sum_{c \in p} M_c \left(\mathbf{v}_p^{n+1} - \mathbf{v}_p^n \right) \cdot \left(\frac{\mathbf{v}_p^{n+1} + \mathbf{v}_p^n}{2} \right)$$

Momentum change

$$\sum_{c \in p} M_c \frac{\Delta \mathbf{v}_p}{\Delta t} = \sum_{c \in p} \mathbf{F}_c$$

$$M_c \Delta \mathbf{v}_p = \Delta t \, \mathbf{F}_c$$

The corner contribution

$$\mathbf{v}_P^{n+\frac{1}{2}}$$

A function of the time integration

Change in kinetic energy is:

$$K_p^{n+1} - K_p^n = \sum_{c \in p} \left(K_c^{n+1} - K_c^n \right) = \Delta t \sum_{c \in p} \left(\mathbf{F}_c^{n+\frac{1}{2}} \cdot \mathbf{v}_P^{n+\frac{1}{2}} \right)$$

1. D. Burton, *Multidimensional discretization of conservation laws for unstructured polyhedral grids*, Lawrence Livermore National Laboratory, 1994, UCRL-JC-118306.

Total energy can be conserved in finite element formulations (cont.)

Change in kinetic energy in a corner is: $K_c^{n+1} - K_c^n = M_c^n \left(\mathbf{v}_p^{n+1} - \mathbf{v}_p^n \right) \cdot \left(\frac{\mathbf{v}_p^{n+1} + \mathbf{v}_p^n}{2} \right) = \Delta t \mathbf{F}_c^{n+\frac{1}{2}} \cdot \mathbf{v}_P^{n+\frac{1}{2}}$

Change in total energy for the domain: $\left(\mathrm{T}_D^{n+1} - \mathrm{T}_D^n \right) = \sum_{c \in D} \left(E_c^{n+1} - E_c^n + K_c^{n+1} - K_c^n \right) = 0 \rightarrow E_c^{n+1} - E_c^n = -\left(K_c^{n+1} - K_c^n \right)$

$E_c^{n+1} = M_c e_z^{n+1}$

$E_c^n = M_c e_z^n$

$M_z = \sum_{c \in z} M_c$

The temporal average

$$M_z \frac{e_z^{n+1} - e_z^n}{\Delta t} = -\sum_{c \in h} \mathbf{F}_c^{n+\frac{1}{2}} \cdot \mathbf{v}_p^{n+\frac{1}{2}}$$

Compatible change in internal energy

or

$$M_z \frac{e_z^{n+1} - e_z^n}{\Delta t} = \sum_{p \in h} \left(\left(\int_{w(t)} \left(\nabla \phi_p \right) \cdot \boldsymbol{\sigma} \, dw \right)^{n+\frac{1}{2}} \cdot \mathbf{v}_p^{n+\frac{1}{2}} \right) = \left(\int_{w(t)} \left(\boldsymbol{\sigma} : \nabla \otimes \mathbf{v} \right) dw \right)^{n+\frac{1}{2}}$$

$$\mathbf{v}_p^{n+\frac{1}{2}} = \frac{1}{2} \left(\mathbf{v}_p^{n+1} - \mathbf{v}_p^n \right)$$

$$\left(\nabla \phi_p \right) \cdot \boldsymbol{\sigma} \cdot \mathbf{v}_p = \left(\partial_i \phi_p \right) \sigma_{ij} v_{pj} = \left(\partial_i \phi_p v_{pj} \right) \sigma_{ij} = \boldsymbol{\sigma} : \left(\nabla \otimes \left(\phi_p \mathbf{v}_p \right) \right)$$

$$\mathbf{v}(\xi) = \sum_{p \in z} \mathbf{v}_p \phi_p(\xi) = \phi_p v_{pj}$$

A strong-form finite element method for solving the internal energy equation

- **Analytic equation**

A metal:

$$\rho \frac{de}{dt} = \boldsymbol{\sigma} : \nabla \otimes \mathbf{v}$$

A gas:

$$\rho \frac{de}{dt} = -p\left(\nabla \cdot \mathbf{v}\right)$$

- **Discretization**

$$\rho_g \left.\frac{de}{dt}\right|_g = \boldsymbol{\sigma}_g : \left(\nabla \otimes \mathbf{v}\right)_g$$

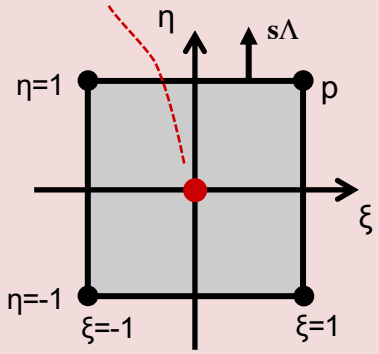
Internal energy is evolved at the quadrature points

Jacobian matrix

$$\rho_g \left.\frac{de}{dt}\right|_g = \boldsymbol{\sigma}_g : \mathbf{J}_g^{-T} \cdot \left(\nabla_\xi \otimes \mathbf{v}\right)_g$$

Velocity gradient: $\nabla_\xi \otimes \mathbf{v}(\boldsymbol{\xi}) = \sum_{p \in z} \nabla_\xi \otimes \left(\phi_p(\boldsymbol{\xi}) \mathbf{v}_p\right) = \sum_{p \in z} \left(\nabla_\xi \phi_p(\boldsymbol{\xi})\right) \otimes \mathbf{v}_p$

Internal energy location for 1 Gauss-Legendre quadrature point



Internal energy location for 25 Lobatto quadrature point

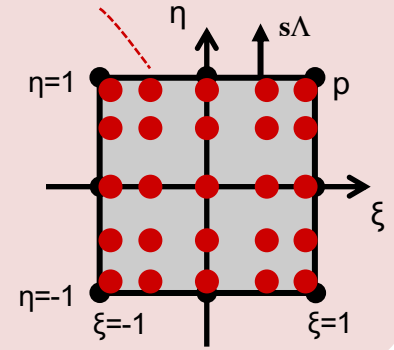


Fig: A 2D example

A strong-form finite element method for solving the internal energy equation (cont.)

- **The strong-form conserves total energy**

$$\rho_g \left.\frac{de}{dt}\right|_g = \boldsymbol{\sigma}_g : \left(\mathbf{J}_g^{-T} \cdot \nabla_\xi \otimes \mathbf{v}\right)_g$$

Internal energy is evolved at the quadrature points

quadrature weight

$$\varpi_g \rho_g j_g \left.\frac{de}{dt}\right|_g = \varpi_g \boldsymbol{\sigma}_g : \left(\mathbf{J}_g^{-T} \cdot \nabla_\xi \otimes \mathbf{v}\right)_g j_g$$

$$\sum_g \left(M_g \left.\frac{de}{dt}\right|_g \right) = \sum_g \varpi_g \boldsymbol{\sigma}_g : \left(\mathbf{J}_g^{-T} \cdot \nabla_\xi \otimes \mathbf{v}\right)_g j_g$$

$$M_z \frac{de_z}{dt} = \int_\Omega \left(\boldsymbol{\sigma} : \mathbf{J}^{-T} \cdot \nabla_\xi \otimes \mathbf{v}\right) j \, d\Omega \qquad \text{or} \qquad M_z \frac{de_z}{dt} = \int_{w(t)} \left(\boldsymbol{\sigma} : \nabla \otimes \mathbf{v}\right) dw$$

Which is the compatible internal energy change for the cell if the velocity is the temporal average, please see earlier derivation

$$M_z \frac{e_z^{n+1} - e_z^n}{\Delta t} = \left(\int_{w(t)} \left(\boldsymbol{\sigma} : \nabla \otimes \mathbf{v}\right) dw \right)^{n+\frac{1}{2}}$$

The weak-form for solving the internal energy equation

- **Discretization:**

A metal: $\rho \frac{de}{dt} = \boldsymbol{\sigma} : \nabla \otimes \mathbf{v}$

A gas: $\rho \frac{de}{dt} = -p(\nabla \cdot \mathbf{v})$

$$e(\xi) = \sum_{k \in h} e_k \psi_k(\xi)$$

The basis expansion for the internal energy field is 1-order lower than the velocity basis expansion

$$\int_{w(t)} \rho \psi_q \frac{de}{dt} dw = \int_{w(t)} \psi_q \boldsymbol{\sigma} : \nabla \otimes \mathbf{v} \, dw$$

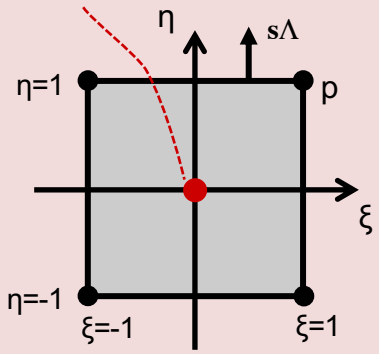
$$\left[\int_{w(t)} \rho \psi_q \psi_k dw \right] \frac{de_k}{dt} = \int_{w(t)} \psi_q \boldsymbol{\sigma} : \nabla \otimes \mathbf{v} \, dw$$

Dense, but local matrix

$$\mathbf{M}_{qk}^{(e)} \frac{de_k}{dt} = R_q^{(e)}$$

A side note:
the internal energy at a quadrature point is calculated by evaluating the energy field at the quadrature points $e_g = \sum_{k \in h} e_k \psi_k(\xi_g)$

Internal energy location for a linear quadrilateral element (constant)



Internal energy location for a quadratic quadrilateral element (bi-linear)

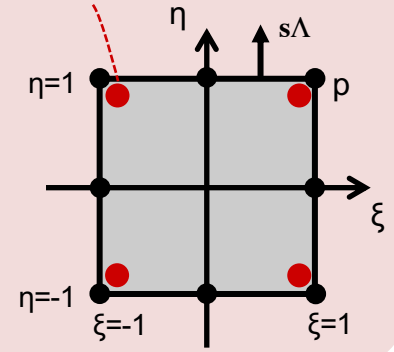


Fig: A 2D example

The weak-form for solving the internal energy equation (cont.)

- **The weak-form conserves total energy**

Case 1: linear element

$$\psi_0(\xi) = 1$$

$$\left[\int_{w(t)} \rho\psi_q\psi_k dw\right]\frac{de_k}{dt} = \int_{w(t)} \psi_q \boldsymbol{\sigma} : \nabla \otimes \mathbf{v}\, dw \quad \rightarrow \quad M_z \frac{e_z^{n+1} - e_z^n}{\Delta t} = \left(\int_{w(t)} \left(\boldsymbol{\sigma} : \nabla \otimes \mathbf{v}\right) dw\right)^{n+\frac{1}{2}}$$

Which is the compatible internal energy change for the cell if the velocity is the temporal average, please see earlier derivation

Case 2: Arbitrary order element

$$\mathbf{M}_{qk}^{(e)} \frac{de_k}{dt} = \left(\int_{w(t)} \left(\psi_q \boldsymbol{\sigma} : \nabla \otimes \mathbf{v}\right) dw\right) = \sum_{p \in h} \left(\left(\int_{w(t)} \psi_q \left(\nabla \phi_p\right) \cdot \boldsymbol{\sigma}\, dw\right) \cdot \mathbf{v}_p\right)$$

The force used in the momentum evolution equation must account for the energy basis, see [1] for details

1. V. Dobrev, T. Ellis, T. Kolev, and R. Rieben, *Curvilinear finite elements for Lagrangian hydrodynamics*, IJNMF, 2010

Stability on shock problems

Dissipation must be added to the stress tensor for stability on shock problems

- **Artificial viscosity approach**

$$M_p \frac{d\mathbf{v}_p}{dt} = -\sum_{h\in p}\int_{w(t)} \left(\nabla\phi_p\right)\cdot\left(\boldsymbol{\sigma}+\mathbf{q}\right)dw = \sum_{c\in p}\mathbf{F}_c$$

Artificial viscosity tensor

Sound speed

Scalar Q:

$$\Delta v = l\nabla\cdot\mathbf{v}$$

$$\mathbf{q} = \rho\left(b_0 c + b_1\left|\Delta v\right|\right)\left(l\nabla\cdot\mathbf{v}\right)\mathbf{I} = \mu\Delta v\mathbf{I}$$

$$U = b_0 c + b_1\left|\Delta v\right| \qquad \mu = \rho U$$

Shock velocity

Tensor Q:

$$\mathbf{q} = \rho\left(b_0 c + b_1\left|\Delta v\right|\right) l\nabla\otimes\mathbf{v}^{sym} = \mu l\nabla\otimes\mathbf{v}^{sym}$$

- **Riemann solver approach**

$$M_p \frac{d\mathbf{v}_p}{dt} = \sum_{i\in p} a_i\mathbf{n}_i\cdot\delta\boldsymbol{\sigma}_c^* - \sum_{h\in p}\int_{w(t)} \left(\nabla\phi_p\right)\cdot\boldsymbol{\sigma}\,dw = \sum_{c\in p}\mathbf{F}_c$$

Riemann solver

or

$$M_p \frac{d\mathbf{v}_p}{dt} = -\sum_{h\in p}\int_{w(t)} \left(\nabla\phi_p\right)\cdot\left(\boldsymbol{\sigma}+\delta\boldsymbol{\sigma}^*\right)dw = \sum_{c\in p}\mathbf{F}_c$$

Riemann solver

$$\mathbf{n}\cdot\delta\sigma = \mathbf{n}\cdot\left(\sigma^* - \sigma_0\right) = \rho_0 U\left(\mathbf{v}^* - \mathbf{v}_0\right)$$

$$U = b_0 c + b_1\left|\Delta\mathbf{v}\right|$$

Dissipation is essential for numerical stability near shocks

$$\frac{d\theta}{dt} \geq 0$$

Second law of thermodynamics – rate of change of the entropy must be greater than or equal to zero

Specific entropy change

$$T\frac{d\theta}{dt} = \frac{de}{dt} + p\frac{d\gamma}{dt} \qquad \text{gas}$$

$$T\frac{d\theta}{dt} = \frac{de}{dt} - \boldsymbol{\sigma} : \frac{d\mathbf{G}}{dt} \qquad \text{solid}$$

$$\rho_g T_g \left.\frac{d\theta}{dt}\right|_g = \rho_g \left.\frac{de}{dt}\right|_g - \boldsymbol{\sigma}_g : \rho_g \left.\frac{d\mathbf{G}}{dt}\right|_g$$

Specific internal energy evolution

$$\rho_g \left.\frac{de}{dt}\right|_g = \left(\boldsymbol{\sigma}_g + \mathbf{q}_g\right) : \left(\nabla \otimes \mathbf{v}\right)_g^{sym}$$

Specific velocity gradient evolution

$$\rho_g \left.\frac{d\mathbf{G}}{dt}\right|_g = \left(\nabla \otimes \mathbf{v}\right)_g$$

$$\boldsymbol{\sigma}_g : \rho_g \left.\frac{d\mathbf{G}}{dt}\right|_g = \boldsymbol{\sigma}_g : \left(\nabla \otimes \mathbf{v}\right)_g^{sym}$$

The discrete change in entropy at a quadrature point

$$\rho_g T_g \left.\frac{d\theta}{dt}\right|_g = \mathbf{q}_g : \left(\nabla \otimes \mathbf{v}\right)_g = \mu_g l \left(\nabla \otimes \mathbf{v}\right)_g^{sym} : \left(\nabla \otimes \mathbf{v}\right)_g \geq 0$$

$$\mathbf{q}_g = \mu_g l \left(\nabla \otimes \mathbf{v}\right)_g^{sym}$$

✔ Increases entropy

The canonical artificial viscosity in 1D is the first-order Riemann solver

- **Artificial viscosity**

$$q = \rho\left(b_0 c + b_1 |\Delta v|\right)\left(l\nabla\cdot\mathbf{v}\right) = \rho b_0 c(v_{p+} - v_{p-}) + \rho b_1 |\Delta v|(v_{p+} - v_{p-})$$

$$\Delta v = l\nabla\cdot\mathbf{v} \qquad \Delta x \frac{\Delta v}{\Delta x}$$

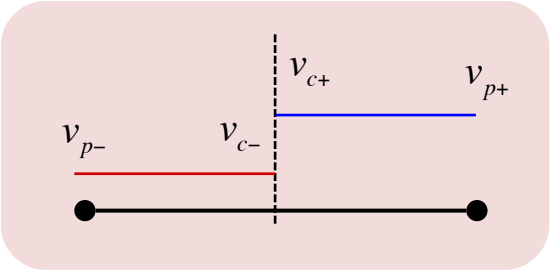


Fig: 1st-order Riemann solver assumes a constant velocity field

- **1D Riemann solver**

$$\delta\sigma = \left(\sigma_z^* - \sigma_c\right) = \mu\left(v_z^* - v_c\right) \qquad \mu = \rho U = \rho b_0 c + \rho b_1 |\Delta v|$$

Solve for Riemann velocity

$$v_z^* = \frac{\mu v_{p+} + \mu v_{p-}}{\mu + \mu} = \frac{v_{p+} + v_{p-}}{2}$$

$$\frac{v_{p+} + v_{p-}}{2} - v_{p-} = \frac{1}{2}\left(v_{p+} + v_{p-}\right)$$

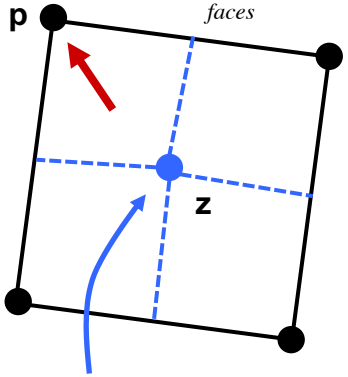
The jump in stress is then

$$\delta\sigma_z = \mu\left(\frac{v_{p+} + v_{p-}}{2} - v_{p-}\right) = \frac{1}{2}\mu\left(v_{p+} - v_{p-}\right) = \frac{1}{2}\rho b_0 c(v_{p+} - v_{p-}) + \frac{1}{2}\rho b_1 \left|\frac{1}{2}\Delta v\right|(v_{p+} - v_{p-})$$

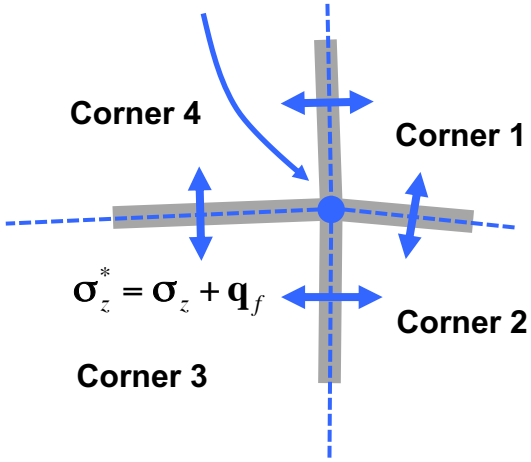
$$\delta\sigma_z = \frac{1}{2}\rho b_0 c(v_{p+} - v_{p-}) + \frac{1}{4}\rho b_1 |\Delta v|(v_{p+} - v_{p-})$$

A multidirectional approximate Riemann solution (MARS) seeks to apply dissipation in a more accurate manner

$$\mathbf{F}_c = \sum_{faces} \boldsymbol{\sigma}_c^* \cdot \mathbf{n}_f a_f$$



Dissipation is found by solving a multidirectional Riemann problem here



Dissipation: $\mathbf{q}_f = \mu_c \left(\mathbf{v}_z^* - \mathbf{v}_c \right) \left| \hat{\mathbf{e}}_c \cdot \mathbf{n}_f \right|$

$$\boldsymbol{\sigma}_z^* - \boldsymbol{\sigma}_z = \mathbf{q}$$

$$\mu_c \left(\mathbf{v}_z^* - \mathbf{v}_1 \right) \left| \hat{\mathbf{e}}_1 \cdot \mathbf{n}_{12} \right| = \left(\boldsymbol{\sigma}_1^* \cdot \mathbf{n}_{12} - \boldsymbol{\sigma}_z \cdot \mathbf{n}_{12} \right)$$

$$\mu_c \left(\mathbf{v}_z^* - \mathbf{v}_1 \right) \left| \hat{\mathbf{e}}_1 \cdot \mathbf{n}_{14} \right| = \left(\boldsymbol{\sigma}_1^* \cdot \mathbf{n}_{14} - \boldsymbol{\sigma}_z \cdot \mathbf{n}_{14} \right)$$

$$\mu_c \left(\mathbf{v}_z^* - \mathbf{v}_2 \right) \left| \hat{\mathbf{e}}_2 \cdot \mathbf{n}_{21} \right| = \left(\boldsymbol{\sigma}_2^* \cdot \mathbf{n}_{21} - \boldsymbol{\sigma}_z \cdot \mathbf{n}_{21} \right)$$

$$\mu_c \left(\mathbf{v}_z^* - \mathbf{v}_2 \right) \left| \hat{\mathbf{e}}_2 \cdot \mathbf{n}_{23} \right| = \left(\boldsymbol{\sigma}_2^* \cdot \mathbf{n}_{23} - \boldsymbol{\sigma}_z \cdot \mathbf{n}_{23} \right)$$

$$\mu_c \left(\mathbf{v}_z^* - \mathbf{v}_3 \right) \left| \hat{\mathbf{e}}_3 \cdot \mathbf{n}_{32} \right| = \left(\boldsymbol{\sigma}_3^* \cdot \mathbf{n}_{32} - \boldsymbol{\sigma}_z \cdot \mathbf{n}_{32} \right)$$

$$\mu_c \left(\mathbf{v}_z^* - \mathbf{v}_3 \right) \left| \hat{\mathbf{e}}_3 \cdot \mathbf{n}_{34} \right| = \left(\boldsymbol{\sigma}_3^* \cdot \mathbf{n}_{34} - \boldsymbol{\sigma}_z \cdot \mathbf{n}_{34} \right)$$

$$\mu_c \left(\mathbf{v}_z^* - \mathbf{v}_4 \right) \left| \hat{\mathbf{e}}_4 \cdot \mathbf{n}_{43} \right| = \left(\boldsymbol{\sigma}_4^* \cdot \mathbf{n}_{43} - \boldsymbol{\sigma}_z \cdot \mathbf{n}_{43} \right)$$

$$\mu_c \left(\mathbf{v}_z^* - \mathbf{v}_4 \right) \left| \hat{\mathbf{e}}_4 \cdot \mathbf{n}_{41} \right| = \left(\boldsymbol{\sigma}_4^* \cdot \mathbf{n}_{41} - \boldsymbol{\sigma}_z \cdot \mathbf{n}_{41} \right)$$

$$\sum_{faces} \mathbf{F} = \sum_{faces} \boldsymbol{\sigma}_c^* \cdot \mathbf{n}_f a_f = 0$$

$$\mathbf{u}_z^* = \frac{\sum_{f=1}^{8} \mu_c \left| \hat{\mathbf{e}}_c \cdot \mathbf{n}_f a_f \right| \mathbf{v}_c - \boldsymbol{\sigma}_z \cdot \mathbf{N}_f}{\sum_{f=1}^{8} \mu_c \left| \hat{\mathbf{e}}_c \cdot \mathbf{n}_f a_f \right|}$$

The momentum control volume faces are denoted with subscript f

$$\hat{\mathbf{e}}_c \equiv \frac{\mathbf{v}_p - \mathbf{v}_z}{\left\| \mathbf{v}_p - \mathbf{v}_z \right\|}$$

Unit vector for ensuring dissipation is in the direction of the velocity jump

$$\mathbf{v}_c = \mathbf{v}_p + \mathbf{r}_{pz} \cdot \bar{\nabla} \mathbf{v}_p \qquad 2^{\text{nd}}\text{-order}$$

1. N. Morgan, K. Lipnikov, D. Burton, and M. Kenamond, *A Lagrangian staggered grid Godunov-like approach for hydrodynamics*, J. Comp. Phys., 259 (2014) 568-597

Time integration

The central difference time integration approach solves the momentum and energy equations staggered in time

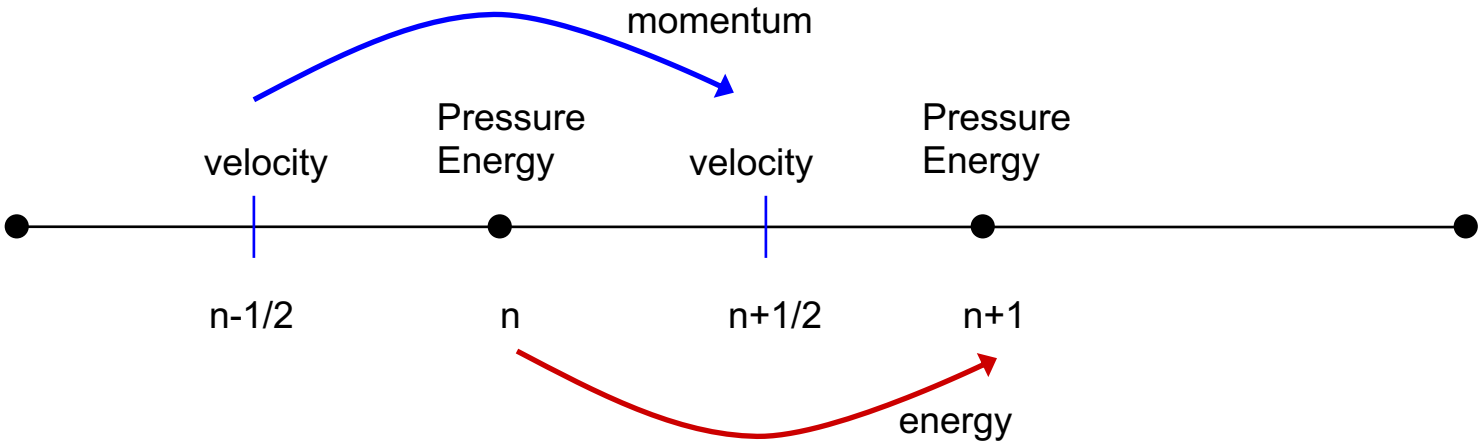
- **Staggered time integration**

Order of operation

(A) $$\mathbf{v}_p^{n+\frac{1}{2}} = \mathbf{v}_p^{n-\frac{1}{2}} + \frac{1}{M_P}\left(\frac{\Delta t^{n+\frac{1}{2}} + \Delta t^{n-\frac{1}{2}}}{2}\right)\sum_{c\in p}\mathbf{F}_c^n$$ Objective: get the velocity at t=n+1/2

(B) $$\mathbf{x}_p^{n+1} = \mathbf{x}_p^{n-\frac{1}{2}} + \Delta t^{n+\frac{1}{2}}\mathbf{v}_p^{n+\frac{1}{2}}$$ Objective: move the mesh to t=n+1

(C) $$e_z^{n+1} = e_z^n + \frac{\Delta t}{M_z}\sum_{c\in p}\mathbf{F}_c^{n+\frac{1}{2}}\cdot\mathbf{v}_p^{n+\frac{1}{2}}$$ Objective: get the energy at n+1



The collocated in time integration methods solve

- **A compatible 2-step RK time integration method**

(A) $\mathbf{x}^{n+\frac{1}{2}} = \mathbf{x}^n + \frac{1}{2}\Delta t \mathbf{v}^n$

$e_z^{n+\frac{1}{2}} = e_z^n - \frac{\Delta t}{2}\frac{1}{M_z}\sum_{c\in h}\left(\mathbf{F}_c \cdot \mathbf{v}_p\right)^n \qquad \tilde{\mathbf{v}}_p^{n+\frac{1}{2}} = \mathbf{v}_p^n + \frac{\Delta t}{2}\frac{1}{M_P}\sum_{c\in p}\mathbf{F}_c^n$

} Objective: get stress at n+1/2

Order of operation ↓

(B) $p^{n+\frac{1}{2}} = EOS\left(\rho^{n+\frac{1}{2}}, e^{n+\frac{1}{2}}\right) \qquad \boldsymbol{\sigma}_{dev}^{n+\frac{1}{2}} = \boldsymbol{\sigma}_{dev}^n + \frac{\Delta t}{2} f\left(\mathbf{L}^n\right) \qquad \mathbf{L}^n = \nabla \otimes \mathbf{v}^n$

EOS = equation of state; e.g., integrate Jaummann rate

(C) $\mathbf{v}^{n+1} = \mathbf{v}^n + \frac{\Delta t}{M_P}\sum_{c\in p}\mathbf{F}_c^{n+\frac{1}{2}}$ Objective: get the velocity at n+1

(D) $\mathbf{v}^{n+\frac{1}{2}} = \frac{\mathbf{v}^{n+1} + \mathbf{v}^n}{2}$ Objective: get the velocity at n+1/2

(E) $e_z^{n+1} = e_z^n - \frac{\Delta t}{M_z}\sum_{c\in h}\left(\mathbf{F}_c \cdot \mathbf{v}_p\right)^{n+\frac{1}{2}}$ Objective: get the energy at n+1

(F) $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t\ \mathbf{v}^{n+\frac{1}{2}}$ Objective: move the mesh to n+1

(G) $p^{n+1} = EOS\left(\rho^{n+1}, e^{n+1}\right) \qquad \boldsymbol{\sigma}_{dev}^{n+1} = \boldsymbol{\sigma}_{dev}^n + \Delta t\ f\left(\mathbf{L}^{n+\frac{1}{2}}\right) \qquad \mathbf{L}^{n+\frac{1}{2}} = \nabla \otimes \mathbf{v}^{n+\frac{1}{2}}$

The time step size must be chosen to ensure stability

- **Each time integration method places constraints on the time step size**
  - Caution: Some time integration methods are unconditionally unstable for some equations, which means there is no hope, but can be stable for other equations
- **The Courant-Friedrichs-Lewy number (CFL)**

$$CFL = \frac{c\Delta t}{\Delta x}$$

$CFL < 1$ For some explicit schemes, a value less than 1 is required. Much smaller values are also possible.

$$\Delta t = CFL\frac{\Delta x}{c}$$

Lagrangian methods (Fixed mesh)

$$\Delta t = CFL\frac{\Delta x}{c+|v|}$$

Eulerian methods (Fixed mesh)

- **The von Neumann number**

$$r = \frac{\mu_{visc}\Delta t}{\rho\Delta x^2}$$

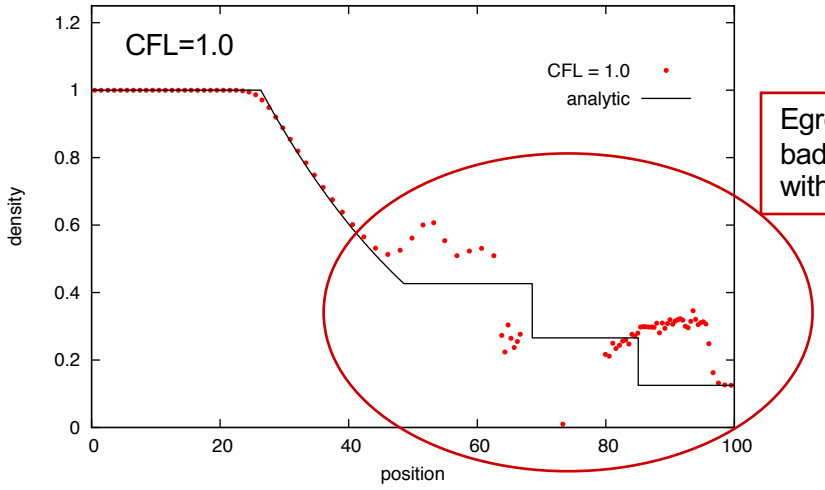
$$\Delta t = \frac{r\rho\Delta x^2}{\mu_{visc}}$$

For viscous (Navier-Stoke equations), the time step goes as the square of the mesh size with explicit time integration

Von Neumann stability analysis (or some other technique) **is required to prove** the stability of a time integration method with a particular spatial discretization.

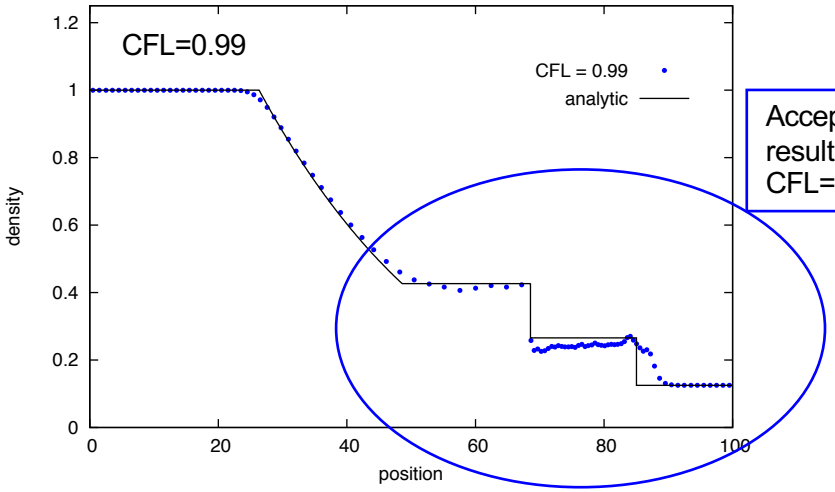
Do not violate the time step stability conditions

- **Sod shock tube results as a function of the CFL number (see 1D code)**

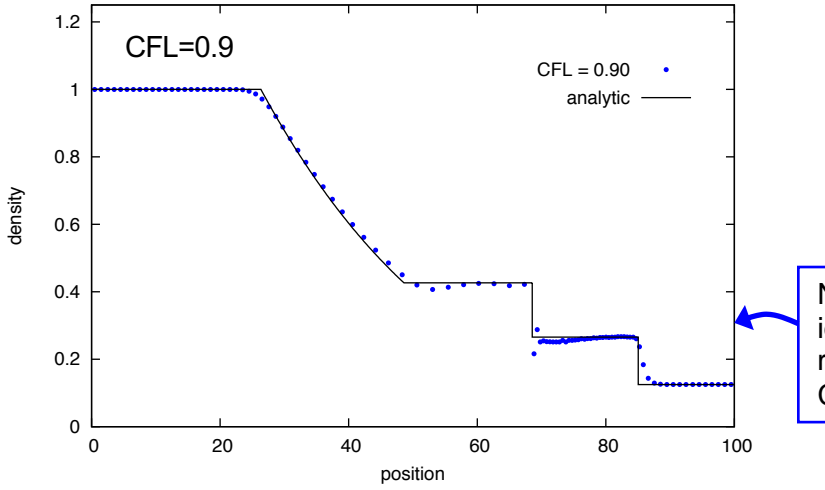


Egregiously bad results with CFL=1

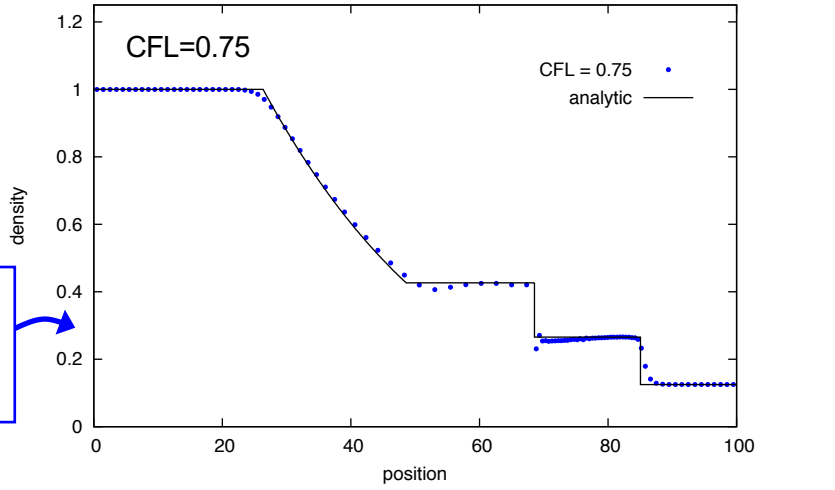
compare



Acceptable results with CFL=0.99



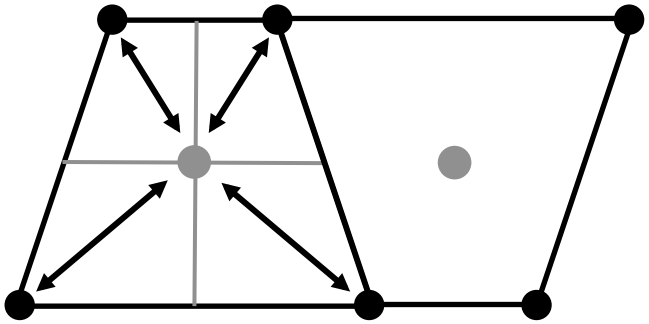
Nearly identical results for CFL<0.9



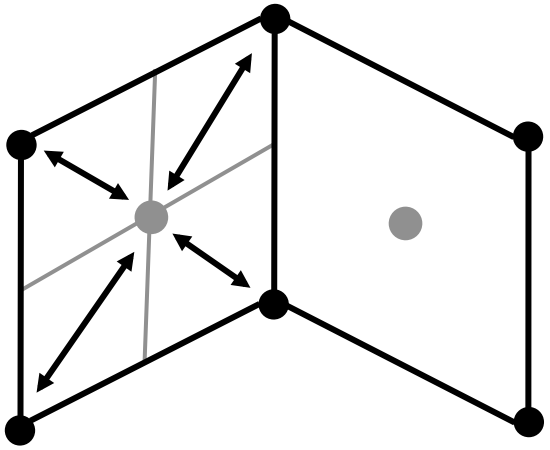
Mesh stability

The mesh can deform in unstable ways

- **With a single integration point, the cell can deform such that the volume never changes – termed zero-energy modes or null modes**



Hourglass null mode



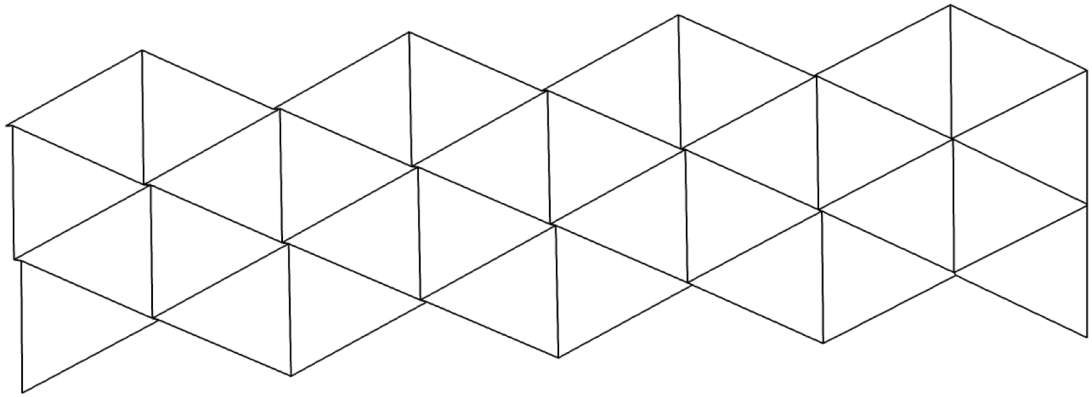
Chevron null mode

- **Increasing the number of quadrature points resolves the null mode issue, but it can make the mesh artificially resist physical deformation**
- **Mesh stability models can be added to resist null modes, but they also can prevent physical deformation**
- **The MARS method ensures stable motion with one quadrature point**

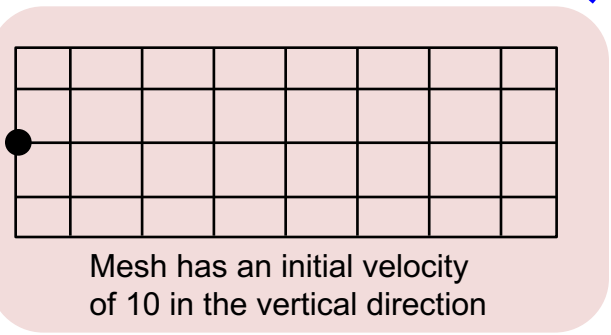
Mesh stability is important for accurate calculations

- **Bending beam test problem results [1]**

Bad results without a mesh stabilization method



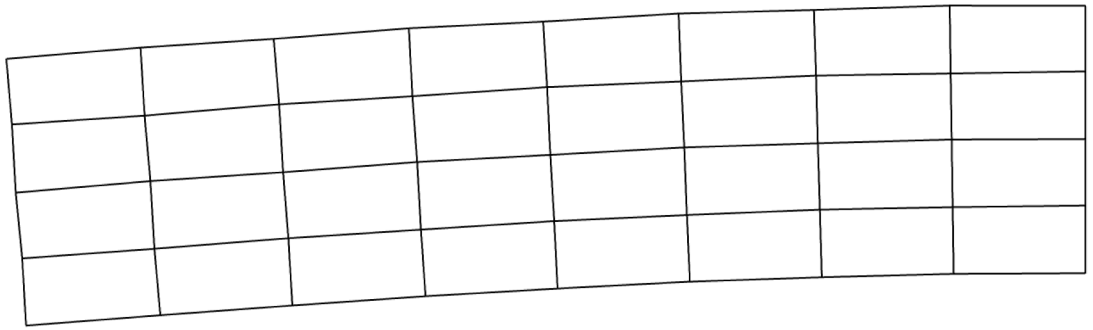
The hourglass null mode is excited when the mesh bends



Mesh has an initial velocity of 10 in the vertical direction

Fig: Initial mesh for the test problem

Great results using MARS, which intrinsically damps null modes



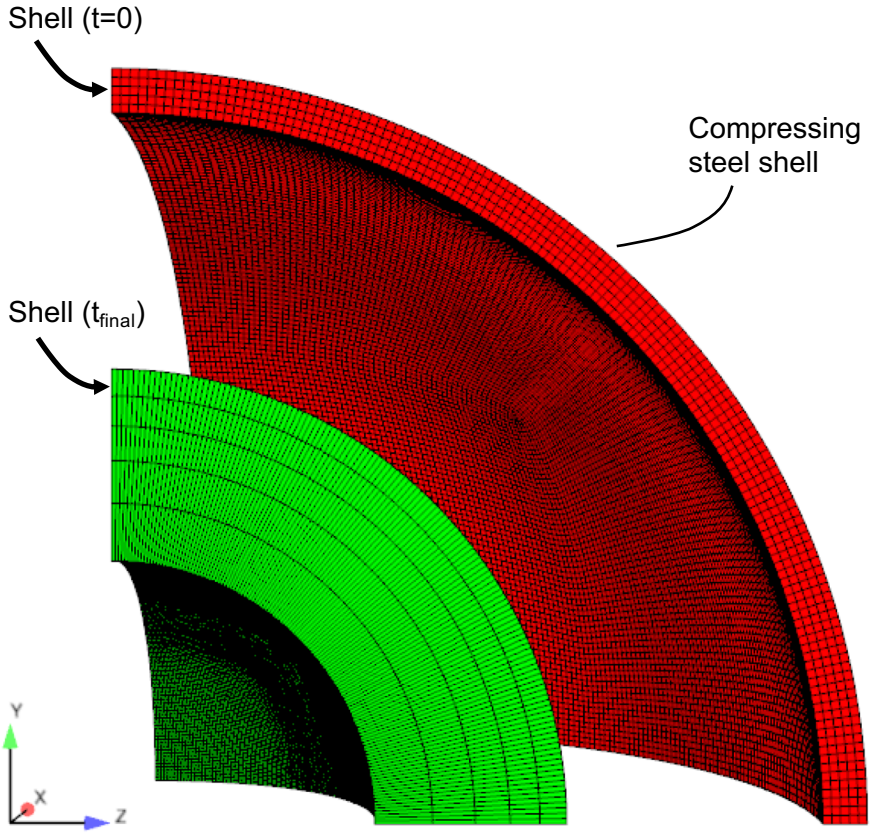
Using MARS ensures stable and accurate mesh deformation

1. D. Flanagan and T. Belytchko, *A uniform strain hexahedron and quadrilateral with orthogonal hourglass control*, IJNME, 1981; 17:679-706

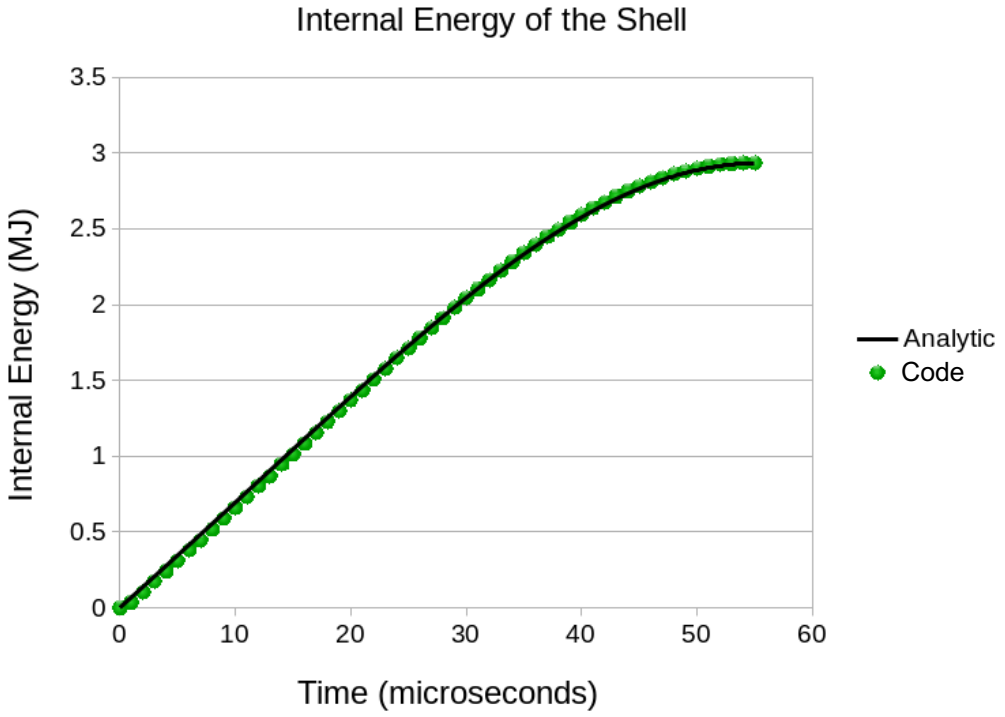
Examples

The finite element method with MARS is well suited for large deformation solid mechanics and shock driven flows

- **The 3D Verney test problem is an isentropic implosion of a metal shell**

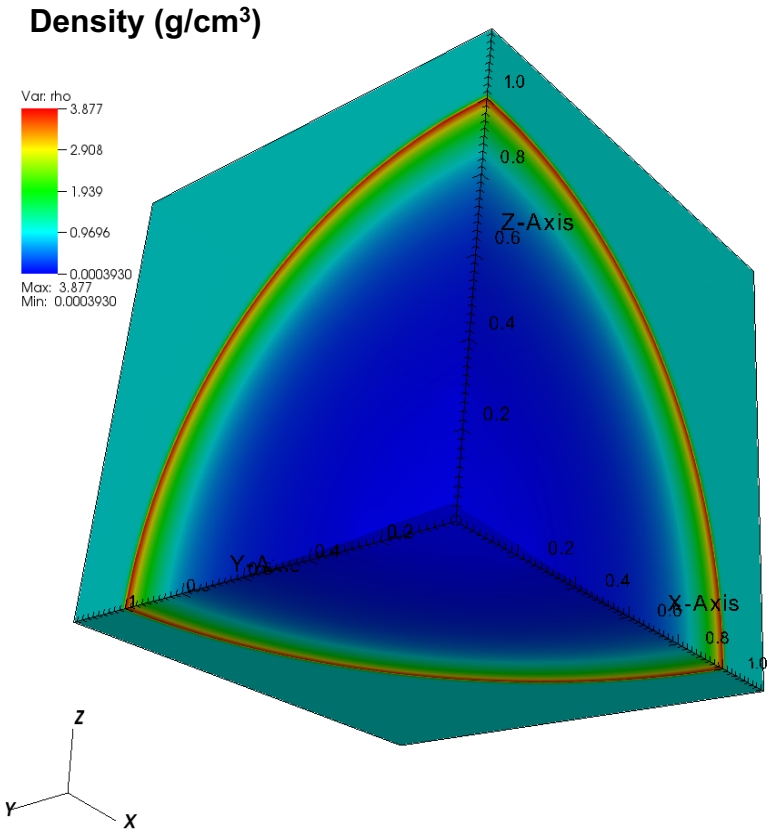


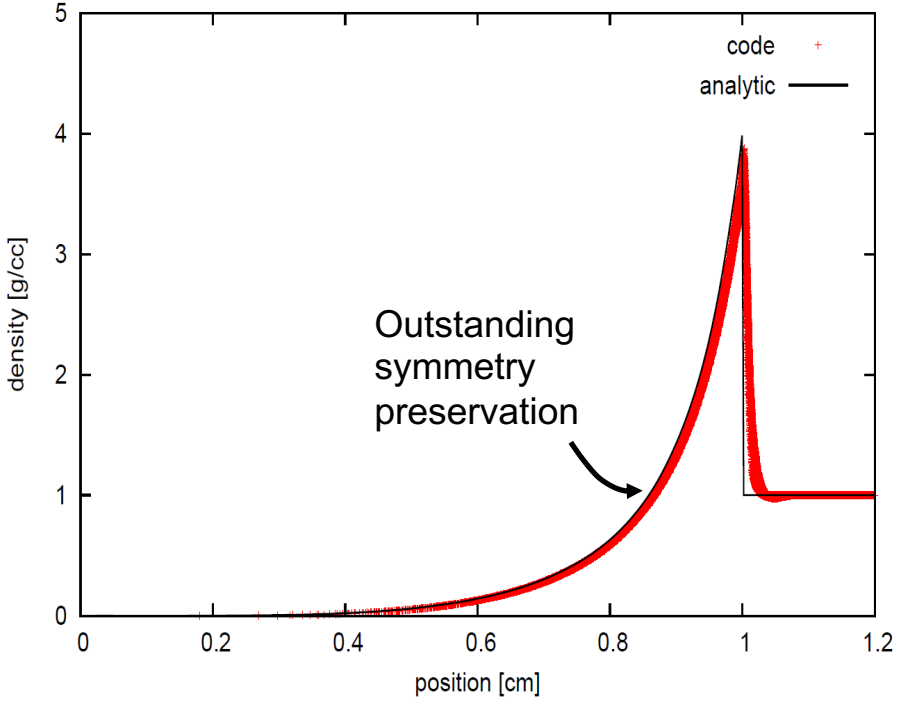
A 3D, unstructured mesh



The finite element method with MARS is well suited for large deformation solid mechanics and shock driven flows

- The Sedov test problem is blast wave in a gamma-law gas

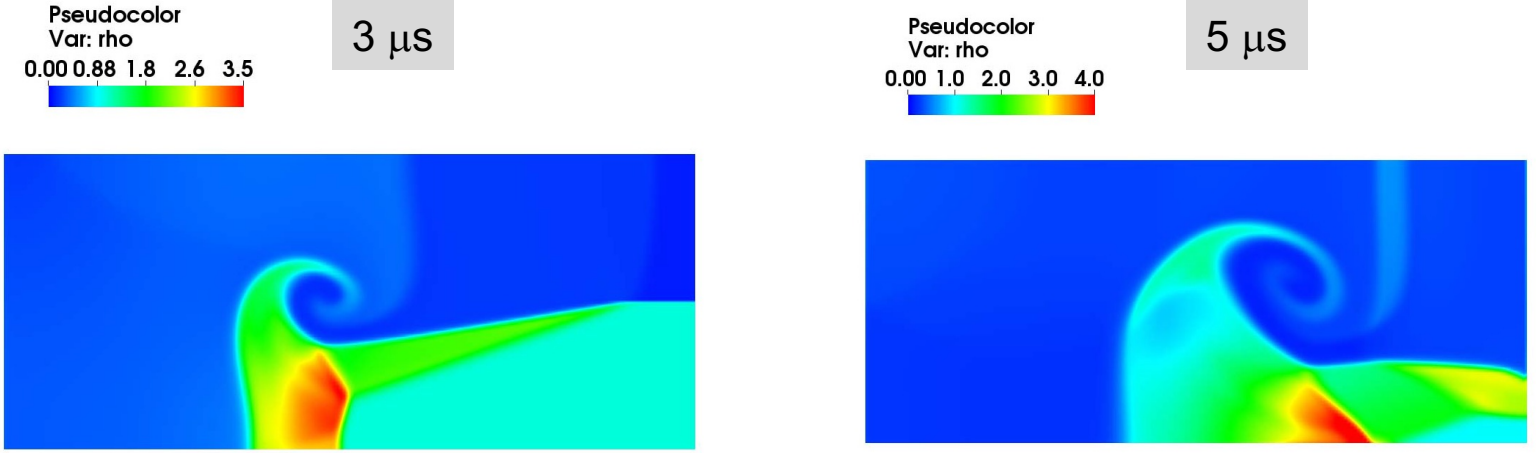




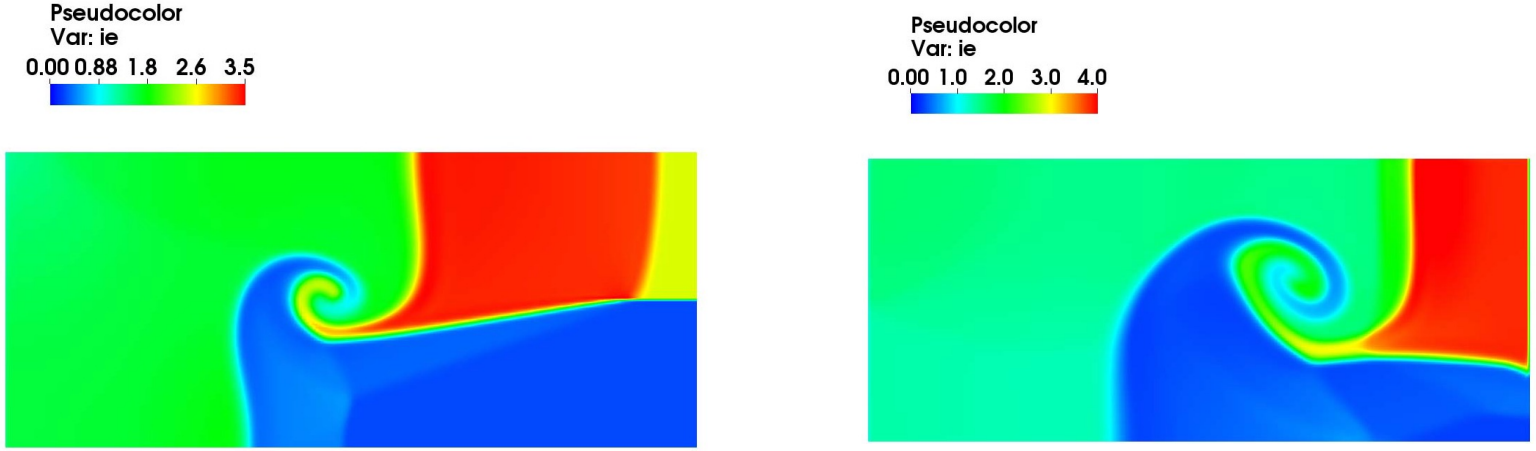
The finite element method with MARS is well suited for large deformation solid mechanics and shock driven flows

- **The Triple Point vortex problem results in the Eulerian limit**

Density



Internal Energy



These calculations used a remap on every step. A remap will interpolate the state to a new mesh after a Lagrangian step, which in this case, is the original one so it is an Eulerian calculation.

Summary

A summary

- **For a range of FE Lagrangian methods**
  - The density is evolved using a strong formulation that conserves mass at every point in space
  - The displacement field in a cell is represented with Lagrange interpolation polynomials that are defined in a reference coordinate system
    - The Lagrange basis functions satisfy the partition of unity - sum to 1 at any point in the cell
  - The velocity field over the cell is the time derivative of the displacement field
  - A variational formulation for the velocity evolution equation creates a global, sparse mass matrix.
    - The mass matrix is lumped
      - A lumping approach is to sum over all columns in a row of the mass matrix
      - A simple approach is to partition the cell mass to the nodes
      - Warning: challenges can arise with how the lumping is performed
  - The internal energy evolution equation is local to the element
    - A weak formulation, calculates an internal energy field for the element that is given by a Lagrange polynomial that is 1-order lower than the velocity field
    - A strong formulation, evolves internal energy at the quadrature points
  - Time integration can be spatially staggered or collocated

A 1D C++ lumped mass FE Lagrangian code

```
// ---------------------------------------------------------------------
//
//    This is a 1D c++ finite element code for gas/solid dynamics
//
//    Written by: Nathaniel Morgan
//      July 31, 2018
//
//    To compile the code, type in the command line
//         g++ -std=c++14 *cpp
//
//    To run the code, type in the command line
//         ./a.out
//
// ---------------------------------------------------------------------


#include <stdio.h>
#include <iostream> // std::cout
#include <fstream>
#include <vector>
#include <algorithm>
#include <cmath>    // std::abs,

using namespace std;

// ---------------------------------------------------------------------
//    Global variables
// ---------------------------------------------------------------------
double fuzz    = 1.0E-16;
double huge    = 1.0E16;
```

```
// ---------------------------------------------------------------------------
//    The state and eos variables
// ---------------------------------------------------------------------------
struct cell_vars_t{

  double den;      // density
  double pres;     // pressure
  double sspd;     // sound speed
  double sie;      // specific internal energy
  double sie_n;    // specific internal energy at t_n

  double visc;     // viscous dissipation
  double visc_HO;  // high-order viscous dissipation

  double mass;     // mass of the cell
  double workrate; // righthand side of energy eqn

  double force[2]; // force in the cell corners

  double gamma;    // equation of state
};


struct node_vars_t{
  double vel;      // velocity
  double vel_n;    // the velocity at t_n

  double mass;     // mass of node
  double force;    // righthand side of momentum eqn
};
```

```
// ---------------------------------------------------------------------------
//    The mesh
// ---------------------------------------------------------------------------
struct mesh_cells_t{
   double coords;
   double vol;      // volume of the cell
};


struct mesh_nodes_t{
   double coords;   // coordinates
   double coords_n; // coordinates at t_n
};


/ ---------------------------------------------------------------------------
//    A region
// ---------------------------------------------------------------------------
struct region_t{
   double x_min;
   double x_max;
   double den;
   double sie;
   double vel;
   double gamma;
};
```

```
// ---------------------------------------------------------------------------
//    The Main function
// ---------------------------------------------------------------------------
int main(){


   FILE * myfile;


   // --------------------
   // user settable values
   // --------------------

   // global
   double time     = 0.0;
   double sspd_min = 1.0E-3;

   // time step settings
   double time_max = 20.0;
   double dt       = 0.01;
   double dt_max   = 100;
   double dt_cfl   = 0.3;
   int rk_num_stages = 2;

   // mesh information
   double x_min = 0.0;
   double x_max = 100.0;
   double cell_num = 5000;
```

```
// intial conditions for each region

  // Sod
  vector<region_t> ics(2);

  ics[0].x_min = 0.0;
  ics[0].x_max = 50.0;
  ics[0].den   = 1.0;
  ics[0].sie   = 2.5;
  ics[0].vel   = 0.0;
  ics[0].gamma = 1.4;

  ics[1].x_min = 50.0;
  ics[1].x_max = 100.0;
  ics[1].den   = 0.125;
  ics[1].sie   = 2.0;
  ics[1].vel   = 0.0;
  ics[1].gamma = 1.4;
```

```
// ---------------------

  // 1D element routine
  vector<double> grad_basis = {-1.0, 1.0};

  // create the mesh
  int node_num = cell_num+1;
  double dx = (x_max-x_min)/cell_num;
  vector<mesh_cells_t> mesh_cells(cell_num);
  vector<mesh_nodes_t> mesh_nodes(node_num);

  // calculate coordinates
  for (int node_id=0; node_id<node_num; node_id++)
  {
    mesh_nodes[node_id].coords = double(node_id) * dx;

    if (node_id>0)
      mesh_cells[node_id-1].coords =
        (mesh_nodes[node_id-1].coords + mesh_nodes[node_id].coords)/2.0;


  } // end for
```

```
// initialize the state on the mesh
  vector<cell_vars_t> cell_vars(cell_num);
  vector<node_vars_t> node_vars(node_num);

  // number of regions
  int reg_num = ics.size();

  // initialize node mass to zero
  for (int node_id=0; node_id<node_num; node_id++) node_vars[node_id].mass = 0.0;
```

```
 // loop over the cells
for (int cell_id=0; cell_id<cell_num; cell_id++){

  // loop over the regions
  for (int reg=0; reg<reg_num; reg++){


    if (mesh_cells[cell_id].coords >= ics[reg].x_min &&
        mesh_cells[cell_id].coords <= ics[reg].x_max){

      cell_vars[cell_id].den   = ics[reg].den;
      cell_vars[cell_id].sie   = ics[reg].sie;
      cell_vars[cell_id].gamma = ics[reg].gamma;

      cell_vars[cell_id].pres  =
          cell_vars[cell_id].den*cell_vars[cell_id].sie*(cell_vars[cell_id].gamma - 1.0);

      cell_vars[cell_id].sspd  =
          sqrt(cell_vars[cell_id].gamma*cell_vars[cell_id].pres/cell_vars[cell_id].den);

      node_vars[cell_id].vel   = ics[reg].vel;
      node_vars[cell_id+1].vel = ics[reg].vel;

      mesh_cells[cell_id].vol  = mesh_nodes[cell_id+1].coords - mesh_nodes[cell_id].coords;
      cell_vars[cell_id].mass  = ics[reg].den*mesh_cells[cell_id].vol;

      node_vars[cell_id].mass   += ics[reg].den*mesh_cells[cell_id].vol/2.0;
      node_vars[cell_id+1].mass += ics[reg].den*mesh_cells[cell_id].vol/2.0;
    } // end if

  } // end for

} // end for
```

```
// write out the intial conditions to a file
  myfile=fopen("time0.txt","w");
  fprintf(myfile,"# x  den  pres  sie vel \n");

  for (int cell_id=0; cell_id<cell_num; cell_id++)
    fprintf(myfile,"%f\t%f\t%f\t%f\t%f\n",
          mesh_cells[cell_id].coords,
          cell_vars[cell_id].den,
          cell_vars[cell_id].pres,
          cell_vars[cell_id].sie,
          0.5*(node_vars[cell_id].vel+node_vars[cell_id+1].vel));
  fclose(myfile);



  // total energy check
  double total_e = 0.0;
  for (int cell_id=0; cell_id<cell_num; cell_id++){
    total_e += cell_vars[cell_id].mass*cell_vars[cell_id].sie +
            0.5*cell_vars[cell_id].mass*0.5*pow(node_vars[cell_id].vel, 2) +
            0.5*cell_vars[cell_id].mass*0.5*pow(node_vars[cell_id+1].vel, 2);
  }
```

```
// ------------------------------------
// Solve equations until time=time_max
// ------------------------------------
while (time <= time_max){

  cout << "time = " << time <<endl;


  // get the new time step
  dt = huge;
  for (int cell_id=0; cell_id<cell_num; cell_id++){
    dx = mesh_nodes[cell_id+1].coords - mesh_nodes[cell_id].coords;
    dt = min(dt,dt_cfl*dx/(cell_vars[cell_id].sspd + fuzz));
    dt = min(dt,dt_max);
    dt = min(dt, time_max-time);
  } // end for

  if (dt<=fuzz) break;
```

```
// integrate solution forward in time
for (int rk_stage=0; rk_stage<rk_num_stages; rk_stage++ ){

  // rk coefficient on dt
  double rk_alpha = 1.0/(double(rk_num_stages) - double(rk_stage));


  // initialize work rate to zero
  for (int cell_id=0; cell_id<cell_num; cell_id++){
    cell_vars[cell_id].workrate = 0.0;

    // save the t_n specific internal energy
    if (rk_stage==0) cell_vars[cell_id].sie_n = cell_vars[cell_id].sie;
  }


  // initialize the forces to zero and new velocity
  for (int node_id=0; node_id<node_num; node_id++){
    node_vars[node_id].force = 0.0;

    // save the t_n velocity
    if (rk_stage==0){
      node_vars[node_id].vel_n     = node_vars[node_id].vel;
      mesh_nodes[node_id].coords_n = mesh_nodes[node_id].coords;
    }
  } // end for
```

```
// loop over the mesh and calculate the forces
// force is calculated with single point quadrature
for (int cell_id=0; cell_id<cell_num; cell_id++){

  // solve Riemann problem in compression
  if (node_vars[cell_id+1].vel - node_vars[cell_id].vel < 0.0){

    // first-order dissipation from Riemann problem
    cell_vars[cell_id].visc =
       1.0/2.0*cell_vars[cell_id].den*cell_vars[cell_id].sspd*
            abs(node_vars[cell_id+1].vel - node_vars[cell_id].vel + 1.2/4.0*cell_vars[cell_id].den*pow( (node_vars[cell_id+1].vel - node_vars[cell_id].vel), 2.0);

    // higher-order dissipation, Order(h^2)
    cell_vars[cell_id].visc_HO = 0.0;

  }
  else{
    // no first-order dissipation in expansion
    cell_vars[cell_id].visc = 0.0;

    // higher-order dissipation in expansion, Order(h^2)
    cell_vars[cell_id].visc_HO = 0.0;
  }

  // apply the viscosity only in regions near a shock
  // visc_limited = alpha*visc where alpha=0 is smooth flow and alpha=1 is a shock
  // alpha = coef*abs(delta vel)/sound_speed
  //      set coeff < 1 (but > 0) to bias limiter towards high-order dissipation
  //      set coeff > 1 to bias limiter towards low-order dissipation
  double ratio = 20.0*abs(node_vars[cell_id+1].vel - node_vars[cell_id].vel)/(cell_vars[cell_id].sspd + fuzz);
  double alpha = max(0.0, min(1.0, ratio));

  // left corner force
  cell_vars[cell_id].force[0] = -grad_basis[0]*(-cell_vars[cell_id].pres
                                  -alpha*cell_vars[cell_id].visc
                                  -(1.0-alpha)*cell_vars[cell_id].visc_HO);

  // right corner force
  cell_vars[cell_id].force[1] = -grad_basis[1]*(-cell_vars[cell_id].pres
                                  -alpha*cell_vars[cell_id].visc
                                  -(1.0-alpha)*cell_vars[cell_id].visc_HO);

} // end for
```

```
 // v_new = v_n + alpha*dt/mass*Sum(forces)
for (int node_id=0; node_id<node_num; node_id++){

  // scatter the force contributions
  node_vars[node_id].force += cell_vars[node_id-1].force[1]; // from left corner
  node_vars[node_id].force += cell_vars[node_id].force[0];   // from right corner

  // update vel
  node_vars[node_id].vel = node_vars[node_id].vel_n +
       rk_alpha*dt/node_vars[node_id].mass*node_vars[node_id].force;


  // applying a wall BC
  if (node_id==0) node_vars[node_id].vel = 0.0;
  if (node_id==node_num-1) node_vars[node_id].vel = 0.0;


  // update mesh coordinates
  mesh_nodes[node_id].coords = mesh_nodes[node_id].coords_n +
       rk_alpha*dt*(node_vars[node_id].vel_n + node_vars[node_id].vel)/2.0;

} // end for
```

```
// e_new = e_n + alpha*dt/mass*Sum(forces*vel)
for (int cell_id=0; cell_id<cell_num; cell_id++){

  // workrate contribution from the left corner
  cell_vars[cell_id].workrate -= cell_vars[cell_id].force[0]*
                          (node_vars[cell_id].vel_n + node_vars[cell_id].vel)/2.0;

  // workrate contribution from the right corner
  cell_vars[cell_id].workrate -= cell_vars[cell_id].force[1]*
                          (node_vars[cell_id+1].vel_n + node_vars[cell_id+1].vel)/2.0;

  // update specific interal energy
  cell_vars[cell_id].sie = cell_vars[cell_id].sie_n +
    rk_alpha*dt/cell_vars[cell_id].mass*cell_vars[cell_id].workrate;

  if (cell_vars[cell_id].sie < fuzz)
    cout << cell_vars[cell_id].workrate << endl;
```

```
        // update vol
        mesh_cells[cell_id].vol  = mesh_nodes[cell_id+1].coords - mesh_nodes[cell_id].coords;

        // update coordinates
        mesh_cells[cell_id].coords = (mesh_nodes[cell_id+1].coords + mesh_nodes[cell_id].coords)/2.0;

        // update density
        cell_vars[cell_id].den = cell_vars[cell_id].mass/mesh_cells[cell_id].vol;

        // update pressure
        cell_vars[cell_id].pres = cell_vars[cell_id].den*cell_vars[cell_id].sie*(cell_vars[cell_id].gamma - 1.0);

        // update sound speed
        cell_vars[cell_id].sspd = sqrt(cell_vars[cell_id].gamma*cell_vars[cell_id].pres/cell_vars[cell_id].den);
        cell_vars[cell_id].sspd = max(cell_vars[cell_id].sspd, sspd_min);
      } // end for

    } // end for



    // update the time
    time += dt;
    if (abs(time-time_max)<=fuzz) time=time_max;

} // end of the
```

```
  myfile=fopen("timeEnd.txt","w");
  fprintf(myfile,"# x  den  pres  sie vel \n");


  for (int cell_id=0; cell_id<cell_num; cell_id++)
    fprintf(myfile,"%f\t%f\t%f\t%f\t%f\n",
      mesh_cells[cell_id].coords,
      cell_vars[cell_id].den,
      cell_vars[cell_id].pres,
      cell_vars[cell_id].sie,
      0.5*(node_vars[cell_id].vel+node_vars[cell_id+1].vel));
  fclose(myfile);


  // total energy check
  double total_e_final = 0;
  for (int cell_id=0; cell_id<cell_num; cell_id++){
    total_e_final += cell_vars[cell_id].mass*cell_vars[cell_id].sie +
                  0.5*cell_vars[cell_id].mass*0.5*pow(node_vars[cell_id].vel, 2) +
                  0.5*cell_vars[cell_id].mass*0.5*pow(node_vars[cell_id+1].vel, 2);
  }
  cout << "total energy, t=0: " << total_e
       << " , t=final: "        << total_e_final
       << " , error = " << total_e - total_e_final << endl;


  return 0;

} // end main function
```