Generalized Interpolation Material Point (GIMP) Method

John A. Nairn

December 5, 2023

Contents

Chapter 1

GIMP Derivation

1.1 Introduction

These notes provide a detailed derivation of the GIMP method for MPM, which was first described by Bardenhagen and Kober. The notes pay attention to large deformation. This chapter uses Cartesian coordinates; subsequent chapters extend to axisymmetry and give 1D equations. Finally, another chapter derives GIMP methods for implementing transport equations.

1.2 Virtual Work or Power

MPM implementation is based on virtual work equation, but in Sulsky's original paper, it was derived from momentum equation using the divergence theorem (which is also way to derive virtual work equation). For completeness, both are give here. The momentum equation is:

$$\nabla \cdot \boldsymbol{\sigma} + \rho \boldsymbol{b} = \rho \boldsymbol{a} \tag{1.1}$$

where $\boldsymbol{\sigma}$ is Cauchy stress, ρ is density, $\boldsymbol{b}$ is body force, and $\boldsymbol{a}$ is acceleration. To solve in the weak form, multiply by weighting vector $\boldsymbol{w}$ and integrate over domain Ω:

$$\int_\Omega \boldsymbol{w} \cdot \nabla \cdot \boldsymbol{\sigma}\, dV + \int_\Omega \rho \boldsymbol{w} \cdot \boldsymbol{b}\, dV = \int_\Omega \rho \boldsymbol{w} \cdot \boldsymbol{a}\, dV \tag{1.2}$$

The first term expands to

$$\boldsymbol{w} \cdot \nabla \cdot \boldsymbol{\sigma} = \nabla \cdot (\boldsymbol{\sigma} \boldsymbol{w}) - \mathrm{Tr}(\boldsymbol{\sigma} \nabla \boldsymbol{w}) = \nabla \cdot (\boldsymbol{\sigma} \boldsymbol{w}) - \boldsymbol{\sigma} \cdot \nabla \boldsymbol{w} \tag{1.3}$$

and by divergence theorem:

$$\int_\Omega \nabla \cdot (\boldsymbol{\sigma} \boldsymbol{w})\, dV = \int_{\delta\Omega} \boldsymbol{T} \cdot \boldsymbol{w}\, dS \tag{1.4}$$

where $\boldsymbol{T}$ is traction due to the stress on surface ($\delta\Omega$) of the domain Ω. Substitution to eliminate the divergence term gives:

$$\int_\Omega \rho \boldsymbol{w} \cdot \boldsymbol{b}\, dV + \int_{\delta\Omega} \boldsymbol{T} \cdot \boldsymbol{w}\, dS = \int_\Omega \rho \boldsymbol{w} \cdot \boldsymbol{a}\, dV + \int_\Omega \boldsymbol{\sigma} \cdot \nabla \boldsymbol{w}\, dV \tag{1.5}$$

But the above results is just virtual work. We replace $\boldsymbol{w}$ by a virtual displacement ($\delta \boldsymbol{u}$) and add a virtual work due to force applied directly to the particle. The virtual work (or power) gives the following starting equation:

$$\int_V \rho \boldsymbol{b} \cdot \delta \boldsymbol{u}\, dV + \int_{S_T} \boldsymbol{T} \cdot \delta \boldsymbol{u}\, dS + \sum_p \boldsymbol{F_p} \cdot \delta \boldsymbol{u} = \int_V \rho \boldsymbol{a} \cdot \delta \boldsymbol{u}\, dV + \int_V \boldsymbol{\sigma} \cdot \nabla \delta \boldsymbol{u}\, dV \tag{1.6}$$

where $\boldsymbol{b}$ is specific body force (*e.g.*, gravity), $\boldsymbol{T}$ is traction applied on the surface S_T, $\boldsymbol{F_p}$ is a force applied directly to a particle (an extra turn added for MPM modeling), $\boldsymbol{a}$ is acceleration, ρ is density, and $\boldsymbol{\sigma}$ is the Cauchy stress.

1.3 Particle Basis Expansion

First, we expand $\rho \boldsymbol{b}$, $\rho \boldsymbol{a}$, and $\boldsymbol{\sigma}$ in a particle basis, where any function of $\boldsymbol{x}$ is written as:

$$f(\boldsymbol{x}) = \sum_p f_p \chi_p(\boldsymbol{x}) \tag{1.7}$$

where f_p is the value of the function on particle p and $\chi_p(\boldsymbol{x})$ is a particle basis function. The function integrates to total particle volume:

$$V_p = \int_V \chi_p(\boldsymbol{x})\, dV \tag{1.8}$$

The most common function is for $\chi_p(\boldsymbol{x}) = 1$ inside the particle domain and 0 outside the domain, but any other function could be used (hence the using of the term "Generalized"). The main particle property expansions are:

$$\rho \boldsymbol{b} = \sum_p \rho_p \boldsymbol{b}_p \chi_p(\boldsymbol{x}) = \sum_p \frac{m_p}{V_p} \boldsymbol{b}_p \chi_p(\boldsymbol{x}) \tag{1.9}$$

$$\rho \boldsymbol{a} = \sum_p \frac{m_p}{V_p} \frac{d\boldsymbol{v_p}}{dt} \chi_p(\boldsymbol{x}) = \sum_p \frac{\dot{\boldsymbol{p}}_{\boldsymbol{p}}}{V_p} \chi_p(\boldsymbol{x}) \tag{1.10}$$

$$\boldsymbol{\sigma} = \sum_p \boldsymbol{\sigma}_p \chi_p(\boldsymbol{x}) \tag{1.11}$$

where m_p is particle mass, V_p is current particle volume, $\boldsymbol{v_p}$ is particle velocity, $\boldsymbol{p_p}$ is particle momentum, and $\boldsymbol{\sigma}_p$ is particle stress.

To fit better in the analysis, the external work caused by particle forces can be expanded as work per unit volume spread out over the particle using

$$\boldsymbol{F_p} \cdot \delta \boldsymbol{u} = \int_V \frac{\boldsymbol{F_p} \cdot \delta \boldsymbol{u}}{V_p} \chi_F(\boldsymbol{x})\, dV \tag{1.12}$$

where $\chi_F(\boldsymbol{x})$ is another particle function. It need not be the same as $\chi_p(\boldsymbol{x})$, but must also integrate to particle volume, V_p

The new virtual work equation becomes:

$$\sum_p \int_V \frac{m_p}{V_p} \chi_p(\boldsymbol{x}) \boldsymbol{b}_p \cdot \delta \boldsymbol{u}\, dV + \int_{S_T} \boldsymbol{T} \cdot \delta \boldsymbol{u}\, dS + \sum_p \frac{\boldsymbol{F}_p}{V_p} \int_V \chi_F(\boldsymbol{x}) \cdot \delta \boldsymbol{u}\, dV \tag{1.13}$$

$$= \sum_p \int_V \frac{\dot{\boldsymbol{p}}_{\boldsymbol{p}}}{V_p} \chi_p(\boldsymbol{x}) \cdot \delta \boldsymbol{u}\, dV + \sum_p \int_V \boldsymbol{\sigma}_p \chi_p(\boldsymbol{x}) \cdot \nabla \delta \boldsymbol{u}\, dV \tag{1.14}$$

1.4 Grid Expansion

Next, we expand the virtual displacements in grid-based shape functions using:

$$\delta \boldsymbol{u} = \sum_i \delta \boldsymbol{u_i} N_i(\boldsymbol{x}) \tag{1.15}$$

where the sum is over nodes (i), $\delta \boldsymbol{u_i}$ is the virtual displacement on node i, and $N_i(\boldsymbol{x})$ are the nodal shape functions (standard finite element shape functions). Each term is revised as follows:

$$\begin{aligned}
\sum_i \sum_p \int_V \frac{m_p}{V_p} \chi_p(\boldsymbol{x}) \boldsymbol{b}_p \cdot \delta \boldsymbol{u_i} N_i(\boldsymbol{x})\, dV &= \sum_i \sum_p S_{ip} m_p \boldsymbol{b}_p \cdot \delta \boldsymbol{u_i} && (1.16)\\
\int_{S_T} \boldsymbol{T} \cdot \delta \boldsymbol{u} &= \sum_i \delta \boldsymbol{u_i} \cdot \int_{S_T} N_i(\boldsymbol{x}) \boldsymbol{T}\, dS && (1.17)\\
\sum_i \sum_p \frac{\boldsymbol{F}_p}{V_p} \int_V \chi_F(\boldsymbol{x}) \cdot \delta \boldsymbol{u_i} N_i(\boldsymbol{x})\, dV &= \sum_i \sum_p F_{ip} \boldsymbol{F}_p \cdot \delta \boldsymbol{u_i} && (1.18)\\
\sum_i \sum_p \int_V \frac{\dot{\boldsymbol{p}}_{\boldsymbol{p}}}{V_p} \chi_p(\boldsymbol{x}) \cdot \delta \boldsymbol{u_i} N_i(\boldsymbol{x})\, dV &= \sum_i \sum_p S_{ip} \dot{\boldsymbol{p}}_{\boldsymbol{p}} \cdot \delta \boldsymbol{u_i} && (1.19)\\
\sum_i \sum_p \int_V \boldsymbol{\sigma}_p \chi_p(\boldsymbol{x}) \cdot \left(\delta \boldsymbol{u_i} \otimes \nabla N_i(\boldsymbol{x})\right) dV &= \sum_i \sum_p \int_V \chi_p(\boldsymbol{x}) \sum_j \sum_k (\boldsymbol{\sigma}_p)_{jk} \delta u_{i,j} \nabla N_i(\boldsymbol{x})_k\, dV \\
&= \sum_i \sum_p \int_V \chi_p(\boldsymbol{x}) \sum_j (\boldsymbol{\sigma}_p \nabla N_i(\boldsymbol{x}))_j \delta u_{i,j}\, dV \\
&= \sum_i \sum_p (V_p \boldsymbol{\sigma}_p \boldsymbol{G}_{ip}) \cdot \delta \boldsymbol{u_i} && (1.20)
\end{aligned}$$

where the GIMP shape functions are:

$$\begin{aligned}
S_{ip} &= \frac{1}{V_p} \int_V \chi_p(\boldsymbol{x}) N_i(\boldsymbol{x})\, dV && (1.21)\\
F_{ip} &= \frac{1}{V_p} \int_V \chi_F(\boldsymbol{x}) N_i(\boldsymbol{x})\, dV && (1.22)\\
\boldsymbol{G}_{ip} &= \frac{1}{V_p} \int_V \chi_p(\boldsymbol{x}) \nabla N_i(\boldsymbol{x})\, dV && (1.23)
\end{aligned}$$

Two logical choices for $\chi_F(\boldsymbol{x})$ are $\chi_F(\boldsymbol{x}) = \chi_p(\boldsymbol{x})$, which leads to $F_{ip} = S_{ip}$ (and is the choice used below), or $\chi_F(\boldsymbol{x}) = V_p \delta(\boldsymbol{x} - \boldsymbol{x}_p)$, which leads to $F_{ip} = N_i(\boldsymbol{x}_p)$

The particle stress term can be revised to use particle mass as follows:

$$V_p \boldsymbol{\sigma}_p = m_p \frac{\boldsymbol{\sigma}_p}{\rho_p} = m_p \frac{\boldsymbol{\sigma}_p}{\rho_p} \frac{\rho_0}{\rho_0} = m_p \frac{J \sigma_p}{\rho_0} = m_p \frac{\tau_p}{\rho_0} \tag{1.24}$$

Here $J = V_p / V_0 = \rho_0 / \rho_p$ is the relative volume and $\boldsymbol{\tau}_p$ is the particle's Kirchhoff stress. In this form the stress term depends most naturally on Kirchhoff stresses normalized to initial density (and this approach avoids the need to find current particle density or volume when calculating that term on each time step). For this reason, `NairnMPM` always tracks stress as $\boldsymbol{\tau}/\rho_0$ (and all material models are expected to

calculate this stress). When archiving stress, NairnMPM writes Cauchy stress using $\boldsymbol{\sigma}_p = (\rho_0/J)*(\tau/\rho_0)$. Note that when doing energy calculations that $\boldsymbol{\tau}\cdot\mathrm{du}$, where du is incremental deformation gradient, is energy per unit initial volume and therefore the energy calculated in code using $(\tau/\rho_0)\cdot\mathrm{du}$ is energy per unit mass or $dU/(\rho_0 V_0)$.

1.5 MPM Equations

Making use of the fact that $\delta\boldsymbol{u_i}$ is arbitrary, the summand of all terms can be equated to arrive at the controlling MPM equation on the background grid of:

$$\sum_p S_{ip}^{(n)}\frac{d\boldsymbol{p_p^{(n)}}}{dt} = \frac{d\boldsymbol{p}_i^{(n)}}{dt} = \boldsymbol{f}_{i,int}^{(n)} + \boldsymbol{f}_{i,ext}^{(n)} \tag{1.25}$$

where

$$\boldsymbol{f}_{i,int}^{(n)} = \sum_p\left(-m_p\frac{\boldsymbol{\tau}_p^{(n)}\cdot\boldsymbol{G}_{ip}^{(n)}}{\rho_0}\right) \tag{1.26}$$

$$\boldsymbol{f}_{i,ext}^{(n)} = \boldsymbol{f}_{i,T}^{(n)} + \sum_p\left(m_p S_{ip}^{(n)}\boldsymbol{b}_p + \boldsymbol{F}_p^{(n)}S_{ip}^{(n)}\right) \tag{1.27}$$

$$\boldsymbol{f}_{i,T}^{(n)} = \int_{S_T}\boldsymbol{N}_i(\boldsymbol{x})\boldsymbol{T}\,dS \tag{1.28}$$

where $\boldsymbol{f}_{i,int}^{(n)}$ are internal forces due to stresses only and $\boldsymbol{f}_{i,ext}^{(n)}$ are external forces due to body forces, particle loads and traction forces ($\boldsymbol{f}_{i,T}^{(n)}$). Superscript (n) has been added to mean terms calculated from the state of all particles at the start of time step n. Commonly the body force, $\boldsymbol{b}_p$, will be independent of the particle state (*e.g.,* gravity, which is a constant, or a general body force that depends only on nodal position and time and not on particle state). When that holds, it can be removed from the sum. The total internal force then becomes

$$\boldsymbol{f}_{i,ext}^{(n)} = \boldsymbol{f}_{i,T}^{(n)} + m_i^{(n)}\boldsymbol{b}(\boldsymbol{x}_i,t) + \sum_p \boldsymbol{F}_p^{(n)}S_{ip}^{(n)} \tag{1.29}$$

where $\boldsymbol{b}(\boldsymbol{x}_i,t)$ is body force at the nodal location and

$$m_i^{(n)} = \sum_p m_p S_{ip}^{(n)} \tag{1.30}$$

is total nodal mass.

Once the forces are found, the momentum can be updated on the grid using

$$\boldsymbol{p}_i^{(n+1)} = \boldsymbol{p}_i^{(n)} + \boldsymbol{f}_i^{(n)}\Delta t \qquad \text{where} \qquad \boldsymbol{f}_i^{(n)} = \boldsymbol{f}_{i,int}^{(n)} + \boldsymbol{f}_{i,ext}^{(n)} \tag{1.31}$$

where Δt is the time step and:

$$\boldsymbol{p}_i^{(n)} = \sum_p \boldsymbol{p}_p^{(n)}S_{ip}^{(n)} \tag{1.32}$$

1.6 An Alternate Approach

The following section try to use spatial coordinates. I am not sure if it is correct.

1.6.1 Virtual Work or Power — An Alternate Approach

MPM implementation is based on virtual work equation, but in Sulsky's original paper, it was derived from momentum equation using the divergence theorem (which is also way to derive virtual work equation). For completeness, both are given here. Note that Sulsky (the MPM standard) uses the Cauchy equation of motion. That seems correct when MPM is cast as a "velocity" methods, but all (or most) MPM codes extrapolate momentum to the grid and end up updating momentum. It thus seems we should base MPM on a momentum equation in spatial coordinates. It is not easy to find this equation. So far, I found it in two places and they disagree on a sign. Here I am using a sign I can derive and matches the sign in the source I trust more (Ogden) text rather to equation I found on Wikipedia. Note that if I have a sign error, that would be easy to correct. The spatial momentum equation, or time derivative of displacement at fixed position in the deformed configuration, is:

$$\left(\frac{d\boldsymbol{p}}{dt}\right)_x = \nabla\cdot(\boldsymbol{\sigma}-\rho\boldsymbol{v}\otimes\boldsymbol{v})+\rho\boldsymbol{b} = \nabla\cdot\left(\boldsymbol{\sigma}-\frac{\boldsymbol{p}\otimes\boldsymbol{p}}{\rho}\right)+\rho\boldsymbol{b} \tag{1.33}$$

where $\boldsymbol{\sigma}$ is Cauchy stress, ρ is density, $\boldsymbol{v}$ is velocity, $\boldsymbol{b}$ is body force, and $\boldsymbol{p}=\rho\boldsymbol{v}$ is momentum. To solve in the weak form, multiply by weighting vector $\delta\boldsymbol{u}$ (which physcially is a displacement increment in virtual work methods) and integrate over domain Ω:

$$\int_\Omega \delta\boldsymbol{u}\cdot\nabla\cdot\left(\boldsymbol{\sigma}-\frac{\boldsymbol{p}\otimes\boldsymbol{p}}{\rho}\right)dV+\int_\Omega\rho\,\delta\boldsymbol{u}\cdot\boldsymbol{b}\,dV=\int_\Omega\delta\boldsymbol{u}\cdot\frac{d\boldsymbol{p}}{dt}\,dV \tag{1.34}$$

The stress term expands to

$$\delta\boldsymbol{u}\cdot\nabla\cdot\boldsymbol{\sigma}=\nabla\cdot(\boldsymbol{\sigma}\,\delta\boldsymbol{u})-\mathrm{Tr}(\boldsymbol{\sigma}\nabla\delta\boldsymbol{u})=\nabla\cdot(\boldsymbol{\sigma}\,\delta\boldsymbol{u})-\boldsymbol{\sigma}\cdot\nabla\delta\boldsymbol{u} \tag{1.35}$$

and by divergence theorem, the first time simplifies to:

$$\int_\Omega\nabla\cdot(\boldsymbol{\sigma}\,\delta\boldsymbol{u})\,dV=\int_{\delta\Omega}\boldsymbol{T}\cdot\delta\boldsymbol{u}\,dS \tag{1.36}$$

where $\boldsymbol{T}=\sigma\hat{\boldsymbol{n}}$ is traction due to the stress on surface ($\delta\Omega$) of the domain Ω. Notice the momentum divergence is not included here because it might not be possible to evaluate it on the boundary. The traction one, however, is done with traction boundary conditions (or zero if no applied tractions). Substitution to eliminate the stress divergence term gives:

$$\int_\Omega\rho\,\delta\boldsymbol{u}\cdot\boldsymbol{b}\,dV+\int_{\delta\Omega}\boldsymbol{T}\cdot\delta\boldsymbol{u}\,dS=\int_\Omega\delta\boldsymbol{u}\cdot\frac{d\boldsymbol{p}}{dt}\,dV+\int_\Omega\boldsymbol{\sigma}\cdot\nabla\delta\boldsymbol{u}\,dV+\int_\Omega\delta\boldsymbol{u}\cdot\nabla\cdot\left(\frac{\boldsymbol{p}\otimes\boldsymbol{p}}{\rho}\right)dV \tag{1.37}$$

1.6.2 Particle Basis Expansion — An Alternate Form

First, we expand $\rho\boldsymbol{b}$, $\boldsymbol{p}$, and $\boldsymbol{\sigma}$ in a particle basis, where any function of $\boldsymbol{x}$ is written as:

$$f(\boldsymbol{x})=\sum_p f_p\,\chi_p(\boldsymbol{x}) \tag{1.38}$$

where f_p is the value of the function on particle p and $\chi_p(\boldsymbol{x})$ is a particle basis function. The function integrates to total particle volume:

$$V_p=\int_V\chi_p(\boldsymbol{x})\,dV \tag{1.39}$$

The most common function is for $\chi_p(\boldsymbol{x}) = 1$ inside the particle domain and 0 outside the domain, but any other function could be used (hence the use of the term "Generalized"). The main particle property expansions are:

$$\rho \boldsymbol{b} = \sum_p \rho_p \boldsymbol{b}_p \chi_p(\boldsymbol{x}) = \sum_p \frac{m_p}{V_p} \boldsymbol{b}_p \chi_p(\boldsymbol{x}) \tag{1.40}$$

$$\frac{d\boldsymbol{p}}{dt} = \frac{d(\rho \boldsymbol{v})}{dt} = \sum_p \frac{m_p}{V_p} \frac{d\boldsymbol{v_p}}{dt} \chi_p(\boldsymbol{x}) = \sum_p \frac{\dot{\boldsymbol{p}}_{\boldsymbol{p}}}{V_p} \chi_p(\boldsymbol{x}) \tag{1.41}$$

$$\boldsymbol{\sigma} = \sum_p \boldsymbol{\sigma}_p \chi_p(\boldsymbol{x}) \tag{1.42}$$

$$\tag{1.43}$$

where m_p is particle mass, V_p is current particle volume, $\boldsymbol{v_p}$ is particle velocity, $\boldsymbol{p_p}$ is particle momentum, and $\boldsymbol{\sigma}_p$ is particle stress. The momentum divergence term is new and I am not certain how to proceed. Here is one option for Cartesian coordinates typically used in MPM (it would need to change for curvilinear coordinates, including axisymmetry). The divergence of a tensor in Cartesian coordinates is

$$(\nabla \cdot \mathsf{T})_i = \sum_j \frac{\partial T_{ij}}{\partial x_j}$$

When can then write

$$\left(\nabla \cdot (\rho \boldsymbol{v} \otimes \boldsymbol{v})\right)_i = \sum_j \frac{\partial (v_j v_i)}{\partial x_j} = \sum_j \left(\rho \left(v_i (\nabla v)_{jj} + (\nabla v)_{ij} v_j \right) + v_i v_j \frac{\partial \rho}{\partial x_j} \right)$$

which used $(\nabla v)_{ij} = \partial v_i / \partial x_j$. We can then write

$$\nabla \cdot (\rho \boldsymbol{v} \otimes \boldsymbol{v}) = \rho \left(\boldsymbol{v} \mathrm{Tr}(\nabla \boldsymbol{v}) + \nabla \boldsymbol{v} \boldsymbol{v} \right) + (\boldsymbol{v} \otimes \boldsymbol{v}) \nabla \rho = \rho \left(\boldsymbol{v} (\nabla \cdot \boldsymbol{v}) + (\boldsymbol{v} \cdot \nabla) \boldsymbol{v} \right) + (\boldsymbol{v} \otimes \boldsymbol{v}) \nabla \rho$$

The first form is easier to use in implementation when code is tracking $\boldsymbol{v}$ and $\nabla \boldsymbol{v}$, the later of which is calculated each time step and can be saved for the next time step. We then expand in particle basis using

$$\frac{\boldsymbol{p} \otimes \boldsymbol{p}}{\rho} = \rho \boldsymbol{v} \otimes \boldsymbol{v} = \sum_p \frac{m_p}{V_p} \left(\boldsymbol{v}_p \mathrm{Tr}(\nabla \boldsymbol{v}_p) + \nabla \boldsymbol{v}_p \boldsymbol{v}_p \right) \chi_p(\boldsymbol{x})$$

Note that in 1D, the summand is $\rho_p \partial v_p^2 / \partial x$ which looks like spatial derivative of (twice) the kinetic energy. I think the same applies to 3D as well.

An external work caused by forces applied at particle centers can be added here and expanded as work per unit volume spread out over the particle using

$$\boldsymbol{F_p} \cdot \delta \boldsymbol{u} = \int_V \frac{\boldsymbol{F_p} \cdot \delta \boldsymbol{u}}{V_p} \chi_F(\boldsymbol{x}) \, dV \tag{1.44}$$

where $\chi_F(\boldsymbol{x})$ is another particle function. It need not be the same as $\chi_p(\boldsymbol{x})$, but must also integrate to

particle volume, V_p. The new virtual work equation becomes:

$$\sum_p \int_V \frac{m_p}{V_p}\chi_p(\boldsymbol{x})\boldsymbol{b}_p \cdot \delta\boldsymbol{u}\, dV + \int_{S_T} \boldsymbol{T}\cdot\delta\boldsymbol{u}\, dS + \sum_p \frac{\boldsymbol{F}_p}{V_p}\int_V \chi_F(\boldsymbol{x})\cdot\delta\boldsymbol{u}\, dV \tag{1.45}$$

$$= \sum_p \int_V \frac{\dot{\boldsymbol{p}}_p}{V_p}\chi_p(\boldsymbol{x})\cdot\delta\boldsymbol{u}\, dV + \sum_p \int_V \boldsymbol{\sigma}_p\chi_p(\boldsymbol{x})\cdot\nabla\delta\boldsymbol{u}\, dV \tag{1.46}$$

$$+ \sum_p \int_V \delta\boldsymbol{u}\cdot\frac{m_p}{V_p}\left(\boldsymbol{v}_p \mathrm{Tr}(\nabla\boldsymbol{v}_p) + \nabla\boldsymbol{v}_p\boldsymbol{v}_p\right)\chi_p(\boldsymbol{x})\, dV \tag{1.47}$$

1.6.3 Grid Expansion — An Alternate Approach

Next, we expand the virtual displacements in grid-based shape functions using:

$$\delta\boldsymbol{u} = \sum_i \delta\boldsymbol{u_i} N_i(\boldsymbol{x}) \tag{1.48}$$

where the sum is over nodes (i), $\delta\boldsymbol{u_i}$ is the virtual displacement on node i, and $N_i(\boldsymbol{x})$ are the nodal shape functions (standard finite element shape functions). Each standard MPM term is revised as follows:

$$\begin{aligned}
\sum_i\sum_p \int_V \frac{m_p}{V_p}\chi_p(\boldsymbol{x})\boldsymbol{b}_p\cdot\delta\boldsymbol{u_i}N_i(\boldsymbol{x})\, dV &= \sum_i\sum_p S_{ip} m_p \boldsymbol{b}_p\cdot\delta\boldsymbol{u_i} \\
\int_{S_T}\boldsymbol{T}\cdot\delta\boldsymbol{u} &= \sum_i \delta\boldsymbol{u_i}\cdot\int_{S_T} N_i(\boldsymbol{x})\boldsymbol{T}\, dS \\
\sum_i\sum_p \frac{\boldsymbol{F}_p}{V_p}\int_V \chi_F(\boldsymbol{x})\cdot\delta\boldsymbol{u_i}N_i(\boldsymbol{x})\, dV &= \sum_i\sum_p F_{ip}\boldsymbol{F}_p\cdot\delta\boldsymbol{u_i} \\
\sum_i\sum_p \int_V \frac{\dot{\boldsymbol{p}}_p}{V_p}\chi_p(\boldsymbol{x})\cdot\delta\boldsymbol{u_i}N_i(\boldsymbol{x})\, dV &= \sum_i\sum_p S_{ip}\dot{\boldsymbol{p}}_p\cdot\delta\boldsymbol{u_i} \\
\sum_i\sum_p \int_V \boldsymbol{\sigma}_p\chi_p(\boldsymbol{x})\cdot\left(\delta\boldsymbol{u_i}\otimes\nabla N_i(\boldsymbol{x})\right)dV &= \sum_i\sum_p\int_V \chi_p(\boldsymbol{x})\sum_j\sum_k (\boldsymbol{\sigma}_p)_{jk}\delta u_{i,j}\nabla N_i(\boldsymbol{x})_k\, dV \\
&= \sum_i\sum_p\int_V \chi_p(\boldsymbol{x})\sum_j(\boldsymbol{\sigma}_p\nabla N_i(\boldsymbol{x}))_j\delta u_{i,j}\, dV \\
&= \sum_i\sum_p (V_p\boldsymbol{\sigma}_p\boldsymbol{G}_{ip})\cdot\delta\boldsymbol{u_i}
\end{aligned}$$

and the new term is

$$\sum_p\int_V \delta\boldsymbol{u}\cdot\frac{m_p}{V_p}\left(\boldsymbol{v}_p\mathrm{Tr}(\nabla\boldsymbol{v}_p)+\nabla\boldsymbol{v}_p\boldsymbol{v}_p\right)\chi_p(\boldsymbol{x})\, dV = \sum_i\sum_p\left(S_{ip}m_p\left(\boldsymbol{v}_p\mathrm{Tr}(\nabla\boldsymbol{v}_p)+\nabla\boldsymbol{v}_p\boldsymbol{v}_p\right)\right)\cdot\delta\boldsymbol{u_i}$$

where the GIMP shape functions are:

$$S_{ip} = \frac{1}{V_p}\int_V \chi_p(\boldsymbol{x}) N_i(\boldsymbol{x})\, dV \tag{1.49}$$

$$F_{ip} = \frac{1}{V_p}\int_V \chi_F(\boldsymbol{x}) N_i(\boldsymbol{x})\, dV \tag{1.50}$$

$$\boldsymbol{G}_{ip} = \frac{1}{V_p}\int_V \chi_p(\boldsymbol{x}) \nabla N_i(\boldsymbol{x})\, dV \tag{1.51}$$

Two logical choices for $\chi_F(\boldsymbol{x})$ are $\chi_F(\boldsymbol{x}) = \chi_p(\boldsymbol{x})$, which leads to $F_{ip} = S_{ip}$ (and is the choice used below), or $\chi_F(\boldsymbol{x}) = V_p\delta(\boldsymbol{x} - \boldsymbol{x}_p)$, which leads to $F_{ip} = N_i(\boldsymbol{x}_p)$

The particle stress term can be revised to use particle mass as follows:

$$V_p\boldsymbol{\sigma}_p = m_p\frac{\boldsymbol{\sigma}_p}{\rho_p} = m_p\frac{\boldsymbol{\sigma}_p}{\rho_p}\frac{\rho_0}{\rho_0} = m_p\frac{J\sigma_p}{\rho_0} = m_p\frac{\tau_p}{\rho_0} \tag{1.52}$$

Here $J = V_p/V_0 = \rho_0/\rho_p$ is the relative volume and $\boldsymbol{\tau}_p$ is the particle's Kirchhoff stress. In this form the stress term depends most naturally on Kirchhoff stresses normalized to initial density (and this approach avoids the need to find current particle density or volume when calculating that term on each time step). For this reason, `NairnMPM` always tracks stress as $\boldsymbol{\tau}/\rho_0$ (and all material models are expected to calculate this stress). When archiving stress, `NairnMPM` writes Cauchy stress using $\boldsymbol{\sigma}_p = (\rho_0/J)*(\tau/\rho_0)$. Note that when doing energy calculations that $\boldsymbol{\tau}\cdot$ du, where du is incremental deformation gradient, is energy per unit initial volume and therefore the energy calculated in code using $(\tau/\rho_0)\cdot$ du is energy per unit mass or $dU/(\rho_0 V_0)$.

1.6.4 MPM Equations — An Alternate Approach

Making use of the fact that $\delta\boldsymbol{u_i}$ is arbitrary, the summand of all terms can be equated to arrive at the controlling MPM equation on the background grid of:

$$\sum_p S_{ip}^{(n)}\frac{d\boldsymbol{p_p^{(n)}}}{dt} = \frac{d\boldsymbol{p}_i^{(n)}}{dt} = \boldsymbol{f}_{i,int}^{(n)} + \boldsymbol{f}_{i,ext}^{(n)} \tag{1.53}$$

where

$$\boldsymbol{f}_{i,int}^{(n)} = \sum_p -m_p\left(\frac{\boldsymbol{\tau}_p^{(n)}\cdot\boldsymbol{G}_{ip}^{(n)}}{\rho_0} + \left(\boldsymbol{v}_p\mathrm{Tr}(\nabla\boldsymbol{v}_p) + \nabla\boldsymbol{v}_p\boldsymbol{v}_p\right)S_{ip}\right) \tag{1.54}$$

$$\boldsymbol{f}_{i,ext}^{(n)} = \boldsymbol{f}_{i,T}^{(n)} + \sum_p\left(m_p S_{ip}^{(n)}\boldsymbol{b}_p + \boldsymbol{F}_p^{(n)}S_{ip}^{(n)}\right) \tag{1.55}$$

$$\boldsymbol{f}_{i,T}^{(n)} = \int_{S_T}\boldsymbol{N}_i(\boldsymbol{x})\boldsymbol{T}\, dS \tag{1.56}$$

where $\boldsymbol{f}_{i,int}^{(n)}$ are internal forces due to stresses only and $\boldsymbol{f}_{i,ext}^{(n)}$ are external forces due to body forces, particle loads and traction forces ($\boldsymbol{f}_{i,T}^{(n)}$). Superscript (n) has been added to mean terms calculated from the state of all particles at the start of time step n. Commonly the body force, $\boldsymbol{b}_p$, will be independent of the particle state (*e.g.,* gravity, which is a constant, or a general body force that depends only on nodal

position and time and not on particle state). When that holds, it can be removed from the sum. The total internal force then becomes

$$\boldsymbol{f}_{i,ext}^{(n)} = \boldsymbol{f}_{i,T}^{(n)} + m_i^{(n)} \boldsymbol{b}(\boldsymbol{x}_i, t) + \sum_p \boldsymbol{F}_p^{(n)} S_{ip}^{(n)} \tag{1.57}$$

where $\boldsymbol{b}(\boldsymbol{x}_i, t)$ is body force at the nodal location and

$$m_i^{(n)} = \sum_p m_p S_{ip}^{(n)} \tag{1.58}$$

is total nodal mass.

Once the forces are found, the momentum can be updated on the grid using

$$\boldsymbol{p}_i^{(n+1)} = \boldsymbol{p}_i^{(n)} + \boldsymbol{f}_i^{(n)} \Delta t \qquad \text{where} \qquad \boldsymbol{f}_i^{(n)} = \boldsymbol{f}_{i,int}^{(n)} + \boldsymbol{f}_{i,ext}^{(n)} \tag{1.59}$$

where Δt is the time step and:

$$\boldsymbol{p}_i^{(n)} = \sum_p \boldsymbol{p}_p^{(n)} S_{ip}^{(n)} \tag{1.60}$$

Chapter 2

Strain and Particle Updates

2.1 Introduction

This chapter describes MPM methods for updating the stress and strain on particles and updating particle velocity and positions. More details on particle updates are given in the technical notes `Extrap.tex`.

2.2 Incremental Displacement Gradient

For particle stress and strain updates, the algorithm needs to evaluate the spatial velocity gradient, $\nabla \boldsymbol{v}(x,t)$, and use that to find the incremental deformation gradient, dF, defined by

$$\mathsf{dF}^{(n)} = \exp(\nabla \boldsymbol{v}\, \Delta t) \tag{2.1}$$

which assumes $\nabla \boldsymbol{v}$ is constant over the time step. The gradient is found by extrapolating spatial velocity from grid to particles using nodal velocity and gradient shape functions:

$$\nabla \boldsymbol{v} = \sum_i \boldsymbol{v}_i^{(n)} \otimes \boldsymbol{G}_{ip} \tag{2.2}$$

and the spatial velocity is found from

$$\boldsymbol{v}_i^{(n)} = \frac{\boldsymbol{p}_i^{(n)}}{m_i^{(n)}} \tag{2.3}$$

This velocity gradient is used to update stresses and strains on the particles. It can be done at the beginning of the time step (using $\boldsymbol{v}_i^{(n)}$) or at the end of the time step using the updated grid velocity ($\boldsymbol{v}_i^{(n+1)}$). `NairnMPM` allows either of the options and adds a third option to update both at the beginning and at the end. This later method (which is default method called `USAVG`) is analogous to a midpoint rule integration. Although it is less efficient (due to two updates), it may provide improved convergence as a function of time step size.

2.3 Particle Updates with Damping

In each time step, the Eulerian update on the grid is used to update the Lagrangian velocities and positions on the particles, and it is useful to include damping options in both the grid and particle updates. Two global damping strategies are to add damping terms proportional to either the grid velocity or the particle velocity. In most cases, these two approaches to damping should have similar results. By

including them both, however, it is possible to propose several different types of damping schemes. The fact that MPM has two velocities (grid and particle velocity), is the reason MPM has some interesting damping options.

If damping is based on grid velocity, then the nodal momentum update changes to a damped acceleration or:

$$p_i^{*(n+1)} = p_i^{(n)} + m_i^{(n)}(\boldsymbol{a}_i^{(n)} - \alpha_g(t)\boldsymbol{v}_i^{(n)})\Delta t = p_i^{(n)} + m_i^{(n)}\boldsymbol{a}_i^{*(n)}\Delta t \tag{2.4}$$

where "*" indicates a term that is revised to include damping, where $\alpha_g(t)$ is the grid damping constant (with units 1/sec) that applies to all nodes (*i.e.*, independent of $\boldsymbol{x}_i$), but may evolve in time (such as a kinetic energy thermostat). See below for how to deal with position-dependent damping.

Particle methods have historically tried two methods for updating particle velocity — a FLIP update that increments particle velocity using accelerations extrapolated to the grid and a PIC update that extrapolates velocity directly to the particle. Combining the grid damping above with a second particle damping constant, $\alpha_p(t)$, these two updates become:

$$\boldsymbol{v}_{p,FLIP}^{(n+1)} = \boldsymbol{v}_p^{(n)} + \boldsymbol{a}_{g\to p}^{**(n)}\Delta t \tag{2.5}$$

$$\boldsymbol{v}_{p,PIC}^{(n+1)} = \boldsymbol{v}_{g\to p}^{(n)} + \boldsymbol{a}_{g\to p}^{**(n)}\Delta t = \boldsymbol{v}_{g\to p}^{*(n+1)} - \alpha_p(t)\boldsymbol{v}_p^{(n)}\Delta t \tag{2.6}$$

where $\boldsymbol{a}_{g\to p}^{**(n)}$ is a doubly-damped acceleration extrapolated to the particle (*i.e.*, subscript $g \to p$ means extrapolation from the grid to the particles):

$$\boldsymbol{a}_{g\to p}^{**(n)} = \boldsymbol{a}_{g\to p}^{*(n)} - \alpha_p(t)\boldsymbol{v}_p^{(n)} = \boldsymbol{a}_{g\to p}^{(n)} - \alpha_g(t)\boldsymbol{v}_{g\to p}^{(n)} - \alpha_p(t)\boldsymbol{v}_p^{(n)} \tag{2.7}$$

and the needed extrapolations to the particle are:

$$\boldsymbol{a}_{g\to p}^{*(n)} = \sum_i \boldsymbol{a}_i^{*(n)} S_{ip}^{(n)}, \qquad \boldsymbol{a}_{g\to p}^{(n)} = \sum_i \frac{\boldsymbol{f}_i^{(n)}}{m_i^{(n)}} S_{ip}^{(n)}, \quad \text{and} \quad \boldsymbol{v}_{g\to p}^{(n)} = \sum_i \boldsymbol{v}_i^{(n)} S_{ip}^{(n)} \tag{2.8}$$

The extrapolated velocities are

$$\boldsymbol{v}_{g\to p}^{*(n+1)} = \sum_i \frac{\boldsymbol{p}_i^{*(n+1)}}{m_i^{(n)}} S_{ip}^{(n)} = \boldsymbol{v}_{g\to p}^{(n+1)} - \alpha_g(t)\boldsymbol{v}_{g\to p}^{(n)}\Delta t = \boldsymbol{v}_{g\to p}^{(n)} + \boldsymbol{a}_{g\to p}^{*(n)}\Delta t \tag{2.9}$$

$$\boldsymbol{v}_{g\to p}^{(n+1)} = \sum_i \left(\frac{\boldsymbol{p}_i^{(n)}}{m_i^{(n)}} + \frac{\boldsymbol{f}_i^{(n)}}{m_i^{(n)}}\Delta t \right) S_{ip}^{(n)} = \boldsymbol{v}_{g\to p}^{(n)} + \boldsymbol{a}_{g\to p}^{(n)}\Delta t \tag{2.10}$$

Both damping strategies can be done entirely within the particle update. The two methods can be used individually or combined. Because grid and particle velocities should be similar, the total damping constant is $\alpha(t) = \alpha_g(t) + \alpha_p(t)$. In the absence of damping ($\alpha(t) = \alpha_g(t) = 0$), the FLIP and PIC updates and the simpler form often presented in the literature:

$$\boldsymbol{v}_{p,FLIP}^{(n+1)} = \boldsymbol{v}_p^{(n)} + \boldsymbol{a}_{g\to p}^{(n)}\Delta t \tag{2.11}$$

$$\boldsymbol{v}_{p,PIC}^{(n+1)} = \boldsymbol{v}_{g\to p}^{(n+1)} \tag{2.12}$$

The results above extend these updates for two types of damping (grid or particle damping).

Next, the MPM position update with damping is found by integrating the extrapolated velocity over the time step. This integration is done on the grid and therefore should use the PIC form of velocity

(*i.e.*, the grid velocity). Integrating using the midpoint rule:

$$\boldsymbol{x}_p^{(n+1)} = \boldsymbol{x}_p^{(n)} + \int_0^{\Delta t} \boldsymbol{v}_{p,PIC}^{(n+1)}(\Delta t = t)\, dt \approx \boldsymbol{x}_p^{(n)} + \frac{1}{2}\left(\boldsymbol{v}_{g\to p}^{(n)} + \boldsymbol{v}_{p,PIC}^{(n+1)}\right)\Delta t \tag{2.13}$$

$$= \boldsymbol{x}_p^{(n)} + \boldsymbol{v}_{g\to p}^{(n)}\Delta t + \frac{1}{2}\boldsymbol{a}_{g\to p}^{**(n)}(\Delta t)^2 \tag{2.14}$$

This update along with the FLIP velocity update (which is usually recommended due to its conservation of momentum properties) can be written using terms convenient for coding (*i.e.*, $\boldsymbol{v}_{g\to p}^{*(n+1)}$ and $\boldsymbol{a}_{g\to p}^{*(n)}$) as:

$$\boldsymbol{x}_p^{(n+1)} = \boldsymbol{x}_p^{(n)} + \boldsymbol{v}_{g\to p}^{*(n+1)}\Delta t - \frac{1}{2}\left(\boldsymbol{a}_{g\to p}^{*(n)} + \alpha_p(t)\boldsymbol{v}_p^{(n)}\right)(\Delta t)^2 \tag{2.15}$$

$$\boldsymbol{v}_p^{(n+1)} = \boldsymbol{v}_p^{(n)} + \left(\boldsymbol{a}_{g\to p}^{*(n)} - \alpha_p(t)\boldsymbol{v}_p^{(n)}\right)\Delta t \tag{2.16}$$

These are the updates always used by `NairnMPM`.

It is interesting to redefine the grid and particle damping terms using:

$$\alpha_g(t) \to -\frac{1-\beta}{\Delta t} + \alpha_g' \qquad \text{and} \qquad \alpha_p(t) \to \frac{1-\beta}{\Delta t} + \alpha_p' \tag{2.17}$$

The time dependence of primed damping terms have been dropped, but they can all still depend on time. By unraveling all effective terms, the net updates become

$$\boldsymbol{x}_p^{(n+1)} = \boldsymbol{x}_p^{(n)} + \boldsymbol{v}_{g\to p}^{(n+1)}\Delta t - \frac{1}{2}\left(\boldsymbol{a}_{g\to p}^{(n)} + \alpha_{PIC}(\beta)\left(\boldsymbol{v}_p^{(n)} - \boldsymbol{v}_{g\to p}^{(n)}\right) + \alpha_g'\boldsymbol{v}_{g\to p}^{(n)} + \alpha_p'\boldsymbol{v}_p^{(n)}\right)(\Delta t)^2 \tag{2.18}$$

$$\boldsymbol{v}_p^{(n+1)} = \boldsymbol{v}_p^{(n)} + \left(\boldsymbol{a}_{g\to p}^{(n)} - \alpha_{PIC}(\beta)\left(\boldsymbol{v}_p^{(n)} - \boldsymbol{v}_{g\to p}^{(n)}\right) - \alpha_g'\boldsymbol{v}_{g\to p}^{(n)} - \alpha_p'\boldsymbol{v}_p^{(n)}\right)\Delta t \tag{2.19}$$

where the new "PIC Damping" term is

$$\alpha_{PIC}(\beta) = \frac{1-\beta}{\Delta t} \tag{2.20}$$

It is called PIC damping, because in the absence of other damping terms ($\alpha_g' = \alpha_p' = 0$), the still-damped position and velocity updates become:

$$\boldsymbol{x}_p^{(n+1)} = \boldsymbol{x}_p^{(n)} + \left(\boldsymbol{v}_{g\to p}^{(n+1)} - \frac{1-\beta}{2}\left(\boldsymbol{v}_p^{(n)} - \boldsymbol{v}_{g\to p}^{(n)}\right)\right)\Delta t - \frac{1}{2}\boldsymbol{a}_{g\to p}^{(n)}(\Delta t)^2 \tag{2.21}$$

$$\boldsymbol{v}_p^{(n+1)} = \boldsymbol{v}_p^{(n)} - (1-\beta)\left(\boldsymbol{v}_p^{(n)} - \boldsymbol{v}_{g\to p}^{(n)}\right) + \boldsymbol{a}_{g\to p}^{(n)}\Delta t \tag{2.22}$$

If $\beta = 1$, the update is an undamped FLIP update. If $\beta = 0$, the update is an undamped, PIC update. Thus β can be interpreted as the fraction FLIP in the velocity update. But the above analysis was formally all a FLIP analysis. A better interpretation is that β adds an new form of artificial damping that is proportional to the inversion error between the particle velocity and the grid velocity extrapolated to the particle ($\boldsymbol{v}_p^{(n)} - \boldsymbol{v}_{g\to p}^{(n)}$). The net damping effect will be to damp out this error.

2.3.1 Some Implementation Details

There are two approaches to coding updates with damping. First, it can be done entirely within the particle update task. The input to this task from the nodes will be $p_i^{(n+1)}$ (which will have replaced $p_i^{(n)}$

in the momentum update), $\boldsymbol{f}_i^{(n)}$, and $m_i^{(n)}$. We can define two new effective terms

$$\begin{aligned}
\boldsymbol{a}_{g\to p}^{(n)'} &= \boldsymbol{a}_{g\to p}^{(n)} - \alpha_{PIC}(\beta)\big(\boldsymbol{v}_p^{(n)} - \boldsymbol{v}_{g\to p}^{(n)}\big) - \alpha_g' \boldsymbol{v}_{g\to p}^{(n)} - \alpha_p' \boldsymbol{v}_p^{(n)} && (2.23)\\
&= \boldsymbol{a}_{g\to p}^{(n)} - \alpha_g(t)\boldsymbol{v}_{g\to p}^{(n)} - \alpha_p(t)\boldsymbol{v}_p^{(n)} && (2.24)\\
\boldsymbol{v}_{g\to p}^{(n+1)'} &= \boldsymbol{v}_{g\to p}^{(n+1)} - \Big(\alpha_{PIC}(\beta)\big(\boldsymbol{v}_p^{(n)} - \boldsymbol{v}_{g\to p}^{(n)}\big) + \alpha_g' \boldsymbol{v}_{g\to p}^{(n)} + \alpha_p' \boldsymbol{v}_p^{(n)}\Big)\Delta t && (2.25)\\
&= \boldsymbol{v}_{g\to p}^{(n+1)} - \Big(\alpha_g(t)\boldsymbol{v}_{g\to p}^{(n)} + \alpha_p(t)\boldsymbol{v}_p^{(n)}\Big)\Delta t && (2.26)
\end{aligned}$$

where $\boldsymbol{v}_{g\to p}^{(n)}$ is found from available input properties using

$$\boldsymbol{v}_{g\to p}^{(n)} = \boldsymbol{v}_{g\to p}^{(n+1)} - \boldsymbol{a}_{g\to p}^{(n)}\Delta t \tag{2.27}$$

The final updates become

$$\begin{aligned}
\boldsymbol{x}_p^{(n+1)} &= \boldsymbol{x}_p^{(n)} + \boldsymbol{v}_{g\to p}^{(n+1)'}\Delta t - \frac{1}{2}\boldsymbol{a}_{g\to p}^{(n)'}(\Delta t)^2 && (2.28)\\
\boldsymbol{v}_p^{(n+1)} &= \boldsymbol{v}_p^{(n)} + \boldsymbol{a}_{g\to p}^{(n)'}\Delta t && (2.29)
\end{aligned}$$

which are standard position and velocity updates to second order in position. Notice that it was important to keep the second order term on the position update in order to end up with consistent definitions for extrapolated velocity and acceleration.

In `NairnMPM`, the grid damping is added in the update particles task, but the particle damping is not added until calling the material point class. Separating them out gives

$$\begin{aligned}
\boldsymbol{x}_p^{(n+1)} &= \boldsymbol{x}_p^{(n)} + \big(\boldsymbol{v}_* - \alpha_p(t)\boldsymbol{v}_p^{(n)}\Delta t\big)\Delta t - \frac{1}{2}\big(\boldsymbol{a}_* - \alpha_p(t)\boldsymbol{v}_p^{(n)}\big)(\Delta t)^2 && (2.30)\\
&= \boldsymbol{x}_p^{(n)} + \Delta t\left(\boldsymbol{v}_* - \frac{\Delta t}{2}\big(\boldsymbol{a}_* + \alpha_p(t)\boldsymbol{v}_p^{(n)}\big)\right) && (2.31)\\
\boldsymbol{v}_p^{(n+1)} &= \boldsymbol{v}_p^{(n)} + \big(\boldsymbol{a}_* - \alpha_p(t)\boldsymbol{v}_p^{(n)}\big)\Delta t = \boldsymbol{v}_p^{(n)}\big(1 - \alpha_p(t)\Delta t\big) + \boldsymbol{a}_*\Delta t && (2.32)
\end{aligned}$$

where

$$\begin{aligned}
\boldsymbol{v}_* &= \boldsymbol{v}_{g\to p}^{(n+1)} - \alpha_g(t)\boldsymbol{v}_{g\to p}^{(n)}\Delta t && (2.33)\\
\boldsymbol{a}_* &= \boldsymbol{a}_{g\to p}^{(n)} - \alpha_g(t)\boldsymbol{v}_{g\to p}^{(n)} && (2.34)
\end{aligned}$$

The position update must be called first, because it needs $\boldsymbol{v}_p^{(n)}$, which is changed in the velocity update.

2.3.2 Applying Damping to Force Calculations

When all damping is done in the particle updates (as is possible from above equations), it is possible that damping will affect other sections of the code that use nodal momentum or force, such as contact, differently. To avoid these effects (if they matter), a second approach would be to apply the grid damping to the force calculation. But, when PIC damping is used, this could cause very unrealistic forces (because that term can be large when Δt is small). A potential solution is to apply only α_g' to force calculations and then apply PIC damping during the particle updates. At the time of the particle update, the nodal

input values would be

$$\boldsymbol{f}_i^{**(n)} = \boldsymbol{f}_i^{(n)} - \alpha_g' \boldsymbol{p}_i^{(n)} \tag{2.35}$$

$$\boldsymbol{p}_i^{**(n+1)} = p_i^{(n)} + m_i^{(n)} \boldsymbol{a}_i^{**(n)} \Delta t \tag{2.36}$$

$$\boldsymbol{a}_i^{**(n)} = \frac{\boldsymbol{f}_i^{(n)}}{m_i^{(n)}} - \alpha_g' \boldsymbol{v}_i^{(n)} \tag{2.37}$$

The effective terms become

$$\boldsymbol{a}_{g\to p}^{(n)'} = \boldsymbol{a}_{g\to p}^{**(n)} - \alpha_{PIC}(\beta)\left(\boldsymbol{v}_p^{(n)} - \boldsymbol{v}_{g\to p}^{(n)}\right) - \alpha_p' \boldsymbol{v}_p^{(n)} \tag{2.38}$$

$$\boldsymbol{v}_{g\to p}^{(n+1)'} = \boldsymbol{v}_{g\to p}^{**(n+1)} - \left(\alpha_{PIC}(\beta)\left(\boldsymbol{v}_p^{(n)} - \boldsymbol{v}_{g\to p}^{(n)}\right) + \alpha_p' \boldsymbol{v}_p^{(n)}\right)\Delta t \tag{2.39}$$

where $\boldsymbol{v}_{g\to p}^{(n)}$ is found from available input properties using

$$\boldsymbol{v}_{g\to p}^{(n)} = \boldsymbol{v}_{g\to p}^{**(n+1)} - \boldsymbol{a}_{g\to p}^{**(n)} \Delta t \tag{2.40}$$

The final updates are the same as above except when coding

$$\boldsymbol{v}_* = \boldsymbol{v}_{g\to p}^{**(n+1)} + \alpha_{PIC} \boldsymbol{v}_{g\to p}^{(n)} \Delta t \tag{2.41}$$

$$\boldsymbol{a}_* = \boldsymbol{a}_{g\to p}^{**(n)} + \alpha_{PIC} \boldsymbol{v}_{g\to p}^{(n)} \tag{2.42}$$

2.3.3 Position Dependent Damping Strategies

Potentially it might be useful to damp certain regions differently than other regions. For example, damping could be used in a crack plane to absorb released energy. This section considers changes when the damping terms depend on position. For damping based on grid velocity, the nodal momentum update changes to:

$$p_i^{*(n+1)} = p_i^{(n)} + \left(\boldsymbol{a}_i^{*(n)} - \alpha_g(t, \boldsymbol{x}_i)\boldsymbol{v}_i^{(n)}\right)\Delta t = p_i^{(n)} + \boldsymbol{a}_i^{*(n)} \Delta t \tag{2.43}$$

where "*" indicates a term that is revised to including position-dependent damping term where $\boldsymbol{x}_i$ is the nodal position. The particle velocity update would be:

$$\boldsymbol{v}_{p,FLIP}^{(n+1)} = \boldsymbol{v}_p^{(n)} + \left(\boldsymbol{a}_{g\to p}^{(n)} - \left(\sum_i \alpha_g(t, \boldsymbol{x}_i)\boldsymbol{v}_i^{(n)} S_{ip}\right) - \alpha_p(t, \boldsymbol{x}_p)\boldsymbol{v}_p^{(n)}\right)\Delta t \tag{2.44}$$

The summation terms causes some problems. First, it is inefficient because it needs another extrapolation and likely repeated calculation of $\alpha_g(t, \boldsymbol{x}_i)$. Second, it loses connection to simple velocity terms. The situation can be improved by replacing $\alpha_g(t, \boldsymbol{x}_i)$ with $\alpha_g(t, \boldsymbol{x}_p)$, which is constant for the sum and can be removed from the sum to give

$$\boldsymbol{v}_{p,FLIP}^{(n+1)} = \boldsymbol{v}_p^{(n)} + \left(\boldsymbol{a}_{g\to p}^{(n)} - \alpha_g(t, \boldsymbol{x}_p)\boldsymbol{v}_{g\to p}^{(n)} - \alpha_p(t, \boldsymbol{x}_p)\boldsymbol{v}_p^{(n)}\right)\Delta t \tag{2.45}$$

The remainder of the update analysis can be followed as above. The only difference in code is that $\alpha_g(t, \boldsymbol{x}_p)$ and $\alpha_p(t, \boldsymbol{x}_p)$ need to be calculated for each particle in the update loop while for time dependence only, they only need to be calculated once in the entire particle update task. In this approach, the damping could be implemented as a particle property rather then as a function of Eulerian grid coordinates.

2.4 GIMP Derivation Comments

The nodal force $\boldsymbol{f}_i$ is sometimes interpreted as a sum of internal force (due to stresses only), body force, external force, and traction forces, but there is no need to separate them during calculations; they all add to the nodal force. The force $\boldsymbol{f}_{i,T}$ is a force due to tractions and needs extra work to integrate the tractions over traction-loaded surfaces in the extent of the grid shape function for node i.

Note that Eq. (1.53) is diagonal in the nodal system. Thus unlike other derivations of MPM, which get to here with a non-diagonal mass matrix, there is no need to invoke lumping of that mass matrix to make the problem tractable. The GIMP derivation naturally results in a lumped mass matrix.

Within this generalized framework, the various style of MPM are characterized by how they calculate S_{ip} and $\boldsymbol{G}_{ip}$. Given particle functions, $\chi_p(\boldsymbol{x})$, the above equations give an "exact" result that is sometimes called "finite GIMP." In general, exact integration of arbitrary particle domains is difficult and is therefore not done. Instead various schemes have developed to approximate the integrals. The `NairnMPM` wiki page as one approach to classify the MPM family tree of methods.

Chapter 3

Velocity Boundary Conditions on the Grid

3.1 Grid Velocity Conditions

After extrapolating to the grid, each node will have $\boldsymbol{p}_i^{(n)} = m_i^{(n)}\boldsymbol{v}_i^{(n)}$, and $\boldsymbol{f}_i^{(n)}$. Consider a node with a prescribed velocity in a direction $\hat{n}$ of $v_i^{(BC)}$ (a scalar). Let $\hat{t}$ be vector tangential to the $\hat{n}$ direction. The updated momentum we want becomes

$$\begin{aligned} \boldsymbol{p}_i^{(n+1)} &= m_i^{(n)} v_i^{(BC)} \hat{n} + \left(\left(\boldsymbol{p}_i^{(n)} + \boldsymbol{f}_i^{(n)} \Delta t \right) \cdot \hat{t} \right) \hat{t} && (3.1) \\ &= \boldsymbol{p}_i^{(n)} + \left(\boldsymbol{f}_i^{(n)} + f_i^{(BC)} \hat{n} \right) \Delta t && (3.2) \end{aligned}$$

where $f_i^{(BC)}$ is scalar force caused by boundary condition applied in the $\hat{n}$ direction. The normal component gives:

$$\boldsymbol{p}_i^{(n+1)} \cdot \hat{n} = m_i^{(n)} v_i^{(BC)} = \boldsymbol{p}_i^{(n)} \cdot \hat{n} + (\boldsymbol{f}_i^{(n)} \cdot \hat{n}) \Delta t + f_i^{(BC)} \Delta t \tag{3.3}$$

which is solved for

$$f_i^{(BC)} = \frac{m_i^{(n)} v_i^{(BC)} - \boldsymbol{p}_i^{(n)} \cdot \hat{n}}{\Delta t} - \boldsymbol{f}_i^{(n)} \cdot \hat{n} \tag{3.4}$$

Thus, a grid boundary condition can be done by adding $f_i^{(BC)} \hat{n}$ to each node having prescribed velocity $v_i^{(BC)}$ in the $\hat{n}$ direction. Note the velocity BCs are done by changing force rather then momentum to be sure the particle acceleration is consistent with reaction force (and the acceleration affects particle updates).

More details on grid BC calculations and additional calculations done an other locations in the MPM time step are given in the `contectetc.tex` technical notes (see chapter on "MPM Time Step Stasks").

3.2 Velocity Control using Rigid Material Contact

We can contrast prescribed velocity to contact between material a and rigid material b where $\hat{n}$ is the contacting normal (determined by various methods). The two options are very similar, but contact with a rigid material adds option to have normals determined by geometry of a collection of rigid particles (while velocity BCs have to preset the normal) and support for other contact laws (while velocity BCs are frictionless only, although maybe friction could be added). Another difference is that contact with rigid particles occurs on all nodes with non-zero GIMP shape functions while grid BCs created by rigid particles are only on nodes for the element containing that particle. The code tasks associated with rigid contact implementation are:

1. The first multimaterial mode extrapolation to the grid finds $\boldsymbol{v}_{i,a}^{(n)}$, $m_{i,a}^{(n)}$, and $\boldsymbol{v}_{i,a}^{(n)}$

2. After momentum extrapolation (and before velocity boundary conditions), the nodes determined to be in contact and sliding due to frictional contact (*i.e.*, not sticking) will replace momenta for material a with

$$\begin{aligned} \boldsymbol{p}_{i,a}^{(n)'} &= \boldsymbol{p}_{i,a}^{(n)} + \Delta\boldsymbol{p}_{i,a}^{(1)'} && (3.5)\\ &= \boldsymbol{p}_{i,a}^{(n)} + \left((m_{i,a}^{(n)}\boldsymbol{v}_{i,b}^{(n)} - \boldsymbol{p}_{i,a}^{(n)})\cdot\hat{n}\right)(\hat{n}-\mu\hat{t}) && (3.6)\end{aligned}$$

   where $\Delta\boldsymbol{p}_{i,a}^{(1)'}$ is momentum change due to contact law (as defined in my papers). The second form is specific for Coulomb friction contact with a rigid material and with μ as the coefficient of friction. For frictionless contact, this is the same change as a nodal velocity boundary condition of $v_i^{(BC)} = \boldsymbol{v}_{i,b}^{(n)}\cdot\hat{n}$ applied in the $\hat{n}$ direction.

3. After extrapolating forces and updating the revised momenta, revisit nodes that had boundary conditions and subtract $\Delta\boldsymbol{p}_{i,a}^{(1)'}$ to restore nodal momentum to values uncorrected for contact (note that this feature was added in `OSParticulas`, revision 1179; old method of not restoring momentum could show problems in particles near contact mostly when using FLIP methods and mostly small effects). The next step updates momenta, which will be:

$$\boldsymbol{p}_{i,a}^{(n+1)} = \boldsymbol{p}_{i,a}^{(n)} + \boldsymbol{f}_i^{(n)}\Delta t \qquad (3.7)$$

   This new momentum will likely violate contact laws and thus another momentum change is needed:

$$\begin{aligned} \boldsymbol{p}_{i,a}^{(n+1)'} &= \boldsymbol{p}_{i,a}^{(n+1)} + \Delta\boldsymbol{p}_{i,a}^{(2)'} && (3.8)\\ &= \boldsymbol{p}_{i,a}^{(n+1)} + \left((m_{i,a}^{(n)}\boldsymbol{v}_{i,b}^{(n)} - \boldsymbol{p}_{i,a}^{(n+1)})\cdot\hat{n}\right)(\hat{n}-\mu\hat{t}) && (3.9)\end{aligned}$$

   The final update based on the initial extrapolated momentum becomes:

$$\boldsymbol{p}_{i,a}^{(n+1)'} = \boldsymbol{p}_{i,a}^{(n)} + \left((m_{i,a}^{(n)}\boldsymbol{v}_{i,b}^{(n)} - \boldsymbol{p}_{i,a}^{(n)} - \boldsymbol{f}_i^{(n)}\Delta t)\cdot\hat{n}\right)(\hat{n}-\mu\hat{t}) + \boldsymbol{f}_i^{(n)}\Delta t \qquad (3.10)$$

   The total contact force from initial momentum to final updated momentum is

$$\boldsymbol{f}_i^{(contact)} = \left(\frac{(m_{i,a}^{(n)}\boldsymbol{v}_{i,b}^{(n)} - \boldsymbol{p}_{i,a}^{(n)})\cdot n}{\Delta t} - \boldsymbol{f}_i^{(n)}\cdot\hat{n}\right)(\hat{n}-\mu\hat{t}) = \frac{\Delta\boldsymbol{p}_{i,a}^{(2)'}}{\Delta t} \qquad (3.11)$$

   The force is analogous to the $f_i^{(BC)}$.

4. If using `USAVG+` or `USL+`, the code does a second extrapolation to the grid to find $\boldsymbol{p}_i^{(n+1)''}$. These extrapolated momenta are replaced with new momenta:

$$\begin{aligned} \boldsymbol{p}_{i,a}^{(n+1)'} &= \boldsymbol{p}_{i,a}^{(n+1)} + \Delta\boldsymbol{p}_{i,a}^{(3)'} && (3.12)\\ &= \boldsymbol{p}_{i,a}^{(n+1)} + \left((m_{i,a}^{(n)}\boldsymbol{v}_{i,b}^{(n)} - \boldsymbol{p}_{i,a}^{(n+1)})\cdot\hat{n}\right)(\hat{n}-\mu\hat{t}) && (3.13)\end{aligned}$$

   and used to update particle stresses and strains.

Chapter 4

One Dimensional GIMP

4.1 1D Virtual Work

Sometimes it is useful to look at 1D calculations to gain insights into MPM. It could be derived from above result, but here derived starting with virtual power in 1D:

$$A\int_L \rho b\delta u\,dx + TA\delta u|_{x_{min}}^{x_{max}} + \sum_p F_p\delta u = A\int_L \rho a\delta u\,dx + A\int_L \boldsymbol{\sigma}\frac{d\delta u}{dx}\,dx \tag{4.1}$$

where A is assumed cross-sectional area of the 1D grid.

4.2 MPM Equations

Inserting particle and grid expansions (as above) and making use of the fact that δu is arbitrary, the summand of all terms can be equated to arrive at the controlling MPM equation on the background grid of:

$$\sum_p S_{ip}^{(n)}\frac{dp_p^{(n)}}{dt} = \frac{dp_i^{(n)}}{dt} = f_i^{(n)} \tag{4.2}$$

where

$$p_i^{(n)} = \sum_p M_p v_p^{(n)} S_{ip}^{(n)} \tag{4.3}$$

$$m_i^{(n)} = \sum_p M_p S_{ip}^{(n)} \tag{4.4}$$

$$f_i^{(n)} = f_{i,T}^{(n)} + \sum_p \left(-m_p\frac{\tau_p^{(n)} G_{ip}^{(n)}}{\rho_0} + m_p S_{ip}^{(n)} b_p + F_p^{(n)} S_{ip}^{(n)} \right) \tag{4.5}$$

$$f_{i,T}^{(n)} = AN_i(x_{face})T(x_{face}) \tag{4.6}$$

where x_{face} is side of particle with applied traction.

4.2.1 1D Shape Functions

The shape functions are:

$$S_{ip} = \frac{1}{l_p \Delta x} \int_{X_p - l_p}^{X_p + l_p \Delta x/2} \chi_p(x) N_i(\boldsymbol{x})\, dx \tag{4.7}$$

$$G_{ip} = \frac{1}{l_p \Delta x} \int_{X_p - l_p}^{X_p + l_p \Delta x/2} \chi_p(x) \frac{dN_i(x)}{dx}\, dx \tag{4.8}$$

where particle at X_p has length $l_p \Delta x$ and Δx is cell size (of a regular grid) For $N_i(x)$ being linear hat functions and $\chi_p(x) = 1$ in particle domain, these full integrals for node at x_i are:

$$S_{ip}(x_p) = \begin{cases} \dfrac{(4 - l_p) l_p - x_p^2}{4 l_p} & |x_p| < l_p \\ \dfrac{2 - |x_p|}{2} & l_p \le |x_p| < 2 - l_p \\ \dfrac{(2 + l_p - |x_p|)^2}{8 l_p} & 2 - l_p < |x| < 2 + l_p \\ 0 & \text{otherwise} \end{cases} \tag{4.9}$$

$$G_{ip}(x_p) = \frac{2}{\Delta x} \frac{dS_{ip}(x_p)}{dx_p} = \frac{2}{\Delta x} \begin{cases} -\dfrac{x_p}{2 l_p} & |x_p| < l_p \\ -\mathrm{sgn(x_p)}/2 & l_p \le \xi_p < 2 - l_p \\ -\mathrm{sgn(x_p)} \dfrac{2 + l_p - |x_p|}{4 l_p} & 2 - l_p \le \xi_p < 2 + l_p \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

where $x_p = 2(X_p - x_i)/\Delta x$ is distance from particle p to node i in units of semi-cell size. In GIMP with two particle per cell or $l_p = 1/2$, the shape functions become:

$$S_{ip}(x_p) = \begin{cases} \dfrac{7 - 4x_p^2}{8} & |x_p| < 1/2 \\ \dfrac{2 - |x_p|}{2} & 1/2 \le |x_p| < 3/2 \\ \dfrac{(5 - 2|x_p|)^2}{16} & 3/2 < |x| < 5/2 \\ 0 & \text{otherwise} \end{cases} \tag{4.11}$$

$$G_{ip}(x_p) = \frac{2}{\Delta x} \frac{dS_{ip}(x_p)}{dx_p} = \frac{2}{\Delta x} \begin{cases} -x_p & |x_p| < 1/2 \\ -\mathrm{sgn(x_p)}/2 & 1/2 \le \xi_p < 3/2 \\ -\mathrm{sgn(x_p)} \dfrac{5 - 2|x_p|}{4} & 3/2 \le \xi_p < 5/2 \\ 0 & \text{otherwise} \end{cases} \tag{4.12}$$

Chapter 5

Axisymmetric GIMP

In axisymmetric GIMP, the virtual work integrals are converted to cylindrical integrals:

$$\int_A \rho \boldsymbol{b}\cdot\delta\boldsymbol{u}\, r dA + \int_{L_T} \boldsymbol{T}\cdot\delta\boldsymbol{u}\, r dL + \sum_p \boldsymbol{F_p}\cdot\delta\boldsymbol{u} = \int_A \rho\boldsymbol{a}\cdot\delta\boldsymbol{u}\, r dA + \int_A \boldsymbol{\sigma}\cdot\nabla\delta\boldsymbol{u}\, r dA \tag{5.1}$$

where A is the area of integration in the r-z plane, $dA = dr\, dz$, L_T is the path for surfaces having traction loads, and $\boldsymbol{T}$ and $\boldsymbol{F_p}$ have been redefined to be traction and force per radian. The result after expansion in the particle basis changes to:

$$\sum_p \int_A \frac{m_p}{A_p\langle r_p\rangle}\chi_p(\boldsymbol{x})\boldsymbol{b}_p\cdot\delta\boldsymbol{u}\, r dA + \int_{L_T}\boldsymbol{T}\cdot\delta\boldsymbol{u}\, dL + \sum_p \frac{\boldsymbol{F}_p}{A_p\langle r_p\rangle}\int_A \chi_p(\boldsymbol{x})\cdot\delta\boldsymbol{u}\, r dA \tag{5.2}$$

$$= \sum_p \int_A \frac{\dot{\boldsymbol{p}}_{\boldsymbol{p}}}{A_p\langle r_p\rangle}\chi_p(\boldsymbol{x})\cdot\delta\boldsymbol{u}\, r dA + \sum_p \int_A \boldsymbol{\sigma}_p\chi_p(\boldsymbol{x})\cdot\nabla\delta\boldsymbol{u}\, r dA \tag{5.3}$$

where A_p is the particle area in the $r-z$ plane, $\langle r_p\rangle$ is the average radial position of the particle, and particle mass has been redefined to be the mass per radian or $m_p = \rho_p A_p\langle r_p\rangle$. The particle basis functions have the new normalization of

$$A_p\langle r_p\rangle = \int_A \chi_p(\boldsymbol{x})\, r dA \qquad \text{where} \qquad \langle r_p\rangle = \frac{1}{A_p}\int_{A_p} r\, dA \tag{5.4}$$

The first four results above after expansion in the grid-based shape functions are identical except that definition of S_{ip} changes to

$$S_{ip} = \frac{1}{A_p\langle r_p\rangle}\int_A \chi_p(\boldsymbol{x})N_i(\boldsymbol{x})\, r dA \tag{5.5}$$

The stress term needs some extra work by evaluating $\nabla\delta\boldsymbol{u}$ in cylindrical coordinates, which is:

$$\nabla\delta\boldsymbol{u} = \begin{pmatrix} \frac{\partial\delta u_r}{\partial r} & \frac{\partial\delta u_r}{\partial z} & 0 \\ \frac{\partial\delta u_z}{\partial r} & \frac{\partial\delta u_z}{\partial z} & 0 \\ 0 & 0 & \frac{\delta u_r}{r} \end{pmatrix} \tag{5.6}$$

The stress term evaluates to

$$\begin{aligned}\sum_i\sum_p \int_A \boldsymbol{\sigma}_p\chi_p(\boldsymbol{x})\cdot\delta\boldsymbol{u_i}\nabla N_i(\boldsymbol{x})\, r dA \;=\; & \sum_i\sum_p A_p\langle r_p\rangle \begin{pmatrix} \sigma_{rr} & \sigma_{rz} \\ \sigma_{rz} & \sigma_{zz}\end{pmatrix}\boldsymbol{G}_{ip}\cdot\left(\delta u_{i,z}, \delta u_{i,z}\right) \\ & + \sum_i\sum_p A_p\langle r_p\rangle\left(\sigma_{\theta\theta}, 0\right)T_{ip}\cdot\left(\delta u_{i,z}, \delta u_{i,z}\right)\end{aligned} \tag{5.7}$$

where $\boldsymbol{G}_{ip}$ is redefined and T_{ip} is a new shape function:

$$\boldsymbol{G}_{ip} = \frac{1}{A_p\langle r_p\rangle}\int_A \chi_p(\boldsymbol{x})\nabla N_i(\boldsymbol{x})\, r dA \tag{5.8}$$

$$T_{ip} = \frac{1}{A_p\langle r_p\rangle}\int_A \chi_p(\boldsymbol{x}) N_i(\boldsymbol{x})\, dA \tag{5.9}$$

$$\tag{5.10}$$

For the particle stress term, we can revise to use particle mass as follows:

$$A_p\langle r_p\rangle\boldsymbol{\sigma}_p = m_p\frac{\boldsymbol{\sigma}_p}{\rho_p} = m_p\frac{\boldsymbol{\sigma}_p}{\rho_p}\frac{\rho_0}{\rho_0} = m_p\frac{J\sigma_p}{\rho_0} = m_p\frac{\tau}{\rho_0} \tag{5.11}$$

Making use of the fact that $\delta\boldsymbol{u_i}$ is arbitrary, the summands of all terms can be equated to arrive at the controlling MPM equation on the background grid in axisymmetric calculations:

$$\frac{d\boldsymbol{p}_i}{dt} = \boldsymbol{f}_i \tag{5.12}$$

where

$$\boldsymbol{f}_i = \boldsymbol{f}_{i,T} + \sum_p\left(-\frac{m_p}{\rho_0}\begin{pmatrix}\tau_{rr} & \tau_{rz}\\ \tau_{rz} & \tau_{zz}\end{pmatrix}\cdot\boldsymbol{G}_{ip} - \frac{m_p}{\rho_0}(\tau_{\theta\theta},0)T_{ip} + m_p S_{ip}\boldsymbol{b}_p + S_{ip}\boldsymbol{F}_p\right) \tag{5.13}$$

$$\boldsymbol{f}_{i,T} = \int_{S_T}\boldsymbol{N}_i(\boldsymbol{x})\boldsymbol{T}\, dS \tag{5.14}$$

This final result is very similar to MPM in Cartesian coordinates except it uses revised shape functions, m_p, $\boldsymbol{F}_p$, and $\boldsymbol{T}$ are redefined to be quantities per radian, and there is an extra stress term in $\boldsymbol{f}_i$.

Chapter 6

GIMP Analysis for Transport

A generalized transport equation to be solved on the grid (which includes things like conduction, diffusion, pore pressure, phase-field evolution, *etc.*) is

$$C_T \frac{\partial T}{\partial t} = -\nabla \cdot \boldsymbol{q} + q_s(\boldsymbol{x}) \tag{6.1}$$

where T is any transport property (*e.g.*, temperature, concentration, pore pressure), C_T is constant that may depend on extrapolated particle states (physically C_T is a "capacity" that determines how much T changes due to fluxed quantity), $\boldsymbol{q} = -\boldsymbol{\kappa}\nabla(T/C_{ref})$ is an associated flux with units L-(units of C_T)-(units of T)/sec, and $q_s(\boldsymbol{x})$ is a source term with units (units of C_T)-(units of T)/sec. Note that flux specifies a dimensionless scaling constant, C_{ref}, if calculations need to modify T by a position-dependent scaling factor (*e.g.*, position-dependent saturation concentration in diffusion). Flux also depends on $\boldsymbol{\kappa}$, which is a "diffusion" tensor with units of L^2-(units of C_T)/sec and it may depend on position. Solving this equation in the MPM weak form gives

$$\int_V \left(C_T \frac{\partial T}{\partial t} + \nabla \cdot \boldsymbol{q} - q_s(\boldsymbol{x}) \right) w(\boldsymbol{x})\, dV = 0 \tag{6.2}$$

where $w(\boldsymbol{x})$ is an arbitrary weighting function. Using the vector identity:

$$(\nabla \cdot \boldsymbol{q})\, w(\boldsymbol{x}) = \nabla \cdot (w(\boldsymbol{x})\boldsymbol{q}) - (\nabla w(\boldsymbol{x})) \cdot \boldsymbol{q} \tag{6.3}$$

and the divergence theorem, the weak form equation becomes:

$$\int_V \left(C_T w(\boldsymbol{x}) \frac{\partial T}{\partial t} - \nabla w(\boldsymbol{x}) \cdot \boldsymbol{q} - q_s(\boldsymbol{x}) w(\boldsymbol{x}) \right) dV + \int_{\delta V} (w(\boldsymbol{x})\boldsymbol{q}) \cdot \hat{n}\, dS = 0 \tag{6.4}$$

where δV is the border of V and $\hat{n}$ is a surface normal vector.

As in GIMP for the momentum equation, we expand the transport quantities in the particle basis to get:

$$C_T \frac{\partial T}{dt} = \sum_p C_{T,p} \frac{\partial T_p}{dt} \chi_p(\boldsymbol{x}) \qquad q_s(\boldsymbol{x}) = \sum_p q_{s,p} \chi_p(\boldsymbol{x}) \qquad \boldsymbol{q} = \sum_p \boldsymbol{q}_p \chi_p(\boldsymbol{x}) \tag{6.5}$$

where subscript p denotes a particle property and $\chi_p(\boldsymbol{x})$ is the particle basis function for particle p (which is typically 1 within the deformed particle domain and zero elsewhere). Next expand weight function and its gradient in the grid shape functions:

$$w(\boldsymbol{x}) = \sum_i w_i N_i(\boldsymbol{x}) \qquad \text{and} \qquad \nabla w(\boldsymbol{x}) = \sum_i w_i \nabla N_i(\boldsymbol{x}) \tag{6.6}$$

After substituting all expansions, the weak form equation becomes

$$-\sum_i \int_{\delta V} (w_i N_i(\boldsymbol{x})\boldsymbol{q})\cdot\hat{n}\,dS = \int_V \Big\{-\sum_i\sum_p \big[\big(w_i \nabla N_i(\boldsymbol{x})\big)\cdot \boldsymbol{q}_p \chi_p(\boldsymbol{x})\big] \\ -\sum_i\sum_p q_{s,p}\chi_p(\boldsymbol{x}) w_i N_i(\boldsymbol{x}) + \sum_i\sum_p \chi_p(\boldsymbol{x}) w_i N_i(\boldsymbol{x}) C_{T,p}\frac{\partial T_p}{dt}\Big\} dV \tag{6.7}$$

Using Eqs. (1.49) and (1.51) for GIMP shape functions and arbitrary nature of $w(\boldsymbol{x})$, this equation transforms to an equation for each node:

$$\sum_p V_p C_{T,p}\frac{\partial T_p}{dt} S_{ip} = \sum_p V_p \boldsymbol{q}_p\cdot \boldsymbol{G}_{ip} + \sum_p V_p q_{s,p} S_{ip} - \int_{\delta V} (N_i(\boldsymbol{x})\boldsymbol{q})\cdot\hat{n}\,dS \tag{6.8}$$

This equation works for large deformation provided V_P is the current particle volume and Eqs. (1.49) and (1.51) use finite GIMP methods. If Eqs. (1.49) and (1.51) are replaced by uniform GIMP (or CPDI), it still approximately accounts for large deformation by using current particle volume (V_p) in the sums.

6.1 MPM Equation

We define transport content on node i and a possibly a scaled content for gradients by

$$\tau_{Ti}^{(n)} = \sum_p V_p^{(n)} C_{T,p} T_p^{(n)} S_{ip}^{(n)} \quad \text{and} \quad c_{rel,i}^{(n)} = \sum_p V_p^{(n)} C_{T,p} C_{rel,p}^{(n)} S_{ip}^{(n)} \tag{6.9}$$

with units L^3-(units of C_T)-(units of T). In a momentum analog, transport content on a node is momentum on node i. A transport content equation, as scalar analog of the momentum equation, can be written as

$$\frac{d\tau_{Ti}^{(n)}}{dt} = Q_i^{(n)} + Q_{i,q}^{(n)} \tag{6.10}$$

where

$$Q_i^{(n)} = \sum_p V_p^{(n)}\Big(\boldsymbol{q}_p^{(n)}\cdot \boldsymbol{G}_{ip}^{(n)} + q_{s,p}^{(n)} S_{ip}^{(n)}\Big) \tag{6.11}$$

$$Q_{i,q}^{(n)} = -\int_{\delta V} (N_i(\boldsymbol{x})\boldsymbol{q})\cdot\hat{n}\,dS \tag{6.12}$$

are total transport flows (or transport forces) with units of L^3-(units of C_T)-(units of T)/sec. The first flow is internal flow while the second is flow at the boundaries due to flux boundary conditions. The $\boldsymbol{q}_p^{(n)}$ term is a specific flux term with units L-(units of C_T)-(units of T)/sec. The $q_{s,p}^{(n)}$ is particle source term with units (units of C_T)-(units of T)/sec. The particle flux is analogous to particle specific stress in the momentum equation, but rather then track and update $\boldsymbol{q}_p^{(n)}$ on the particle (as done for stress), it is calculated on each time step using the possibly-scale transport value

$$\boldsymbol{q}_p^{(n)} = -\boldsymbol{\kappa}_p^{(n)}\nabla\frac{T_p^{(n)}}{C_{rel,p}^{(n)}} = -\boldsymbol{\kappa}_p^{(n)}\sum_i \frac{T_i^{(n)}}{C_{rel,i}^{(n)}}\boldsymbol{G}_{ip}^{(n)} \tag{6.13}$$

where $\boldsymbol{\kappa}_p^{(n)}$ is a specific "diffusion" tensor with units L^2-(units of C_T)/sec. The nodal transport properties are defined by

$$T_i^{(n)} = \frac{\tau_{Ti}^{(n)}}{c_i^{(n)}} \quad \text{and} \quad C_{rel,i}^{(n)} = \frac{c_{rel,i}^{(n)}}{c_i^{(n)}} \qquad \text{where} \qquad c_i^{(n)} = \sum_p V_p^{(n)} C_{T,p} S_{ip}^{(n)} \tag{6.14}$$

is extrapolated transport volume (or volume weighted by $C_{T,p}$) with units L^3-(units of C_T).

6.1.1 Momentum and Particle Updates

The transport content or temperature updates on a node are:

$$\tau_{Ti}^{(n+1)} = \tau_{Ti}^{(n)} + \left(Q_i^{(n)} + Q_{i,q}^{(n)}\right)\Delta t = \tau_{Ti}^{(n)} + c_i^{(n)} v_{Ti}^{(n)} \Delta t \tag{6.15}$$

$$T_i^{(n+1)} = T_i^{(n)} + \frac{Q_i^{(n)} + Q_{i,q}^{(n)}}{c_i^{(n)}}\Delta t = T_i^{(n+1)} + v_{Ti}^{(n)} \Delta t \tag{6.16}$$

where the nodal transport velocity is

$$v_{Ti}^{(n)} = \frac{T_i^{(n+1)} - T_i^{(n)}}{\Delta t} = \frac{Q_i^{(n)} + Q_{i,q}^{(n)}}{c_i^{(n)}} \tag{6.17}$$

All transport updates are using FLIP (because PIC causes "damping" in the form of numerical diffusion. The FLIP update is

$$T_{p,1}^{(n+1)} = T_p^{(n)} + v_{T,g\to p,1}^{(n)} \Delta t \qquad \text{where} \qquad v_{T,g\to p,1}^{(n)} = \sum_i v_{Ti}^{(n)} S_{ip}^{(n)}$$

extrapolates rate (units of T)/sec) to the particle (much like mechanics extrapolates acceleration).

Like particle stresses, particle transport property can develop variations within a cell. Because of these variations, particle transport does not give the best measure of the local transport property when implementing features such as particle properties that depend on transport property. To implement transport-property-dependent features on time step n, it is better to assume the transport property at the particle is equal to

$$T_{g\to p}^{(n)} = \sum_i T_i^{(n)} S_{ip}^{(n)} \tag{6.18}$$

In other words, the particle transport property evolves as described above, but whenever some other feature of the code needs to know the transport property at the particle, it should use the grid-based transport property and not the evolving particle transport property.

6.1.2 Material Constitutive Law Coupling

Some transport properties may be associated with material model that changes the transport value on a particle. We define an "adiabatic" transport term as $dT_{p,0}/dt$ as property change rate for conditions with no transport (*i.e.*, when no flux). Two options for inputting this coupling term are to treat is as a transport source term that is extrapolated to the grid in the $q_{s,p}^{(n)}$ term on the grid or to treat as transport change in the particle update. The source term method would be done when extrapolating transport forces to the grid, but typically $dT_{p,0}/dt$ is calculated in strain updates and not available during force

extrapolation. The second approach is done in the particle update where transport property update becomes:

$$T_{p,FLIP}^{(n+1)} = T_p^{(n)} + v_{T,g\to p}^{(n)}\Delta t + \frac{dT_{p,0}}{dt}\Delta t \tag{6.19}$$

using the preferred transport velocity. This method is used in code and probably both more efficient and more accurate than adding source term during force extrapolations to the grid.

One issue is that constitutive law calculations might be done before, after, and both before and after the one particle update. To account for all these options, the $dT_{p,0}/dt$ has to be buffered. Let $dT_{p,0,1}^{(n)}/dt$ and $dT_{p,0,2}^{(n)}/dt$ be adiabatic transport term found in strain update before and after the particle update, then the final particle update for three different update methods become:

$$\texttt{USF} \qquad T_{p,FLIP}^{(n+1)} = T_p^{(n)} + v_{T,g\to p}^{(n)}\Delta t + \frac{dT_{p,0,1}^{(n)}}{dt}\Delta t \tag{6.20}$$

$$\texttt{USL} \qquad T_{p,FLIP}^{(n+1)} = T_p^{(n)} + v_{T,g\to p}^{(n)}\Delta t + \frac{dT_{p,0,2}^{(n-1)}}{dt}\Delta t \tag{6.21}$$

$$\texttt{USAVG} \qquad T_{p,FLIP}^{(n+1)} = T_p^{(n)} + v_{T,g\to p}^{(n)}\Delta t + \left(\frac{dT_{p,0,2}^{(n-1)}}{dt} + \frac{dT_{p,0,1}^{(n)}}{dt}\right)\frac{\Delta t}{2} \tag{6.22}$$

$$\tag{6.23}$$

The $1/2$ on last term is because each of the prior two updates was founding using half the total time step for the simulations. Other options might be to generate heat in a custom tasks - $dT_{p,c}/dt$. With the above buffered approach, a custom-task temperature change can be added to the buffer and it will be used on the next particle update.

6.1.3 Transport Time Step

For explicit integration of transport equation, the time step must be

$$\Delta t \le \left(\frac{\Delta x}{2}\right)^2 \frac{C_T}{\max(\boldsymbol{\kappa})} \tag{6.24}$$

where Δx is minimum cell dimension and $\max(\boldsymbol{\kappa})$ is maximum value from the "diffusion" tensor. Note the adding of source terms may affect the time step needs. For example, a rapid source will need a smaller time step.

6.2 GIMP Analysis for Conduction

When modeling conduction, the transport property is T or temperature (units of K or Kelvin) and the other properties are in Table 6.1. The transport equation is:

$$\rho C_v \frac{\partial T}{\partial t} = -\nabla\cdot\boldsymbol{q} + q_s(\boldsymbol{x}) \tag{6.25}$$

where $\boldsymbol{q} = -\boldsymbol{\kappa}\nabla T$ and C_v is heat capacity per unit mass with SI units J/(kg-K). Table 6.1 has the full translations for conduction analysis.

Three extra options are used when modeling conduction. First is to model heat due to friction. In this calculation, frictional work in time step is calculated in heat energy units (J in SI units). The friction heat rate, $Q_{i,f}^{(n)}$, is heat work divided by time step and it is added as a heat source flux condition.

Table 6.1: Transport terms for modeling heat conduction

Quantity	Name	SI (m-kg-sec)	mm-g-sec
$C_T = \rho C_v$	density×heat capacity	J/(K-m^3)	nJ/(K-mm^3)
$\boldsymbol{\kappa}$	conductivity	J/(m-K-sec)	nJ/(mm-K-sec)
$\boldsymbol{q}_p^{(n)}$	heat flux	J/(m^2-sec)	nJ/(mm^2-sec)
$q_s(\boldsymbol{x})$	heat sources	J/(m^3-sec)	nJ/(mm^3-sec)
$\tau_{Ti}^{(n)}$	thermal content	J	nJ
$c_i^{(n)}$	nodal heat capacity	J/K	nJ/K
$Q_i^{(n)}$ and $Q_{i,q}^{(n)}$	heat flows	J/sec	J/sec

Second is to model crack tip heating, which is done by adding q_c heat flows to represent heat from crack tip heating. Heat is added to all nodes seen by the crack tip particle. Because that heating is implemented on massless crack particles, the heat added to each node uses modified shape functions:

$$\overline{Q_{i,c}^{(n)}} = q_c S_{ic}^{(n)*} = f_H \frac{G_c t_c \Delta a}{n\Delta t} S_{ic}^{(n)*} \tag{6.26}$$

where c means crack particle, G_c is the total energy release per unit area, and $t_c \Delta a$ is the total area (with t_c being crack thickness). This energy is released over n steps or total time $n\Delta t$ giving the total heating rate of $q_c = f_H G_c t_c \Delta a/(n\Delta t)$. The shape function spreads this heat over nearby nodes with partition of unity, but because crack particles may interact with inactive nodes, the shape functions are renormalized to be partition of unity by dividing by their sum over active nodes only (as indicated by the * on the shape function). The parameter f_H is an option to allow only a fraction of the energy to be converted into heat.

The third option is adiabatic heating due to compression, which is given by:

$$\frac{dT_{p,0}}{dt} = -\frac{\mathsf{M} \cdot \nabla \boldsymbol{v} T}{\rho C_v} \tag{6.27}$$

or in one time step using $\nabla \boldsymbol{v} = \nabla \boldsymbol{u}/\Delta t$ gives:

$$\frac{dT_{p,0}}{dt}\Delta t = -\frac{\mathsf{M} \cdot \nabla \boldsymbol{u} T}{\rho C_v} \tag{6.28}$$

6.3 GIMP Analysis for Diffusion

When modeling diffusion, the transport property is a dimensioness "concentration potential" $\bar{c} = c/c_{ref}$ that is related to chemical potential by

$$\mu = \mu^{\ominus} + RT \ln \bar{c}$$

Here $\mu^{\ominus}$ is a reference chemical potential that is the chemical potential when $c = c_{ref}$. In general (but maybe not always), c_{ref} is set to a maximum saturation potential and $\bar{c}$ is dimensionless value from 0 to 1. The transport equation for diffusion becomes:

$$\frac{\partial \bar{c}}{\partial t} = -\nabla \cdot \boldsymbol{q} + q_s(\boldsymbol{x}) \tag{6.29}$$

Table 6.2: Transport terms for modeling diffusion

Quantity	Name	SI (m-kg-sec)	mm-g-sec
$C_T = 1$	none	none	none
$\boldsymbol{\kappa} = \mathsf{D}$	diffusion tensor	m^2/sec	mm^2/sec
$\boldsymbol{q}_p^{(n)}$	solvent flux	m/sec	mm/sec
$q_s(\boldsymbol{x})$	solvent sources	$1/sec$	$1/sec$
$\tau_{Ti}^{(n)}$	solvent content	m^3	mm^3
$c_i^{(n)} = V_i^{(n)}$	solvent capacity	m^3	mm^3
$Q_i^{(n)}$ and $Q_{i,g}^{(n)}$	solvent flows	m^3/sec	mm^3/sec

where $\boldsymbol{q} = -\mathsf{D}\nabla\bar{c}$ with D as the diffusion tensor. The translations are in Table 6.2. If c_{ref} depends on particle state, this analysis must change to

$$\frac{\partial \bar{c}}{\partial t} = \nabla \cdot \left(\bar{c}_{rel} \mathsf{D} \nabla \frac{\bar{c}}{\bar{c}_{rel}} \right) + q_s(\boldsymbol{x}) \tag{6.30}$$

where c_{ref} used in finding $\bar{c}$ is any constant reference concentration and $\bar{c}_{rel} = c_{ref,p}/c_{ref}$ is actual reference concentration relative to the chosen constant reference concentration. This approach is needed, for example, when coupling to phase field evolution where the current phase field value models a change in the material point such that the saturation concentration changes. It could also model a material where c_{ref} depends on particle temperature.

The source terms, $q_s(\boldsymbol{x})$ can either be adding when extrapolating forces to the grid or can be added as an "adiabatic" style source generated by the particle. The former is done by implementing `GetMatDiffusionSource()`. This method is automatically called for all subclasses of `DiffusionTask`. The base material class returns zero, but other materials can override and return a source term. The call provides a particle volume (`Vp`), shape function (`sh`), and gradient shape functions (`dshdi`). The resulting source must be scaled by `Vp*sh` to work correctly in force calculations. Alternatively, a material can implement `GetParticleDiffusionSource()`. This method, however, it not automatically called. It must be called in all the material's methods for modeling constitutively laws, otherwise that source term will not take effect. Such sources typically depend on particle state and a list of parameters can be passed as needed in an array of doubles. The resulting source must be scaled by $\Delta t/C_T$ using Δt passed as a parameter and not the global time step (this need is to handle methods that call constitutive law models once or twice each time step). Finally, both these methods are pass the diffusion class pointer such that a single method can handle sources in material that might implement multiple diffusion processes (and last term must be true to added this particle source to a buffer)

Particle source terms are handled differently when using the `TRANSPORT_ONLY` method. Because this method does not call constitutive laws, `GetParticleDiffusionSource()` is instead called automatically in the particle update process. The optional parameters pointer will be `NULL` and the last term will be false. The method should still scale the appropriate source by $\Delta t/C_T$.

Table 6.3: Transport terms for modeling pore pressure

Quantity	Name	SI (m-kg-sec)	mm-g-sec
$k = \frac{1}{Q}$	Biot Q	1/Pa	1/Pa
$\boldsymbol{\kappa} = \mathsf{k}$	permeability tensor	m^2/(Pa-sec)	mm^2/(Pa-sec)
$q_s(\boldsymbol{x})$	pressure sources	1/sec	1/sec
$m_{Tp}^{(n)} = \frac{V_p}{Q}$	pressure mass	m^3/Pa	mm^3/Pa
$p_{Ti}^{(n)}$	pressure momentum	m^3	mm^3
$Q_i^{(n)}$ and $Q_{i,g}^{(n)}$	pressure flows	m^3/sec	mm^3/sec
$\boldsymbol{q}_p^{(n)}$	pressure flux	m/sec	mm/sec
$\bar{q}_{s,p}^{(n)}$	solvent source	m^3/(kg-sec)	mm^3/(g-sec)

6.4 GIMP Analysis for Pore Pressure

When modeling pore pressure, the transport property is pore pressure, p, with units of Pa of force per unit area. The transport equation is:

$$\frac{1}{Q}\frac{\partial p}{\partial t} = -\nabla \cdot \boldsymbol{q} + q_s(\boldsymbol{x}) \tag{6.31}$$

where $\boldsymbol{q} = -\mathsf{k}\nabla p$ with k as the permeability tensor (or Darcy's tensor times fluid viscosity). The translations are in Table 6.3.

Pore pressure couples stresses in pore pressure with an "adiabatic" term. Allowing for anisotropy, the term is

$$\frac{dp_{p,0}}{dt} = -\boldsymbol{\alpha} \cdot \frac{d\boldsymbol{\varepsilon}}{dt} \tag{6.32}$$

where $\boldsymbol{\alpha} = (\alpha_x, \alpha_y, \alpha_z, 0, 0, 0)$ allows for anisotropic Biot properties. If the material is isotropic, $\alpha_i = \alpha$ and the adiabatic term is:

$$\frac{dp_{p,0}}{dt} = -\alpha \frac{d\boldsymbol{\varepsilon}_{kk}}{dt} \tag{6.33}$$

where repeated index is summed or in one time step with volume change ΔV:

$$\frac{dp_{p,0}}{dt} = -\alpha \frac{\Delta V}{V} \tag{6.34}$$

Although this term could be added by implementing in `GetParticleDiffusionSource()`, it was added to the code before that method was available. Instead, it is implement by each material supporting poroelasticity calling `UndrainedPressureIncrement()`. This method scales the above terms by $\Delta t/C_T = Q\Delta t$ (and note that the Δt is already included in strain increments) and adds to the butter term.

6.5 Time Dependent Phase Field Equation

Following Miehe, the viscous version of the phase field equation is

$$\eta \frac{d\phi}{dt} = G_c \ell \nabla^2 \phi - \frac{G_c}{\ell}\phi - g'(\phi)\mathscr{H}$$

$$\eta\frac{d\phi}{dt} = G_c\ell\nabla^2\phi - \frac{G_c}{\ell}\left(\phi + g'(\phi)\frac{\mathscr{H}\ell}{G_c}\right)$$

Where ϕ is a scalar damage parameter that is 0 for undamaged and increases to 1 when failed. Other papers instead define $c = 1-\phi$ where c decays from 1 for undamaged to 0 when failed. In terms of c, the equation transforms to

$$\eta\frac{dc}{dt} = 2G_c\ell_0\nabla^2 c - \left(\frac{G_c}{2\ell_0}c + g'(c)\mathscr{H}\right) + \frac{G_c}{2\ell_0}$$

where $\ell_0 = \ell/2$ is commonly used on such papers instead of ℓ from Miehe. Both Miehe and others take $g'(\phi)$ (or $g'(c)$) to be linear functions, but this equation could be solved with non-linear functions as well. This equation can be solved in MPM by standard, explicit transport analysis. A problem, however, is that ideally η would be made small to approach the time-independent results more quickly. But if η is too small, the time step needed may be limiting factor in the analysis.

To avoid small times steps, it might be worthwhile using an implicit method. To implement an implicit solver using backward Euler, write the left hand side as $\eta(c^{n+1}-c^n)/\Delta t$ and equate to right hand side evaluated for c^{n+1} to get

$$\left(\frac{G_c}{2\eta\ell_0}c^{n+1} + \frac{c^{n+1}}{\Delta t} + \frac{g'\left(c^{n+1}\right)\mathscr{H}}{\eta}\right) - \frac{2G_c\ell_0}{\eta}\nabla^2 c^{n+1} = \frac{G_c}{2\eta\ell_0} + \frac{c^n}{\Delta t}$$

or

$$\left(\frac{2\ell_0 g'\left(c^{n+1}\right)\mathscr{H}M_c}{G_c} + \left(M_c + \frac{1}{\Delta t}\right)c^{n+1}\right) - 4\ell_0^2 M_c\nabla^2 c^{n+1} = M_c + \frac{c^n}{\Delta t}$$

where $M_c = G_c/2\eta\ell_0$. If $g'(c) = 2(1-k)c$, which is usual assumption in phase field fracture, this equation matches Eq. (6) in Wolper *et al* (2019) (with their $R_c = \eta$ and first $r = k$). They claim this equation was "constructed using Ginzburg-Landau theory," but straight-forward backward-Euler methods on Miehe equation gives the same results. Expanding this equation in MPM method gives a linear equation provided $g'(c)$ is linear. That linear equation can be solved iteratively using conjugate gradient methods. Proceeding when $g'(c)$ is nonlinear might need different methods, such as incrementally linear or return to explicit methods.

6.5.1 Generic Phase Field Equation

The above section was written for phase-field fracture in isotropic materials. Other phase field methods will have similar equation written generically as

$$\frac{d\phi}{dt} = \mathscr{M}\left(\epsilon\Delta_\Gamma\phi - \frac{1}{\epsilon}F'(\phi)\right) + Q_p$$

Phase-field fracture is special case of this generic equation where

$$\mathscr{M} \to \frac{G_c}{\eta}, \quad \Delta_\Gamma\phi \to \nabla^2\phi, \quad \epsilon \to \ell, \quad F'(\phi) \to \phi, \quad Q_p \to \frac{-g'(\phi)\mathscr{H}}{\eta}$$

where $Delta_\Gamma\phi \to \nabla^2\phi$ is for isotropic phase diffusion but could be changed to handle anisotropic diffusion. In the isotropic case, this result becomes standard isotropic diffusion. The only questions how to handle the "source" terms with three options being;

1. Handle all in `GetParticleDiffusionSource()` by adding

$$\left(-\frac{\mathscr{M}}{\epsilon}F'(\phi)+Q_p\right)\Delta t$$

   Note that this generic phase field equation has $C_T = 1$.

2. Split the two source terms such that

$$-\frac{\mathscr{M}}{\epsilon}F'(\phi)C_T V_p s$$

   is added in `GetMatDiffusionSource()` (where V_p is particle volume and s is shape function) and then add $Q_p\Delta t$ only in `GetParticleDiffusionSource()`.

3. Handle all in `GetMatDiffusionSource()` by adding

$$\left(-\frac{\mathscr{M}}{\epsilon}F'(\phi)+Q_p\right)C_T V_p s$$

In trial runs for phase field fracture runs, options 1 and 2 appear the same while 3 is different. Option 2 is likely a good choice especially when Q_p is some particle state driving phase evolution.

6.6 Time Independent Phase Field Equation

In phase field fracture, the time-independent phase field equation for ϕ is:

$$G_c\ell\nabla^2\phi-\frac{G_c}{\ell}\phi-g'(\phi)\mathscr{H}=0$$

where G_c is toughness, ℓ is length scale, $g'(\phi)$ is derivative of damage function, and $\mathscr{H}$ is history variable for tensile energy (see papers by Miehe *at al.* (2010) or MPM version in Kakouris and Triantafyllou (2017) recast using $c = 1-\phi$ and $\ell_0 = \ell/2$). Nearly all prior work uses $g(\phi) = (1-k)(1-\phi)^2 + k$ with $g'(\phi) = -2(1-k)(1-\phi)$ (where k is small factor for stability), but left general here to see what is possible. Solving this equation in the MPM weak form gives

$$\int_V\left(\nabla\cdot G_c\ell\nabla\phi-\frac{G_c}{\ell}\phi-g'(\phi)\mathscr{H}\right)w(\boldsymbol{x})\,dV=0$$

where $w(\boldsymbol{x})$ is an arbitrary weighting function. Using the vector identity:

$$(\nabla\cdot G_c\ell\nabla\phi)\,w(\boldsymbol{x})=\nabla\cdot(w(\boldsymbol{x})G_c\ell\nabla\phi)-(\nabla w(\boldsymbol{x}))\cdot G_c\ell\nabla\phi$$

and the divergence theorem, the weak form equation becomes:

$$\int_V\left((-\nabla w(\boldsymbol{x}))\cdot G_c\ell\nabla\phi-\frac{G_c}{\ell}\phi w(\boldsymbol{x})-g'(\phi)\mathscr{H}w(\boldsymbol{x})\right)dV+\int_{\delta V}(w(\boldsymbol{x})G_c\ell\nabla\phi)\cdot\hat{n}\,dS=0$$

The boundary integral is flux of damage on the surface, which we take as always zero. Expand the field quantities in the particle basis and weighting function in grid shape functions to get:

$$0=\int_V\left\{\sum_i\sum_p\left[-\left(w_i\nabla N_i(\boldsymbol{x})\right)\cdot G_{c,p}\ell_p\nabla\phi_p\chi_p(\boldsymbol{x})\right]-\sum_i\sum_p\left(\frac{G_{c,p}}{\ell_p}\phi_p+g'(\phi_p)\mathscr{H}_p\right)\chi_p(\boldsymbol{x})w_iN_i(\boldsymbol{x})\right\}dV$$

Using Eqs. (1.49) and (1.51) for GIMP shape functions and arbitrary nature of $w(\boldsymbol{x})$, this equation transforms to an equation for node i:

$$\sum_p V_p \left(G_{c,p}\ell_p \nabla\phi_p \cdot \boldsymbol{G}_{ip} + \left(\frac{G_{c,p}}{\ell_p}\phi_p + g'(\phi_p)\mathscr{H}_p \right) S_{ip} \right) = 0 \tag{6.35}$$

Now expand in grid basis

$$\phi_p = \sum_j \phi_j S_{jp} \quad \text{and} \quad \nabla\phi_p = \sum_j \phi_j \boldsymbol{G}_{jp}$$

This substitution will create a term $g'\left(\sum_j \phi_j S_{jp}\right)$ which will prevent this analysis from resulting in a linear equation *unless* $g'(\phi_p) = a + b\phi_p$ or it is linear in ϕ_p. Using requirements from literature that $g(0) = 1$, $g(1) = k$ (for a small residual stiffness) leads to

$$g(\phi) = 1 + a\phi - (1-k+a)\phi^2 \qquad \text{and} \qquad g'(\phi) = a - 2(1-k+a)\phi$$

or redefine $a \to -1 - a/(1-k)$ to get

$$g(\phi) = (1-k)(1-\phi)(1-a\phi) + k \qquad \text{and} \qquad g'(\phi) = (1-k)(-(1+a) + 2a\phi)$$

Because $g'(\phi)$ is related to dissipated energy, it must remain negative on the interval $0 \le \phi \le 1$. Requiring $g'(0) \le 0$ and $g'(1) \le 0$ results in range of acceptable a values of $-1 < a \le 1$. While any such a is acceptable, the bulk of the literature uses $a = 1$ resulting in $g(\phi) = (1-k)(1-\phi)^2 + k$ and $g'(\phi) = -2(1-k)(1-\phi)$. Here we keep a to be potentially more general (and note that non-quadratic $g(\phi)$ cannot be solved in this linear analysis, but should work in an explicit diffusion analysis). Using quadratic $g(\phi)$ changes previous result to

$$\sum_p V_p \left(G_{c,p}\ell_p \nabla\phi_p \cdot \boldsymbol{G}_{ip} + \left(\frac{G_{c,p}}{\ell_p} + 2(1-k)a\mathscr{H}_p \right) \phi_p S_{ip} \right) = \sum_p (1+a)(1-k)V_p \mathscr{H}_p S_{ip}$$

Now complete the grid basis expansion to get

$$\sum_j \sum_p V_p \left(G_{c,p}\ell_p \boldsymbol{G}_{jp} \cdot \boldsymbol{G}_{ip} + \left(\frac{G_{c,p}}{\ell_p} + 2(1-k)a\mathscr{H}_p \right) S_{jp} S_{ip} \right) \phi_j = \sum_p (1+a)(1-k)V_p \mathscr{H}_p S_{ip}$$

In matrix form:

$$\mathsf{K}\boldsymbol{\phi} = \boldsymbol{r}$$

where

$$K_{ij} = \sum_p V_p \left(G_{c,p}\ell_p \boldsymbol{G}_{jp} \cdot \boldsymbol{G}_{ip} + \left(\frac{G_{c,p}}{\ell_p} + 2(1-k)a\mathscr{H}_p \right) S_{jp} S_{ip} \right) \quad \text{and} \quad r_i = \sum_p (1+a)(1-k)V_p \mathscr{H}_p S_{ip}$$

If $G_{c,p}$ and ℓ_p are the same for all particles, we can multiply through by $\ell_p / G_{c,p}$ to get units of volume on both sides:

$$K_{ij} = \sum_p V_p \left(\ell_p^2 \boldsymbol{G}_{jp} \cdot \boldsymbol{G}_{ip} + \left(1 + a\mathscr{F}_p\right) S_{jp} S_{ip} \right) \quad \text{and} \quad r_i = \sum_p \frac{1+a}{2} \mathscr{F}_p V_p S_{ip}$$

where $\mathscr{F}_p = 2(1-k)\ell_p \mathscr{H}_p / G_{c,p}$, which is dimensionless metric on energy input, $\mathscr{H}_p$, divided by energy required to fail the material, $G_{c,p}/(2\ell_p)$. If properties vary in space, the equation has to be left in previous form (or at least that is form that arises from energy analysis) leading to

$$K_{ij} = \sum_p \frac{G_{c,p}V_p}{\ell_p} \left(\ell_p^2 \boldsymbol{G}_{jp} \cdot \boldsymbol{G}_{ip} + \left(1 + a\mathscr{F}_p\right) S_{jp} S_{ip} \right) \quad \text{and} \quad r_i = \sum_p \frac{G_{c,p}(1+a)\mathscr{F}_p}{2\ell_p} V_p S_{ip}$$

Now each term has units of energy.

Some literature methods solve instead for an "integrity" factor $c_i = 1 - \phi_i$. The equation becomes

$$\sum_j K_{ij} c_j = -r_i + \sum_j K_{ij}$$

where

$$\sum_j K_{ij} = \sum_p V_p \left(\ell_p^2 \left(\sum_j \boldsymbol{G}_{jp} \right) \cdot \boldsymbol{G}_{ip} + \left(1 + a\mathscr{F}_p\right) \left(\sum_j S_{jp} \right) S_{ip} \right) = \sum_p \left(1 + a\mathscr{F}_p\right) V_p S_{ip}$$

because of partition of unity of shape functions and sum of derivatives is zero. The final equation is

$$\mathsf{K}\boldsymbol{c} = \boldsymbol{R} \quad \text{where} \quad R_i = \sum_p \left(1 - \frac{1-a}{2} \mathscr{F}_p \right) V_p S_{ip}$$

and add $G_{c,p}/\ell_p$ to all terms when using energy equation. This result is matches Kakouris and Triantafyllou (2017) (where their $\ell_0 = \ell/2$) for the special case of $a = 1$).

Gaussian elimination works for this banded and symmetric matrix, but is slow for large problems. Minimizing the bandwidth will likely require renumbering the nodes. Perhaps an iterative solution will be more efficient?

Possible algorithm is

1. Renumber nodes to minimize bandwidth. I think this mapping can be done once at the beginning using standard finite element methods. The result will likely need two maps: Map i in the grid find j in the banded matrix. Once problem is solved, need map from j in the banded matrix to i in the grid.

2. On each time step, calculate the bandwidth of the problem as maximum of $\max(i-j)_p$ where $(i-j)_p$ is maximum node difference among nodes seen by practical p. This may not change often or may only change after large deformation. It might have an upper limit when using uGIMP.

3. If b is bandwidth and n is number of active nodes, allocate K as n vectors of length b and zero all terms. Allocate vector of length n for the right hand side.

4. Loop over particles and find elements of K and $\boldsymbol{r}$. Note that K_{ij} is stored in $K[i][i-j+1]$ and only need to fill in values with $i \geq j$ (my legacy code used 1-based numbering).

5. Final results can either by solved by Gaussian elimination or by iterative method (*e.g.*, weighted Jacobi method). An iterative method can start with previous values extrapolated to the grid.

6. Extrapolate nodal values to the particles.

Returning to Eq. 6.35, but now allowing arbitrary particle value $\mathscr{J}_p = g'(\phi_p)\mathscr{H}_p$ gives

$$\sum_p V_p \left(G_{c,p} \ell_p \nabla\phi_p \cdot \boldsymbol{G}_{ip} + \left(\frac{G_{c,p}}{\ell_p} \phi_p + \mathscr{J}_p \right) S_{ip} \right) = 0$$

Expanding ϕ is particle basis leads to matrix equation:

$$\mathsf{K}^* \boldsymbol{\phi} = \boldsymbol{r}^*$$

where

$$K^*_{ij} = \sum_p \frac{G_{c,p}V_p}{\ell_p}\left(\ell_p^2 \cdot \boldsymbol{G}_{ip} + S_{jp}S_{ip}\right) \quad \text{and} \quad r^*_i = -\sum_p V_p \mathscr{J}_p S_{ip}$$

Switching to $c_i = 1-\phi_i$ leads to

$$\mathsf{K}^*\boldsymbol{c} = \boldsymbol{R}^* \quad \text{where} \quad R^*_i = \sum_p V_p\left(\mathscr{J}_p - \frac{G_{c,p}}{\ell_p}\right)S_{ip}$$

Notice that MPM solution to phase-field evolution alone where particles are non-deforming and stationary, K^* is constant. If this approach works, it could be inverted once (or perhaps Gaussian elimination initial step could be done once). It is also possible, that non-linear analysis would need to write $g'(\phi) = G_{c,p}(A_p + B_p\phi)/\ell_p$ again, but now done as incrementally tangent to $g(\phi_p)$. The equation would the same as above, but left in terms of A_p and B_p terms or

$$K^*_{ij} = \sum_p \frac{G_{c,p}V_p}{\ell_p}\left(\ell_p^2\boldsymbol{G}_{jp}\cdot\boldsymbol{G}_{ip} + \left(1+B_p\mathscr{H}_p\right)S_{jp}S_{ip}\right) \quad \text{and} \quad r^*_i = -\sum_p \frac{G_{c,p}V_p}{\ell_p}A_p\mathscr{H}_p S_{ip}$$

and when using c_i

$$R^*_i = \sum_p \frac{G_{c,p}V_p}{\ell_p}\left(A_p\mathscr{H}_p - \left(1+B_p\mathscr{H}_p\right)\right)S_{ip}$$

And final solution would need incremental methods (like non-linear FEA).

6.7 Time Independent Poisson's Equation

Imagine a Poisson's equation for electric potential (in V or J/C or $kg\ m^2/(s^3A)$ where units of C are $A\,s$) including position-dependent permittivity tensor ($\boldsymbol{\epsilon}_0$ with SI units F/m or $C/(V\ m)$ or $A^2s^4/(kg\ m^3)$) and equated to a nonzero and position-dependent charge density $\rho(\boldsymbol{x})$ (SI units C/m^3 or $A\,s/m^3$):

$$\nabla\cdot\boldsymbol{\epsilon}_0\nabla\phi + \rho(\boldsymbol{x}) = 0$$

Solving this equation in the MPM weak form starts with:

$$\int_V \left(\nabla\cdot\boldsymbol{\epsilon}_0\nabla\phi + \rho(\boldsymbol{x})\right)w(\boldsymbol{x})\,dV = 0$$

where $w(\boldsymbol{x})$ is an arbitrary weighting function. Using the vector identity:

$$\left(\nabla\cdot\boldsymbol{\epsilon}_0\nabla\phi\right)w(\boldsymbol{x}) = \nabla\cdot\left(w(\boldsymbol{x})\boldsymbol{\epsilon}_0\nabla\phi\right) - \left(\nabla w(\boldsymbol{x})\right)\cdot\boldsymbol{\epsilon}_0\nabla\phi$$

and the divergence theorem, the weak form equation becomes:

$$\int_V \left(\left(-\nabla w(\boldsymbol{x})\right)\cdot\boldsymbol{\epsilon}_0\nabla\phi + \rho(\boldsymbol{x})w(\boldsymbol{x})\right)dV + \int_{\delta V}\left(w(\boldsymbol{x})\boldsymbol{\epsilon}_0\nabla\phi\right)\cdot\hat{n}\,dS = 0$$

The boundary integral is flux on the surface which can be set using boundary conditions. Expand the field quantities in the particle basis and weighting function in grid shape functions to get:

$$0 = \int_V\left\{\sum_i\sum_p\left[-\left(w_i\nabla N_i(\boldsymbol{x})\right)\cdot\boldsymbol{\epsilon}_{0,p}\nabla\phi_p\chi_p(\boldsymbol{x})\right] + \sum_i\sum_p\rho_p\chi_p(\boldsymbol{x})w_iN_i(\boldsymbol{x})\right\}dV$$

where boundary condition term is omitted, but added back in the next step. Using Eqs. (1.49) and (1.51) for GIMP shape functions and arbitrary nature of $w(\boldsymbol{x})$, this equation transforms to an equation for node i (now with flux boundary condition):

$$\sum_p -V_p \boldsymbol{\epsilon}_{0,p} \nabla \phi_p \cdot \boldsymbol{G}_{ip} + \sum_{\partial p} \phi_{p,s}^{flux} \int_{A_s} N_i(\boldsymbol{x})\, dS = -\sum_p \rho_p V_p S_{ip}$$

where $\partial p = \{p, s : \text{particle } p \text{ has flux } \phi_{p,s}^{flux} \text{ on surface } s\}$ and A_s is the deformed area for surface s of particle p. Now evaluate the gradient from particle values:

$$\nabla \phi_p = \sum_j \phi_j \boldsymbol{G}_{jp}$$

to get

$$\sum_p \sum_j \left(V_p \boldsymbol{\epsilon}_{0,p} \boldsymbol{G}_{jp} \cdot \boldsymbol{G}_{ip} \right) \phi_j = \sum_p \rho_p V_p S_{ip} + \sum_{\partial p} \phi_{p,s}^{flux} \int_{A_s} N_i(\boldsymbol{x})\, dS$$

This form accommodates both position-dependent and anisotropic permittivity. For isotropic materials, $\boldsymbol{\epsilon}_{0,p}$ is replaced by a position dependent scalar $\epsilon_{0,p}$. In matrix form:

$$\mathsf{K}\boldsymbol{\phi} = \boldsymbol{r}$$

where

$$K_{ij} = \sum_p V_p \boldsymbol{\epsilon}_{0,p} \boldsymbol{G}_{jp} \cdot \boldsymbol{G}_{ip} \quad \text{and} \quad r_i = \sum_p \rho_p V_p S_{ip} + \sum_{\partial p} \phi_{p,s}^{flux} \int_{A_s} N_i(\boldsymbol{x})\, dS$$

Solution requires inversion of K, which is symmetric and banded (with bandwidth equal to maximum difference between nodes seen by any particle). Minimizing the bandwidth will likely require renumbering the nodes. If the particles are not moving (such as in transport only), the GIMP shape functions can be replaced by ordinary grid shape functions (potentially even using higher-order shape functions on the same grid. Perhaps an iterative solution will be more efficient?

Note when running using mm-g-s units (which are internal units when using Legacy units), potential units are 10^9 V = nV, permittivity units are 10^{-12} F/m = T(F/m) = C/(nV mm), and units of charge density are 10^{-9} C/m^3 = G(C/m^3) = C/mm^3. The permittivity of vacuum is $\epsilon_0 = 8.8541878128 \times 10^{-24}$ C/(nV mm).

Chapter 7

Consistent Units

internally, both NairnMPM and OSParticulas use mm-g-sec units system. Despite this internal use, the input of properties are not always based on these units. The two ways to enter properties are using "Legacy Units" or picking and using you own "Consistent Units" system. Currently scripts interpreted by NairnFEAMPM or NairnFEAMPMViz myst use Legacy units, but input files created directly in XML can pick a different set by adding the <ConsistentUnits> command. Some MPM units options are given in Tables 7.1 and 7.2. The units including "Alt" means their treatment in Legacy units is not consistent. Refer to documentation to see how to input properties that need those units and to see how quantities with those units are output.

7.1 Consistent Units in Static FEA

In static FEA, the only units needed are for length and force. The legacy units for NairnFEA are close to consistent using length in mm and force in N, which leads to stress and moduli in MPa = N/m^2 and interface parameters in MPa/mm. Boundary conditions are set using mm, N, and MPa for displacements, loads, and tractions, respectively. The only inconsistency is that the output energies re in J while the energy consistent with mm and N should be mJ = N-mm. The consistent units options are in Table 7.3.

Table 7.1: Consistent units for the momentum equation

Quantity	Legacy	SI (m-kg-sec)	mm-g-sec
Length	mm	m	mm
Mass	g	kg	g
Time	sec	sec	sec
Alt Time	msec	sec	sec
Density	$\frac{g}{cm^3}$	$\frac{kg}{m^3}$	$\frac{g}{mm^3}$
Velocity	$\frac{mm}{sec}$	$\frac{m}{sec}$	$\frac{mm}{sec}$
Alt Velocity	$\frac{m}{sec}$	$\frac{m}{sec}$	$\frac{mm}{sec}$
Force	N	$N = \frac{kg \cdot m}{sec^2}$	$\mu N = \frac{g \cdot mm}{sec^2}$
Pressure	MPa	$Pa = \frac{N}{m^2}$	$Pa = \frac{\mu N}{mm^2}$
Alt Strain	%	none	none
Energy	J	$J = N \cdot m$	$nJ = \mu N \cdot mm$
ERR	$\frac{J}{m^2}$	$\frac{J}{m^2}$	$\frac{nJ}{mm^2}$
Stress Int.	$MPa\sqrt{m}$	$Pa\sqrt{m}$	$Pa\sqrt{mm}$
Viscosity	cP	$\frac{kg}{m \cdot sec}$	$\frac{g}{mm \cdot sec}$

Table 7.2: Consistent units for conduction and diffusion equation

Quantity	Legacy	SI (m-kg-sec)	mm-g-sec
Heat Capacity	J/(kg-K)	J/(kg-K)	nJ/(g-K)
Conductivity	W/(m-K)	W/(m-K)	nW/(mm-K)
Heat Flux	W/m^2	W/m^2	nW/mm^2
Diffusion	m^2/sec	m^2/sec	mm^2/sec
Solvent Flux	$kg/(m^2 \cdot sec)$	$kg/(m^2 \cdot sec)$	$g/(mm^2 \cdot sec)$

Table 7.3: Consistent units for FEA

Units	Length	Force	Stress	Energy
Legacy	mm	N	MPa	J
SI	m	N	Pa	J
mm-g-sec	mm	μN	Pa	nJ
cm-g-sec	cm	$10^{-5}N = dyne$	$10^{-1}Pa = Ba$	$10^{-7}J = erg$
mm-g-msec	mm	N	MPa	mJ