Immersed Boundary Tutorial
Part 1

Aaron Barrett

July 22, 2025

Introduction

THE UNIVERSITY OF UTAH®

- What to expect:

- What not to expect:

Introduction

- What to expect:
  - Introduction to IB
  - Introduction to IBAMR
  - Run IBAMR examples
  - Visualize IBAMR output

- What not to expect:

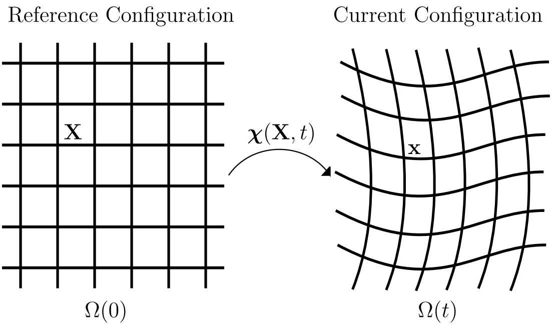
Introduction

- What to expect:
  - Introduction to IB
  - Introduction to IBAMR
  - Run IBAMR examples
  - Visualize IBAMR output
  - Basic understanding of design, example layout, input file
  - Basis to start exploring and considering implementation
- What not to expect:

Introduction

- What to expect:
  - Introduction to IB
  - Introduction to IBAMR
  - Run IBAMR examples
  - Visualize IBAMR output
  - Basic understanding of design, example layout, input file
  - Basis to start exploring and considering implementation
- What not to expect:
  - Expertise in IBAMR
  - Expertise in C++
  - Expertise in VisIt

Lagrangian vs Eulerian Variables

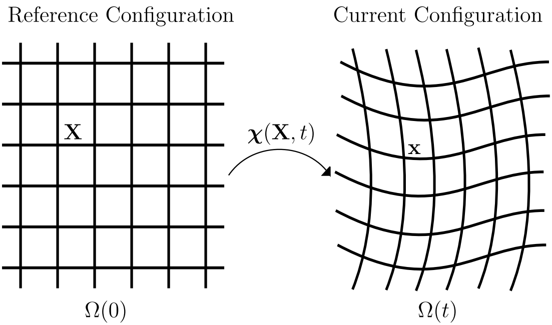


- Fluid parcels are associated with a label $\mathbf{X}$.
- Velocity in a Lagrangian frame

$$\frac{\partial \boldsymbol{\chi}(\mathbf{X}, t)}{\partial t}$$

- Spatial coordinates $\mathbf{x}$ are fixed.
- Eulerian velocity: $\mathbf{u}(\mathbf{x}, t)$.

Lagrangian vs Eulerian Variables



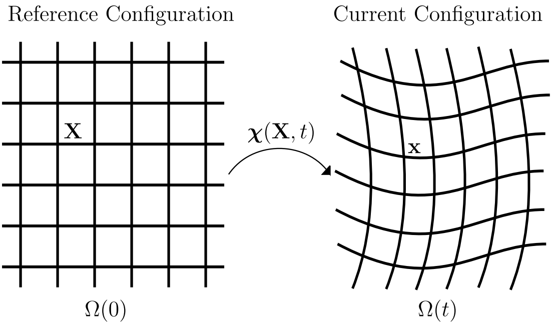
- Fluid parcels are associated with a label $\mathbf{X}$.
- Velocity in a Lagrangian frame

$$\frac{\partial \chi(\mathbf{X}, t)}{\partial t}$$

- Spatial coordinates $\mathbf{x}$ are fixed.
- Eulerian velocity: $\mathbf{u}(\mathbf{x}, t)$.
- We have that

$$\begin{aligned} \frac{\partial \chi(\mathbf{X}, t)}{\partial t} &= \mathbf{u}(\chi(\mathbf{X}, t), t) \\ &= \int_\Omega \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \chi(\mathbf{X}, t))\mathrm{d}\mathbf{x} \end{aligned}$$

Lagrangian vs Eulerian Variables

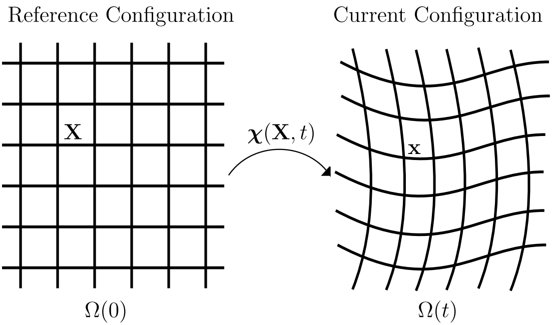


- Fluid parcels are associated with a label $\mathbf{X}$.
- Rates of change of a scalar variable F is written as $\frac{\partial}{\partial t}F(\mathbf{X}, t)$

- Spatial coordinates $\mathbf{x}$ are fixed.
- Rates of change of a scalar variable $f(\mathbf{x}, t) = f(\chi(\mathbf{X}, t), t) = F(\mathbf{X}, t)$:

$$\frac{\mathrm{d}}{\mathrm{d}t} f(\chi(\mathbf{X}, t), t) = \frac{\partial f}{\partial t} + \frac{\partial \chi}{\partial t}\frac{\partial f}{\partial \chi}$$

Lagrangian vs Eulerian Variables



- Fluid parcels are associated with a label $\mathbf{X}$.
- Rates of change of a scalar variable F is written as $\frac{\partial}{\partial t}F(\mathbf{X}, t)$

- Spatial coordinates $\mathbf{x}$ are fixed.
- Rates of change of a scalar variable $f(\mathbf{x}, t) = f(\chi(\mathbf{X}, t), t) = F(\mathbf{X}, t)$:

$$\frac{\mathrm{d}}{\mathrm{d}t} f(\chi(\mathbf{X}, t), t) = \left(\frac{\partial}{\partial t} + \mathbf{u} \cdot \nabla\right) f$$

Lagrangian vs Eulerian Variables

- We can define the acceleration of the fluid in two ways:
  - Lagrangian:
$$\frac{\partial^2 \boldsymbol{\chi}}{\partial t^2}$$
  - Eulerian:
$$\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}$$

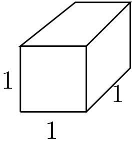
Lagrangian vs Eulerian Variables

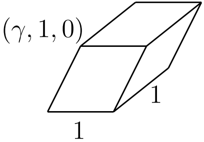
- We can define the acceleration of the fluid in two ways:
  - Lagrangian:
$$\frac{\partial^2 \chi}{\partial t^2}$$
  - Eulerian:
$$\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}$$
- The deformation gradient $\mathbb{F} = \frac{\partial \chi}{\partial \mathbf{X}}$ maps line elements $\mathrm{d}\mathbf{X}$ in the reference configuration to line elements $\mathrm{d}\mathbf{x}$ in the current configuration:

$$\mathrm{d}\mathbf{x} = \mathbb{F} \cdot \mathrm{d}\mathbf{X}$$

Lagrangian vs Eulerian Variables

- We can define the acceleration of the fluid in two ways:
  - Lagrangian:
$$\frac{\partial^2 \chi}{\partial t^2}$$
  - Eulerian:
$$\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}$$
- The deformation gradient $\mathbb{F} = \frac{\partial \chi}{\partial \mathbf{X}}$ maps line elements $\mathrm{d}\mathbf{X}$ in the reference configuration to line elements $\mathrm{d}\mathbf{x}$ in the current configuration:
$$\mathrm{d}\mathbf{x} = \mathbb{F} \cdot \mathrm{d}\mathbf{X}$$
- Consider a cube being *sheared*.

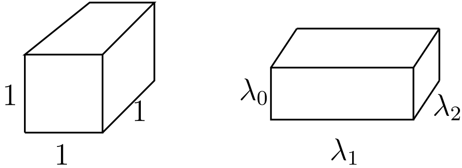




$$\mathbb{F} = \begin{pmatrix} 1 & 0 & 0 \\ \gamma & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Lagrangian vs Eulerian Variables

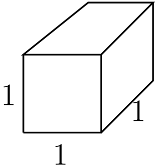
- Consider a cube being stretched (extensional deformation).

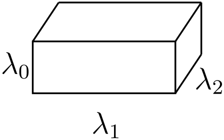


$$\mathbb{F} = \begin{pmatrix} \lambda_0 & 0 & 0 \\ 0 & \lambda_1 & 0 \\ 0 & 0 & \lambda_2 \end{pmatrix}$$

Lagrangian vs Eulerian Variables

- Consider a cube being stretched (extensional deformation).



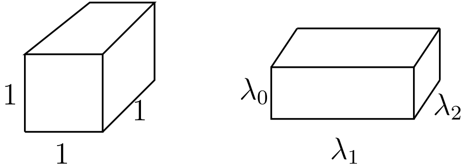


$$\mathbb{F} = \begin{pmatrix} \lambda_0 & 0 & 0 \\ 0 & \lambda_1 & 0 \\ 0 & 0 & \lambda_2 \end{pmatrix}$$

- For $\mathbb{F}$ to represent an *incompressible deformation*, we need $\lambda_0 \lambda_1 \lambda_2 = 1$.

Lagrangian vs Eulerian Variables

- Consider a cube being stretched (extensional deformation).



$$\mathbb{F} = \begin{pmatrix} \lambda_0 & 0 & 0 \\ 0 & \lambda_1 & 0 \\ 0 & 0 & \lambda_2 \end{pmatrix}$$

- For $\mathbb{F}$ to represent an *incompressible deformation*, we need $\lambda_0\lambda_1\lambda_2 = 1$.
- The Jacobian $J = \det\mathbb{F}$ measures volume changes:

$$\frac{\mathrm{d}J}{\mathrm{d}t} = (\nabla \cdot \mathbf{u})J$$

- Incompressible materials satisfy $J = 1$ or $\nabla \cdot \mathbf{u} = 0$.

Cauchy Stress Tensor

- Stress measures the response of material to strain.
- Traction on a surface with normal $\mathbf{n}$ is

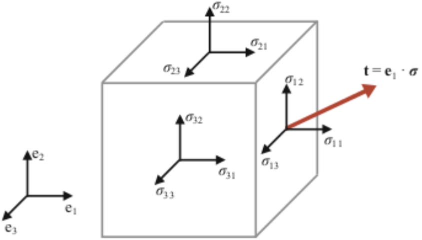
$$\mathbf{t} = \sigma \cdot \mathbf{n}$$

- Net force on surface S

$$\mathbf{F} = \int_{\partial S} \sigma \cdot \mathbf{n} \mathrm{d}S$$

- Conservation of angular momentum:

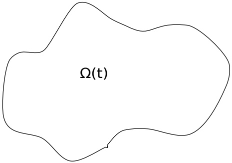
$$\sigma = \sigma^T$$



Conservation of Mass

- Consider a volume of fluid $\Omega(t)$ with density $\rho(\mathbf{x}, t)$. Total mass:

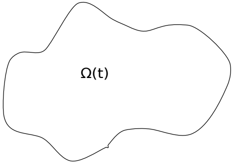
$$M(t) = \int_{\Omega(t)} \rho(\mathbf{x}, t) \mathrm{d}V$$



Conservation of Mass

- Consider a volume of fluid $\Omega(t)$ with density $\rho(\mathbf{x},t)$. Total mass:

$$M(t) = \int_{\Omega(t)} \rho(\mathbf{x},t)\mathrm{d}V$$



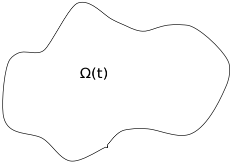
- Conservation of mass:

$$\begin{aligned} 0 &= \frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega(t)} \rho(\mathbf{x},t)\mathrm{d}V_{\mathbf{x}} = \frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega(0)} \rho(\chi(\mathbf{X},t),t)J\mathrm{d}V_{\mathbf{X}} \\ &= \int_{\Omega(0)} \left(\frac{\mathrm{D}\rho(\chi(\mathbf{X},t),t)}{\mathrm{D}t} + \rho(\chi(\mathbf{X},t),t)(\nabla\cdot\mathbf{u})\right)J\mathrm{d}V_{\mathbf{X}} \\ &= \int_{\Omega(t)} \left(\frac{\mathrm{D}\rho(\mathbf{x},t)}{\mathrm{D}t} + \rho(\mathbf{x},t)(\nabla\cdot\mathbf{u})\right)\mathrm{d}V_{\mathbf{x}} \end{aligned}$$

Conservation of Mass

- Consider a volume of fluid $\Omega(t)$ with density $\rho(\mathbf{x},t)$. Total mass:

$$M(t) = \int_{\Omega(t)} \rho(\mathbf{x},t)\mathrm{d}V$$



- Conservation of mass:

$$\begin{aligned} 0 &= \frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega(t)} \rho(\mathbf{x},t)\mathrm{d}V_{\mathbf{x}} = \frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega(0)} \rho(\chi(\mathbf{X},t),t)J\mathrm{d}V_{\mathbf{X}} \\ &= \int_{\Omega(0)} \left(\frac{\mathrm{D}\rho(\chi(\mathbf{X},t),t)}{\mathrm{D}t} + \rho(\chi(\mathbf{X},t),t)(\nabla\cdot\mathbf{u})\right)J\mathrm{d}V_{\mathbf{X}} \\ &= \int_{\Omega(t)} \left(\frac{\mathrm{D}\rho(\mathbf{x},t)}{\mathrm{D}t} + \rho(\mathbf{x},t)(\nabla\cdot\mathbf{u})\right)\mathrm{d}V_{\mathbf{x}} \end{aligned}$$

- This holds for all material volumes:

$$\frac{\mathrm{D}\rho}{\mathrm{D}t} + \rho(\nabla\cdot\mathbf{u}) = 0 \quad \text{or} \quad \frac{\mathrm{D}\rho}{\mathrm{D}t} = 0$$

Momentum Equation

- Momentum of the fluid

$$\mathbf{p}(t) = \int_{\Omega(t)} \rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t)\mathrm{d}V_{\mathbf{x}}$$

Momentum Equation

- Momentum of the fluid

$$\mathbf{p}(t) = \int_{\Omega(t)} \rho(\mathbf{x}, t)\mathbf{u}(\mathbf{x}, t)\mathrm{d}V_{\mathbf{x}}$$

- We have

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{p}(t) &= \frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega(t)} (\rho\mathbf{u})(\mathbf{x}, t)\mathrm{d}V_{\mathbf{x}} = \frac{\mathrm{d}}{\mathrm{d}t}\int_{\Omega(0)} (\rho\mathbf{u})(\chi(\mathbf{X}, t), t)J\mathrm{d}V_{\mathbf{X}} \\
&= \int_{\Omega(0)} \left(\frac{\mathrm{D}\rho}{\mathrm{D}t}\mathbf{u} + \rho\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} + \rho\mathbf{u}(\nabla\cdot\mathbf{u})\right)J\mathrm{d}V_{\mathbf{X}} \\
&= \int_{\Omega(0)} \rho\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t}J\mathrm{d}V_{\mathbf{X}} = \int_{\Omega(t)} \rho\frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t}(\mathbf{x}, t)\mathrm{d}V_{\mathbf{x}}
\end{aligned}$$

Momentum Equation

- Forces on fluid:
  - External body forces $\mathbf{f}$, e.g. gravity.
  - Surface forces $\mathbf{t} = \mathbf{n} \cdot \sigma$ e.g. viscous or elastic forces.
  - σ is the Cauchy stress tensor.

- Rate of change of momentum:

$$\frac{\mathrm{d}}{\mathrm{d}t}\mathbf{p}(t) = \int_{\Omega(t)} \rho \frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} \mathrm{d}V_{\mathbf{x}}$$

- Forces on fluid:

$$\int_{\Omega(t)} \mathbf{f} dV_{\mathbf{x}} + \int_{\partial\Omega(t)} \mathbf{t} \mathrm{d}S_{\mathbf{x}}$$

- Newton's second law:

$$\int_{\Omega(t)} \rho \frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} \mathrm{d}V_{\mathbf{x}} = \int_{\Omega(t)} \mathbf{f} \mathrm{d}V_{\mathbf{x}} + \int_{\partial\Omega(t)} \mathbf{t} \mathrm{d}S_{\mathbf{x}}$$

or

$$\rho \frac{\mathrm{D}\mathbf{u}}{\mathrm{D}t} = \mathbf{f} + \nabla \cdot \sigma$$

Newtonian Fluids

- Different fluids of interest reduce to specifying a constitutive equation for $\mathbb{\sigma}$.
- We first decompose the stress into *hydrostatic pressure* p and the *deviatoric stress* $\mathbb{\tau}$.

$$\mathbb{\sigma} = -p\mathbb{I} + \mathbb{\tau}$$

- In Newtonian fluids, the deviatoric stress $\mathbb{\tau}$ is linear in the rate of strain and the fluid is isotropic.

Newtonian Fluids

- Different fluids of interest reduce to specifying a constitutive equation for $\mathbb{\sigma}$.
- We first decompose the stress into *hydrostatic pressure* p and the *deviatoric stress* $\mathbb{\tau}$.

$$\mathbb{\sigma} = -p\mathbb{I} + \mathbb{\tau}$$

- In Newtonian fluids, the deviatoric stress $\mathbb{\tau}$ is linear in the rate of strain and the fluid is isotropic.

- Velocity gradient:

$$\frac{\partial}{\partial t}\mathbb{F} = \frac{\partial}{\partial t}\frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \frac{\partial \mathbf{u}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \mathbf{X}} = \nabla \mathbf{u} \cdot \mathbb{F}$$

- Extensional and shear flows:

$$\nabla \mathbf{u} = \begin{pmatrix} \dot{\epsilon}_1 & 0 & 0 \\ 0 & \dot{\epsilon}_2 & 0 \\ 0 & 0 & \dot{\epsilon}_3 \end{pmatrix}, \quad \nabla \mathbf{u} = \begin{pmatrix} 0 & 0 & 0 \\ \dot{\gamma} & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

- Rate of strain:

$$\dot{\mathbb{\gamma}} = \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T\right)$$

- Vorticity tensor:

$$\mathbb{\omega} = \left(\nabla \mathbf{u} - (\nabla \mathbf{u})^T\right)$$

Newtonian Fluids

- Different fluids of interest reduce to specifying a constitutive equation for $\mathbb{\sigma}$.
- We first decompose the stress into *hydrostatic pressure* p and the *deviatoric stress* $\mathbb{\tau}$.

$$\mathbb{\sigma} = -p\mathbb{I} + \mathbb{\tau}$$

- In Newtonian fluids, the deviatoric stress $\mathbb{\tau}$ is linear in the rate of strain and the fluid is isotropic.

- Rate of strain:

$$\dot{\mathbb{\gamma}} = \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T\right)$$

- Vorticity tensor:

$$\mathbb{\omega} = \left(\nabla \mathbf{u} - (\nabla \mathbf{u})^T\right)$$

- Navier-Stokes equations:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}\right) = \nabla \cdot \mathbb{\sigma}$$

$$\mathbb{\sigma} = -p\mathbb{I} + \mu\dot{\mathbb{\gamma}}, \quad \nabla \cdot \mathbf{u} = 0$$

or that

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u}\right) = -\nabla p + \mu \Delta \mathbf{u}$$

Immersed Boundary Methods

- Consider a thin, massles boundary immersed in the fluid.
- Track the interface in Lagrangian coordinates:
  - Velocity:
$$\frac{\partial \boldsymbol{\chi}(\mathbf{X}, t)}{\partial t} = \int_{\Omega} \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \boldsymbol{\chi}(\mathbf{X}, t))\mathrm{d}\mathbf{x}$$
  - Elastic energy:
$$E[\boldsymbol{\chi}] \rightarrow \text{ Depends on deformation!}$$

Immersed Boundary Methods

- Consider a thin, massles boundary immersed in the fluid.
- Track the interface in Lagrangian coordinates:
  - Velocity:

$$\frac{\partial \boldsymbol{\chi}(\mathbf{X},t)}{\partial t} = \int_{\Omega} \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x}-\boldsymbol{\chi}(\mathbf{X},t))\mathrm{d}\mathbf{x}$$

  - Elastic energy:

$$E[\boldsymbol{\chi}] \rightarrow \text{ Depends on deformation!}$$

- Forces are given by derivative of energy

$$\delta E = \int -\mathbf{F}\delta\boldsymbol{\chi}\mathrm{d}\mathbf{X}$$

Immersed Boundary Methods

- Consider a thin, massles boundary immersed in the fluid.
- Track the interface in Lagrangian coordinates:
  - Velocity:
$$\frac{\partial \boldsymbol{\chi}(\mathbf{X},t)}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x}-\boldsymbol{\chi}(\mathbf{X},t))\mathrm{d}\mathbf{x}$$
  - Elastic energy:
$E[\boldsymbol{\chi}] \rightarrow$ Depends on deformation!
- Forces are given by derivative of energy
$$\delta E = \int -\mathbf{F}\delta\boldsymbol{\chi}\mathrm{d}\mathbf{X}$$
- Ex: Let q parameterize $\boldsymbol{\chi}(q,t)$. Suppose
$$E = \int \frac{k}{2}\left(\left|\frac{\partial \boldsymbol{\chi}}{\partial q}\right| - 1\right)^2 \mathrm{d}q,$$

Energy is minimized when tangent vector is magnitude 1

Immersed Boundary Methods

- Forces are given by derivative of energy and q parameterizes $\boldsymbol{\chi}(q,t)$,

$$\delta E = \int -\mathbf{F}\delta\boldsymbol{\chi}\mathrm{d}q, \quad E = \int \frac{k}{2}\left(\left|\frac{\partial\boldsymbol{\chi}}{\partial q}\right| - 1\right)^2 \mathrm{d}q$$

- Then

$$\begin{aligned}
\delta E &= \int \delta\left[\left(\left|\frac{\partial\boldsymbol{\chi}}{\partial q}\right| - 1\right)^2\right]\mathrm{d}q \\
&= \int k\left(\left|\frac{\partial\boldsymbol{\chi}}{\partial q}\right| - 1\right)\frac{\partial\boldsymbol{\chi}/\partial q}{|\partial\boldsymbol{\chi}/\partial q|}\frac{\partial}{\partial q}\delta\boldsymbol{\chi}\mathrm{d}q \\
&= \int -\frac{\partial}{\partial q}\left(k\left(\left|\frac{\partial\boldsymbol{\chi}}{\partial q}\right| - 1\right)\frac{\partial\boldsymbol{\chi}/\partial q}{|\partial\boldsymbol{\chi}/\partial q|}\right)\delta\boldsymbol{\chi}\mathrm{d}q
\end{aligned}$$

Immersed Boundary Methods

- Forces are given by derivative of energy and q parameterizes $\chi(q,t)$,

$$\delta E = \int -\mathbf{F}\delta\chi \mathrm{d}q, \quad E = \int \frac{k}{2}\left(\left|\frac{\partial \chi}{\partial q}\right| - 1\right)^2 \mathrm{d}q$$

- Then

$$\begin{aligned}
\delta E &= \int \delta\left[\left(\left|\frac{\partial \chi}{\partial q}\right| - 1\right)^2\right] \mathrm{d}q \\
&= \int k\left(\left|\frac{\partial \chi}{\partial q}\right| - 1\right)\frac{\partial \chi/\partial q}{|\partial \chi/\partial q|}\frac{\partial}{\partial q}\delta\chi \mathrm{d}q \\
&= \int -\frac{\partial}{\partial q}\left(k\left(\left|\frac{\partial \chi}{\partial q}\right| - 1\right)\frac{\partial \chi/\partial q}{|\partial \chi/\partial q|}\right)\delta\chi \mathrm{d}q
\end{aligned}$$

- This gives us that

$$F = \frac{\partial}{\partial q}\left(k\left(\left|\frac{\partial \chi}{\partial q}\right| - 1\right)\frac{\partial \chi/\partial q}{|\partial \chi/\partial q|}\right)$$

Immersed Boundary Methods

- Consider a *thin massless interface*.
  - Structure moves according to local velocity

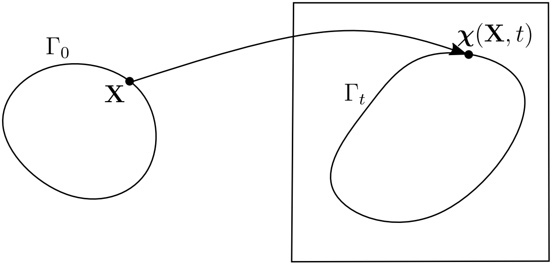
$$\frac{\partial \boldsymbol{\chi}}{\partial t} = \int_{\Omega} \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \boldsymbol{\chi}(\mathbf{X}, t))\mathrm{d}\mathbf{x}$$

  - Lagrangian force density is

$$\mathbf{F}(\mathbf{X}, t) = \frac{\delta E}{\delta \boldsymbol{\chi}}$$

  - Eulerian force density is

$$\mathbf{f}(\mathbf{x}, t) = \int_{\Gamma} \mathbf{F}(\mathbf{X}, t)\delta(\mathbf{x} - \boldsymbol{\chi}(\mathbf{X}, t))\mathrm{d}\mathbf{X}$$



Immersed Boundary Methods

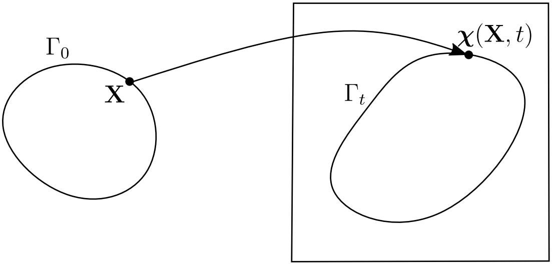
- The structure is *immersed* in a fluid

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \nabla \cdot \sigma + \mathbf{f}$$

$$\nabla \cdot \mathbf{u} = 0$$

$$\frac{\partial \chi}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \chi(\mathbf{X}, t))\mathrm{d}\mathbf{x}$$

$$\mathbf{f}(\mathbf{x}, t) = \int_\Gamma \mathbf{F}(\mathbf{X}, t)\delta(\mathbf{x} - \chi(\mathbf{X}, t))\mathrm{d}\mathbf{X}$$



Immersed Boundary Methods

- IB equations:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \nabla \cdot \sigma + \mathbf{f}$$

$$\nabla \cdot \mathbf{u} = 0$$

$$\frac{\partial \chi}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \chi(\mathbf{X}, t))\mathrm{d}\mathbf{x}$$

$$\mathbf{f}(\mathbf{x}, t) = \int_\Gamma \mathbf{F}(\mathbf{X}, t)\delta(\mathbf{x} - \chi(\mathbf{X}, t))\mathrm{d}\mathbf{X}$$

- Solution strategy:
  - *Explicit structure, implicit fluid*
  - Move structure by interpolating velocity
  - Compute forces from structure deformation
  - Spread forces to Eulerian grid
  - Solve Navier-Stokes equations

Immersed Boundary Methods

- IB equations:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} = \nabla\cdot\sigma + \mathbf{f}$$

$$\nabla\cdot\mathbf{u} = 0$$

$$\frac{\partial \chi}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x}-\chi(\mathbf{X},t))\mathrm{d}\mathbf{x}$$

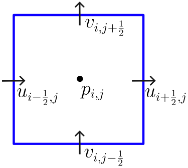
$$\mathbf{f}(\mathbf{x},t) = \int_\Gamma \mathbf{F}(\mathbf{X},t)\delta(\mathbf{x}-\chi(\mathbf{X},t))\mathrm{d}\mathbf{X}$$

- Navier-Stokes solution

- Discretize operators with centered differences on MAC grid.
- Discretize using *forward and backward Euler*

$$\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t} + \mathbf{u}^n\cdot\nabla_h\mathbf{u}^n = -\nabla_h p^{n+1} + \mu\Delta_h\mathbf{u}^{n+1} + \mathbf{f}^n$$

$$\nabla_h\cdot\mathbf{u}^{n+1} = 0$$



- Linear system to solve for $\mathbf{u}^{n+1}$ and p^{n+1}.

Immersed Boundary Methods

- IB equations:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \nabla \cdot \sigma + \mathbf{f}$$
$$\nabla \cdot \mathbf{u} = 0$$
$$\frac{\partial \chi}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \chi(\mathbf{X}, t))\mathrm{d}\mathbf{x}$$
$$\mathbf{f}(\mathbf{x}, t) = \int_\Gamma \mathbf{F}(\mathbf{X}, t)\delta(\mathbf{x} - \chi(\mathbf{X}, t))\mathrm{d}\mathbf{X}$$

- Lagrangian discretization:
  - Structure is point cloud (or FE mesh): about 1 Lagrangian point per Eulerian grid cell.
  - Assume $\mathbf{F} = k\frac{\partial^2 \chi}{\partial \mathbf{X}^2}$: tension corresponding to zero rest length springs
  - Forward Euler discretization

$$\chi^{n+1} = \chi^n + \Delta t \mathbf{u}^n(\chi^n)$$
$$\mathbf{F}_\ell^n = k\frac{\chi_{\ell+1}^n - 2\chi_\ell^n + \chi_{\ell-1}^n}{\Delta X^2}$$

Immersed Boundary Methods

- IB equations:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \nabla \cdot \sigma + \mathbf{f}$$

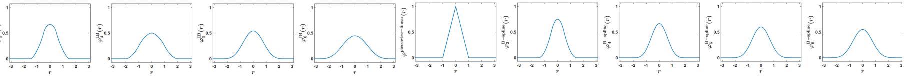
$$\nabla \cdot \mathbf{u} = 0$$

$$\frac{\partial \chi}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \chi(\mathbf{X}, t))\mathrm{d}\mathbf{x}$$

$$\mathbf{f}(\mathbf{x}, t) = \int_\Gamma \mathbf{F}(\mathbf{X}, t)\delta(\mathbf{x} - \chi(\mathbf{X}, t))\mathrm{d}\mathbf{X}$$

- Interaction equations:
  - Replace singular delta function with *regularized delta function*

$$\delta(\mathbf{x}) = \delta(x)\delta(y) \approx \delta_h(\mathbf{x}) = \delta_h(x)\delta_h(y)$$



(a) three-point IB (b) four-point IB (c) five-point IB (d) six-point IB (e) two-point B-spline (f) three-point B-spline (g) four-point B-spline (h) five-point B-spline (i) six-point B-spline

Immersed Boundary Methods

- IB equations:

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = \nabla \cdot \sigma + \mathbf{f}$$

$$\nabla \cdot \mathbf{u} = 0$$

$$\frac{\partial \chi}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \chi(\mathbf{X}, t))\mathrm{d}\mathbf{x}$$

$$\mathbf{f}(\mathbf{x}, t) = \int_\Gamma \mathbf{F}(\mathbf{X}, t)\delta(\mathbf{x} - \chi(\mathbf{X}, t))\mathrm{d}\mathbf{X}$$

- Interaction equations:
  - Replace singular delta function with *regularized delta function*

$$\delta(\mathbf{x}) = \delta(x)\delta(y) \approx \delta_h(\mathbf{x}) = \delta_h(x)\delta_h(y)$$

  - Interpolation:

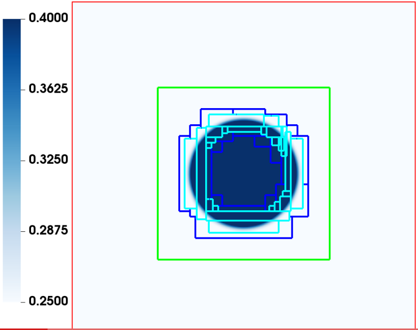
$$\mathbf{U}(\mathbf{X}_\ell) = \sum_{i,j} \mathbf{u}(\mathbf{x}_{i,j})\delta_h(\mathbf{x}_{i,j} - \chi(\mathbf{X}_\ell))\Delta x^2$$

  - Spreading:

$$\mathbf{f}(\mathbf{x}_{i,j}) = \sum_{\ell} \mathbf{F}(\mathbf{X}_\ell)\delta_h(\mathbf{x}_{i,j} - \chi(\mathbf{X}_\ell))\Delta X$$

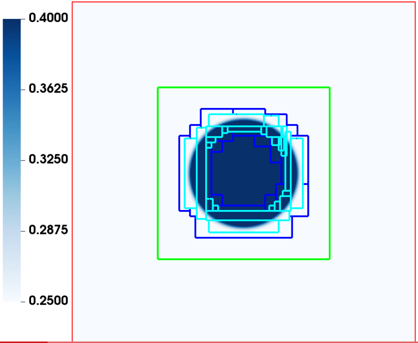
IBAMR

- Immersed Boundary Adaptive Mesh Refinement (IBAMR) is
  - MPI parallelized: scalable to hundreds of cores
  - Adaptive: can tag cells based on IB location, vorticity, etc.
  - Generalizable: Support for different fluid models, multiple structure types, FSI strategies, Lagrangian discretizations
  - Easy to install: `autoibamr` will build on most systems
  - Easy to use: YMMV, but developers are reachable
- `ibamr.github.io`



IBAMR

- IB Method requires:
  - Integrating fluid
  - Integrating structural
  - Coupling operators
- Key principles:
  - Eulerian patch hierarchy
  - Integrator classes
  - IB model, forces, and initial position
  - Input file
- First example: `IB/explicit/ex1`: stretched elastic membrane, v0.16.0
- Elastic springs with zero rest length between each Lagrangian point



IBAMR

- IB Method requires:
  - Integrating fluid
  - Integrating structural
  - Coupling operators
- Key principles:
  - Eulerian patch hierarchy
  - Integrator classes
  - IB model, forces, and initial position
  - Input file
- The `PatchHierarchy<NDIM>` object maintains all Eulerian data.
- The `StandardTagAndInitialize<NDIM>` and `GriddingAlgorithm<NDIM>` objects manages AMR and regridding.

example.cpp

```
        Pointer<CartesianGridGeometry<NDIM> > grid_geometry = new CartesianGridGeometry<NDIM>(
            "CartesianGeometry", app_initializer->getComponentDatabase("CartesianGeometry"));
        Pointer<PatchHierarchy<NDIM> > patch_hierarchy = new PatchHierarchy<NDIM>("PatchHierarchy", grid_geometry);
        Pointer<StandardTagAndInitialize<NDIM> > error_detector =
            new StandardTagAndInitialize<NDIM>("StandardTagAndInitialize",
                                               time_integrator,
                                               app_initializer->getComponentDatabase("StandardTagAndInitialize"));
        Pointer<BergerRigoutsos<NDIM> > box_generator = new BergerRigoutsos<NDIM>();
        Pointer<LoadBalancer<NDIM> > load_balancer =
            new LoadBalancer<NDIM>("LoadBalancer", app_initializer->getComponentDatabase("LoadBalancer"));
        Pointer<GriddingAlgorithm<NDIM> > gridding_algorithm =
            new GriddingAlgorithm<NDIM>("GriddingAlgorithm",
                                        app_initializer->getComponentDatabase("GriddingAlgorithm"),
                                        error_detector,
                                        box_generator,
                                        load_balancer);
```

IBAMR

- IB Method requires:
  - Integrating fluid
  - Integrating structural
  - Coupling operators
- Key principles:
  - Eulerian patch hierarchy
  - Integrator classes
  - IB model, forces, and initial position
  - Input file
- `HierarchyIntegrator` objects can integrate specific equations
- `INSStaggeredHierarchyIntegrator` and `IBExplicitHierarchyIntegrator` are the most important for IB

example.cpp

```cpp
        Pointer<INSHierarchyIntegrator> navier_stokes_integrator;
        const string solver_type =
            app_initializer->getComponentDatabase("Main")->getStringWithDefault("solver_type", "STAGGERED");
        if (solver_type == "STAGGERED")
        {
            navier_stokes_integrator = new INSStaggeredHierarchyIntegrator(
                "INSStaggeredHierarchyIntegrator",
                app_initializer->getComponentDatabase("INSStaggeredHierarchyIntegrator"));
        }
        else if (solver_type == "COLLOCATED")
        {
            navier_stokes_integrator = new INSCollocatedHierarchyIntegrator(
                "INSCollocatedHierarchyIntegrator",
                app_initializer->getComponentDatabase("INSCollocatedHierarchyIntegrator"));
        }
        else
        {
            TBOX_ERROR("Unsupported solver type: " << solver_type << "\n"
                                                   << "Valid options are: COLLOCATED, STAGGERED");
        }
        Pointer<IBMethod> ib_method_ops = new IBMethod("IBMethod", app_initializer->getComponentDatabase("IBMethod"));
        Pointer<IBHierarchyIntegrator> time_integrator =
            new IBExplicitHierarchyIntegrator("IBHierarchyIntegrator",
                                              app_initializer->getComponentDatabase("IBHierarchyIntegrator"),
                                              ib_method_ops,
                                              navier_stokes_integrator);
```

IBAMR

- IB Method requires:
  - Integrating fluid
  - Integrating structural
  - Coupling operators
- Key principles:
  - Eulerian patch hierarchy
  - Integrator classes
  - IB model, forces, and initial position
  - Input file
- Each IB model has a different class, e.g. `IBMethod`, `IBFEMethod`, etc.
- Initialize and setup force generator

example.cpp

```
        Pointer<IBMethod> ib_method_ops = new IBMethod("IBMethod", app_initializer->getComponentDatabase("IBMethod"));
        Pointer<IBHierarchyIntegrator> time_integrator =
            new IBExplicitHierarchyIntegrator("IBHierarchyIntegrator",
                                              app_initializer->getComponentDatabase("IBHierarchyIntegrator"),
                                              ib_method_ops,
                                              navier_stokes_integrator);
```

```
        // Configure the IB solver.
        Pointer<IBStandardInitializer> ib_initializer = new IBStandardInitializer(
            "IBStandardInitializer", app_initializer->getComponentDatabase("IBStandardInitializer"));
        ib_method_ops->registerLInitStrategy(ib_initializer);
        Pointer<IBStandardForceGen> ib_force_fcn = new IBStandardForceGen();
        ib_method_ops->registerIBLagrangianForceFunction(ib_force_fcn);
```

IBAMR

- IB Method requires:
  - Integrating fluid
  - Integrating structural
  - Coupling operators
- Key principles:
  - Eulerian patch hierarchy
  - Integrator classes
  - IB model, forces, and initial position
  - Input file
- `IBStandardForceGen` has support for many standard elastic structures

$$\text{Spring (.spring): } \mathbf{F}_i = \kappa \frac{|\mathbf{X}_i - \mathbf{X}_j| - R_0}{|\mathbf{X}_i - \mathbf{X}_j|}(\mathbf{X}_i - \mathbf{X}_j)$$

$$\mathbf{F}_j = -\kappa \frac{|\mathbf{X}_i - \mathbf{X}_j| - R_0}{|\mathbf{X}_i - \mathbf{X}_j|}(\mathbf{X}_i - \mathbf{X}_j)$$

$$\text{Beams (.beam): } \mathbf{F} = K(\mathbf{X}_j + \mathbf{X}_k - 2\mathbf{X}_i) - \gamma$$

$$\mathbf{F}_i = 2\mathbf{F}, \quad \mathbf{F}_j = -\mathbf{F}, \quad \mathbf{F}_k = -\mathbf{F}$$

$$\text{Target (.target): } \mathbf{F}_i = \kappa(\mathbf{X}_0 - \mathbf{X}_i) - E\mathbf{U}_i$$

- Note: Scaling for ΔX is not considered here. Discretization for spreading depends on dimension of structure (1/2/3D?)
- Final factor for ΔX should be provided in coefficients.

IBAMR

- IB Method requires:
  - Integrating fluid
  - Integrating structural
  - Coupling operators
- Key principles:
  - Eulerian patch hierarchy
  - Integrator classes
  - IB model, forces, and initial position
  - Input file
- Input file is parsed and provides runtime setup
- Set physical parameters (viscosity, density)

input2d

```
// constants
PI = 3.14159265358979

// physical parameters
L   = 1.0
MU  = 1.0e-2
RHO = 1.0
K   = 1.0

// grid spacing parameters
MAX_LEVELS = 1                                 // maximum number of levels in locally refined grid
REF_RATIO  = 4                                 // refinement ratio between levels
N = 64                                         // actual    number of grid cells on coarsest grid level
NFINEST = (REF_RATIO^(MAX_LEVELS - 1))*N       // effective number of grid cells on finest   grid level
DX_FINEST = L/NFINEST

// solver parameters
DELTA_FUNCTION      = "IB_4"
SOLVER_TYPE         = "STAGGERED"              // the fluid solver to use (STAGGERED or COLLOCATED)
START_TIME          = 0.0e0                    // initial simulation time
END_TIME            = (3.0^log10(K))*0.55/K    // final simulation time
GROW_DT             = 2.0e0                    // growth factor for timesteps
NUM_CYCLES          = 1                        // number of cycles of fixed-point iteration
CONVECTIVE_TS_TYPE  = "ADAMS_BASHFORTH"        // convective time stepping type
CONVECTIVE_OP_TYPE  = "PPM"                    // convective differencing discretization type
CONVECTIVE_FORM     = "ADVECTIVE"              // how to compute the convective terms
NORMALIZE_PRESSURE  = TRUE                     // whether to explicitly force the pressure to have mean zero
CFL_MAX             = 0.3                      // maximum CFL number
DT                  = (1.0/K)*1.6e-2*DX_FINEST // maximum timestep size
ERROR_ON_DT_CHANGE  = TRUE                     // whether to emit an error message if the time step size changes
```

IBAMR

- IB Method requires:
  - Integrating fluid
  - Integrating structural
  - Coupling operators
- Key principles:
  - Eulerian patch hierarchy
  - Integrator classes
  - IB model, forces, and initial position
  - Input file
- Each IB model has a different class, e.g. `IBMethod`, `IBFEMethod`, etc.
- Input file specifies structure name and level placement.

input2d

```
IBStandardInitializer {
   max_levels      = MAX_LEVELS
   structure_names = "curve2d_64"

   beta  = 0.35
   alpha = 0.25^2/beta

   A = PI*alpha*beta  // area of ellipse
   R = sqrt(A/PI)     // radius of disc with equivalent area as the ellipse
   perim = 2*PI*R     // perimeter of the equivalent disc

   dx = L/NFINEST
   dx_64 = L/64
   num_node_circum = (dx_64/dx)*ceil(perim/(dx_64/3)/4)*4
   ds = 2.0*PI*R/num_node_circum

   curve2d_64 {
      level_number = MAX_LEVELS - 1
      uniform_spring_stiffness = K/ds
   }
}
```

IBAMR

THE UNIVERSITY OF UTAH®

- IB Method requires:
  - Integrating fluid
  - Integrating structural
  - Coupling operators
- Key principles:
  - Eulerian patch hierarchy
  - Integrator classes
  - IB model, forces, and initial position
  - Input file
- Each IB model has a different class, e.g. `IBMethod`, `IBFEMethod`, etc.
- IB structures can be setup as text files. Details on format can be found `ibamr.github.io/docs`: search for `IBStandardInitializer`

curve2d_64.vertex

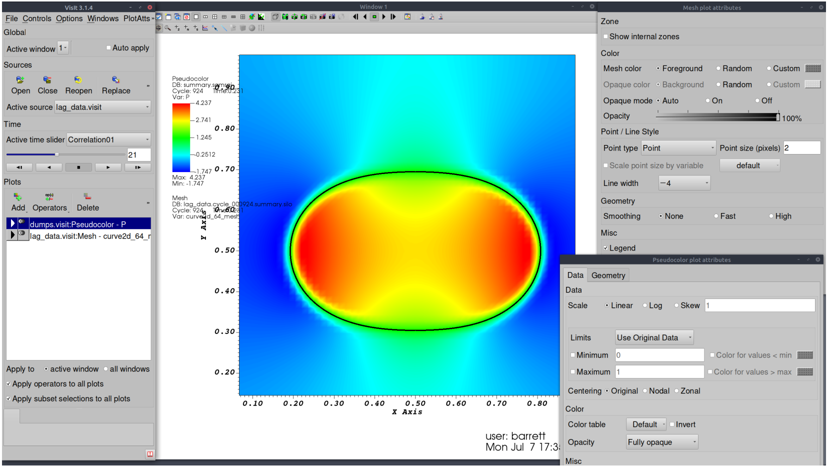
```
304
6.7857142857142860e-01 5.0000000000000000e-01
6.7853328871213048e-01 5.0723341542964395e-01
6.7841888542630380e-01 5.1446374098708458e-01
6.7822826758319077e-01 5.2168788812000810e-01
6.7796151660833393e-01 5.2890277091531634e-01
6.7761874644879327e-01 5.3610530741732487e-01
6.7720010352447235e-01 5.4329242094427166e-01
6.7670576666557192e-01 5.5046104140257135e-01
6.7613594703620039e-01 5.5760810659825688e-01
6.7549088804417134e-01 5.6473056354504547e-01
6.7477086523702756e-01 5.7182536976847198e-01
6.7397618618433708e-01 5.7888949460553218e-01
6.7310719034630906e-01 5.8591992049927966e-01
6.7216424892878812e-01 5.9291364428782534e-01
6.7114776472468796e-01 5.9986767848718681e-01
6.7005817194193118e-01 6.0677905256744202e-01
6.6889593601797048e-01 6.1364481422163919e-01
6.6766155342096956e-01 6.2046203062692351e-01
6.6635555143772840e-01 6.2722778969734039e-01
```

curve2d_64.spring

```
304
     0       1 1.9353241079974475e+02 0.0000000000000000e+00
     1       2 1.9353241079974475e+02 0.0000000000000000e+00
     2       3 1.9353241079974475e+02 0.0000000000000000e+00
     3       4 1.9353241079974475e+02 0.0000000000000000e+00
     4       5 1.9353241079974475e+02 0.0000000000000000e+00
     5       6 1.9353241079974475e+02 0.0000000000000000e+00
     6       7 1.9353241079974475e+02 0.0000000000000000e+00
     7       8 1.9353241079974475e+02 0.0000000000000000e+00
     8       9 1.9353241079974475e+02 0.0000000000000000e+00
     9      10 1.9353241079974475e+02 0.0000000000000000e+00
    10      11 1.9353241079974475e+02 0.0000000000000000e+00
    11      12 1.9353241079974475e+02 0.0000000000000000e+00
    12      13 1.9353241079974475e+02 0.0000000000000000e+00
    13      14 1.9353241079974475e+02 0.0000000000000000e+00
    14      15 1.9353241079974475e+02 0.0000000000000000e+00
    15      16 1.9353241079974475e+02 0.0000000000000000e+00
    16      17 1.9353241079974475e+02 0.0000000000000000e+00
    17      18 1.9353241079974475e+02 0.0000000000000000e+00
    18      19 1.9353241079974475e+02 0.0000000000000000e+00
```

IBAMR

- Use VisIt or Paraview to visualize data
- `dumps.visit` contains Eulerian data, `lag_data.visit` contains Lagrangian data



IBAMR

- To use AMR, increase `MAX_LEVELS`.
- `GRADIENT_DETECTOR` will refine based on IB location and vorticity values. Add `REFINE_BOXES` to specify portion of domain, see `navier_stokes/ex0`. Change `MAX_LEVELS=2`.
- Note: Make sure structure is refined as well: set structure to `curve2d_256`

```
// grid spacing parameters
MAX_LEVELS = 1                                 // maximum number of levels in locally refined grid
REF_RATIO  = 4                                 // refinement ratio between levels
N = 64                                         // actual    number of grid cells on coarsest grid level
NFINEST = (REF_RATIO^(MAX_LEVELS - 1))*N       // effective number of grid cells on finest   grid level
DX_FINEST = L/NFINEST
```

```
GriddingAlgorithm {
   max_levels = MAX_LEVELS
   ratio_to_coarser {
      level_1 = REF_RATIO,REF_RATIO
      level_2 = REF_RATIO,REF_RATIO
      level_3 = REF_RATIO,REF_RATIO
      level_4 = REF_RATIO,REF_RATIO
      level_5 = REF_RATIO,REF_RATIO
   }
   largest_patch_size {
      level_0 = 512,512  // all finer levels will use same values as level_0
   }
   smallest_patch_size {
      level_0 =   8,  8  // all finer levels will use same values as level_0
   }
   efficiency_tolerance = 0.85e0  // min % of tag cells in new patch level
   combine_efficiency   = 0.85e0  // chop box if sum of volumes of smaller boxes < efficiency * vol of large box
}

StandardTagAndInitialize {
   tagging_method = "GRADIENT_DETECTOR"
}
```

Choice of IB Kernel

- Example:
  `IB/explicit/ex0`:

$$
\begin{aligned}
\mathbf{u}(\mathbf{x},0) &= 0,\\
\boldsymbol{\chi}(\theta,0) &= (X_0,Y_0) + R(\cos\theta,\sin\theta),\\
\mathbf{F}(\theta,0) &= \kappa\frac{\partial^2\boldsymbol{\chi}}{\partial\theta^2}\\
p(\mathbf{x},0) &= \kappa H(\|\mathbf{x}-(X_0,Y_0)\|) + C
\end{aligned}
$$

- Same as `ex1`, but exact initial conditions.
- Note: Despite exact initial conditions, spurious flows around structure

Choice of IB Kernel

- Example:
  `IB/explicit/ex0`:

$$
\begin{aligned}
\mathbf{u}(\mathbf{x},0) &= 0,\\
\chi(\theta,0) &= (X_0,Y_0)+R(\cos\theta,\sin\theta),\\
\mathbf{F}(\theta,0) &= \kappa\frac{\partial^2\chi}{\partial\theta^2}\\
p(\mathbf{x},0) &= \kappa H(\|\mathbf{x}-(X_0,Y_0)\|)+C
\end{aligned}
$$

- Same as `ex1`, but exact initial conditions.
- Note: Despite exact initial conditions, spurious flows around structure

- One problem: Discrete interpolated velocities are not divergence free.

$$
\nabla_h\cdot\mathbf{u}=0 \not\Rightarrow
$$

$$
\nabla_h\cdot\sum_{i,j}\mathbf{u}_{i,j}\delta_h(\mathbf{x}_{i,j}-\chi)h^2=0
$$

Choice of IB Kernel

- Example:
  `IB/explicit/ex0`:

$$\mathbf{u}(\mathbf{x},0) = 0,$$
$$\chi(\theta,0) = (X_0,Y_0) + R(\cos\theta,\sin\theta),$$
$$\mathbf{F}(\theta,0) = \kappa\frac{\partial^2\chi}{\partial\theta^2}$$
$$p(\mathbf{x},0) = \kappa H(\|\mathbf{x}-(X_0,Y_0)\|) + C$$

- Same as `ex1`, but exact initial conditions.
- Note: Despite exact initial conditions, spurious flows around structure

- One problem: Discrete interpolated velocities are not divergence free.

$$\nabla_h\cdot\mathbf{u} = 0 \;\not\Rightarrow$$
$$\nabla_h\cdot\sum_{i,j}\mathbf{u}_{i,j}\delta_h(\mathbf{x}_{i,j}-\chi)h^2 = 0$$

- Another problem: **f** corresponds to a gradient (pressure)

$$\mathbf{f} = \int\mathbf{F}(\theta)\delta(\mathbf{x}-\chi)\mathrm{d}\theta = \nabla\phi$$

but

$$\int\mathbf{F}(\theta)\delta_h(\mathbf{x}-\chi)\mathrm{d}\theta \neq \nabla_h\phi$$

Choice of IB Kernel

- Example:
  `IB/explicit/ex0`:

$$
\begin{aligned}
\mathbf{u}(\mathbf{x},0) &= 0,\\
\chi(\theta,0) &= (X_0,Y_0) + R(\cos\theta,\sin\theta),\\
\mathbf{F}(\theta,0) &= \kappa\frac{\partial^2\chi}{\partial\theta^2}\\
p(\mathbf{x},0) &= \kappa H(\|\mathbf{x}-(X_0,Y_0)\|) + C
\end{aligned}
$$

- Same as `ex1`, but exact initial conditions.
- Note: Despite exact initial conditions, spurious flows around structure

- One problem: Discrete interpolated velocities are not divergence free.

$$
\nabla_h\cdot\mathbf{u} = 0 \;\not\Rightarrow
$$
$$
\nabla_h\cdot\sum_{i,j}\mathbf{u}_{i,j}\delta_h(\mathbf{x}_{i,j}-\chi)h^2 = 0
$$

- Another problem: $\mathbf{f}$ corresponds to a gradient (pressure)

$$
\mathbf{f} = \int \mathbf{F}(\theta)\delta(\mathbf{x}-\chi)\mathrm{d}\theta = \nabla\phi
$$

but

$$
\int \mathbf{F}(\theta)\delta_h(\mathbf{x}-\chi)\mathrm{d}\theta \neq \nabla_h\phi
$$

- Choice of delta function can dramatically affect results

Choice of IB Kernel

- Example:
  `IB/explicit/ex0`:

$$\mathbf{u}(\mathbf{x},0) = 0,$$
$$\chi(\theta,0) = (X_0,Y_0) + R(\cos\theta,\sin\theta),$$
$$\mathbf{F}(\theta,0) = \kappa\frac{\partial^2\chi}{\partial\theta^2}$$
$$p(\mathbf{x},0) = \kappa H(\|\mathbf{x}-(X_0,Y_0)\|) + C$$

- Same as `ex1`, but exact initial conditions.
- Note: Despite exact initial conditions, spurious flows around structure

- One problem: Discrete interpolated velocities are not divergence free.

$$\nabla_h\cdot\mathbf{u} = 0 \not\Rightarrow$$
$$\nabla_h\cdot\sum_{i,j}\mathbf{u}_{i,j}\delta_h(\mathbf{x}_{i,j}-\chi)h^2 = 0$$

- Another problem: $\mathbf{f}$ corresponds to a gradient (pressure)

$$\mathbf{f} = \int \mathbf{F}(\theta)\delta(\mathbf{x}-\chi)\mathrm{d}\theta = \nabla\phi$$

but

$$\int \mathbf{F}(\theta)\delta_h(\mathbf{x}-\chi)\mathrm{d}\theta \neq \nabla_h\phi$$

- Change `IB_4` to `COMPOSITE_BSPLINE_54`

Slender Body Models

- Track the centerline of the rod, denoted by χ.
- Introduce *director vectors* that account for twist of body.
- Use rod theories to describe both force and torque on thin bodies.

$$\rho \frac{D\mathbf{u}}{Dt} = \nabla \cdot \sigma + \mathbf{f}(\mathbf{x}, t) + \frac{1}{2}\nabla \times \mathbf{n}(\mathbf{x}, t)$$

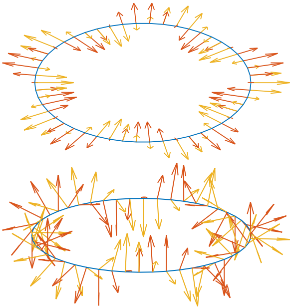
$$\nabla \cdot \mathbf{u}(\mathbf{x}, t) = 0$$

$$\mathbf{f}(\mathbf{x}, t) = \int_\Omega \mathbf{F}(s, t)\phi(\mathbf{x} - \boldsymbol{\chi}(s, t))\mathrm{d}s$$

$$\mathbf{n}(\mathbf{x}, t) = \int_\Omega \mathbf{N}(s, t)\phi(\mathbf{x} - \boldsymbol{\chi}(s, t))\mathrm{d}s$$

$$\frac{\partial \chi}{\partial t}(s, t) = \int_U \mathbf{u}(\mathbf{x}, t)\phi(\mathbf{x} - \boldsymbol{\chi}(s, t))\mathrm{d}\mathbf{x}$$

$$\frac{\partial \mathbf{D}^i}{\partial t}(s, t) = \left(\frac{1}{2}\int_U \nabla \times \mathbf{u}(\mathbf{x}, t)\phi(\mathbf{x} - \boldsymbol{\chi}(s, t))\mathrm{d}\mathbf{x}\right) \times \mathbf{D}^i(s, t)$$



Slender Body Models

- Track the centerline of the rod, denoted by χ.
- Introduce *director vectors* that account for twist of body.
- Use rod theories to describe both force and torque on thin bodies.

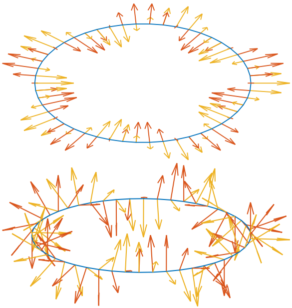
$\mathbf{F}(s,t)$ and $\mathbf{N}(s,t)$ come from a energy formulation:

$$E_{\text{twist}} = \frac{1}{2}\int_0^L a_3\left(\Omega_3 - \tau_1\right)^2 \mathrm{d}s$$

$$E_{\text{bend}} = \frac{1}{2}\int_0^L \left(a_1\left(\Omega_1 - \kappa_1\right)^2 + a_2\left(\Omega_2 - \kappa_2\right)^2\right)\mathrm{d}s$$

$$E_{\text{shear}} = \frac{1}{2}\int_0^L \left(b_1\left(\frac{\partial \chi}{\partial s}\cdot \mathbf{D}^1\right)^2 + b_2\left(\frac{\partial \chi}{\partial s}\cdot \mathbf{D}^2\right)^2\right)\mathrm{d}s$$

$$E_{\text{stretch}} = \frac{1}{2}\int_0^L b_3\left(\frac{\partial \chi}{\partial s}\cdot \mathbf{D}^3 - 1\right)^2 \mathrm{d}s$$



with $\Omega_1 = \frac{\partial \mathbf{D}^2}{\partial s}\cdot \mathbf{D}^3$, $\Omega_2 = \frac{\partial \mathbf{D}^3}{\partial s}\cdot \mathbf{D}^1$, $\Omega_3 = \frac{\partial \mathbf{D}^1}{\partial s}\cdot \mathbf{D}^2$.

Slender Body Models

- Example: `IB/explicit/ex4`: twisted elastic ring
- Register a `GeneralizedIBMethod` object with the integrator.
- Physical parameters specified in input files.

```
        Pointer<GeneralizedIBMethod> ib_method_ops = new GeneralizedIBMethod(
            "GeneralizedIBMethod", app_initializer->getComponentDatabase("GeneralizedIBMethod"));
        Pointer<IBHierarchyIntegrator> time_integrator =
            new IBExplicitHierarchyIntegrator("IBHierarchyIntegrator",
                                              app_initializer->getComponentDatabase("IBHierarchyIntegrator"),
                                              ib_method_ops,
                                              navier_stokes_integrator);
```

```
        // Configure the IB solver.
        Pointer<IBStandardInitializer> ib_initializer = new IBStandardInitializer(
            "IBStandardInitializer", app_initializer->getComponentDatabase("IBStandardInitializer"));
        ib_method_ops->registerLInitStrategy(ib_initializer);
        Pointer<IBKirchhoffRodForceGen> ib_force_and_torque_fcn = new IBKirchhoffRodForceGen();
        ib_method_ops->registerIBKirchhoffRodForceGen(ib_force_and_torque_fcn);
```

Slender Body Models

- Example: `IB/explicit/ex4`: twisted elastic ring
- Register a `GeneralizedIBMethod` object with the integrator.
- Physical parameters specified in input files.

```
for r = 0:nr-1
  theta = 2*pi*r/nr;

  R = [cos(theta) , sin(theta) , 0];
  Theta = [-sin(theta), cos(theta) , 0];
  Z = [0 , 0 , 1];

  X = r1*R;
  vertices(r+1,:) = X;

  fprintf(vertex_fid, '%1.16e %1.16e %1.16e\n', X(1), X(2), X(3));

  r_curr = r;
  r_next = mod(r+1,nr);

  fprintf(rod_fid, ['%6d %6d %1.16e %1.16e %1.16e %1.16e %1.16e %1.16e ' ...
                     '%1.16e %1.16e %1.16e %1.16e\n'], r_curr, r_next, ds, ...
          a1, a2, a3, b1, b2, b3, kappa1, kappa2, tau);

  D3 =  cos(beta)*Theta + sin(beta)*Z;
  E  = -sin(beta)*Theta + cos(beta)*Z;
  D1 =  cos(p*theta + epsilon*sin(theta))*E + sin(p*theta + epsilon*sin(theta))*R;
  D2 = -sin(p*theta + epsilon*sin(theta))*E + cos(p*theta + epsilon*sin(theta))*R;

  D1_vals(r+1,:) = D1;
  D2_vals(r+1,:) = D2;
  D3_vals(r+1,:) = D3;

  fprintf(director_fid, '%1.16e %1.16e %1.16e\n', D1(1), D1(2), D1(3));
  fprintf(director_fid, '%1.16e %1.16e %1.16e\n', D2(1), D2(2), D2(3));
  fprintf(director_fid, '%1.16e %1.16e %1.16e\n', D3(1), D3(2), D3(3));
end %for
```

Viscoelastic Fluid Models

- Viscoelastic fluids exhibit both *elastic* and *viscous* responses to forces.

Viscoelastic Fluid Models

- Viscoelastic fluids exhibit both *elastic* and *viscous* responses to forces.

Springs:

$$\sigma = G\gamma$$

Dashpot:

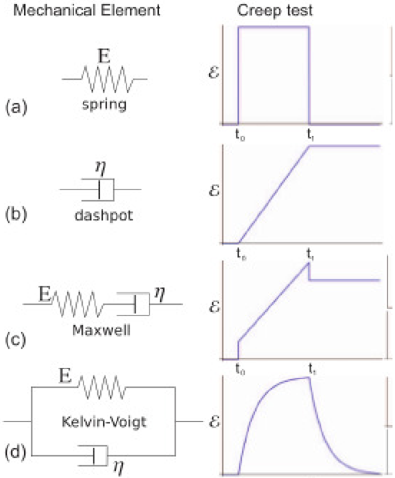
$$\sigma = \mu\dot{\gamma}$$

Maxwell Element:

$$\dot{\sigma} = \frac{\mu}{\lambda}\dot{\gamma} - \frac{1}{\lambda}\sigma$$
$$\lambda = \frac{\mu}{G}$$

Kelvin-Voigt:

$$\sigma = G\gamma + \mu\dot{\gamma}$$



Viscoelastic Fluid Models

- Maxwell model:

$$\lambda \overset{\nabla}{\boldsymbol{\tau}}_{\mathsf{p}} = \eta \dot{\boldsymbol{\gamma}} - \boldsymbol{\tau}_{\mathsf{p}}$$

- Oldroyd type models: let $\boldsymbol{\sigma} = -p\mathbb{I} + \mu_s \dot{\boldsymbol{\gamma}} + \boldsymbol{\tau}_p$
  - Oldroyd-B model:

$$\boldsymbol{\tau}_p + \lambda \overset{\nabla}{\boldsymbol{\tau}}_p = \mu \dot{\boldsymbol{\gamma}}$$

  - Upper-convected derivative:

$$\overset{\nabla}{\boldsymbol{\tau}} = \lambda \left(\frac{\partial \boldsymbol{\tau}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} - \nabla \mathbf{u} \cdot \boldsymbol{\tau} - \boldsymbol{\tau} \cdot \nabla \mathbf{u}^{\mathsf{T}} \right)$$

Viscoelastic Fluid Models

- Maxwell model:

$$\lambda \overset{\nabla}{\boldsymbol{\tau}}_{\mathrm{p}} = \eta \dot{\boldsymbol{\gamma}} - \boldsymbol{\tau}_{\mathrm{p}}$$

- Oldroyd type models: let $\boldsymbol{\sigma} = -p\mathbb{I} + \mu_s \dot{\boldsymbol{\gamma}} + \boldsymbol{\tau}_p$
  - Oldroyd-B model:

$$\boldsymbol{\tau}_p + \lambda \overset{\nabla}{\boldsymbol{\tau}}_p = \mu \dot{\boldsymbol{\gamma}}$$

  - Upper-convected derivative:

$$\overset{\nabla}{\boldsymbol{\tau}} = \lambda \left(\frac{\partial \boldsymbol{\tau}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} - \nabla \mathbf{u} \cdot \boldsymbol{\tau} - \boldsymbol{\tau} \cdot \nabla \mathbf{u}^{\mathsf{T}} \right)$$

- Giesekus model:

$$\boldsymbol{\tau}_p + \lambda \overset{\nabla}{\boldsymbol{\tau}}_p + \alpha \frac{\lambda}{\mu_p} \boldsymbol{\tau}_p \cdot \boldsymbol{\tau}_p = \mu \dot{\boldsymbol{\gamma}}$$

- Here, $\boldsymbol{\tau}_p \cdot \boldsymbol{\tau}_p$ is an *anisotropic drag term*

Models of Viscoelastic Fluids

- Complete Model:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = \nabla\cdot\boldsymbol{\sigma} + \mathbf{f}$$
$$\nabla\cdot\mathbf{u} = 0$$
$$\boldsymbol{\sigma} = -p\mathbb{I} + \mu_s\dot{\boldsymbol{\gamma}} + \boldsymbol{\tau}$$
$$\lambda\overset{\nabla}{\boldsymbol{\tau}} = \mu_{\mathsf{p}}\dot{\boldsymbol{\gamma}} - \boldsymbol{\tau}$$

- Relevant dimensionless numbers:

$$\mathsf{Re} = \frac{\rho U L}{\mu}, \quad \mathsf{Wi} = \dot{\gamma}\lambda, \quad \mathsf{De} = \frac{\lambda}{t_p}$$

- Example: `examples/complex_fluids/ex2`, flow past a confined cylinder.
- Note: set `OUTPUT_STRAIN = FALSE`.

Complex Fluids in IB Methods

- Example: `examples/complex_fluids/ex2`
- Advect a stress tensor, apply an extra force to momentum

```
        Pointer<INSHierarchyIntegrator> navier_stokes_integrator;
        const string solver_type = app_initializer->getComponentDatabase("Main")->getString("solver_type");
        if (solver_type == "STAGGERED")
        {
            navier_stokes_integrator = new INSStaggeredHierarchyIntegrator(
                "INSStaggeredHierarchyIntegrator",
                app_initializer->getComponentDatabase("INSStaggeredHierarchyIntegrator"));
        }
        else if (solver_type == "COLLOCATED")
        {
            navier_stokes_integrator = new INSCollocatedHierarchyIntegrator(
                "INSCollocatedHierarchyIntegrator",
                app_initializer->getComponentDatabase("INSCollocatedHierarchyIntegrator"));
        }
        else
        {
            TBOX_ERROR("Unsupported solver type: " << solver_type << "\n"
                                                   << "Valid options are: COLLOCATED, STAGGERED");
        }

        // Create the advection diffusion integrator for the extra stress.
        Pointer<AdvDiffSemiImplicitHierarchyIntegrator> adv_diff_integrator;
        adv_diff_integrator = new AdvDiffSemiImplicitHierarchyIntegrator(
            "AdvDiffSemiImplicitHierarchyIntegrator",
            app_initializer->getComponentDatabase("AdvDiffSemiImplicitHierarchyIntegrator"));
        navier_stokes_integrator->registerAdvDiffHierarchyIntegrator(adv_diff_integrator);
```

```
        // Generate the extra stress component and set up necessary components. The constructor registers the extra
        // stress with the advection diffusion integrator.
        Pointer<CFINSForcing> polymericStressForcing;
        double viscosity = 0.0, relaxation_time = 0.0;
        if (input_db->keyExists("ComplexFluid"))
        {
            polymericStressForcing = new CFINSForcing("PolymericStressForcing",
                                                     app_initializer->getComponentDatabase("ComplexFluid"),
                                                     navier_stokes_integrator,
                                                     grid_geometry,
                                                     adv_diff_integrator,
                                                     visit_data_writer);
            time_integrator->registerBodyForceFunction(polymericStressForcing);
            viscosity = input_db->getDatabase("ComplexFluid")->getDouble("viscosity");
            relaxation_time = input_db->getDatabase("ComplexFluid")->getDouble("relaxation_time");
        }
```

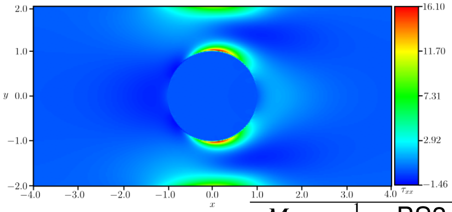
Complex Fluids in IB Methods

- Example: `examples/complex_fluids/ex2`
- *Impose* no motion by applying force $\mathbf{F} = \kappa(\boldsymbol{\chi}^0 - \boldsymbol{\chi}) - \eta \frac{\partial \boldsymbol{\chi}}{\partial t}$
- As $\kappa \to \infty$, we must have that $\boldsymbol{\chi} \to \boldsymbol{\chi}^0$.

```
void
tether_force_function(VectorValue<double>& F,
                      const libMesh::VectorValue<double>& /*n*/,
                      const libMesh::VectorValue<double>& /*N*/,
                      const TensorValue<double>& /*FF*/,
                      const libMesh::Point& x,
                      const libMesh::Point& X,
                      Elem* const /*elem*/,
                      unsigned short int /*side*/,
                      const vector<const vector<double>*>& var_data,
                      const vector<const vector<VectorValue<double> >*>& /*grad_var_data*/,
                      double /*time*/,
                      void* /*ctx*/)
{
    const std::vector<double>& U = *var_data[0];
    for (unsigned int d = 0; d < NDIM; ++d)
    {
        F(d) = kappa_s * (X(d) - x(d)) - eta_s * U[d];
    }
    std::vector<double> d(NDIM);
    d[0] = std::abs(x(0) - X(0));
    d[1] = std::abs(X(1) - x(1));
    // Quit the simulation if the structure has moved more than a quarter of the grid spacing.
    if (ERROR_ON_MOVE && ((d[0] > 0.25 * dx) || (d[1] > 0.25 * dx)))
    {
        TBOX_ERROR("Structure has moved too much.\n");
    }
```

```
        // Configure the IBFE solver.
        ib_method_ops->initializeFEEquationSystems();
        std::vector<int> vars(NDIM);
        for (unsigned int d = 0; d < NDIM; ++d) vars[d] = d;
        vector<SystemData> sys_data(1, SystemData(IBFESurfaceMethod::VELOCITY_SYSTEM_NAME, vars));
        IBFESurfaceMethod::LagSurfaceForceFcnData body_fcn_data(tether_force_function, sys_data);
        ib_method_ops->registerLagSurfaceForceFunction(body_fcn_data);
```

Immersed Boundary Method



	L_1 rate	L_2 rate	L_∞ rate
$\mathbf{u}$	1.693	1.470	1.020
τ_{xx}	1.173	0.537	-0.093
τ_{xy}	1.341	0.632	0.130
τ_{yy}	1.254	0.872	0.017
p	1.275	1.204	-0.541

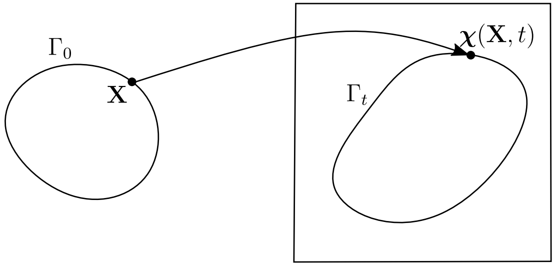
- Drag coefficient

$$C_{\mathrm{d}} = \int_\Gamma \sigma \cdot \mathbf{n} \cdot \mathbf{e}_x \mathrm{d}s$$

- IB stresses fail to converge pointwise!

$M_{\text{fac}} = \frac{1}{2}$	BS3	IB3	PL	IB4	IB6
Coarse	6.15%	9.55%	12.92%	10.50%	11.16%
Medium	2.78%	4.11%	5.39%	4.73%	5.02%
Fine	1.39%	2.04%	2.71%	2.32%	2.38%
$M_{\text{fac}} = 1$					
Coarse	5.45%	6.20%	4.77%	9.51%	11.16%
Medium	2.61%	2.92%	2.54%	4.37%	5.02%
Fine	1.24%	1.42%	1.19%	2.07%	2.38%
$M_{\text{fac}} = 2$					
Coarse	4.57%	5.86%	1.57%	9.06%	11.16%
Medium	2.12%	2.66%	1.03%	4.16%	5.02%
Fine	1.01%	1.28%	0.46%	1.99%	2.38%

Immersed Boundary Method

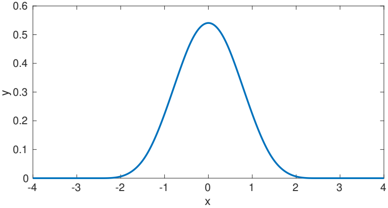


$$\rho \frac{D\mathbf{u}(\mathbf{x},t)}{Dt} = \nabla \cdot \sigma(\mathbf{x},t) + \mathbf{f}(\mathbf{x},t)$$

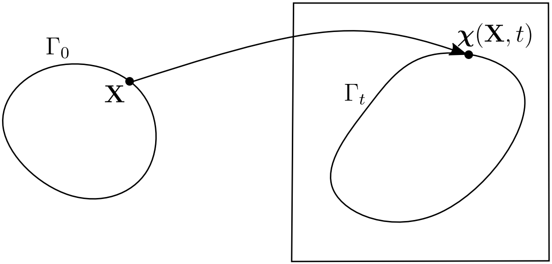
$$\nabla \cdot \mathbf{u}(\mathbf{x},t) = 0$$

$$\mathbf{f}(\mathbf{x},t) = \int_{\Gamma_0} \mathbf{F}(\mathbf{X},t)\delta(\mathbf{x} - \chi(\mathbf{X},t))\mathrm{d}\mathbf{X}$$

$$\frac{\partial \chi(\mathbf{X},t)}{\partial t} = \int_{\Omega} \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x} - \chi(\mathbf{X},t))\mathrm{d}\mathbf{x}$$



Immersed Boundary Method



$$\rho \frac{D\mathbf{u}(\mathbf{x},t)}{Dt} = \nabla \cdot \sigma(\mathbf{x},t) + \mathbf{f}(\mathbf{x},t)$$

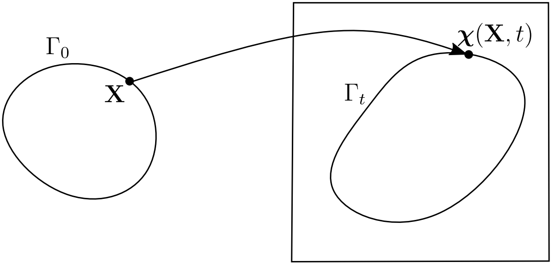
$$\nabla \cdot \mathbf{u}(\mathbf{x},t) = 0$$

$$[\![\sigma]\!] \cdot \mathbf{n} = -\mathbf{F}J^{-1}$$

$$\frac{\partial \chi(\mathbf{X},t)}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x} - \chi(\mathbf{X},t))\mathrm{d}\mathbf{x}$$

- Singular delta function induces jumps in the stress.
- Regular finite differences have $\mathcal{O}(1)$ errors for discontinuous variables
- Immersed Interface: replace δ_h with corrected differences.

Immersed Boundary Method



$$\rho \frac{D\mathbf{u}(\mathbf{x},t)}{Dt} = \nabla \cdot \sigma(\mathbf{x},t) + \mathbf{f}(\mathbf{x},t)$$

$$\nabla \cdot \mathbf{u}(\mathbf{x},t) = 0$$

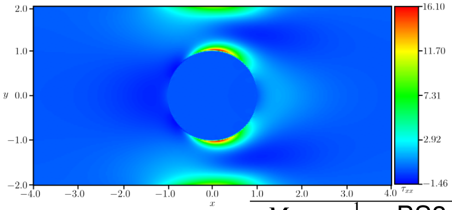
$$[\![\sigma]\!] \cdot \mathbf{n} = -\mathbf{F}J^{-1}$$

$$\frac{\partial \chi(\mathbf{X},t)}{\partial t} = \int_\Omega \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x} - \chi(\mathbf{X},t))\mathrm{d}\mathbf{x}$$

- If $\phi(x)$ has a discontinuity at $\hat{x} \in \left[x_{i+\frac{1}{2}}, x_{i+1}\right]$, then

$$\frac{\partial \phi}{\partial x}\left(x_{i+\frac{1}{2}}\right) = \frac{\phi_{i+1} - \phi_i}{h} - \frac{1}{h}\sum_{m=0}^{2} \frac{(x_{i+1} - \hat{x})^m}{m!} \left[\!\left[\frac{\partial^m \phi}{\partial x^m}\right]\!\right](\hat{x}) + \mathcal{O}\left(h^2\right)$$

Immersed Boundary Method



	L_1 rate	L_2 rate	L_∞ rate
$\mathbf{u}$	1.693	1.470	1.020
τ_{xx}	1.173	0.537	-0.093
τ_{xy}	1.341	0.632	0.130
τ_{yy}	1.254	0.872	0.017
p	1.275	1.204	-0.541

- Drag coefficient

$$C_\mathrm{d} = \int_\Gamma \sigma \cdot \mathbf{n} \cdot \mathbf{e}_x \mathrm{d}s$$

- IB stresses fail to converge pointwise!
- Averaged quantities are accurate.

$M_\text{fac} = \frac{1}{2}$	BS3	IB3	PL	IB4	IB6
Coarse	6.15%	9.55%	12.92%	10.50%	11.16%
Medium	2.78%	4.11%	5.39%	4.73%	5.02%
Fine	1.39%	2.04%	2.71%	2.32%	2.38%
$M_\text{fac} = 1$					
Coarse	5.45%	6.20%	4.77%	9.51%	11.16%
Medium	2.61%	2.92%	2.54%	4.37%	5.02%
Fine	1.24%	1.42%	1.19%	2.07%	2.38%
$M_\text{fac} = 2$					
Coarse	4.57%	5.86%	1.57%	9.06%	11.16%
Medium	2.12%	2.66%	1.03%	4.16%	5.02%
Fine	1.01%	1.28%	0.46%	1.99%	2.38%

Conclusions

- Derived conservation of mass and momentum equations
- Developed the IB equations:
  - Lagrangian structure moves at local fluid velocity
  - Deformations induce elastic forces which are applied to momentum equation
  - Fluid viscosity exists throughout entire domain.
- IBAMR provides a framework to implement different IB strategies strategies