Efficient solutions of implicit discretizations of the immersed boundary method

Jordan Fisher

September 6, 2009

What is the Immersed Boundary Method?

What is the Immersed Boundary Method?

Originally developed by Charles Peskin for his model of an artificial heart valve

What is the Immersed Boundary Method?

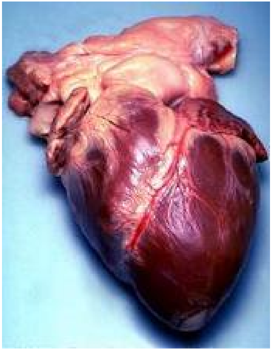
Originally developed by Charles Peskin for his model of an artificial heart valve

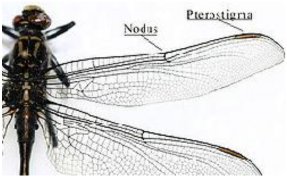
The method proved to be useful for modeling a wide range of phenomenon

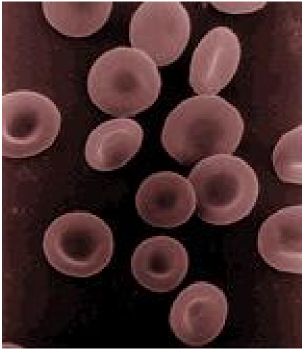
What is the Immersed Boundary Method?

Originally developed by Charles Peskin for his model of an artificial heart valve

The method proved to be useful for modeling a wide range of phenomenon







The immersed boundary method

The immersed boundary method

The standard Navier-Stokes equation for incompressible flow with an arbitrary forcing function $\mathbf{f}$

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + \mathbf{f}, \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{2}$$

The immersed boundary method

The standard Navier-Stokes equation for incompressible flow with an arbitrary forcing function **f**

$$\rho \left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} \right) = -\nabla p + \mu \nabla^2 \mathbf{u} + \mathbf{f}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0 \tag{2}$$

Introduce a surface **X** with no-slip boundary conditions

The immersed boundary method

The standard Navier-Stokes equation for incompressible flow with an arbitrary forcing function **f**

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t}+\mathbf{u}\cdot\nabla\mathbf{u}\right)=-\nabla p+\mu\nabla^2\mathbf{u}+\mathbf{f}, \tag{1}$$

$$\nabla\cdot\mathbf{u}=0 \tag{2}$$

Introduce a surface **X** with no-slip boundary conditions

$$\frac{\partial \mathbf{X}}{\partial t}=\mathbf{u}(\mathbf{X},t)=\int_\Omega \mathbf{u}(\mathbf{x},t)\delta(\mathbf{x}-\mathbf{X}(s,t))d\mathbf{x} \tag{3}$$

The immersed boundary method

The standard Navier-Stokes equation for incompressible flow with an arbitrary forcing function $\mathbf{f}$

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + \mathbf{f}, \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{2}$$

Introduce a surface $\mathbf{X}$ with no-slip boundary conditions

$$\frac{\partial \mathbf{X}}{\partial t} = \mathbf{u}(\mathbf{X}, t) = \int_\Omega \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \mathbf{X}(s, t))d\mathbf{x} \tag{3}$$

Allow the surface to exert force back on the fluid

The immersed boundary method

The standard Navier-Stokes equation for incompressible flow with an arbitrary forcing function $\mathbf{f}$

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + \mathbf{f}, \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{2}$$

Introduce a surface $\mathbf{X}$ with no-slip boundary conditions

$$\frac{\partial \mathbf{X}}{\partial t} = \mathbf{u}(\mathbf{X}, t) = \int_\Omega \mathbf{u}(\mathbf{x}, t)\delta(\mathbf{x} - \mathbf{X}(s,t))d\mathbf{x} \tag{3}$$

Allow the surface to exert force back on the fluid

$$\mathbf{f}(\mathbf{x}, t) = \int_B \mathbf{F}(\mathbf{X}(\cdot,\cdot), s, t)\delta(\mathbf{x} - \mathbf{X}(s,t))ds \tag{4}$$

Standard discretization: Forward Euler/Backward Euler

Standard discretization: Forward Euler/Backward Euler

We approximate δ with a discrete function

$$d_h(r) = \begin{cases} \frac{1}{4h}\left(1+\cos(\frac{\pi r}{2h})\right) & \text{if } |r| \leq 2h \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

Standard discretization: Forward Euler/Backward Euler

We approximate δ with a discrete function

$$d_h(r) = \begin{cases} \frac{1}{4h}\left(1+\cos(\frac{\pi r}{2h})\right) & \text{if } |r| \leq 2h \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

and approximate the spreading and smoothing via

$$(\mathcal{S}_n G)(\mathbf{x}) = \sum_{s \in \mathcal{G}_B} G(s)\delta_h(\mathbf{x} - \mathbf{X}^n(s))h_B, \tag{6}$$

$$(\mathcal{S}_n^* w)(s) = \sum_{\mathbf{x} \in \mathcal{G}_\Omega} w(\mathbf{x})\delta_h(\mathbf{x} - \mathbf{X}^n(s))h^2. \tag{7}$$

$$\tag{8}$$

Standard discretization: Forward Euler/Backward Euler

With these operators, we can write our discretization

Standard discretization: Forward Euler/Backward Euler

With these operators, we can write our discretization

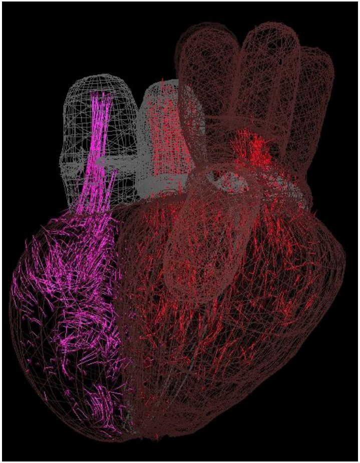
$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^{n}}{\Delta t}+\mathbf{u}^{n}\cdot\mathbf{D}_h\mathbf{u}^{n}\right)=-\mathbf{D}_h p^{n+1}+\mu L_h\mathbf{u}^{n+1}+\mathcal{S}_n\mathcal{A}_{h_B}(\mathbf{X}^{n}), \quad (9)$$

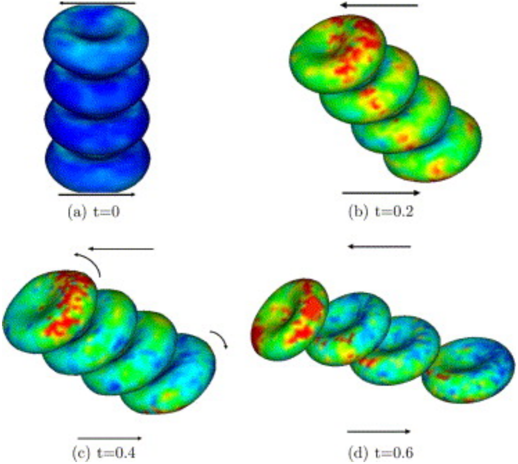
$$\mathbf{D}_h\cdot\mathbf{u}^{n+1}=0, \quad (10)$$

$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^{n}}{\Delta t}=\mathcal{S}_n^{*}\mathbf{u}^{n+1} \quad (11)$$

We can accommodate complex structures via the Lagrangian representation

We can accommodate complex structures via the Lagrangian representation





Looks good

Looks good

Problem:

Looks good

Problem:

Restorative forces in stiff structures operate on timescales much smaller than the timescales of macro scale behavior of the surrounding fluid. AKA, the problem is stiff.

Looks good

Problem:

Restorative forces in stiff structures operate on timescales much smaller than the timescales of macro scale behavior of the surrounding fluid. AKA, the problem is stiff.

Possible solution:

Looks good

Problem:

Restorative forces in stiff structures operate on timescales much smaller than the timescales of macro scale behavior of the surrounding fluid. AKA, the problem is stiff.

Possible solution:

Implicit discretizations!

What should we make implicit?

$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^{n}}{\Delta t}+\mathbf{u}^{n}\cdot\mathbf{D}_h\mathbf{u}^{n}\right)=-\mathbf{D}_h p^{n+1}+\mu L_h\mathbf{u}^{n+1}+\mathcal{S}_n\mathcal{A}_{h_B}(\mathbf{X}^{n}),$$

$$\mathbf{D}_h\cdot\mathbf{u}^{n+1}=0,$$

$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^{n}}{\Delta t}=\mathcal{S}_n^{*}\mathbf{u}^{n+1},$$

1A discretization of Newren, Fogelson, Guy, Kirby

$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^{n}}{\Delta t}+\mathbf{u}^{n}\cdot\mathbf{D}_h\mathbf{u}^{n}\right)=-\mathbf{D}_h p^{n+1}+\mu L_h\mathbf{u}^{n+1}+\mathcal{S}_n\mathcal{A}_{h_B}(\mathbf{X}^{n+1}),$$
$$\mathbf{D}_h\cdot\mathbf{u}^{n+1}=0,$$
$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^{n}}{\Delta t}=\mathcal{S}_n^{*}\mathbf{u}^{n+1},$$

1A discretization of Newren, Fogelson, Guy, Kirby

$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^{n}}{\Delta t}+\mathbf{u}^{n}\cdot\mathbf{D}_h\mathbf{u}^{n}\right)=-\mathbf{D}_h p^{n+1}+\mu L_h\mathbf{u}^{n+1}+\mathcal{S}_n\mathcal{A}_{h_B}(\mathbf{X}^{n+1}),$$

$$\mathbf{D}_h\cdot\mathbf{u}^{n+1}=0,$$

$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^{n}}{\Delta t}=\mathcal{S}_n^{*}\mathbf{u}^{n+1},$$

Originally proposed by Peskin in 1971

1A discretization of Newren, Fogelson, Guy, Kirby

$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^{n}}{\Delta t}+\mathbf{u}^{n}\cdot\mathbf{D}_h\mathbf{u}^{n}\right)=-\mathbf{D}_h p^{n+1}+\mu L_h\mathbf{u}^{n+1}+\mathcal{S}_n\mathcal{A}_{h_B}(\mathbf{X}^{n+1}),$$

$$\mathbf{D}_h\cdot\mathbf{u}^{n+1}=0,$$

$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^{n}}{\Delta t}=\mathcal{S}_n^{*}\mathbf{u}^{n+1},$$

Originally proposed by Peskin in 1971

Proven to be unconditionally stable for non-convective flows with linear, self-adjoint fiber force,

$$(\mathbf{u}^{n+1},\mathbf{u}^{n+1})_\Omega+(\mathcal{A}_{h_B}\mathbf{X}^{n+1},\mathbf{X}^{n+1})_B\leq(\mathbf{u}^{n},\mathbf{u}^{n})_\Omega+(\mathcal{A}_{h_B}\mathbf{X}^{n},\mathbf{X}^{n})_B \quad (12)$$

1A discretization of Newren, Fogelson, Guy, Kirby

$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t}+\mathbf{u}^n\cdot\mathbf{D}_h\mathbf{u}^n\right)=-\mathbf{D}_h p^{n+1}+\mu L_h\mathbf{u}^{n+1}+\mathcal{S}_n\mathcal{A}_{h_B}(\mathbf{X}^{n+1}),$$
$$\mathbf{D}_h\cdot\mathbf{u}^{n+1}=0,$$
$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^n}{\Delta t}=\mathcal{S}_n^*\mathbf{u}^{n+1},$$

Originally proposed by Peskin in 1971

Proven to be unconditionally stable for non-convective flows with linear, self-adjoint fiber force,

$$(\mathbf{u}^{n+1},\mathbf{u}^{n+1})_\Omega+(\mathcal{A}_{h_B}\mathbf{X}^{n+1},\mathbf{X}^{n+1})_B\le(\mathbf{u}^n,\mathbf{u}^n)_\Omega+(\mathcal{A}_{h_B}\mathbf{X}^n,\mathbf{X}^n)_B \quad (12)$$

Thought to be prohibitively expensive to solve

1A discretization of Newren, Fogelson, Guy, Kirby

$$\rho\left(\frac{\mathbf{u}^{n+1}-\mathbf{u}^n}{\Delta t}+\mathbf{u}^n\cdot\mathbf{D}_h\mathbf{u}^n\right)=-\mathbf{D}_h p^{n+1}+\mu L_h\mathbf{u}^{n+1}+\mathcal{S}_n\mathcal{A}_{h_B}(\mathbf{X}^{n+1}),$$
$$\mathbf{D}_h\cdot\mathbf{u}^{n+1}=0,$$
$$\frac{\mathbf{X}^{n+1}-\mathbf{X}^n}{\Delta t}=\mathcal{S}_n^*\mathbf{u}^{n+1},$$

Originally proposed by Peskin in 1971

Proven to be unconditionally stable for non-convective flows with linear, self-adjoint fiber force,

$$(\mathbf{u}^{n+1},\mathbf{u}^{n+1})_\Omega+(\mathcal{A}_{h_B}\mathbf{X}^{n+1},\mathbf{X}^{n+1})_B\leq(\mathbf{u}^n,\mathbf{u}^n)_\Omega+(\mathcal{A}_{h_B}\mathbf{X}^n,\mathbf{X}^n)_B \quad (12)$$

Thought to be prohibitively expensive to solve

Our work has focused on solving this system efficiently

Let's simplify

Let's simplify

We introduce a discrete projection operator P_h

Let's simplify

We introduce a discrete projection operator P_h defined as

$$\mathbf{v} = P_h\mathbf{v} + \mathbf{D}_h\phi_v, \qquad \mathbf{D}_h \cdot P_h\mathbf{v} = 0, \qquad P_h\mathbf{D}_h\phi_v = 0. \tag{13}$$

for any smooth vector field $\mathbf{v}$ defined on the grid $\mathcal{G}_\Omega$.

Let's simplify

We introduce a discrete projection operator P_h defined as

$$\mathbf{v} = P_h \mathbf{v} + \mathbf{D}_h \phi_v, \qquad \mathbf{D}_h \cdot P_h \mathbf{v} = 0, \qquad P_h \mathbf{D}_h \phi_v = 0. \tag{13}$$

for any smooth vector field $\mathbf{v}$ defined on the grid $\mathcal{G}_\Omega$. We then define

$$\mathcal{L}_h = (I - \nu \Delta t L_h)^{-1} P_h, \tag{14}$$

Let's simplify

We introduce a discrete projection operator P_h defined as

$$\mathbf{v} = P_h\mathbf{v} + \mathbf{D}_h\phi_v, \qquad \mathbf{D}_h \cdot P_h\mathbf{v} = 0, \qquad P_h\mathbf{D}_h\phi_v = 0. \tag{13}$$

for any smooth vector field $\mathbf{v}$ defined on the grid $\mathcal{G}_\Omega$. We then define

$$\mathcal{L}_h = (I - \nu\Delta t L_h)^{-1} P_h, \tag{14}$$

which we refer to as the fluid solver.

Now we may write

$$\mathbf{u}^{n+1} = \mathcal{L}_h \left[\frac{\Delta t}{\rho} \mathcal{S}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{u}^n - \Delta t \ \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n \right] \tag{15}$$

$$\mathbf{X}^{n+1} = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathbf{u}^{n+1}. \tag{16}$$

Reduced system

Reduced system

Eliminating $\mathbf{u}^{n+1}$ gives

$$\mathbf{X}^{n+1} = \mathcal{M}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{b}^n \tag{17}$$

Reduced system

Eliminating $\mathbf{u}^{n+1}$ gives

$$\mathbf{X}^{n+1} = \mathcal{M}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{b}^n \tag{17}$$

where

$$\mathcal{M}_n = \frac{(\Delta t)^2}{\rho} \mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n, \tag{18}$$

and

$$\mathbf{b}^n = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathcal{L}_h [\mathbf{u}^n - \Delta t \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n]. \tag{19}$$

Reduced system

Eliminating $\mathbf{u}^{n+1}$ gives

$$\mathbf{X}^{n+1} = \mathcal{M}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{b}^n \tag{17}$$

where

$$\mathcal{M}_n = \frac{(\Delta t)^2}{\rho} \mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n, \tag{18}$$

and

$$\mathbf{b}^n = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathcal{L}_h [\mathbf{u}^n - \Delta t \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n]. \tag{19}$$

If $\mathcal{A}_{h_B}$ is linear, then (17) is linear!

Reduced system

Eliminating $\mathbf{u}^{n+1}$ gives

$$\mathbf{X}^{n+1} = \mathcal{M}_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1}) + \mathbf{b}^n \tag{17}$$

where

$$\mathcal{M}_n = \frac{(\Delta t)^2}{\rho} \mathcal{S}_n^* \mathcal{L}_h \mathcal{S}_n, \tag{18}$$

and

$$\mathbf{b}^n = \mathbf{X}^n + \Delta t \mathcal{S}_n^* \mathcal{L}_h [\mathbf{u}^n - \Delta t \mathbf{u}^n \cdot \nabla_h \mathbf{u}^n]. \tag{19}$$

If $\mathcal{A}_{h_B}$ is linear, then (17) is linear!

In fact, $I - \mathcal{M}_n \mathcal{A}_{h_B}$ is positive-definite if $\mathcal{A}_{h_B}$ is negative-semidefinite.

How can we efficiently solve $(I - \mathcal{M}_n \mathcal{A}_{h_B})\mathbf{X}^{n+1} = \mathbf{b}^n$?

Iterative methods require an expensive fluid solve at every iteration

How can we efficiently solve $(I - \mathcal{M}_n \mathcal{A}_{h_B})\mathbf{X}^{n+1} = \mathbf{b}^n$?

Iterative methods require an expensive fluid solve at every iteration

A matrix representation of $I - \mathcal{M}_n \mathcal{A}_{h_B}$ would greatly speed iterative methods

Can we efficiently construct the matrix representation of $\mathcal{M}_n$?

Can we efficiently construct the matrix representation of $\mathcal{M}_n$?

Previously thought to be prohibitively expensive at $O(N_b N^2 \log N)$.

Can we efficiently construct the matrix representation of $\mathcal{M}_n$?

Previously thought to be prohibitively expensive at $O(N_b N^2 \log N)$.

Our work shows it can be constructed at $O(N_b^2)$

Can we efficiently construct the matrix representation of $\mathcal{M}_n$?

Previously thought to be prohibitively expensive at $O(N_b N^2 \log N)$.

Our work shows it can be constructed at $O(N_b^2)$

The key is an approximation inspired by the continuous equations, as well as a reliance on lookup tables.

Disadvantages to using the matrix representation of $\mathcal{M}_n$

Disadvantages to using the matrix representation of $\mathcal{M}_n$

- Some error introduced via our lookup tables

Disadvantages to using the matrix representation of $\mathcal{M}_n$

- Some error introduced via our lookup tables
- Prohibitively expensive when $N_b \gg N$, generally the case in 3D

Disadvantages to using the matrix representation of $\mathcal{M}_n$

- Some error introduced via our lookup tables
- Prohibitively expensive when $N_b \gg N$, generally the case in 3D
- Slightly more complicated code

Advantages to having the matrix representation of $\mathcal{M}_n$

- Exact solutions via Gaussian elimination

Advantages to having the matrix representation of $\mathcal{M}_n$

- Exact solutions via Gaussian elimination
- Faster evaluations of quantities of the form $\mathcal{M}_n\mathbf{F}$, by a factor of 20 to 100. This greatly streamlines iterative methods.

$N_B = 2N$	256	512	1024
Matrix-Vector product	1/26.8	1/58	1/71

Advantages to having the matrix representation of $\mathcal{M}_n$

- Exact solutions via Gaussian elimination
- Faster evaluations of quantities of the form $\mathcal{M}_n\mathbf{F}$, by a factor of 20 to 100. This greatly streamlines iterative methods.

$N_B = 2N$	256	512	1024
Matrix-Vector product	1/26.8	1/58	1/71

- Simpler implementation of a multigrid code using an algebraic multigrid rather than a geometric multigrid

Advantages to having the matrix representation of $\mathcal{M}_n$

- Exact solutions via Gaussian elimination
- Faster evaluations of quantities of the form $\mathcal{M}_n\mathbf{F}$, by a factor of 20 to 100. This greatly streamlines iterative methods.

$N_B = 2N$	256	512	1024
Matrix-Vector product	1/26.8	1/58	1/71

- Simpler implementation of a multigrid code using an algebraic multigrid rather than a geometric multigrid
- Potential for a wider range of iterative methods, including Gauss-Seidel and S.O.R. Importantly, these are the only viable smoothers we've found

Advantages to having the matrix representation of $\mathcal{M}_n$

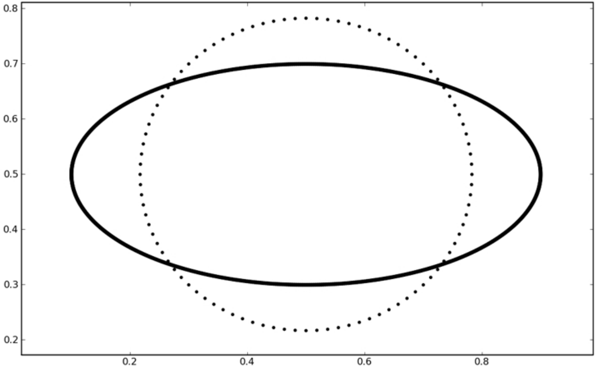
- Exact solutions via Gaussian elimination
- Faster evaluations of quantities of the form $\mathcal{M}_n\mathbf{F}$, by a factor of 20 to 100. This greatly streamlines iterative methods.

$N_B = 2N$	256	512	1024
Matrix-Vector product	1/26.8	1/58	1/71

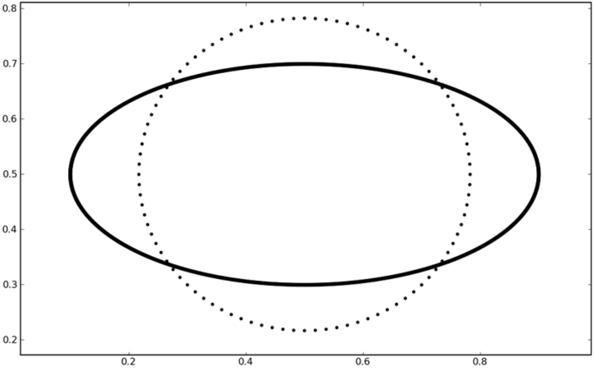
- Simpler implementation of a multigrid code using an algebraic multigrid rather than a geometric multigrid
- Potential for a wider range of iterative methods, including Gauss-Seidel and S.O.R. Importantly, these are the only viable smoothers we've found
- Multigrid!

Simple application: ellipse relaxation

Simple application: ellipse relaxation

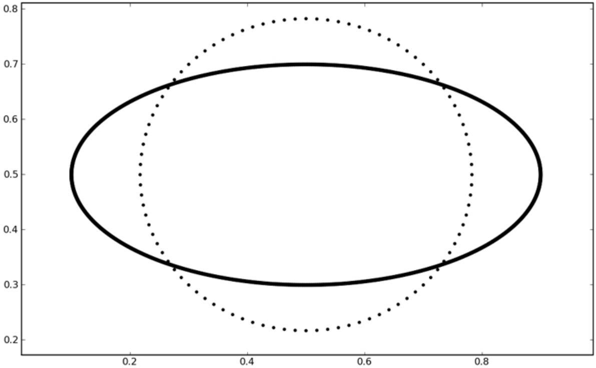


Simple application: ellipse relaxation



$$\mathcal{A}_{h_B}(\mathbf{X}) = \sigma \mathbf{X}_{ss} \tag{20}$$

Simple application: ellipse relaxation



$$\mathcal{A}_{h_B}(\mathbf{X}) = \sigma \mathbf{X}_{ss} \tag{20}$$

$\mathcal{A}_{h_B}$ is linear, negative-semidefinite, hence we can apply our multigrid.

Comparison of total CPU between an explicit method and our implicit implementation

Table: Elliptical drop relaxation for Navier Stokes. The average CPU time per time-step and the total CPU time up to a simulation time of $T = 0.005$ is given. Δt is the time-step taken and is the maximum allowed while maintaining stability.

	Explicit			Implicit		
N	Δt	Average	Total	Δt	Average	Total
128	$1.95 \cdot 10^{-6}$	0.03	73.35	$1.17 \cdot 10^{-4}$	0.10	4.34
256	$9.76 \cdot 10^{-7}$	0.14	730.21	$7.81 \cdot 10^{-5}$	0.55	35.06
384	$6.51 \cdot 10^{-7}$	0.34	2613.19	$5.21 \cdot 10^{-5}$	1.24	118.98
512	$4.87 \cdot 10^{-7}$	0.63	6491.95	$3.91 \cdot 10^{-5}$	2.34	299.10

Comparison of total CPU between an explicit method and our implicit implementation

Table: Elliptical drop relaxation for Navier Stokes. The average CPU time per time-step and the total CPU time up to a simulation time of $T = 0.005$ is given. Δt is the time-step taken and is the maximum allowed while maintaining stability.

	Explicit			Implicit		
N	Δt	Average	Total	Δt	Average	Total
128	$1.95 \cdot 10^{-6}$	0.03	73.35	$1.17 \cdot 10^{-4}$	0.10	4.34
256	$9.76 \cdot 10^{-7}$	0.14	730.21	$7.81 \cdot 10^{-5}$	0.55	35.06
384	$6.51 \cdot 10^{-7}$	0.34	2613.19	$5.21 \cdot 10^{-5}$	1.24	118.98
512	$4.87 \cdot 10^{-7}$	0.63	6491.95	$3.91 \cdot 10^{-5}$	2.34	299.10

What if $\mathcal{A}_{h_B}$ isn't linear?

- When the Jacobian of $\mathcal{A}_{h_B}$ is negative semidefinite we can solve Newton iterations via multigrid. This works well.
- Unfortunately, multigrid fails if the Jacobian isn't definite
- In certain important cases, we can fix the problem by opting for a fixed-point iteration

Fixed-point iteration

Fixed-point iteration

The most immediate choice for a fixed point iteration is

$$\mathbf{X}^{n+1,k+1} = M_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1,k}) + \mathbf{b}^n. \tag{21}$$

Fixed-point iteration

The most immediate choice for a fixed point iteration is

$$\mathbf{X}^{n+1,k+1} = M_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1,k}) + \mathbf{b}^n. \tag{21}$$

However, when the eigenvalues of the Jacobian of $\mathcal{A}_{h_B}$ are large, this iteration diverges spectacularly.

Fixed-point iteration

The most immediate choice for a fixed point iteration is

$$\mathbf{X}^{n+1,k+1} = M_n \mathcal{A}_{h_B}(\mathbf{X}^{n+1,k}) + \mathbf{b}^n. \tag{21}$$

However, when the eigenvalues of the Jacobian of $\mathcal{A}_{h_B}$ are large, this iteration diverges spectacularly.

Intuitively, the reverse of this iteration should be stable.

Fixed-point iteration

Fixed-point iteration

We dub the following the splitting method

$$\begin{cases} M_n \mathbf{F}^{k+1} = \mathbf{X}^{n+1,k} - \mathbf{b}^n, \\ \mathcal{A}_{h_B}(\mathbf{X}^{n+1,k+1}) = \mathbf{F}^{k+1}. \end{cases} \tag{22}$$

Fixed-point iteration

We dub the following the splitting method

$$\begin{cases} M_n \mathbf{F}^{k+1} = \mathbf{X}^{n+1,k} - \mathbf{b}^n, \\ \mathcal{A}_{h_B}(\mathbf{X}^{n+1,k+1}) = \mathbf{F}^{k+1}. \end{cases} \tag{22}$$

We can invert M_n via our multigrid solver

Fixed-point iteration

We dub the following the splitting method

$$\begin{cases} M_n \mathbf{F}^{k+1} = \mathbf{X}^{n+1,k} - \mathbf{b}^n, \\ \mathcal{A}_{h_B}(\mathbf{X}^{n+1,k+1}) = \mathbf{F}^{k+1}. \end{cases} \tag{22}$$

We can invert M_n via our multigrid solver

We can invert $\mathcal{A}_{h_B}$ via Newton iterations.
Typically the Jacobian of $\mathcal{A}_{h_B}$ is sparse, so this process is cheap.

Fixed-point iteration

We dub the following the splitting method

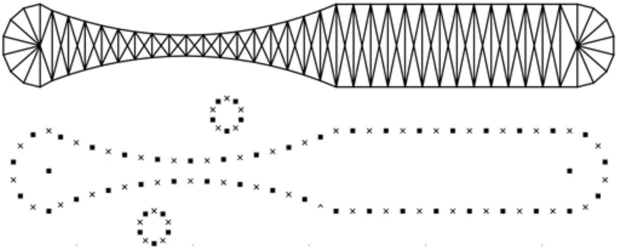
$$\begin{cases} M_n \mathbf{F}^{k+1} = \mathbf{X}^{n+1,k} - \mathbf{b}^n, \\ \mathcal{A}_{h_B}(\mathbf{X}^{n+1,k+1}) = \mathbf{F}^{k+1}. \end{cases} \tag{22}$$

We can invert M_n via our multigrid solver

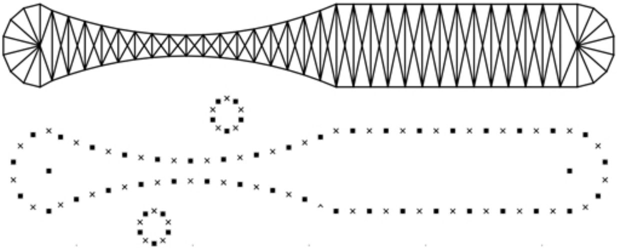
We can invert $\mathcal{A}_{h_B}$ via Newton iterations.
Typically the Jacobian of $\mathcal{A}_{h_B}$ is sparse, so this process is cheap.

Provided the eigenvalues of the Jacobian of $\mathcal{A}_{h_B}$ are negative and large enough, (22) converges rapidly.

Application: blood flow past a heart valve



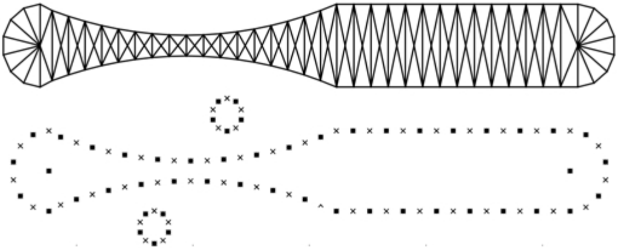
Application: blood flow past a heart valve



The force between two nodes is given by Hooke's law

$$\mathbf{f}_{i,j} = -k_{i,j} \frac{\mathbf{X}_i - \mathbf{X}_j}{|\mathbf{X}_i - \mathbf{X}_j|}(|\mathbf{X}_i - \mathbf{X}_j| - L_{i,j}) \tag{23}$$

Application: blood flow past a heart valve

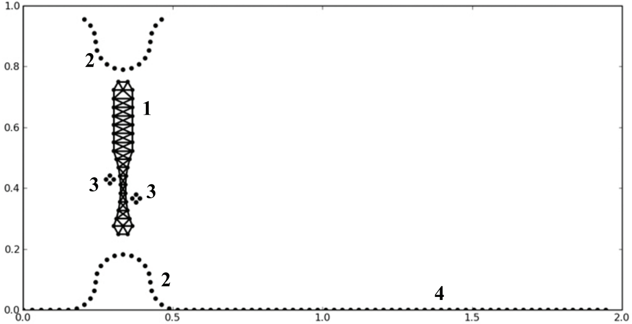


The force between two nodes is given by Hooke's law

$$\mathbf{f}_{i,j} = -k_{i,j} \frac{\mathbf{X}_i - \mathbf{X}_j}{|\mathbf{X}_i - \mathbf{X}_j|} (|\mathbf{X}_i - \mathbf{X}_j| - L_{i,j}) \tag{23}$$

This is nonlinear when $L_{i,j} \neq 0$!

Application: blood flow past a heart valve



- 1. Valve
- 2. Cushions
- 3. Hinges
- 4. Artery wall

Application: blood flow past a heart valve

A horizontal flow is induced by adding a a time dependent component to the force field:

$$v_{flow}(t) = \begin{cases} 60000t(0.1 - t) & t < 0.1 \\ -60000(t - 0.1)(0.2 - t) & t \geq 0.1 \end{cases} . \tag{24}$$

Application: blood flow past a heart valve

Problem:

Application: blood flow past a heart valve

Problem:

$\mathcal{A}_{h_B}$ isn't invertible! It is neither 1-1 nor onto.

Application: blood flow past a heart valve

Problem:

$\mathcal{A}_{h_B}$ isn't invertible! It is neither 1-1 nor onto.

$\mathcal{A}_{h_B}$ can't produce translational or rotational forces.
$\mathcal{A}_{h_B}$ is insensitive to translation of its input

Factoring out components outside the image of $\mathcal{A}_{h_B}$

We define vectors

$$\mathbf{V}_I^1 = \begin{cases} (1,0) & \text{if } I \in \mathcal{G}_V, \\ (0,0) & \text{otherwise}, \end{cases} \tag{25}$$

$$\mathbf{V}_I^2 = \begin{cases} (0,1) & \text{if } I \in \mathcal{G}_V, \\ (0,0) & \text{otherwise}, \end{cases} \tag{26}$$

$$\mathbf{V}_I^3 = \begin{cases} (-Y_I + y_c, X_I - x_c) & \text{if } I \in \mathcal{G}_V, \\ (0,0) & \text{otherwise}, \end{cases} \tag{27}$$

where $\mathcal{G}_V$ is the collection of all indices I such that $\mathbf{X}_I$ is a fiber point belonging to the valve, and (x_c, y_c) is some fixed point.

Factoring out components outside the image of $\mathcal{A}_{h_B}$

The vectors $\mathbf{V}^1, \mathbf{V}^2, \mathbf{V}^3$ are points outside the image of $\mathcal{A}_{h_B}$

Factoring out components outside the image of $\mathcal{A}_{h_B}$

The vectors $\mathbf{V}^1, \mathbf{V}^2, \mathbf{V}^3$ are points outside the image of $\mathcal{A}_{h_B}$
When solving $\mathcal{A}_{h_B}\mathbf{X}^{n+1,k+1} = \mathbf{F}^{k+1}$ we use a modified Newton iteration

Factoring out components outside the image of $\mathcal{A}_{h_B}$

The vectors $\mathbf{V}^1, \mathbf{V}^2, \mathbf{V}^3$ are points outside the image of $\mathcal{A}_{h_B}$
When solving $\mathcal{A}_{h_B}\mathbf{X}^{n+1,k+1} = \mathbf{F}^{k+1}$ we use a modified Newton iteration

$$\begin{cases} \mathbf{X}^{n+1,k+1,0} = \mathbf{X}^{n+1,k}, \\ J(\mathbf{X}^{n+1,k+1,l})(\mathbf{X}^{n+1,k+1,l+1} - \mathbf{X}^{n+1,k+1,l}) \\ \qquad = P(\mathbf{F}^{k+1} - \mathcal{A}_{h_B}(\mathbf{X}^{n+1,k+1,l})), \end{cases} \tag{28}$$

where

$$P(\mathbf{F}) = \mathbf{F} - \sum_{j=1}^{3} \mathbf{V}^j(\mathbf{X}^{n+1,k})(\mathbf{F}, \mathbf{V}^j(\mathbf{X}^{n+1,k}))_B \tag{29}$$

Reintroducing translation and rotation

Modified Newton iterations converge quickly

Reintroducing translation and rotation

Modified Newton iterations converge quickly

Unfortunately, they don't converge to the right answer

Reintroducing translation and rotation

Modified Newton iterations converge quickly

Unfortunately, they don't converge to the right answer

We must reintroduce translation and rotation

Reintroducing translation and rotation

Modified Newton iterations converge quickly

Unfortunately, they don't converge to the right answer

We must reintroduce translation and rotation

There are simple, fully explicit, and stable procedures for calculating the translation and rotation.

Comparison of total CPU time for the heart valve simulation

Table: Heart valve simulation. The average CPU time per time-step and the total CPU time up to a simulation time of $T = 0.5$ is given. For the explicit method Δt is the time-step taken and is the maximum allowed while maintaining stability. For the implicit method Δt is held constant, and is $O(h)$ or smaller.

	Explicit			Implicit		
N	Δt	Average	Total	Δt	Average	Total
128	$1.15 \cdot 10^{-7}$	0.024	105089*	0.0025	2.001	118
256	$2.89 \cdot 10^{-8}$	0.109	1884358*	0.0025	7.640	402
384	$1.29 \cdot 10^{-8}$	0.249	9658428*	0.0025	18.735	949
512	$7.24 \cdot 10^{-9}$	0.503	34714584*	0.0025	34.984	1907

Comparison of total CPU time for the heart valve simulation

Table: Heart valve simulation. The average CPU time per time-step and the total CPU time up to a simulation time of $T = 0.5$ is given. For the explicit method Δt is the time-step taken and is the maximum allowed while maintaining stability. For the implicit method Δt is held constant, and is $O(h)$ or smaller.

	Explicit			Implicit		
N	Δt	Average	Total	Δt	Average	Total
128	$1.15 \cdot 10^{-7}$	0.024	105089*	0.0025	2.001	118
256	$2.89 \cdot 10^{-8}$	0.109	1884358*	0.0025	7.640	402
384	$1.29 \cdot 10^{-8}$	0.249	9658428*	0.0025	18.735	949
512	$7.24 \cdot 10^{-9}$	0.503	34714584*	0.0025	34.984	1907

Current research

Current research is focused on extending the methodology to 3D. The main problems are

- The matrix $\mathcal{M}_n$ has $O(N^4)$ entrees, so can't be used directly
- Without the matrix we can't make use of Gauss-Seidel
- Without a good smoother we can't implement a multigrid

Current research

Current research is focused on extending the methodology to 3D. The main problems are

- The matrix $\mathcal{M}_n$ has $O(N^4)$ entrees, so can't be used directly
- Without the matrix we can't make use of Gauss-Seidel
- Without a good smoother we can't implement a multigrid

Current lines of attack include

- Looking at alternative smoothers, potentially involving banded components of $\mathcal{M}_n$
- Looking at good preconditioners for non-multigrid solvers, such as GMRES
- Modifying a Fast Multipole Method to implement Gauss-Seidel in $O(N_b)$ steps