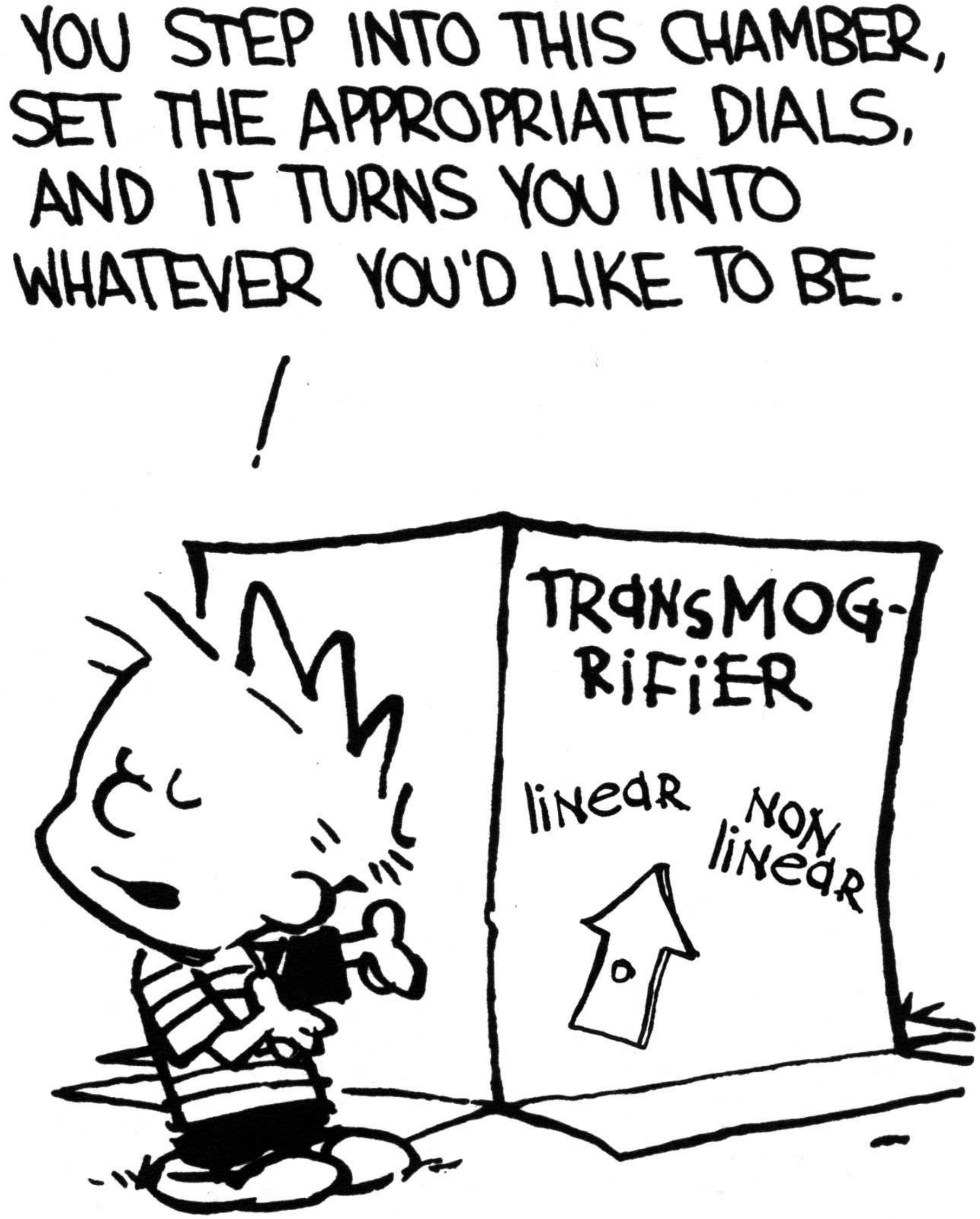

YOU STEP INTO THIS CHAMBER, SET THE APPROPRIATE DIALS, AND IT TURNS YOU INTO WHATEVER YOU'D LIKE TO BE.
TRANSMOG-RIFIER
linear
NON linear

MULTISCALE BASIS OPTIMIZATION FOR DARCY FLOW

James Rath

Supervisor: Todd Arbogast
Committee: Steve Bryant
Clint Dawson
Robert van de Geijn
Mary Wheeler

Friday, April 13, 2007

Outline

- ♣ Quick review of linear algebra
- ♣ Application of interest
- ♣ Some empirical results
- ♣ Future research directions

Outline

- Quick review of linear algebra
- Application of interest
- Some empirical results
- Future research directions

Problem: solving linear systems

Solving large, sparse linear systems often requires the user of iterative solvers. Storage and speed motivate their use.

Nice features of iterative solvers

Solving large, sparse linear systems often requires the user of iterative solvers. Storage and speed motivate their use.

- Global convergence for any initialization and problem data.
- Monotone convergence in some norm.
- Can be optimal in storage and speed.

Not-so-nice features of iterative solvers

Solving large, sparse linear systems often requires the user of iterative solvers. Storage and speed motivate their use.

- Global convergence for any initialization and problem data.
- Monotone convergence in some norm.
- Can be optimal in storage and speed.

However, generally speaking:

- Solvers get only linear convergence rates toward the solution from the initial guess.
- Large condition numbers mean convergence slows down: solvers require more and more iterations.

What we're after

Solving large, sparse linear systems often requires the user of iterative solvers. Storage and speed motivate their use.

- Global convergence for any initialization and problem data.
- Monotone convergence in some norm.
- Can be optimal in storage and speed.

However, generally speaking:

- Solvers get only linear convergence rates toward the solution from the initial guess.
- Large condition numbers mean convergence slows down: solvers require more and more iterations.

We want to do better on both these counts while keeping the attractive features.

Solving linear systems requires nonlinear operations, namely, division.

$$10x = 17 \quad \Rightarrow \quad x = \frac{17}{10}$$

The little Newton engine that could

Solving linear systems requires nonlinear operations, namely, division.

$$10x = 17 \quad \Rightarrow \quad x = \frac{17}{10}$$

Iterative nonlinear solvers (Newton's method and its ilk):

- Get fast quadratic convergence to a solution, and
- As applied to discretizations of nonlinear PDE,
  are insensitive to mesh size and other problem parameters.

Our goal

Solving linear systems requires nonlinear operations, namely, division.

$$10x = 17 \quad \Rightarrow \quad x = \frac{17}{10}$$

Iterative nonlinear solvers (Newton's method and its ilk):

- Get fast quadratic convergence to a solution, and
- As applied to discretizations of nonlinear PDE,
  are insensitive to mesh size and other problem parameters.

We want to carry over these properties to solving linear systems.

That was fast ...

A naive application of Newton's method to solving a linear system results in a one-step procedure.

One small step for an iterative procedure

A naive application of Newton's method to solving a linear system results in a one-step procedure.

Solving:

$$Au = f$$

Objection function:

$$F(u) = f - Au$$

Jacobian:

$$F'(u) = -A$$

Newton step:

$$\begin{aligned} \color{purple}{u_{i+1}} &= u_i - (-A)^{-1}(f - Au_i) \\ &= u_i + u - u_i \\ &= \color{purple}{u} \end{aligned}$$

One giant leap for the Jacobian solver

To solve your linear system ...

Solving:

$$Au = f$$

Newton step:

$$u_{i+1} = u_i - (-A)^{-1}(f - Au_i)$$

... you must solve your linear system.

Whoops

To solve your linear system ...

Solving:

$$Au = f$$

Newton step:

$$u_{i+1} = u_i - (-A)^{-1}(f - Au_i)$$

... you must solve your linear system.

And that's no fun!

Especially if it's a $10^6 \times 10^6$ sparse, ill-conditioned sytem you want to solve.

We need a smaller piece to chew on

To solve your linear system ...

Solving:

$$Au = f$$

Newton step:

$$u_{i+1} = u_i - (-A)^{-1}(f - Au_i)$$

... you must solve your linear system.

We have to try harder to find a nonlinear piece to attack, but it's not obvious where to begin or what will be successful.

Let's examine a 3×3 linear system just to keep things simple.

$$
\begin{array}{ccc}
A & u & = & f \\
\begin{bmatrix} 10 & -6 & 4 \\ -6 & 17 & 0 \\ 4 & 0 & 9 \end{bmatrix} & \begin{bmatrix} x \\ y \\ z \end{bmatrix} & = & \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}
\end{array}
$$

A 3×3 example with a twist

Let's examine a 3×3 linear system just to keep things simple.

$$
\begin{array}{ccc}
A & u & = f \\
\begin{bmatrix} 10 & -6 & 4 \\ -6 & 17 & 0 \\ 4 & 0 & 9 \end{bmatrix} & \begin{bmatrix} \rho \cos\theta \sin\phi \\ \rho \sin\theta \sin\phi \\ \rho \cos\phi \end{bmatrix} & = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}
\end{array}
$$

But let's use polar coordinates to represent the unknown.

Let's examine a 3×3 linear system just to keep things simple.

$$A \qquad U_\sigma \rho \qquad = \qquad f$$

$$\begin{bmatrix} 10 & -6 & 4 \\ -6 & 17 & 0 \\ 4 & 0 & 9 \end{bmatrix} \begin{bmatrix} \cos\theta \sin\phi \\ \sin\theta \sin\phi \\ \cos\phi \end{bmatrix} \rho = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

But let's use polar coordinates to represent the unknown.
And separate direction (or shape) from magnitude.

Let's examine a 3×3 linear system just to keep things simple.

$$A \qquad U_\sigma \rho = f$$

$$\begin{bmatrix} 10 & -6 & 4 \\ -6 & 17 & 0 \\ 4 & 0 & 9 \end{bmatrix} \begin{bmatrix} \cos\theta \sin\phi \\ \sin\theta \sin\phi \\ \cos\phi \end{bmatrix} \rho = \begin{bmatrix} 10 \\ 5 \\ -1 \end{bmatrix}$$

But let's use polar coordinates to represent the unknown.
And separate direction (or shape) from magnitude.

$$\sigma = (\theta, \phi)$$

A nonlinear problem

Goal: Find a zero of the objective function

$$r(\sigma, \rho) = f - AU_\sigma \rho$$

Split: some linear, some nonlinear

Goal: Find a zero of the objective function

$$r(\sigma) = f - AU_\sigma \rho(\sigma)$$

Determine ρ as the "best" magnitude for a fixed σ:

$$AU_\sigma \rho = f$$

Split: some linear, some nonlinear

Goal: Find a zero of the objective function

$$r(\sigma) = f - AU_\sigma \rho(\sigma)$$

Determine ρ as the "best" magnitude for a fixed σ:

$$(U_\sigma^T A U_\sigma)\rho = U_\sigma^T f$$

Where:

- "Best" $=$ best in least-squares sense (in the energy or A-norm).

Split: some linear, some nonlinear

Goal: Find a zero of the objective function

$$r(\sigma) = f - AU_\sigma \rho(\sigma)$$

Determine ρ as the "best" magnitude for a fixed σ:

$$(U_\sigma^T A U_\sigma)\rho = U_\sigma^T f$$

Where:

- "Best" $=$ best in least-squares sense (in the energy or A-norm).
- The system $U_\sigma^T A U_\sigma$ is a smaller/coarser linear system to solve.

Algorithm à la Newton

1. Choose a shape σ. (Fix for now.)

Algorithm à la Newton

1. Choose a shape σ.
2. Solve for ρ:

$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$

   This is an "easy" coarsened problem.

Algorithm à la Newton

1. Choose a shape σ.
2. Solve for ρ:
$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$
3. Calculate objective/residual:
$$r(\sigma) = f - AU_\sigma \rho$$

Algorithm à la Newton

1. Choose a shape σ.
2. Solve for ρ:
$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$
3. Calculate objective/residual:
$$r(\sigma) = f - AU_\sigma\rho$$
4. Calculate Jacobian $r'(\sigma)$.

Algorithm à la Newton

1. Choose a shape σ.
2. Solve for ρ:
$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$
3. Calculate objective/residual:
$$r(\sigma) = f - AU_\sigma \rho$$
4. Calculate Jacobian $r'(\sigma)$.
   Oops, oh yeah ...

Algorithm à la Newton

1. Choose a shape σ.
2. Solve for ρ:
$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$
3. Calculate objective/residual:
$$r(\sigma) = f - AU_\sigma \rho$$
4. Calculate Jacobian $r'(\sigma)$.
5. Calculate Newton step:
$$\delta\sigma = -\left(r'\right)^\dagger r$$

Algorithm à la Newton

1. Choose a shape σ.
2. Solve for ρ:
$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$
3. Calculate objective/residual:
$$r(\sigma) = f - AU_\sigma\rho$$
4. Calculate Jacobian $r'(\sigma)$.
5. Calculate Newton step:
$$\delta\sigma = -\left(r'\right)^\dagger r$$
6. Update shape σ:
$$\sigma \leftarrow \sigma + \delta\sigma$$

Algorithm à la Newton

1. Choose a shape σ.
2. Solve for ρ:
$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$
3. Calculate objective/residual:
$$r(\sigma) = f - AU_\sigma\rho$$
4. Calculate Jacobian $r'(\sigma)$.
5. Calculate Newton step:
$$\delta\sigma = -\left(r'\right)^\dagger r$$
6. Update shape σ:
$$\sigma \leftarrow \sigma + \delta\sigma$$
7. Repeat as necessary.

Bummer

- Calculating Jacobian $r'(\sigma)$
- Solving the linear system $\left(r'\right)^{\dagger} r$

Bummer

- Calculating Jacobian $r'(\sigma)$
- Solving the linear system $(r')^{\dagger} r$

Calculus ... yuck!

- Calculating Jacobian $r'(\sigma)$
- Solving the linear system $(r')^{\dagger} r$

Jacobians require calculus, and who wants to do calculus?

Linear algebra is much easier

- Calculating Jacobian $r'(\sigma)$
- Solving the linear system $(r')^\dagger r$

Jacobians require calculus, and who wants to do calculus?
Blech! I wanna do linear algebra ...

Jacobians are expensive, anyway

- Calculating Jacobian $r'(\sigma)$
- Solving the linear system $(r')^{\dagger} r$

Jacobians require calculus, and who wants to do calculus?
Blech! I wanna do linear algebra ...

(Jacobians are expensive to compute, anyway.)

To be lazy, one must do work ...

- Calculating Jacobian $r'(\sigma)$
- Solving the linear system $(r')^{\dagger} r$

Jacobians require calculus, and who wants to do calculus?
Blech! I wanna do linear algebra ...

We'll use calculus to avoid calculus.

Chain rule to the rescue!

S'pose instead of computing the Newton step:

$$\delta\sigma = -\left(r'\right)^{\dagger} r$$

Chain rule to the rescue!

S'pose instead of computing the Newton step:

$$\delta\sigma = -\left(r'\right)^{\dagger} r$$

We compute the effect that the Newton step would have on the residual:

$$\delta r = \quad \left(r'\right) \delta\sigma$$

Chain rule to the rescue!

S'pose instead of computing the Newton step:

$$\delta\sigma = -\left(r'\right)^{\dagger} r$$

We compute the effect that the Newton step would have on the residual:

$$\begin{aligned}\delta r &= \quad \left(r'\right)\delta\sigma \\ &= -\left(r'\right)\left(r'\right)^{\dagger} r\end{aligned}$$

A-ha!

S'pose instead of computing the Newton step:

$$\delta\sigma = -\left(r'\right)^{\dagger} r$$

We compute the effect that the Newton step would have on the residual:

$$\begin{aligned} \delta r &= \quad \left(r'\right) \delta\sigma \\ &= -\left(r'\right)\left(r'\right)^{\dagger} r \end{aligned}$$

The operation $\left(r'\right)\left(r'\right)^{\dagger}$ is something familiar: the projection onto the range of r'!

A-ha!

S'pose instead of computing the Newton step:

$$\delta\sigma = -(r')^{\dagger} r$$

We compute the effect that the Newton step would have on the residual:

$$\begin{aligned} \delta r &= \quad (r')\delta\sigma \\ &= -(r')(r')^{\dagger} r \end{aligned}$$

The operation $(r')(r')^{\dagger}$ is something familiar: the projection onto the range of r'!

The range of r' is the tangent space to the manifold of all possible residual vectors — an ellipsoid. The normal, it turns out, is easy to compute.

S'pose instead of computing the Newton step:

$$\delta\sigma = -\left(r'\right)^{\dagger} r$$

We compute the effect that the Newton step would have on the residual:

$$\begin{aligned}\delta r &= \quad \left(r'\right)\delta\sigma \\ &= -\left(r'\right)\left(r'\right)^{\dagger} r\end{aligned}$$

The operation $\left(r'\right)\left(r'\right)^{\dagger}$ is something familiar: the projection onto the range of r'!

The range of r' is the tangent space to the manifold of all possible residual vectors — an ellipsoid. The normal, it turns out, is easy to compute.

A projection is a linear algebra sorta thing. And it's a projection onto a low-dimensional space (1-D here). So it's "easy"!

S'pose instead of computing the Newton step:

$$\delta\sigma = -\left(r'\right)^{\dagger} r$$

We compute the effect that the Newton step would have on the residual:

$$\begin{aligned}\delta r &= \left(r'\right)\delta\sigma \\ &= -\left(r'\right)\left(r'\right)^{\dagger} r\end{aligned}$$

The operation $\left(r'\right)\left(r'\right)^{\dagger}$ is something familiar: the projection onto the range of r'!

The range of r' is the tangent space to the manifold of all possible residual vectors — an ellipsoid. The normal, it turns out, is easy to compute.

A projection is a linear algebra sorta thing. And it's a projection onto a low-dimensional space (1-D here). So it's "easy"!

So how do we use this?

Algorithm v. 2.0

1. Choose a shape σ.

Algorithm v. 2.0

1. Choose a shape σ.
2. Solve coarse problem for ρ:

$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$

Algorithm v. 2.0

1. Choose a shape σ.
2. Solve coarse problem for ρ:

$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$

3. Calculate objective/residual:

$$r(\sigma) = f - AU_\sigma \rho$$

Algorithm v. 2.0

1. Choose a shape σ.
2. Solve coarse problem for ρ:
$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$
3. Calculate objective/residual:
$$r(\sigma) = f - AU_\sigma \rho$$
4. Calculate residual change (Newton step):
$$\delta r = -P_{\tan} r$$

Algorithm v. 2.0

1. Choose a shape σ.
2. Solve coarse problem for ρ:

$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$

3. Calculate objective/residual:

$$r(\sigma) = f - AU_\sigma \rho$$

4. Calculate residual change (Newton step):

$$\delta r = -P_{\mathrm{tan}} r$$

5. Impute new shape from updated residual $r + \delta r$.

Algorithm v. 2.0

1. Choose a shape σ.
2. Solve coarse problem for ρ:

$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$

3. Calculate objective/residual:

$$r(\sigma) = f - AU_\sigma \rho$$

4. Calculate residual change (Newton step):

$$\delta r = -P_{\tan} r$$

5. Impute new shape from updated residual $r + \delta r$.
6. Repeat as necessary.

Wait, aren't nonlinear problems hard?

We went from a linear problem to a nonlinear one, but ...

What have we *done*?!

We went from a linear problem to a nonlinear one, but ...

We have traded solving a large, ill-conditioned linear problem $Au = f$ for

- solving a much smaller, better conditioned linear problem $\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$, and
- solving a small non-linear system (for the shape σ).

But it comes with a catch

1. Choose a shape σ.
2. Solve coarse problem for ρ:
$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$
3. Calculate objective/residual:
$$r(\sigma) = f - AU_\sigma \rho$$
4. Calculate residual change (Newton step):
$$\delta r = -P_{\mathrm{tan}} r$$
5. Impute new shape from updated residual $r + \delta r$.
6. Repeat as necessary.

But it comes with a catch

1. Choose a shape σ.
2. Solve coarse problem for ρ:

$$(U_\sigma^T A U_\sigma)\rho = U_\sigma^T f$$

3. Calculate objective/residual:

$$r(\sigma) = f - AU_\sigma\rho$$

4. Calculate residual change (Newton step):

$$\delta r = -P_{\tan} r$$

5. Impute new shape from updated residual $r + \delta r$.
6. Repeat as necessary.

But it comes with a catch

1. Choose a shape σ.
2. Solve coarse problem for ρ:

$$\left(U_\sigma^T A U_\sigma\right)\rho = U_\sigma^T f$$

3. Calculate objective/residual:

$$r(\sigma) = f - AU_\sigma\rho$$

4. Calculate residual change (Newton step):

$$\delta r = -P_{\tan} r$$

5. Impute new shape from updated residual $r + \delta r$.
6. Repeat as necessary.

We need an error estimate.

So use it as an accelerator

1. Choose a shape σ.
2. Solve coarse problem for ρ:

$$(U_\sigma^T A U_\sigma)\rho = U_\sigma^T f$$

3. Calculate objective/residual:

$$r(\sigma) = f - AU_\sigma \rho$$

4. Calculate residual change (Newton step):

$$\delta r = -P_{\tan} r$$

5. Impute new shape from updated residual $r + \delta r$.
6. Repeat as necessary.

We need an error estimate so use our algorithm as an accelerator for that error estimating procedure.

Outline

- Quick review of linear algebra
- Application of interest
- Some empirical results
- Future research directions

Problem

Steady single-phase flow through a porous medium can be described by:

$$-\nabla \cdot a\nabla p = f$$

Solving this sort of problem is at the heart of more sophisticated models.

Problem

Steady single-phase flow through a porous medium can be described by:

$$-\nabla \cdot a \nabla p = f$$

Solving this sort of problem is at the heart of more sophisticated models.

- Time-dependent, multiphase, non-linear flow
- Well optimization
- Uncertainty in coefficients

All these require repeated solves of problems of the type above.

Problem

Steady single-phase flow through a porous medium can be described by:

$$-\nabla \cdot a\nabla p = f$$

Solving this sort of problem is at the heart of more sophisticated models.

This PDE can be discretized in a number of ways. For simplicity we will focus on applying 2D piecewise linear finite elements on triangles.

Challenges

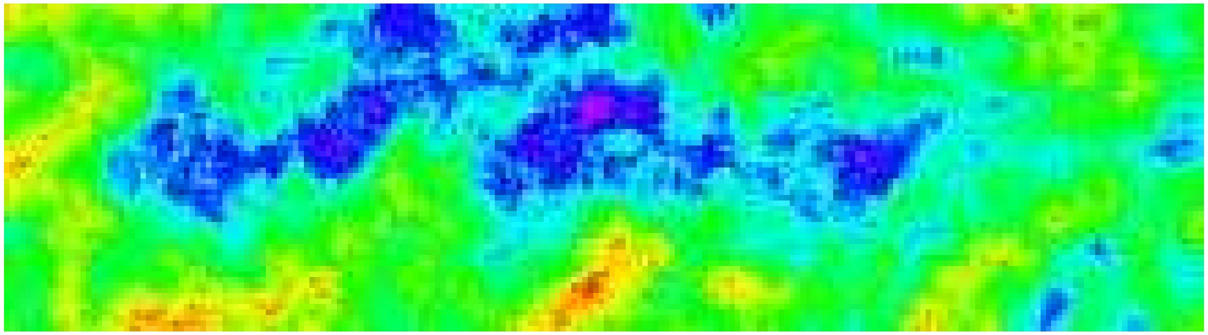
$$-\nabla \cdot a \nabla p = f$$

The coefficient a depends on the permeability.

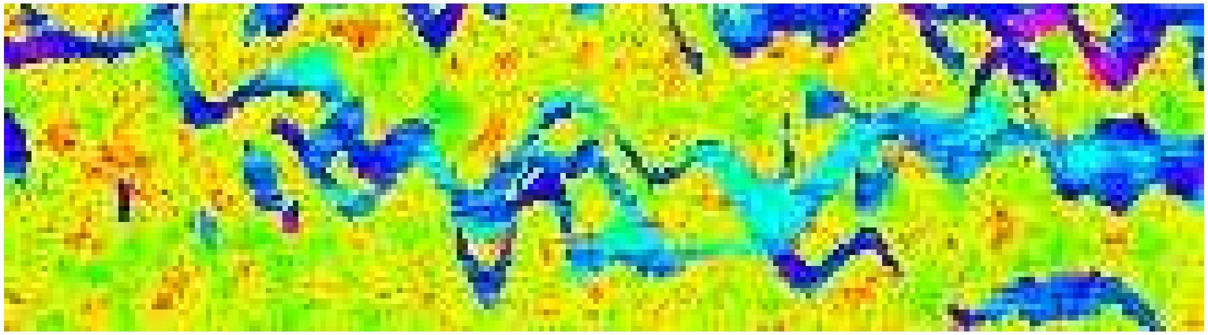
The permeability is often geostatistically generated at high resolution. It can be very heterogeneous.

Together these conditions make for an ill-conditioned and computationally expensive problem.

What's "heterogeneous"?



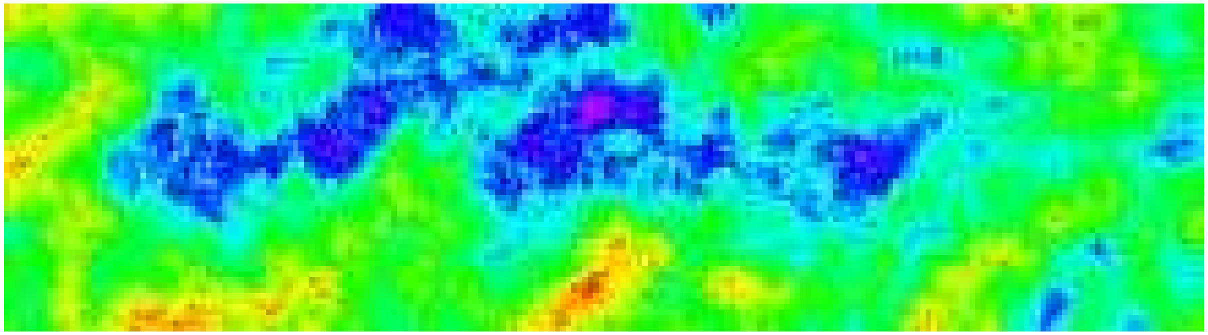
Top 35 slices simulate a Tarbert formation, a prograding near shore environment



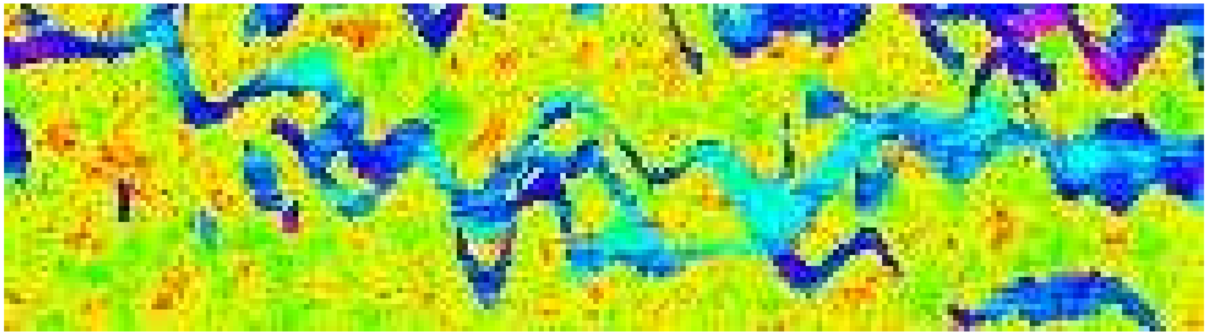
Lower 50 slices simulate an Upper Ness, a fluvial environment

Simulated field from the SPE CSP10

Why's "heterogeneity" important?



Top 35 slices simulate a Tarbert formation, a prograding near shore environment



Lower 50 slices simulate an Upper Ness, a fluvial environment

Small scale details can have a big impact on predictions that rely on the flow.

Goal

Calculate the approximation at the full resolution of the problem capturing all the details of the flow.

Method

Calculate the approximation at the full resolution of the problem capturing all the details of the flow.

We propose a new iterative method for solving the problem. The principle per iteration costs are only a coarse problem solve and a fine-scale residual evaluation. The principle start-up costs are a static condensation of subgrid DOFs into the coarse problem, and a coarse solve.

Teaser

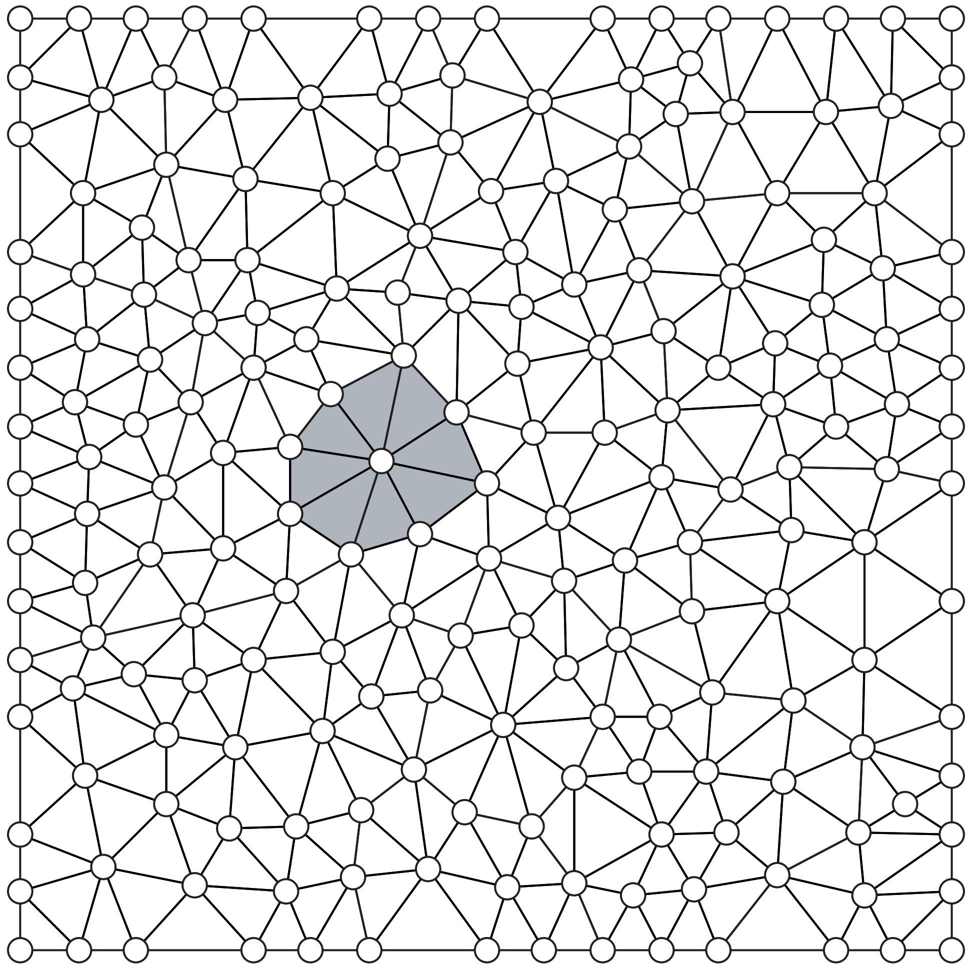
Calculate the approximation at the full resolution of the problem capturing all the details of the flow.

We propose a new iterative method for solving the problem. The principle per iteration costs are only a coarse problem solve and a fine-scale residual evaluation. The principle start-up costs are a static condensation of subgrid DOFs into the coarse problem, and a coarse solve.

As a stand-alone method, it has a number of unusual features:

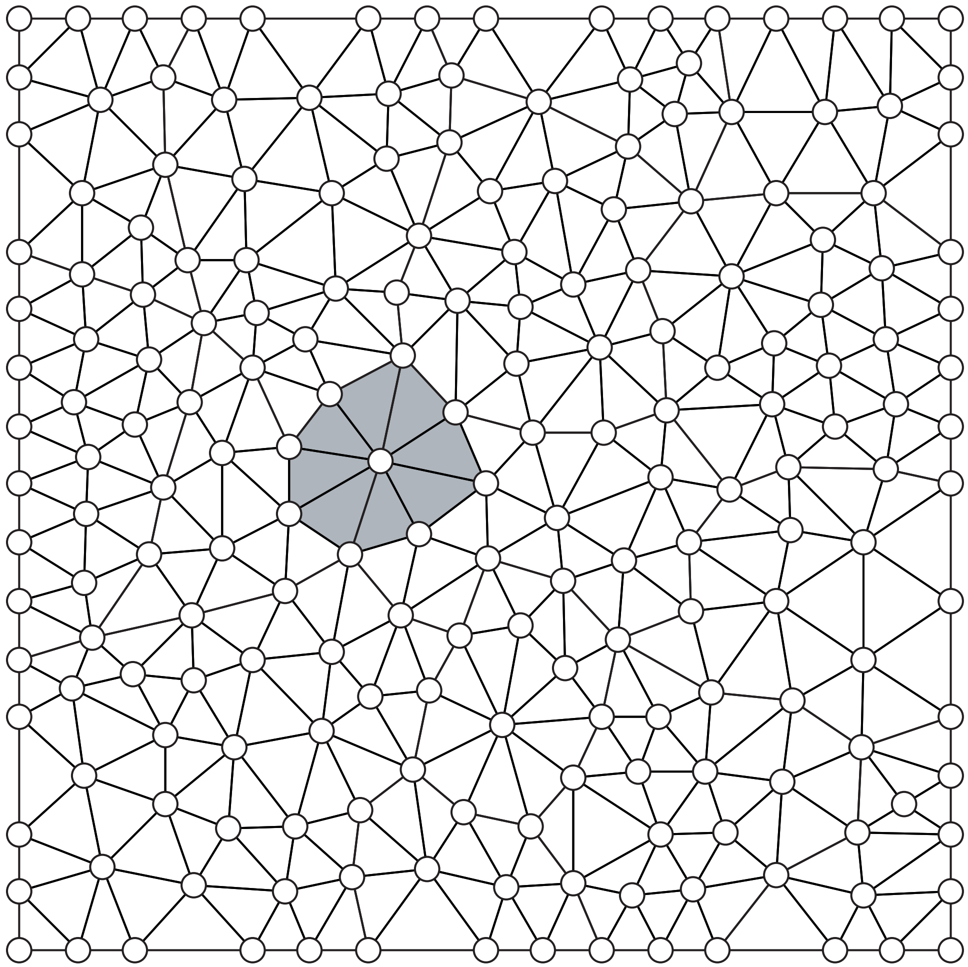
- number of iterations appears insenstive to fine-scale resolution (mesh size)
- number of iterations appears insensitive to heterogeneity (the a in $-\nabla \cdot a \nabla p = f$)
- provable global, monotone, asymptotically quadratic convergence

Fine-scale degrees of freedom



Let V be piecewise linear functions on a fine mesh.
Degrees of freedom (DOFs) are shown above.

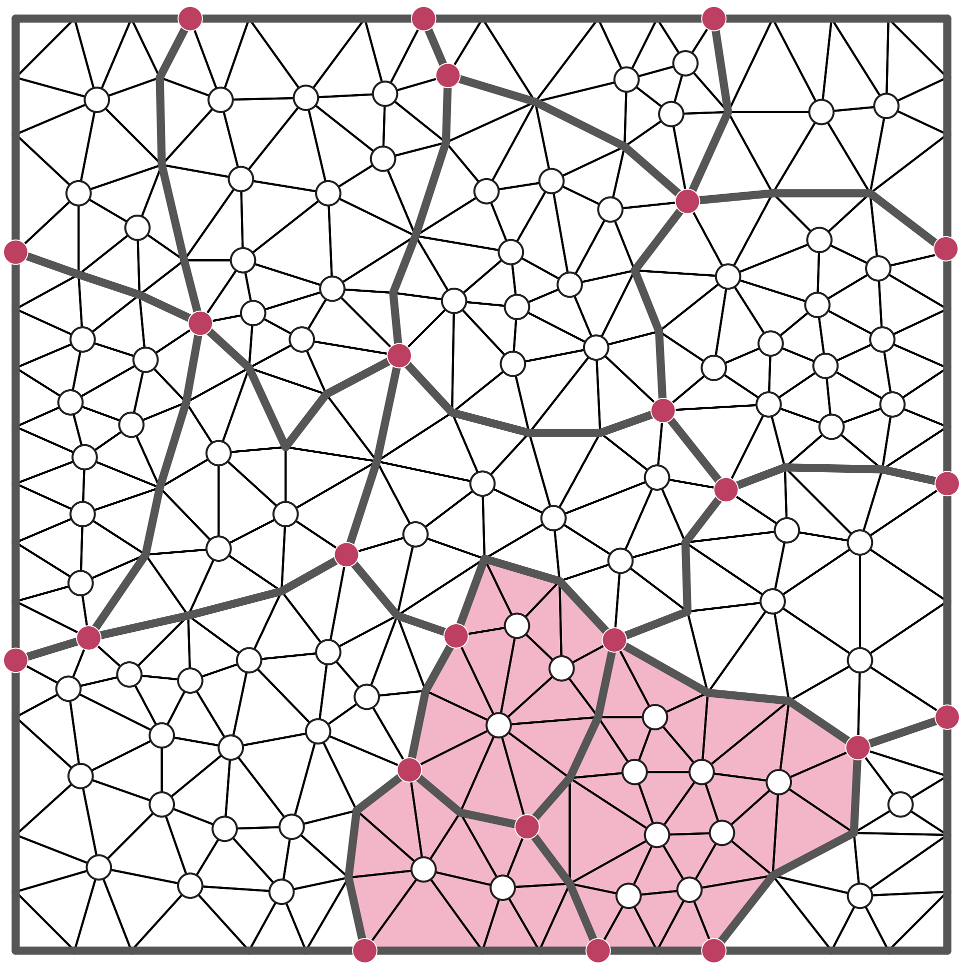
Goal



Solve $Ap = f$

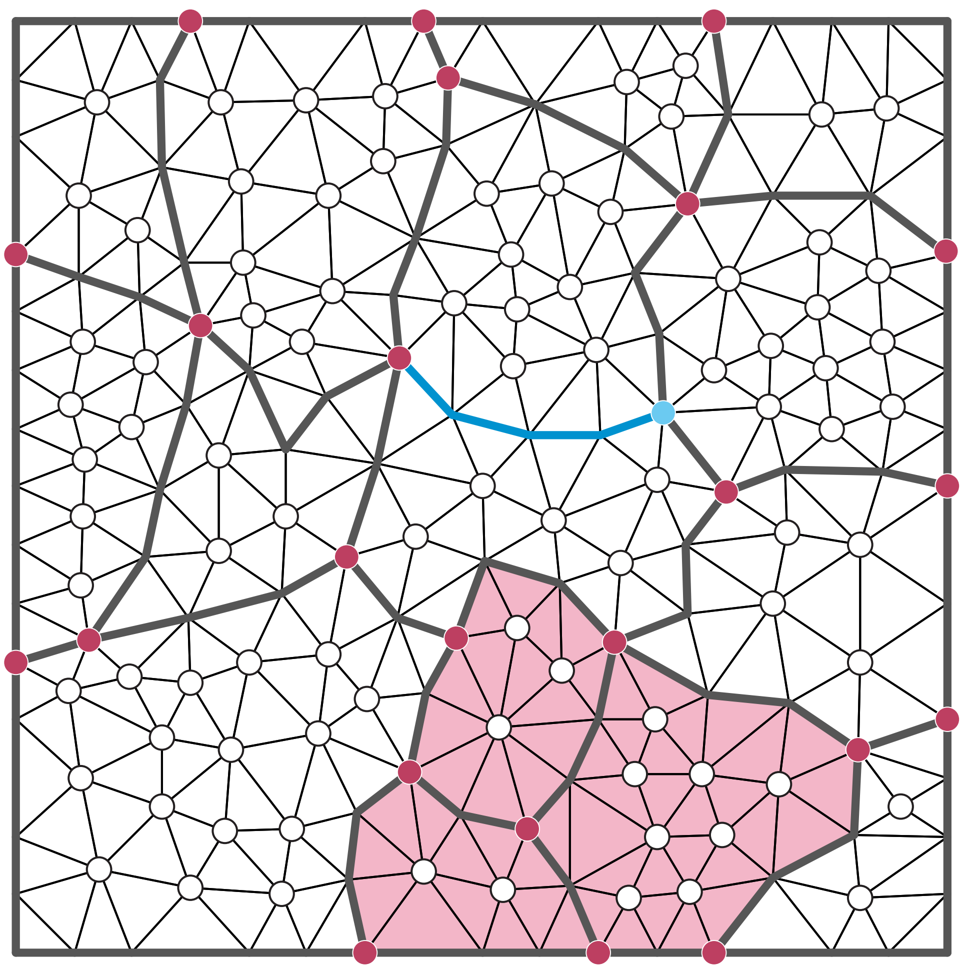
with pressure $p \in V$, data $f \in V'$, and matrix $A : V \to V'$.

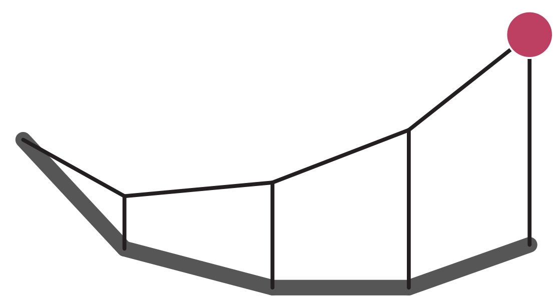
Multiscale degrees of freedom



From V, take out coarse edge DOFs to get V_H.

Multiscale degrees of freedom: corner shape





From V, take out coarse edge DOFs to get V_H.
Fix shapes for corner DOFs using the usual multiscale basis shapes.

Algebra of the multiscale problem

Solve $A_H p_H = f_H$ for $p_H \in V_H$ where

- $I_H : V_H \to V$ is the natural inclusion,
- $A_H = I_H^T A I_H$ is the coarsened matrix,
- and $f_H = I_H^T f$ is the coarsened data.

Note that these follow from the Galerkin procedure applied to $V_H \subset V$.

Multiscale solution quality

Solve $A_H p_H = f_H$ for $p_H \in V_H$ where

- $I_H : V_H \to V$ is the natural inclusion,
- $A_H = I_H^T A I_H$ is the coarsened matrix,
- and $f_H = I_H^T f$ is the coarsened data.

Note that these follow from the Galerkin procedure applied to $V_H \subset V$.

The multiscale solution p_H is a pretty good approximation for p: we use almost all the same DOFs and just take out a few. (And multiscale problems are just as easy to solve as coarse ones.)

Oops

Solve $A_H p_H = f_H$ for $p_H \in V_H$ where

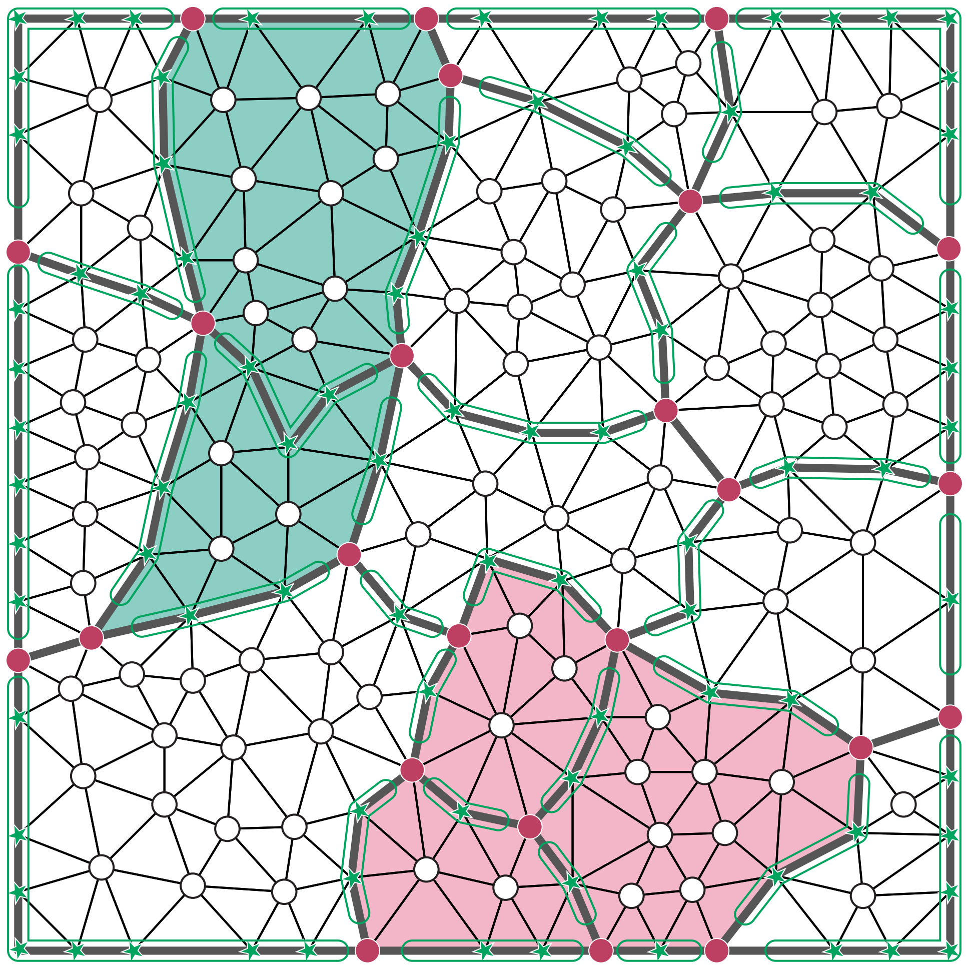
- $I_H : V_H \to V$ is the natural inclusion,
- $A_H = I_H^T A I_H$ is the coarsened matrix,
- and $f_H = I_H^T f$ is the coarsened data.

Note that these follow from the Galerkin procedure applied to $V_H \subset V$.

The multiscale solution p_H is a pretty good approximation for p: we use almost all the same DOFs and just take out a few. (And multiscale problems are just as easy to solve as coarse ones.)

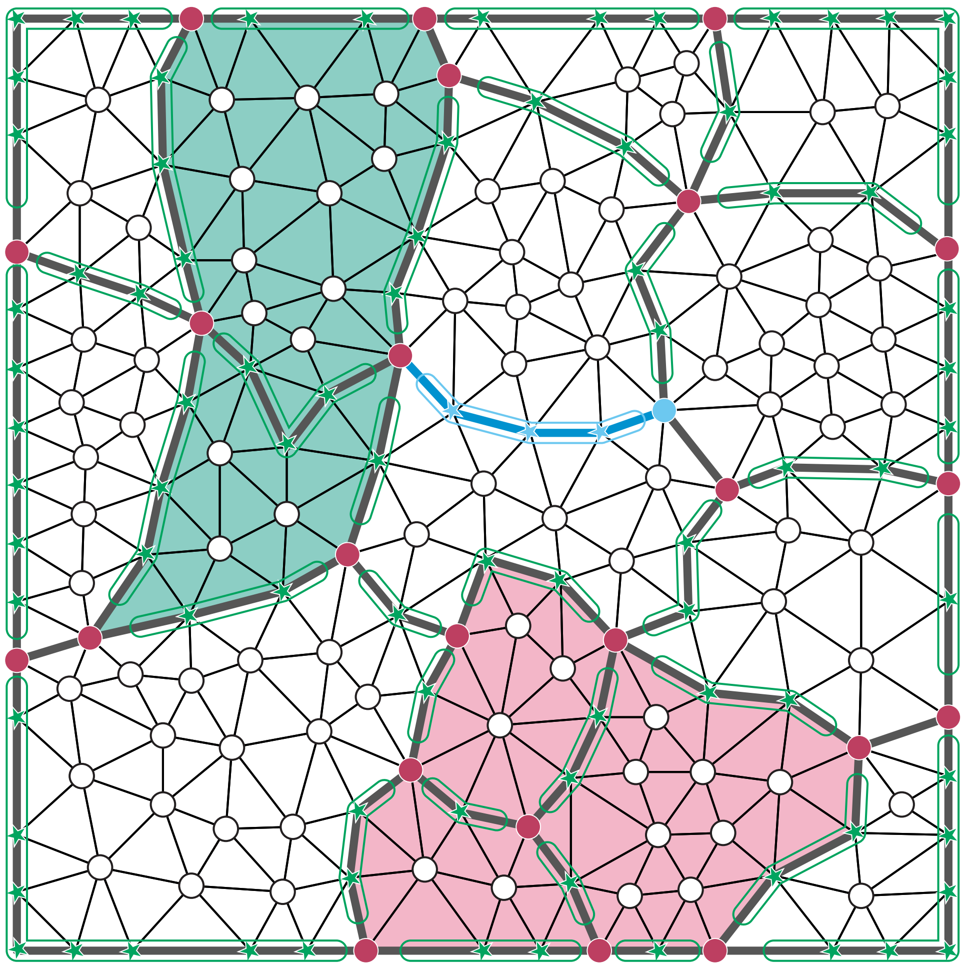
But almost always $p \neq p_H$, and — even worse — $p \notin V_H$. That is, we couldn't possibly get p as the result of a multiscale problem no matter how hard we try; we're missing some degrees of freedom.

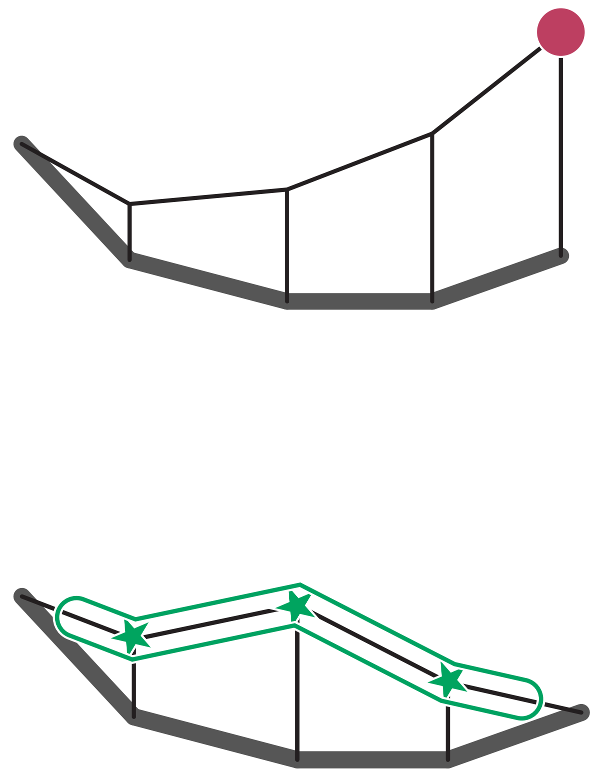
Supplemented multiscale degrees of freedom



From V_H, add back in some edge shapes to form V_β.

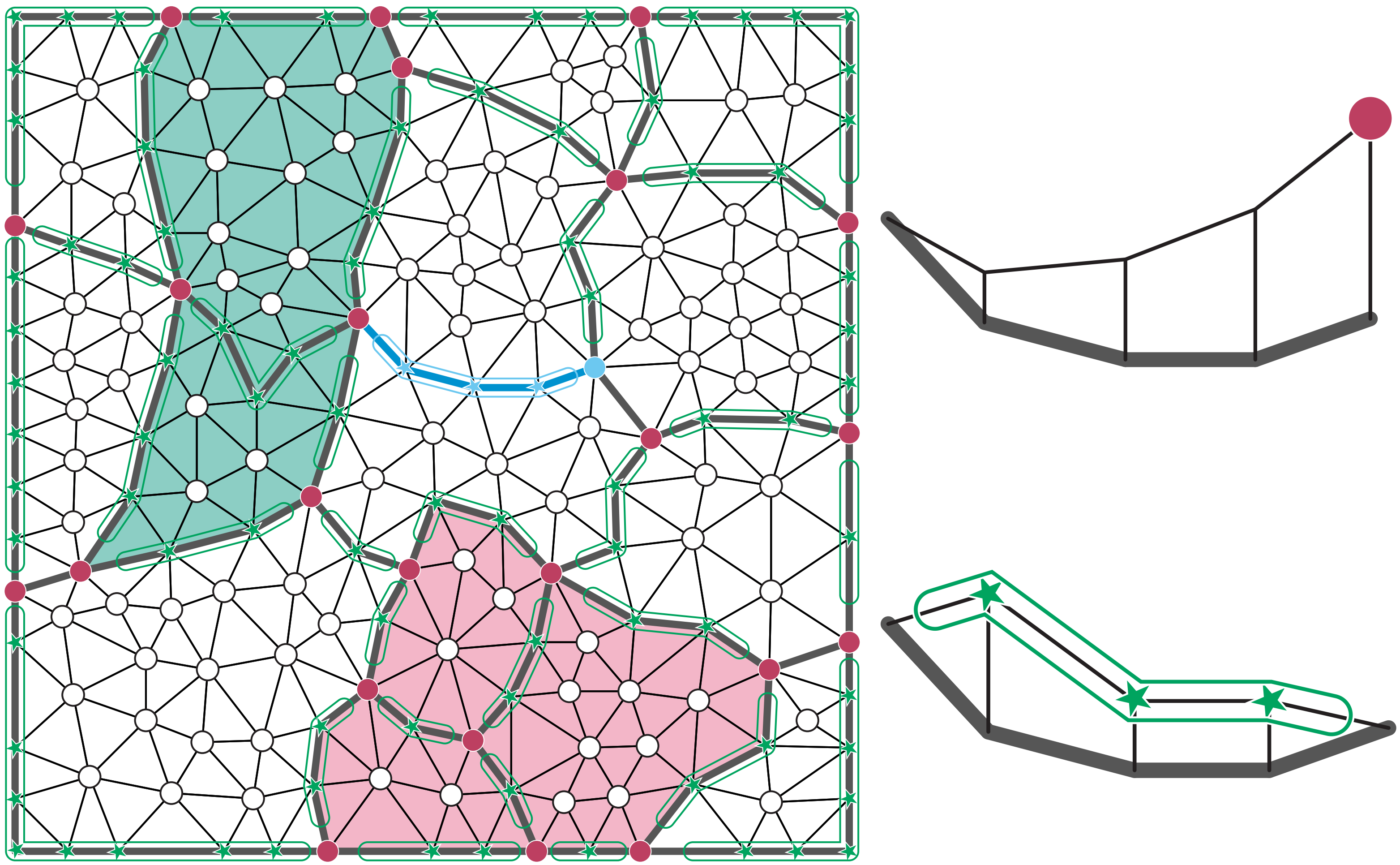
Supplemented multiscale degrees of freedom: edge shape





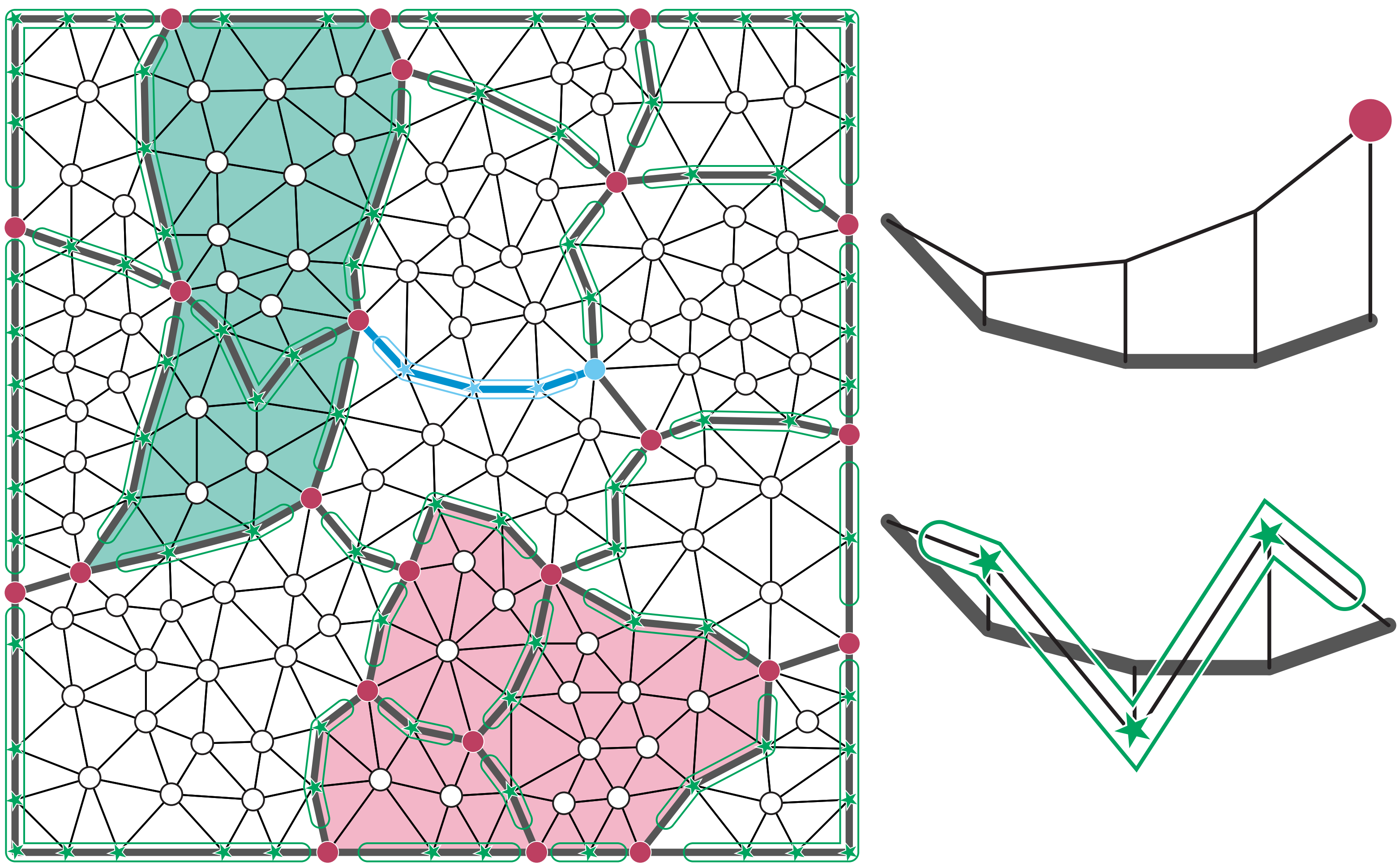
From V_H, add back in some edge shapes to form V_β.
Fix shapes along each coarse edge.

Supplemented multiscale degrees of freedom: another edge shape



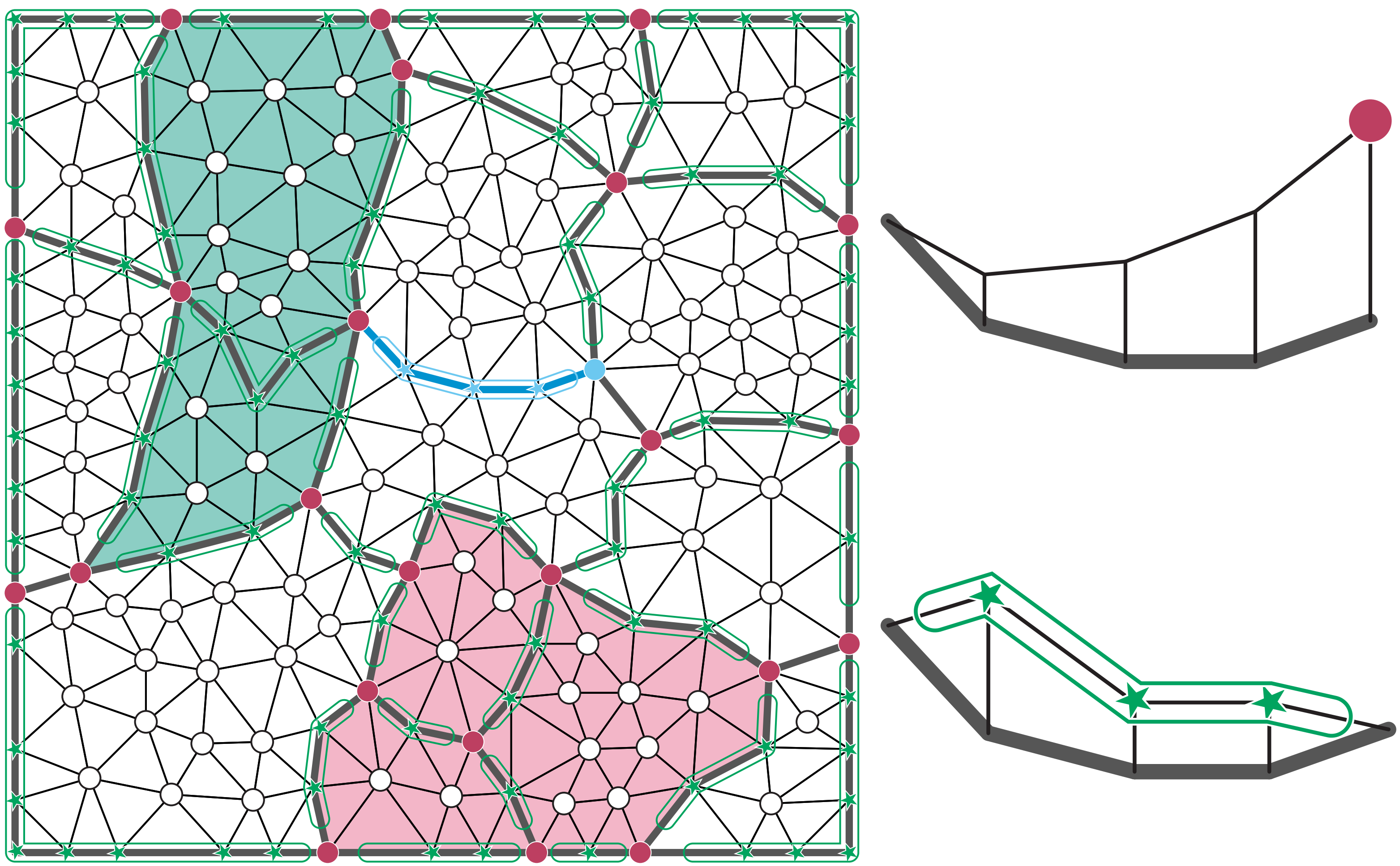
From V_H, add back in some edge shapes to form V_β.
Fix shapes along each coarse edge. Pick any shape you like ...

Supplemented multiscale degrees of freedom: another edge shape



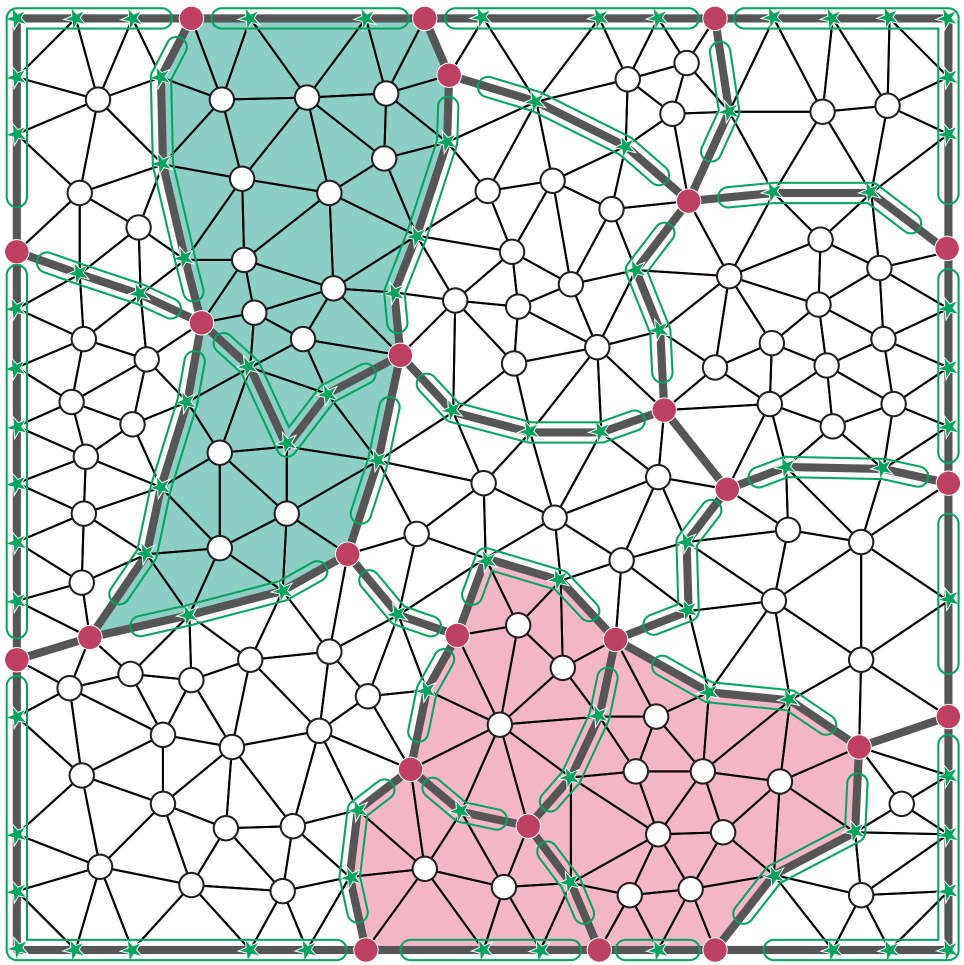
From V_H, add back in some edge shapes to form V_β.
Fix shapes along each coarse edge. Pick any shape you like ...

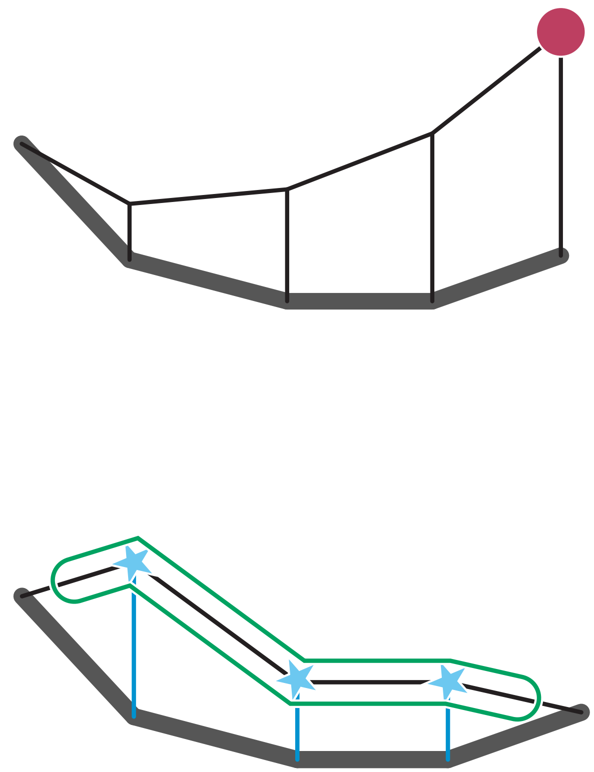
Supplemented multiscale degrees of freedom: another edge shape



From V_H, add back in some edge shapes to form V_β.
Fix shapes along each coarse edge. Pick any shape you like ...
But just pick one (for each coarse edge) for any given computation.

Supplemented multiscale degrees of freedom: parameterized family





From V_H, add back in some edge shapes to form V_β
Fix shapes along each coarse edge. Pick any shape you like ...
Record the heights (of the shapes along coarse edges) in a list β.

Supplemented multiscale problem

As before, solve $A_\beta p_\beta = f_\beta$ for $p_\beta \in V_\beta$ with

- $I_\beta : V_\beta \longrightarrow V$ as the natural inclusion,
- $A_\beta = I_\beta^T A I_\beta$ as the coarsened matrix,
- and $f_\beta = I_\beta^T f$ as the coarsened data.

As before, solve $A_\beta p_\beta = f_\beta$ for $p_\beta \in V_\beta$ with

- $I_\beta : V_\beta \longrightarrow V$ as the natural inclusion,
- $A_\beta = I_\beta^T A I_\beta$ as the coarsened matrix,
- and $f_\beta = I_\beta^T f$ as the coarsened data.

As before, usually $p \neq p_\beta$ and $p \notin V_\beta$. That is, for any particular β we're still missing some degrees of freedom.

But at least now $V = \bigcup_\beta V_\beta$.

By adjusting β we can find a V_β that accomodates p.

As before, solve $A_\beta p_\beta = f_\beta$ for $p_\beta \in V_\beta$ with

- $I_\beta : V_\beta \longrightarrow V$ as the natural inclusion,
- $A_\beta = I_\beta^T A I_\beta$ as the coarsened matrix,
- and $f_\beta = I_\beta^T f$ as the coarsened data.

As before, usually $p \neq p_\beta$ and $p \notin V_\beta$. That is, for any particular β we're still missing some degrees of freedom.

But at least now $V = \bigcup_\beta V_\beta$.

By adjusting β we can find a V_β that accomodates p.

This shares features with deflation and operator-based interpolation, but here we will vary our basis — change the inclusion I_β.

As before, solve $A_\beta p_\beta = f_\beta$ for $p_\beta \in V_\beta$ with

- $I_\beta : V_\beta \longrightarrow V$ as the natural inclusion,
- $A_\beta = I_\beta^T A I_\beta$ as the coarsened matrix,
- and $f_\beta = I_\beta^T f$ as the coarsened data.

As before, usually $p \neq p_\beta$ and $p \notin V_\beta$. That is, for any particular β we're still missing some degrees of freedom.

But at least now $V = \bigcup_\beta V_\beta$.

By adjusting β we can find a V_β that accomodates p.

We trade solving a linear problem for solving a non-linear one. But this is sensible because we trade a large linear system for a smaller, better-conditioned linear system along with a small non-linear one.

An algorithm

- Use feedback from the fine-scale residual:

$$r_\beta = f - AI_\beta p_\beta$$

to adjust the shapes.

An algorithm

- Use feedback from the fine-scale residual:

$$r_\beta = f - AI_\beta p_\beta$$

to adjust the shapes.

- We want to find shapes β so that $r_\beta = 0$.

An algorithm

- Use feedback from the fine-scale residual:

$$r_\beta = f - AI_\beta p_\beta$$

to adjust the shapes.

- We want to find shapes β so that $r_\beta = 0$.

Use Newton's method to tell us how to adjust the shapes.

An algorithm

- Use feedback from the fine-scale residual:
$$r_\beta = f - AI_\beta p_\beta$$
to adjust the shapes.
- We want to find shapes β so that $r_\beta = 0$.
Use Newton's method to tell us how to adjust the shapes.
- Newton's method requires computing an expensive Jacobian.

An algorithm

- Use feedback from the fine-scale residual:
$$r_\beta = f - AI_\beta p_\beta$$
to adjust the shapes.

- We want to find shapes β so that $r_\beta = 0$.
Use Newton's method to tell us how to adjust the shapes.

- Newton's method requires computing an expensive Jacobian.

- We avoid this by using the specially structured geometry of our algebraic problem (symmetry and positive definiteness).

An algorithm?

- Use feedback from the fine-scale residual:
$$r_\beta = f - AI_\beta p_\beta$$
to adjust the shapes.
- We want to find shapes β so that $r_\beta = 0$.
Use Newton's method to tell us how to adjust the shapes.
- Newton's method requires computing an expensive Jacobian.
- We avoid this by using the specially structured geometry of our algebraic problem (symmetry and positive definiteness).

There's a catch: we require an externally provided error estimate.

An algorithm?

The catch: we need someone else to give us an error estimate at each iteration.

An accelerator!

The catch: we need someone else to give us an error estimate at each iteration.

We can use our algorithm as an accelerator for some other iterative procedure. The other procedure acts as an error estimator for us.

An accelerator!

The catch: we need someone else to give us an error estimate at each iteration.

We can use our algorithm as an accelerator for some other iterative procedure. The other procedure acts as an error estimator for us.

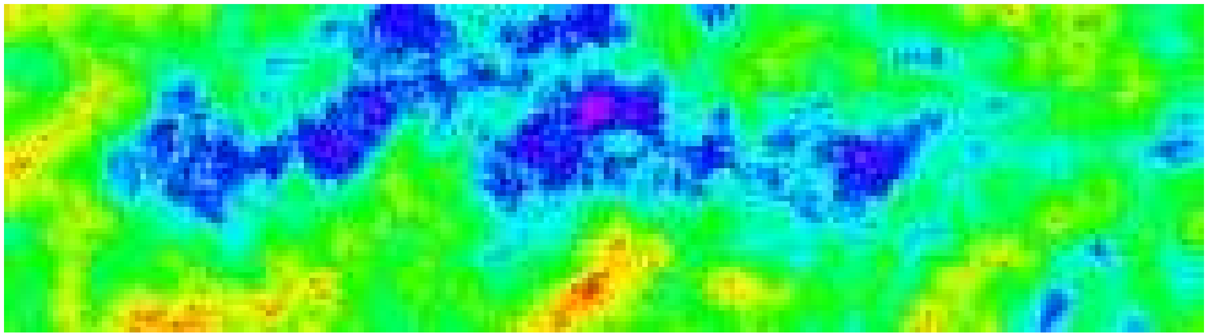
As a stand-alone method:

- global, monotone, asymptotically quadratic convergence
- number of iterations insensitive to resolution
- number of iterations insensitive to heterogeneity

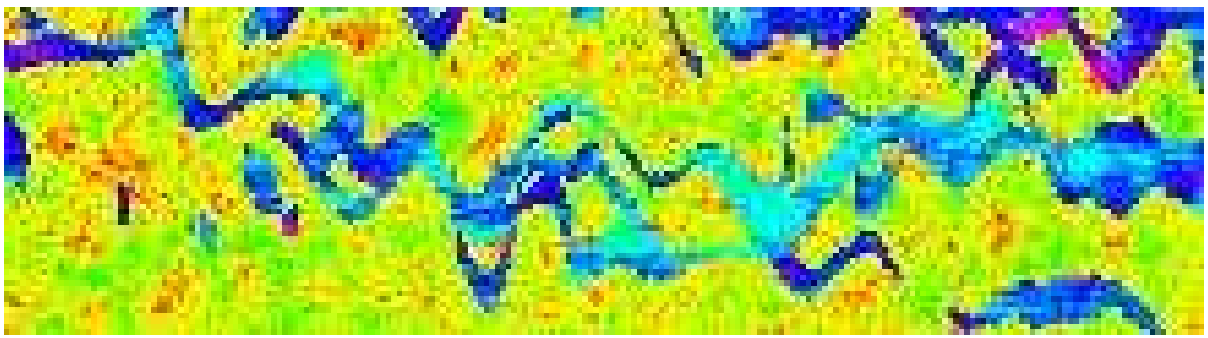
Outline

- Quick review of linear algebra
- Application of interest
- Some empirical results
- Future research directions

First example

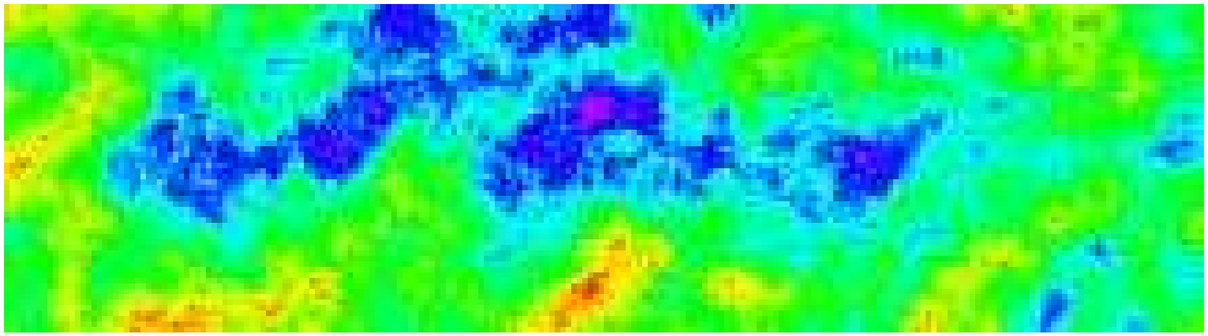


Top 35 slices simulate a Tarbert formation, a prograding near shore environment

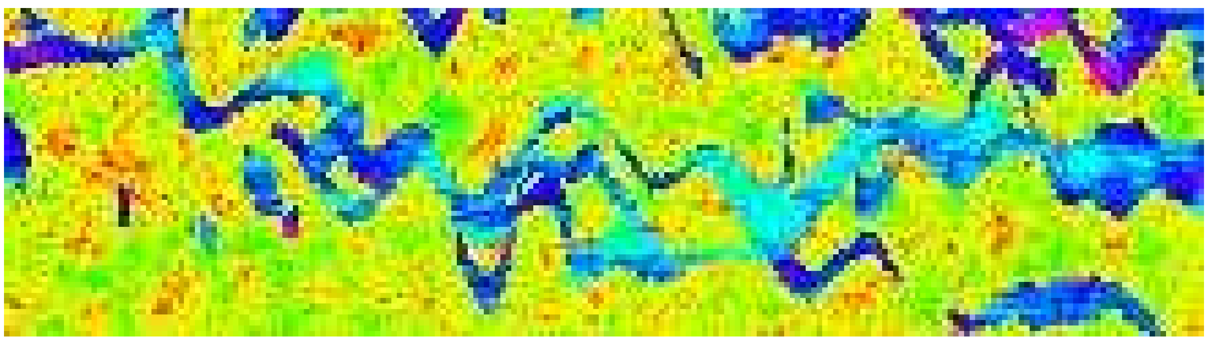


Lower 50 slices simulate an Upper Ness, a fluvial environment

Simulated field from the SPE CSP10



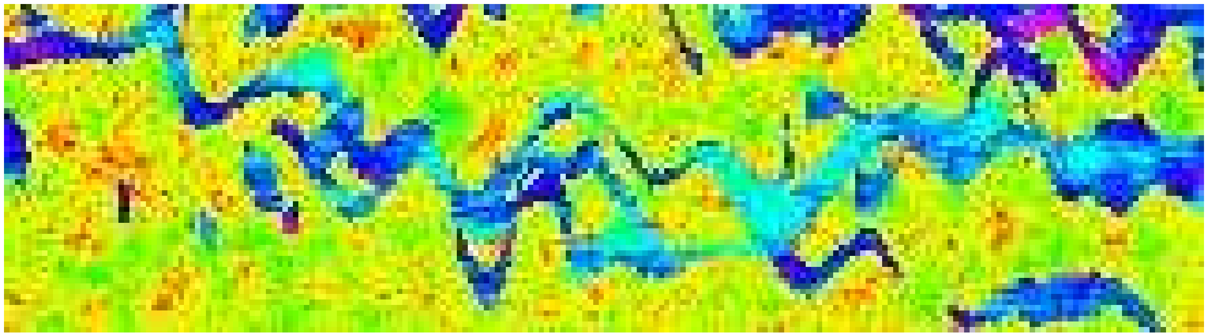
Top 35 slices simulate a Tarbert formation, a prograding near shore environment



Lower 50 slices simulate an Upper Ness, a fluvial environment

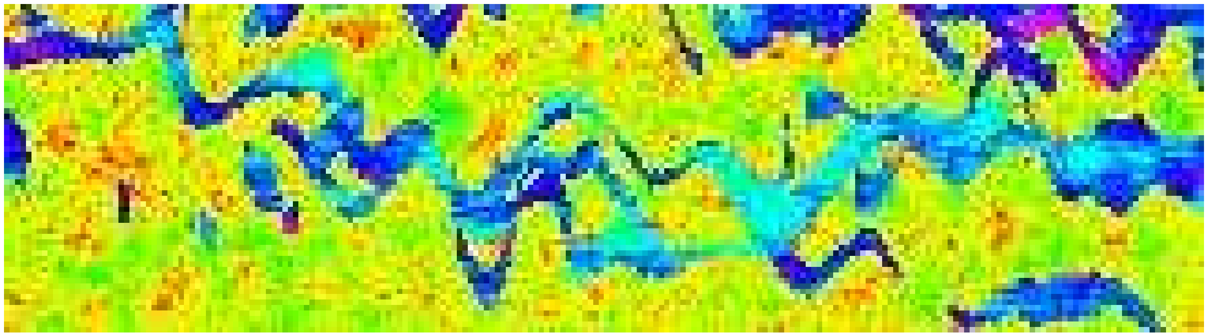
The flow from a pressure flood was computed for each slice. High pressure is imposed on the left and low on the right with no-flow conditions on the top and bottom.

Typical flow from a slice

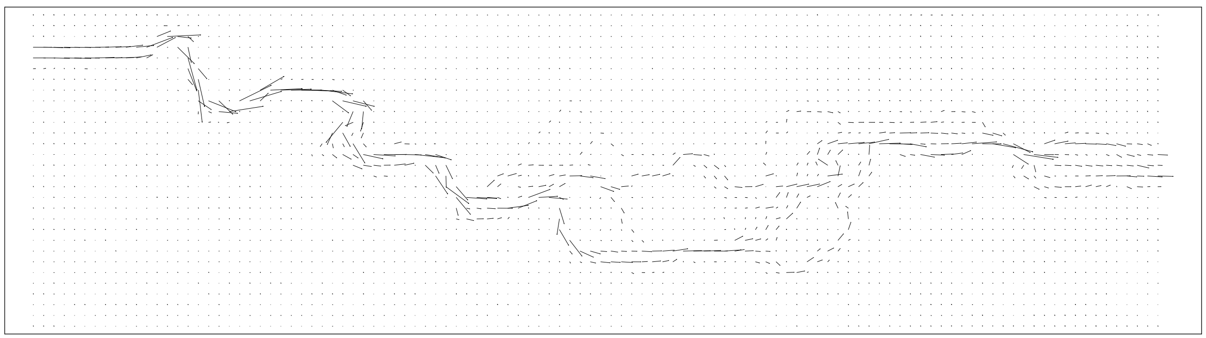


permeability (a)

Typical flow from a slice

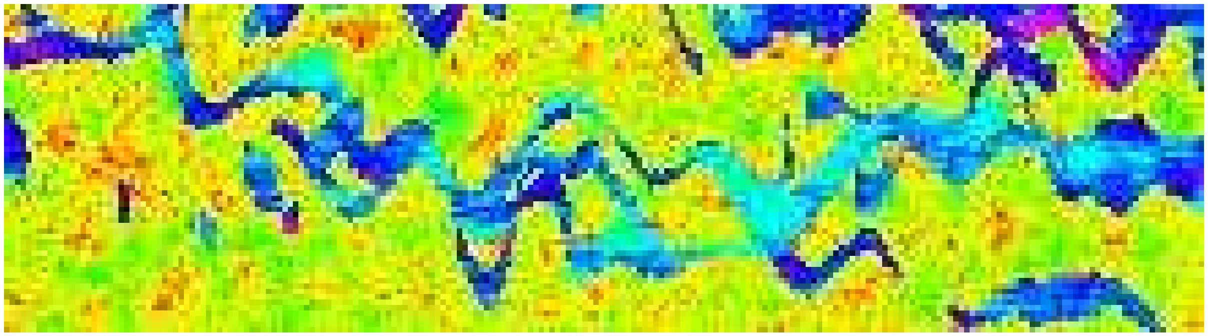


permeability (a)

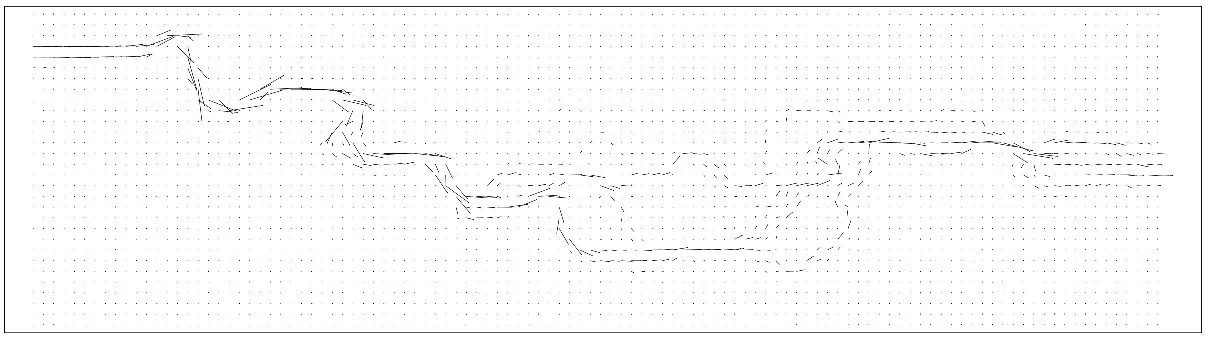


velocity ($-a\nabla p$)

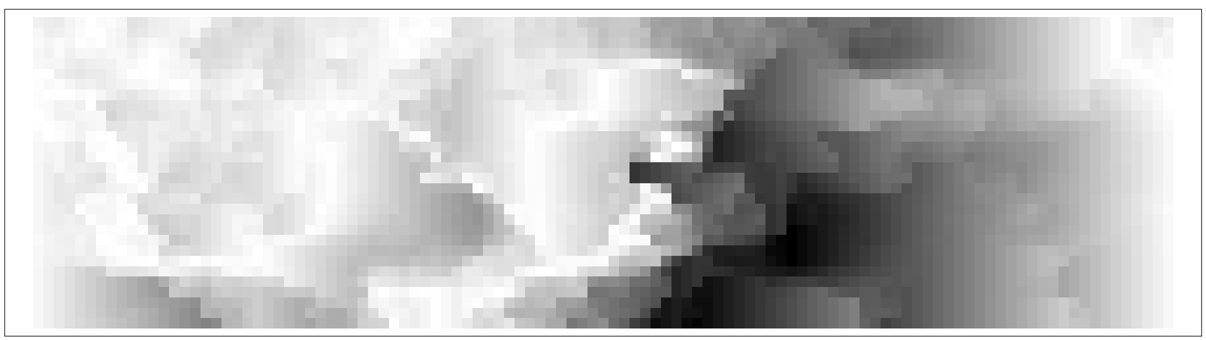
Typical flow from a slice



permeability (a)



velocity ($-a\nabla p$)



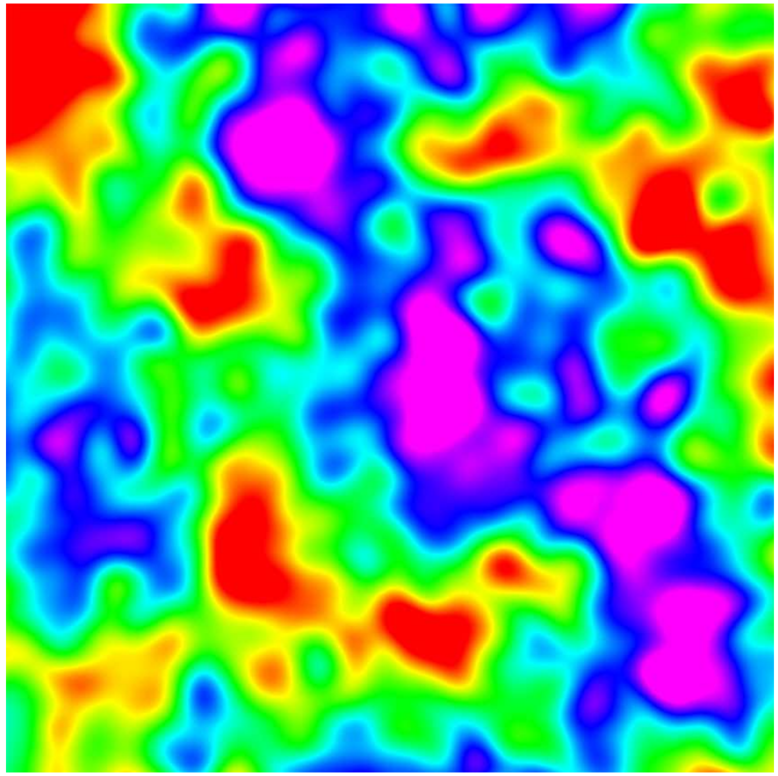
difference of pressure from uniform gradient ($p - p_D$)

Number of iterations

		# of Newton iterations				
		Min	Med	Max	Avg	Freq of Med
5×5 subgrid	All	4	5	6	5.26	73%
	Onshore	5	5	6	5.04	96%
	Fluvial	4	5	6	5.41	56%
10×10 subgrid	All	5	6	7	5.95	89%
	Onshore	5	6	7	5.93	89%
	Fluvial	5	6	7	5.97	89%

Only a few iterations are needed to get an accurate answer.

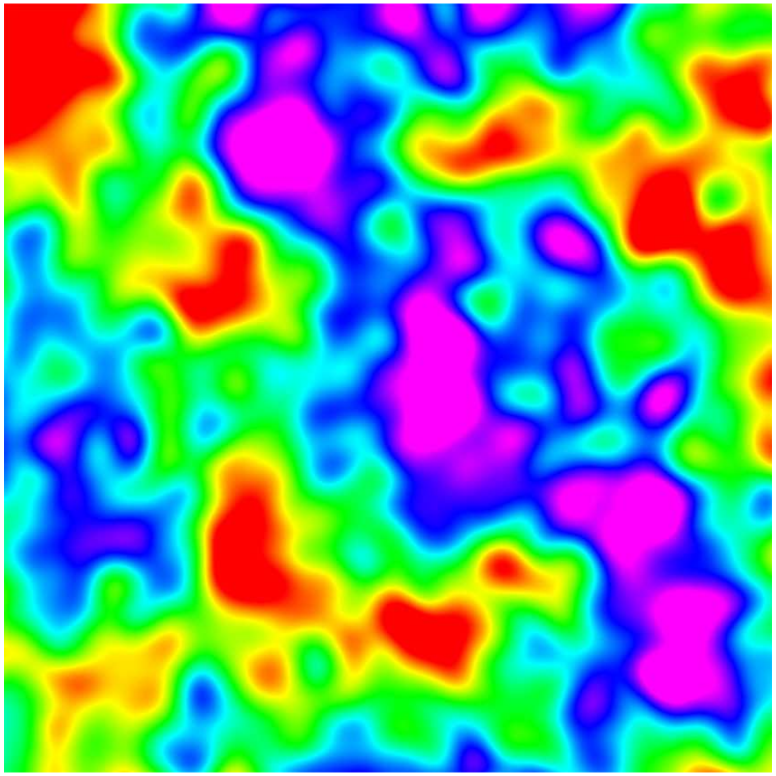
An artificial permeability field



The above graphic plots the variation from a statistically generated permeability field. Red areas indicate low permeability; blue areas indicate high permeability.

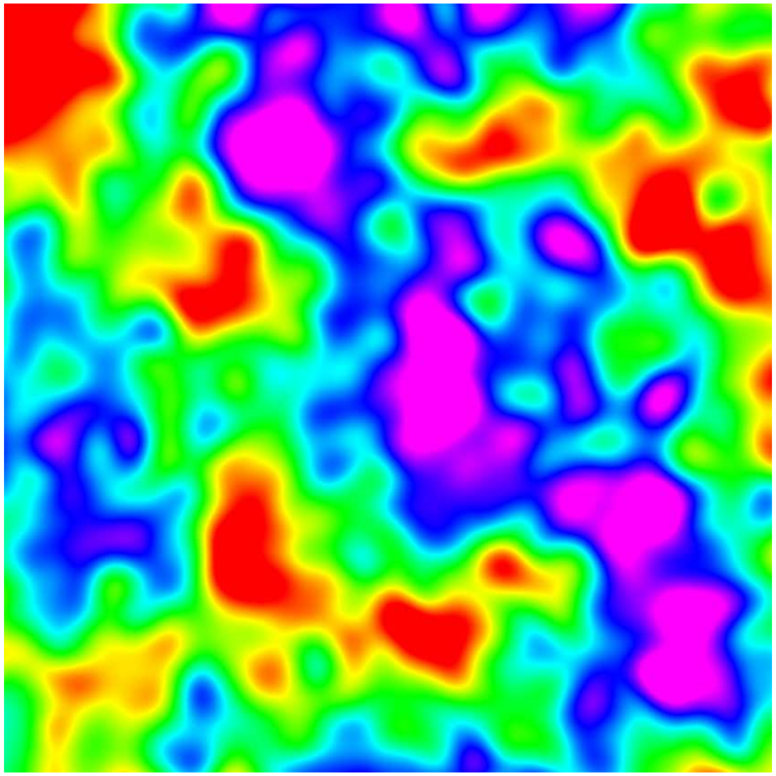
The permeabilities span about five orders of magnitude (10^5).

An artificial permeability field



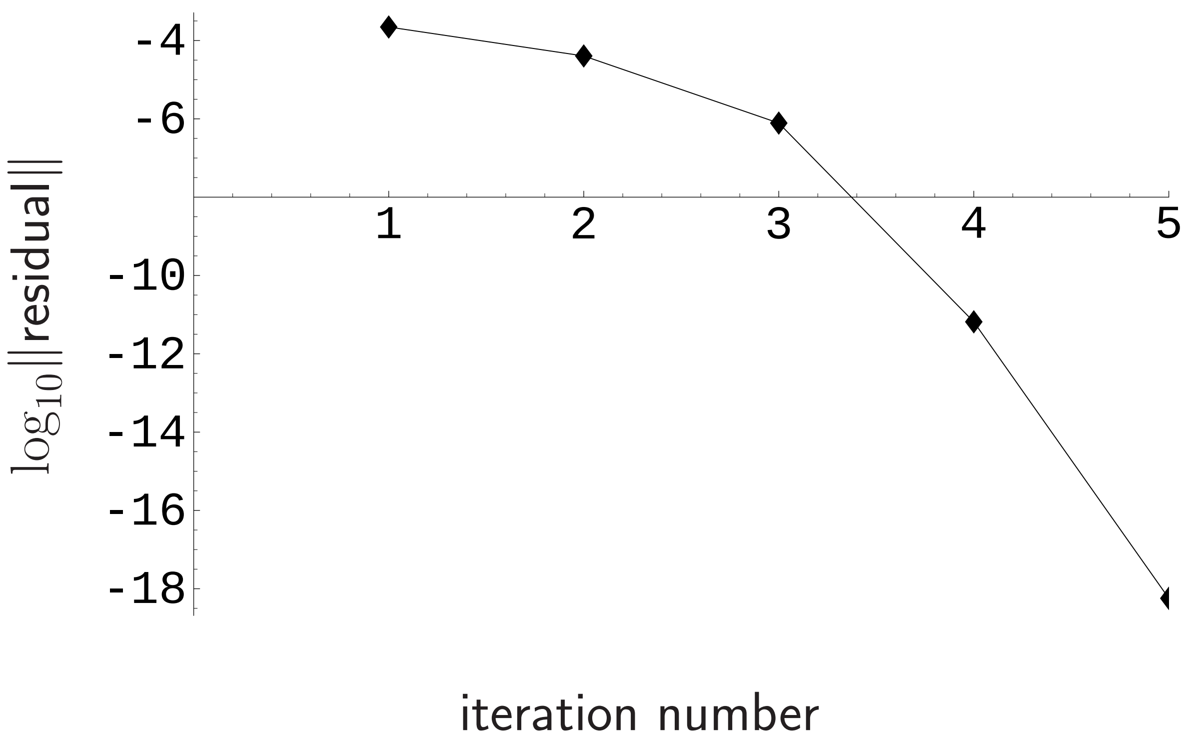
It was generated at high-resolution to be able to compare results between subsamples of various resolutions. It was also rescaled to produce fields of varying heterogeneity.

An artificial permeability field



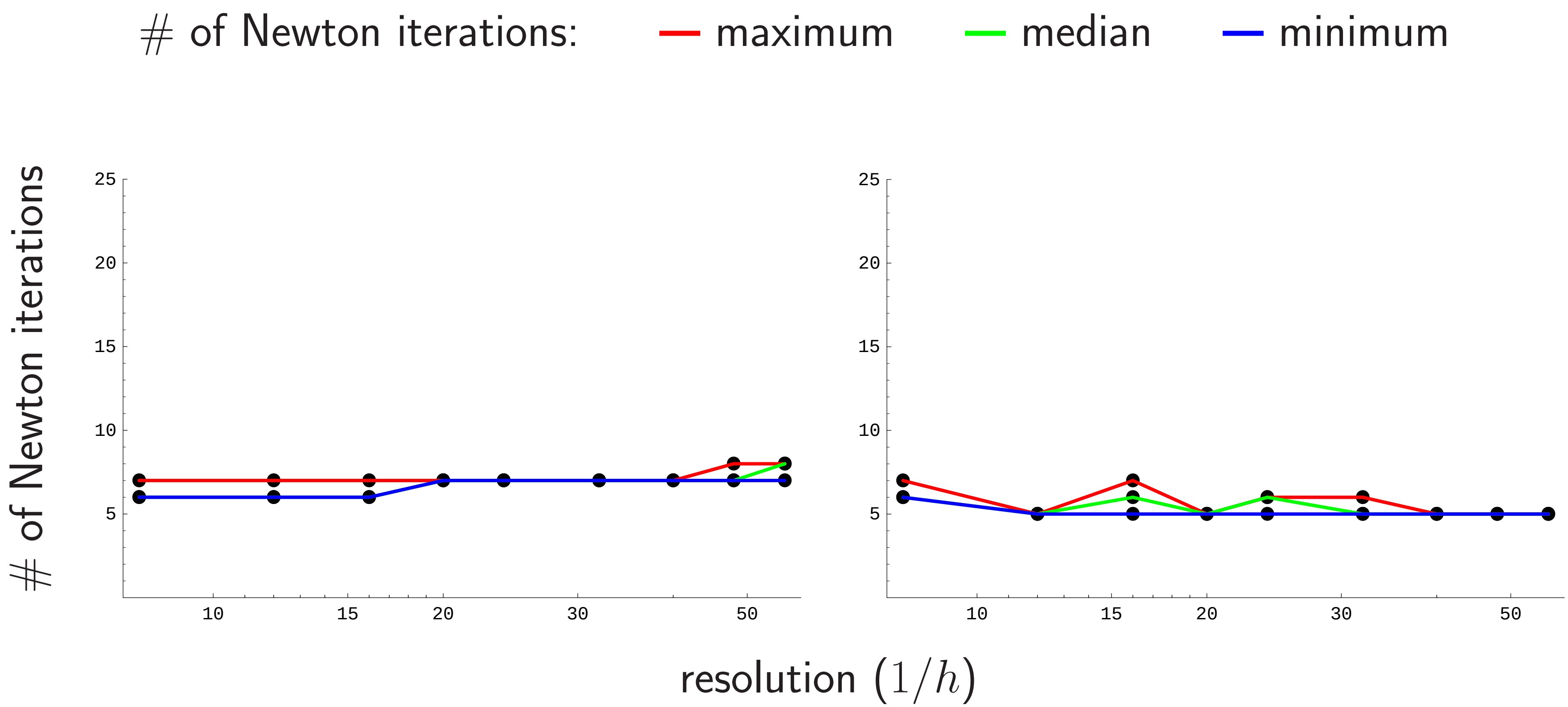
We solve a quarter five-spot-like problem with a source at the bottom-left and a sink at the top-right.

Typical convergence history



Note the axis scales: we get quadratic convergence — on a linear problem!
The convergence is monotone; no special initial shape was used.

Resolution independence

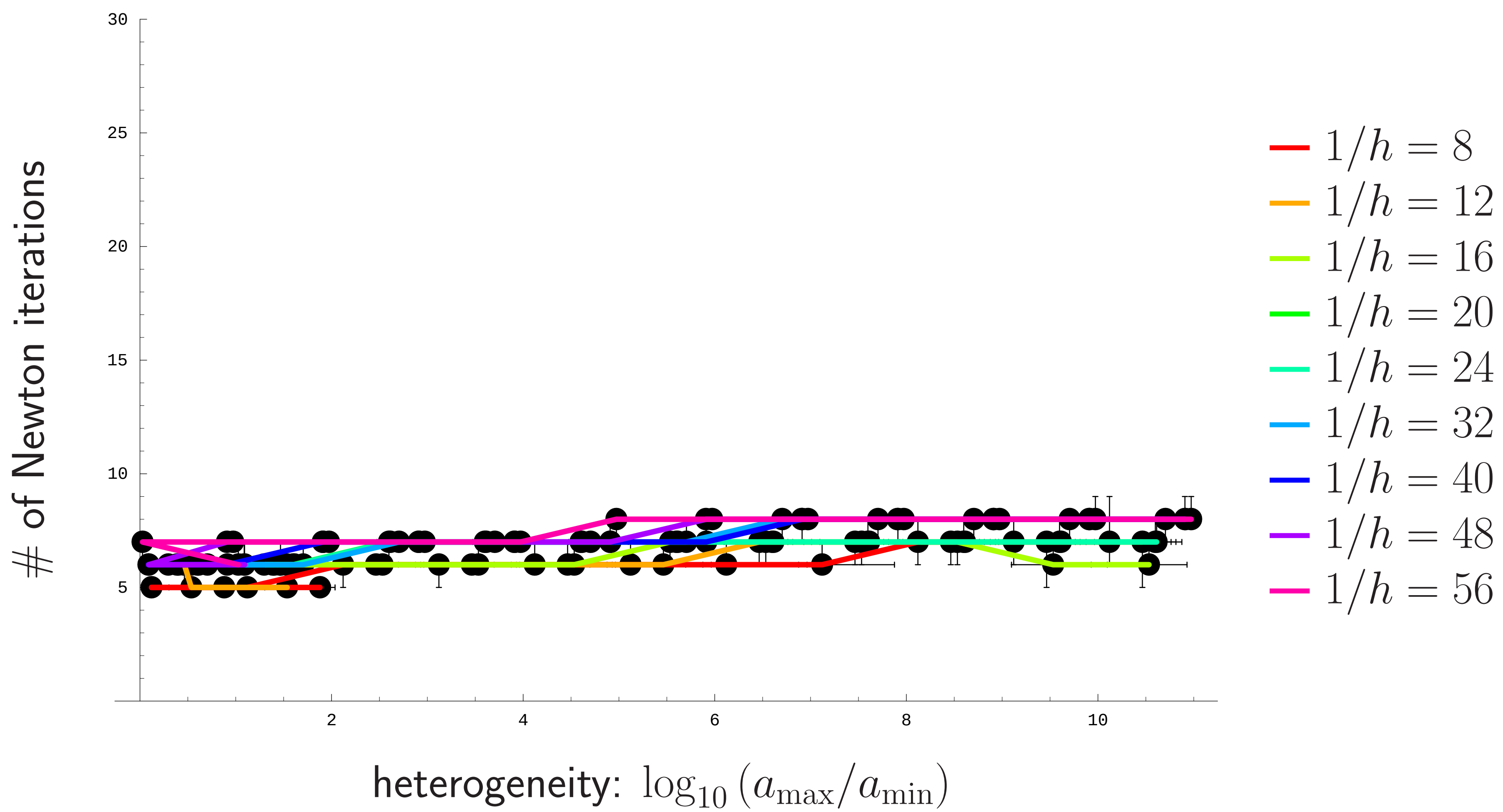


Fixed coarse grid

Fixed coarse/fine ratio

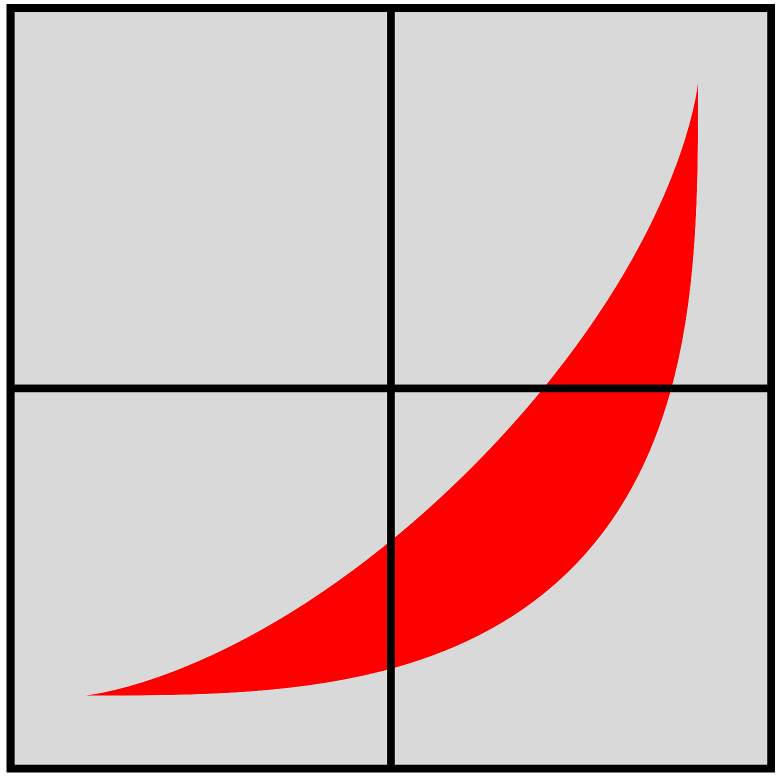
Let the grid get finer and finer (let the resolution increase). At each resolution, grab several statistical subsamples of the permeability field. Roughly a constant number of Newton iterations is needed.

Heterogeneity independence



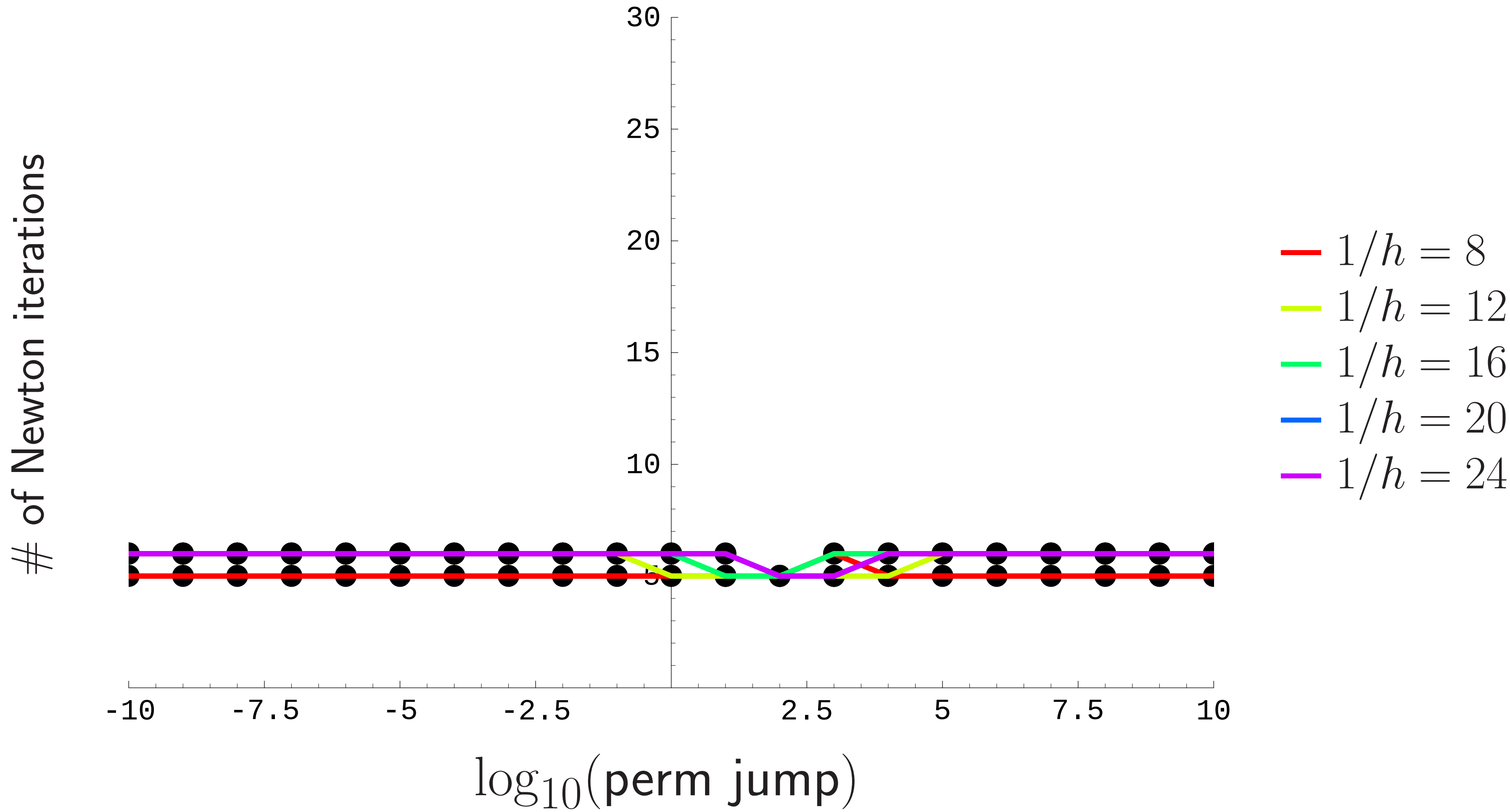
Take a subsample of the heterogeneous permeability field and rescale it so that $a_{\max}/a_{\min}$ gets large.

A channel/barrier permeability field



In the above diagram, gray represents a permeability of 1 and red represents either a high or low permeability. When high, we have a channel; when low, we have a barrier.

Heterogeneity insensitivity



The horizontal axis shows the base-10 log of the permeability of the barrier/channel. Points on the left are for a barrier; points in the center are constant permeability everywhere; points on the right are for a channel.

Outline

- Quick review of linear algebra
- Application of interest
- Some empirical results
- Future research directions

Onward and upward

- Investigate algorithm as an accelerator
- Proof of insensitivity to resolution and heterogeneity
- Recursion (multilevel method)
- Extensions to other problems