

# Numerical linear algebra in the Canadian hydrological model

## Part II: Iterative methods

Kevin R. Green

Department of Computer Science
University of Saskatchewan

Math 313
University of Saskatchewan
October 13, 2020

# Outline

# Brief review of last lecture

- Hydrological models consist of many sub-components.
  - Some involve spatial coupling via PDEs
- Discretizing PDEs on a mesh gives sparse linear algebra problems
- Time for sparse direct solves is sensitive to the sparsity structure

# Sparse linear algebra

Solving

$$Au = b$$

# Sparse linear algebra

Solving

$$Au = b$$

- $A$ - sparse matrix, discrete version of spatial influence

# Sparse linear algebra

Solving

$$Au = b$$

- $A$ - sparse matrix, discrete version of spatial influence
- $b$ - known vector, discrete version of $f(x, y)$

# Sparse linear algebra

Solving

$$Au = b$$

- $A$ - sparse matrix, discrete version of spatial influence
- $b$ - known vector, discrete version of $f(x, y)$
- $u$ - unknown vector, discrete version of $u(x, y)$

# Sparse linear algebra

Solving

$$Au = b$$

*There are many ways to do this too. Iterative methods!*

- $A$ - sparse matrix, discrete version of spatial influence
- $b$ - known vector, discrete version of $f(x, y)$
- $u$ - unknown vector, discrete version of $u(x, y)$

# Iterative methods

Rely on matrix-vector multiplications.

- Iterate until a desired tolerance is satisfied

$$|Au_k - b| < tol$$

*tol – user chosen tolerance*
$k$ – iteration number

# Iterative methods

Rely on matrix-vector multiplications.

- Iterate until a desired tolerance is satisfied

$$|Au_k - b| < tol$$

*tol* – *user chosen tolerance*
$k$ – iteration number

Methods:

- Splitting methods
  $A = M - N \rightarrow Mu_{k+1} = b + Nu_k$

# Iterative methods

Rely on matrix-vector multiplications.

- Iterate until a desired tolerance is satisfied

$$|Au_k - b| < tol$$

*tol* – *user chosen tolerance*
$k$ – iteration number

Methods:

- Splitting methods
  $A = M - N \rightarrow Mu_{k+1} = b + Nu_k$
  - Jacobi [Jacobi1845]

# Iterative methods

Rely on matrix-vector multiplications.

- Iterate until a desired tolerance is satisfied

$$|Au_k - b| < tol$$

*tol – user chosen tolerance*
$k$ – iteration number

Methods:

- Splitting methods
  $A = M - N \rightarrow Mu_{k+1} = b + Nu_k$
  - Jacobi [Jacobi1845]
  - SOR [Southwell1946]

# Iterative methods

Rely on matrix-vector multiplications.

- Iterate until a desired tolerance is satisfied

$$|Au_k - b| < tol$$

*tol* – *user chosen tolerance*
$k$ – iteration number

Methods:

- Splitting methods
  $A = M - N \rightarrow Mu_{k+1} = b + Nu_k$
  - Jacobi [Jacobi1845]
  - SOR [Southwell1946]
- Krylov methods
  $u_k \in \mathrm{span}\{b, Ab, A^2b, \ldots, A^{k-1}b\}$

# Iterative methods

Rely on matrix-vector multiplications.

- Iterate until a desired tolerance is satisfied

$$|Au_k - b| < tol$$

*tol* – *user chosen tolerance*
$k$ – iteration number

Methods:

- Splitting methods
  $A = M - N \rightarrow Mu_{k+1} = b + Nu_k$
  - Jacobi [Jacobi1845]
  - SOR [Southwell1946]
- Krylov methods
  $u_k \in \mathrm{span}\{b, Ab, A^2b, \ldots, A^{k-1}b\}$
  - Conjugate gradient [Hestenes1952]

# Iterative methods

Rely on matrix-vector multiplications.

- Iterate until a desired tolerance is satisfied

$$|Au_k - b| < tol$$

*tol* – *user chosen tolerance*
$k$ – iteration number

Methods:

- Splitting methods
  $A = M - N \rightarrow Mu_{k+1} = b + Nu_k$
  - Jacobi [Jacobi1845]
  - SOR [Southwell1946]
- Krylov methods
  $u_k \in \mathrm{span}\{b, Ab, A^2b, \ldots, A^{k-1}b\}$
  - Conjugate gradient [Hestenes1952]
  - GMRES [Saad1986]

# Pros and cons of iterative methods

Pros:

Cons:

# Pros and cons of iterative methods

Pros:

- Algorithms scale well ($\mathcal{O}(km)$, typically with $k \ll m$).

Cons:

# Pros and cons of iterative methods

Pros:

- Algorithms scale well ($\mathcal{O}(km)$, typically with $k \ll m$).

Cons:

- Generally can't re-use solutions.

# Pros and cons of iterative methods

Pros:

- Algorithms scale well ($\mathcal{O}(km)$, typically with $k \ll m$).
- Only solve as accurately as you need/want.

Cons:

- Generally can't re-use solutions.

# Pros and cons of iterative methods

Pros:

- Algorithms scale well ($\mathcal{O}(km)$, typically with $k \ll m$).
- Only solve as accurately as you need/want.

Cons:

- Generally can't re-use solutions.
- Spectral properties may not be known for a given matrix.

Pro/Con:

- Time to solution dependent on matrix *spectrum*
  - Preconditioning methods are spectrum-dependent

# Preconditioning

Multiply linear system

$$Au = b$$

# Preconditioning

Multiply linear system

$$Au = b$$
$$M_L^{-1} Au = M_L^{-1} b$$

# Preconditioning

Multiply linear system

$$\begin{aligned} Au &= b \\ M_L^{-1} Au &= M_L^{-1} b \\ M_L^{-1} A \left( M_R M_R^{-1} \right) u &= M_L^{-1} b \end{aligned}$$

# Preconditioning

Multiply linear system

$$
\begin{aligned}
Au &= b \\
M_L^{-1} Au &= M_L^{-1} b \\
M_L^{-1} A \left( M_R M_R^{-1} \right) u &= M_L^{-1} b \\
\left( M_L^{-1} A M_R \right) \left( M_R^{-1} u \right) &= M_L^{-1} b
\end{aligned}
$$

# Preconditioning

Multiply linear system

$$
\begin{aligned}
Au &= b \\
M_L^{-1} Au &= M_L^{-1} b \\
M_L^{-1} A \left( M_R M_R^{-1} \right) u &= M_L^{-1} b \\
\left( M_L^{-1} A M_R \right) \left( M_R^{-1} u \right) &= M_L^{-1} b \\
\tilde{A} \tilde{u} &= \tilde{b}
\end{aligned}
$$

# Preconditioning

Multiply linear system

$$
\begin{aligned}
Au &= b \\
M_L^{-1} A u &= M_L^{-1} b \\
M_L^{-1} A \left( M_R M_R^{-1} \right) u &= M_L^{-1} b \\
\left( M_L^{-1} A M_R \right) \left( M_R^{-1} u \right) &= M_L^{-1} b \\
\tilde{A} \tilde{u} &= \tilde{b}
\end{aligned}
$$

# Preconditioning

Multiply linear system

$$
\begin{aligned}
Au &= b \\
M_L^{-1} Au &= M_L^{-1} b \\
M_L^{-1} A \left( M_R M_R^{-1} \right) u &= M_L^{-1} b \\
\left( M_L^{-1} A M_R \right) \left( M_R^{-1} u \right) &= M_L^{-1} b \\
\tilde{A} \tilde{u} &= \tilde{b}
\end{aligned}
$$

# Preconditioning

Multiply linear system

$$
\begin{aligned}
Au &= b \\
M_L^{-1} Au &= M_L^{-1} b \\
M_L^{-1} A \left( M_R M_R^{-1} \right) u &= M_L^{-1} b \\
\left( M_L^{-1} A M_R \right) \left( M_R^{-1} u \right) &= M_L^{-1} b \\
\tilde{A} \tilde{u} &= \tilde{b}
\end{aligned}
$$

Krylov solutions look like

$$\tilde{u}_k \in \text{span}\{\tilde{b}, \tilde{A}\tilde{b}, \ldots, \tilde{A}^{k-1}\tilde{b}\}$$

$$u_k = M_R \tilde{u}_k$$

# Preconditioning

Multiply linear system

$$\begin{aligned} Au &= b \\ M_L^{-1} Au &= M_L^{-1} b \\ M_L^{-1} A \left( M_R M_R^{-1} \right) u &= M_L^{-1} b \\ \left( M_L^{-1} A M_R \right) \left( M_R^{-1} u \right) &= M_L^{-1} b \\ \tilde{A} \tilde{u} &= \tilde{b} \end{aligned}$$

Krylov solutions look like

$$\tilde{u}_k \in \text{span}\{\tilde{b}, \tilde{A}\tilde{b}, \ldots, \tilde{A}^{k-1}\tilde{b}\}$$

$$u_k = M_R \tilde{u}_k$$

. . . depends on spectrum of $\tilde{A}$ rather than $A$.

# Preconditioning

Multiply linear system

$$\begin{aligned} Au &= b \\ M_L^{-1}Au &= M_L^{-1}b \\ M_L^{-1}A\left(M_R M_R^{-1}\right)u &= M_L^{-1}b \\ \left(M_L^{-1}AM_R\right)\left(M_R^{-1}u\right) &= M_L^{-1}b \\ \tilde{A}\tilde{u} &= \tilde{b} \end{aligned}$$

Krylov solutions look like

$$\tilde{u}_k \in \text{span}\{\tilde{b}, \tilde{A}\tilde{b}, \ldots, \tilde{A}^{k-1}\tilde{b}\}$$

$$u_k = M_R\tilde{u}_k$$

. . . depends on spectrum of $\tilde{A}$ rather than $A$.

- $M_L^{-1}$ – left preconditioning matrix
- $M_R$ – right preconditioning matrix

Choose so that $\tilde{A}$ has a *nice* spectrum.

# Specific problem (from PBSM3D)

For stability of time-integration, requires solution to a steady-state PDE at each time step:

$$\left(\mathbb{I} - D\nabla^2\right) u = f(x, y)$$

# Specific problem (from PBSM3D)

For stability of time-integration, requires solution to a steady-state PDE at each time step:

$$\left(\mathbb{I} - D\nabla^2\right) u = f(x, y)$$

- $D$ – smoothing coefficient
- $u(x, y)$ – rate of snow mass transport at $x, y$ coordinate
- $f(x, y)$ – known quantity dependent on other physical parameters

# Specific problem (from PBSM3D)

For stability of time-integration, requires solution to a steady-state PDE at each time step:

$$\left(\mathbb{I} - D\nabla^2\right) u = f(x, y)$$

*(Requires much more, but we're just going to stick to this again)*

- $D$ – smoothing coefficient
- $u(x, y)$ – rate of snow mass transport at $x, y$ coordinate
- $f(x, y)$ – known quantity dependent on other physical parameters

# Discretization of specific problem

$$Au = b$$

# Discretization of specific problem

$$Au = b$$

$A$ represents a finite volume discretization of $(\mathbb{I} - D\nabla^2)$.
($A$ is symmetric positive definite.)

# Discretization of specific problem

$$Au = b$$

$A$ represents a finite volume discretization of $(\mathbb{I} - D\nabla^2)$.
($A$ is symmetric positive definite.)

Components of $u$ represent the rate of snow mass transport, constant on each element.

# Discretization of specific problem

$$Au = b$$

$A$ represents a finite volume discretization of $(\mathbb{I} - D\nabla^2)$. ($A$ is symmetric positive definite.)

Components of $u$ represent the rate of snow mass transport, constant on each element.

Components of $b$ represent known quantities dependent on current snow mass (among other things), constant on each element.

# Discretization of specific problem

$$Au = b$$

$A$ represents a finite volume discretization of $(\mathbb{I} - D\nabla^2)$. ($A$ is symmetric positive definite.)

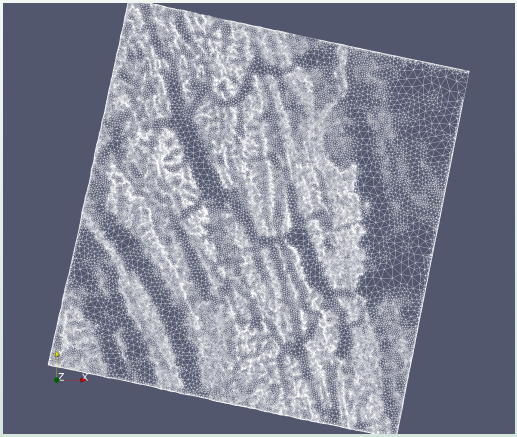
Components of $u$ represent the rate of snow mass transport, constant on each element.

Components of $b$ represent known quantities dependent on current snow mass (among other things), constant on each element.

Use known solution $u^* = 1^{N\times 1}$, $b^* = Au^*$ for testing, i.e., solve $Au = b^*$.

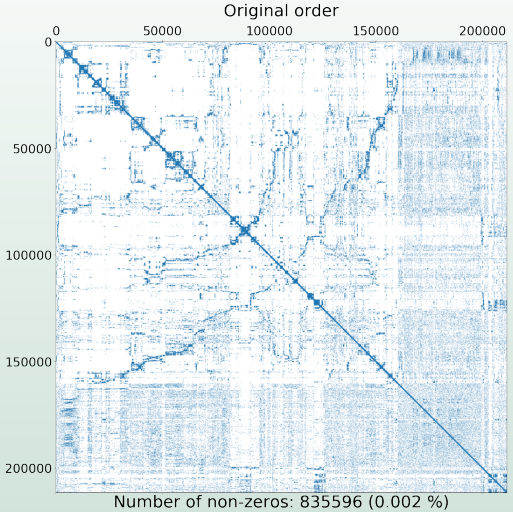
# Mesh: fine resolution

High resolution (211415 elements)



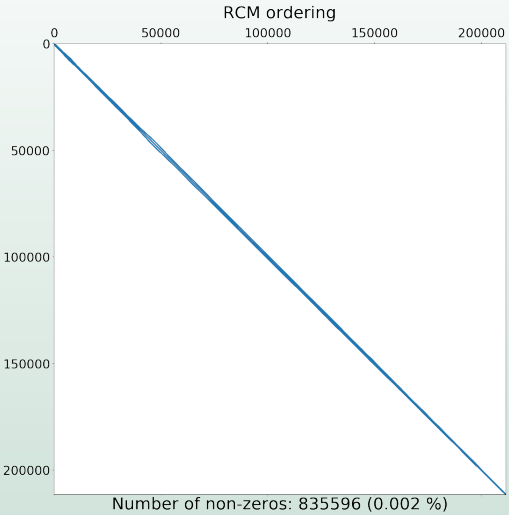
# Matrix structure: original order

Bandwidth: 210855



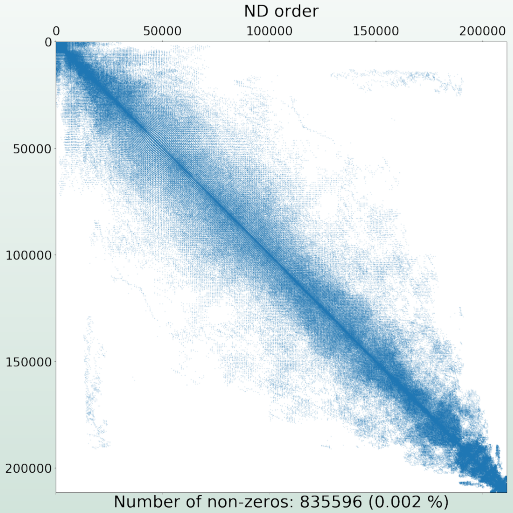
# Matrix structure: RCM order

Bandwidth: 1320



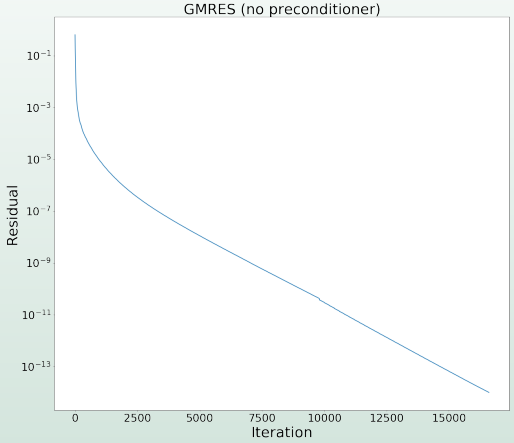
# Matrix: ND order

Bandwidth: 173136



# Performance: GMRES

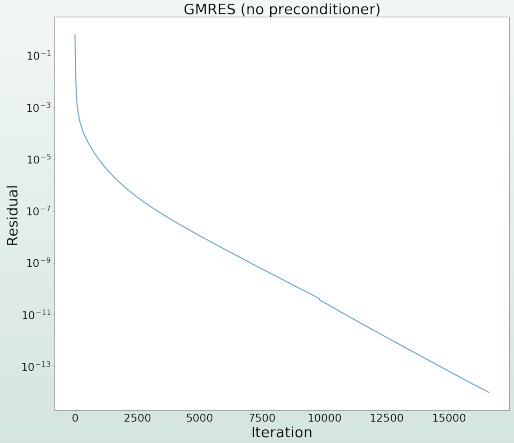
`tol=1e-14`, `N_restart=100`
No preconditioner.



# Performance: GMRES

`tol=1e-14`, `N_restart=100`
No preconditioner.

Number of iterations: 16607

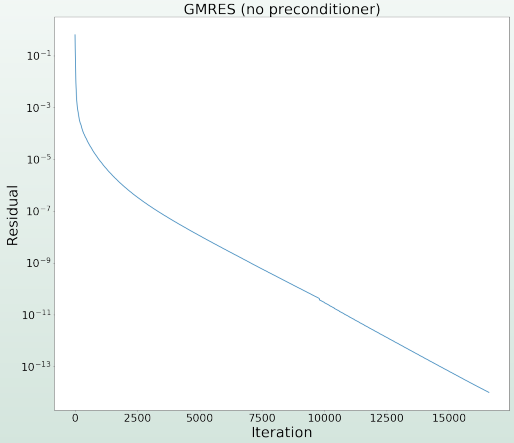


# Performance: GMRES

`tol=1e-14`, `N_restart=100`
No preconditioner.

Number of iterations: 16607

Original order
`Wall time:` `1min 52s`



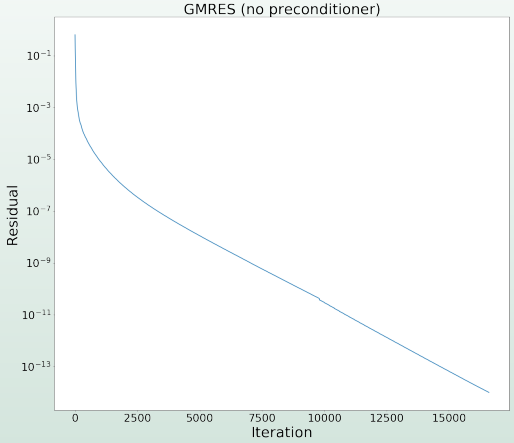
# Performance: GMRES

`tol=1e-14`, `N_restart=100`
No preconditioner.

Number of iterations: 16607

Original order
`Wall time: 1min 52s`

RCM order
`Wall time: 1min 23s`



# Performance: GMRES

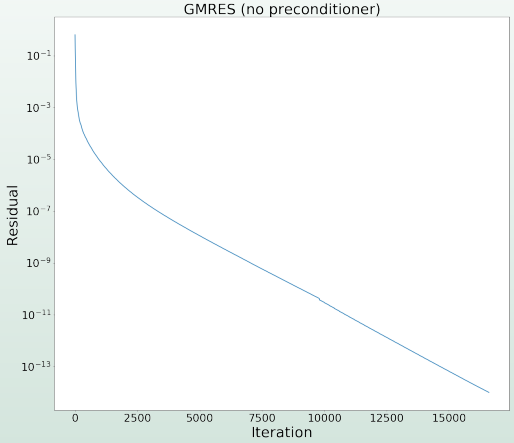
`tol=1e-14`, `N_restart=100`
No preconditioner.

Number of iterations: 16607

Original order
`Wall time: 1min 52s`

RCM order
`Wall time: 1min 23s`

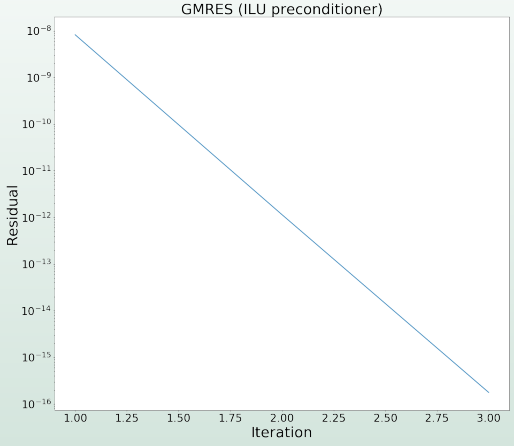
ND order
`Wall time: 1min 42s`



# Performance: GMRES

`tol=1e-14`

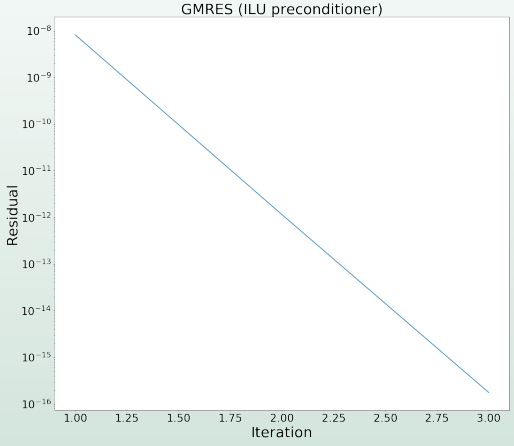
ILU preconditioner.



# Performance: GMRES

`tol=1e-14`
ILU preconditioner.

Number of iterations: 3

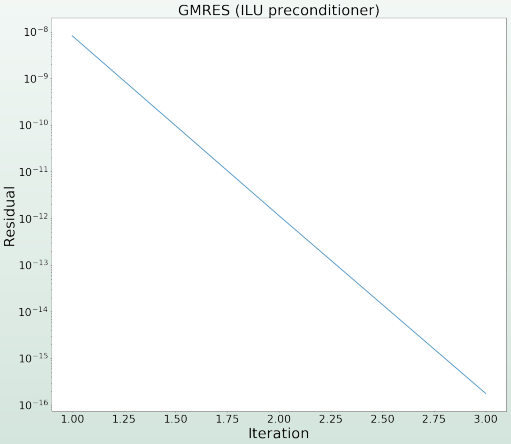


# Performance: GMRES

`tol=1e-14`
ILU preconditioner.

Number of iterations: 3

| Order | Setup (s) | Solve (s) | Total (s) |
|---|---|---|---|
| original | 0.682 | 0.497 | 1.179 |
| RCM | 0.602 | 0.426 | 1.028 |
| ND | 0.644 | 0.466 | 1.110 |

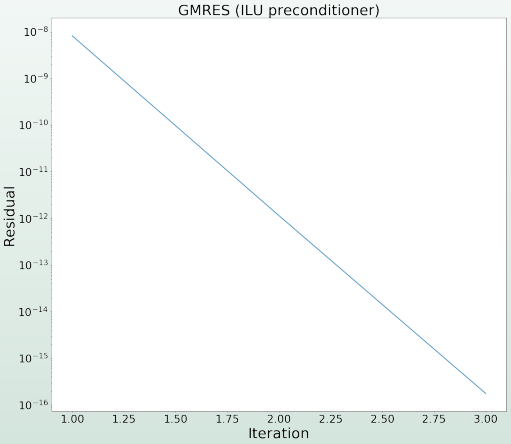


# Performance: GMRES

`tol=1e-14`
ILU preconditioner.

Number of iterations: 3

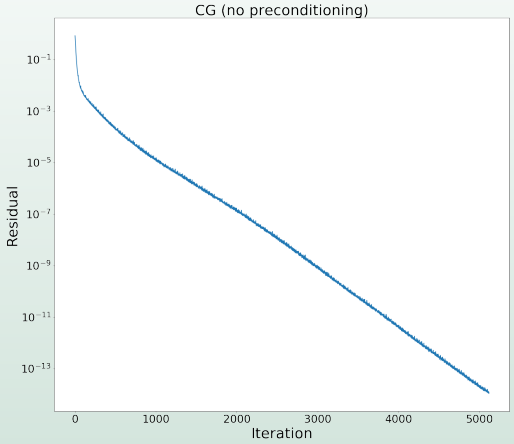
| Order | Setup (s) | Solve (s) | Total (s) |
|---|---|---|---|
| original | 0.682 | 0.497 | 1.179 |
| RCM | 0.602 | 0.426 | 1.028 |
| ND | 0.644 | 0.466 | 1.110 |



# Performance: CG

`tol=1e-14`

No preconditioner.

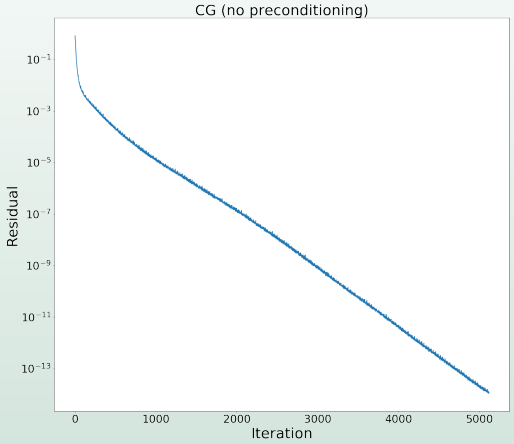


# Performance: CG

`tol=1e-14`

No preconditioner.

Number of iterations: 5117

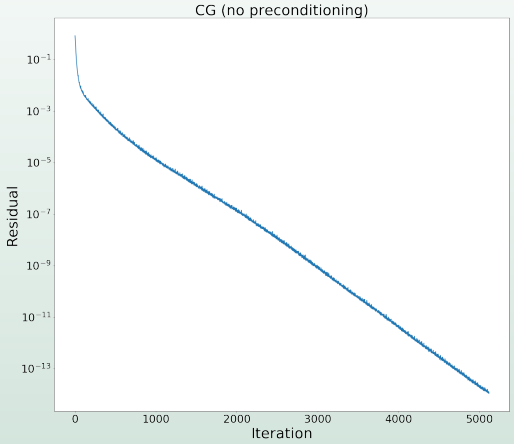


# Performance: CG

`tol=1e-14`
No preconditioner.

Number of iterations: 5117

Original order
`Wall time:` 21.3s



# Performance: CG

`tol=1e-14`

No preconditioner.

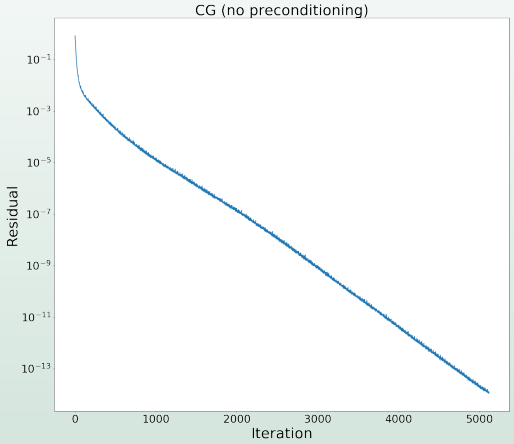
Number of iterations: 5117

Original order

`Wall time:` 21.3s

RCM order

`Wall time:` 13.6s



# Performance: CG

`tol=1e-14`

No preconditioner.

Number of iterations: 5117

Original order

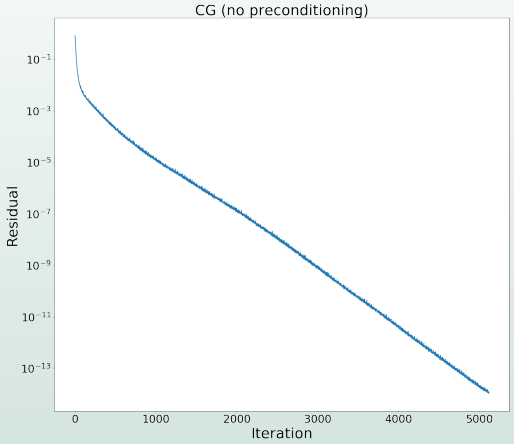
`Wall time:` 21.3s

RCM order

`Wall time:` 13.6s

ND order

`Wall time:` 20.4s



# Performance: CG

`tol=1e-14`
ILU preconditioner.

*No plot*

# Performance: CG

`tol=1e-14`
ILU preconditioner.

Number of iterations: 3

*No plot*

# Performance: CG

`tol=1e-14`
ILU preconditioner.

Number of iterations: 3

| Order | Setup (s) | Solve (s) | Total (s) |
|---|---|---|---|
| original | 0.682 | 0.341 | 1.023 |
| RCM | 0.602 | 0.301 | 0.903 |
| ND | 0.644 | 0.333 | 0.977 |

*No plot*

# Performance: CG

`tol=1e-14`
ILU preconditioner.

Number of iterations: 3

| Order | Setup (s) | Solve (s) | Total (s) |
|---|---|---|---|
| original | 0.682 | 0.341 | 1.023 |
| RCM | 0.602 | 0.301 | 0.903 |
| ND | 0.644 | 0.333 | 0.977 |

*No plot*

# Fine mesh summary

211415 elements $\rightarrow$ 835596 non-zero matrix entries

| | LU | | GMRES | | CG | |
|---|---|---|---|---|---|---|
| Order | Time (s) | Accuracy | Time (s) | Accuracy | Time (s) | Accuracy |
| original | 25145 | 1.2e-13 | 1.179 | 6.3e-10 | 1.023 | 1.2e-13 |
| RCM | 53.3 | 1.6e-13 | 1.028 | 5.9e-10 | 0.903 | 1.2e-13 |
| ND | 1.3 | 1.5e-13 | 1.110 | 6.1e-10 | 0.977 | 1.1e-13 |

# Fine mesh summary

211415 elements $\rightarrow$ 835596 non-zero matrix entries

| | LU | | GMRES | | CG | |
|---|---|---|---|---|---|---|
| Order | Time (s) | Accuracy | Time (s) | Accuracy | Time (s) | Accuracy |
| original | 25145 | 1.2e-13 | 1.179 | 6.3e-10 | 1.023 | 1.2e-13 |
| RCM | 53.3 | 1.6e-13 | 1.028 | 5.9e-10 | 0.903 | 1.2e-13 |
| ND | 1.3 | 1.5e-13 | 1.110 | 6.1e-10 | 0.977 | 1.1e-13 |

Overall Winner: Preconditioned CG

# Fine mesh summary

211415 elements $\rightarrow$ 835596 non-zero matrix entries

| LU | | | GMRES | | CG | |
|---|---|---|---|---|---|---|
| Order | Time (s) | Accuracy | Time (s) | Accuracy | Time (s) | Accuracy |
| original | 25145 | 1.2e-13 | 1.179 | 6.3e-10 | 1.023 | 1.2e-13 |
| RCM | 53.3 | 1.6e-13 | 1.028 | 5.9e-10 | 0.903 | 1.2e-13 |
| ND | 1.3 | 1.5e-13 | 1.110 | 6.1e-10 | 0.977 | 1.1e-13 |

Overall Winner: Preconditioned CG

Notable showing: LU with ND order

*If many solves can be performed from a single factorization.*

# Exercise

You have the matrix in a text file in

`https:github.com/uofs-simlab/Math313`.

Load that into something like PETSc/SLEPc to play around with solving algorithms and eigenvalue computaitons.

# Concluding remarks

- A lot of pieces have to come together in the real world before we get to linear algebra.

# Concluding remarks

- A lot of pieces have to come together in the real world before we get to linear algebra.
- There's still a lot to investigate once we get there.

# Concluding remarks

- A lot of pieces have to come together in the real world before we get to linear algebra.
- There's still a lot to investigate once we get there.

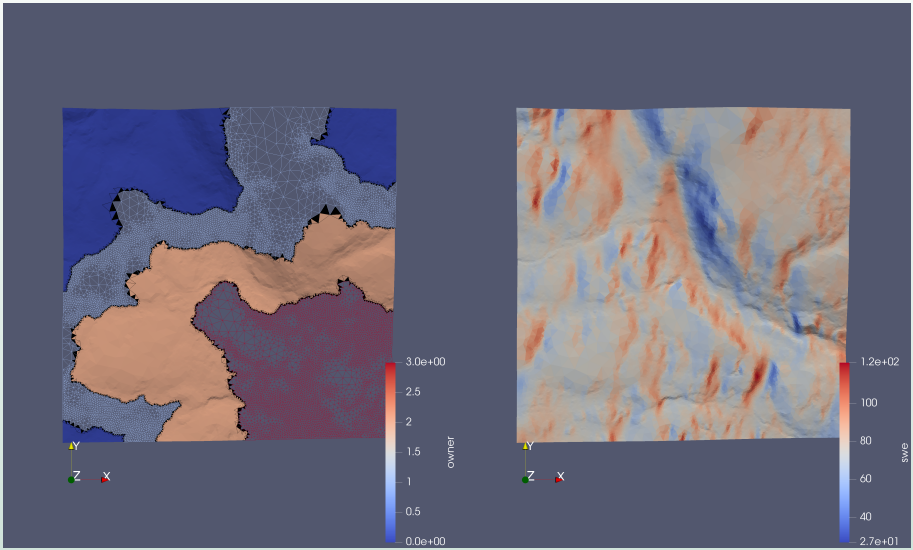
- Substantial performance improvement obtained through preconditioning.

# Distributed linear algebra

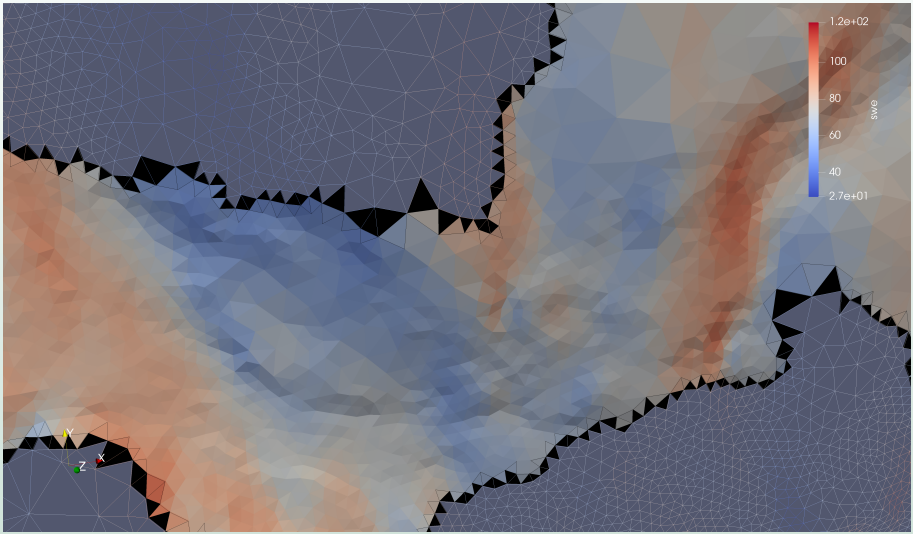
*Done within CHM, not in the jupyter notebook.*

`https://github.com/ChrisMarsh/CHM/tree/dev`

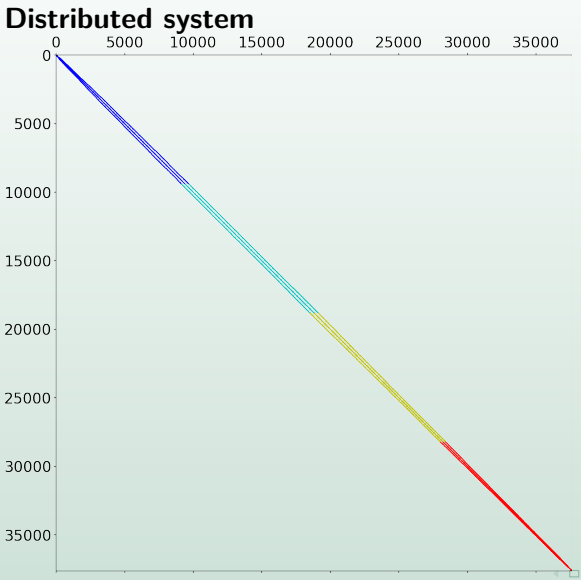
# Ownership of distributed mesh



# . . . introduces ghosted elements



# . . . and distributed Linear Algebra



# Performance

**Linear system iterations**

Some preconditioners aren't as effective as the number of distributed processes are increased.

| Np | Iterations* |
|---|---|
| 1 | 11 |
| 2 | 34 |
| 4 | 35 |
| 8 | 35 |
| 16 | 42 |

* Trilinos/Belos: GMRES(1e-8)
Trilinos/Ifpack2: BlockJacobi(Np)*ILUT(3,1e-4)