Scientific Computing WS 2019/2020

Lecture 22+23

Jürgen Fuhrmann

juergen.fuhrmann@wias-berlin.de

Elements of iterative methods (Saad Ch.4)

Let $V = \mathbb{R}^n$ be equipped with the inner product $(\cdot, \cdot)$. Let A be an $n \times n$ nonsingular matrix.

Solve $Au = b$ iteratively. For this purpose, two components are needed:

- **Preconditioner**: a matrix $M \approx A$ "approximating" the matrix A but with the property that the system $Mv = f$ is easy to solve
- **Iteration scheme**: algorithmic sequence using M and A which updates the solution step by step

Simple iteration with preconditioning

Idea: $A\hat{u} = b \Rightarrow$

$$\hat{u} = \hat{u} - M^{-1}(A\hat{u} - b)$$

$\Rightarrow$ iterative scheme

$$u_{k+1} = u_k - M^{-1}(Au_k - b) \quad (k = 0, 1 \dots)$$

1. Choose initial value u_0, tolerance ε, set $k = 0$
2. Calculate *residuum* $r_k = Au_k - b$
3. Test convergence: if $||r_k|| < \varepsilon$ set $u = u_k$, finish
4. Calculate *update*: solve $Mv_k = r_k$
5. Update solution: $u_{k+1} = u_k - v_k$, set $k = k + 1$, repeat with step 2.

The Jacobi method

- Let $A = D - E - F$, where D: main diagonal, E: negative lower triangular part F: negative upper triangular part
- Preconditioner: $M = D$, where D is the main diagonal of $A \Rightarrow$

$$u_{k+1,i} = u_{k,i} - \frac{1}{a_{ii}} \left(\sum_{j=1 \ldots n} a_{ij} u_{k,j} - b_i \right) \quad (i = 1 \ldots n)$$

- Equivalent to the succesive (row by row) solution of

$$a_{ii} u_{k+1,i} + \sum_{j=1 \ldots n, j \neq i} a_{ij} u_{k,j} = b_i \quad (i = 1 \ldots n)$$

- Already calculated results not taken into account
- Alternative formulation with $A = M - N$:

$$\begin{aligned} u_{k+1} &= D^{-1}(E + F)u_k + D^{-1}b \\ &= M^{-1}Nu_k + M^{-1}b \end{aligned}$$

- Variable ordering does not matter

The Gauss-Seidel method

- Solve for main diagonal element row by row
- Take already calculated results into account

$$a_{ii}u_{k+1,i} + \sum_{j<i} a_{ij}u_{k+1,j} + \sum_{j>i} a_{ij}u_{k,j} = b_i \qquad (i = 1 \ldots n)$$

$$(D - E)u_{k+1} - Fu_k = b$$

- May be it is faster
- Variable order probably matters
- Preconditioners: forward $M = D - E$, backward: $M = D - F$
- Splitting formulation: $A = M - N$
  forward: $N = F$, backward: $M = E$
- Forward case:

$$\begin{aligned} u_{k+1} &= (D - E)^{-1}Fu_k + (D - E)^{-1}b \\ &= M^{-1}Nu_k + M^{-1}b \end{aligned}$$

Incomplete LU factorizations (ILU)

Idea (Varga, Buleev, 1960):

- fix a predefined zero pattern
- apply the standard LU factorization method, but calculate only those elements, which do not correspond to the given zero pattern
- Result: incomplete LU factors L, U, remainder R:

$$A = LU - R$$

- Problem: with complete LU factorization procedure, for any nonsingular matrix, the method is stable, i.e. zero pivots never occur. Is this true for the incomplete LU Factorization as well ?

Stability of ILU

Theorem (Saad, Th. 10.2): If A is an M-Matrix, then the algorithm to compute the incomplete LU factorization with a given nonzero pattern

$$A = LU - R$$

is stable. Moreover, $A = LU - R$ is a regular splitting.

ILU(0)

- Special case of ILU: ignore any fill-in.
- Representation:

$$M = (\tilde{D} - E)\tilde{D}^{-1}(\tilde{D} - F)$$

- $\tilde{D}$ is a diagonal matrix (wich can be stored in one vector) which is calculated by the incomplete factorization algorithm.
- Setup:

```
for i=1:n
    d[i]=a[i,i]
end

for i=1:n
    d[i]=1.0/d[i]
    for j=i+1:n
        d[j]=d[j]-a[i,j]*d[i]*a[j,i]
    end
end
```

ILU(0)

- Generally better convergence properties than Jacobi, Gauss-Seidel
- One can develop block variants
- Alternatives:
  - ILUM: ("modified"): add ignored off-diagonal entries to $\tilde{D}$
  - ILUT: zero pattern calculated dynamically based on drop tolerance
- Dependence on ordering
- Can be parallelized using graph coloring
- Not much theory: experiment for particular systems
- I recommend it as the default initial guess for a sensible preconditioner
- Incomplete Cholesky: symmetric variant of ILU

Preconditioned CG II

Assume $\tilde{r}_i = E^{-1} r_i$, $\tilde{d}_i = E^T d_i$, we get the equivalent algorithm

$$
\begin{aligned}
r_0 &= b - Au_0 \\
d_0 &= M^{-1} r_0 \\
\alpha_i &= \frac{(M^{-1} r_i, r_i)}{(Ad_i, d_i)} \\
u_{i+1} &= u_i + \alpha_i d_i \\
r_{i+1} &= r_i - \alpha_i A d_i \\
\beta_{i+1} &= \frac{(M^{-1} r_{i+1}, r_{i+1})}{(r_i, r_i)} \\
d_{i+1} &= M^{-1} r_{i+1} + \beta_{i+1} d_i
\end{aligned}
$$

It relies on the solution of the preconditioning system, the calculation of the matrix vector product and the calculation of the scalar product.

The convergence rate of the method is

$$
||e_i||_{E^{-1}AE^{-T}} \leq 2 \left(\frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i ||e_0||_{E^{-1}AE^{-T}}
$$

where $\kappa = \frac{\gamma_{max}}{\gamma_{min}}$ comes from $\gamma_{min}(Mu, u) \leq (Au, u) \leq \gamma_{max}(Mu, u)$.

Issues and consequences

- Usually we stop the iteration when the residual r becomes small. However during the iteration, floating point errors occur which distort the calculations and lead to the fact that the accumulated residuals

$$r_{i+1} = r_i - \alpha_i A d_i$$

  give a much more optimistic picture on the state of the iteration than the real residual

$$r_{i+1} = b - A u_{i+1}$$

- The convergence rate estimate in terms of $\frac{\sqrt{\kappa}-1}{\sqrt{\kappa}+1}$ indeed provides a qualitatively better complexity estimate for the solution algorithm
- Always consider CG when solving symmetric positive definite linear systems iteratively

Iterative solver complexity I

- Solve linear system iteratively until $||e_k|| = ||(I - M^{-1}A)^k e_0|| \leq \epsilon$

$$\rho^k e_0 \leq \epsilon$$
$$k \ln \rho < \ln \epsilon - \ln e_0$$
$$k \geq k_\rho = \left\lceil \frac{\ln e_0 - \ln \epsilon}{\ln \rho} \right\rceil$$

- $\Rightarrow$ we need at least k_ρ iteration steps to reach accuracy ϵ
- Optimal iterative solver complexity - assume:
  - $\rho < \rho_0 < 1$ independent of h resp. N
  - A sparse ($A \cdot u$ has complexity $O(N)$)
  - Solution of $Mv = r$ has complexity $O(N)$.

  $\Rightarrow$ Number of iteration steps k_ρ independent of N
  $\Rightarrow$ Overall complexity $O(N)$

Iterative solver complexity II

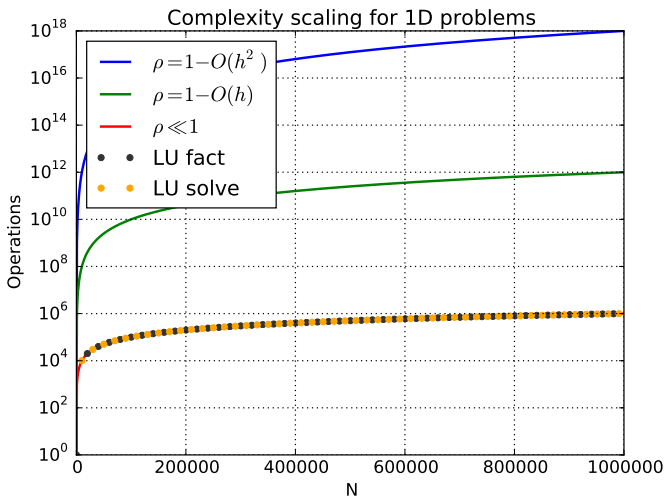
- Assume
  - $\rho = 1 - h^\delta \Rightarrow \ln \rho \approx -h^\delta \rightarrow k_\rho = O(h^{-\delta})$
  - d: space dimension $\Rightarrow h \approx N^{-\frac{1}{d}} \Rightarrow k_\rho = O(N^{\frac{\delta}{d}})$
  - $O(N)$ complexity of one iteration step (e.g. Jacobi, Gauss-Seidel)

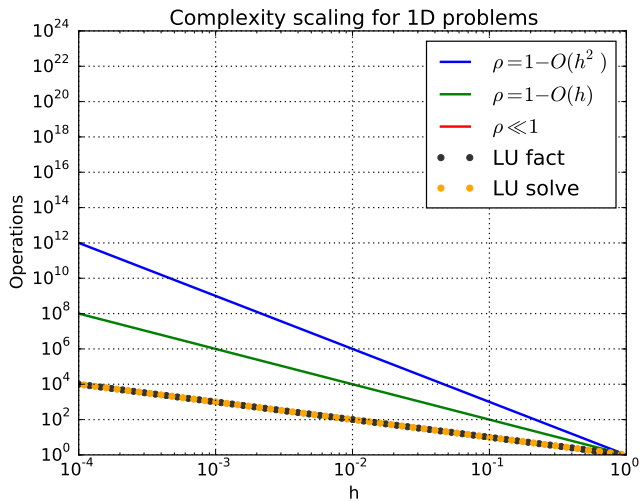
  $\Rightarrow$ Overall complexity $O(N^{1+\frac{\delta}{d}}) = O(N^{\frac{d+\delta}{d}})$
- Jacobi: $\delta = 2$
- Hypothetical "Improved iterative solver" with $\delta = 1$?
- Overview on complexity estimates

dim	$\rho = 1 - O(h^2)$	$\rho = 1 - O(h)$	LU fact.	LU solve
1	$O(N^3)$	$O(N^2)$	$O(N)$	$O(N)$
2	$O(N^2)$	$O(N^{\frac{3}{2}})$	$O(N^{\frac{3}{2}})$	$O(N \log N)$
3	$O(N^{\frac{5}{3}})$	$O(N^{\frac{4}{3}})$	$O(N^2)$	$O(N^{\frac{4}{3}})$

Solver complexity scaling for 1D problems

dim	$\rho = 1 - O(h^2)$	$\rho = 1 - O(h)$	LU fact.	LU solve
1	$O(N^3)$	$O(N^2)$	$O(N)$	$O(N)$

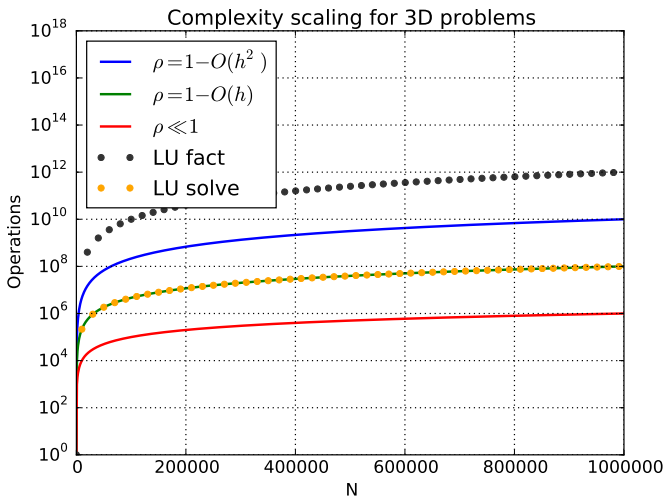


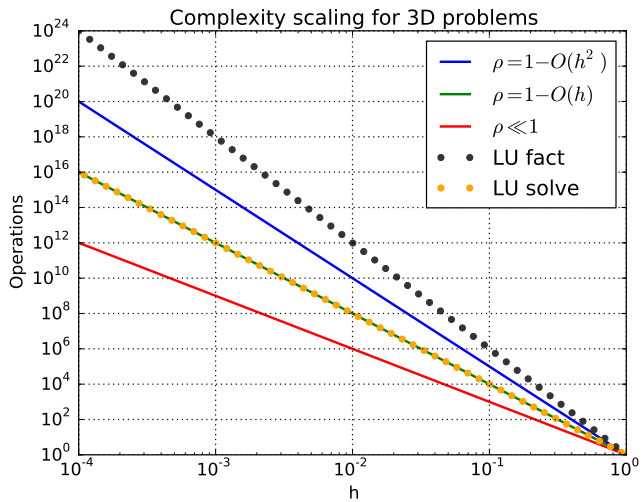


- Direct solvers significantly better than iterative ones

Solver complexity scaling for 3D problems

dim	$\rho = 1 - O(h^2)$	$\rho = 1 - O(h)$	LU fact.	LU solve
3	$O(N^{\frac{5}{3}})$	$O(N^{\frac{4}{3}})$	$O(N^2)$	$O(N^{\frac{4}{3}})$





- LU factorization is expensive
- LU solve on par with improved iterative solvers

Multigrid: Iterative solver with O(N) complexity

Idea: combine classical preconditioners with coarse grid correction

- Assume embedded finite element spaces $V_0 \dots V_l$ such tha $V_0 \subset V_1 \subset \dots V_l$
- V_k is produced from V_{k-1} by subdividing each triangle into four
  Alternative: finite difference refinement by halving x and y intervals
- $\Rightarrow$ interpolation operator from embedding of spaces $I_{k-1}^k : V_{k-1} \rightarrow V_k$
- $\Rightarrow$ restriction operator $R_{k-1}^k = (I_{k-1}^k)^T : V_k \rightarrow V_{k-1}$
- Discretization matrix A_k on each level $k = 0 \dots l$
- "Smoother" (Jacobi, ILU, ...) M_k on each level $k = 1 \dots l$
- Number of smoothig steps n_s
- Coarse grid solver
- Number of coarse grid correction steps γ

Multigrid Algorithm

Procedure `Multigrid`(l, u_l, f_l)

  if $l = 0$ **then**

    $u_0 = A_0^{-1} f_0$ `// coarse grid solution`

  else

    for $i = 1, n_s$ **do**

      $u_l = u_l - M_l^{-1}(A_l u_l - f_l)$ `// pre-smoothing`

    end

    $f_{l-1} = R_{l-1}^{l}(A_l u_l - f_l)$ `// restriction`

    $u_{l-1} = 0$

    for $i = 1, \gamma$ **do**

      `Multigrid`$(l-1, u_{l-1}, f_{l-1})$ `// coarse grid corr.`

    end

    $u_l = u_l - I_{l-1}^{l} u_{l-1}$ `// interpolation`

    for $i = 1, n_s$ **do**

      $u_l = u_l - M_l^{-1}(A_l u_l - f_l)$ `// post-smoothing`

    end

  end

end

Multigrid remarks

- $\gamma = 1 \Rightarrow$ V-Cycle, $\gamma = 2 \Rightarrow$ W-Cycle
- Use as a preconditioner in CG methods
- First development in early 60ies by Bakhvalov, Fedorenko
- Works well for hierarchically embedded grid systems and smooth problem coefficients: $O(N)$ solution complexity
- Other variant can use embedding of FEM spaces of growing polynomial degree
- "Algebraic multigrid": define coarse grid, interpolations in an algebraic way by choosing coarse grid points and an interpolation from matrix entries
- Hybrid variant: structured grid, matrix dependent transfer operators for problems with strongly varying coefficients (my PhD. thesis)

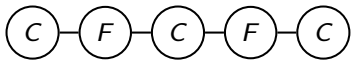
Matrix dependent transfer operators

- Finite Difference/finite volume discretizations on tensor product grids give rise to canonical multigrid hierarchies
- For standard (linear) interpolation/restriction, bad convergence behaviour in the case of strongly varying coefficients, e.g. for equations of type

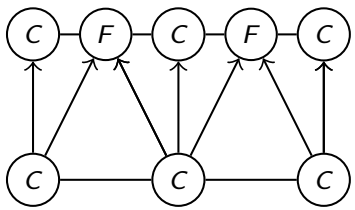
$$-\nabla \cdot a(\mathbf{x})\nabla u = f \quad \text{in} \Omega \subset \mathbb{R}^{1,2,3}$$

  where $a(\mathbf{x})$ has jumps $\Rightarrow$ loss of regularity
- Remedy: multigrid with matrix dependent transfer operators
- Hybrid between algebraic and geometric multigrid

Matrix dependent transfer operators: the 1D case

- One-dimensional grid: 

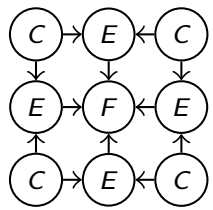
$A = \begin{pmatrix} A_{FF} & A_{FC} \\ A_{CF} & A_{CC} \end{pmatrix}$
- Interpolation from C-points to F-points: $I = \begin{pmatrix} -A_{FF}^{-1} A_{FC} \\ I \end{pmatrix}$



- Restriction to C points: $R = \begin{pmatrix} -A_{CF} A_{FF}^{-1} & I \end{pmatrix}$
- Coarse grid operator on grid of C-points:
  $A_{coarse} = RAI = A_{CC} - A_{CF} A_{FF}^{-1} A_{FC} \equiv$ Schur complement
- $\equiv$ Gaussian elemination where F points are eliminated first
- Continue recursively, as Schur complement has linear graph
- **Cyclic Reduction** – a variant of Gaussian elemination

Matrix dependent transfer operators: the 2D case

- Two-dimensional case, "5-point star": F and E points are fine grid points
- $A = \begin{pmatrix} \begin{pmatrix} A_{FF} & A_{FE} \\ A_{EF} & A_{EE} \end{pmatrix} & \begin{pmatrix} 0 \\ A_{EC} \end{pmatrix} \\ \begin{pmatrix} 0 & A_{CE} \end{pmatrix} & A_{CC} \end{pmatrix}$
- $I = \begin{pmatrix} \begin{pmatrix} A_{FF}^{-1} A_{FE} \tilde{A}_{EE}^{-1} A_{EN} \\ -\tilde{A}_{EE}^{-1} A_{EN} \end{pmatrix} \\ I \end{pmatrix}$

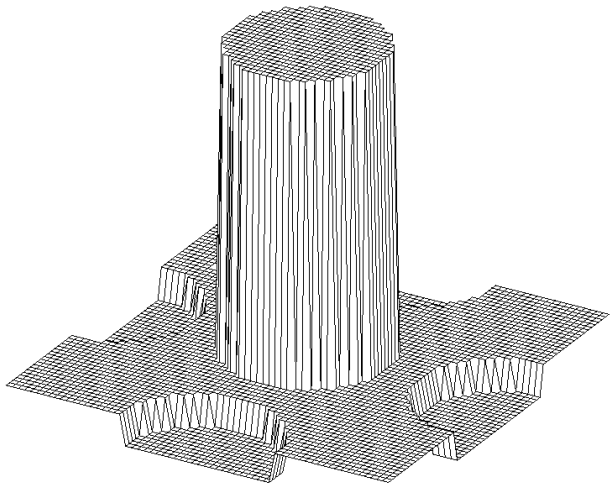


- $\tilde{A}_{EE} = A_{EE} -$ off diagonal elements A_{EF} added to diagonal
- $R = I^T$, $A_C = 2(A_{CC} - A_{CE} \tilde{A}_{EE}^{-1} A_{EC}) \approx RAI$

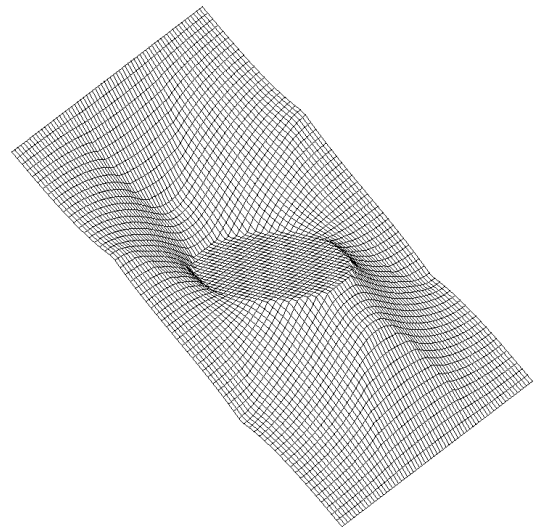
J. Fuhrmann, Ph.D thesis, 1994

Matrix dependent transfer operators: numerical results

2D test problem, similar structure for 3D:



log $a(\mathbf{x})$



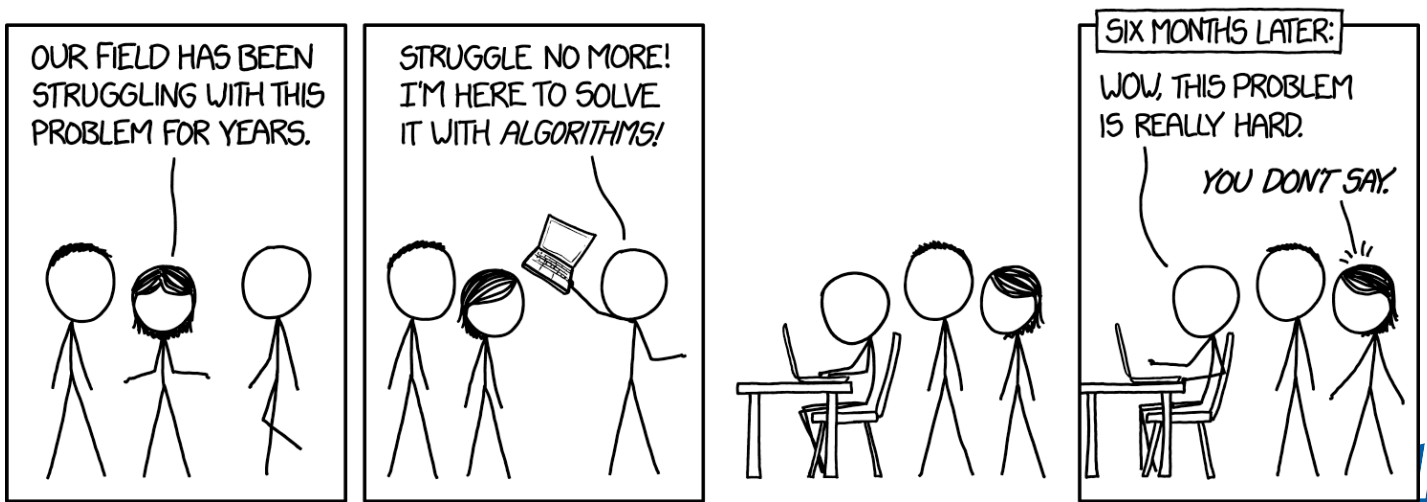
solution $u(\mathbf{x})$

Grid	d=2, ρ_S	d=2, ρ_{CG}	d=3, ρ_S	d=3, ρ_{CG}
16^d	0.087	0.040	0.086	0.049
32^d	0.086	0.047	0.094	0.046
64^d	0.086	0.065	0.114	0.064

J. Fuhrmann, Ph.D thesis, 1994

Algebraic Multigrid

- Heuristic choice of coarse points by "strong connections" in matrix graph
- Definition of interpolation from algebraic considerations
- Galerkin construction $A_C = RAI$ in general leads to unwanted fill-in $\Rightarrow$ what can we omit ?
- Agglomeration variant: cluster fine grid nodes together, use piecewise linear interpolation $\Rightarrow$ easier to build, but slower convergence
- For a recent comprehensive intro to AMG see Xu&Zikatanov, Acta Numerica 2017
- Hard to prove anything due to ubiquitous heuristic



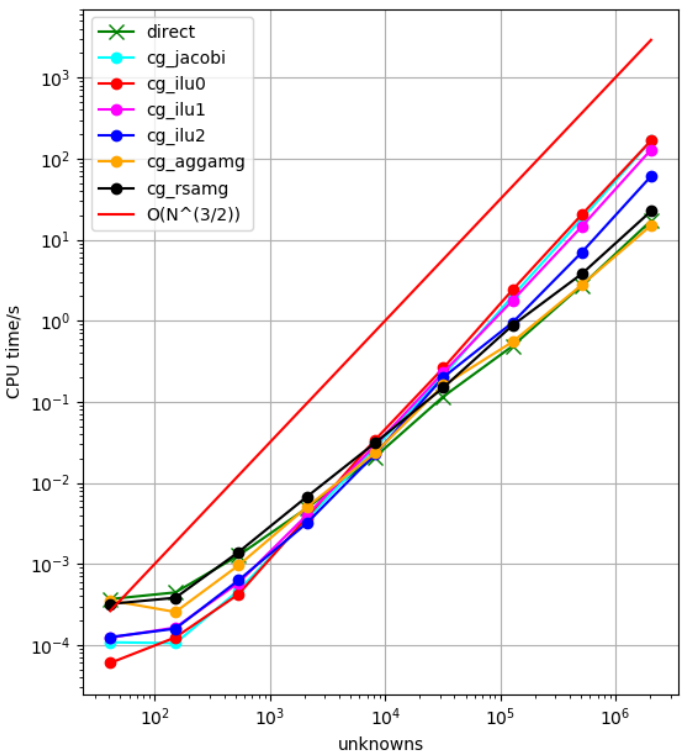
Smoothed Aggregation Multigrid

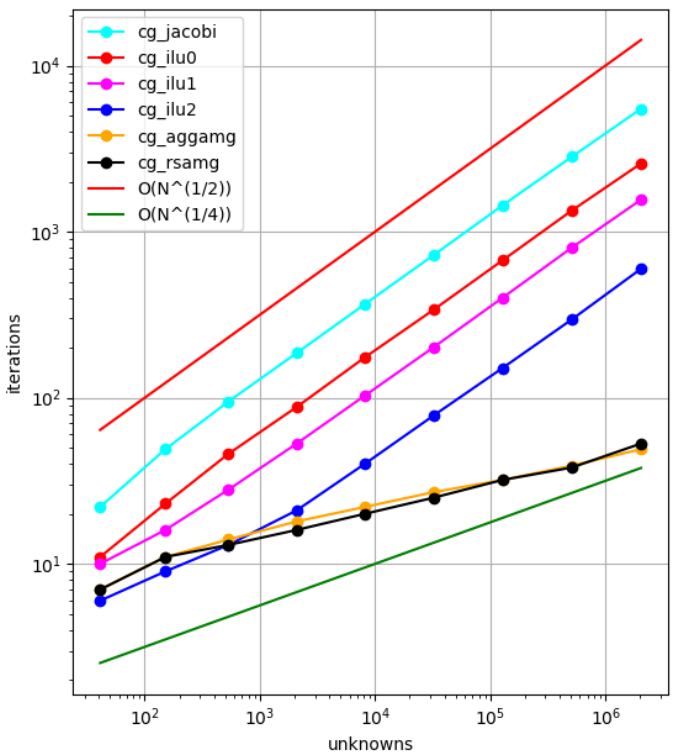
- Coarsening $\equiv$ aggregation of nodes into clusters
- In the finite volume context we can see this as joining control volumes
- Piecewise constant interpolation: all fine grid nodes get the same value from the coarse grid cluster
- Adjoint restriction adds up values.

“Ruge-Stüben” Multigrid

- Coarsening: selection of coarse grid points by certain heuristics
- Matrix dependent interpolation and restriction operators

2D Convergence experiments





Summary of 2D iteration experiments I

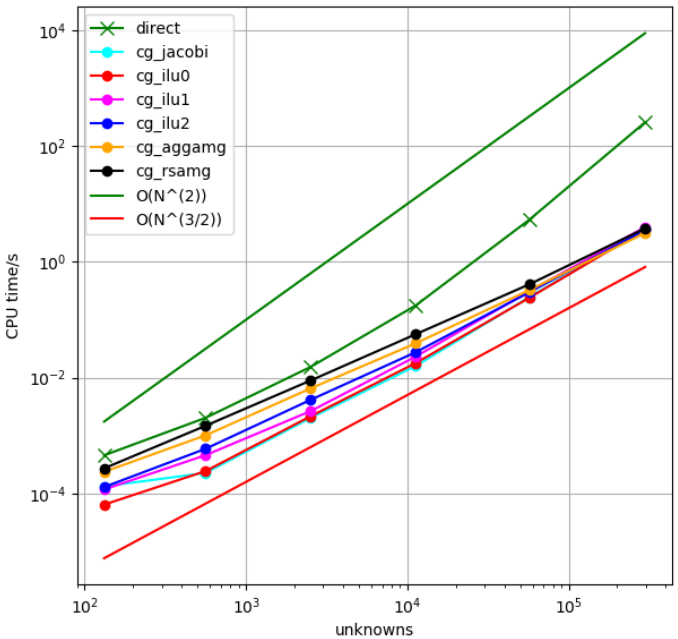
Let N be the number of unknowns and space dimension $d = 2$

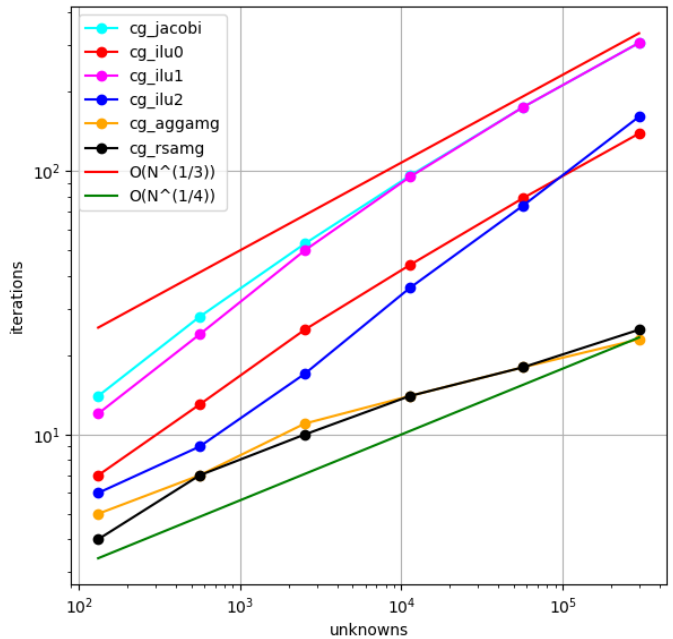
- In the estimates in lecture 7, based $\kappa(A) = O(h^{-2})$ the combination of CG and Jacobi preconditioner gives an effective iteration rate estimate of $\rho \sim 1 - O(h^{\delta})$ with $\delta = 1$
- $\Rightarrow$ Number of iteration steps $k_\rho = O(N^{\frac{\delta}{d}}) = O(N^{\frac{1}{2}})$
- $O(N)$ complexity of each iteration step (sparsity of matrix *and* preconditioner) $\Rightarrow$ overall complexity $O(N^{1+\frac{\delta}{d}}) = O(N^{\frac{3}{2}})$
- This is the same complexity estimate as for the direct solver
- Confirmed in the experiments:
  Jacobi method has this complexity scaling
- For ILU it is the same, just different prefactors which become smaller as we allow for more fill-in
- The direct solver performs better here than predicted and is an order of magnitude faster than most simple iterative schemes

Summary of 2D iteration experiments II

- Multigrid methods:
  - We observe a scaling of the iteration numbers with $O(N^{\frac{1}{4}}$ or better, ideally we would hope for $O(1)$, currently this is possible only on highly structured mesh hierarchies
  - Solution times on par with the direct solver (which is highly optimized)
- Conclusion: in 2D, well designed direct solvers are easy to use, and we can assume that the perform well up to 10^6 unknowns (... which should be checked for any particular application)
- On highly structured grid hierarchies, multigrid might "win".
- How about 3D ? Complexity scaling for PCG and for LU solve is $O(N^{\frac{4}{3}})$, but for LU factorization it is $O(N^2)$

3D Convergence experiments





Summary of 3D iteration experiments in Julia

- Direct solver is barely usable - it was not possible to run an example with 10^6 unknowns on this laptop (as it was in 2D)
- Predicted complexity of PCG methods seems to appear
- Algebraic multigrid saves iteration numbers, but due to inherent complexity they perform on par with the simpler preconditioners
- Potential for parallelization: simpler with simple preconditioners

Exam dates

Tentative dates

Feb 18	6 slots, are they needed ?
Feb 19 or Feb 21	2-3 slots if really necessary
March 4	
March 5	
March 17	
March 18	
March 19	

If these do not suffice, March 3, 16 would be possible in addition.

Inscription starts on Tuesday