

Parallel Numerics, WT 2017/2018

5 Iterative Methods for Sparse Linear Systems of Equations



Contents

- 1 Introduction
 - 1.1 Computer Science Aspects
 - 1.2 Numerical Problems
 - 1.3 Graphs
 - 1.4 Loop Manipulations
- 2 Elementary Linear Algebra Problems
 - 2.1 BLAS: Basic Linear Algebra Subroutines
 - 2.2 Matrix-Vector Operations
 - 2.3 Matrix-Matrix-Product
- 3 Linear Systems of Equations with Dense Matrices
 - 3.1 Gaussian Elimination
 - 3.2 Parallelization
 - 3.3 QR-Decomposition with Householder matrices
- 4 Sparse Matrices
 - 4.1 General Properties, Storage
 - 4.2 Sparse Matrices and Graphs
 - 4.3 Reordering
 - 4.4 Gaussian Elimination for Sparse Matrices
- 5 Iterative Methods for Sparse Linear Systems of Equations**
 - 5.1 Stationary Methods
 - 5.2 Nonstationary Methods
 - 5.3 Preconditioning
- 6 Domain Decomposition
 - 6.1 Overlapping Domain Decomposition
 - 6.2 Non-overlapping Domain Decomposition
 - 6.3 Schur Complements



- Disadvantages of direct methods (in parallel):
 - strongly sequential
 - may lead to dense matrices
 - sparsity pattern changes, additional entries necessary
 - indirect addressing
 - storage
 - computational effort



- Disadvantages of direct methods (in parallel):
 - strongly sequential
 - may lead to dense matrices
 - sparsity pattern changes, additional entries necessary
 - indirect addressing
 - storage
 - computational effort
- Iterative solver:
 - choose initial guess = starting vector $x^{(0)}$, e.g., $x^{(0)} = \mathbf{0}$
 - iteration function $x^{(k+1)} := \Phi(x^{(k)})$



- Disadvantages of direct methods (in parallel):
 - strongly sequential
 - may lead to dense matrices
 - sparsity pattern changes, additional entries necessary
 - indirect addressing
 - storage
 - computational effort
- Iterative solver:
 - choose initial guess = starting vector $x^{(0)}$, e.g., $x^{(0)} = \mathbf{0}$
 - iteration function $x^{(k+1)} := \Phi(x^{(k)})$
- Applied on solving a linear system:
 - Main part of Φ should be a matrix-vector multiplication Ax (matrix-free!?)
 - Easy to parallelize, no change in the pattern of A .

$$x^{(k)} \xrightarrow{k \rightarrow \infty} \bar{x} = A^{-1}b$$

- Main problem: Fast convergence!



5.1. Stationary Methods

5.1.1. Richardson Iteration



- Construct from $Ax = b$ an iteration process:

$$b = Ax = \left(\underbrace{A - I + I}_{\text{(artificial) splitting of } A} \right) x = x - (I - A)x \Rightarrow x = b + (I - A)x$$
$$= b + Nx$$



- Construct from $Ax = b$ an iteration process:

$$\begin{aligned} b = Ax &= (\underbrace{A - I + I})x = x - (I - A)x \Rightarrow x = b + (I - A)x \\ &\text{(artificial) splitting of } A \\ &= b + Nx \end{aligned}$$

- Leads to equation $x = \Phi(x)$ with $\Phi(x) := b + Nx$:

start: $x^{(0)}$;

$$x^{(k+1)} := \Phi(x^{(k)}) = b + Nx^{(k)} = b + (I - A)x^{(k)}$$



Richardson Iteration (cont.)

start: $x^{(0)}$;

$$x^{(k+1)} := \Phi(x^{(k)}) = b + Nx^{(k)} = b + (I - A)x^{(k)}$$

If $x^{(k)}$ is convergent, $x^{(k)} \rightarrow \tilde{x}$,

then

$$\tilde{x} = \Phi(\tilde{x}) = b + N\tilde{x} = b + (I - A)\tilde{x} \Rightarrow A\tilde{x} = b$$

and therefore it holds

$$x^{(k)} \rightarrow \tilde{x} = \bar{x} := A^{-1}b$$



Richardson Iteration (cont.)

start: $x^{(0)}$;

$$x^{(k+1)} := \Phi(x^{(k)}) = b + Nx^{(k)} = b + (I - A)x^{(k)}$$

If $x^{(k)}$ is convergent, $x^{(k)} \rightarrow \tilde{x}$,

then

$$\tilde{x} = \Phi(\tilde{x}) = b + N\tilde{x} = b + (I - A)\tilde{x} \Rightarrow A\tilde{x} = b$$

and therefore it holds

$$x^{(k)} \rightarrow \tilde{x} = \bar{x} := A^{-1}b$$

Residual-based formulation:

$$\begin{aligned} x^{(k+1)} &= \Phi(x^{(k)}) = b + (I - A)x^{(k)} = b + x^{(k)} - Ax^{(k)} \\ &= x^{(k)} + \underbrace{(b - Ax^{(k)})}_{r(x) = \text{residual}} = x^{(k)} + r(x^{(k)}) \end{aligned}$$



Convergence Analysis via Neumann Series

$$\begin{aligned}x^{(k)} &= b + Nx^{(k-1)} = b + N(b + Nx^{(k-2)}) = b + Nb + N^2x^{(k-2)} = \\ \dots &= b + Nb + N^2b + \dots + N^{k-1}b + N^kx^{(0)} = \\ &= \sum_{j=0}^{k-1} N^j b + N^kx^{(0)} = \left(\sum_{j=0}^{k-1} N^j \right) b + N^kx^{(0)}\end{aligned}$$

Special case $x^{(0)} = 0$:

$$x^{(k)} = \left(\sum_{j=0}^{k-1} N^j \right) b$$



Convergence Analysis via Neumann Series

$$\begin{aligned}
 x^{(k)} &= b + Nx^{(k-1)} = b + N(b + Nx^{(k-2)}) = b + Nb + N^2x^{(k-2)} = \\
 \dots &= b + Nb + N^2b + \dots + N^{k-1}b + N^kx^{(0)} = \\
 &= \sum_{j=0}^{k-1} N^j b + N^kx^{(0)} = \left(\sum_{j=0}^{k-1} N^j \right) b + N^kx^{(0)}
 \end{aligned}$$

Special case $x^{(0)} = 0$:

$$x^{(k)} = \left(\sum_{j=0}^{k-1} N^j \right) b$$

$$\begin{aligned}
 \Rightarrow x^{(k)} \in \text{span}\{b, Nb, N^2b, \dots, N^{k-1}b\} &= \text{span}\{b, Ab, A^2b, \dots, A^{k-1}b\} \\
 &= K_k(A, b)
 \end{aligned}$$

which is called the Krylov space to A and b .



Convergence Analysis via Neumann Series

$$\begin{aligned}
 x^{(k)} &= b + Nx^{(k-1)} = b + N(b + Nx^{(k-2)}) = b + Nb + N^2x^{(k-2)} = \\
 \dots &= b + Nb + N^2b + \dots + N^{k-1}b + N^kx^{(0)} = \\
 &= \sum_{j=0}^{k-1} N^j b + N^kx^{(0)} = \left(\sum_{j=0}^{k-1} N^j \right) b + N^kx^{(0)}
 \end{aligned}$$

Special case $x^{(0)} = 0$:

$$x^{(k)} = \left(\sum_{j=0}^{k-1} N^j \right) b$$

$$\begin{aligned}
 \Rightarrow x^{(k)} \in \text{span}\{b, Nb, N^2b, \dots, N^{k-1}b\} &= \text{span}\{b, Ab, A^2b, \dots, A^{k-1}b\} \\
 &= K_k(A, b)
 \end{aligned}$$

which is called the Krylov space to A and b .

For $\|N\| < 1$ holds:

$$\sum_{j=0}^{k-1} N^j \rightarrow \sum_{j=0}^{\infty} N^j = (I - N)^{-1} = (I - (I - A))^{-1} = A^{-1}$$



Convergence Analysis via Neumann Series (cont.)

$$x^{(k)} \rightarrow \left(\sum_{j=0}^{\infty} N^j \right) b = (I - N)^{-1} b = A^{-1} b = \bar{x}$$

Richardson iteration is convergent for $\|N\| < 1$ or $A \approx I$.



Convergence Analysis via Neumann Series (cont.)

$$x^{(k)} \rightarrow \left(\sum_{j=0}^{\infty} N^j \right) b = (I - N)^{-1} b = A^{-1} b = \bar{x}$$

Richardson iteration is convergent for $\|N\| < 1$ or $A \approx I$.

Error analysis for $e^{(k)} := x^{(k)} - \bar{x}$:

$$\begin{aligned} e^{(k+1)} &= x^{(k+1)} - \bar{x} = \Phi(x^{(k)}) - \Phi(\bar{x}) = (b + Nx^{(k)}) - (b + N\bar{x}) = \\ &= N(x^{(k)} - \bar{x}) = Ne^{(k)} \end{aligned}$$

$$\|e^{(k)}\| \leq \|N\| \|e^{(k-1)}\| \leq \|N\|^2 \|e^{(k-2)}\| \leq \dots \leq \|N\|^k \|e^{(0)}\|$$

$$\|N\| < 1 \Rightarrow \|N\|^k \xrightarrow{k \rightarrow \infty} 0 \Rightarrow \|e^{(k)}\| \xrightarrow{k \rightarrow \infty} 0$$



Convergence Analysis via Neumann Series (cont.)

$$x^{(k)} \rightarrow \left(\sum_{j=0}^{\infty} N^j \right) b = (I - N)^{-1} b = A^{-1} b = \bar{x}$$

Richardson iteration is convergent for $\|N\| < 1$ or $A \approx I$.

Error analysis for $e^{(k)} := x^{(k)} - \bar{x}$:

$$\begin{aligned} e^{(k+1)} &= x^{(k+1)} - \bar{x} = \Phi(x^{(k)}) - \Phi(\bar{x}) = (b + Nx^{(k)}) - (b + N\bar{x}) = \\ &= N(x^{(k)} - \bar{x}) = Ne^{(k)} \end{aligned}$$

$$\|e^{(k)}\| \leq \|N\| \|e^{(k-1)}\| \leq \|N\|^2 \|e^{(k-2)}\| \leq \dots \leq \|N\|^k \|e^{(0)}\|$$

$$\|N\| < 1 \Rightarrow \|N\|^k \xrightarrow{k \rightarrow \infty} 0 \Rightarrow \|e^{(k)}\| \xrightarrow{k \rightarrow \infty} 0$$

- Convergence, if $\rho(N) = \rho(I - A) < 1$, where ρ is spectral radius

$$\rho(N) = |\lambda_{\max}| = \max_i (|\lambda_i|) \quad (\lambda_i \text{ is eigenvalue of } N)$$

- Eigenvalues of A have to be all in circle around 1 with radius 1.



Splittings of A

- Convergence of Richardson only in very special cases!
Try to improve the iteration for better convergence!
- Write A in form $A := M - N$

$$b = Ax = (M - N)x = Mx - Nx \Leftrightarrow x = M^{-1}b + M^{-1}Nx = \Phi(x)$$

$$\begin{aligned}\Phi(x) &= M^{-1}b + M^{-1}Nx = M^{-1}b + M^{-1}(M - A)x = \\ &= M^{-1}(b - Ax) + x = x + M^{-1}r(x)\end{aligned}$$



Splittings of A

- Convergence of Richardson only in very special cases!
Try to improve the iteration for better convergence!
- Write A in form $A := M - N$

$$b = Ax = (M - N)x = Mx - Nx \Leftrightarrow x = M^{-1}b + M^{-1}Nx = \Phi(x)$$

$$\begin{aligned}\Phi(x) &= M^{-1}b + M^{-1}Nx = M^{-1}b + M^{-1}(M - A)x = \\ &= M^{-1}(b - Ax) + x = x + M^{-1}r(x)\end{aligned}$$

- N should be such that Ny can be evaluated efficiently.
- M should be such that $M^{-1}y$ can be evaluated efficiently.

$$x^{(k+1)} = x^{(k)} + M^{-1}r^{(k)}$$

- Iteration with splitting $M - N$ is equivalent to Richardson on

$$M^{-1}Ax = M^{-1}b$$



Convergence

- Iteration with splitting $A = M - N$ is convergent if

$$\rho(M^{-1}N) = \rho(I - M^{-1}A) < 1$$

- For fast convergence it should hold
 - $M^{-1}A \approx I$
 - $M^{-1}A$ should be better conditioned than A itself



Convergence

- Iteration with splitting $A = M - N$ is convergent if

$$\rho(M^{-1}N) = \rho(I - M^{-1}A) < 1$$

- For fast convergence it should hold
 - $M^{-1}A \approx I$
 - $M^{-1}A$ should be better conditioned than A itself
- Such a matrix M is called a preconditioner for A .
Is used in other iterative methods to accelerate convergence.
- Condition number:

$$\kappa(A) = \|A^{-1}\| \|A\|, \quad \left| \frac{\lambda_{\max}}{\lambda_{\min}} \right|, \quad \text{or} \quad \frac{\sigma_{\max}}{\sigma_{\min}}$$



Jacobi (Diagonal) Splitting (cont.)

Iteration process written elementwise:

$$x^{(k+1)} = D^{-1}(b - (A - D)x^{(k)}) \Rightarrow x_j^{(k+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{m=1, m \neq j}^n a_{j,m} x_m^{(k)} \right)$$

$$a_{jj} x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{j,m} x_m^{(k)} - \sum_{m=j+1}^n a_{j,m} x_m^{(k)}$$



Jacobi (Diagonal) Splitting (cont.)

Iteration process written elementwise:

$$x^{(k+1)} = D^{-1}(b - (A - D)x^{(k)}) \Rightarrow x_j^{(k+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{m=1, m \neq j}^n a_{j,m} x_m^{(k)} \right)$$

$$a_{jj} x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{j,m} x_m^{(k)} - \sum_{m=j+1}^n a_{j,m} x_m^{(k)}$$

- Damping or relaxation for improving convergence
- Idea: Iterative method as correction of last iterate in search direction.



Jacobi (Diagonal) Splitting (cont.)

Iteration process written elementwise:

$$x^{(k+1)} = D^{-1}(b - (A - D)x^{(k)}) \Rightarrow x_j^{(k+1)} = \frac{1}{a_{jj}} \left(b_j - \sum_{m=1, m \neq j}^n a_{j,m} x_m^{(k)} \right)$$

$$a_{jj} x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{j,m} x_m^{(k)} - \sum_{m=j+1}^n a_{j,m} x_m^{(k)}$$

- Damping or relaxation for improving convergence
- Idea: Iterative method as correction of last iterate in search direction.
- Introduce step length for this correction step:

$$x^{(k+1)} = x^{(k)} + D^{-1} r^{(k)} \quad \rightarrow \quad x^{(k+1)} = x^{(k)} + \omega D^{-1} r^{(k)}$$

with additional damping parameter ω .

- Damped Jacobi iteration:

$$x_{\text{damped}}^{(k+1)} = (\omega + 1 - \omega)x^{(k)} + \omega D^{-1} r^{(k)} = \omega x^{(k+1)} + (1 - \omega)x^{(k)}$$



Damped Jacobi Iteration

$$\begin{aligned}x^{(k+1)} &= x^{(k)} + \omega D^{-1} r^{(k)} = x^{(k)} + \omega D^{-1} (b - Ax^{(k)}) = \\ &= \dots \\ &= \omega D^{-1} b + [(1 - \omega)I + \omega D^{-1} (L + U)] x^{(k)}\end{aligned}$$

is convergent for

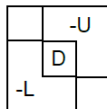
$$\rho(\underbrace{[(1 - \omega)I + \omega D^{-1} (L + U)]}_{\xrightarrow{\omega \rightarrow 0} I}) < 1$$

Look for optimal ω with best convergence (add. degree of freedom).



Parallelism in the Jacobi Iteration

- Jacobi method is easy to parallelize: only Ax and $D^{-1}x$.
- But often too slow convergence!
- Improvement: block Jacobi iteration



5.1.4. Gauss-Seidel Iteration

Always use newest information available!

Jacobi iteration:

$$a_{jj}x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{j,m} \underbrace{x_m^{(k)}}_{\text{already computed}} - \sum_{m=j+1}^n a_{j,m}x_m^{(k)}$$

Gauss-Seidel iteration:

$$a_{jj}x_j^{(k+1)} = b_j - \sum_{m=1}^{j-1} a_{j,m} \underbrace{x_m^{(k+1)}}_{\text{already computed}} - \sum_{m=j+1}^n a_{j,m}x_m^{(k)}$$



Gauss-Seidel Iteration (cont.)

- Compare dependency graphs for general iterative algorithms.
Here:

$$x = f(x) = D^{-1}(b + (D - A)x) = D^{-1}(b - (L + U)x)$$

to splitting $A = (D - L) - U = M - N$

$$\begin{aligned}x^{(k+1)} &= (D - L)^{-1}b + (D - L)^{-1}Ux^{(k)} = \\ &= (D - L)^{-1}b + (D - L)^{-1}(D - L - A)x^{(k)} = \\ &= x^{(k)} + (D - L)^{-1}r^{(k)}\end{aligned}$$

- Convergence depends on spectral radius $\rho(I - (D - L)^{-1}A) < 1$



Parallelism in the Gauss-Seidel Iteration

- Linear system in $D - L$ is easy to solve because $D - L$ is lower triangular but
- strongly sequential!
- Use red-black ordering or graph colouring for compromise:
parallel \leftrightarrow convergence



Successive Over Relaxation (SOR)

- Damping or relaxation:

$$x^{(k+1)} = x^{(k)} + \omega(D-L)^{-1}r^{(k)} = \omega(D-L)^{-1}b + [(1-\omega) + \omega(D-L)^{-1}U]x^{(k)}$$

- Convergence depends on spectral radius of iteration matrix

$$(1 - \omega) + \omega(D - L)^{-1}U$$

- Parallelization of SOR == parallelization of GS



Stationary Methods (in General)

- Can always be written in the two normal forms

$$x^{(k+1)} = c + Bx^{(k)}$$

with convergence depending on $\rho(B)$ of iteration matrix and

$$x^{(k+1)} = x^{(k)} + Fr^{(k)}$$

with preconditioner F , $B = I - FA$



Stationary Methods (in General)

- Can always be written in the two normal forms

$$x^{(k+1)} = c + Bx^{(k)}$$

with convergence depending on $\rho(B)$ of iteration matrix and

$$x^{(k+1)} = x^{(k)} + Fr^{(k)}$$

with preconditioner F , $B = I - FA$

- For $x^{(0)} = \mathbf{0}$:

$$x^{(k+1)} \subseteq K_k(B, c),$$

which is the Krylov space with respect to matrix B and vector c .

- Slow convergence (but good smoothing properties! \rightarrow multigrid)



MATLAB Example

- `clear; n=100;k=10;omega=1;stationary`
- `tridiag(-.5, 1, -.5):`
 - Jacobi norm $|\cos(\pi/n)|$
 - GS norm $\cos(\pi/n)^2$
 - both $< 1 \rightarrow$ convergence, but slow
- To improve convergence \rightarrow nonstationary methods (or multigrid)



Chair of Informatics V—SCCS

Efficient Numerical Algorithms—Parallel & HPC

- High-dimensional numerics (sparse grids)
- Fast iterative solvers (multi-level methods, preconditioners, eigenvalue solvers)
- Uncertainty Quantification
- Space-filling curves
- Numerical linear algebra
- Numerical algorithms for image processing
- HW-aware numerical programming

Fields of application in simulation

- CFD (incl. fluid-structure interaction)
- Plasma physics
- Molecular dynamics
- Quantum chemistry

Further info → www5.in.tum.de

Feel free to come around and ask for thesis topics!



5.2. Nonstationary Methods

5.2.1. Gradient Method

- Consider $A = A^T > 0$ (A SPD)

$$\text{Function } \Phi(x) = \frac{1}{2}x^T Ax - b^T x$$

- n-dim. paraboloid $\mathbb{R}^n \rightarrow \mathbb{R}$
- Gradient $\nabla\Phi(x) = Ax - b$
- Position with $\nabla\Phi(x) = 0$ is exactly minimum of paraboloid



5.3. Nonstationary Methods

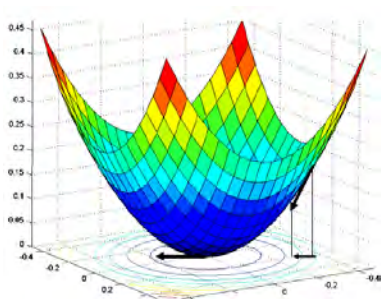
5.3.1. Gradient Method

- Consider $A = A^T > 0$ (A SPD)

$$\text{Function } \Phi(x) = \frac{1}{2}x^T Ax - b^T x$$

- n -dim. paraboloid $\mathbb{R}^n \rightarrow \mathbb{R}$
- Gradient $\nabla\Phi(x) = Ax - b$
- Position with $\nabla\Phi(x) = 0$ is exactly minimum of paraboloid
- Instead of solving $Ax = b$ consider $\min_x \Phi(x)$
- Local descent direction in y :
 $\nabla\Phi(x) \cdot y$ is minimum for

$$y = -\nabla\Phi(x)$$



Gradient Method (cont.)

- Optimization: start with $x^{(0)}$

$$x^{(k+1)} := x^{(k)} + \alpha_k d^{(k)}$$

with search direction $d^{(k)}$ and step size α_k .

- In view of previous results the optimal (local) search direction is

$$-\nabla\Phi(x^{(k)}) =: d^{(k)}$$



Gradient Method (cont.)

- Optimization: start with $x^{(0)}$

$$x^{(k+1)} := x^{(k)} + \alpha_k d^{(k)}$$

with search direction $d^{(k)}$ and step size α_k .

- In view of previous results the optimal (local) search direction is

$$-\nabla\Phi(x^{(k)}) =: d^{(k)}$$

- To define α_k :

$$\min_{\alpha} g(\alpha) := \min_{\alpha} (\Phi(x^{(k)} + \alpha(b - Ax^{(k)})))$$

$$= \min_{\alpha} \left(\frac{1}{2} (x^{(k)} + \alpha d^{(k)})^T A (x^{(k)} + \alpha d^{(k)}) - b^T (x^{(k)} + \alpha d^{(k)}) \right)$$

$$= \min_{\alpha} \left(\frac{1}{2} \alpha^2 d^{(k)T} A d^{(k)} - \alpha d^{(k)T} d^{(k)} + \frac{1}{2} x^{(k)T} A x^{(k)} - x^{(k)T} b \right)$$

$$\alpha_k = \frac{d^{(k)T} d^{(k)}}{d^{(k)T} A d^{(k)}}$$

$$d^{(k)} = -\nabla\Phi(x^{(k)}) = b - Ax^{(k)}$$



Gradient Method (cont. 2)

$$x^{(k+1)} = x^{(k)} + \frac{\|b - Ax^{(k)}\|_2^2}{(b - Ax^{(k)})^T A (b - Ax^{(k)})} (b - Ax^{(k)})$$

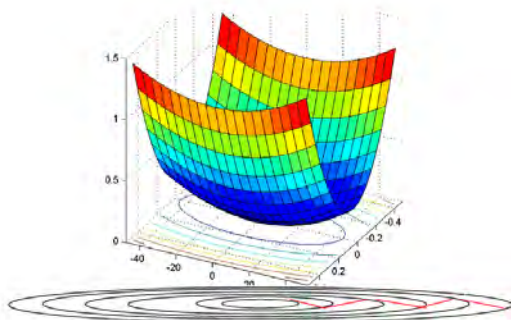
- Method of steepest descent.



Gradient Method (cont. 2)

$$x^{(k+1)} = x^{(k)} + \frac{\|b - Ax^{(k)}\|_2^2}{(b - Ax^{(k)})^T A (b - Ax^{(k)})} (b - Ax^{(k)})$$

- Method of steepest descent.
- Disadvantage: Distorted contour lines.
- Slow convergence (zig zag path)
- Local descent direction is not globally optimal



Analysis of the Gradient Method

- Definition A -norm:

$$\|x\|_A := \sqrt{x^T A x}$$

- Consider error:

$$\begin{aligned}\|x - \bar{x}\|_A^2 &= \|x - A^{-1}b\|_A^2 = (x^T - b^T A^{-1})A(x - A^{-1}b) \\ &= x^T A x - 2b^T x + b^T A^{-1}b \\ &= 2\Phi(x) + b^T A^{-1}b\end{aligned}$$



Analysis of the Gradient Method

- Definition A -norm:

$$\|x\|_A := \sqrt{x^T A x}$$

- Consider error:

$$\begin{aligned}\|x - \bar{x}\|_A^2 &= \|x - A^{-1}b\|_A^2 = (x^T - b^T A^{-1})A(x - A^{-1}b) \\ &= x^T A x - 2b^T x + b^T A^{-1}b \\ &= 2\Phi(x) + b^T A^{-1}b\end{aligned}$$

- Therefore, minimizing Φ is equivalent to minimizing the error in the A -norm! More detailed analysis reveals:

$$\|x^{(k+1)} - \bar{x}\|_A^2 \leq \left(1 - \frac{1}{\kappa(A)}\right) \cdot \|x^{(k)} - \bar{x}\|_A^2$$

- Therefore, for $\kappa(A) \gg 1$ very slow convergence!



5.3.2. The Conjugate Gradient Method

- Improving descent direction being globally optimal.
- $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ with search direction not being negative gradient, but projection of gradient that is A -conjugate to all previous search directions:

$$p^{(k)} \perp Ap^{(j)} \text{ for all } j < k \text{ or}$$

$$p^{(k)} \perp_A p^{(j)} \text{ or}$$

$$p^{(k)T} Ap^{(j)} = 0 \text{ for } j < k$$



5.3.3. The Conjugate Gradient Method

- Improving descent direction being globally optimal.
- $x^{(k+1)} := x^{(k)} + \alpha_k p^{(k)}$ with search direction not being negative gradient, but projection of gradient that is A -conjugate to all previous search directions:

$$p^{(k)} \perp Ap^{(j)} \text{ for all } j < k \text{ or}$$

$$p^{(k)} \perp_A p^{(j)} \text{ or}$$

$$p^{(k)T} Ap^{(j)} = 0 \text{ for } j < k$$

- We choose new search direction as component of last residual that is A -conjugate to all previous search directions.
- α_k again by 1-dim. minimization as before (for chosen $p^{(k)}$)



The Conjugate Gradient Algorithm

$$p^{(0)} = r^{(0)} = b - Ax^{(0)}$$

for $k = 1, 2, \dots$ do

$$\alpha^{(k)} = -\frac{\langle r^{(k)}, r^{(k)} \rangle}{\langle p^{(k)}, Ap^{(k)} \rangle}$$

$$x^{(k+1)} = x^{(k)} - \alpha^{(k)} p^{(k)}$$

$$r^{(k+1)} = r^{(k)} + \alpha^{(k)} Ap^{(k)}$$

if $\|r^{(k+1)}\|_2^2 \leq \epsilon$ **then break**

$$\beta^{(k)} = \frac{\langle r^{(k+1)}, r^{(k+1)} \rangle}{\langle r^{(k)}, r^{(k)} \rangle}$$

$$p^{(k+1)} = r^{(k+1)} + \beta^{(k)} p^{(k)}$$



Properties of Conjugate Gradients

- It holds

$$p^{(j)T} A p^{(k)} = 0 = r^{(j)T} r^{(k)} \quad \text{for } j \neq k$$

- and

$$\begin{aligned} \text{span}(p^{(1)}, \dots, p^{(j)}) &= \text{span}(r^{(0)}, \dots, r^{(j-1)}) = \\ &= \text{span}(r^{(0)}, A r^{(0)}, \dots, A^{j-1} r^{(0)}) = K_j(A, r^{(0)}) \end{aligned}$$

- Especially for $x^{(0)} = \mathbf{0}$ it holds

$$K_j(A, r^{(0)}) = \text{span}(b, Ab, \dots, A^{j-1} b)$$



Properties of Conjugate Gradients

- It holds

$$p^{(j)T} A p^{(k)} = 0 = r^{(j)T} r^{(k)} \quad \text{for } j \neq k$$

- and

$$\begin{aligned} \text{span}(p^{(1)}, \dots, p^{(j)}) &= \text{span}(r^{(0)}, \dots, r^{(j-1)}) = \\ &= \text{span}(r^{(0)}, A r^{(0)}, \dots, A^{j-1} r^{(0)}) = K_j(A, r^{(0)}) \end{aligned}$$

- Especially for $x^{(0)} = \mathbf{0}$ it holds

$$K_j(A, r^{(0)}) = \text{span}(b, Ab, \dots, A^{j-1} b)$$

- $x^{(k)}$ is best approximate solution to $Ax = b$ in subspace $K_k(A, r^{(0)})$. For $x^{(0)} = \mathbf{0}$: $x^{(k)} \in K_k(A, b)$

- Error:

$$\|x^{(k)} - \bar{x}\|_A = \min_{x \in K_k(A, b)} \|x - \bar{x}\|_A$$

- Cheap 1-dim. minimization gives optimal k -dim. solution for free!



Properties of Conjugate Gradients (cont.)

- Consequence: After n steps $K_n(A, b) = \mathbb{R}^n$ and therefore $x^{(n)} = A^{-1}b$ is solution in exact arithmetic.
- Unfortunately, this is not true in floating point arithmetic.
- Furthermore, $\mathcal{O}(n)$ iteration steps would be too costly:
costs: #iterations * matrix-vector-product



Properties of Conjugate Gradients (cont.)

- Consequence: After n steps $K_n(A, b) = \mathbb{R}^n$ and therefore $x^{(n)} = A^{-1}b$ is solution in exact arithmetic.
- Unfortunately, this is not true in floating point arithmetic.
- Furthermore, $\mathcal{O}(n)$ iteration steps would be too costly:
costs: #iterations * matrix-vector-product
- Matrix-vector-product easy in parallel.
- But, how to get fast convergence and reduce #iterations?



Error Estimation ($x^{(0)} = 0$)

$$\begin{aligned}
 \|e^{(k)}\|_A &= \|x^{(k)} - \bar{x}\|_A = \min_{x \in K_k(A,b)} \|x - \bar{x}\|_A = \\
 &= \min_{\alpha_0, \dots, \alpha_{k-1}} \left\| \sum_{j=0}^{k-1} \alpha_j (A^j b) - \bar{x} \right\|_A = \\
 &= \min_{\mathcal{P}^{(k-1)}(x)} \left\| \mathcal{P}^{(k-1)}(A)b - \bar{x} \right\|_A = \\
 &= \min_{\mathcal{P}^{(k-1)}(x)} \left\| \mathcal{P}^{(k-1)}(A)A\bar{x} - \bar{x} \right\|_A = \\
 &= \min_{\mathcal{P}^{(k-1)}(x)} \left\| (\mathcal{P}^{(k-1)}(A)A - I)(\bar{x} - x^{(0)}) \right\|_A = \\
 &= \min_{\mathcal{Q}^{(k)}(x), \mathcal{Q}^{(k)}(0)=1} \left\| \mathcal{Q}^{(k)}(A)e^{(0)} \right\|_A
 \end{aligned}$$

for polynomial $\mathcal{Q}^{(k)}(x)$ of degree k with $\mathcal{Q}^{(k)}(0) = 1$.



Error Estimation

- Matrix A has orthonormal basis of eigenvectors u_j , $j = 1, \dots, n$, eigenvalues λ_j

- It holds

$$Au_j = \lambda_j u_j, \quad j = 1, \dots, n, \quad \text{and} \quad u_j^T u_k = 0 \quad \text{for} \quad j \neq k \quad \text{and} \quad 1 \quad \text{for} \quad j = k$$

- Start error in ONB: $e^{(0)} = \sum_{j=1}^n \rho_j u_j$



Error Estimation

- Matrix A has orthonormal basis of eigenvectors u_j , $j = 1, \dots, n$, eigenvalues λ_j
- It holds

$$Au_j = \lambda_j u_j, \quad j = 1, \dots, n, \quad \text{and} \quad u_j^T u_k = 0 \quad \text{for} \quad j \neq k \quad \text{and} \quad 1 \quad \text{for} \quad j = k$$

- Start error in ONB: $e^{(0)} = \sum_{j=1}^n \rho_j u_j$

$$\begin{aligned} \|e^{(k)}\|_A &= \min_{Q^{(k)}(0)=1} \left\| Q^{(k)}(A) \sum_{j=1}^n \rho_j u_j \right\|_A = \min_{Q^{(k)}(0)=1} \left\| \sum_{j=1}^n \rho_j Q^{(k)}(A) u_j \right\|_A = \\ &= \min_{Q^{(k)}(0)=1} \left\| \sum_{j=1}^n \rho_j Q^{(k)}(\lambda_j) u_j \right\|_A \leq \min_{Q^{(k)}(0)=1} \left\{ \max_{j=1, \dots, n} |Q^{(k)}(\lambda_j)| \right\} \left\| \sum_{j=1}^n \rho_j u_j \right\|_A = \\ &= \min_{Q^{(k)}(0)=1} \left\{ \max_{j=1, \dots, n} |Q^{(k)}(\lambda_j)| \right\} \|e^{(0)}\|_A \end{aligned}$$



Error Estimates

By choosing polynomials with $Q^{(k)}(0) = 1$, we can derive error estimates for the error after the k th step:

$$\text{Choose, e.g. } Q^{(k)}(x) := \left| 1 - \frac{2}{\lambda_{\max} + \lambda_{\min}} x \right|^k$$

Error Estimates

By choosing polynomials with $Q^{(k)}(0) = 1$, we can derive error estimates for the error after the k th step:

$$\text{Choose, e.g. } Q^{(k)}(x) := \left| 1 - \frac{2}{\lambda_{\max} + \lambda_{\min}} x \right|^k$$

$$\begin{aligned} \|e^{(k)}\|_A &\leq \max_{j=1, \dots, n} |Q^{(k)}(\lambda_j)| \|e^{(0)}\|_A = \left| 1 - \frac{2\lambda_{\max}}{\lambda_{\max} + \lambda_{\min}} \right|^k \|e^{(0)}\|_A \\ &= \left(\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\max} + \lambda_{\min}} \right)^k \|e^{(0)}\|_A = \left(\frac{\kappa(A) - 1}{\kappa(A) + 1} \right)^k \|e^{(0)}\|_A \end{aligned}$$



Better Estimates

- Choose normalized Chebyshev polynomials

$$T_n(x) = \cos(n \arccos(x))$$

$$\|e^{(k)}\|_A \leq \frac{1}{T_k\left(\frac{\kappa(A)+1}{\kappa(A)-1}\right)} \|e^{(0)}\|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \|e^{(0)}\|_A$$



Better Estimates

- Choose normalized Chebyshev polynomials

$$T_n(x) = \cos(n \arccos(x))$$

$$\|e^{(k)}\|_A \leq \frac{1}{T_k\left(\frac{\kappa(A)+1}{\kappa(A)-1}\right)} \|e^{(0)}\|_A \leq 2 \left(\frac{\sqrt{\kappa(A)} - 1}{\sqrt{\kappa(A)} + 1} \right)^k \|e^{(0)}\|_A$$

- For clustered eigenvalues choose special polynomial, e.g. assume that A has only two eigenvalues λ_1 and λ_2 :

$$Q^{(2)}(x) := \frac{(\lambda_1 - x)(\lambda_2 - x)}{\lambda_1 \lambda_2}$$

$$\|e^{(2)}\|_A \leq \max_{j=1,2} |Q^{(2)}(\lambda_j)| \|e^{(0)}\|_A = 0$$

Convergence of CG after two steps!



Outliers/Cluster

Assume the matrix has an eigenvalue $\lambda_1 > 1$ and all other eigenvalues are contained in an ϵ -neighborhood of 1:

$$\forall \lambda \neq \lambda_1 : |\lambda - 1| < \epsilon$$

$$Q^{(2)}(x) := \frac{(\lambda_1 - x)(1 - x)}{\lambda_1}$$

$$\|e^{(2)}\|_A \leq \max_{|\lambda-1|<\epsilon} \left| \frac{(\lambda_1 - \lambda)(1 - \lambda)}{\lambda_1} \right| \|e^{(0)}\|_A \leq \frac{(\lambda_1 - 1 + \epsilon)\epsilon}{\lambda_1} = \mathcal{O}(\epsilon)$$

Very good approximation of CG after only two steps!

Important: small number of outliers combined with cluster.



Conjugate Gradients Summary

- To get fast convergence and reduce the number of iterations:
→ find preconditioner M , such that $M^{-1}Ax = M^{-1}b$ with clustered eigenvalues.
- Conjugate gradients (CG) is always the method of choice for symmetric positive definite A (in general).
- To improve convergence, include preconditioning (PCG).
- CG has two important properties: optimal and cheap.



Parallel Conjugate Gradients Algorithm

```

while ( sqrt(gamma) > epsilon * error_0 ) {
  if (iteration > 1)
    q = r + gamma / gamma_old * q;
  MVP → v = A * q;
  delta = dot(v,q);
  alpha = delta / gamma;
  x = x + alpha * q;
  r = r - alpha * v;
  gamma_old = gamma;
  gamma = dot(r,r);
  iteration = iteration + 1;
}

```

MVP

scalar

local vector

synchronisation
points

dot product: requires communication (MPI_ALLREDUCE)



Non-Blocking Collective Operations

- Use to hide communication!
- Allows overlap of numerical computations and communications.
- In MPI-1/MPI-2 only possible for point-to-point communication:
MPI_Isend and MPI_Irecv.

Additional libraries necessary for collective operations!
Example: LibNBC (non-blocking collectives)

- Are included in new MPI-3 standard.



Example: Pseudocode for Nonblock. Reduct.

```
MPI_Request req;
int sbuf1[SIZE], rbuf1[SIZE], buf2[SIZE];
// compute sbuf1
compute(sbuf1, SIZE);

// start non-blocking allreduce of sbuf1
// computation and communication overlap
MPI_Iallreduce(sbuf1,rbuf1,SIZE,MPI_INT,MPI_SUM,MPI_COMM, &req);

// compute buf2 (independent of sbuf1)
compute(buf2, SIZE);

// synchronization
MPI_WAIT(&req, &stat);

// use data in rbuf1; final computation
evaluate(rbuf1, buf2, SIZE);
```



Iter. Methods for general (nonsymmetric) A : BiCGSTAB

- Applicable if little memory at hand and A^T not available.
- Computational costs per iteration similar to BiCG and CGS.
- Alternative for CGS that avoids irregular convergence patterns of CGS maintaining similar convergence speed.
- Less loss of accuracy in the updated residual.



Iter. Methods for general (nonsymmetric) A : GMRES

- Leads to smallest residual for fixed number of iteration steps, but these steps become increasingly expensive.
- To limit increasing storage requirements and work per iteration step, restarting is necessary. When to do so depends on A and b ; it requires skill and experience.



Iter. Methods for general (nonsymmetric) A : GMRES

- Leads to smallest residual for fixed number of iteration steps, but these steps become increasingly expensive.
- To limit increasing storage requirements and work per iteration step, restarting is necessary. When to do so depends on A and b ; it requires skill and experience.
- Requires only matrix-vector products with the coeff. matrix.
- Number of inner products grows linearly with iteration number (up to restart point).
- Implementation based on Gram-Schmidt \rightarrow inner products independent \rightarrow only one synchronization point.
Using modified Gram-Schmidt \rightarrow one synchronization point per inner product.

We consider GMRES in the following.



5.3.4. GMRES

- Iterative solution method for general A
- Consider small subspace U_m and determine optimal approximate solution for $Ax = b$ in U_m . Hence x is of the form $x := U_m y$

$$\min_{x \in U_m} \|Ax - b\|_2 = \min_y \|A(U_m y) - b\|_2$$

- Can be solved by normal equations: $(U_m^T A^T A U_m) y = U_m^T A^T b$



5.3.5. GMRES

- Iterative solution method for general A
- Consider small subspace U_m and determine optimal approximate solution for $Ax = b$ in U_m . Hence x is of the form $x := U_m y$

$$\min_{x \in U_m} \|Ax - b\|_2 = \min_y \|A(U_m y) - b\|_2$$

- Can be solved by normal equations: $(U_m^T A^T A U_m) y = U_m^T A^T b$
- Try to find sequence of "good" subspaces $U_1 \rightarrow U_2 \rightarrow U_3 \rightarrow \dots$ such that iteratively we can update the optimal solutions

$$x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow \dots \rightarrow A^{-1} b$$

using mainly matrix-vector products.



GMRES: Subspace

What subspace for fast convergence and easy computations?



GMRES: Subspace

What subspace for fast convergence and easy computations?

$$U_m := K_m(A, b) = \text{span}(b, Ab, \dots, A^{m-1}b) \quad (\text{Krylov space})$$

Problem: b, Ab, A^2b, \dots is bad basis for this subspace!

So we need a first step to compute a good basis for U_m :



GMRES: Subspace

What subspace for fast convergence and easy computations?

$$U_m := K_m(A, b) = \text{span}(b, Ab, \dots, A^{m-1}b) \quad (\text{Krylov space})$$

Problem: b, Ab, A^2b, \dots is bad basis for this subspace!

So we need a first step to compute a good basis for U_m :

Start with $u_1 := \frac{b}{\|b\|_2}$ and do for $j = 2 : m$:

$$\begin{aligned} \tilde{u}_j &:= Au_{j-1} - \sum_{k=1}^{j-1} (u_k^T Au_{j-1}) u_k = Au_{j-1} - \sum_{k=1}^{j-1} h_{k,j-1} u_k \\ u_j &:= \frac{\tilde{u}_j}{\|\tilde{u}_j\|_2} = \frac{\tilde{u}_j}{h_{j,j-1}} \end{aligned}$$

which is the standard orthogonalization method (Arnoldi method)



Matrix Form of Arnoldi ONB

$$U_m = \text{span}(b, Ab, \dots, A^{m-1}b) = \text{span}(u_1, u_2, \dots, u_m) \quad (\text{ONB})$$

Write this orthogonalization method in matrix form

$$Au_{j-1} = \sum_{k=1}^{j-1} h_{k,j-1} u_k + \tilde{u}_j = \sum_{k=1}^j h_{k,j-1} u_k$$



Matrix Form of Arnoldi ONB

$$U_m = \text{span}(b, Ab, \dots, A^{m-1}b) = \text{span}(u_1, u_2, \dots, u_m) \quad (\text{ONB})$$

Write this orthogonalization method in matrix form

$$AU_{j-1} = \sum_{k=1}^{j-1} h_{k,j-1} u_k + \tilde{u}_j = \sum_{k=1}^j h_{k,j-1} u_k$$

$$AU_m = A(u_1 \quad \dots \quad u_m) = (u_1 \quad \dots \quad u_{m+1}) \tilde{H}_{m+1,m} = U_{m+1} \tilde{H}_{m+1,m}$$

$$\tilde{H}_{m+1,m} = \begin{pmatrix} h_{11} & \dots & \dots & h_{1m} \\ h_{21} & \ddots & & \vdots \\ 0 & \ddots & \ddots & \vdots \\ & & \ddots & h_{mm} \\ 0 & & 0 & h_{m+1,m} \end{pmatrix}$$



GMRES: Minimization

This leads to minimization problem

$$\begin{aligned}\min_{x \in U_m} \|Ax - b\| &= \min_y \|AU_m y - b\| \\ &= \min_y \left\| U_{m+1} \tilde{H}_{m+1,m} y - \|b\| u_1 \right\| \\ &= \min_y \left\| U_{m+1} (\tilde{H}_{m+1,m} y - \|b\| e_1) \right\| \\ &= \min_y \left\| \tilde{H}_{m+1,m} y - \|b\| e_1 \right\|\end{aligned}$$

Because U_{m+1} is part of an orthogonal matrix.



GMRES: QR

Use Givens matrices to compute a QR-factorization of the upper Hessenberg matrix $\tilde{H}_{m+1,m}$.

$$G_1 \cdot \begin{pmatrix} * & \cdots & * \\ * & \ddots & \vdots \\ & \ddots & * \\ & & * \end{pmatrix} = \begin{pmatrix} * & \cdots & * \\ 0 & \ddots & \vdots \\ & \ddots & * \\ & & * \end{pmatrix} =$$

$$G_m \cdots G_2 G_1 \cdot \tilde{H}_{m+1,m} = Q \cdot \tilde{H}_{m+1,m} = \begin{pmatrix} R_m \\ 0 \end{pmatrix}$$

QR via Givens matrices is a simplified version of QR via Householder matrices.



GMRES

$$\begin{aligned}\min_{x \in K_m} \|Ax - b\| &= \min_y \left\| \tilde{H}_{m+1,m} y - \|b\| e_1 \right\| \\ &= \min_y \left\| Q^T R y - \|b\| e_1 \right\| = \min_y \|R y - \|b\| Q e_1\| \\ &= \min_y \left\| \begin{pmatrix} R_m \\ 0 \end{pmatrix} y - \tilde{b} \right\| = \min_y \left\| \begin{pmatrix} R_m y - \tilde{b}_m \\ -\tilde{\beta}_m \end{pmatrix} \right\|\end{aligned}$$

Solution:

$$R_m y = \tilde{b}_m, \quad x_m = U_m y$$



GMRES Algorithm

- GMRES is a clever implementation of this sequence of least squares problems.
- Computing step by step for increasing m
 - next Au_m
 - enlarged $H_{m+1,m}$ by Arnoldi orthogonalization (gives new u_{m+1})
 - next Givens matrix G_m
 - update triangular solves to get next y_m and x_m .



GMRES Algorithm

- GMRES is a clever implementation of this sequence of least squares problems.
- Computing step by step for increasing m
 - next Au_m
 - enlarged $H_{m+1,m}$ by Arnoldi orthogonalization (gives new u_{m+1})
 - next Givens matrix G_m
 - update triangular solves to get next y_m and x_m .
- Disadvantage: large Hessenberg matrices!
- Therefore, use restarted GMRES, e.g. GMRES(20).



GMRES Algorithm: First Step

$$\text{Start: } b, u_1 := \frac{b}{\|b\|}$$

$$h_{2,1}u_2 = Au_1 - h_{1,1}u_1, \quad H_{2,1} = \begin{pmatrix} h_{1,1} \\ h_{2,1} \end{pmatrix}, \quad G_1 H_{2,1} = \begin{pmatrix} \nu_{1,1} \\ 0 \end{pmatrix},$$



GMRES Algorithm: First Step

$$\text{Start: } b, u_1 := \frac{b}{\|b\|}$$

$$h_{2,1}u_2 = Au_1 - h_{1,1}u_1, \quad H_{2,1} = \begin{pmatrix} h_{1,1} \\ h_{2,1} \end{pmatrix}, \quad G_1 H_{2,1} = \begin{pmatrix} \nu_{1,1} \\ 0 \end{pmatrix},$$

$$\begin{aligned} \|Ax - b\| &= \|AU_1 y_1 - b\| = \|H_{2,1} y_1 - \|b\| e_1\| \\ &= \left\| G_1^T \begin{pmatrix} \nu_{1,1} \\ 0 \end{pmatrix} y_1 - \|b\| e_1 \right\| = \left\| \begin{pmatrix} \nu_{1,1} \\ 0 \end{pmatrix} y_1 - \|b\| G_1 e_1 \right\| \end{aligned}$$



GMRES Algorithm: First Step

$$\text{Start: } b, u_1 := \frac{b}{\|b\|}$$

$$h_{2,1}u_2 = Au_1 - h_{1,1}u_1, \quad H_{2,1} = \begin{pmatrix} h_{1,1} \\ h_{2,1} \end{pmatrix}, \quad G_1 H_{2,1} = \begin{pmatrix} \nu_{1,1} \\ 0 \end{pmatrix},$$

$$\begin{aligned} \|Ax - b\| &= \|AU_1 y_1 - b\| = \|H_{2,1} y_1 - \|b\| e_1\| \\ &= \left\| G_1^T \begin{pmatrix} \nu_{1,1} \\ 0 \end{pmatrix} y_1 - \|b\| e_1 \right\| = \left\| \begin{pmatrix} \nu_{1,1} \\ 0 \end{pmatrix} y_1 - \|b\| G_1 e_1 \right\| \end{aligned}$$

$$y_1 = \frac{(G_1 e_1 \|b\|)_1}{\nu_{1,1}}, \quad x_1 = u_1 y_1 = U_1 y_1$$



GMRES Algorithm: Second Step

$$h_{3,2}u_3 = Au_2 - h_{2,2}u_2 - h_{1,2}u_1, \quad H_{3,2} = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \\ 0 & h_{32} \end{pmatrix},$$

$$G_1 H_{3,2} = \begin{pmatrix} \nu_1 & * \\ 0 & * \\ 0 & * \end{pmatrix}, \quad G_2 G_1 H_{3,2} = \begin{pmatrix} \nu_{1,1} & \nu_{1,2} \\ 0 & \nu_{2,2} \\ 0 & 0 \end{pmatrix}$$



GMRES Algorithm: Second Step

$$h_{3,2}u_3 = Au_2 - h_{2,2}u_2 - h_{1,2}u_1, \quad H_{3,2} = \begin{pmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \\ 0 & h_{32} \end{pmatrix},$$

$$G_1 H_{3,2} = \begin{pmatrix} \nu_1 & * \\ 0 & * \\ 0 & * \end{pmatrix}, \quad G_2 G_1 H_{3,2} = \begin{pmatrix} \nu_{1,1} & \nu_{1,2} \\ 0 & \nu_{2,2} \\ 0 & 0 \end{pmatrix}$$

$$\begin{aligned} \|Ax - b\| &= \|AU_2 y_2 - b\| = \|H_{3,2} y_2 - \|b\| e_1\| \\ &= \left\| G_1^T G_2^T \begin{pmatrix} \nu_{1,1} & \nu_{1,2} \\ 0 & \nu_{2,2} \\ 0 & 0 \end{pmatrix} y_2 - \|b\| e_1 \right\| = \left\| \begin{pmatrix} \nu_{1,1} & \nu_{1,2} \\ 0 & \nu_{2,2} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} y_{2,1} \\ y_{2,2} \end{pmatrix} - \|b\| G_2 G_1 e_1 \right\| \end{aligned}$$

$$y_{2,2} = \frac{(G_2 G_1 e_1 \|b\|)_2}{\nu_{2,2}}, \quad y_{2,1} = \frac{(G_2 G_1 e_1 \|b\|)_1 - \nu_{1,2} y_{2,2}}{\nu_{1,1}}$$

$$x_2 = U_2 y_2 = u_1 y_{2,1} + u_2 y_{2,2} \quad \text{and so forth...}$$



GMRES Error (A diagonalisable)

$$\begin{aligned}
 \|r_m\| &= \|Ax_m - b\| = \min_{x \in U_m} \|Ax - b\| \\
 &= \min_{\alpha_1, \dots, \alpha_m} \left\| A \left(\sum_{j=0}^{m-1} \alpha_j A^j b \right) - b \right\| = \min_{\mathcal{P}^{(m-1)}} \|A \mathcal{P}^{(m-1)}(A)b - b\| \\
 &= \min_{Q^{(m)}(0)=1} \|Q^{(m)}(A)b\| = \min_{Q^{(m)}(0)=1} \|V Q^{(m)}(\Lambda) V^{-1} b\| \\
 &= \min_{Q^{(m)}(0)=1} \|V \operatorname{diag}(Q^{(m)}(\lambda_j)) V^{-1} b\| \leq \\
 &\leq \kappa_2(V) \cdot \min_{Q^{(m)}(0)=1} \left[\max_{j=1, \dots, n} |Q^{(m)}(\lambda_j)| \right] \|r_0\|
 \end{aligned}$$



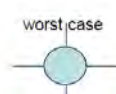
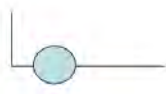
Iterative Methods: Survey

- Basic iterative methods (not considering multigrid!)
- SPD A : PCG (preconditioned conjugate gradient method)
- symmetric indefinite A : MINRES, SYMMLQ, (PCG on normal equations)
- general A :
 - GMRES (optimal like CG, but expensive)
 - BiCG, BiCGSTAB, CGS (not optimal, but cheap, instable?)
 - QMR (quasi optimal, cheap)



Iterative Methods: Survey

- Basic iterative methods (not considering multigrid!)
- SPD A : PCG (preconditioned conjugate gradient method)
- symmetric indefinite A : MINRES, SYMMLQ, (PCG on normal equations)
- general A :
 - GMRES (optimal like CG, but expensive)
 - BiCG, BiCGSTAB, CGS (not optimal, but cheap, instable?)
 - QMR (quasi optimal, cheap)
- Essential for convergence is position of eigenvalues (singular values)
- Fast convergence for well conditioned problems or matrices with clustered spectrum



5.4. Preconditioning

- Direct solvers: sequential, losing sparsity
- Iterative solvers: easy parallel and sparse, but possibly slowly convergent

5.5. Preconditioning

- Direct solvers: sequential, losing sparsity
- Iterative solvers: easy parallel and sparse, but possibly slowly convergent
- Combination of both methods (2 variants):
 1. Include preconditioner $M \approx A$ in the form $M^{-1}Ax = M^{-1}b$, such that
 - M is easy to deal with in parallel (reduced approximate direct solver)
 - spectrum of $M^{-1}A$ is much better clustered
 2. Or include preconditioner $M \approx A^{-1}$ in the form $MAx = Mb$, such that
 - M is easy to deal with in parallel (reduced approximate inverse)
 - Spectrum of MA is much better clustered



5.6. Preconditioning

- Direct solvers: sequential, losing sparsity
- Iterative solvers: easy parallel and sparse, but possibly slowly convergent
- Combination of both methods (2 variants):
 1. Include preconditioner $M \approx A$ in the form $M^{-1}Ax = M^{-1}b$, such that
 - M is easy to deal with in parallel (reduced approximate direct solver)
 - spectrum of $M^{-1}A$ is much better clustered
 2. Or include preconditioner $M \approx A^{-1}$ in the form $MAx = Mb$, such that
 - M is easy to deal with in parallel (reduced approximate inverse)
 - Spectrum of MA is much better clustered
- General both-sided preconditioning:

$$Ax = b \Leftrightarrow MA(Ky) = Mb \text{ and } Ky = x$$



Overview of Following Preconditioners

- Stationary preconditioners
- ILU preconditioner

Overview of Following Preconditioners

- Stationary preconditioners
- ILU preconditioner
- Polynomial preconditioners
- Sparse approximate inverse preconditioners:
 - SPAI for the general nonsymmetric case
 - FSPAI for the SPD case
 - MSPAI using SPAI with probing

5.6.1. Stationary Preconditioners

Consider preconditioner $M \approx A$ in the form $M^{-1}Ax = M^{-1}b$.

- Stationary iteration to splitting $A = M - N$. Convergence depends on

$$\|I - M^{-1}A\| < 1$$

- That is exactly a condition for a good preconditioner: spectrum clustered around 1, $M^{-1}A \approx I$



5.6.2. Stationary Preconditioners

Consider preconditioner $M \approx A$ in the form $M^{-1}Ax = M^{-1}b$.

- Stationary iteration to splitting $A = M - N$. Convergence depends on

$$\|I - M^{-1}A\| < 1$$

- That is exactly a condition for a good preconditioner: spectrum clustered around 1, $M^{-1}A \approx I$
- If splitting leads to fast convergence, than M is also a good preconditioner.
- In this sense, PCG with stationary preconditioner M can be seen as an acceleration of the stationary method with splitting $M - N$.



Stationary Preconditioners (cont.)

- Jacobi splitting with $D = \text{diag}(A)$ gives Jacobi preconditioner $M := D$.
- Gauss-Seidel splitting $M = D - L$ leads to GS preconditioner.



Stationary Preconditioners (cont.)

- Jacobi splitting with $D = \text{diag}(A)$ gives Jacobi preconditioner $M := D$.
- Gauss-Seidel splitting $M = D - L$ leads to GS preconditioner.
- Relaxation:

$$x^{(k,\text{new})} := (1 - \omega)x^{(k-1)} + \omega x^{(k)}$$

As convex combination of old and new iterate (Jacobi or GS)



Stationary Preconditioners (cont.)

- Jacobi splitting with $D = \text{diag}(A)$ gives Jacobi preconditioner $M := D$.
- Gauss-Seidel splitting $M = D - L$ leads to GS preconditioner.
- Relaxation:

$$x^{(k,\text{new})} := (1 - \omega)x^{(k-1)} + \omega x^{(k)}$$

As convex combination of old and new iterate (Jacobi or GS)

- Symmetrization: first iteration with preconditioner M , second iteration with M^T .

$$M_{\text{new}} := M + M^T - M^T A M$$



Stationary Preconditioners (cont.)

- Jacobi splitting with $D = \text{diag}(A)$ gives Jacobi preconditioner $M := D$.
- Gauss-Seidel splitting $M = D - L$ leads to GS preconditioner.
- Relaxation:

$$x^{(k,\text{new})} := (1 - \omega)x^{(k-1)} + \omega x^{(k)}$$

As convex combination of old and new iterate (Jacobi or GS)

- Symmetrization: first iteration with preconditioner M , second iteration with M^T .

$$M_{\text{new}} := M + M^T - M^T A M$$

- Special case: damped GS \rightarrow SSOR:

$$M_{\text{new}} = \frac{1}{2 - \omega} \left(\frac{1}{\omega} D - L \right) \left(\frac{1}{\omega} D \right)^{-1} \left(\frac{1}{\omega} D - L \right)^T$$



5.6.3. ILU Preconditioner, Sequentially



ILU Preconditioner, Sequentially

- Apply Gaussian elimination algorithm, but only on allowed pattern \Rightarrow incomplete LU factorization called ILU.
- Reduce in the `for`-loops the indices to the indices with
 - allowed pattern, e.g. ILU(0) for pattern of A
 - values that are not too small, ILUT for ILU with threshold



ILU Preconditioner, Sequentially

- Apply Gaussian elimination algorithm, but only on allowed pattern \Rightarrow incomplete LU factorization called ILU.
- Reduce in the `for`-loops the indices to the indices with
 - allowed pattern, e.g. ILU(0) for pattern of A
 - values that are not too small, ILUT for ILU with threshold
- Leads to approximate LU factorization

$$A = LU + R, \quad \text{preconditioner } M = LU$$

with all ignored fill-in entries collected in R .



ILU Preconditioner, Sequentially

- Apply Gaussian elimination algorithm, but only on allowed pattern \Rightarrow incomplete LU factorization called ILU.
- Reduce in the `for`-loops the indices to the indices with
 - allowed pattern, e.g. ILU(0) for pattern of A
 - values that are not too small, ILUT for ILU with threshold
- Leads to approximate LU factorization

$$A = LU + R, \quad \text{preconditioner } M = LU$$

with all ignored fill-in entries collected in R .

- MILU (modified ILU): collect all ignored fill-in entries on the related main diagonal elements \rightarrow maintains the row sum or the action on $(1, 1, \dots, 1)^T$.



5.6.4. Parallel ILU

Consider iterative solution of the system $A = LU$ on given pattern \mathcal{S} . So L and U are sparse with pattern \mathcal{S} , and the equations have to be satisfied only on the pattern \mathcal{S} :

$$A_{i,j} = \sum_{k, (i,k) \in \mathcal{S}, (k,j) \in \mathcal{S}} L_{i,k} U_{k,j} \text{ for } (i,j) \in \mathcal{S}$$



5.6.5. Parallel ILU

Consider iterative solution of the system $A = LU$ on given pattern \mathcal{S} . So L and U are sparse with pattern \mathcal{S} , and the equations have to be satisfied only on the pattern \mathcal{S} :

$$A_{i,j} = \sum_{k, (i,k) \in \mathcal{S}, (k,j) \in \mathcal{S}} L_{i,k} U_{k,j} \text{ for } (i,j) \in \mathcal{S}$$

Use $A = LU$ as fixed point iteration in the form:

- freeze L , compute new U
- freeze U , compute new L
- repeat until convergence



5.6.6. Parallel ILU

Consider iterative solution of the system $A = LU$ on given pattern \mathcal{S} . So L and U are sparse with pattern \mathcal{S} , and the equations have to be satisfied only on the pattern \mathcal{S} :

$$A_{i,j} = \sum_{k, (i,k) \in \mathcal{S}, (k,j) \in \mathcal{S}} L_{i,k} U_{k,j} \text{ for } (i,j) \in \mathcal{S}$$

Use $A = LU$ as fixed point iteration in the form:

- freeze L , compute new U
- freeze U , compute new L
- repeat until convergence

$$l_{i,j} = \frac{1}{u_{j,j}} (a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} u_{k,j}) \quad (1)$$

$$u_{i,j} = \frac{1}{l_{i,i}} (a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} u_{k,j}) . \quad (2)$$



5.6.7. Iterative ILU Algorithm

Set unknowns $l_{i,j}$ and $u_{i,j}$ to initial values

FOR sweep = 1,2,... until convergence

 WHILE $(i,j) \in S$

 IF $i > j$

$$l_{i,j} = \frac{1}{u_{j,j}} (a_{i,j} - \sum_{k=1}^{j-1} l_{i,k} u_{k,j})$$

 ELSE

$$u_{i,j} = a_{i,j} - \sum_{k=1}^{i-1} l_{i,k} u_{k,j}$$

 END

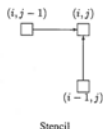
 END

END

5.6.8. Direct Triangular Solves

Remaining problem: Solve $Lx = b$ in parallel!

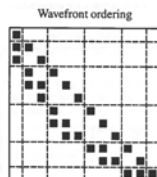
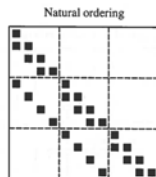
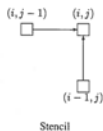
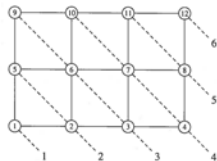
- There exist parallel direct solvers based on graph coloring, but with restricted parallelism.
- Using data dependency graph:



5.6.9. Direct Triangular Solves

Remaining problem: Solve $Lx = b$ in parallel!

- There exist parallel direct solvers based on graph coloring, but with restricted parallelism.
- Using data dependency graph:



- Collect indices that can be directly derived: LEVEL 1
- From data dependency graph collect indices depending only on the first level: LEVEL 2
- and so on.

Unknowns in the same level can be updated in parallel. Compare colored GS, Cuthill-McKee and multifrontal solvers!

5.6.10. Iterative Triangular Solves

Resort: Jacobi iteration for $b = Lx = (D + L_0)x$ resulting in the iteration

$$Dx^{k+1} = b - L_0x^k, k = 0, 1, 2, \dots$$



5.6.11. Iterative Triangular Solves

Resort: Jacobi iteration for $b = Lx = (D + L_0)x$ resulting in the iteration

$$Dx^{k+1} = b - L_0x^k, k = 0, 1, 2, \dots$$

Nice feature: Iteration matrix L_0 has zero eigenvalues. Therefore convergence guaranteed via special norm:

$$\|e_k\| \leq \|L_0\|^k \|e_0\|$$

and there exists a norm with $\|L_0\| < 1$. But often slow convergence!

Improvement: Block-Jacobi.

Search for iterative fast converging and fully parallel methods! Later on more!

Overview Explicit Preconditioners

- ILU and stationary methods use approximations on A itself.
- The resulting preconditioners are given by triangular matrices L , that have to be solved in each iteration step: $L^{-1}x^{(k)}$!
- Jacobi easy to parallelize, but slow convergence.



Overview Explicit Preconditioners

- ILU and stationary methods use approximations on A itself.
- The resulting preconditioners are given by triangular matrices L , that have to be solved in each iteration step: $L^{-1}x^{(k)}$!
- Jacobi easy to parallelize, but slow convergence.
- Question: How to derive preconditioners that lead to fast convergence and are easy to parallelize?
- Until now considered variant 1.; In the following variant 2.:
Find approximations on $M \approx A^{-1}$. Then the solution of the linear system given by the preconditioner, is only $Mx^{(k)}$, a matrix vector product!



Parallel Preconditioning

- Find preconditioner M , that satisfies three conditions:
 - (i) The computation of M is fast in parallel
 - (ii) The application Mx in each iteration step is easy in parallel
 - (iii) The spectrum AM or MA is clustered \rightarrow fast convergence



Parallel Preconditioning

- Find preconditioner M , that satisfies three conditions:
 - (i) The computation of M is fast in parallel
 - (ii) The application Mx in each iteration step is easy in parallel
 - (iii) The spectrum AM or MA is clustered \rightarrow fast convergence
- Examples:
 - For GS is (i) ok, but not (ii)
 - For Jacobi (i) and (ii) ok, but not (iii)
 - For ILU (iii) ok, (i) also iteratively, but not (ii)
- Note that preconditioners also have to be well defined! Zero on diagonals?



5.6.12. Polynomial Preconditioners

- Characteristic polynomial for A :

$$0 = q(A) = \gamma_n A^n + \gamma_{n-1} A^{n-1} + \dots + \gamma_1 A + \gamma_0 I$$

Gives polynomial representation for A^{-1} ($\gamma_0 \neq 0$):

$$A^{-1} = \frac{1}{\gamma_0} (-\gamma_n A^{n-1} - \gamma_{n-1} A^{n-2} - \dots - \gamma_1 I) = p(A)$$

- Therefore, it makes sense to approximate A^{-1} by a polynomial.



5.6.13. Polynomial Preconditioners

- Characteristic polynomial for A :

$$0 = q(A) = \gamma_n A^n + \gamma_{n-1} A^{n-1} + \dots + \gamma_1 A + \gamma_0 I$$

Gives polynomial representation for A^{-1} ($\gamma_0 \neq 0$):

$$A^{-1} = \frac{1}{\gamma_0} (-\gamma_n A^{n-1} - \gamma_{n-1} A^{n-2} - \dots - \gamma_1 I) = p(A)$$

- Therefore, it makes sense to approximate A^{-1} by a polynomial.
- Better approximation by finding region $S \in \mathbb{R}$ or \mathbb{C} that contains nearly all eigenvalues, and then find polynomial p that is near the inverse in S :

$$\min_{p_n} \|P(A)A - I\|, \quad \min_{p_n(x)} \left(\max_{\lambda \text{ eigenvalue}} |p(\lambda)\lambda - 1| \right)$$

- Solution: Normalized Chebyshev polynomials.



5.6.14. Polynomial Preconditioners

- Characteristic polynomial for A :

$$0 = q(A) = \gamma_n A^n + \gamma_{n-1} A^{n-1} + \dots + \gamma_1 A + \gamma_0 I$$

Gives polynomial representation for A^{-1} ($\gamma_0 \neq 0$):

$$A^{-1} = \frac{1}{\gamma_0} (-\gamma_n A^{n-1} - \gamma_{n-1} A^{n-2} - \dots - \gamma_1 I) = p(A)$$

- Therefore, it makes sense to approximate A^{-1} by a polynomial.
- Better approximation by finding region $S \in \mathbb{R}$ or \mathbb{C} that contains nearly all eigenvalues, and then find polynomial p that is near the inverse in S :

$$\min_{p_n} \|P(A)A - I\|, \quad \min_{p_n(x)} \left(\max_{\lambda \text{ eigenvalue}} |p(\lambda)\lambda - 1| \right)$$

- Solution: Normalized Chebyshev polynomials.
- + Advant. of pol. prec.: better in parallel (mtx-vec product, BLAS 2)
- Disadvantage: non-optimal approximation in Krylov subspace



5.6.15. Sparse Approximate Inverses

- Other approach for approximating A^{-1} by norm minimization

$$\min_{M \in \mathcal{P}} \|AM - I\| \quad \text{over some sparsity pattern } \mathcal{P}.$$

5.6.16. Sparse Approximate Inverses

- Other approach for approximating A^{-1} by norm minimization

$$\min_{M \in \mathcal{P}} \|AM - I\| \quad \text{over some sparsity pattern } \mathcal{P}.$$

- Choice of the norm?
 - analytic (to allow the explicit solution of this problem)
 - easy to compute (in parallel)



5.6.17. Sparse Approximate Inverses

- Other approach for approximating A^{-1} by norm minimization

$$\min_{M \in \mathcal{P}} \|AM - I\| \quad \text{over some sparsity pattern } \mathcal{P}.$$

- Choice of the norm?
 - analytic (to allow the explicit solution of this problem)
 - easy to compute (in parallel)
- Optimal norm is Frobenius norm:

$$\|A\|_F^2 := \sum_{i,j=1}^n a_{i,j}^2 = \sum_{j=1}^n \|A_{\bullet,j}\|^2 = \text{trace}(A^T A)$$



SPAI in Parallel

- First, we choose pattern \mathcal{P} in a static way a priori, e.g. as the pattern of A

$$\min_{M \in \mathcal{P}} \|AM - I\|_F^2 = \min_{M \in \mathcal{P}} \sum_{k=1}^n \|(AM - I)e_k\|_2^2 = \sum_{k=1}^n \min_{M_k \in \mathcal{P}_k} \|AM_k - e_k\|_2^2$$



SPAI in Parallel

- First, we choose pattern \mathcal{P} in a static way a priori, e.g. as the pattern of A

$$\min_{M \in \mathcal{P}} \|AM - I\|_F^2 = \min_{M \in \mathcal{P}} \sum_{k=1}^n \|(AM - I)e_k\|_2^2 = \sum_{k=1}^n \min_{M_k \in \mathcal{P}_k} \|AM_k - e_k\|_2^2$$

- Hence, to minimize the Frobenius norm, we have to solve n least squares problems in the sparse columns of M !
- This can be done fully in parallel!
- But costs for least squares problems?



SPAI and Least Squares

$$\min_{M_k \in \mathcal{P}_k} \|AM_k - e_k\|_2^2 = \min_{M_k \in \mathcal{P}_k} \left\| A \begin{pmatrix} 0 \\ * \\ 0 \\ * \\ 0 \\ 0 \end{pmatrix} - e_k \right\|_2^2 = \min_{M_k \in \mathcal{P}_k} \|A(:, \mathcal{J}_k)M_k(\mathcal{J}_k) - e_k\|_2^2$$



SPAI and Least Squares

$$\min_{M_k \in \mathcal{P}_k} \|AM_k - e_k\|_2^2 = \min_{M_k \in \mathcal{P}_k} \left\| A \begin{pmatrix} 0 \\ * \\ 0 \\ * \\ 0 \\ 0 \\ 0 \end{pmatrix} - e_k \right\|_2^2 = \min_{M_k \in \mathcal{P}_k} \|A(:, \mathcal{J}_k)M_k(\mathcal{J}_k) - e_k\|_2^2$$

Denote by \mathcal{J}_k the set of allowed indices in the k th column of M .

$$A \cdot \begin{matrix} \star \\ \star \end{matrix} - \begin{matrix} \star \end{matrix} = A(:, \mathcal{J}_k) \cdot M(\mathcal{J}_k) - e_k = \begin{matrix} \text{red bar} \end{matrix} \cdot \begin{matrix} \star \end{matrix} - \begin{matrix} \star \end{matrix}$$

SPAI and Sparse Least Squares

- $A(:, \mathcal{J}_k)$ is a sparse rectangular matrix. \mathcal{I}_k is index set called *shadow* of \mathcal{J}_k .
- Delete superfluous zeros in least squares problem:
 - For index set \mathcal{J}_k in M_k keep only $A(:, \mathcal{J}_k)$
 - In $A(:, \mathcal{J}_k)$ keep only nonzero rows $A(\mathcal{I}_k, \mathcal{J}_k)$



SPAI and Sparse Least Squares

- $A(:, \mathcal{J}_k)$ is a sparse rectangular matrix. \mathcal{I}_k is index set called *shadow* of \mathcal{J}_k .
- Delete superfluous zeros in least squares problem:
 - For index set \mathcal{J}_k in M_k keep only $A(:, \mathcal{J}_k)$
 - In $A(:, \mathcal{J}_k)$ keep only nonzero rows $A(\mathcal{I}_k, \mathcal{J}_k)$



- Hence, reduction of sparse LS to $A(\mathcal{I}_k, \mathcal{J}_k)$:

$$\min_{M_k \in \mathcal{P}_k} \|A(\mathcal{I}_k, \mathcal{J}_k)M_k - e(\mathcal{I}_k)\|_2^2 = \min_{M_k \in \mathcal{P}_k} \|\tilde{A}\tilde{M}_k - \tilde{e}_k\|_2^2$$

- Solve small LS problem by Householder QR for

$$A(\mathcal{I}_k, \mathcal{J}_k) \rightarrow \tilde{M}_k \rightarrow M_k$$

Sparsity Pattern: Static A Priori Choice

- A^{-1} will be no more sparse!
- As a priori choice of a good approximate sparsity pattern for M we can choose the pattern of
 - A^k or $(A^T)^k$
 - $(A^T A)^k A^T$ for some $k = 1, 2$
 - A_ϵ with sparsified A
 - a combination of above
- A_ϵ by sparsification of A : delete all entries with $|A_{ij}| < \epsilon$



Sparsity Pattern: Dynamic Pattern Finding

- Start with thin approximate pattern \mathcal{J}_k for M_k
- Compute optimal column $M_{k,\text{opt}}(\mathcal{J}_k)$ by least squares



Sparsity Pattern: Dynamic Pattern Finding

- Start with thin approximate pattern \mathcal{J}_k for M_k
- Compute optimal column $M_{k,\text{opt}}(\mathcal{J}_k)$ by least squares
- Find new entry j for M_k such that $M_{k,\text{opt}}(\mathcal{J}_k) + \lambda e_j$ has smaller residual in the Frobenius norm.

$$\begin{aligned}\min_{M_k \in \mathcal{P}_k} \|A(M_k + \lambda e_j) - e_k\|_2^2 &= \min_{M_k \in \mathcal{P}_k} \|(AM_k - e_k) + \lambda A e_j\|_2^2 = \\ &= \min \left(\|r_k\|_2^2 + 2\lambda(r_k^T A_j) + \lambda^2 \|A_j\|_2^2 \right)\end{aligned}$$

For each possible index candidate j compute

$$\lambda_j = -\frac{r_k^T A_j}{\|A_j\|_2^2}$$



Sparsity Pattern: Dynamic Pattern Finding

- Start with thin approximate pattern \mathcal{J}_k for M_k
- Compute optimal column $M_{k,\text{opt}}(\mathcal{J}_k)$ by least squares
- Find new entry j for M_k such that $M_{k,\text{opt}}(\mathcal{J}_k) + \lambda e_j$ has smaller residual in the Frobenius norm.

$$\begin{aligned} \min_{M_k \in \mathcal{P}_k} \|A(M_k + \lambda e_j) - e_k\|_2^2 &= \min_{M_k \in \mathcal{P}_k} \|(AM_k - e_k) + \lambda A e_j\|_2^2 = \\ &= \min \left(\|r_k\|_2^2 + 2\lambda(r_k^T A_j) + \lambda^2 \|A_j\|_2^2 \right) \end{aligned}$$

For each possible index candidate j compute

$$\lambda_j = -\frac{r_k^T A_j}{\|A_j\|_2^2}$$

Choose index j with $r_k^T A_j \neq 0$ and $j = \arg \min(\lambda_j)$ since

$$\min = \|r_k\|_2^2 - \frac{(r_k^T A_j)^2}{\|A_j\|_2^2}.$$



Error Estimates

- Assume that for all columns holds

$$\|r_k\| = \|AM_k - e_k\| < \epsilon$$

- Then

$$\|AM - I\|_F \leq \sqrt{n}\epsilon$$

$$\|AM - I\|_2 \leq \sqrt{n}\epsilon$$

$$\|AM - I\|_1 \leq \sqrt{p}\epsilon$$

with p being the maximum number of nonzero entries in r_k ,
 $k = 1, \dots, n$.



Overview Following SPAI Variants

1. Block SPAI
2. Iterative SPAI
3. Factorized SPAI (FSPA)
4. Modified SPAI (MSPA)



Further Variants of SPAI

1. Block SPAI

- Partition the matrix M in small blocks (e.g., 2×2 or 3×3) and apply the Frobenius norm minimization with blockwise pattern.
- Advantages:
 - Underlying block structure will also appear in the pattern of $A^{-1} \rightarrow$ improved pattern
 - Block operations are more efficient
 - Less least squares problems to solve



Further Variants of SPAI

1. Block SPAI

- Partition the matrix M in small blocks (e.g., 2×2 or 3×3) and apply the Frobenius norm minimization with blockwise pattern.
- Advantages:
 - Underlying block structure will also appear in the pattern of $A^{-1} \rightarrow$ improved pattern
 - Block operations are more efficient
 - Less least squares problems to solve

2. Iterative SPAI

- Start with pattern of $A \rightarrow M_1$
- Construct M_2 relative to new matrix AM_1
- Construct M_3 relative to new matrix AM_1M_2
- ...
- Advantage: cheaper (but inferior approximation)



3. Factorized SPAI, FSPAI

- Parallel Preconditioner for SPD matrices
- Approximate the inverse Cholesky factor of $A = L_A^T L_A$

$$A^{-1} = (L_A^T L_A)^{-1} = L_A^{-1} L_A^{-T} \approx L L^T$$

$$L_A^{-1} \approx L \quad \rightarrow \quad \min \|L_A L - I\|_F$$

- Normal equations give $A(\mathcal{J}_k, \mathcal{J}_k)L(\mathcal{J}_k) = \alpha \mathbf{e}_k$



3. Factorized SPAI, FSPA

- Parallel Preconditioner for SPD matrices
- Approximate the inverse Cholesky factor of $A = L_A^T L_A$

$$A^{-1} = (L_A^T L_A)^{-1} = L_A^{-1} L_A^{-T} \approx L L^T$$

$$L_A^{-1} \approx L \quad \rightarrow \quad \min \|L_A L - I\|_F$$

- Normal equations give $A(\mathcal{J}_k, \mathcal{J}_k)L(\mathcal{J}_k) = \alpha \mathbf{e}_k$

4. Modified SPAI, MSPAI: Combining Targeting and Probing in classical SPAI formulation.

