

## 4.4. Gaussian Elimination for Sparse Matrices

### 4.4.1. Algebraic Pivoting in GE

- Numerical pivoting: for eliminating elements in column  $k$  choose large(st) entry in column/row/block  $k$  and permute this element on the diagonal position.
- Disadvantage: may lead to large fill in in the sparsity pattern of  $A$ .
- Idea: Choose pivot element according to minimum fill in! Note that for well-conditioned  $A = A^T > 0$  no numerical pivoting is necessary.
- Heuristic: Choose pivot element according to the degree in graph  
→ minimum degree reordering



## Special Case $A = A^T$

- For elimination in the  $k$ th column of  $A$ :
  - Define  $r_m :=$  number of nonzero entries in row  $m$
  - Choose pivot index  $i$  by  $r_i = \min_m r_m$
  - Do the pivot permutation and the elimination
  - Go to next column  $k$
- $r_m$  is #nonzeros in the  $m$ th row = #vertices directly connected with vertex  $m$   
Hence, pivot vertex is vertex with minimum degree in  $G(A_k)$
- Heuristics: few entries in  $m$ th row/column  $\rightarrow$  few fill in because
  - only few elements to eliminate
  - the pivot row used in the elimination is very sparse $\rightarrow$  Multiple minimum degree reordering



## Multiple Minimum Degree reordering MMD

- Often, many vertices may have the minimum degree. Which one should be chosen?
- Choose an independent set: Indices with the same minimum degree, but that are no neighbors.
  - Eliminating any will have no effect on the degree of the others, hence we can eliminate in Gaussian Elimination process in parallel (compare Multifrontal methods)
  - It reduces also the runtime because the outer loop is less than  $n$ .
  - It often leads to fewer nonzeros.
- Approximative Minimum Degree Reordering (AMD) uses approximations of the degrees.



# Generalization to Nonsymmetric Problems: Markowitz

- Define  $r_m := \text{nnz in row } m$ ;  $c_p := \text{nnz in column } p$
- Choose pivot element with index pair  $(i, j)$  such that

$$(r_i - 1)(c_j - 1) = \min_{m,p} (r_m - 1)(c_p - 1)$$

- Heuristics:
  - small  $c_j$  leads to few elimination steps
  - small  $r_i$  leads to sparse pivot row used in the elimination.
- Special case  $r_i = 1$  or  $c_j = 1$ : no fill in.
- Include numerical pivoting by applying algebraic pivoting only on indices with absolute value that is not too small, e.g.,

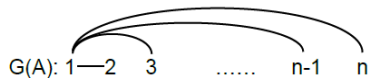
$$|a_{i,j}| \geq 0.1 \cdot \max_{r,s} |a_{r,s}|$$



# Comparison

Example:

$$A = \begin{pmatrix} * & * & * & \dots & * \\ * & * & & & \\ * & & * & & \\ \vdots & & & \ddots & \\ * & & & & * \end{pmatrix}$$



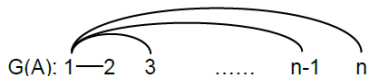
First elimination step leads to dense matrix!

$$\begin{pmatrix} * & * & * & \dots & * \\ 0 & * & * & \dots & * \\ 0 & * & * & \dots & * \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & * & * & \dots & * \end{pmatrix}$$



## Comparison (cont.)

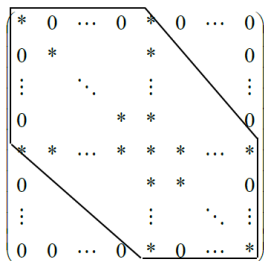
Cuthill McKee:



With starting vertex 1:

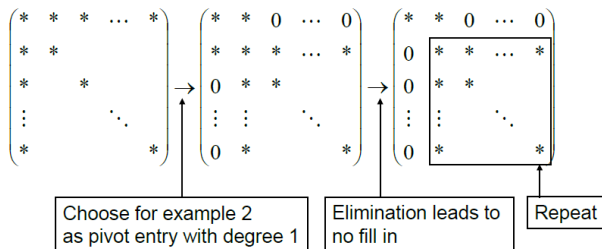
keeps numbers unchanged  $\rightarrow$  no improvement

Optimal bandwidth is not satisfactory: bandwidth  $\frac{n}{2}$



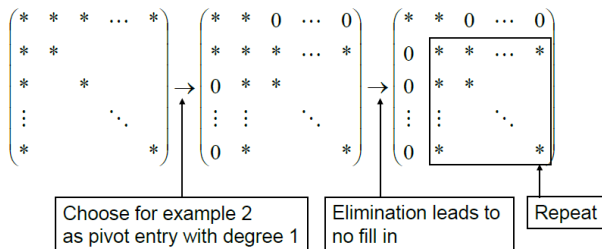
## Comparison (cont. 2)

Minimum degree is efficient for this example ( $PAP^T$ ):



## Comparison (cont. 2)

Minimum degree is efficient for this example ( $PAP^T$ ):



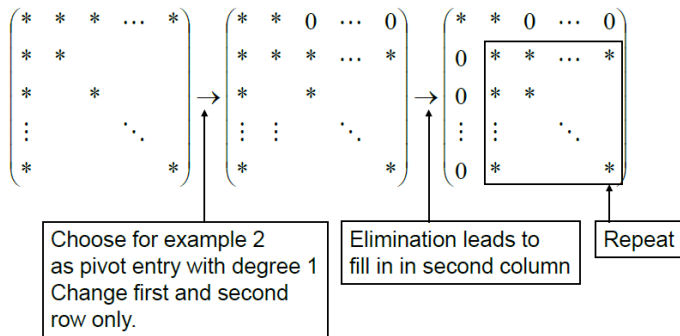
Optimal reordering in one step:  $1 \leftrightarrow n$       Optimal costs:  $\mathcal{O}(n)$ .

$$\begin{pmatrix} * & 0 & 0 & \dots & * \\ 0 & * & & \dots & * \\ 0 & & * & & \vdots \\ \vdots & \vdots & \vdots & \ddots & * \\ * & * & \dots & * & * \end{pmatrix}$$



# Example: Nonsymmetric Permutation

Costs:  $\mathcal{O}(n^2)$



Test: MATLAB (symamd, colamd, amd, colperm, symrcm)

```
load('west0479.mat'); a=west0479; s=a'*a; p=symamd(s); spy(s(p,p));
```

See matrix 'west0479.mat' on Matrix Market:

<http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/chemwest/west0479.html>



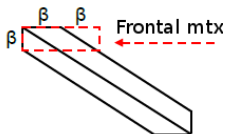
## 4.4.2. Gaussian Elimination in Graph

- Gaussian elimination can be modelled without numerical computations only algebraically by computing the sequence of related graphs (in terms of dense subgraphs (matrices) = clique)
- Modification of GE:
  1. Apply algebraic prestep for GE, determining the graphs related to the elimination matrices  $A_k$  in GE.
  2. Based on these graphs we can decide
    - whether GE will lead to nearly dense matrices (do not use GE in this case!)
    - what additional entries will appear during GE (prepare the storage)
- Algebraic prestep is cheap, can be implemented using cliques.



### 4.4.3. A Parallel Direct Solver

Frontal methods for band matrices (from PDE)



- Frontal dense matrix of size  $(\beta + 1) \times (2\beta + 1)$
- Move frontal matrix to CPU and treat as dense matrix.
- Then move frontal matrix one entry down-right and do next elimination.
- Repeat until done.
- No parallelism until now.

## Multifrontal in Parallel

To make efficient use of parallelism search for “independent” elimination steps!

$$A = \begin{pmatrix} * & 0 & * & * \\ 0 & * & * & * \\ * & * & * & 0 \\ * & * & 0 & * \end{pmatrix}$$

$A_{1,1}$  as first pivot element is related to a first frontal matrix that contains all information to eliminate the first column:

Dense submatrix for  $k = 1$ :

$$\begin{array}{ccc} A_{1,1} & A_{1,3} & A_{1,4} \\ A_{3,1} & \frac{A_{3,1} \cdot A_{1,3}}{A_{1,1}} & \frac{A_{3,1} \cdot A_{1,4}}{A_{1,1}} \\ A_{4,1} & \frac{A_{4,1} \cdot A_{1,3}}{A_{1,1}} & \frac{A_{4,1} \cdot A_{1,4}}{A_{1,1}} \end{array}$$



## Multifrontal in Parallel (cont.)

Because  $A_{1,2} = A_{2,1} = 0$  we can in parallel consider already the frontal matrix related to  $k = 2$ , the second step:

$$\begin{array}{ccc}
 A_{2,2} & A_{2,3} & A_{2,4} \\
 A_{3,2} & \frac{A_{3,2} \cdot A_{2,3}}{A_{2,2}} & \frac{A_{3,2} \cdot A_{2,4}}{A_{2,2}} \\
 A_{4,2} & \frac{A_{4,2} \cdot A_{2,3}}{A_{2,2}} & \frac{A_{4,2} \cdot A_{2,4}}{A_{2,2}}
 \end{array}$$

$$A = \begin{pmatrix}
 * & 0 & * & * \\
 0 & * & * & * \\
 * & * & * & 0 \\
 * & * & 0 & *
 \end{pmatrix}$$



## Multifrontal in Parallel (cont. 2)

The computations for  $k = 1$  and  $k = 2$  are independent and can be done in parallel (in the  $2 \times 2$  block):

$k = 1$ :

$$A_{i,j} \rightarrow A_{i,j} - \frac{A_{i,1} \cdot A_{1,j}}{A_{1,1}}$$

$k = 2$ :

$$A_{i,j} \rightarrow A_{i,j} - \frac{A_{i,2} \cdot A_{2,j}}{A_{2,2}}$$

Number of frontal matrices that can be used in parallel depends on the sparsity pattern.



# Parallel Numerics, WT 2013/2014

## *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
- 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  $\Phi$  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:

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

- 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

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

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

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

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

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  $A \approx I$ .

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  $\rho$  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  $A := M - N$

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

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

- $N$  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
- 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:

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

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

- 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  $\omega$ .

- 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

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

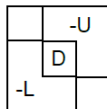
is convergent for

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

Look for optimal  $\omega$  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



## 5.1.3. 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  $\rho(B)$  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  $|\cos(x)|$
  - GS norm  $\frac{\sin(x)}{\sqrt{2(1-\cos(x))}}$
  - both  $< 1 \rightarrow$  convergence, but slow
- To improve convergence  $\rightarrow$  nonstationary methods (or multigrid)



# Chair of Informatics V—SCCS

## Efficient Numerical Algorithms—Parallel & HPC

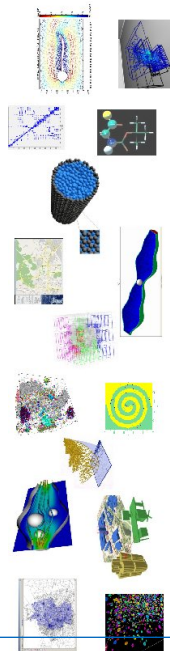
- High-dimensional numerics (sparse grids)
- Fast iterative solvers (multi-level methods, preconditioners)
- Adaptive, octree-based grid generation
- Space-filling curves
- Numerical linear algebra
- Numerical algorithms for image processing
- HW-aware numerical programming

## Fields of application in simulation

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

Further info → [www5.in.tum.de](http://www5.in.tum.de)

Feel free to come around and ask for thesis topics!



## 5.2. Nonstationary Methods

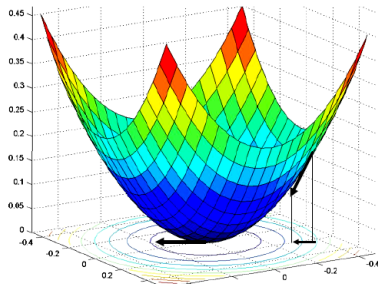
### 5.2.1. Gradient Method

- 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
- 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  $\alpha_k$ .

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

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

- To define  $\alpha_k$ :

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

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

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

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

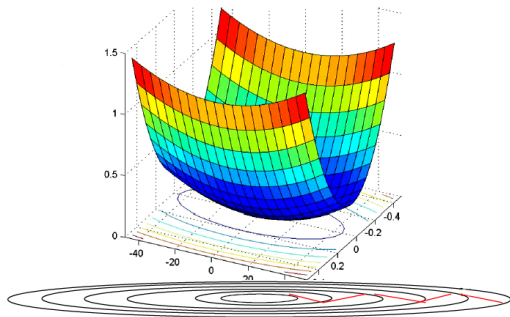
$$d^{(k)} = -\nabla\Phi(x^{(k)}) = b - Ax^{(k)}$$



## 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.
- 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:

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

- Therefore, minimizing  $\Phi$  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!



## 5.2.2. 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:

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

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

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

- We choose new search direction as component of last residual that is  $A$ -conjugate to all previous search directions.
- $\alpha_k$  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

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

- 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
- 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  $\lambda_j$
- 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:

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

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

## Better Estimates

- Choose normalized Chebyshev polynomials

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

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

- For clustered eigenvalues choose special polynomial, e.g. assume that  $A$  has only two eigenvalues  $\lambda_1$  and  $\lambda_2$ :

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

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

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  $\epsilon$ -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;
}

```

scalar

local vector

synchronisation  
points

dot product: requires communication (MPI\_ALLREDUCE)

