

# Scientific Computing II

## Conjugate Gradient Methods

Michael Bader  
Technical University of Munich

Summer 2017



*TUM Uhrenturm*

# Families of Iterative Solvers

- **relaxation methods:**
  - Jacobi-, Gauss-Seidel-Relaxation, ...
  - Over-Relaxation-Methods
- **Krylov methods:**
  - Steepest Descent, Conjugate Gradient, ...
  - GMRES, ...
- **Multilevel/Multigrid methods,**  
Domain Decomposition, ...

## Remember: The Residual Equation

- for  $Ax = b$ , we defined the **residual** as:

$$r^{(i)} = b - Ax^{(i)}$$

- and the error:  $e^{(i)} := x - x^{(i)}$
- leads to the **residual equation**:

$$Ae^{(i)} = r^{(i)}$$

- relaxation methods: solve a modified (easier) SLE:

$$B\hat{e}^{(i)} = r^{(i)} \quad \text{where } B \sim A$$

- multigrid methods: coarse-grid correction on residual equation

$$A_H e_H^{(i)} = r_H^{(i)} \quad \text{and} \quad x^{(i+1)} := x^{(i)} + I_H^h e_H^{(i)}$$

# Part I

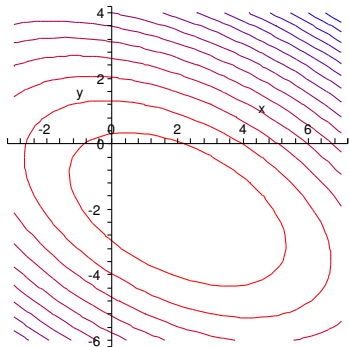
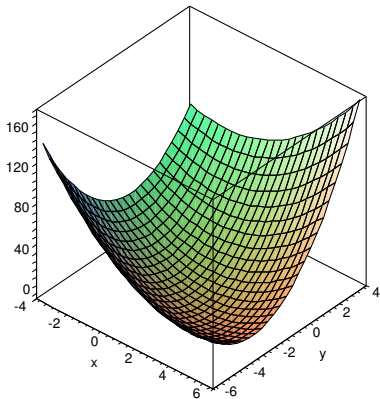
## Quadratic Forms and Steepest Descent

Quadratic Forms  
Direction of Steepest Descent  
Steepest Descent

# Quadratic Forms

A *quadratic form* is a scalar, quadratic function of a vector of the form:

$$f(x) = \frac{1}{2}x^T A x - b^T x + c. \quad \text{where } A = A^T$$



## Quadratic Forms (2)

The *gradient* of a quadratic form is defined as

$$f'(x) = \begin{pmatrix} \frac{\partial}{\partial x_1} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{pmatrix}$$

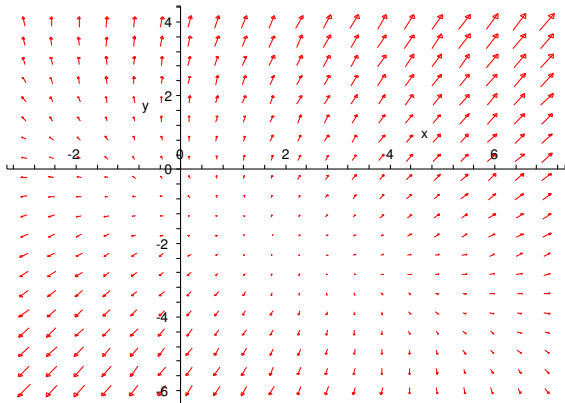
- apply to  $f(x) = \frac{1}{2}x^T Ax - b^T x + c$ , then
- $f'(x) = Ax - b$
- $f'(x) = 0 \Leftrightarrow Ax - b = 0 \Leftrightarrow Ax = b$

$\Rightarrow Ax = b$  equivalent to a **minimisation problem**

$\Rightarrow$  proper minimum **only if  $A$  positive definite**

# Direction of Steepest Descent

- gradient  $f'(x)$ : direction of “steepest ascent”
- $f'(x) = Ax - b = -r$  (with residual  $r = b - Ax$ )
- residual  $r$ : direction of “steepest descent”



# Solving SLE via Minimum Search

- basic idea to find minimum:  
move into direction of steepest descent
- most simple scheme:

$$x^{(i+1)} = x^{(i)} + \alpha r^{(i)}$$

- $\alpha$  constant  $\Rightarrow$  **Richardson** iteration  
(usually considered as a relaxation method)
- better choice of  $\alpha$ :  
move to lowest point in that direction  
 $\Rightarrow$  **Steepest Descent**



## Steepest Descent – find an optimal $\alpha$

- task: *line search* along the line  $x^{(1)} = x^{(0)} + \alpha r^{(0)}$
- choose  $\alpha$  such that  $f(x^{(1)})$  is minimal:

$$\frac{\partial}{\partial \alpha} f(x^{(1)}) = 0$$

- use chain rule:

$$\frac{\partial}{\partial \alpha} f(x^{(1)}) = f'(x^{(1)})^T \frac{\partial}{\partial \alpha} x^{(1)} = f'(x^{(1)})^T r^{(0)}$$

- remember  $f'(x^{(1)}) = -r^{(1)}$ , thus:

$$- \left( r^{(1)} \right)^T r^{(0)} \stackrel{!}{=} 0$$

hence,  $f'(x^{(1)}) = -r^{(1)}$  should be orthogonal to  $r^{(0)}$

## Steepest Descent – find $\alpha$ (2)

$$\begin{aligned}(r^{(1)})^T r^{(0)} &= (b - Ax^{(1)})^T r^{(0)} = 0 \\(b - A(x^{(0)} + \alpha r^{(0)}))^T r^{(0)} &= 0 \\(b - Ax^{(0)})^T r^{(0)} - \alpha (Ar^{(0)})^T r^{(0)} &= 0 \\(r^{(0)})^T r^{(0)} - \alpha (r^{(0)})^T Ar^{(0)} &= 0\end{aligned}$$

Solve for  $\alpha$ :

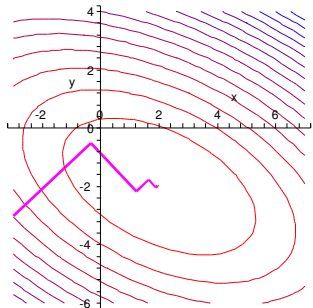
$$\alpha = \frac{(r^{(0)})^T r^{(0)}}{(r^{(0)})^T Ar^{(0)}}$$

# Steepest Descent – Algorithm

1.  $r^{(i)} = b - Ax^{(i)}$
2.  $\alpha_i = \frac{(r^{(i)})^T r^{(i)}}{(r^{(i)})^T Ar^{(i)}}$
3.  $x^{(i+1)} = x^{(i)} + \alpha_i r^{(i)}$

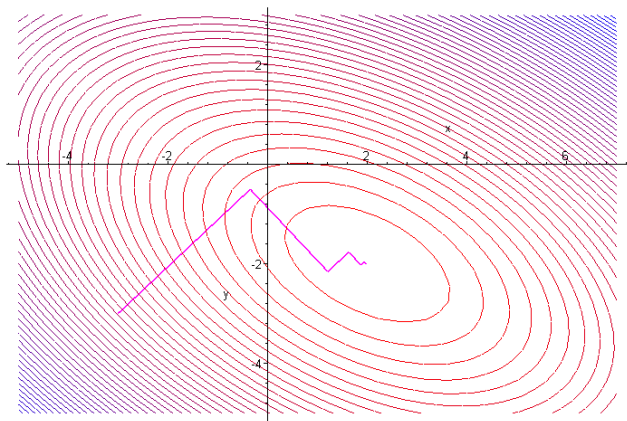
Observations:

- slow convergence (sim. to Jacobi relaxation)
- $\|e^{(i)}\|_A \leq \left(\frac{\kappa-1}{\kappa+1}\right)^i \|e^{(0)}\|_A$
- for positive definite  $A$ :  $\kappa = \lambda_{\max}/\lambda_{\min}$   
(largest/smallest eigenvalues of  $A$ )
- many steps in the same direction



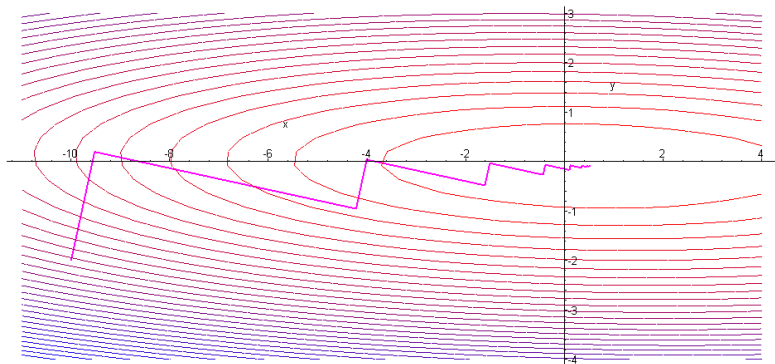
## Steepest Descent – Example 1

Consider example  $\begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix} x = \begin{pmatrix} 2 \\ -8 \end{pmatrix}$ , starting solution  $\begin{pmatrix} -3 \\ -3 \end{pmatrix}$



## Steepest Descent – Example 2

Consider example  $\begin{pmatrix} 3 & 2 \\ 2 & 100 \end{pmatrix} x = \begin{pmatrix} 2 \\ -8 \end{pmatrix}$ , starting solution  $\begin{pmatrix} -10 \\ -2 \end{pmatrix}$



**Multiple steps in the same search direction!**

## Part II

# Conjugate Gradients

**Conjugate Directions**  
**A-Orthogonality**  
**Conjugate Gradients**  
**A Miracle Occurs ...**  
**CG Algorithm**

# Conjugate Directions

- observation:  
Steepest Descent takes repeated steps in the same direction
- obvious idea:  
try to do only one step in each direction
- possible approach:  
choose orthogonal search directions  $d^{(0)} \perp d^{(1)} \perp d^{(2)} \perp \dots$
- notice:  
errors orthogonal to previous directions:

$$e^{(1)} \perp d^{(0)}, e^{(2)} \perp d^{(1)} \perp d^{(0)}, \dots$$

## Conjugate Directions (2)

- compute  $\alpha$  from

$$\left(\mathbf{d}^{(0)}\right)^T \mathbf{e}^{(1)} = \left(\mathbf{d}^{(0)}\right)^T \left(\mathbf{e}^{(0)} - \alpha \mathbf{d}^{(0)}\right) = 0$$

requires propagation of the error  $\mathbf{e}^{(1)} = \mathbf{x} - \mathbf{x}^{(1)}$

$$\begin{aligned} \mathbf{x}^{(1)} &= \mathbf{x}^{(0)} + \alpha_i \mathbf{d}^{(0)} \\ \mathbf{x} - \mathbf{x}^{(1)} &= \mathbf{x} - \mathbf{x}^{(0)} - \alpha_i \mathbf{d}^{(0)} \\ \mathbf{e}^{(1)} &= \mathbf{e}^{(0)} - \alpha_i \mathbf{d}^{(0)} \end{aligned}$$

- formula for  $\alpha$ :

$$\alpha = \frac{\left(\mathbf{d}^{(0)}\right)^T \mathbf{e}^{(0)}}{\left(\mathbf{d}^{(0)}\right)^T \mathbf{d}^{(0)}}$$

- **but:** we don't know  $\mathbf{e}^{(0)}$



# A-Orthogonality

- make the search directions *A-orthogonal*:

$$\left(d^{(i)}\right)^T A d^{(j)} = 0$$

- again: errors *A-orthogonal* to previous directions:

$$\left(e^{(i+1)}\right)^T A d^{(i)} \stackrel{!}{=} 0$$

- equiv. to minimisation in search direction  $d^{(i)}$ :

$$\begin{aligned} \frac{\partial}{\partial \alpha} f\left(x^{(i+1)}\right) &= \left(f'\left(x^{(i+1)}\right)\right)^T \frac{\partial}{\partial \alpha} x^{(i+1)} = 0 \\ &\Leftrightarrow -\left(r^{(i+1)}\right)^T d^{(i)} = 0 \\ &\Leftrightarrow -\left(d^{(i)}\right)^T A e^{(i+1)} = 0 \end{aligned}$$

# A-Conjugate Directions

- remember the formula for conjugate directions:

$$\alpha = \frac{(d^{(0)})^T e^{(0)}}{(d^{(0)})^T d^{(0)}}$$

- same computation, but with  $A$ -orthogonality:

$$\alpha_i = \frac{(d^{(i)})^T A e^{(i)}}{(d^{(i)})^T A d^{(i)}} = \frac{(d^{(i)})^T r^{(i)}}{(d^{(i)})^T A d^{(i)}}$$

(for the  $i$ -th iteration)

- these  $\alpha_j$  can be computed!**
- still to do: find  $A$ -orthogonal search directions

## A-Conjugate Directions (2)

classical approach to find orthogonal directions  $\rightarrow$

**conjugate Gram-Schmidt process:**

- from linearly independent vectors  $u^{(0)}, u^{(1)}, \dots, u^{(i-1)}$
- construct orthogonal directions  $d^{(0)}, d^{(1)}, \dots, d^{(i-1)}$

$$d^{(i)} = u^{(i)} + \sum_{k=0}^{i-1} \beta_{ik} d^{(k)}$$
$$\beta_{ik} = -\frac{(u^{(i)})^T A d^{(k)}}{(d^{(k)})^T A d^{(k)}}$$

- needs to keep all old search vectors in memory
- $\mathcal{O}(n^3)$  computational complexity  $\Rightarrow$  infeasible

# Conjugate Gradients

- use residuals (i.e.,  $u^{(i)} := r^{(i)}$ ) to construct conjugate directions:

$$d^{(i)} = r^{(i)} + \sum_{k=0}^{i-1} \beta_{ik} d^{(k)}$$

- new direction  $d^{(i)}$  should be  $A$ -orthogonal to all  $d^{(j)}$ :

$$0 \stackrel{!}{=} (d^{(i)})^T A d^{(j)} = (r^{(i)})^T A d^{(j)} + \sum_{k=0}^{i-1} \beta_{ik} (d^{(k)})^T A d^{(j)}$$

- all directions  $d^{(k)}$  (for  $k = 0, \dots, i-1$ ) are already  $A$ -orthogonal (and  $j < i$ ), hence:

$$0 = (r^{(i)})^T A d^{(j)} + \beta_{ij} (d^{(j)})^T A d^{(j)} \Rightarrow \beta_{ij} = -\frac{(r^{(i)})^T A d^{(j)}}{(d^{(j)})^T A d^{(j)}}$$

# Conjugate Gradients – Status

1. conjugate directions and computation of  $\alpha_j$ :

$$\alpha_j = \frac{(d^{(j)})^T r^{(j)}}{(d^{(j)})^T A d^{(j)}}$$

$$x^{(j+1)} = x^{(j)} + \alpha_j d^{(j)}$$

2. use residuals to compute search directions:

$$d^{(i)} = r^{(i)} + \sum_{k=0}^{i-1} \beta_{ik} d^{(k)}$$

$$\beta_{ik} = -\frac{(r^{(i)})^T A d^{(k)}}{(d^{(k)})^T A d^{(k)}}$$

→ **still too expensive**, as we need to store all vectors  $d^{(k)}$

# A Miracle Occurs – Part 1

Two small contributions:

1. propagation of the error  $e^{(i)} = x - x^{(i)}$

$$\begin{aligned}x^{(i+1)} &= x^{(i)} + \alpha_j d^{(i)} \\x - x^{(i+1)} &= x - x^{(i)} - \alpha_j d^{(i)} \\e^{(i+1)} &= e^{(i)} - \alpha_j d^{(i)}\end{aligned}$$

(we have used this once, already)

2. propagation of residuals

$$\begin{aligned}r^{(i+1)} &= Ae^{(i+1)} = A(e^{(i)} - \alpha_j d^{(i)}) \\ \Rightarrow r^{(i+1)} &= r^{(i)} - \alpha_j Ad^{(i)}\end{aligned}$$

## A Miracle Occurs – Part 2

Orthogonality of the residuals:

- search directions are  $A$ -orthogonal
- only one step in each direction
- hence: error is  $A$ -orthogonal to all previous search directions:

$$(d^{(i)})^T A e^{(j)} = 0, \text{ for } i < j$$

- residuals are orthogonal to all previous search directions:

$$(d^{(i)})^T r^{(j)} = 0, \text{ for } i < j$$

- search directions are built from residuals:  
 $\text{span} \{d^{(0)}, \dots, d^{(i-1)}\} = \text{span} \{r^{(0)}, \dots, r^{(i-1)}\}$
- hence: **residuals are orthogonal**

$$(r^{(i)})^T r^{(j)} = 0, \quad i < j$$

## A Miracle Occurs – Part 3

- combine orthogonality and recurrence for residuals:

$$\begin{aligned}(r^{(i)})^T r^{(j+1)} &= (r^{(i)})^T r^{(j)} - \alpha_j (r^{(i)})^T Ad^{(j)} \\ \Rightarrow \alpha_j (r^{(i)})^T Ad^{(j)} &= (r^{(i)})^T r^{(j)} - (r^{(i)})^T r^{(j+1)}\end{aligned}$$

- $(r^{(i)})^T r^{(j)} = 0$ , if  $i \neq j$ :

$$(r^{(i)})^T Ad^{(j)} = \begin{cases} \frac{1}{\alpha_i} (r^{(i)})^T r^{(i)}, & i = j \\ -\frac{1}{\alpha_{i-1}} (r^{(i)})^T r^{(i)}, & i = j + 1 \\ 0 & \text{otherwise.} \end{cases}$$



## A Miracle Occurs – Part 4

- computation of  $\beta_{ik}$  (for  $k = 0, \dots, i - 1$ ):

$$\beta_{ik} = -\frac{(r^{(i)})^T \text{Ad}^{(k)}}{(d^{(k)})^T \text{Ad}^{(k)}} = \begin{cases} \frac{(r^{(i)})^T r^{(i)}}{\alpha_{i-1} (d^{(i-1)})^T \text{Ad}^{(i-1)}}, & \text{if } i = k + 1 \\ 0, & \text{if } i > k + 1 \end{cases}$$

- thus: search directions

$$d^{(i)} = r^{(i)} + \sum_{k=0}^{i-1} \beta_{ik} d^{(k)} = r^{(i)} + \beta_{i,i-1} d^{(i-1)}$$

$$\beta_i := \beta_{i,i-1} = \frac{(r^{(i)})^T r^{(i)}}{\alpha_{i-1} (d^{(i-1)})^T \text{Ad}^{(i-1)}}$$

⇒ reduces to a **simple iterative scheme for  $\beta_i$**

## A Miracle Occurs – Part 5

- build search directions

$$\begin{aligned} \mathbf{d}^{(i+1)} &= \mathbf{r}^{(i+1)} + \beta_i \mathbf{d}^{(i)} \\ \beta_{i+1} &= \frac{(\mathbf{r}^{(i+1)})^T \mathbf{r}^{(i+1)}}{\alpha_i (\mathbf{d}^{(i)})^T \mathbf{A} \mathbf{d}^{(i)}} \end{aligned}$$

- remember:  $\alpha_j = \frac{(\mathbf{d}^{(j)})^T \mathbf{r}^{(j)}}{(\mathbf{d}^{(j)})^T \mathbf{A} \mathbf{d}^{(j)}}$
- thus:  $\alpha_j (\mathbf{d}^{(j)})^T \mathbf{A} \mathbf{d}^{(j)} = (\mathbf{d}^{(j)})^T \mathbf{r}^{(j)}$

$$\Rightarrow \beta_{i+1} = \frac{(\mathbf{r}^{(i+1)})^T \mathbf{r}^{(i+1)}}{(\mathbf{d}^{(i)})^T \mathbf{r}^{(i)}} = \frac{(\mathbf{r}^{(i+1)})^T \mathbf{r}^{(i+1)}}{(\mathbf{r}^{(i)})^T \mathbf{r}^{(i)}}$$

- last step:  $(\mathbf{d}^{(i)})^T \mathbf{r}^{(i)} = (\mathbf{r}^{(i)} + \beta_{i-1} \mathbf{d}^{(i-1)})^T \mathbf{r}^{(i)} = (\mathbf{r}^{(i)})^T \mathbf{r}^{(i)} + \beta_{i-1} (\mathbf{d}^{(i-1)})^T \mathbf{r}^{(i)} = (\mathbf{r}^{(i)})^T \mathbf{r}^{(i)}$   
(residual  $\mathbf{r}^{(i)}$  orthogonal to previous search direction  $\mathbf{d}^{(i-1)}$ )

# Conjugate Gradients – Algorithm

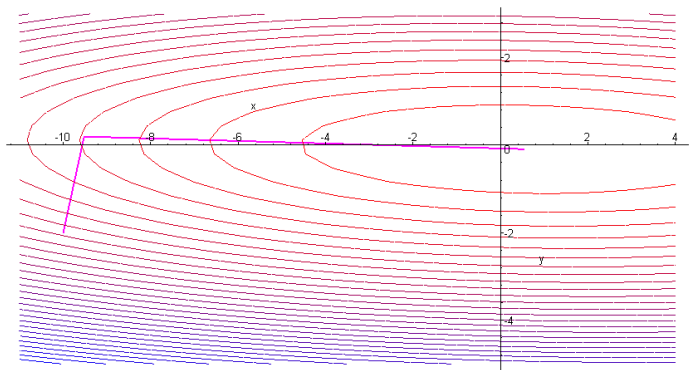
Start with  $d^{(0)} = r^{(0)} = b - Ax^{(0)}$

While  $r^{(i)} > \epsilon$  iterate over:

1.  $\alpha_i = \frac{(r^{(i)})^T r^{(i)}}{(d^{(i)})^T A d^{(i)}}$
2.  $x^{(i+1)} = x^{(i)} + \alpha_i d^{(i)}$
3.  $r^{(i+1)} = r^{(i)} - \alpha_i A d^{(i)}$
4.  $\beta_{i+1} = \frac{(r^{(i+1)})^T r^{(i+1)}}{(r^{(i)})^T r^{(i)}}$
5.  $d^{(i+1)} = r^{(i+1)} + \beta_{i+1} d^{(i)}$

## Conjugate Gradients – Example

Consider example  $\begin{pmatrix} 3 & 2 \\ 2 & 100 \end{pmatrix} x = \begin{pmatrix} 2 \\ -8 \end{pmatrix}$ , starting solution  $\begin{pmatrix} -10 \\ -2 \end{pmatrix}$



**Convergence to solution after  $n = 2$  steps!**

## Part III

# Preconditioning

**CG Convergence**

**Preconditioning**

**CG with “Change-of-Basis” Preconditioning**

**Hierarchical Transforms**

**CG with Matrix Preconditioner**

**Preconditioners – Examples**

**ILU and Incomplete Cholesky**

# Conjugate Gradients – Convergence

## Convergence Analysis:

- uses *Krylov subspace*:

$$\text{span} \left\{ r^{(0)}, Ar^{(0)}, A^2r^{(0)}, \dots, A^{i-1}r^{(0)} \right\}$$

- “Krylov subspace method”

## Convergence Results:

- in principle: direct method ( $n$  steps)  
(*however: orthogonality lost due to round-off errors → exact solution not found*)
- in practice: iterative scheme

$$\|e^{(i)}\|_A \leq 2 \left( \frac{\sqrt{\kappa} - 1}{\sqrt{\kappa} + 1} \right)^i \|e^{(0)}\|_A, \quad \kappa = \lambda_{\max}/\lambda_{\min}$$

# Preconditioning

- convergence depends on matrix  $A$
- idea: modify linear system

$$Ax = b \quad \rightsquigarrow \quad M^{-1}Ax = M^{-1}b,$$

then: convergence depends on matrix  $M^{-1}A$

- optimal preconditioner:  $M^{-1} = A^{-1}$ :

$$A^{-1}Ax = A^{-1}b \Leftrightarrow x = A^{-1}b.$$

- in practice:
  - avoid explicit computation of  $M^{-1}A$
  - find an  $M$  similar to  $A$ , compute effect of  $M^{-1}$  (i.e., approximate solution of SLE)
  - or: find an  $M^{-1}$  similar to  $A^{-1}$
- also possible: solve  $AMy = b$  for  $y$ , and then  $x = My$

# CG and Preconditioning

- just replace  $A$  by  $M^{-1}A$  in the algorithm??
- problem:  $M^{-1}A$  not necessarily symmetric (even if  $M$  and  $A$  both are)
- we will try an alternative first:  
**symmetric preconditioning**

$$Ax = b \quad \rightsquigarrow \quad L^T A L \hat{x} = L^T b, \quad x = L \hat{x}$$

- Remember: for Finite Element discretization, this corresponds to a change of basis functions!
- can we implement CG without having to set up  $L^T A L$ ?
- requires some re-computations in the CG algorithm (see following slides)



## “Change-of-Basis” Preconditioning

- preconditioned system of equations:

$$Ax = b \rightsquigarrow \underbrace{(L^T AL)}_{=: \hat{A}} \hat{x} = \underbrace{L^T b}_{=: \hat{b}}, \quad x = L\hat{x}$$

- computation of residual:

$$\hat{r} = \hat{b} - \hat{A}\hat{x} = L^T b - L^T AL\hat{x} = L^T(b - Ax) = L^T r$$

- computation of  $\alpha$  (for preconditioned system):

$$\alpha_j := \frac{(\hat{r}^{(j)})^T \hat{r}^{(j)}}{(\hat{d}^{(j)})^T \hat{A} \hat{d}^{(j)}} = \frac{(\hat{r}^{(j)})^T \hat{r}^{(j)}}{(\hat{d}^{(j)})^T L^T AL \hat{d}^{(j)}} = \frac{(\hat{r}^{(j)})^T \hat{r}^{(j)}}{(\tilde{d}^{(j)})^T A \tilde{d}^{(j)}}$$

where we defined  $L\tilde{d}^{(j)} =: \hat{d}^{(j)}$

- update of solution:

$$\begin{aligned} \hat{x}^{(j+1)} &= \hat{x}^{(j)} + \alpha_j \hat{d}^{(j)} \\ \Rightarrow x^{(j+1)} = L\hat{x}^{(j+1)} &= L\hat{x}^{(j)} + L\alpha_j \hat{d}^{(j)} = x^{(j)} + \alpha_j \tilde{d}^{(j)} \end{aligned}$$

## “Change-of-Basis” Preconditioning (2)

- update residuals  $\hat{r}$ :

$$\begin{aligned}\hat{r}^{(i+1)} &= \hat{r}^{(i)} - \alpha_j \hat{A} \hat{d}^{(i)} = \hat{r}^{(i)} - \alpha_j L^T A L \hat{d}^{(i)} \\ &= \hat{r}^{(i)} - \alpha_j L^T A \tilde{d}^{(i)}\end{aligned}$$

- computation of  $\beta_i$ :

$$\beta_{i+1} = \frac{(\hat{r}^{(i+1)})^T \hat{r}^{(i+1)}}{(\hat{r}^{(i)})^T \hat{r}^{(i)}}$$

- update of search directions:

$$\begin{aligned}\hat{d}^{(i+1)} &= \hat{r}^{(i+1)} + \beta_i \hat{d}^{(i)} \\ \Rightarrow \tilde{d}^{(i+1)} &= L \hat{d}^{(i+1)} = L \hat{r}^{(i+1)} + L \beta_i \hat{d}^{(i)} \\ &= L \hat{r}^{(i+1)} + \beta_i \tilde{d}^{(i)}\end{aligned}$$

## CG with “Change-of-Basis” Preconditioning

Start with  $\hat{r}^{(0)} = L^T(b - Ax^{(0)})$  and  $\tilde{d}^{(0)} = L\hat{r}^{(0)}$ ;

While  $\hat{r}^{(i)} > \epsilon$  iterate over:

1.  $\alpha_i = \frac{(\hat{r}^{(i)})^T \hat{r}^{(i)}}{(\tilde{d}^{(i)})^T A \tilde{d}^{(i)}}$
2.  $x^{(i+1)} = x^{(i)} + \alpha_i \tilde{d}^{(i)}$
3.  $\hat{r}^{(i+1)} = \hat{r}^{(i)} - \alpha_i L^T A \tilde{d}^{(i)}$
4.  $\beta_{i+1} = \frac{(\hat{r}^{(i+1)})^T \hat{r}^{(i+1)}}{(\hat{r}^{(i)})^T \hat{r}^{(i)}}$
5.  $\tilde{d}^{(i+1)} = L\hat{r}^{(i+1)} + \beta_i \tilde{d}^{(i)}$

# Hierarchical Basis Preconditioning

Some specifics for the CG implementation:

- $L$  transforms coefficient vector from hierarchical basis to nodal basis, for example  $\hat{x} = Lx$  or  $\tilde{d} = L\hat{d}$
- $L^T$  transforms the vector of basis functions from nodal basis to hierarchical basis (cmp. FEM), thus  $\hat{r} = L^T r$
- effect of  $L$  and  $L^T$  can be computed in  $\mathcal{O}(N)$  operations

## HB-CG for the Poisson problem:

- in 1D: convergence after a  $\log N$  iterations!  
(in this case:  $L^T A L$  diagonal matrix with  $\log N$  different eigenvalues)
- in 2D and 3D very fast convergence!
- further improved by additional diagonal preconditioning
- so-called *hierarchical generating systems* (change to a multigrid basis) achieve multigrid-like performance

# CG with Hierarchical Generating Systems

Recall: system of linear equations  $A^{\text{GS}} v^{\text{GS}} = b^{\text{GS}}$  given as

$$\begin{pmatrix} A_h & A_h P_{2h}^h & A_h P_{4h}^h \\ R_h^{2h} A_h & A_{2h} & A_{2h} P_{4h}^{2h} \\ R_h^{4h} A_h & R_{2h}^{4h} A_{2h} & A_{4h} \end{pmatrix} \begin{pmatrix} v_h \\ v_{2h} \\ v_{4h} \end{pmatrix} = \begin{pmatrix} b_h \\ R_h^{2h} b_h \\ R_h^{4h} b_h \end{pmatrix}$$

## Preconditioning for CG?

- system  $A^{\text{GS}} v^{\text{GS}} = b^{\text{GS}}$  is singular  
 $\rightarrow$  subspace of solutions  $v^{\text{GS}}$  (minima of the quadratic form!)
- however: any of the solutions will do!
- convergence result:

$$\|e^{(i)}\|_A \leq 2 \left( \frac{\sqrt{\hat{\kappa}_{\text{GS}}} - 1}{\sqrt{\hat{\kappa}_{\text{GS}}} + 1} \right)^i \|e^{(0)}\|_A, \quad \hat{\kappa}_{\text{GS}} = \hat{\lambda}_{\max} / \hat{\lambda}_{\min}$$

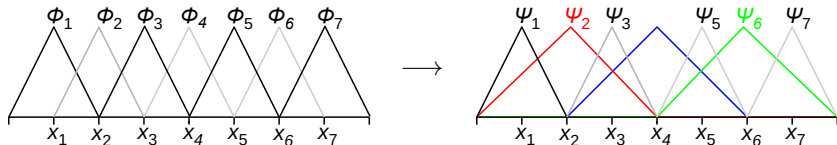
where  $\hat{\kappa}_{\text{GS}}$  is the ratio of largest vs. smallest **non-zero** eigenvalue

- for Poisson eq.:  $\hat{\kappa}_{\text{GS}}$  independent of  $h \rightsquigarrow$  multigrid convergence

# Hierarchical Basis Transformation

Towards level-wise approach

Consider “semi-hierarchical” transform:



Matrices for change of basis are then: ( $H_3^{(2)}$  to transform to hierarchical basis)

$$H_3^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & 1 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$$H_3^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 1 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

# Hierarchical Basis Transformation

## Level-wise hierarchical transform:

- hierarchical basis transformation:  $\psi_{n,i}(x) = \sum_j H_{i,j} \phi_{n,j}(x)$
- written as matrix-vector product:  $\vec{\psi}_n = H_n \vec{\phi}_n$
- $H_n \vec{\phi}_n$  can be performed as a sequence of level-wise transforms:

$$H_n \vec{\phi}_n = H_n^{(n-1)} H_n^{(n-2)} \dots H_n^{(2)} H_n^{(1)} \vec{\phi}_n$$

- consider Hierarchical-Basis-Preconditioning for FEM-discretization:  $L^T := H_n$  and we need to compute  $\hat{r}^{(0)} = L^T r^{(0)}$  and  $L^T A \tilde{d}^{(i)}$
- each level-wise transform  $H_n^{(k)}$  has a simple loop implementation:

*For k from 1 to n-1 do*  
*For j from  $2^k$  to  $2^n$  step  $2^k$*   
 $r_j := \frac{1}{2} r_{j-2^{k-1}} + r_k + \frac{1}{2} r_{j+2^{k-1}}$

## Hierarchical Coordinate Transformation

- transform  $b = H_n^T a$  turns “hierachical” coefficients  $a$  into “nodal” coefficients  $b$ :

$$\sum_j b_j \phi_{n,j}(x) = \sum_i a_i \psi_{n,i}(x) \approx f(x)$$

- $H_n = H_n^{(n-1)} H_n^{(n-2)} \dots H_n^{(2)} H_n^{(1)}$  has a level-wise representation, thus:

$$H_n^T = \left(H_n^{(1)}\right)^T \left(H_n^{(2)}\right)^T \dots \left(H_n^{(n-2)}\right)^T \left(H_n^{(n-1)}\right)^T$$

- consider Hierarchical-Basis-Preconditioning for FEM-discretization:  
 $L := H_n^T$  for computation of  $\tilde{d}^{(0)} = L \hat{r}^{(0)}$  and  $\tilde{d}^{(i+1)} = L \hat{r}^{(i+1)} + \beta_i \tilde{d}^{(i)}$
- again: use loop-based implementations for  $\left(H_n^{(k)}\right)^T a$

*For k from n-1 downto 1*

*For i from  $2^{k-1}$  to  $2^n$  step  $2^k$*

$$d_i := \frac{1}{2} d_{i-2^{k-1}} + d_i + \frac{1}{2} d_{i+2^{k-1}} \quad (\text{with } d_0 = a_{d^n} = 0)$$



## CG and Preconditioning (revisited)

- preconditioning: replace  $A$  by  $M^{-1}A$
- problem:  $M^{-1}A$  not necessarily symmetric
- compare symmetric preconditioning

$$Ax = b \rightsquigarrow L^T AL\hat{x} = Lb, \quad x = L\hat{x}$$

- workaround: find  $E^T E = M$  (Cholesky fact.), then

$$Ax = b \rightsquigarrow E^{-T}AE^{-1}\hat{x} = E^{-T}b, \quad \hat{x} = Ex$$

- what if  $E$  cannot be computed (efficiently)?  
(neither  $M$  nor  $M^{-1}$  might be known explicitly!)
- $E, E^{-T}, E^{-1}$  can be eliminated from algorithm  
(again requires some re-computations):

$$\text{set } \hat{d} = Ed \quad \text{and use } \hat{r} = E^{-T}r, \quad \hat{x} = Ex, \quad E^{-1}E^{-T} = M^{-1}$$

## CG with Preconditioner

Start:  $r^{(0)} = b - Ax^{(0)}$ ;  $d^{(0)} = M^{-1}r^{(0)}$

1. 
$$\alpha_i = \frac{(r^{(i)})^T M^{-1} r^{(i)}}{(d^{(i)})^T A d^{(i)}}$$

2. 
$$x^{(i+1)} = x^{(i)} + \alpha_i d^{(i)}$$

3. 
$$r^{(i+1)} = r^{(i)} - \alpha_i A d^{(i)}$$

4. 
$$\beta_{i+1} = \frac{(r^{(i+1)})^T M^{-1} r^{(i+1)}}{(r^{(i)})^T M^{-1} r^{(i)}}$$

5. 
$$d^{(i+1)} = M^{-1} r^{(i+1)} + \beta_{i+1} d^{(i)}$$

(for detailed derivation, see Shewchuck)

# Implementation

Preconditioning steps:  $M^{-1}r^{(i)}, M^{-1}r^{(i+1)}$

- $M^{-1}$  known then multiply  $M^{-1}r^{(i)}$
  - $M$  known? Then solve  $My = r^{(i)}$  to obtain  $y = M^{-1}r^{(i)}$
  - neither  $M$ , nor  $M^{-1}$  are known explicitly:
    - algorithm to solve  $My = r^{(i)}$  is sufficient!
    - algorithm to compute action of  $M^{-1}$  on a vector is sufficient!
- ⇒ any approximate solver for  $Ae = r^{(i)}$  is sufficient  
(if it is equivalent to applying a symmetric and pos. definite matrix)

## Preconditioners for CG – Examples

- find  $M \approx A$  and compute effect of  $M^{-1}$ :
  - Jacobi preconditioning:  $M := D_A$
  - (Symmetric) Gauss-Seidel preconditioning:  $M := L_A$  or  $M = (D_A + L'_A)D_A^{-1}(D_A + (L'_A)^T)$ , etc.
- just compute effect of  $M^{-1}$ :
  - any approximate solver might do  
→ incl. multigrid methods
  - incomplete Cholesky factorization  
→ i.e., incomplete  $LU$ -decomposition (ILU) for symmetric positive definite matrix
  - use a multigrid method as preconditioner(?)  
→ worthwhile (only) in situations where multigrid does not work (well) as stand-alone solver
- find an  $M^{-1}$  similar to  $A^{-1}$ 
  - “sparse approximate inverse” (SPAI)
  - tries to minimise  $\|I - MA\|_F$ , where  $M$  is a matrix with (given) sparse non-zero pattern

# Preconditioners – ILU and Incomplete Cholesky

Recall LU decomposition and Cholesky factorization:

- LU decomposition: given  $A$ , find lower/upper triangular matrices  $L$  and  $U$  such that  $A = LU$
- Cholesky factorization: given  $A = A^T$ , find lower triangular matrix  $L$  such that  $A = LL^T \rightarrow$  symmetric preconditioning!
- variants with explicit diagonal matrix  $D$ :

$$A = LD^{-1}U \quad \text{or} \quad A = LD^{-1}L^T,$$

where  $L = D + L'$  and  $R = D + R'$  with *strict* lower/upper triangular  $L', R'$

- **but:** for sparse  $A$ ,  $L$  and  $U$  may be non-sparse

**Idea:** disregard all fill-in during factorization

# Cholesky Factorization

$$\begin{pmatrix} L_{11} & & \\ L_{21} & L_{22} & \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} D_{11}^{-1} & & \\ & D_{22}^{-1} & \\ & & D_{33}^{-1} \end{pmatrix} \begin{pmatrix} L_{11}^T & L_{21}^T & L_{31}^T \\ & L_{22}^T & L_{32}^T \\ & & L_{33}^T \end{pmatrix} = \begin{pmatrix} A_{11} & A_{21}^T & A_{31}^T \\ A_{21} & A_{22} & A_{32}^T \\ A_{31} & A_{32} & A_{33} \end{pmatrix}$$

## Derive the factorization algorithm:

- assume that  $A_{11} \stackrel{!}{=} L_{11} D_{11}^{-1} L_{11}^T$  is already factorized
- let  $L_{21}$  be a  $1 \times k$  submatrix, i.e., to compute next row of  $L$ :

$$L_{21} D_{11}^{-1} L_{11}^T \stackrel{!}{=} A_{21}$$

$D_{11}^{-1} L_{11}^T$  upper triangular matrix  $\rightarrow$  solve triangular system for  $L_{21}$

- by convention  $L_{22} = D_{22}$  ( $1 \times 1$  “matrix”), which is computed from:

$$L_{21} D_{11}^{-1} L_{21}^T + L_{22} D_{22}^{-1} L_{22}^{-T} \stackrel{!}{=} A_{22} \quad \Rightarrow \quad L_{22} = D_{22} := A_{22} - L_{21} D_{11}^{-1} L_{21}^T$$

# Incomplete Cholesky Factorization

**Algorithm:**  $(A \rightarrow LD^{-1}L^T)$

- initialize  $D := 0, L := 0$
- for  $i = 1, \dots, n$ :
  1. for  $k = 1, \dots, i - 1$ :
    - if  $(i, k) \in S$  then set  $L_{ik} := A_{ik} - \sum_{j < k} L_{ij} D_{jj}^{-1} L_{kj}$
  2. set  $L_{ii} = D_{ii} := A_{ii} - \sum_{j < i} L_{ij} D_{jj}^{-1} L_{ij}$
- note: sums  $\sum_{j < k}$  and  $\sum_{j < i}$  only consider non-zero elements  $\in S$
- uses given **pattern**  $S$  of non-zero elements in the factorization (frequent choice: use non-zeros of  $A$  for  $S$ )
- Cholesky factorization computed in  $\mathcal{O}(n)$  operations for sparse matrices (with  $c \cdot n$  non-zeros)
- frequently used for preconditioning

# Literature/References

## Conjugate Gradients:

- Shewchuk: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*.
- Hackbusch: *Iterative Solution of Large Sparse Systems of Equations*, Springer 1993.
- M. Griebel: *Multilevelmethoden als Iterationsverfahren über Erzeugendensystemen*, Teubner Skripten zur Numerik, 1994  
M. Griebel: *Multilevel algorithms considered as iterative methods on semidefinite systems*, SIAM Int. J. Sci. Stat. Comput. 15(3), 1994.