ATHENS 2007 – Parallel Numerical Simulation

Iterative Solution of Linear Systems

Michael Bader

March, 20th–24th, 2006

Part I: Relaxation Methods

Part II: Conjugate Gradients

Part III: Multigrid Methods

Systems of Linear Equations in Scientific Computing

- discretization of both ODE and PDE leads to large systems of linear equations (LSE)
- solving these LSE is one of the most important/expensive tasks in scientific computing
- LSE resulting from ODE or PDE are typically:
  - **sparse**
    (because of the *local discretization stencils*)
  - **large**
    (because of the desired *accuracy*)

Direct Solvers for Sparse LSEs

Direct solvers are often not competitive:

- computing time grows quickly with the number of unknowns:
  - 2D-Poisson:
    $\mathcal{O}(N^2)$ required for band elimination
    $\mathcal{O}(N^{3/2})$ required for nested dissection
- classical elimination destroys sparsity:
  - hence, additional memory is required
  - 2D-Poisson:
    $\mathcal{O}(N^{3/2})$ required for band elimination
    $\mathcal{O}(N \log N)$ required for nested dissection
- exact solution is not necessarily required, as the SLE itself is only an approximation

Iterative Solvers for Sparse LSEs

Goals for iterative solvers:

- take advantage of the sparsity pattern; use little or no additional memory
- compute a series of approximations

$$x^{(0)} \longrightarrow x^{(1)} \longrightarrow \ldots \longrightarrow x^{(i)} \longrightarrow \ldots \longrightarrow \lim_{i \longrightarrow \infty} x^{(i)} = x$$

  that converges *quickly* and *uniformly* to the solution x
- modest growth of computing time; objective: rule-of-thumb like

  "for 3 digits, you need 10 steps"

  (regardless of number of unknowns)

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, ...

Part I

Relaxation Methods

The Residual Equation

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

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

- and the error: $e^{(i)} := x - x^{(i)}$
  (thus $x := x^{(i)} + e^{(i)}$);
- short computation:

$$r^{(i)} = b - Ax^{(i)} = Ax - Ax^{(i)} = A(x - x^{(i)}) = Ae^{(i)}.$$

- **residual equation:**

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

Residual Based Correction

Solve $Ax = b$ using the residual equation $Ae^{(i)} = r^{(i)}$

- r (which can be computed) is an indicator for the size of the error e (which is not known).
- use residual equation to compute a *correction* to $x^{(i)}$
- basic idea: solve a modified (easier) SLE:

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

- use $\hat{e}^{(i)}$ as an approximation for $e^{(i)}$, and set

$$x^{(i+1)} = x^{(i)} + \hat{e}^{(i)}.$$

Relaxation

How should we choose B?

- $B \sim A$ (B "similar" to A),
  i.e. $B^{-1} \approx A^{-1}$,
  or at least $B^{-1}y \approx A^{-1}y$ for most vectors y.
- $Be = r$ should be easy/fast to solve

Examples:

- $B = \text{diag}(A) = D_A$ (diagonal part of A)
  $\Rightarrow$ Jacobi iteration
- $B = L_A$ (lower triangular part of A)
  $\Rightarrow$ Gauss-Seidel iteration

Jacobi Relaxation

Iteration formulas in matrix-vector notation:

1. residual notation:

$$x^{(i+1)} = x^{(i)} + D_A^{-1} r^{(i)} = x^{(i)} + D_A^{-1} \left(b - A x^{(i)} \right)$$

2. for implementation:

$$x^{(i+1)} = D_A^{-1} \left(b - (A - D_A) x^{(i)} \right)$$

3. for analysis:

$$x^{(i+1)} = \left(I - D_A^{-1} A \right) x^{(i)} + D_A^{-1} b =: M x^{(i)} + N b$$

Jacobi Relaxation – Algorithm

- based on: $x^{(i+1)} = D_A^{-1}\left(b - (A - D_A)x^{(i)}\right)$

```
for i from 1 to n do
    xnew[i] := ( b[i]
          - sum( A[i,j]*x[j], j=1..i-1)
          - sum( A[i,j]*x[j], j=i+1..n)
        ) / A[i,i];
end do;
for i from 1 to n do
    x[i] := xnew[i];
end do;
```

- **properties:**
  - additional storage required (`xnew`)
  - `x`, `xnew` can be computed **in any order**
  - `x`, `xnew` can be computed **in parallel**

Gauss-Seidel Relaxation

Iteration formulas in matrix-vector notation:

1. residual notation:

$$x^{(i+1)} = x^{(i)} + L_A^{-1} r^{(i)} = x^{(i)} + L_A^{-1} \left(b - A x^{(i)} \right)$$

2. for implementation:

$$x^{(i+1)} = L_A^{-1} \left(b - (A - L_A) x^{(i)} \right)$$

3. for analysis:

$$x^{(i+1)} = \left(I - L_A^{-1} A \right) x^{(i)} + L_A^{-1} b =: M x^{(i)} + N b$$

Gauss-Seidel Relaxation – Algorithm

- based on: $x^{(i+1)} = L_A^{-1}\left(b - (A - L_A)x^{(i)}\right)$
- solve $L_A x^{(i+1)} = b - (A - L_A)x^{(i)}$
  via backwards substitution:

```
for i from 1 to n do
    x[i] := ( b[i]
          - sum( A[i,j]*x[j], j=1..i-1)
          - sum( A[i,j]*x[j], j=i+1..n)
        ) / A[i,i];
end do;
```

- **properties:**
  - no additional storage required
  - inherently **sequential** computation of `x`
  - usually faster than Jacobi

Successive-Over-Relaxation (SOR)

- observation: Gauss-Seidel corrections are "too small"
- add an over-relaxation-factor α:

```
for i from 1 to n do
    x[i] := x[i] + alpha * ( b[i]
          - sum( A[i,j]*x[j], j=1..n)
        ) / A[i,i];
end do;
```

- for 2D-Poisson: optimal α (≈ 1.7) improves convergence: $\mathcal{O}(n^2) \rightarrow \mathcal{O}(n^{3/2})$

Does It Always Work?

- simple answer: no (life is not that easy . . .)
- Jacobi: matrix A needs to be *diagonally dominant*
- Gauß-Seidel: matrix A needs to be *positive definite*
- How about performance?
  $\longrightarrow$ usually quite slow

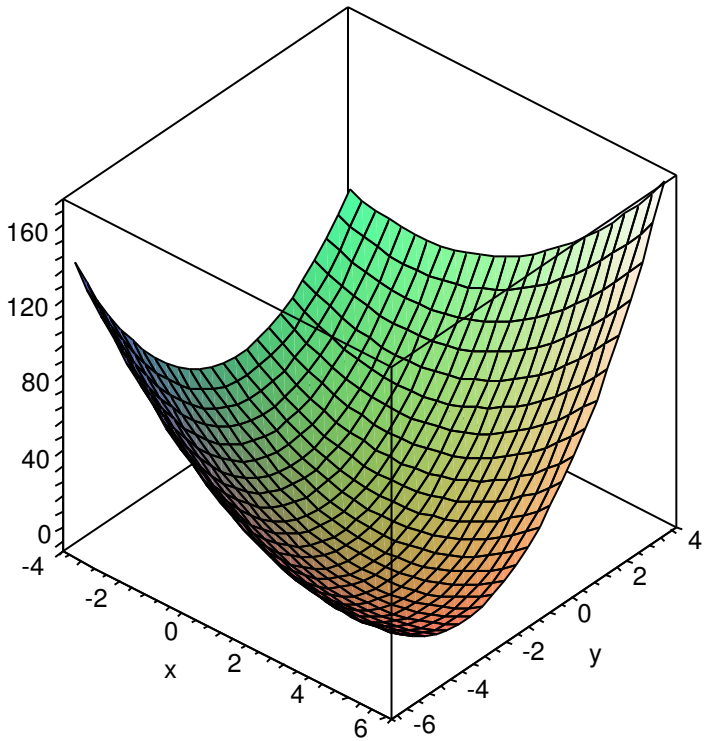
Part II

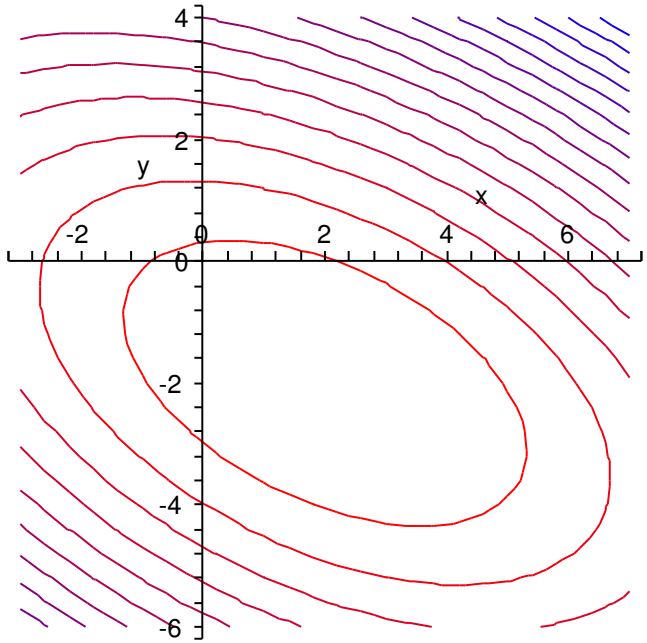
Conjugate Gradients

Quadratic Forms

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

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





Quadratic Forms (2)

The *gradient* of a quadratic form is defined as

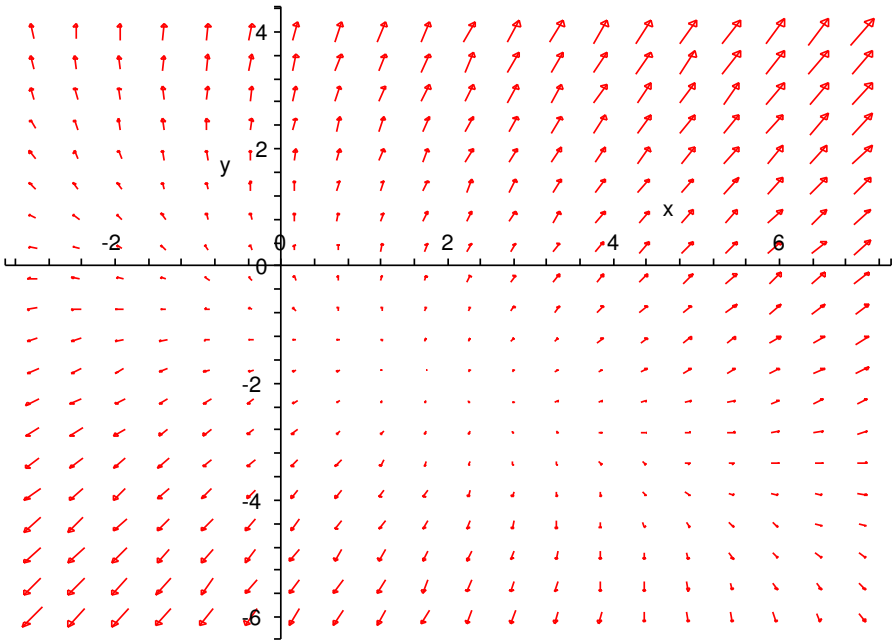
$$f'(x) = \begin{pmatrix} \frac{\partial}{\partial x_1} f(x) \\ \vdots \\ \frac{\partial}{\partial x_n} f(x) \end{pmatrix}$$

- $f'(x) = Ax - b$
- $f'(x) = 0 \quad \Leftrightarrow \quad Ax - b = 0 \quad \Leftrightarrow \quad Ax = b$

$\Rightarrow Ax = b$ equivalent to a **minimisation problem** (if A positive definite)

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"



Steepest Descent (2)

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

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

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

Steepest Descent – find α

- task: *line search* along the line $x^{(1)} = x^{(0)} + \alpha r^{(0)}$
- choose α 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:

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

Steepest Descent – find α (2)

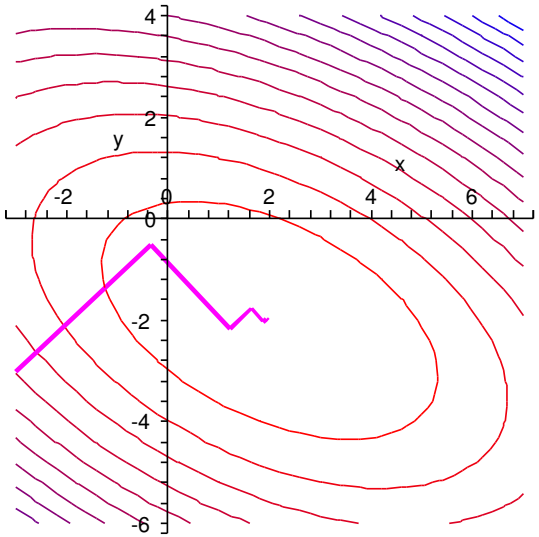
$$\begin{aligned}
\left(r^{(1)}\right)^T r^{(0)} = \left(b - Ax^{(1)}\right)^T r^{(0)} &= 0 \\
\left(b - A(x^{(0)} + \alpha r^{(0)})\right)^T r^{(0)} &= 0 \\
\left(b - A(x^{(0)}\right)^T r^{(0)} - \alpha \left(Ar^{(0)}\right)^T r^{(0)} &= 0 \\
\left(r^{(0)}\right)^T r^{(0)} - \alpha \left(r^{(0)}\right)^T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

Solve for α:

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

Steepest Descent – Algorithm

1. $r^{(i)} = b - Ax^{(i)}$
2. $\alpha = \frac{\left(r^{(0)}\right)^T A r^{(0)}}{\left(r^{(0)}\right)^T r^{(0)}}$
3. $x^{(i+1)} = x^{(i)} + \alpha r^{(i)}$



Observations:

- rather slow convergence
- $\left\|e^{(i)}\right\|_A \leq \left(\frac{\kappa-1}{\kappa+1}\right)^i \left\|e^{(0)}\right\|_A$, where $\kappa = \lambda_{\max}/\lambda_{\min}$ (largest/smallest eigenvalues of A)
- many steps in the same direction

Conjugate Directions

- 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
  $d^{(0)} \perp d^{(1)} \perp d^{(2)} \perp \ldots$
- notice: errors orthogonal to previous directions: $e^{(1)} \perp d^{(0)}, e^{(2)} \perp d^{(1)} \perp d^{(0)}, \ldots$

Conjugate Directions (2)

- compute α from

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

- formula for α:

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

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

A-Orthogonality

- make the search directions *A-orthogonal*:

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

- again: errors orthogonal to previous directions:

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

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

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

A-Conjugate Directions

- remember the formula for conjugate directions:

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

- with A-orthogonality:

$$\alpha = -\frac{\left(d^{(i)}\right)^T A e^{(i)}}{\left(d^{(i)}\right)^T A d^{(i)}} = \frac{\left(d^{(i)}\right)^T r^{(i)}}{\left(d^{(i)}\right)^T A d^{(i)}}$$

- only one task left:
  find A-orthogonal search directions

A-Conjugate Directions (2)

classical approach to find orthogonal directions: *conjugate Gram-Schmidt process*:

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

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

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

Conjugate Gradients

use residuals to construct conjugate directions:

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

directions $d^{(i)}$ should be A-orthogonal:

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

d-vectors are A-orthogonal, hence:

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

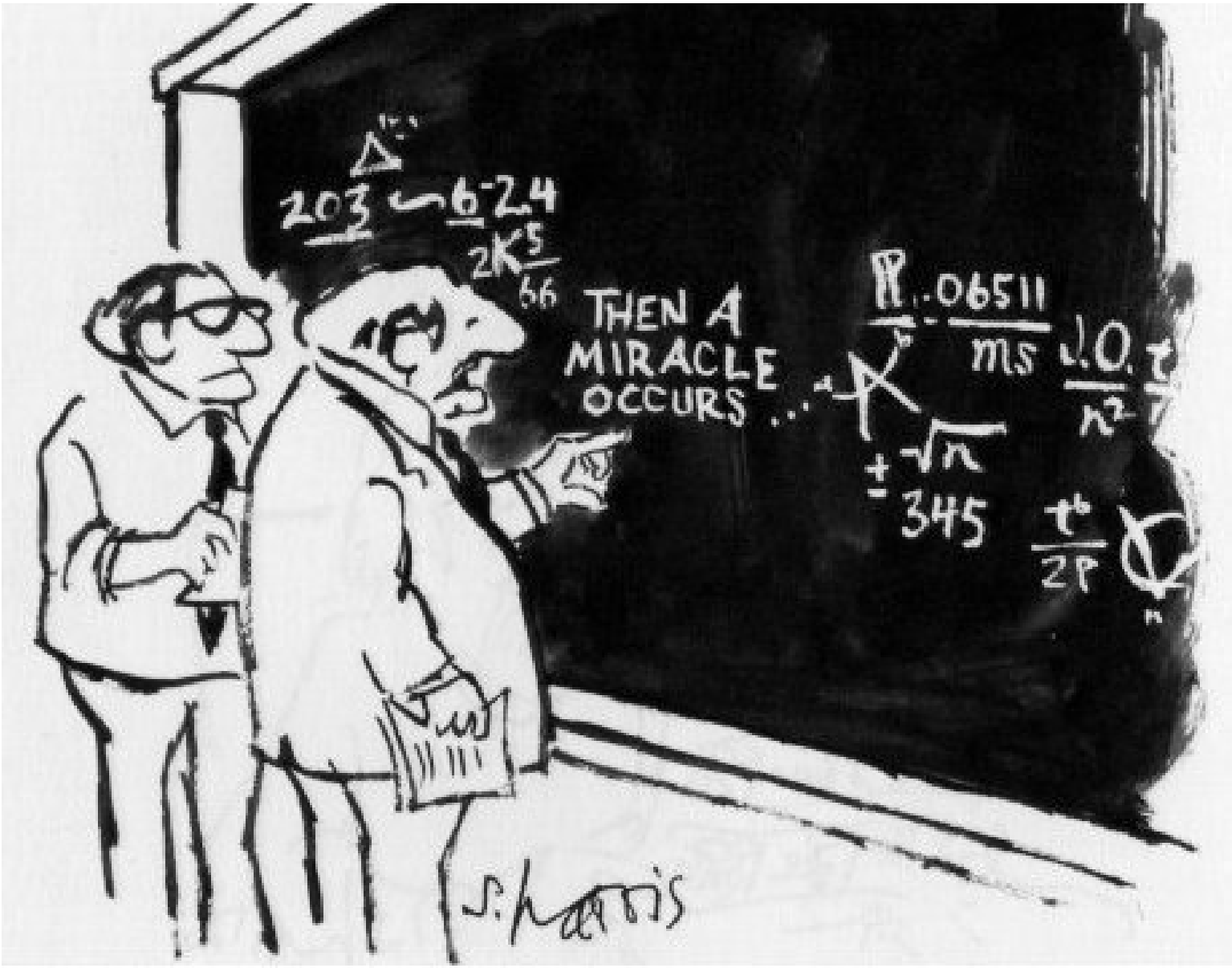
Conjugate Gradients – Status

1. conjugate directions:

$$\begin{aligned} \alpha_i &= \frac{\left(d^{(i)}\right)^T r^{(i)}}{\left(d^{(i)}\right)^T A d^{(i)}} \\ x^{(i+1)} &= x^{(i)} + \alpha_i d^{(i)} \end{aligned}$$

2. use residuals to compute search directions:

$$\begin{aligned} d^{(i)} &= r^{(i)} + \sum_{k=0}^{i-1} \beta_{ik} d^{(k)} \\ \beta_{ik} &= -\frac{\left(r^{(i)}\right)^T A d^{(k)}}{\left(d^{(k)}\right)^T A d^{(k)}} \end{aligned}$$



"I think you should be more explicit here in step two."

from *What's so Funny about Science?* by Sidney Harris (1977)

A Miracle Occurs – Part 1

Two small contributions:

1. propagation of the error

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

2. propagation of residuals

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

A Miracle Occurs – Part 2

Orthogonality of the residuals:

- search directions are A-orthogonal
- only one step in each directions
- hence: error is A-orthogonal to previous search directions: $\left(d^{(i)}\right)^T A e^{(j)} = 0$, for $i < j$
- residuals are orthogonal to previous search directions: $\left(d^{(i)}\right)^T r^{(j)} = 0$, for $i < j$
- search directions are built from residuals: $\text{span}\left\{d^{(0)}, \ldots, d^{(i-1)}\right\} = \text{span}\left\{r^{(0)}, \ldots, r^{(i-1)}\right\}$
- hence: **residuals are orthogonal**

$$\left(r^{(i)}\right)^T r^{(j)} = 0, \qquad i < j$$

A Miracle Occurs – Part 3

- combine orthogonality and recurrence for residuals:

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

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

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

A Miracle Occurs – Part 4

- computation of β_{ij}:

$$\begin{aligned}\beta_{ik} &= -\frac{\left(r^{(i)}\right)^T A d^{(k)}}{\left(d^{(k)}\right)^T A d^{(k)}} \\ &= \begin{cases} -\dfrac{\left(r^{(i)}\right)^T r^{(i)}}{\alpha_i \left(d^{(i-1)}\right)^T A d^{(i-1)}}, & i = j+1 \\ 0 & i > j+1 \end{cases}\end{aligned}$$

- remember: $\alpha_i = \frac{(d^{(i)})^T r^{(i)}}{(d^{(i)})^T A d^{(i)}}$

$$\Rightarrow \beta_i = -\frac{\left(r^{(i)}\right)^T r^{(i)}}{\left(d^{(i-1)}\right)^T r^{(i-1)}} = \frac{\left(r^{(i)}\right)^T r^{(i)}}{\left(r^{(i-1)}\right)^T r^{(i-1)}}$$

Conjugate Gradients – Algorithm

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

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. $\beta_{i+1} = \frac{\left(r^{(i+1)}\right)^T r^{(i+1)}}{\left(r^{(i)}\right)^T r^{(i)}}$
5. $d^{(i+1)} = r^{(i+1)} + \beta_{i+1} d^{(i)}$

Conjugate Gradients – Convergence

Convergence Analysis:

- uses *Krylow subspace*:

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

- "Krylow subspace method"

Convergence Results:

- in principle: direct method (n steps)
- in practice: iterative scheme

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

Preconditioning

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

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

  then: convegence 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}
  - find an M^{-1} similar to A^{-1}

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)
- workaround: find $EE^T = M$, then:

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

- undesirable, because E has to be computed (however, neither M nor M^{-1} might be known explicitly
- some re-computations $\rightarrow$ next slide

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

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!
    $\rightarrow$ any approximate solver for $Ae = r^{(i)}$
  - algorithm to compute M^{-1} is sufficient!
    $\rightarrow$ compute (sparse) approximate inverse
- Examples: Mutigrid, Jacobi, ILU, SPAI, ...

Part III

Multigrid Methods

Convergence of Relaxation Methods

Observation

- slow convergence
- high frequency error components are damped more efficiently
- smooth error components are reduced very slowly

Convergence Analysis

- remember iteration scheme: $x^{(i+1)} = Mx^{(i)} + Nb$
- derive iterative scheme for the error $e^{(i)} := x - x^{(i)}$:

$$e^{(i+1)} = x - x^{(i+1)} = x - Mx^{(i)} - Nb$$

- for **consistent** scheme, x is a fixpoint of the iteration ($x = Mx - Nb$)
- hence:

$$\begin{aligned} e^{(i+1)} &= Mx + Nb - Mx^{(i)} - Nb = Me^{(i)} \\ e^{(i)} &= M^i e^{(0)}. \end{aligned}$$

Convergence Analysis (2)

- iteration equation for error: $e^{(i)} = M^i e^{(0)}$
- consider eigenvalues λ_j and eigenvectors v_j of iteration matrix M:

$$Mv_j = \lambda_j v_j \quad \Rightarrow \quad M(\underbrace{\sum_j \alpha_j v_j}_{=:e^{(0)}}) = \sum_j \lambda_j \alpha_j v_j$$

$$\Rightarrow \quad M^i e^{(0)} = M^i (\sum_j \alpha_j v_j) = \sum_j \lambda_j^i \alpha_j v_j$$

- convergence, if all $|\lambda_j| < 1$
- speed of convergence dominated by largest λ_j

The Smoothing Property

- for 1D-Poisson: eigenvectors: $\sin(k\pi j/n)$
- eigenvalues: $4\sin^2\left(\frac{k\pi}{2n}\right)$
- decompose the error $e^{(i)}$ into eigenvector ($\sin(k\pi x_j)$, **Fourier mode analysis**)
- smallest eigenvalue of A (for $k = 1$): $\mathcal{O}(n^{-2})$
- largest eigenvalue of $M = I - D_A^{-1}A$: $\mathcal{O}(1 - n^{-2})$
- convergence determined by $\mathcal{O}(1 - n^{-2})$

The Smoothing Property (2)

Result of convergence analysis:

- The *high* frequency part (with respect to the underlying grid) is reduced quite quickly.
- The *low* frequency part (w.r.t. the grid) decreases only very slowly; actually the slower, the finer the grid is.

$\Rightarrow$ **"smoothing property"**

Multigrid Idea No. 1

- result from convergence analysis: "high-frequency error" is relative to mesh size
- on a sufficiently coarse grid, even very low frequencies can be "high-frequency" (if the mesh size is big)

"Multigrid":

- use multiple grids to solve the system of equations
- on each grid, a certain range of error frequencies will be reduced efficiently

Multigrid Idea No. 2

Solve the problem on a coarser grid:

- will be comparably (very) fast
- can give us a good initial guess:
  - **nested iteration**/"poor man's multigrid"
  - unfortunately, will not improve a fine grid solution any further

$\Rightarrow$ **Idea No. 2: use the residual equation**:

- solve $Ae = r$ on a coarser grid
- leads to an approximation of the error e
- add this approximation to the fine-grid solution

A Two-Grid Method

Algorithm:

1. **relaxation/smoothing** on the fine level system $\Rightarrow$ solution x_h
2. compute the **residual** $r_h = b_h - A_h x_h$
3. **restriction** of r_h to the coarse grid Ω_H
4. compute a **solution** to $A_H e_H = r_H$
5. **interpolate** the coarse grid solution e_H to the fine grid Ω_h
6. add the resulting **correction** to x_h
7. again, **relaxation/smoothing** on the fine grid

Correction Scheme – Components

smoother: reduce the high-frequency error components, and get a smooth error

restriction: transfer residual from fine grid to coarse grid, for example by

- injection
- (full) weighting

coarse grid equation: (acts as) discretisation of the PDE on the coarse grid

interpolation: transfer coarse grid solution/correction from coarse grid to fine grid

The Multigrid V-Cycle

1. smoothing on the fine level system $\Rightarrow$ solution x_l
2. compute the residual $r_l = b_l - A_l x_l$
3. restriction of r_l to the coarse grid Ω_{l-1}
4. solve coarse grid system $A_{l-1} e_{l-1} = r_{l-1}$ by a **recursive call to the V-cycle algorithm**
5. interpolate the coarse grid solution e_{l-1} to the fine grid Ω_l
6. add the resulting correction to x_l
7. **post-smoothing** on the fine grid

V-Cycle – Implementation

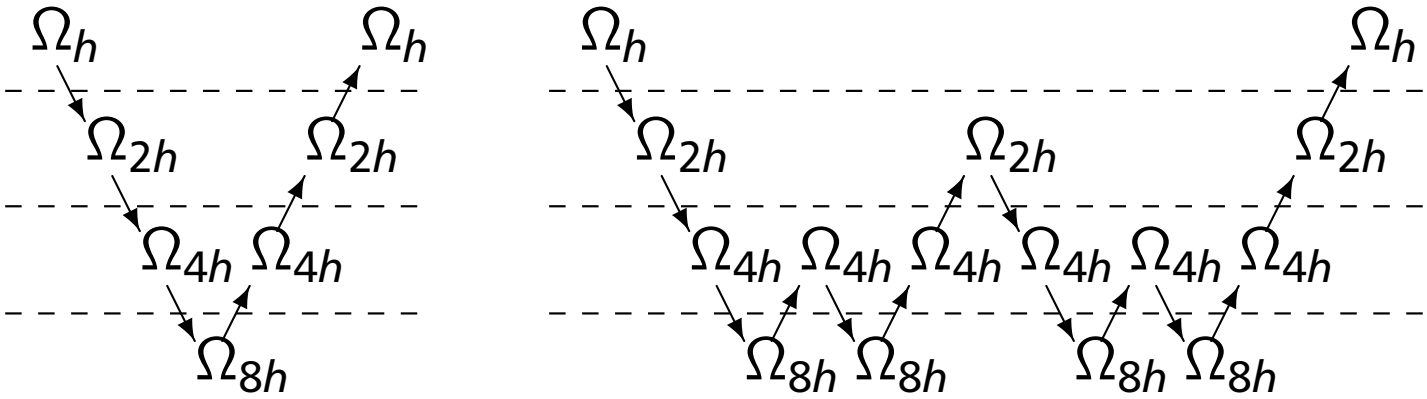
- on the coarsest grid: direct solution
- number of smoothing steps is typically very small (1 or 2)

Cost (storage and computing time):

- 1D: $c \cdot n + c \cdot n/2 + c \cdot n/4 + \ldots \leq 2c \cdot n$
- 2D: $c \cdot n + c \cdot n/4 + c \cdot n/16 + \ldots \leq 4/3c \cdot n$
- 3D: $c \cdot n + c \cdot n/8 + c \cdot n/64 + \ldots \leq 8/7c \cdot n$
- overall costs are dominated by the costs of the finest grid

The W-Cycle

- perform **two** coarse grid correction steps instead of one



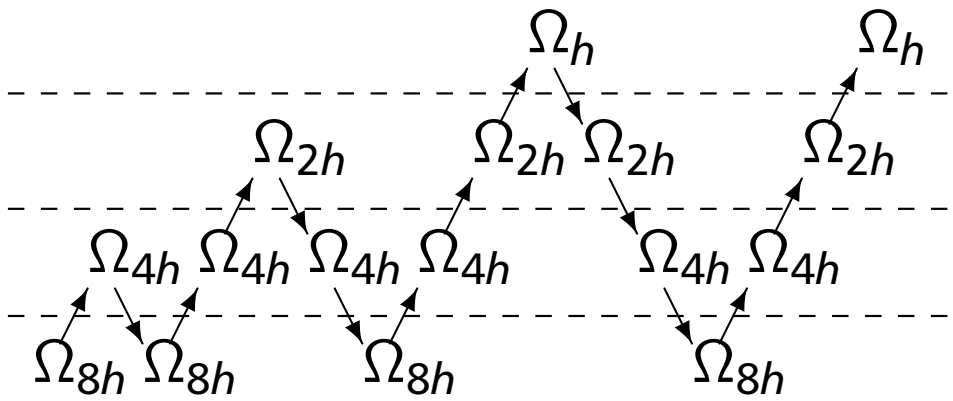
(V-cycle and W-cycle)

- more expensive
- useful in situations where the coarse grid correction is not very accurate

The Full Multigrid V-Cycle (FMV)

Recursive algorithm:

- perform an **FMV-cycle** on the next coarser grid to get a good initial solution
- interpolate this initial guess to the current grid
- perform a **V-cycle** to improve the solution



Speed of Convergence

- fastest method around
  (if all components are chosen carefully)
- **"textbook multigrid efficiency"**:

$$\left\| e^{(m+1)} \right\| \leq \gamma \left\| e^{(m)} \right\|,$$

where convergence rate $\gamma < 1$ (esp. $\gamma << 1$) is independent of the number of unknowns

  $\Rightarrow$ constant number of multigrid steps to obtain a given number of digits

  $\Rightarrow$ overall computational work increases only linearly with the number of unknowns

Convergence Rates (2)

For Poisson Problems ("Model Problem"):

- $\mathcal{O}(n)$ to solve up to "level of truncation"
- **"level of truncation"**: $\mathcal{O}(h^2)$
- $\mathcal{O}(n)$ is achieved by FMV-Cycle
  (1 or 2 cycles sufficient)

For Other Problems:

- OK for strongly elliptic problems
- multigrid variants for non-linear problems, parabolic/hyperbolic, ...
- achieving "textbook efficiency" usually a demanding task

Literature

General:

- Gander, Hrebicek: *Solving Problems in Scientific Computing Using Maple and MATLAB*.
- Golub, Ortega: *Scientific Computing and Differential Equations*.
- Dongarra, et. al.: *Numerical linear algebra for high-performance computers*.

Literature (2)

Multigrid:

- Briggs, Henson, McCormick: *A Multigrid Tutorial* (2nd ed.).

Conjugate Gradients:

- Shewchuk: *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain*.