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)

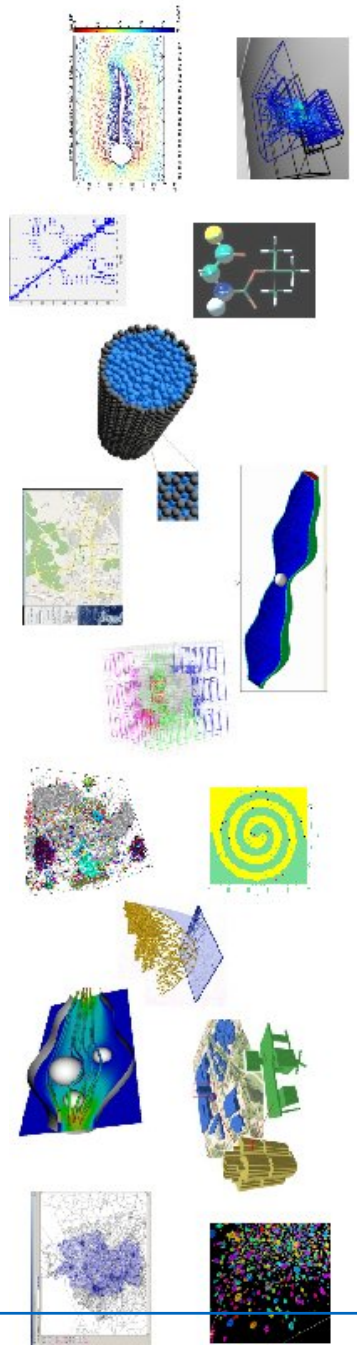
Chair of Informatics V—SCCS

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!

5.3. Nonstationary Methods

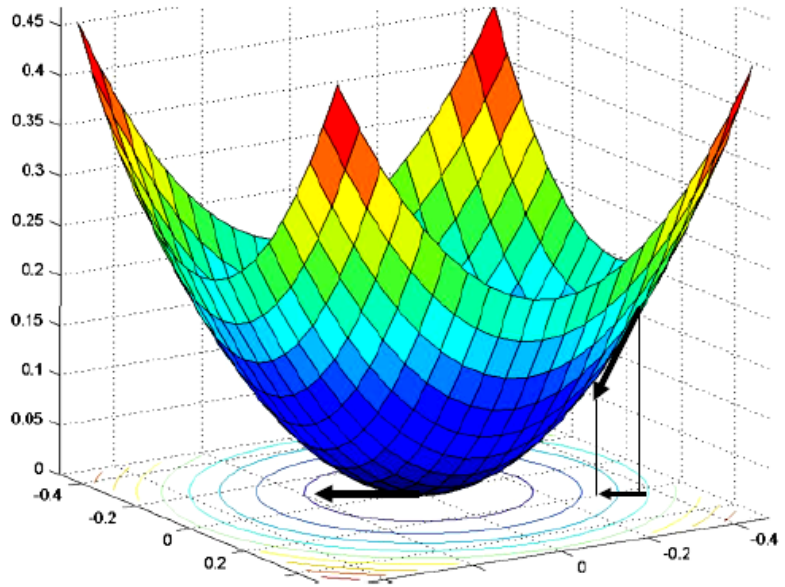
5.3.1. Gradient Method

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

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

- n-dim. paraboloid $\mathbb{R}^n \to \mathbb{R}$
- 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:

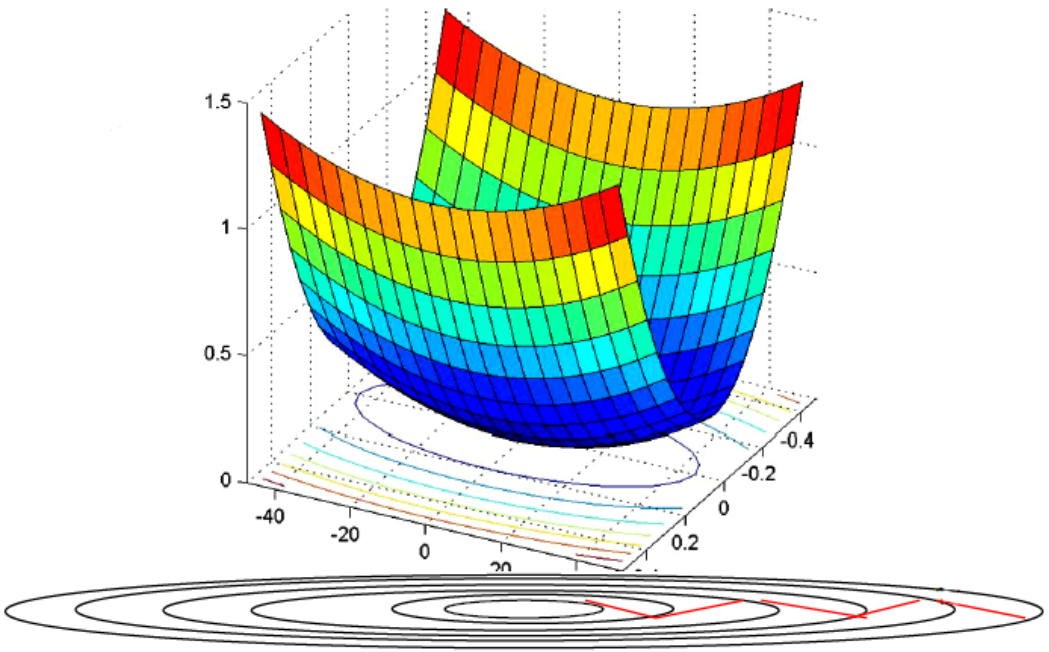
$$\begin{aligned}\min_\alpha g(\alpha) &:= \min_\alpha(\Phi(x^{(k)} + \alpha(b - Ax^{(k)}))) \\ &= \min_\alpha \left(\frac{1}{2}(x^{(k)} + \alpha d^{(k)})^T A(x^{(k)} + \alpha d^{(k)}) - b^T(x^{(k)} + \alpha d^{(k)})\right) \\ &= \min_\alpha \left(\frac{1}{2}\alpha^2 {d^{(k)}}^T A d^{(k)} - \alpha {d^{(k)}}^T d^{(k)} + \frac{1}{2}{x^{(k)}}^T A x^{(k)} - {x^{(k)}}^T b\right)\end{aligned}$$

$$\alpha_k = \frac{{d^{(k)}}^T d^{(k)}}{{d^{(k)}}^T A d^{(k)}} \qquad d^{(k)} = -\nabla\Phi(x^{(k)}) = b - Ax^{(k)}$$

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:

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

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 $k = 1, 2, \ldots$ 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

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

- and

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

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

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

Properties of Conjugate Gradients

- It holds
$$p^{(j)^T} A p^{(k)} = 0 = r^{(j)^T} r^{(k)} \quad \text{for} \quad j \neq k$$
- and
$$\begin{aligned} \text{span}(p^{(1)}, \ldots, p^{(j)}) &= \text{span}(r^{(0)}, \ldots, r^{(j-1)}) = \\ &= \text{span}(r^{(0)}, Ar^{(0)}, \ldots, A^{j-1} r^{(0)}) = K_j(A, r^{(0)}) \end{aligned}$$
- Especially for $x^{(0)} = \mathbf{0}$ it holds
$$K_j(A, r^{(0)}) = \text{span}(b, Ab, \ldots, A^{j-1} b)$$
- $x^{(k)}$ 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 ($x^{(0)} = 0$)

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

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

Error Estimation

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

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

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

Error Estimation

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

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

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

$$\|e^{(k)}\|_A = \min_{\mathcal{Q}^{(k)}(0)=1} \left\| \mathcal{Q}^{(k)}(A) \sum_{j=1}^{n} \rho_j u_j \right\|_A = \min_{\mathcal{Q}^{(k)}(0)=1} \left\| \sum_{j=1}^{n} \rho_j \mathcal{Q}^{(k)}(A) u_j \right\|_A =$$

$$= \min_{\mathcal{Q}^{(k)}(0)=1} \left\| \sum_{j=1}^{n} \rho_j \mathcal{Q}^{(k)}(\lambda_j) u_j \right\|_A \leq \min_{\mathcal{Q}^{(k)}(0)=1} \left\{ \max_{j=1,\ldots,n} \left| \mathcal{Q}^{(k)}(\lambda_j) \right| \right\} \left\| \sum_{j=1}^{n} \rho_j u_j \right\|_A =$$

$$= \min_{\mathcal{Q}^{(k)}(0)=1} \left\{ \max_{j=1,\ldots,n} |\mathcal{Q}^{(k)}(\lambda_j)| \right\} \left\| e^{(0)} \right\|_A$$

Error Estimates

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

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

Error Estimates

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

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

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

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:

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

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

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:

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

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

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

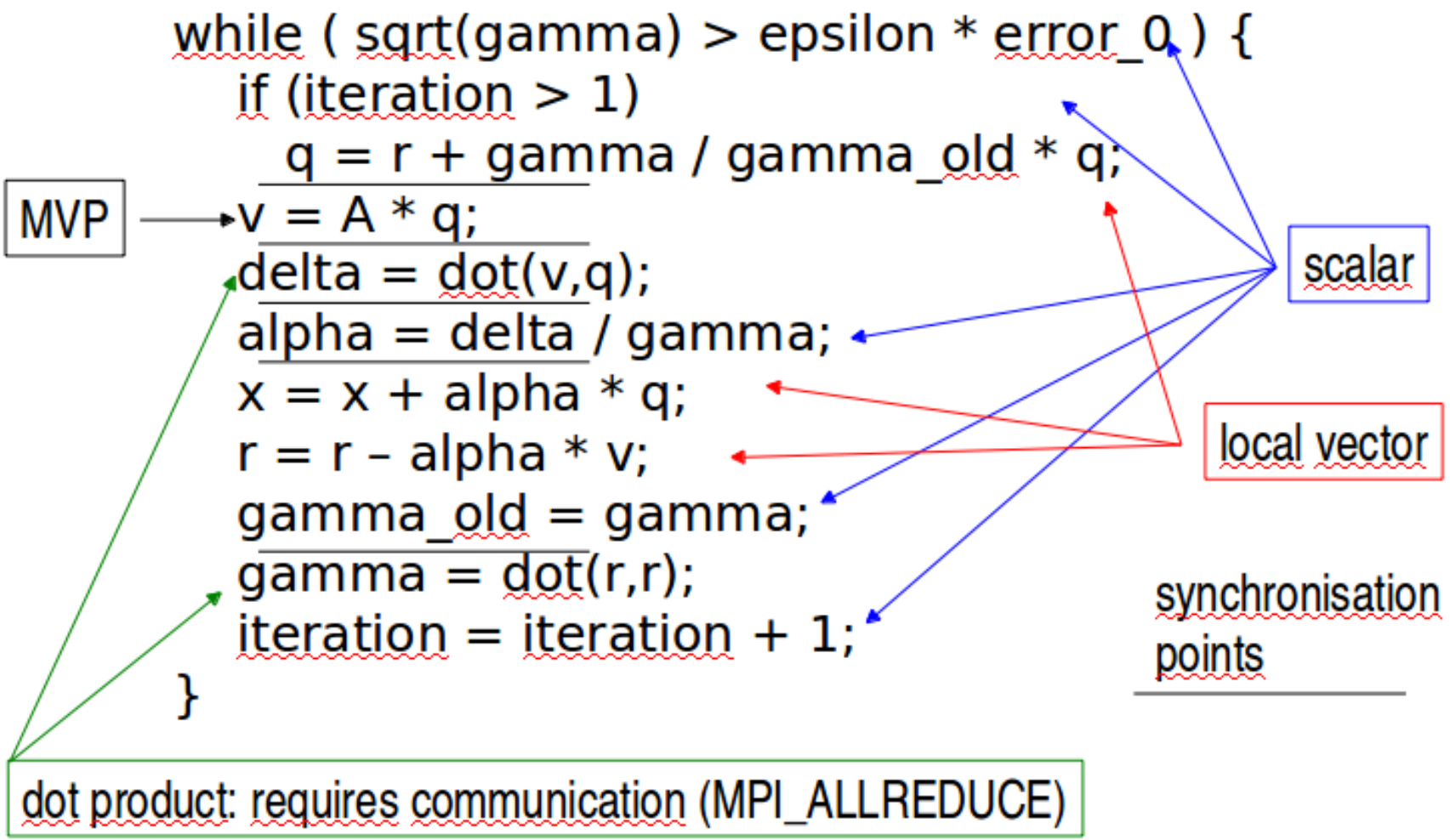
Very good approximation of CG after only two steps!

Important: small number of outliers combined with cluster.

Conjugate Gradients Summary

- To get fast convergence and reduce the number of iterations: $\rightarrow$ 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



Non-Blocking Collective Operations

- Use to hide communication!
- Allows overlap of numerical computations and communcations.
- 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 subf1
// computation and communication overlap
MPI_Iallreduce(sbuf1,rbuf1,SIZE,MPI_INT,MPI_SUM,MPI_COMM, &req);

// compute buf2 (independent of buf1)
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 requirments 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 requirments 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 $U_1 \to U_2 \to U_3 \to \ldots$ such that iteratively we can update the optimal solutions

$$x_1 \to x_2 \to x_3 \to \ldots \to A^{-1} b$$

using mainly matrix-vector products.

GMRES: Subspace

What subspace for fast convergence and easy computations?

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

Problem: b, Ab, A^2b, ... 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?

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

Problem: b, Ab, A^2b, ... 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 $j = 2 : m$:

$$\widetilde{u}_j := Au_{j-1} - \sum_{k=1}^{j-1}(u_k^T Au_{j-1})u_k = Au_{j-1} - \sum_{k=1}^{j-1} h_{k,j-1}u_k$$

$$u_j := \frac{\widetilde{u}_j}{\|\widetilde{u}_j\|_2} = \frac{\widetilde{u}_j}{h_{j,j-1}}$$

which is the standard orthogonalization method (Arnoldi method)

Matrix Form of Arnoldi ONB

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

Write this orthogonalization method in matrix form

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

Matrix Form of Arnoldi ONB

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

Write this orthogonalization method in matrix form

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

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

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

GMRES: Minimization

This leads to minimization problem

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

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 $\widetilde{H}_{m+1,m}$.

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

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

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

GMRES

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

Solution:

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

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).