Numerical Simulation and Scientific Computing I

# Lecture 6:
# Numerical Linear Algebra II – Krylov Methods

Xaver Klemenschits, Paul Manstetten, and
Josef Weinbub

Institute for Microelectronics
TU Wien

nssc@iue.tuwien.ac.at

WS 2022

## Quiz – Question 1

- Imagine you track the balance of a bank account using a single precision floating point number representing EUR. Starting with balance 0 EUR, each day 0.1EUR are transferred to the account, after how many days will the balance not increase anymore?

## Poll 1

- What will be the order of magnitude of the EUR balance when it stops increasing (epsilon ~ $10^{-7}$)?
  - A) $10^{3}$
  - B) $10^{6}$
  - C) $10^{7}$
  - D) $10^{9}$
  - E) $10^{12}$

## Quiz – Question 1

- Back-of-the-envelope calculation:
  - When will the scaled epsilon be on the order of 0.1?

$$x * 10^{-7} \approx 0.1$$
$$\rightarrow x \approx 10^{6}$$

# Quiz – Question 1

```
{
  std::cout << "account balance" << std::endl;
  float f = 0.0f;
  double feps = std::numeric_limits<float>::epsilon();
  int days = 0;

  while (f != f + 0.1f) {
    f += 0.1f;
    ++days;
  }
  std::cout << "feps=" << feps << std::endl;
  std::cout << "balance=" << f << std::endl;
  std::cout << "days=" << days << std::endl;
  std::cout << "years=" << days / 365.0 << std::endl;
  std::cout << "decades=" << days / 365.0 / 10.0 << std::endl;
}
// output:
// arithmetics
// account balance
// feps=1.19209e-07
// balance=2.09715e+06
// days=18073720
// years=49517
// decades=4951.7
```

## Quiz – Question 2

- What are potential advantages/disadvantages of using BLAS/LAPACK or Eigen?

## Quiz – Question 2

- What are potential advantages/disadvantages of using BLAS/LAPACK or Eigen?

  - License (modified BSD vs MPL2)
  - Language (Fortran vs C++)
  - Speed (similar, but vs. implementation from scratch)
  - BLAS/LAPACK: more API than implementation

## Quiz – Question 3

- How could you calculate an upper bound for the spectral radius for a given iteration matrix of the Jacobi method?

# Quiz – Question 3

- Spectral radius (condition for convergence)
  - $\rho\left(D^{-1}(A-D)\right) < 1$

- Smaller than any norm
  - $\rho(X) \leq \|X\|$
  - Why? $\|X\| = \max\limits_{x \neq 0} \frac{\|Xx\|}{\|x\|}$

- Choosing the maximum norm:
  - $\|D^{-1}(A-D)\|_\infty < 1 \rightarrow |a_{ii}| > \sum_{j \neq i} |a_{ij}|$
  - Strict diagonal dominance is a sufficient condition for convergence

## Outline

- Motivation
- Data Structures
- Krylov Subspace
- Methods
  - GMRES
  - Conjugate Gradient (CG)
  - Bi-CG(STAB)
- Preconditioners

## Goal

- Motivate the introduction of iterative methods
  - Key concept: sparsity
- Get a “feeling” of Krylov subspace methods
  - Including a short tour of popular choices
- Explore their limitations
- Some practical considerations
  - Implications for memory and data structures

## What we will NOT cover

- Mathematical proofs
- Convergence analyses
- Eigenvalue problems
- Direct sparse solvers
- Multigrid
- Detail!

## All that we care about

$$Ax = b$$

# Take-home message

- Krylov methods can be useful when there are large, sparse matrices
  - We do not want to lose sparsity
  - We do not want to pay the $O(n^3)$ price

- Convergence can be hard
  - Choosing the correct method for the problem
  - Equal parts "art" and "science" – experience & experimentation are key!

- Preconditioners can significantly increase convergence rate
  - Also make your problem more stable

## Main References

- Iterative Krylov Methods for Large Linear Systems
  - Author: Henk A. van der Vorst
  - eBook available: https://catalogplus.tuwien.ac.at:443/UTW:UTW:TN_cambridge_s10_1017_CBO9780511615115

- Eigen documentation
  - https://eigen.tuxfamily.org/dox/group__TutorialSparse.html

## Additional References

- Numerical Linear Algebra
  - Authors: David Bau and Lloyd N. Trefethen
  - https://catalogplus.tuwien.ac.at:443/UTW:UTW:UTW_alma2143358810003336

- Matrix Computations
  - Authors: Gene H. Golub and Charles F. Van Loan
  - https://catalogplus.tuwien.ac.at:443/UTW:UTW:UTW_alma2149765900003336

- Iterative Methods for Sparse Linear Systems
  - Author: Yousef Saad
  - https://catalogplus.tuwien.ac.at:443/UTW:UTW:UTW_alma2154052560003336

## Libraries

- Any examples of linear algebra libraries?

## Libraries

- Eigen
  - “Eigen is a C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms”
- PETSc
  - “PETSc is a suite of data structures and routines for the scalable solution of scientific applications modeled by PDEs”
- SciPy
  - “SciPy is a Python-based ecosystem of open-source software for mathematics, science, and engineering”
- MTL4, Armadillo, Trilinos
  - Similar in spirit to Eigen
- SLEPc
  - “SLEPc, the Scalable Library for Eigenvalue Problem Computations, is a software library for the solution of large sparse eigenproblems on parallel computers”

## Recap – LU Decomposition

- For a non-singular $n \times n$ matrix $A$, we want to solve:

$$Ax = b$$

- A decomposition exists such that:

$$PA = LU$$

- Where:
  - $P$ is a permutation matrix
  - $L$ is lower triangular
  - $U$ is upper triangular

- Therefore:

$$Ax = b \Leftrightarrow Ly = Pb$$
$$Ux = y$$

## Motivation – Computational Resources

- LU decomposition
  - $O(n^3)$ for dense matrices.
  - In more reasonable scenarios (banded matrix) - $O(n^{2\frac{1}{3}})$

- If the computational power increases 1000x – only a 10x increase in problem size

## Sparsity

- Definitions:
  - $n$ is the matrix dimension -> $n^2$ total entries
  - $m$ is the number of non-zero entries
- Dense Matrix
  - $m \sim n^2$
  - Very common in scientific problems
  - e.g. 5 chemical species in a reaction

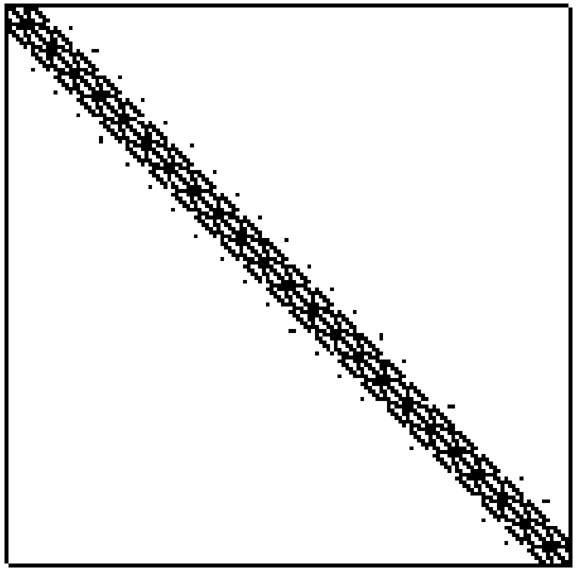
$$\begin{bmatrix} 1 & 4.6 & 0.7 & 0 & 3.8 \\ 9.3 & 1 & 8.5 & 3.7 & 0.7 \\ 2.3 & 6 & 2.7 & 7.6 & 5.9 \\ 1.2 & 0 & 0 & 4.8 & 9.4 \\ 7.1 & 10.9 & 5.6 & 1 & 1 \end{bmatrix}$$

- Sparse Matrix
  - $m \ll n^2$
  - Usually arise from discretization of integral or differential equations
  - Rule of thumb: if $n$ is very large, then it is probably an approximation to $\infty$

$$\begin{bmatrix} 1 & 4.6 & 0 & 0 & 0 \\ 0 & 0 & 8.5 & 3.7 & 0 \\ 0 & 6 & 2.7 & 0 & 0 \\ 0 & 4.6 & 0 & 4.8 & 9.4 \\ 0 & 0 & 5.6 & 0 & 1 \end{bmatrix}$$

## Motivation – Memory

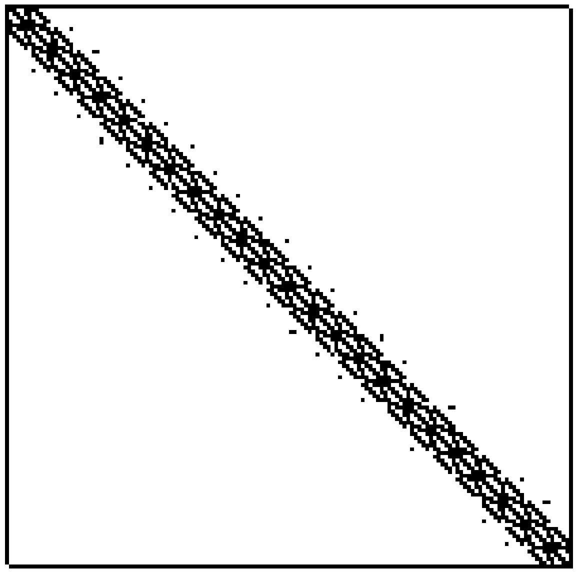
- Source: Matrix Market – A repository for test data
  - https://math.nist.gov/MatrixMarket/
  - Example: Matrix MAN 5976 (Structural Engineering)



- Structure plot: color the non-zero entries
- Bandwidth: maximum separation between non-zero entries
- Is it necessary to store the whole matrix?

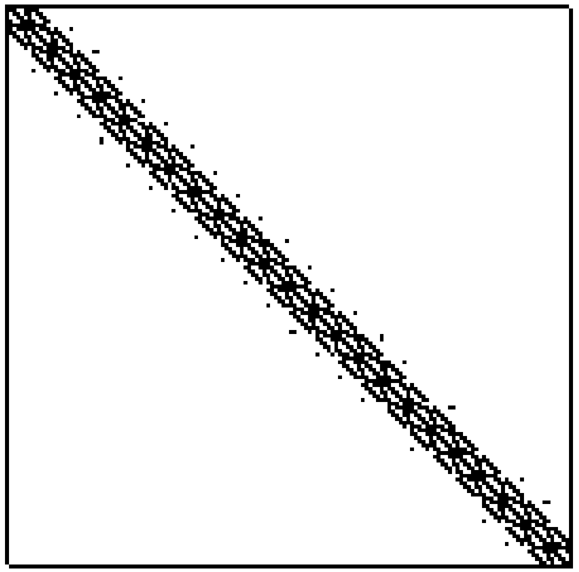
# Poll 2

- What will happen to the sparsity of the matrix with the following structure plot after applying LU factorization?
  - A) Keeps the same sparsity
  - B) Loses some sparsity
  - C) Becomes completely dense



## Poll 2

- What will happen to the sparsity of the matrix with the following structure plot after applying LU factorization?
  - A) Keeps the same sparsity
  - **B) Lose some sparsity**
    (depending on the pivoting!)
  - C) Become completely dense



## Quiz – Question 4

- Assume a large matrix has mostly zero entries, how to store it efficiently in terms of memory footprint?

## Quiz – Question 4

- Assume a large matrix has mostly zero entries, how to store it efficiently in terms of memory footprint?

We don’t want to store zeros!

## Storing Sparse Matrices - COO

- COO: Coordinate Format
  - Define 3 vectors of size $m$: $V$, $IA$ and $JA$
  - $V$ contains all non-zero entries
  - $IA$, $JA$ contain the corresponding $i$ and $j$ indices
  - Storage requirement: $3m$

$$\begin{bmatrix} 1 & 4.6 & 0 & 0 & 0 \\ 0 & 0 & 8.5 & 3.7 & 0 \\ 0 & 6 & 2.7 & 0 & 0 \\ 0 & 4.6 & 0 & 4.8 & 9.4 \\ 0 & 0 & 5.6 & 0 & 1 \end{bmatrix}$$

$$V = [1 \quad 4.6 \quad 6 \quad 4.6 \quad 8.5 \quad 2.7 \quad 5.6 \quad 3.7 \quad 4.8 \quad 9.4 \quad 1]^T$$

$$IA = [0 \quad 0 \quad 2 \quad 3 \quad 1 \quad 2 \quad 4 \quad 1 \quad 3 \quad 3 \quad 4]^T$$

$$JA = [0 \quad 1 \quad 1 \quad 1 \quad 2 \quad 2 \quad 2 \quad 3 \quad 3 \quad 4 \quad 4]^T$$

## Polls 3 & 4

- 3 - Which size would you generally use to store integers?
  - A) 16-bit
  - B) 32-bit
  - C) 64-bit

- 4 - Which size would you generally use to store floating-point numbers?
  - A) 16-bit
  - B) 32-bit
  - C) 64-bit

# Polls 3 & 4

- 3 - Which size would you generally use to store integers?
  - A) 16-bit
  - **B) 32-bit**
  - C) 64-bit

- 4 - Which size would you generally use to store floating-point numbers?
  - A) 16-bit
  - B) 32-bit
  - **C) 64-bit**

- What is the memory cost in bytes for a COO matrix with 16 entries?

## Polls 3 & 4

- 3 - Which size would you generally use to store integers?
  - A) 16-bit
  - **B) 32-bit**
  - C) 64-bit
- 4 - Which size would you generally use to store floating-point numbers?
  - A) 16-bit
  - B) 32-bit
  - **C) 64-bit**
- What is the memory cost in bytes for a COO matrix with 16 entries?
  - 16 * (2*4 (ints) + 8 (double)) = 256 B

## Storing Sparse Matrices - CCS

- CCS: Compressed Column Storage
  - $V$, $IA$ are the same as in COO
  - $JA$ is a vector of size $(n+1)$
  - $JA$ points to the index of $IA$ where the next column stars
  - Storage requirement: $2m+n+1$

$$\begin{bmatrix} 1 & 4.6 & 0 & 0 & 0 \\ 0 & 0 & 8.5 & 3.7 & 0 \\ 0 & 6 & 2.7 & 0 & 0 \\ 0 & 4.6 & 0 & 4.8 & 9.4 \\ 0 & 0 & 5.6 & 0 & 1 \end{bmatrix}$$

$$V = [1 \quad 4.6 \quad 6 \quad 4.6 \quad 8.5 \quad 2.7 \quad 5.6 \quad 3.7 \quad 4.8 \quad 9.4 \quad 1]^T$$

$$IA = [0 \quad 0 \quad 2 \quad 3 \quad 1 \quad 2 \quad 4 \quad 1 \quad 3 \quad 3 \quad 4]^T$$

$$JA = [0 \quad 1 \quad 4 \quad 7 \quad 9 \quad 11]^T$$

## Storing Sparse Matrices – Special Cases

- CRS: Compressed Row Storage
  - Same as CCS, exchanging row and column (and the roles of $IA$, $JA$)

- Eigen:
  - Defaults to column-major storage
  - Variation of CCS
  - Adds a buffer between each column to insert new elements
  - Additional vector to store the number of non-zero entries
  - `SparseMatrix::makeCompressed()` transforms to standard CCS

## Classical Iterative Methods

- Iterative Method: given $Ax = b$, generate a sequence $\{x^{(k)}\}$ which converges to $x = A^{-1}b$
  - Without explicitly calculating $A^{-1}$ - Why?

- Instead of solving $Ax = b$, replace by a simpler $Kx_0 = b$
  - $x_0$ approximates $x$ as $x = x_0 + z$
  - Plugging back in: $A(x_0 + z) = b \ \rightarrow Az = b \ - Ax_0$
- Use the simpler $K$ again
  - $Kz_0 = b - Ax_0$ leading to a new approximation $x_1 = x_0 + z_0$
  - In general: $x_{i+1} = x_i + K^{-1}(b - Ax_i)$

- Examples:
  - Jacobi: $K = D$
  - Gauss-Seidel: $K = D + L$

## Richardson Iteration

- Even simpler approximation: $K = I$

$$x_{k+1} = x_k + K^{-1}(b - Ax_k)$$
$$x_{k+1} = x_k + b - Ax_k$$
$$x_{k+1} = x_k + r_k$$

- One way of rewriting

$$x_{k+1} = (I - A)x_k + b$$

- Residuals

$$x_{k+1} = x_k + r_k$$
$$\Leftrightarrow r_{k+1} = (I - A)r_k = (I - A)(I - A)r_{k-1}$$
$$\Leftrightarrow r_{k+1} = (I - A)^{k+1} r_0$$

- Convergence when

$$\|I - A\| < 1$$

# Richardson Iteration - Subspace

- The total iteration

$$x_{k+1} = x_k + r_k$$
$$x_{k+1} = x_{k-1} + r_{k-1} + r_k$$
$$x_{k+1} = x_0 + r_0 + \cdots + r_k$$

- Finally, plugging $r_i$

$$x_{k+1} = x_0 + \sum_{i=0}^{k} (I - A)^i \, r_0 = x_0 + z$$

- We can assume w.l.o.g. $x_0 = 0$

$$x_{k+1} \in span\{r_0, Ar_0, \dots, A^k r_0\}$$

## Krylov Subspace - Definition

- Definition: given a non-zero vector $v$ and a non-singular square matrix $A$, the $m$-dimensional Krylov subspace is:

$$K^m(A; v) \coloneqq span(v, Av, A^2v, \dots, A^{m-1}v)$$

- Krylov Subspace Methods try to:
  - Use all information available in $K^m$
  - Construct a solution according to some “optimality”
  - More than one option – different methods!

## Optimality Approaches

Ritz-Galerkin: Construct $x_k$ s.t. the residual $r_k = b - Ax_k$ is orthogonal to the current subspace

$$r_k \perp K^k(A, r_0)$$

Ex: Conjugate Gradients, Lanczos

Minimum norm residual: Find $x_k$ from least squares

$$\min \|b - Ax_k\|_2; \; x_k \in K^k(A; r_0)$$

Ex: MINRES, GMRES

Petrov-Galerkin: Construct $x_k$ s.t. the residual $r_k = b - Ax_k$ is orthogonal to some other subspace

Ex: $K^k(A^T; s_0) \rightarrow$ Bi-CG

Also: minimum norm error

## Conjugate Gradients (CG) - Motivation

- Assume that A is symmetric positive definite:

$$A = A^T; \;\; x^T A x > 0 \; \forall x \backslash 0$$

- We can define a function $\phi(x)$

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

- Such that:

$$Ax = b \;\Leftrightarrow\; \min_x \phi(x)$$

- Algorithm idea: from a given $x_k$, construct $x_{k+1}$ from a search direction $p_k$ and optimality criterion $\alpha$

$$x_{k+1} = x_k + \alpha p_k$$

  - If $p_k = r_k$, we have the method of steepest descent

## CG – A-norm

- Definition:

$$\|x\|_A = \sqrt{x^t A x}$$

- We can define the error

$$e_k = x_k - x_*$$

- It can be shown that

$$\phi(x_k) = \frac{1}{2}\|e_k\|_A + \phi(x_*)$$

- CG is [Trefethen & Bau]: a system of recurrence formulas generating the unique sequence $x_k \in K^k$ minimizing $\|e_k\|_A$
  - Trick – Ritz-Galerkin: $r_k \perp K^k(A, r_0)$. Therefore it is “conjugate”!
  - Minimizes the A-norm error, not the residual!

# CG – Basic Algorithm (Hestenes & Stiefel)

```
$x_0 = 0; r_0 = b; p_0 = r_0$
for $k = 1,2,\ldots$
    $\alpha_k = \frac{r_{k-1}^T r_{k-1}}{p_{k-1}^T A p_{k-1}}$        1) step length
    $x_k = x_{k-1} + \alpha_k p_{k-1}$        2) approximate solution
    $r_k = r_{k-1} - \alpha_k A p_{k-1}$        3) residual
    $\beta_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$        4) improvement this step
    $p_k = r_k + \beta_k p_{k-1}$        5) search direction
```

Plus: convergence termination criterion!

## Polls 5 & 6

```
$x_0 = 0; r_0 = b; p_0 = r_0$
for $k = 1,2,\ldots$
    $\alpha_k = \frac{r_{k-1}^T r_{k-1}}{p_{k-1}^T A p_{k-1}}$        1) step length
    $x_k = x_{k-1} + \alpha_k p_{k-1}$        2) approximate solution
    $r_k = r_{k-1} - \alpha_k A p_{k-1}$        3) residual
    $\beta_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$        4) improvement this step
    $p_k = r_k + \beta_k p_{k-1}$        5) search direction
```

- Poll 5: How many matrix-vector products per CG iteration?
- Poll 6: How many vector-vector products per CG iteration?

## Polls 5 & 6

```
$x_0 = 0; r_0 = b; p_0 = r_0$
for $k = 1,2,\ldots$
    $\alpha_k = \frac{r_{k-1}^T r_{k-1}}{p_{k-1}^T A p_{k-1}}$        1) step length
    $x_k = x_{k-1} + \alpha_k p_{k-1}$        2) approximate solution
    $r_k = r_{k-1} - \alpha_k A p_{k-1}$        3) residual
    $\beta_k = \frac{r_k^T r_k}{r_{k-1}^T r_{k-1}}$        4) improvement this step
    $p_k = r_k + \beta_k p_{k-1}$        5) search direction
```

- Poll 5: How many matrix-vector products per CG iteration? **1**
- Poll 6: How many vector-vector products per CG iteration? **3**

## CG – Strengths and Weaknesses

- Strengths
  - Only one matrix-vector operation per iteration
  - Simple implementation
  - Very fast convergence (if the eigenvalues are well distributed…)
  - Low memory requirements
- Weaknesses
  - Only symmetric positive definite (s.p.d.) matrices

## Krylov Subspace - Definition

- Definition: given a non-zero vector $v$ and a non-singular square matrix $A$, the $m$-dimensional Krylov subspace is:

$$K^m(A;v) \coloneqq span(v, Av, A^2v, \dots, A^{m-1}v)$$

- Krylov Subspace Methods try to:
  - Use all information available in $K^m$
  - Construct a solution according to some “optimality”
  - More than one option – different methods!

## Bases on Krylov Subspace – Arnoldi Iteration

- Problem with $A^k r_0$ -> becomes almost linearly dependent

- Arnoldi algorithm (Modified Gram-Schmidt):

```
$v_1 = r_0/\|r_0\|_2$                                  first Krylov vector
for $j = 1, \dots, m-1$
        $t = Av_j$                                     new candidate vector
        for $i = 1, \dots, j$
                $h_{i,j} = v_i^T t$                    subtract the projections on previous vectors
                $t = t - h_{i,j} v_i$
        $h_{j+1,j} = \|t\|_2$
        $v_{j+1} = t/h_{j+1,j}$
```

- In practice: Householder reflections

## Bases on Krylov Subspace – Hessenberg Matrices

- We just generated an $m \times m$ matrix $H_m$ such that:

$$V_m^T A V_m = H_m$$

- $H_m$ is upper Hessenberg

$$H_m = \begin{bmatrix} \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ \blacksquare & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ 0 & \blacksquare & \blacksquare & \blacksquare & \blacksquare \\ 0 & 0 & \blacksquare & \blacksquare & \blacksquare \\ 0 & 0 & 0 & \blacksquare & \blacksquare \end{bmatrix}$$

- Also note:

$$A V_{m-1} = V_m \bar{H}_m$$

where $\bar{H}_m$ has an additional row with only $h_{m+1,m}$

## Generalized Minimal Residual Method (GMRES)

- Objective: $\min \|b - Ax_k\|_2;\ x_k \in K^k(A; r_0)$

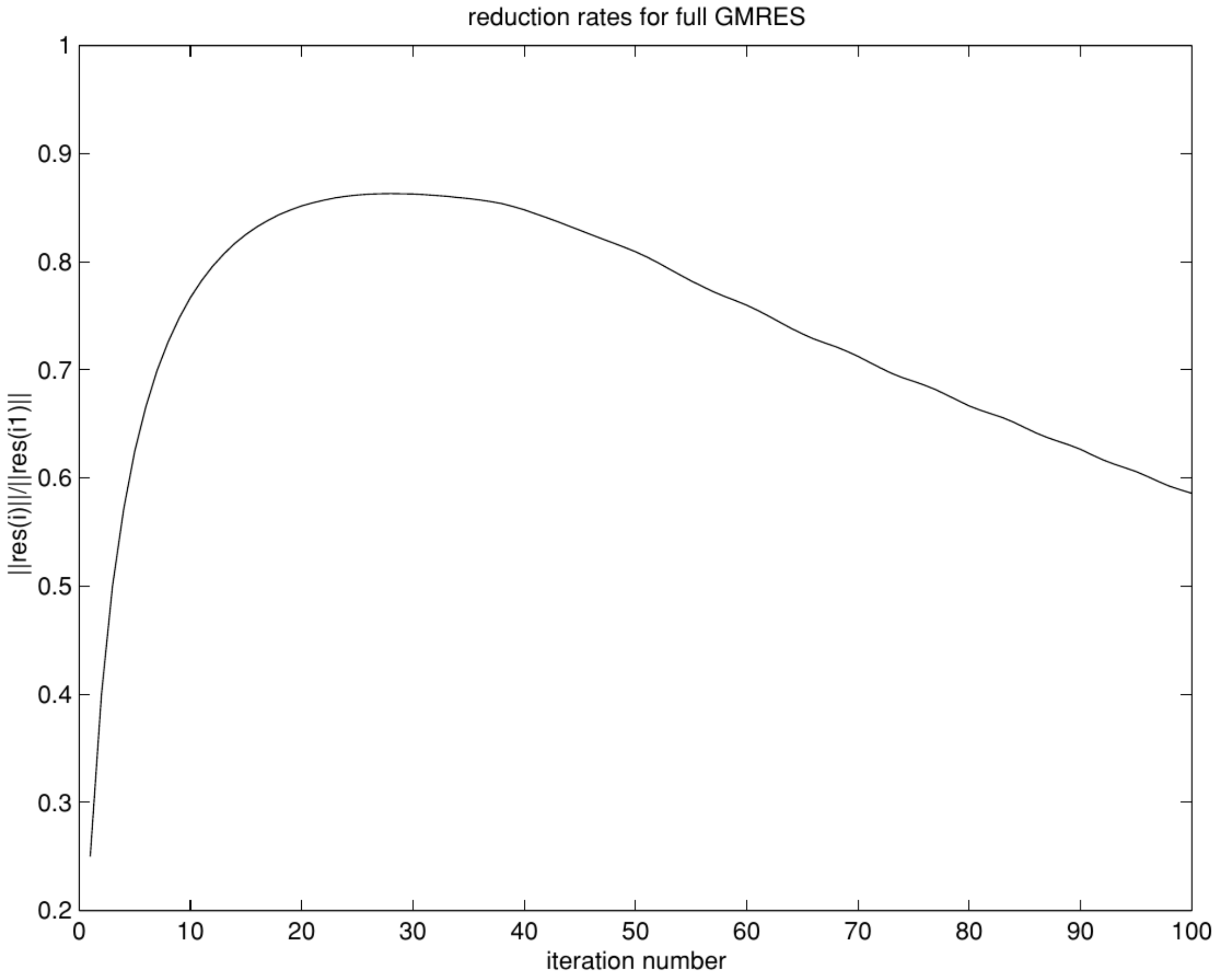
- Approach:
  - 1 - Generate Basis $< Arnoldi\ iteration >$
  - 2 - Optimality Constraint $\min_y \|\beta e_1 - \bar{H}_m y\|_2;\ \beta = \|r_0\|_2$
  - 3 - Solution construction $x_m = x_0 + V_m y$

- Key insight of GMRES: introduce the optimality constraint into the basis generation
  - Usually implemented by Givens rotations during the base generation

- Restarted GMRES: GMRES(m)
  - Only perform GMRES up to dimension m, then restart with current solution $x_m$

# GMRES – Convergence

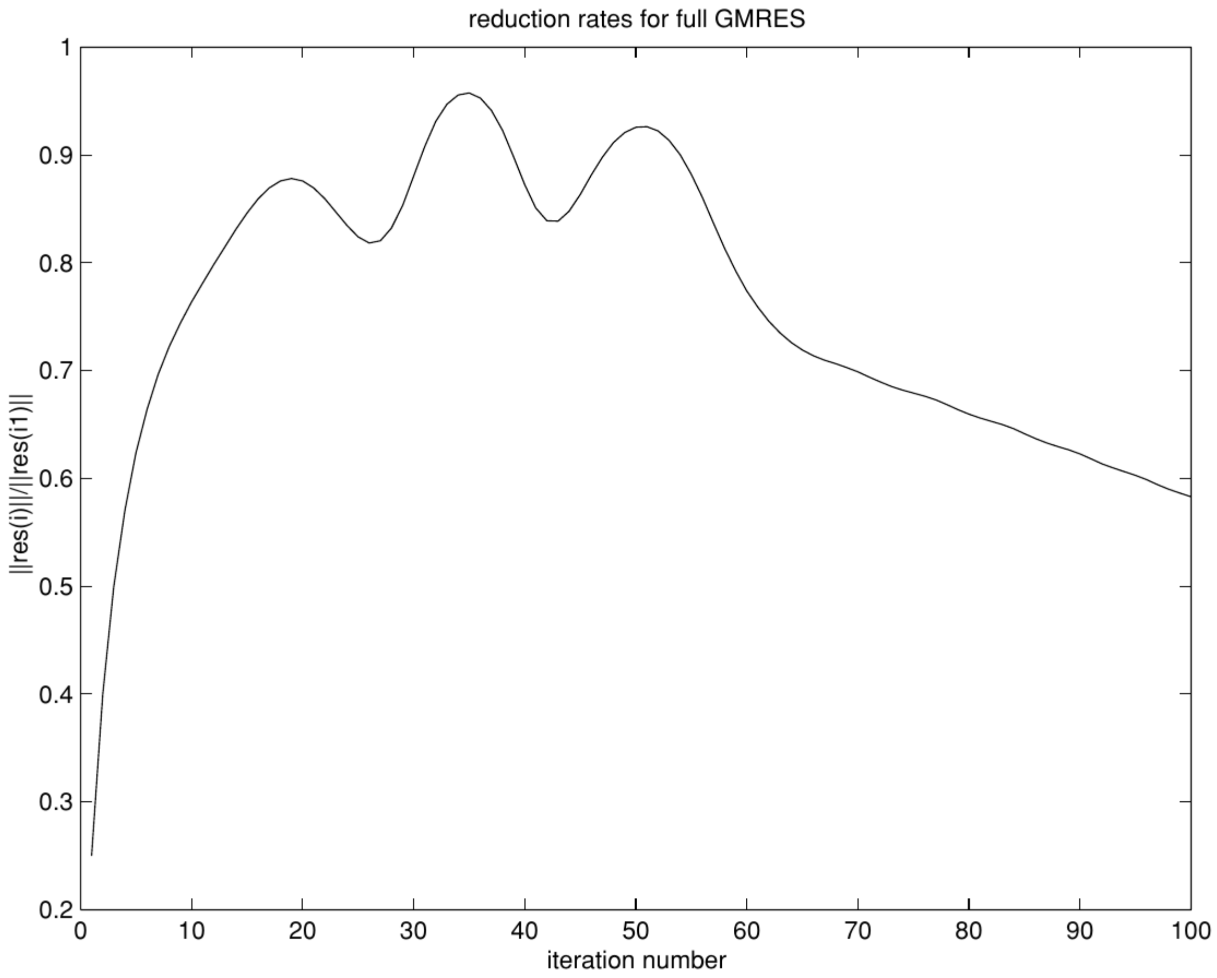
- Matrix with uniform real spectrum



Source: van der Vorst: Iterative Krylov Methods for Large Linear Systems, 1st ed., 2003

## GMRES – Convergence

- Defective (non-diagonalizable) matrix



Source: van der Vorst: Iterative Krylov Methods for Large Linear Systems, 1st ed., 2003

## GMRES – Strengths and Weaknesses

- Strengths
  - The residual is non-increasing
  - Only “good” breakdowns (if $x_{j<m}$ is already the exact solution)
  - Exact for $m = n$
  - In practice, it is the most robust method

- Weaknesses
  - Memory requirement (dense in $m$)
  - Also not the fastest
  - Not in Eigen! (although: Eigen-unsupported)
  - Very hard to prove convergence in arbitrary cases

## Bi-Conjugate Gradients (Bi-CG)

- Is it possible to avoid building the $m$ dimensional Hessenberg matrix?
  - We would like a simple recursion like CG, but for non-s.p.d. matrices!

- Insight: build a similar scheme to CG, but searching now on the subspace $K^k(A^T; s_0)$
  - Petrov-Galerkin

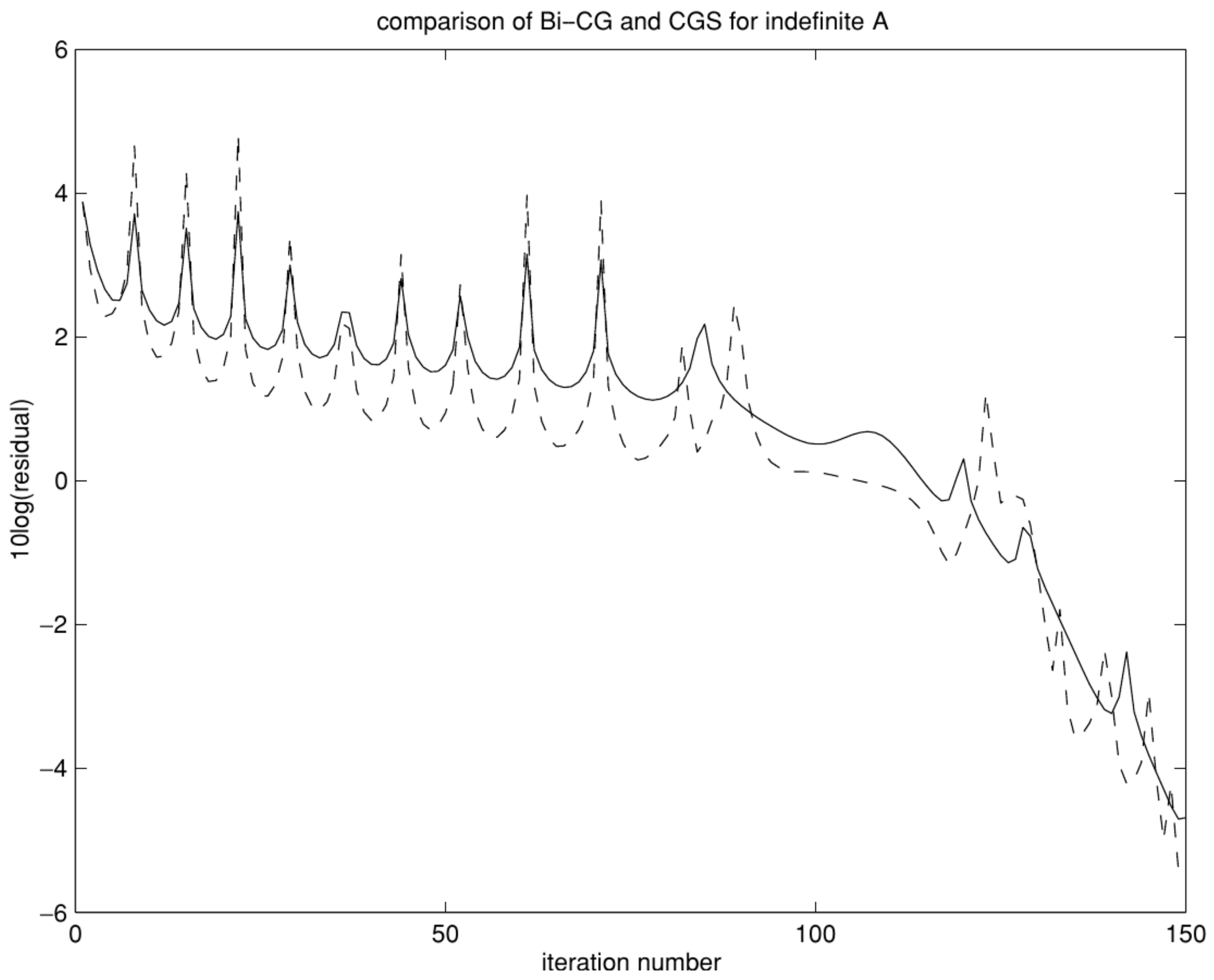
- Disadvantages:
  - Requires computing $A^T x$
  - Subject to serious breakdown: if the dot product between one vector in $K^k(A^T; s_0)$ and other in $K^k(A; r_0)$ (where the solution is being built) is zero, the method fails.
  - Convergence can be erratic

# BiCG – Basic Algorithm

```
$x_0 = 0; p_0 = r_0 = b; q_0 = s_0 = arbitrary$
for $k = 1,2,\ldots$
    $\alpha_k = \frac{s_{k-1}^T r_{k-1}}{q_{k-1}^T A p_{k-1}}$ ⟵ serious breakdown
    $x_k = x_{k-1} + \alpha_k p_{k-1}$
    $r_k = r_{k-1} - \alpha_k A p_{k-1}$
    $s_k = s_{k-1} - \alpha_k A^T q_{k-1}$
    $\beta_k = \frac{s_k^T r_k}{s_{k-1}^T r_{k-1}}$
    $p_k = r_k + \beta_k p_{k-1}$
    $q_k = s_k + \beta_k q_{k-1}$
```

# BiCG – Erratic Convergence

- Indefinite matrix



Source: van der Vorst: Iterative Krylov Methods for Large Linear Systems, 1st ed., 2003

## Bi-CG stabilized (Bi-CGSTAB)

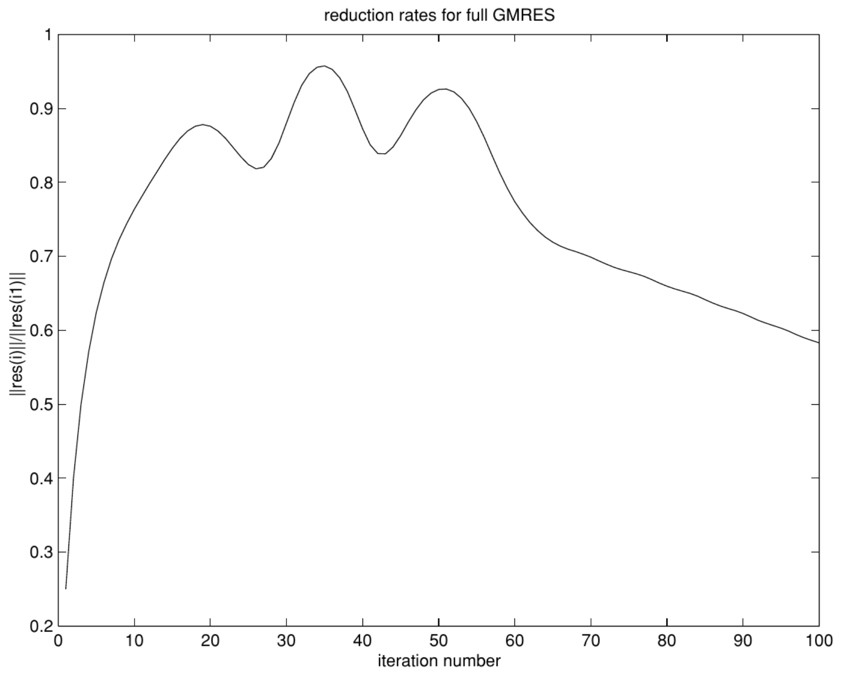
- Problems Bi-CGSTAB addresses:
  - Avoids computing $A^T x$
  - “Smoothens” convergence
- Intuition: combine a GMRES(1) step after each Bi-CG step
  - Natural extension: Bi-CGSTAB($l$) uses GMRES($l$)
- Still susceptible to serious breakdown!
- We will stop at that, the method is (even more) complicated…
  - But: available on EIGEN!

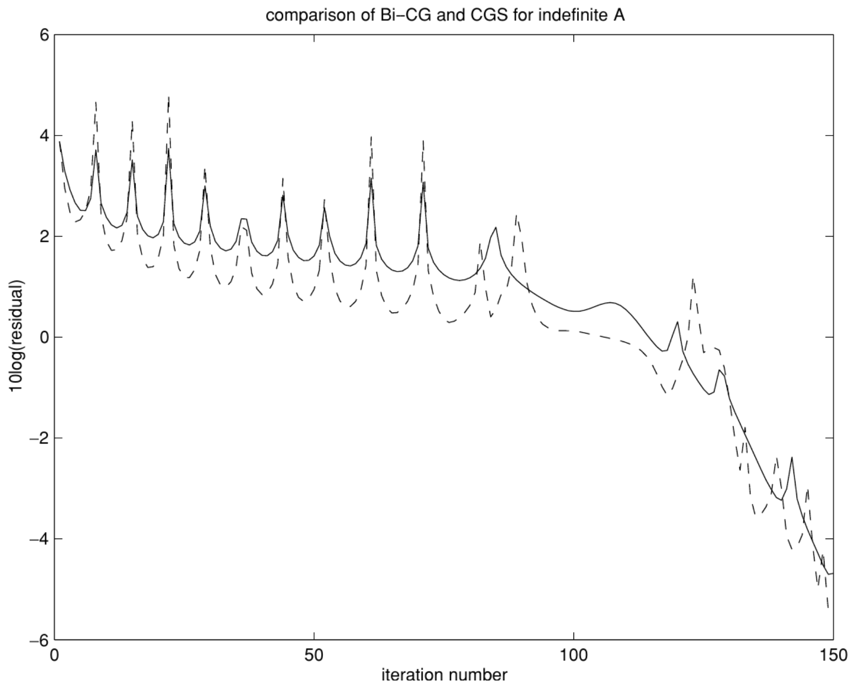
## Quiz – Question 5

- What is the benefit of preconditioning a problem before solving it?

# Preconditioners - Motivation

- We have seen problematic convergence behavior depending on the spectrum of $A$





- Rule of thumb: iterative methods perform better when the eigenvalues are clustered

Source: van der Vorst: Iterative Krylov Methods for Large Linear Systems, 1st ed., 2003

## Preconditioners - Definition

- Construct a matrix $K$ such that the system

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

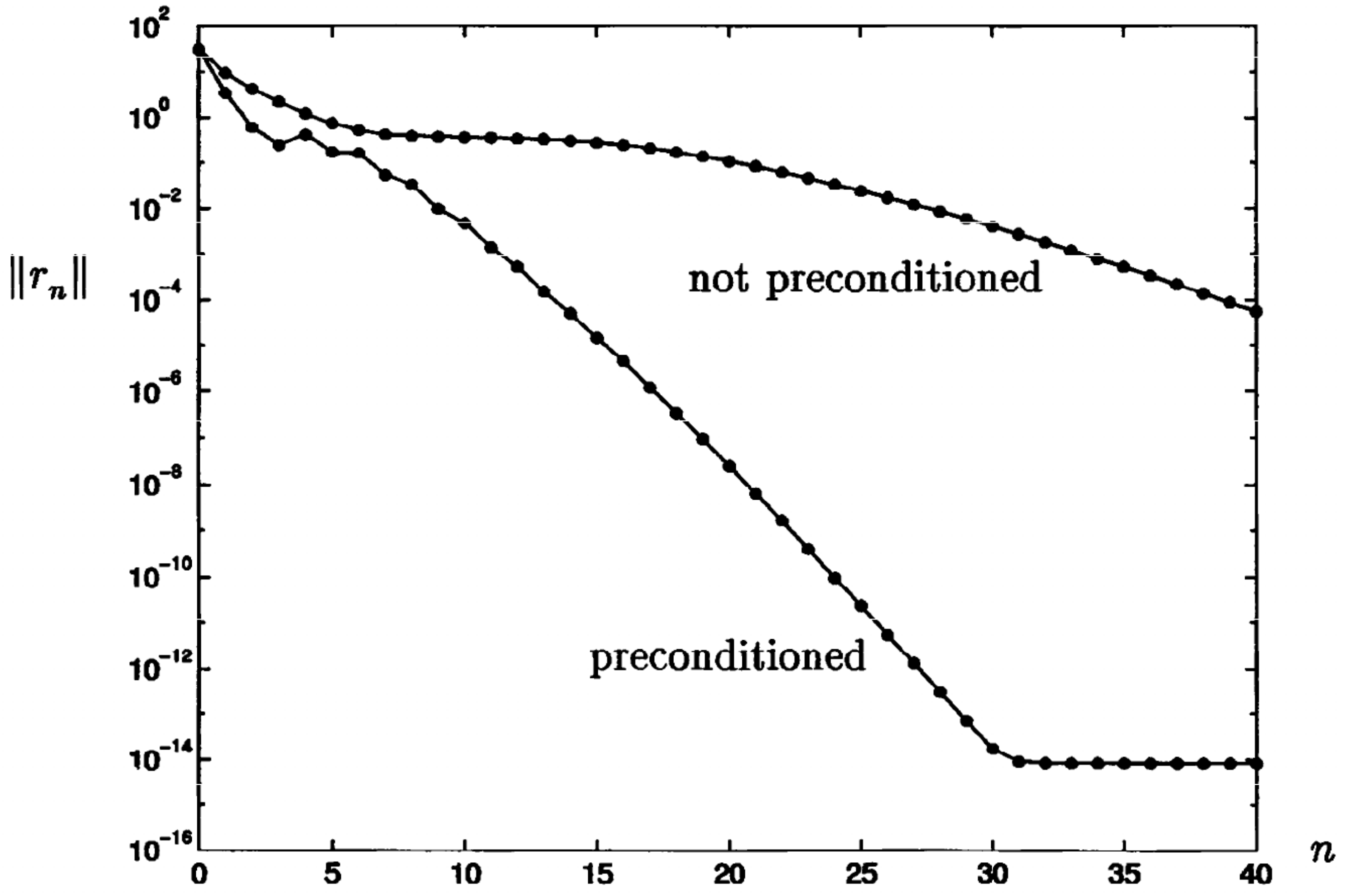
is more easily solvable -> $K^{-1}A$ has better spectral properties

- Note
  - $K$ is in some sense an approximation to $A$
  - $K^{-1}$ is simple to compute: We do not want to store $K^{-1}A$!
  - $Ky = z$ is simpler to solve
  - Also: the Jacobi method is a preconditioned Richardson iteration

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

# Preconditioners - Example

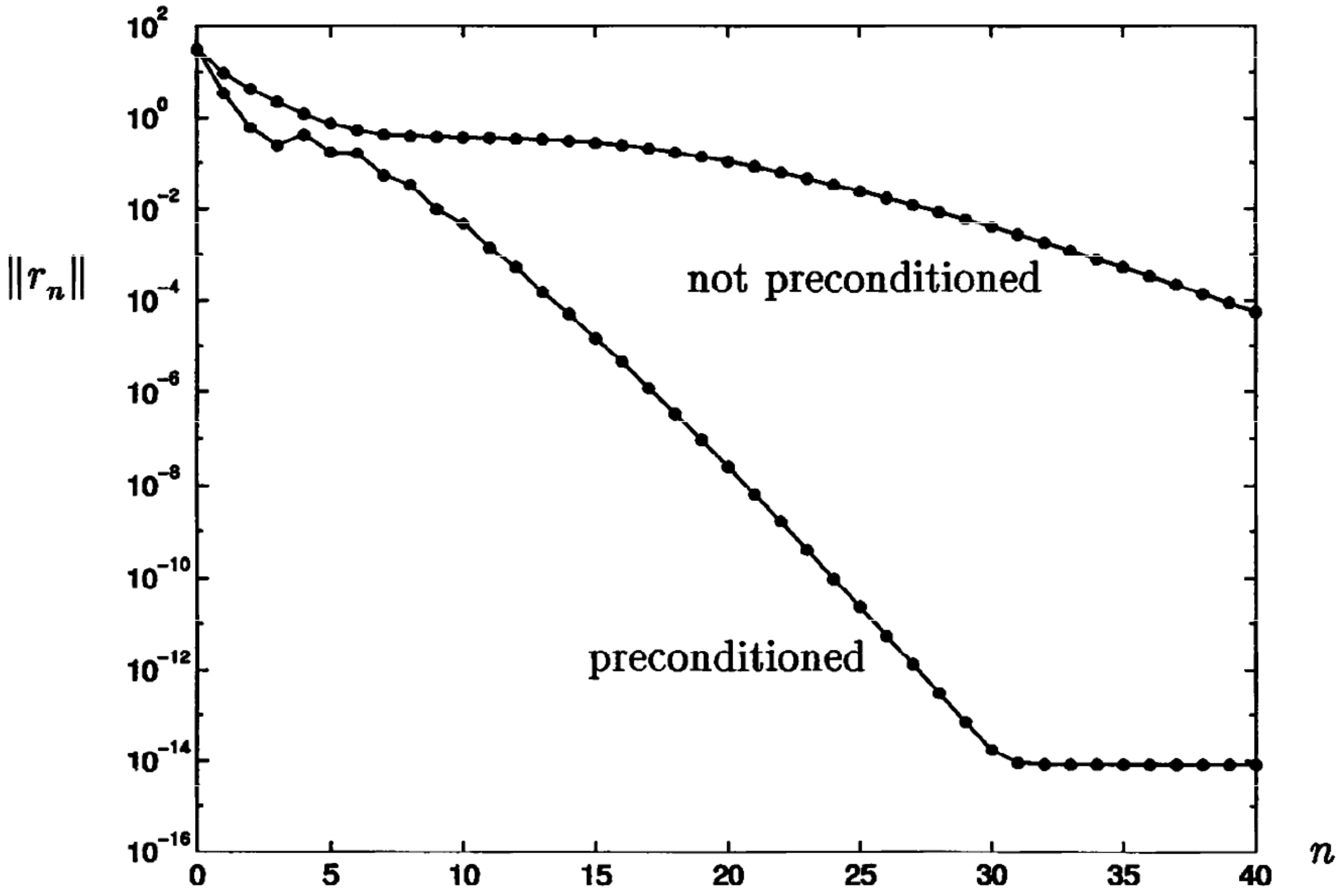
- Diagonal scaling/Jacobi preconditioner: $K = diag(A)$
  - Applied to the CG method



Source: Trefethen, Bau: Numerical Linear Algebra, 1st ed., 1997

# Poll 7

- Are both methods converging?



Source: Trefethen, Bau: Numerical Linear Algebra, 1st ed., 1997

## Preconditioners - Types

- Left Preconditioning

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

  - Note: this means that the residual changes: $\overline{r_0} = K^{-1}(b - Ax_0)$

- Right Preconditioning

$$AK^{-1}Kx = b$$

  - Note: now the solution lies in a different Krylov subspace

- Mixed/Two-sided Preconditioning

$$K_1^{-1}AK_2^{-1}K_2^{-1}x = K_1^{-1}b$$

## Incomplete LU factorization

- We know that, given an $LU$ factorization, it is easy to solve

$$LUx = b$$

- What if we calculate a simpler, incomplete LU?

$$A \approx LU$$

- Simplest idea: only calculate $L$ and $U$ for elements with the same indices that are non-zero in $A$
  - Keep the same sparsity
  - Does not always help convergence…

- More advanced schemes: set tolerances to drop elements
  - Depends on the heuristics…

## Note – Direct solvers are still relevant!

### The Sparse LU Challenge

- Given a square $A$, find the permutation matrices $P$ and $Q$ such that the factorization $PAQ^T = LU$ is:
  - Reasonably stable
  - $L$ and $U$ are close to being optimally sparse

- Many approaches
  - Example: http://eigen.tuxfamily.org/dox/classEigen_1_1SparseLU.html

- Can benefit from BLAS!

- Some sparsity will be lost…

## Take-home message – version 2

- Krylov methods can be useful when there are large, sparse matrices
    - We do not want to lose sparsity
    - We do not want to pay the $O(n^3)$ price
    - *But not always better than direct methods…*

- Convergence can be hard
    - Choosing the correct method for the problem
    - Equal parts “art” and “science” – experience & experimentation are key!
    - *Some idea about the spectrum of the matrix is helpful*

- Preconditioners can significantly increase convergence rate
    - Also make your problem more stable
    - *This is where a lot of mathemagic is hidden*

## Rules of Thumb for Choosing Methods

- If the matrix is symmetric positive definite:
  - Conjugate Gradients
- If the problem is “nasty”:
  - GMRES + lots of memory + lots of computing time
- If performance is key:
  - Bi-CGSTAB (or other hybrid methods) + lots of robustness testing
- Always investigate preconditioners
  - But remember that they involve trial-and-error
- If you can afford the memory, test sparse direct methods

## Quiz

- Q1: Represent the following matrix in the compressed row storage (CRS) format

$$\begin{bmatrix} 1 & 4.6 & 0 & 0 & 0 \\ 0 & 0 & 8.5 & 3.7 & 0 \\ 0 & 6 & 2.7 & 0 & 0 \\ 0 & 4.6 & 0 & 4.8 & 9.4 \\ 0 & 0 & 5.6 & 0 & 1 \end{bmatrix}$$

- Q2: Under which conditions would you use the CG method instead of GMRES?
- Q3: What information from a matrix $A$ could be useful to help choosing an adequate solver?
- Q4: Is it possible to generate true random numbers from a digital computer (e.g. x86 architecture)?
- Q5: How would you estimate the value of $\pi$ using random numbers?

## Next stop

- Random Number Generation
- Introduction to Monte Carlo