Scientific Computing I

Solving Linear Systems With Sparse Matrices

Martin Köhler

Computational Methods in Systems and Control Theory (CSC) Max Planck Institute for Dynamics of Complex Technical Systems

Winter Term 2024/2025

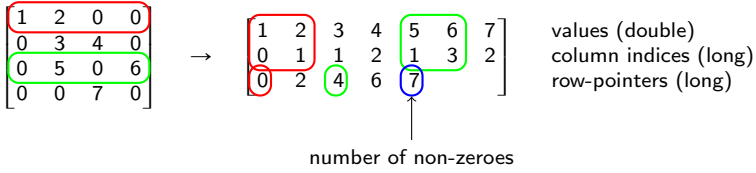
Preliminaries

Recall

Preliminaries

Recall

- sparse matrix: $A \in \mathbb{R}^{n \times n}$, such that $y = Ax$ can be computed in $\mathcal{O}(n)$ complexity.
- storage:
  - only non-zero entries are stored,
  - indirect indexing is mandatory for minimal storage requirements,
  - e.g., CSR (compressed sparse row storage, with C/zero based indexing)



Issues

Preliminaries

Issues – Cache

Indirect indexing requires the value, index and row-pointer vectors to reside in the cache simultaneously for optimal performance.
Consider:

- 64 bit architecture
- in average 10 entries per row
- 4MB cache
- $A \in \mathbb{R}^{24\,000 \times 24\,000}$

Required storage:

$$\begin{aligned}(24\,000 + 240\,000 + 240\,000) \times 8 \text{ Bytes} &= 504\,000 \times 8 \text{ Bytes} \\ &= 4032 \text{ kBytes}\end{aligned}$$

That means we have $(4\,096 - 4\,032)$ kBytes$=$ 64 kByte of cache left for instructions in $y = Ax$. In applications one easily wants to work with $n = 10^6 \ldots 10^8$, which on modern computers usually easily fits into RAM. The execution speed of operations with A are thus strictly limited by data transfer rate from the main memory to the caches.

Preliminaries

Issues – Fill in

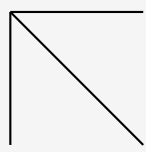
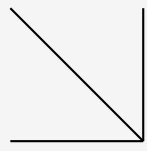
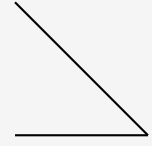
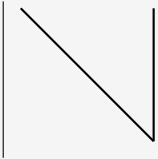
Another important issue with sparse matrices arises with direct solvers. These require matrix factorizations. However, it can not be guaranteed that the factors stay sparse if the matrix A is sparse. Usually the factors get a certain amount of new entries. The new entries are referred to as **fill** or **fill-in**. We will see more details on this phenomenon later.

Preliminaries

Issues – Fill in

Example 8.1 (Fill-In)

The diagrams below show the non-zero entry distribution in A, L and U for A sparse and $A = LU$.

	A	L	U
worst case			
best case			

Definitions

Definition 8.2 (pattern)

Let $A \in \mathbb{R}^{n \times n}$ be a matrix.
We call the set

$$\mathcal{P}(A) = \{(i,j) : a_{ij} \neq 0\}$$

the **pattern** of A.
Furthermore, we define

$$\mathcal{P}_R(A,i) = \{j : a_{ij} \neq 0\}$$

as the pattern of the i-th row of A.

Definition 8.3 (structural rank)

Let $\mathcal{P}(A) \subset \mathbb{N}^2$ be a pattern of a matrix $A \in \mathbb{R}^{n \times n}$. The number

$$\mathrm{rk}_S(A) = \max\{\mathrm{rank}(B) : B \in \mathbb{R}^{n \times n} \text{ with } \mathcal{P}(B) = \mathcal{P}(A)\}$$

is called the **structural rank** of A.
If $\mathrm{rk}_S(A) < n$, then A is called **structural rank deficient**

Definition 8.3 (structural rank)

Let $\mathcal{P}(A) \subset \mathbb{N}^2$ be a pattern of a matrix $A \in \mathbb{R}^{n \times n}$. The number

$$\mathrm{rk}_S(A) = \max\{\mathrm{rank}(B) : B \in \mathbb{R}^{n \times n} \text{ with } \mathcal{P}(B) = \mathcal{P}(A)\}$$

is called the **structural rank** of A.
If $\mathrm{rk}_S(A) < n$, then A is called **structural rank deficient**

Example 8.4

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \qquad C = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix},$$

$$\mathrm{rk}_S(A) = 2 \neq 1 = \mathrm{rank}(A), \qquad \mathrm{rk}_S(C) = 1 = \mathrm{rank}(C).$$

Remark 1

The structural rank of A is:

- a property related to the pattern $\mathcal{P}(A)$,
- much cheaper to compute than the (numerical) rank,
- available via **`sprank()`** in MATLAB®,
- an upper bound to the rank of A.

Preconditioning

Ideas

Preconditioning

Ideas

We have already seen, that the solution of a linear system $Ax = b$ depends strongly on the condition number of the matrix A:

$$\frac{\|\Delta x\|}{\|x\|} \leqslant \kappa(A) \frac{\|\Delta b\|}{\|b\|}.$$

Furthermore, the convergence speed of almost all iterative solvers depend strongly on the condition number $\kappa(A)$.

Preconditioning

Ideas

Remember the following Lemma:

Lemma 8.5

Let $P \in \mathbb{C}^{n\times n}$ be invertible and $A \in \mathbb{C}^{n\times n}$, then the linear systems of equations $Ax = y$ and $PAx = Py$ for $x, y \in \mathbb{C}^n$ are equivalent.

Preconditioning

Ideas

Remember the following Lemma:

Lemma 8.5

Let $P \in \mathbb{C}^{n\times n}$ be invertible and $A \in \mathbb{C}^{n\times n}$, then the linear systems of equations $Ax = y$ and $PAx = Py$ for $x, y \in \mathbb{C}^n$ are equivalent.

Goal: We need to find a $P \in \mathbb{R}^{n\times n}$ such that

$$\kappa(PA) < \kappa(A)$$

Preconditioning

Ideas

Remember the following Lemma:

Lemma 8.5

Let $P \in \mathbb{C}^{n\times n}$ be invertible and $A \in \mathbb{C}^{n\times n}$, then the linear systems of equations $Ax = y$ and $PAx = Py$ for $x, y \in \mathbb{C}^n$ are equivalent.

Goal: We need to find a $P \in \mathbb{R}^{n\times n}$ such that

$$\kappa(PA) < \kappa(A)$$

The perfect candidate for such a matrix P is obviously A^{-1}, since then $PA = I$ and $\kappa(PA) = 1$. However, A^{-1} is not accessible and especially has even worse "fill in" restrictions than the factorizations. Good approximations to A^{-1} are thus required that are:

- cheap to generate,
- easily and efficiently applicable,
- able to get stored with similar memory requirement as A.

Preconditioning

Ideas

Remark 2

Using a matrix P as

$$PAx = Pb$$

is called **left preconditioning**. Other versions like right, or two-sided preconditioning also exist. The ideas are very similar there, therefore we restrict the presentation to the most simple case.

Preconditioning

Ideas

Remark 2

Using a matrix P as

$$PAx = Pb$$

is called **left preconditioning**. Other versions like right, or two-sided preconditioning also exist. The ideas are very similar there, therefore we restrict the presentation to the most simple case.

Remark 3

P does not need to be a matrix, e.g., sometimes other (iterative) solvers are used.

Diagonal Preconditioning

Preconditioning

Diagonal Preconditioning

$$P^{-1} = \text{diag}(A)$$

- also called Jacobi preconditioning
- very simple and cheap
- might improve certain problems, e.g., diagonal dominant systems
- generally not sufficient
- more sophisticated variants use diagonal $k \times k$ ($k > 1$) blocks or multiple diagonals (e.g., tridiagonal preconditioning)

Splitting Methods

Preconditioning

Splitting Methods

Recall the Iterative Refinement. Set $A = B + (A - B)$, then

$$Ax = b \Leftrightarrow Bx = b + (B - A)x.$$

This motivates to define:

$$x_{i+1} = B^{-1}b + \underbrace{B^{-1}(B - A)}_{M} x_i.$$

If we can ensure $\rho(M) < 1$ then by a fixed point argument we can guarantee convergence.

Example 8.6

Two common examples of splitting methods are:

- B = diagonal of A $\rightsquigarrow$ Jacobi method
- B = lower triangle of A $\rightsquigarrow$ Gauß Seidel method

Incomplete Factorizations

Preconditioning

Incomplete Factorizations

Computation of $LU = A$ is often infeasible due to fill-in.
Basic idea: Only allow entries in L, U corresponding to $\mathcal{P}(A)$. This leads to the ILU(0) often written simply as ILU.

- usually only provides poor approximation
- variants allow:
  - "levels (k) of fill" (ILU(k))
  - fill-in that exceeds a drop tolerance ε (ILU(ε))
  - adding dropped fill to the diagonal (MIC)

Sparse Approximate Inverses (SPAI)

Preconditioning

Sparse Approximate Inverses (SPAI)

The basic idea of the sparse approximate inverse (SPAI) is to find the matrix $M \in \mathbb{R}^{n \times n}$ that best approximates A^{-1} among all matrices with $\mathcal{P}(M) = \mathcal{P}(A)$, in the sense

$$\min_M \|AM - I\|_F^2 = \min_M \underbrace{\sum_{j=1}^{n} \|Am_j - e_j\|_F^2}_{n \text{ independent least squares problems}} .$$

The SPAI preconditioner is especially attractive in parallel computing due to the independent column-wise computation.

In order to improve the approximation quality, similar pattern-extension considerations as for the incomplete factorizations can be used.

In any case, only matrix vector products are required for the application of the preconditioner, since PAx would be evaluated as $P(Ax)$, i.e., two subsequent matrix vector products.

Krylov Subspaces and Projection Methods

Krylov Subspaces and Projection Methods

Definition 8.7

$A \in \mathbb{C}^{n\times n}$ regular, $b \in \mathbb{C}^n$. A **projection method** for $Ax = b$ is a procedure for approximation of x by $x_m \in x_0 + \mathcal{K}_m$, which satisfies

$$(b - Ax_m) \perp \mathcal{L}_m. \tag{1}$$

Here, $x_0 \in \mathbb{C}^n$ is an arbitrary initial vector and $\mathcal{K}_m$, $\mathcal{L}_m$ are m-dimensional subspaces of $\mathbb{C}^n$. Condition (1) represents orthogonality in the Euclidean sense.
In case $\mathcal{K}_m = \mathcal{L}_m$, (1) is called **Galerkin-condition** and one has an **orthogonal projection method**. In case $\mathcal{K}_m \neq \mathcal{L}_m$, (1) is called **Petrov-Galerkin-condition** and one has an **oblique projection method**.

Krylov Subspaces and Projection Methods

Definition 8.8

$A \in \mathbb{C}^{n \times n}$ regular, $y \in \mathbb{C}^n$.

1. $\mathcal{K}_m(A, y) = \text{span}\{y, Ay, A^2y, \ldots, A^{m-1}y\}$ is called the m-th Krylov subspace of A for a seed vector y.
2. A projection method with $\mathcal{K}_m = \mathcal{K}_m(A, y)$ is called **Krylov subspace (projection) method**.

Krylov Subspaces and Projection Methods

Definition 8.8

$A \in \mathbb{C}^{n\times n}$ regular, $y \in \mathbb{C}^n$.

1. $\mathcal{K}_m(A, y) = \text{span}\{y, Ay, A^2y, \ldots, A^{m-1}y\}$ is called the m-th Krylov subspace of A for a seed vector y.
2. A projection method with $\mathcal{K}_m = \mathcal{K}_m(A, y)$ is called **Krylov subspace (projection) method**.

Definition 8.9 (minimal polynomial of A)

Let $p_\nu(\lambda) = \sum_{j=0}^{\nu} a_j \lambda^j$. The polinomial p_ν is called **minimal polynomial of A** if $\nu \in \mathbb{N}$ is the smallest degree such that $p_\nu(A) = 0$.

Krylov Subspaces and Projection Methods

In exact arithmetic we get the exact solution with $m = \nu$, since

$$\sum_{j=0}^{\nu} a_j A^j = 0 \Leftrightarrow A \sum_{j=1}^{\nu} a_j A^{j-1} = -a_0 I.$$

Thus

$$A^{-1} = -\frac{1}{a_0} \sum_{j=1}^{\nu} a_j A^{j-1},$$

which, in turn, means

$$x = A^{-1} b = -\frac{1}{a_0} \sum_{j=1}^{\nu} a_j A^{j-1} b \in \mathcal{K}_\nu(A, b).$$

Krylov Subspaces and Projection Methods

Now we let $x_0 \in \mathbb{C}^n$ be the initial vector and $r_0 := b - Ax_0$ the corresponding initial residual. Further, let $\mathcal{K}_m = \mathcal{K}_m(A, r_0)$, $\mathcal{L}_m$ be subspaces, and the columns of $V_m, W_m \in \mathbb{C}^{n \times m}$ bases of $\mathcal{K}_m$ and $\mathcal{L}_m$, respectively.

Krylov Subspaces and Projection Methods

Now we let $x_0 \in \mathbb{C}^n$ be the initial vector and $r_0 := b - Ax_0$ the corresponding initial residual. Further, let $\mathcal{K}_m = \mathcal{K}_m(A, r_0)$, $\mathcal{L}_m$ be subspaces, and the columns of $V_m, W_m \in \mathbb{C}^{n \times m}$ bases of $\mathcal{K}_m$ and $\mathcal{L}_m$, respectively.

Then, for $x_m \in x_0 + \mathcal{K}_m$ there exists a $\sigma_m \in \mathbb{C}^m$ with $x_m = x_0 + V_m \sigma_m$ and (1) holds iff

$$
\begin{aligned}
&\Leftrightarrow 0 = W_m^{\mathrm{H}}(b - A(x_0 + V_m\sigma_m)) \\
&\Leftrightarrow 0 = W_m^{\mathrm{H}}(b - Ax_0) - W_m^{\mathrm{H}} A V_m \sigma_m \\
&\Leftrightarrow W_m^{\mathrm{H}} A V_m \sigma_m = W_m^{\mathrm{H}} r_0 \\
&\Leftrightarrow \sigma_m = (W_m^{\mathrm{H}} A V_m)^{-1} W_m^{\mathrm{H}} r_0.
\end{aligned}
$$

Krylov Subspaces and Projection Methods

Thus $x_m = x_0 + V_m (W_m^{\mathsf{H}} A V_m)^{-1} W_m^{\mathsf{H}} r_0$

$$\begin{aligned} r_m &= b - A x_m \\ &= b - A(x_0 + V_m (W_m^{\mathsf{H}} A V_m)^{-1} W_m^{\mathsf{H}} r_0) \\ &= r_0 - A V_m (W_m^{\mathsf{H}} A V_m)^{-1} W_m^{\mathsf{H}} r_0 \end{aligned}$$

The projection P_m to the m-th subspace is then given as $P_m = I - Q_m$, where $Q_m = A V_m (W_m^{\mathsf{H}} A V_m)^{-1} W_m^{\mathsf{H}}$. The above derivation proves the following simple lemma.

Lemma 8.10

If $W_m^{\mathrm{H}} A V_m$ is invertible, then (1) has a unique solution given as

$$x_m = x_0 + V_m (W_m^{\mathrm{H}} A V_m)^{-1} W_m^{\mathrm{H}} r_0$$

with corresponding residual

$$r_m = r_0 - A V_m (W_m^{\mathrm{H}} A V_m)^{-1} W_m^{\mathrm{H}} r_0$$

Lemma 8.10

If $W_m^{\mathrm{H}} A V_m$ is invertible, then (1) has a unique solution given as

$$x_m = x_0 + V_m (W_m^{\mathrm{H}} A V_m)^{-1} W_m^{\mathrm{H}} r_0$$

with corresponding residual

$$r_m = r_0 - A V_m (W_m^{\mathrm{H}} A V_m)^{-1} W_m^{\mathrm{H}} r_0$$

The invertibility assumption is sometimes easily guaranteed. For example if A is symmetric positive definite (s.p.d.) with $\mathcal{K}_m = \mathcal{K}_m(A, r_0) = \mathcal{L}_m$

$$\Rightarrow W_m = V_m \text{ and } \dim \mathcal{K}_m = m$$
$$\Rightarrow W_m^{\mathrm{H}} A V_m = V_m^{\mathrm{H}} A V_m \text{ s.p.d.}$$

Analogously, for A invertible and $\mathcal{L}_m = A\mathcal{K}_m \Rightarrow W_m = AV_m$ with $\dim \mathcal{K}_m = m = \dim \mathcal{L}_m$, we immediately see that $W_m^{\mathrm{H}} A V_m = V_m^{\mathrm{H}} A^{\mathrm{H}} A V_m$ is s.p.d..

Conjugate Gradients

How to choose $\mathcal{K}$ and $\mathcal{L}$ such that $||x_i - x||$ gets small very fast?

Definition 8.11

The **gradient** $\nabla f : C^1(\mathbb{R}^n) \mapsto (C^0(\mathbb{R}^n))^n$ of a function $f : \mathbb{R}^n \mapsto R$ is given by

$$\nabla f = \left(\frac{\partial f}{\partial x_i} \right)_{i=1}^{n}.$$

Definition 8.11

The **gradient** $\nabla f : C^1(\mathbb{R}^n) \mapsto (C^0(\mathbb{R}^n))^n$ of a function $f : \mathbb{R}^n \mapsto R$ is given by

$$\nabla f = \left(\frac{\partial f}{\partial x_i} \right)_{i=1}^{n}.$$

We consider the quadratic function

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

with $A \in \mathbb{R}^{n \times n}$ and the minimization problem

$$\min_{x \in \mathbb{R}^n} f(x).$$

Krylov Subspaces and Projection Methods

Conjugate Gradients

Solving $\min f(x)$ requires:

$$0 = \nabla f(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b.$$

If A is symmetric and positive definite (spd), this yields

$$\nabla f(x) = Ax - b$$

Krylov Subspaces and Projection Methods

Conjugate Gradients

Solving $\min f(x)$ requires:

$$0 = \nabla f(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b.$$

If A is symmetric and positive definite (spd), this yields

$$\nabla f(x) = Ax - b$$

Lemma 8.12

Let $A \in \mathbb{R}^{n \times n}$ be symmetric positive definite, than finding a the minimum of

$$f(x) = \frac{1}{2}x^T Ax - b^T x + c$$

is equivalent to solving

$$Ax = b$$

.

Proof.

$\Leftarrow$ Let $f(x)$ be minimal, this yields $\nabla f(x) = 0$ and thus $Ax = b$.

$\Rightarrow$ Let $x \in \mathbb{R}^n$ fulfilling $Ax = b$ and $e \in \mathbb{R}^n$, $e \neq 0$, be a perturbation. This gives

$$\begin{aligned} f(x+e) &= \frac{1}{2}(x+e)^T A(x+e) - b^T(x+e) + c \\ &= \frac{1}{2}x^T Ax + e^T Ax + \frac{1}{2}e^T Ae - b^T x - b^T e + c \quad Ax = b \\ &= \frac{1}{2}x^T Ax - b^T x + c + e^T b - b^T e + \frac{1}{2}e^T Ae \\ &= f(x) + \frac{1}{2}e^T Ae \end{aligned}$$

Since A is positive definite, $e^T Ae > 0 \ \forall e \neq 0$, and thus x solving $Ax = b$ is the minimum of $f(x)$.

□

Krylov Subspaces and Projection Methods

Conjugate Gradients

Goal

Now, the goal is to construct a sequence with $Ax^* = b$

$$\{x_i\} \in \mathbb{R}^n \text{ such that } x_i \to x^*,\ i \to \infty$$

by only using matrix-vector products or vector-valued operations.

Goal

Now, the goal is to construct a sequence with $Ax^* = b$

$$\{x_i\} \in \mathbb{R}^n \text{ such that } x_i \to x^*,\ i \to \infty$$

by only using matrix-vector products or vector-valued operations.

Definition 8.13

Let $A \in \mathbb{R}^{n \times n}$, $b \in \mathbb{R}^n$ and $\{x_i\} \in \mathbb{R}^n$ be a constructed sequence. Furthermore, let x^* be the solution of $Ax = b$:

- The **error in the i-th step** is given by $e_i = x_i - x^*$.
- The **residual in the i-th step** is given by $r_i = b - Ax_i$.

Krylov Subspaces and Projection Methods

Conjugate Gradients

Obviously, we have

- $r_i = -Ae_i = -(Ax_i - Ax^*) = (-Ax_i + b)$,
- $r_i = -\nabla f(x_i)$.

From this point of view, the residual can be seen as direction of **steepest descent**. This leads to the idea

$$x_{i+1} = x_i + \alpha r_i,\ \alpha_i \in \mathbb{R}$$

to construct a sequence.

Obviously, we have

- $r_i = -Ae_i = -(Ax_i - Ax^*) = (-Ax_i + b)$,
- $r_i = -\nabla f(x_i)$.

From this point of view, the residual can be seen as direction of **steepest descent**. This leads to the idea

$$x_{i+1} = x_i + \alpha r_i,\ \alpha_i \in \mathbb{R}$$

to construct a sequence.

Theorem 8.14

Let f as above, then

$$\alpha = \frac{r_i^T r_i}{r_i^T A r_i}$$

minimizes f along $x_i + \alpha r_i$, i.e., $\frac{\mathrm{d}}{\mathrm{d}\alpha} f(x_i + \alpha r_i) = 0$

Proof.

We regard

$$\frac{\mathrm{d}}{\mathrm{d}\alpha} f(x_i + \alpha r_i) = \nabla f(x_i + \alpha r_i)^T \frac{\mathrm{d}}{\mathrm{d}\alpha}(x_i + \alpha r_i) = \nabla f(x_i + \alpha r_i)^T r_i$$

This means we have to choose α such that $\nabla f(x_{i+1}) \perp r_i$. This leads

$$\begin{aligned}
-\nabla f(x_i + \alpha r_i)^T r_i = -\nabla f(x_{i+1})^T r_i = r_{i+1}^T r_i &= 0 \\
(b - Ax_{i+1})^T r_i &= 0 \\
(b - A(x_i + \alpha r_i))^T r_i &= 0 \\
(b - Ax_i)^T r_i - \alpha r_i^T A r_i &= 0 \quad \text{using } b - Ax_i = r_i \\
r_i^T r_i &= \alpha r_i^T A r_i. \\
\alpha &= \frac{r_i^T r_i}{r_i^T A r_i}
\end{aligned}$$

□

Krylov Subspaces and Projection Methods

Conjugate Gradients

Algorithm 8.1: Steepest Decent / Gradient Method

Input: $A \in \mathbb{R}^{n \times n}$ spd, $b \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$, $k_{max} \in \mathbb{N}_+$

Output: x_k minimizing $||Ax_k - b||$ w.r.t. τ and k_{max}

$r_0 = b - Ax_0$;

$k \leftarrow 0$;

while $||r_k||_2 > \tau ||r_0||_2$ **and** $k \leqslant k_{max}$ **do**

  $r_k = b - Ax_k$;

  $\alpha = \frac{r_k^T r_k}{r_k^T A r_k}$;

  $x_{k+1} = x_k + \alpha r_k$;

  $k \leftarrow k + 1$;

Krylov Subspaces and Projection Methods

Conjugate Gradients

Algorithm 8.1: Steepest Decent / Gradient Method

Input: $A \in \mathbb{R}^{n \times n}$ spd, $b \in \mathbb{R}^n$, $x_0 \in \mathbb{R}^n$, $\tau \in \mathbb{R}$, $k_{max} \in \mathbb{N}_+$

Output: x_k minimizing $||Ax_k - b||$ w.r.t. τ and k_{max}

$r_0 = b - Ax_0$;

$k \leftarrow 0$;

while $||r_k||_2 > \tau ||r_0||_2$ **and** $k \leqslant k_{max}$ **do**

  $r_k = b - Ax_k$;

  $\alpha = \frac{r_k^T r_k}{r_k^T A r_k}$;

  $x_{k+1} = x_k + \alpha r_k$;

  $k \leftarrow k + 1$;

- only 2 matrix-vector products per step,
- goes in "zick-zack" to the solution, directions can be used twice.

Krylov Subspaces and Projection Methods

Conjugate Gradients

Since we want to use each search direction only once, we change the iteration step to

$$x_{i+1} = x_i + \alpha_i d_i,$$

where d_i are the search directions with $d_i^T d_j = 0$, $\forall i \neq j$.

Krylov Subspaces and Projection Methods

Conjugate Gradients

Since we want to use each search direction only once, we change the iteration step to

$$x_{i+1} = x_i + \alpha_i d_i,$$

where d_i are the search directions with $d_i^T d_j = 0$, $\forall i \neq j$.

Now, we obtain

$$\begin{aligned} x_{i+1} - x^* &= x_i + \alpha_i d_i - x^* \\ e_{i+1} &= e_i + \alpha_i d_i \end{aligned}$$

for the error.

Krylov Subspaces and Projection Methods

Conjugate Gradients

Since we want to use each search direction only once, we change the iteration step to

$$x_{i+1} = x_i + \alpha_i d_i,$$

where d_i are the search directions with $d_i^T d_j = 0$, $\forall i \neq j$.

Now, we obtain

$$\begin{aligned} x_{i+1} - x^* &= x_i + \alpha_i d_i - x^* \\ e_{i+1} &= e_i + \alpha_i d_i \end{aligned}$$

for the error.

Again, we have to select α_i such that

$$\alpha_i = \text{argmin}_\alpha \, f(x_i + \alpha d_i),$$

which is equivalent to $d_i^T e_{i+1} = 0$.

Lemma 8.15

Let $d_0, \ldots, d_{n-1}$ be pairwise orthogonal, i.e. $d_i^T d_j = 0$, $\forall i \neq j$. Selecting

$$\alpha_i = -\frac{d_i^T e_i}{d_i^T d_i}$$

yields $d_i^T e_{i+1} = 0$.

Lemma 8.15

Let $d_0, \ldots, d_{n-1}$ be pairwise orthogonal, i.e. $d_i^T d_j = 0$, $\forall i \neq j$. Selecting

$$\alpha_i = -\frac{d_i^T e_i}{d_i^T d_i}$$

yields $d_i^T e_{i+1} = 0$.

Proof.

$$\alpha_i = -\frac{d_i^T e_i}{d_i^T d_i}$$

$$\alpha_i d_i^T d_i = -d_i^T e_i$$

$$d_i^T (e_i + \alpha_i d_i) = 0 = d_i^T e_{i+1}$$

□

Problem

In order to determine $\alpha_i = -\frac{d_i^T e_i}{d_i^T d_i}$, we need to know e_i. But if e_i is known, the problem is solved.

Problem

In order to determine $\alpha_i = -\frac{d_i^T e_i}{d_i^T d_i}$, we need to know e_i. But if e_i is known, the problem is solved.

Definition 8.16

Two vectors $d_i \in \mathbb{R}^n$ and $d_j \in \mathbb{R}^n$, with $A \in \mathbb{R}^{n \times n}$ spd are called A-**orthogonal**, if

$$d_j^T A d_i = 0.$$

Lemma 8.17

Let $d_0, \ldots, d_{n-1}$ be pairwise A-orthogonal, i.e. $d_i^T A d_j = 0$, $\forall i \neq j$, then

$$\alpha_i = \frac{d_i^T r_i}{d_i^T A d_i}$$

minimizes $f(x_i + \alpha_i d_i)$ along d_i, i.e. $d_i^T A e_{i+1} = 0$.

Proof.

Let $f(x_i + \alpha_i d_i)$ be minimal:

$$\frac{\mathrm{d}}{\mathrm{d}\alpha_i} f(x_i + \alpha_i d_i) = 0$$

$$\nabla f(x_{i+1})^T \frac{\mathrm{d}}{\mathrm{d}\alpha_i}(x_i + \alpha_i d_i) = -r_{i+1}^T d_i = 0$$

$$d_i^T A e_{i+1} = d_i^T A(e_i + \alpha_i d_i) = 0$$

$$d_i^T A e_i + \alpha_i d_i^T A d_i = 0$$

$$\alpha_i = -\frac{d_i^T A e_i}{d_i^T A d_i} = \frac{d_i^T r_i}{d_i^T A d_i}$$

□

Proof.

Let $f(x_i + \alpha_i d_i)$ be minimal:

$$\frac{\mathrm{d}}{\mathrm{d}\alpha_i} f(x_i + \alpha_i d_i) = 0$$

$$\nabla f(x_{i+1})^T \frac{\mathrm{d}}{\mathrm{d}\alpha_i}(x_i + \alpha_i d_i) = -r_{i+1}^T d_i = 0$$

$$d_i^T A e_{i+1} = d_i^T A(e_i + \alpha_i d_i) = 0$$

$$d_i^T A e_i + \alpha_i d_i^T A d_i = 0$$

$$\alpha_i = -\frac{d_i^T A e_i}{d_i^T A d_i} = \frac{d_i^T r_i}{d_i^T A d_i}$$

□

→ Now, α_i can be computed from the existing information.

Theorem 8.18

Let $A \in \mathbb{R}^{n \times n}$ spd, and $d_0, \ldots, d_{n-1} \in \mathbb{R}^n$ pairwise A-orthogonal. Then

$$x_{i+1} = x_i + \alpha_i d_i$$

with

$$\alpha_i = \frac{d_i^T r_i}{d_i^T A d_i}$$

converges to the solution x^* of $Ax = b$ after n steps.

Theorem 8.18

Let $A \in \mathbb{R}^{n\times n}$ spd, and $d_0, \ldots, d_{n-1} \in \mathbb{R}^n$ pairwise A-orthogonal. Then

$$x_{i+1} = x_i + \alpha_i d_i$$

with

$$\alpha_i = \frac{d_i^T r_i}{d_i^T A d_i}$$

converges to the solution x^* of $Ax = b$ after n steps.

From $x_{i+1} = x_i + \alpha_i d_i$ follows:

$$\begin{aligned} r_{i+1} &= -A(e_i + \alpha_i d_i) \\ &= r_i - \alpha_i A d_i. \end{aligned}$$

How to obtain the A-orthogonal vectors d_i?

How to obtain the A-orthogonal vectors d_i?

We choose $u_0, \ldots, u_{n-1} \in \mathbb{R}^n$ linear independent vectors. Starting with $d_0 = u_0$, we A-orthogonalize u_i, $i > 0$, with respect to $d_0, \ldots, d_{i-1}$ to obtain d_i:

$$d_i = u_i + \sum_{j=0}^{i-1} \beta_{ij} d_j$$

How to obtain the A-orthogonal vectors d_i?

We choose $u_0, \ldots, u_{n-1} \in \mathbb{R}^n$ linear independent vectors. Starting with $d_0 = u_0$, we A-orthogonalize u_i, $i > 0$, with respect to $d_0, \ldots, d_{i-1}$ to obtain d_i:

$$d_i = u_i + \sum_{j=0}^{i-1} \beta_{ij} d_j$$

Lemma 8.19

Let $d_0, \ldots, d_{i-1} \in \mathbb{R}^n$ be A-orthogonal vectors and $u_i \in \mathbb{R}^n$ linear independent from $d_0, \ldots, d_{i-1}$. Then β_{ij}, $j < i$, is given by

$$\beta_{ij} = -\frac{u_i^T A d_j}{d_j^T A d_j}$$

and d_i is A-orthogonal to $d_0, \ldots, d_{i-1}$.

Intermediate Summary

1. We have an algorithmic idea, which solves $Ax = b$ in n steps without LU decomposition etc.
2. Search directions $d_0, \ldots, d_{i-1}$ must be stored to compute d_i.
   $\rightarrow$ easily runs out of memory for large n.
3. The computational complexity increases to $\mathcal{O}(n^3)$ due to the orthogonalization procedure.
   $\rightarrow$ same runtime class as the LU decomposition.
4. The initial directions $u_0, \ldots, u_{n-1}$ are some how arbitrary.
   $\rightarrow$ should correspond to $Ax = b$.

Intermediate Summary

1. We have an algorithmic idea, which solves $Ax = b$ in n steps without LU decomposition etc.
2. Search directions $d_0, \ldots, d_{i-1}$ must be stored to compute d_i.
   $\rightarrow$ easily runs out of memory for large n.
3. The computational complexity increases to $\mathcal{O}(n^3)$ due to the orthogonalization procedure.
   $\rightarrow$ same runtime class as the LU decomposition.
4. The initial directions $u_0, \ldots, u_{n-1}$ are some how arbitrary.
   $\rightarrow$ should correspond to $Ax = b$.

$\rightarrow$ We need a strategy to compute obtain u_i (and d_i) without requiring to many "old" search directions d_j, $j < i$.

Since the the residuum r_i is the direction of the steepest decent, we set

$$u_i = r_i.$$

Theorem 8.20

Let $d_0, \ldots, d_{i-1} \in \mathbb{R}^n$ be A-orthogonal search directions, then it holds

(a) *The residual r_j is orthogonal to d_i, $\forall i < j$.*

(b) *The residuals r_i and r_j are orthogonal for all $i < j$.*

(c) $d_i^T r_i = r_i^T r_i$, $\forall i \in \mathbb{N}$.

Proof.

(a)

$$e_j = \sum_{k=j}^{n-1} \delta_k d_k$$

$$-d_i^T A e_j = -\sum_{k=j}^{n-1} \delta_k d_i^T A d_k$$

$$d_i^T r_j = 0, \quad i < j$$

(b)

$$d_i = r_i + \sum_{k=0}^{i-1} \beta_{ik} d_k$$

$$d_i^T r_j = r_i^T r_j + \sum_{k=0}^{i-1} \beta_{ik} d_k^T r_j, \quad i < j$$

$$0 = r_i^T r_j$$

(c) (b) with $i = j$.

□

Krylov Subspaces and Projection Methods

Conjugate Gradients

→ Now, we know that r_{i+1} is orthogonal to all previous search directions $d_0, \ldots, d_i$ by construction.

→ Now, we know that r_{i+1} is orthogonal to all previous search directions $d_0, \ldots, d_i$ by construction.

Theorem 8.21

Let $d_0, \ldots, d_{i-1} \in \mathbb{R}^n$ be A-orthogonal search directions and $r_i \in \mathbb{R}^n$ linear independent from $d_0, \ldots, d_{i-1}$, then the othogonalization coefficients β_{ij} in

$$d_i = r_i + \sum_{j=0}^{i-1} \beta_{ij} d_j$$

are given by

$$\beta_{ij} = \begin{cases} \frac{1}{\alpha_{i-1}} \frac{r_i^T r_i}{d_{i-1}^T A d_{i-1}}, & j = i-1 \\ 0 & j < i-1 \end{cases}.$$

Since $\beta_{ij} = 0$ for $j < i-1$ we do not need d_j, $j < i-1$ and longer. Furthermore, we set $\beta_i = \beta_{i,i-1}$. Now we obtain:

$$\begin{aligned}
\beta_i &= \frac{1}{\alpha_{i-1}} \frac{r_i^T r_i}{d_{i-1}^T A d_{i-1}}, \qquad \text{using } \alpha_{i-1} = \frac{d_{i-1}^T r_{i-1}}{d_{i-1}^T A d_{i-1}} \\
&= \frac{d_{i-1}^T A d_{i-1}}{d_{i-1}^T r_{i-1}} \frac{r_i^T r_i}{d_{i-1}^T A d_{i-1}} \\
&= \frac{r_i^T r_i}{d_{i-1}^T r_{i-1}} \\
&= \frac{r_i^T r_i}{r_{i-1}^T r_{i-1}}
\end{aligned}$$

Furthermore, we have

$$\alpha_i = \frac{r_i^T r_i}{d_i^T A d_i}.$$

Algorithm 8.2: Conjugate Gradient Method

Input: $A \in \mathbb{R}^{n \times n}, b \in \mathbb{R}^n, x_0 \in \mathbb{R}^n, \tau \in \mathbb{R}$
Output: $x = A^{-1}b$

$d_0 = r_0 = b - Ax_0, \delta_0 = \|r_0\|_2^2 = r_0^T r_0;$
for $i = 0, \ldots, n-1$ **do**
  if $\delta_i > \tau \delta_0$ **then**
    $v_i = Ad_i;$
    $\alpha_i = \frac{\delta_i}{d_i^T v_i};$
    $x_{i+1} = x_i + \alpha_i d_i;$
    $r_{i+1} = r_i - \alpha_i v_i;$
    $\delta_{i+1} = \|r_{i+1}\|_2^2 = r_{i+1}^T r_{i+1};$
    $\beta_{i+1} = \frac{\delta_{i+1}}{\delta_i};$
    $d_{i+1} = r_{i+1} + \beta_{i+1} d_i;$
  else
    STOP;

Algorithm 8.3: Preconditioned Conjugate Gradient Method

Input: $A \in \mathbb{R}^{n\times n}, b \in \mathbb{R}^n, x_0 \in \mathbb{R}^n, A^{-1} \approx P \in \mathbb{R}^{n\times n}, \tau \in \mathbb{R}$
Output: $x = A^{-1}b$

$r_0 = b - Ax_0,\ d_0 = z_0 = Pr_0,\ \delta_0 = d_0^T r_0;$
for $i = 0 : n-1$ **do**
  if $\delta_i > \tau\delta_0$ **then**
    $v_i = Ad_i;$
    $\alpha_i = \frac{\delta_i}{d_i^T v_i};$
    $x_{i+1} = x_i + \alpha_i d_i;$
    $r_{i+1} = r_i - \alpha_i v_i;$
    $z_{i+1} = Pr_{i+1};$
    $\delta_{i+1} = z_{i+1}^T r_{i+1};$
    $\beta_{i+1} = \frac{\delta_{i+1}}{\delta_i};$
    $d_{i+1} = z_{i+1} + \beta_{i+1} d_i;$
  else
    STOP;

Theorem 8.22

Let

$$e_m = x_m - A^{-1}b$$

denote the error in the m-th step of the CG algorithm. Then it holds

$$\|e_m\|_A \leqslant 2\left(\frac{\kappa_2(A)-1}{\kappa_2(A)+1}\right)^m \|e_0\|_A .$$

Theorem 8.22

Let

$$e_m = x_m - A^{-1}b$$

denote the error in the m-th step of the CG algorithm. Then it holds

$$\|e_m\|_A \leqslant 2\left(\frac{\kappa_2(A)-1}{\kappa_2(A)+1}\right)^m \|e_0\|_A .$$

Remark 4

The derivation of the CG algorithm results in $\mathcal{K}_m = \mathcal{L}_m = \mathcal{K}_m(A, r_0)$ and $V_m = W_m$. The computation of

$$x_m = x_0 + V_m(V_m^{\mathrm{H}} A V_m)^{-1} V_m^{\mathrm{H}} r_0$$

is done successively by the algorithm and V_m does not need to be set up.

Example 8.23

We take

$$A = \begin{bmatrix} 2 & -1 & \cdots & 0 \\ -1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ 0 & \cdots & -1 & 2 \end{bmatrix}$$

and $b = A \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}^T$ with varying dimension n. One can show that A is symmetric and positive definite. Running **Steepest Decent** and **CG** in double precision with $\tau = \sqrt{u}$ gives:

		Steepest Decent		Conjugate Gradient	
n	$\kappa_1(A)$	$\|\|b - Ax\|\|_2 \|\|b\|\|_2^{-1}$	Steps	$\|\|b - Ax\|\|_2 \|\|b\|\|_2^{-1}$	Steps
100	$5.1 \cdot 10^3$	$1.50 \cdot 10^{-8}$	27 441	$4.28 \cdot 10^{-14}$	50
1 000	$5.01 \cdot 10^5$	$1.49 \cdot 10^{-8}$	1 998 614	$2.10 \cdot 10^{-12}$	500
1 000 000	$5.01 \cdot 10^{11}$	–	–	$1.43 \cdot 10^{-11}$	500 001

Krylov Subspaces and Projection Methods

Conjugate Gradients

What if A is not symmetric and/or not positive definite?

Other Krylov methods exists:

- GMRES – Generalized Minimal Residual
- BiCG – Bi-Conjugate Gradient
- BiCGStab – Bi-Conjugate Gradient Stabilized
- CGS – Conjugate Gradient Squared
- QMR – Quasi Minimal Residual
- TFQMR – Transpose Free Quasi Minimal Residual
- ...
- ... or tweak the CG method.

Conjugate Gradient Normal Equation Residual/Conjugate Gradient Normal Equation Error

Conjugate Gradient Normal Equation Residual

Instead of

$$Ax = b$$

we solve

$$A^T Ax = A^T b,$$

where $A^T A$ is spd for all regular matrices. Since the residual "true" residual $r = b - Ax$ is transformed into

$$\tilde{r} = A^T b - A^T Ax = A^T r,$$

this method is called **Conjugate Gradient Normal Equation Residual (CGNR)**.

Conjugate Gradient Normal Equation Error

By using AA^T instead of A, we obtain

$$AA^T y = b$$

and need to recover the solution x of $Ax = b$ by

$$x = A^T y.$$

Integrating both in the CG algorithm gives the **Conjugate Gradient Normal Equation Error (CGNE)** method, which minimizes the error in each step.

Conjugate Gradient Normal Equation Error

By using AA^T instead of A, we obtain

$$AA^T y = b$$

and need to recover the solution x of $Ax = b$ by

$$x = A^T y.$$

Integrating both in the CG algorithm gives the **Conjugate Gradient Normal Equation Error (CGNE)** method, which minimizes the error in each step.

Remark 5

Both methods, CGNR and CGNE, use some kind of a "squared" matrix in each step. Thus, the condition numbers fulfill

$$\kappa_2(A^T A) \approx (\kappa_2(A))^2 \text{ and } \kappa_2(AA^T) \approx (\kappa_2(A))^2 .$$

Direct Solvers for Sparse Symmetric Systems

Preliminaries

Direct Solvers for Sparse Symmetric Systems

Preliminaries

In the following, to ease the presentations, we will follow the general assumptions that

- $A \in \mathbb{R}^{n \times n}$ is sparse and symmetric,
- and no pivoting is used.

Direct Solvers for Sparse Symmetric Systems

Preliminaries

In the following, to ease the presentations, we will follow the general assumptions that

- $A \in \mathbb{R}^{n \times n}$ is sparse and symmetric,
- and no pivoting is used.

Remark 6

For non-symmetric matrices the presented concepts have to be generalized from undirected to directed graphs.

Direct Solvers for Sparse Symmetric Systems

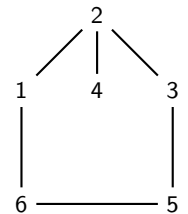
Preliminaries

Definition 8.24

Two graphs are easily related to the matrix $A \in \mathbb{R}^{n \times n}$.

1. $\mathcal{V} = \{1, \ldots, n\}$ is called the set of **vertices**, i.e., variable indices.
2. The set of **edges** $\mathcal{E} \subseteq V^2$ is the set of pairs $(i, j) \in \mathcal{E} \Leftrightarrow a_{ij} \neq 0$.
3. The **directed connectivity graph of** $A - \mathcal{G}_d(A) = (\mathcal{V}, \mathcal{E})$ associates a direction to an edge by the order of indices in the pair.
4. The **undirected connectivity graph of** $A - \mathcal{G}(A) = (\mathcal{V}, \mathcal{E})$ identifies the pairs (i, j) and (j, i), i.e. considers $(i, j) = (j, i)$, and thus neglects the direction.

$$A = \begin{bmatrix} 1 & * & & & & * \\ * & 2 & * & * & & \\ & * & 3 & & * & \\ & * & & 4 & & \\ & & * & & 5 & * \\ * & & & & * & 6 \end{bmatrix}$$



- $\mathcal{V} = \{1, 2, 3, 4, 5, 6\}$
- $\mathcal{E} = \{(1, 2), (1, 6), (2, 3), (2, 4), (3, 5), (5, 6)\}$

Remark 7

We collect some properties of the symmetric case treated in this chapter.

- A symmetric $\Rightarrow a_{ij} = a_{ji} \Rightarrow$ "$(i,j) \in \mathcal{E} \Leftrightarrow (j,i) \in \mathcal{E}$"
  $\Rightarrow$ its is sufficient to the treat the undirected graph
- If A s.p.d. then $\forall i \ a_{ii} > 0 \Rightarrow (i,i) \in \mathcal{E}$, i.e., the graph contains the trivial edges (usually not included in graphical representations of the graph)
- The number of nonzero elements in column i equals the number of neighbors of the vertex i in the graph $\mathcal{G}(A)$.
- Symmetric permutations, i.e., permutations of the matrix where both columns and rows are swapped simultaneously, are equivalent to renumbering the graph, i.e., application of a permutation to the elements of $\mathcal{V}$.
- $\mathcal{E} \hat{=} \mathcal{P}(A)$

The Elimination Graph Model for Symmetric Matrices

Direct Solvers for Sparse Symmetric Systems

The Elimination Graph Model for Symmetric Matrices

If A is spd, we observed that the LU decomposition could be symmetrized as well. By neglecting L to be unit diagonal, we can construct $A = LL^T$ as follows:

$$
\begin{aligned}
A = A_0 = H_0 &= \begin{bmatrix} d_1 & v_1^{\mathsf{T}} \\ v_1 & \tilde{H}_1 \end{bmatrix}, \quad \tilde{H}_1 \in \mathbb{R}^{n-1 \times n-1} \\
&= \underbrace{\begin{bmatrix} \sqrt{d_1} & 0 \\ \frac{1}{\sqrt{d_1}} v_1 & I_{n-1} \end{bmatrix}}_{L_1} \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & H_1 \end{bmatrix}}_{A_1} \underbrace{\begin{bmatrix} \sqrt{d_1} & \frac{1}{\sqrt{d_1}} v_1^{\mathsf{T}} \\ 0 & I_{n-1} \end{bmatrix}}_{L_1^{\mathsf{T}}}, \quad H_1 = \tilde{H}_1 - \frac{1}{d_1} v_1 v_1^T \\
A &= (L_1 L_2 L_3 \dots L_{n-1}) I_n (L_{n-1}^{\mathsf{T}} \dots L_3^{\mathsf{T}} L_2^{\mathsf{T}} L_1^{\mathsf{T}}) \\
&= (L_1 L_2 L_3 \dots L_{n-1}) I_n (L_1 L_2 L_3 \dots L_{n-1})^{\mathsf{T}} \\
&= LL^{\mathsf{T}},
\end{aligned}
$$

which is the **Cholesky Decomposition** of A.

Direct Solvers for Sparse Symmetric Systems

The Elimination Graph Model for Symmetric Matrices

In the case where A is symmetric but not positive definite, we have to introduce a **diagonal matrix** D such that $A = LDL^T$ holds. This gives

$$
\begin{aligned}
A = A_0 = H_0 &= \begin{bmatrix} d_1 & v_1^{\mathsf{T}} \\ v_1 & \tilde{H}_1 \end{bmatrix}, \quad \tilde{H}_1 \in \mathbb{R}^{n-1 \times n-1} \\
&= \underbrace{\begin{bmatrix} 1 & 0 \\ \frac{1}{d_1} v_1 & I_{n-1} \end{bmatrix}}_{L_1} \underbrace{\begin{bmatrix} d_1 & 0 \\ 0 & H_1 \end{bmatrix}}_{A_1} \underbrace{\begin{bmatrix} 1 & \frac{1}{d_1} v_1^{\mathsf{T}} \\ 0 & I_{n-1} \end{bmatrix}}_{L_1^{\mathsf{T}}}, \quad H_1 = \tilde{H}_1 - \frac{1}{d_1} v_1 v_1^T \\
A &= (L_1 L_2 L_3 \dots L_{n-1}) D (L_{n-1}^{\mathsf{T}} \dots L_3^{\mathsf{T}} L_2^{\mathsf{T}} L_1^{\mathsf{T}}) \\
&= (L_1 L_2 L_3 \dots L_{n-1}) D (L_1 L_2 L_3 \dots L_{n-1})^{\mathsf{T}} \\
&= LDL^{\mathsf{T}},
\end{aligned}
$$

This gives the LDL^{T} **decomposition** of A.

Direct Solvers for Sparse Symmetric Systems

The Elimination Graph Model for Symmetric Matrices

In both cases, $A = LL^T$ and $A = LDL^T$, the update $H_j = \tilde{H}_j - \frac{1}{d_j} v_j v_j^T$ influences the structure of the pattern $\mathcal{P}(H_j)$.

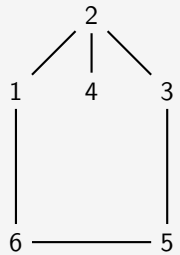
In general: If $\mathcal{P}(v_j v_j^T) \backslash (\mathcal{P}(v_j v_j^T) \cap \mathcal{P}(\tilde{H}_j)) \neq \varnothing$ then step j leads to fill-in in H_j.

Direct Solvers for Sparse Symmetric Systems

The Elimination Graph Model for Symmetric Matrices

Example 8.25 (Graph Representation of the Cholesky Decomposition)

$\mathcal{G}_0 =$



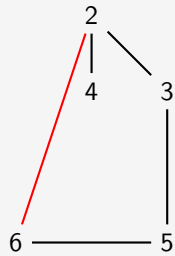
$$H_0 = \begin{bmatrix} 1 & * & & & & * \\ * & 2 & * & * & & \\ & * & 3 & & * & \\ & * & & 4 & & \\ & & * & & 5 & * \\ * & & & & * & 6 \end{bmatrix}$$

Direct Solvers for Sparse Symmetric Systems

The Elimination Graph Model for Symmetric Matrices

Example 8.25 (Graph Representation of the Cholesky Decomposition)

$\mathcal{G}_1 =$



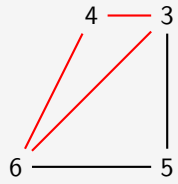
$$H_1 = \begin{bmatrix} 2 & * & * & & * \\ * & 3 & & * & \\ * & & 4 & & \\ & * & & 5 & * \\ * & & & * & 6 \end{bmatrix}$$

Direct Solvers for Sparse Symmetric Systems

The Elimination Graph Model for Symmetric Matrices

Example 8.25 (Graph Representation of the Cholesky Decomposition)

$\mathcal{G}_2 =$

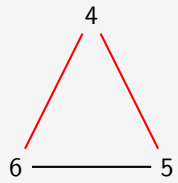


$$H_2 = \begin{bmatrix} 3 & * & * & * \\ * & 4 & & * \\ * & & 5 & * \\ * & * & * & 6 \end{bmatrix}$$

Direct Solvers for Sparse Symmetric Systems

The Elimination Graph Model for Symmetric Matrices

Example 8.25 (Graph Representation of the Cholesky Decomposition)

$\mathcal{G}_3 =$ 

$$H_3 = \begin{bmatrix} 4 & * & * \\ * & 5 & * \\ * & * & 6 \end{bmatrix}$$

How can $\mathcal{P}(F)$ be computed?

Definition 8.26

Let $A = LL^T$ or $A = LDL^T$, then we set

$$F = L + L^T$$

and $\mathcal{P}(F)$ is the **filled pattern** of A and $\mathcal{G}(F) = \mathcal{G}^+(A)$ is the **filled graph** if A.

Definition 8.26

Let $A = LL^T$ or $A = LDL^T$, then we set

$$F = L + L^T$$

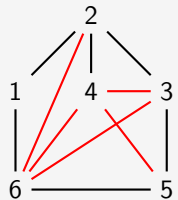
and $\mathcal{P}(F)$ is the **filled pattern** of A and $\mathcal{G}(F) = \mathcal{G}^+(A)$ is the **filled graph** if A.

Remark 8

If there is numerical cancellation during the Cholesky decomposition, i.e. elements in H_j gets zero due to the a "lucky" distribution of the values, these elements belong to the filled pattern and the filled graph as well.

Example 8.27

Obviously, the filled graph $\mathcal{G}^+(A)$ is the union the graph $\mathcal{G}_0, \mathcal{G}_1, \mathcal{G}_2$, and $\mathcal{G}_3$:

$\mathcal{G}(F) = \mathcal{G}^+(A) =$ 

$$F = \begin{bmatrix} 1 & * & & & & * \\ * & 2 & * & * & & * \\ & * & 3 & * & * & * \\ & * & * & 4 & * & * \\ & & * & * & 5 & * \\ * & * & * & * & * & 6 \end{bmatrix}$$

Direct Solvers for Sparse Symmetric Systems

The Elimination Graph Model for Symmetric Matrices

The procedure from the example yields the following observations regarding the underyling graph $\mathcal{G}_0$, $\mathcal{G}_1$, ...:

Processing a node j, i.e., performing the j-th step from $\mathcal{G}_{j-1}$ to $\mathcal{G}_j$ in the Cholesky decomposition amounts to

1. deleting node j and its adjacent edge from the graph,
2. adding edges to graph between any two nodes adjacent to j, i.e. adding edges between all nodes that are directly connected through the node j.

Direct Solvers for Sparse Symmetric Systems

The Elimination Graph Model for Symmetric Matrices

The procedure from the example yields the following observations regarding the underyling graph $\mathcal{G}_0$, $\mathcal{G}_1$, ...:

Processing a node j, i.e., performing the j-th step from $\mathcal{G}_{j-1}$ to $\mathcal{G}_j$ in the Cholesky decomposition amounts to

1. deleting node j and its adjacent edge from the graph,
2. adding edges to graph between any two nodes adjacent to j, i.e. adding edges between all nodes that are directly connected through the node j.

Algorithm 8.4: graph eliminations process

Input: $\mathcal{G}(A) = (\mathcal{V}, \mathcal{E})$ undirected graph of A
Output: $\mathcal{G}_1, \ldots, \mathcal{G}_{n-1}$ sequence of **eliminations graphs**

```
1 for k=1:n-1 do
2     V = V\{k}    (remove vertex k);
3     E = (E\{(k,l) : l neighbor of k}) ∪ {(x,y) : x,y neighbors of k};
```

Lemma 8.28

Let A be a symmetric matrix with the corresponding graph $\mathcal{G}(A)$ and the filled graph $\mathcal{G}^+(A)$. For each edge in the filled graph

$$(i,j) \in \mathcal{G}^+(A),$$

one of following condition holds:

- $(i,j) \in \mathcal{G}(A)$,
- *there exists* $k < \min(i,j)$, *such that* $(i,k) \in \mathcal{G}^+(A)$ *and* $(k,j) \in \mathcal{G}^+(A)$.

Characterization of Fill-in

Theorem 8.29 (Fill-path-theorem)

Let $L = (l_{ij})_{i,j=1,\dots,n}$ be a Cholesky factor of A, i.e., $A = LL^{\mathsf{T}}$ or $A = LDL^{\mathsf{T}}$. Then having an edge $(i,j) \in \mathcal{G}^+(A)$, i.e. $l_{ij} \neq 0$ (neglecting numerical cancellation), is equivalent to the existence of a path between i and j in $\mathcal{G}(A)$ such that all nodes (vertices) in the path have indices smaller than both i and j.

Definition 8.30

The **minimum fill-in problem** describes the problem of finding the optimal permutation of vertex labels that produces the smallest possible number of new edges in $\mathcal{G}^+(A)$ compared to $\mathcal{G}(A)$.

Example 8.31

If we want to rearrange

we have to renumber the nodes in the graph way:

$$\{1, 2, 3, \ldots, n-2, n-1, n\} \mapsto \{n, 2, 3, \ldots, n-2, n-1, 1\}$$

This is equal to swap the first with the last row and the first with last column.

Remark 9

It can be shown that the minimum fill-in problem is **NP-complete** and thus NP-hard in general. Several heuristic approaches exist that come up with sub-optimal solutions.

The name "NP-complete" is short for "nondeterministic polynomial-time complete". That means the correct solution could only be found if all possibilities are checked.

Heuristic Fill-in Reduction

Direct Solvers for Sparse Symmetric Systems

Heuristic Fill-in Reduction

Since we cannot determine the optimal fill-in reducing reordering in a reasonable amount of time, we need a heuristic, which gives a solution which is somehow close to the optimum, or has at least some other properties, which allows the direct solvers to work better.
Mainly three classes of methods exist:

Global Strategies

- structured permutation of the whole matrix
- fill-in only within the resulting structure, i.e. in the band between the entries with the largest distance to the diagonal
- Examples: (reverse) Cuthill-McKee or nested dissection

Local Heuristics

- can be incorporated into pivoting strategies, i.e. they can look at the “actual‘ values as well and not only on the structure.
- in the symmetric case: minimum degree reordering or minimum fill reordering
- in the unsymmetric case: Markowitz criterion

Direct Solvers for Sparse Symmetric Systems

Heuristic Fill-in Reduction

Local Heuristics

- can be incorporated into pivoting strategies, i.e. they can look at the “actual‘ values as well and not only on the structure.
- in the symmetric case: minimum degree reordering or minimum fill reordering
- in the unsymmetric case: Markowitz criterion

Hybrid Variants

1. Permutation to block structures
2. using local or global heuristics in the blocks

(Reverse) Cuthill-McKee Reordering (RCM)

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Definition 8.32

The **bandwidth** of a matrix $A \in \mathbb{R}^{n \times m}$ is given by

$$n_b(A) = \max_{1 \leqslant i \leqslant n} \max_{1 \leqslant j \leqslant m, a_{ij} \neq 0} |i - j|.$$

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Definition 8.32

The **bandwidth** of a matrix $A \in \mathbb{R}^{n \times m}$ is given by

$$n_b(A) = \max_{1 \leqslant i \leqslant n} \max_{1 \leqslant j \leqslant m, a_{ij} \neq 0} |i - j|.$$

The goal of the (Reverse) Cuthill-McKee reordering is to find a permutation P, such that

$$n_b(PAP^T) < n_b(A).$$

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Definition 8.32

The **bandwidth** of a matrix $A \in \mathbb{R}^{n \times m}$ is given by

$$n_b(A) = \max_{1 \leqslant i \leqslant n} \max_{1 \leqslant j \leqslant m, a_{ij} \neq 0} |i - j|.$$

The goal of the (Reverse) Cuthill-McKee reordering is to find a permutation P, such that

$$n_b(PAP^T) < n_b(A).$$

Since fill-in is only generated within the band, the memory requirements could be limited by

$$n \cdot n_b(PAP^T).$$

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Remark 10

If the bandwidth $n_b = n_b(PAP^T)$ is very small or the condition

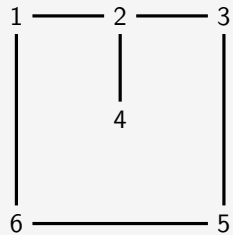
$$\mathrm{nnz}(A) \approx n \cdot n_b$$

hold, one neglects the sparsity structure and converts PAP^T into a dense band matrix. Then the Cholesky or LDL^T decomposition is computed with specialized band solvers.

See: LAPACK **`dpbtrf`** for example

Example 8.33

Influence of the ordering of the degrees of freedom on the resulting fill-in in the Cholesky decomposition is demonstrated in the following two figures.



(a) Graph before reordering.

$$\begin{bmatrix} 1 & * & & & & * \\ * & 2 & * & * & & \\ & * & 3 & & * & \\ & * & & 4 & & \\ & & * & & 5 & * \\ * & & & & * & 6 \end{bmatrix}$$

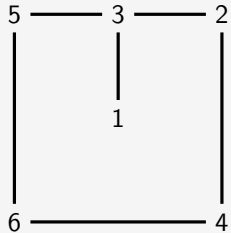
(b) Bandwidth 5 pattern.

Figure: Graph and sparsity pattern before reordering.

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Example 8.33



(a) Graph after RCM reordering.

$$\begin{bmatrix} 1 & & * & & & \\ & 2 & * & * & & \\ * & * & 3 & & * & \\ & * & & 4 & & * \\ & & * & & 5 & * \\ & & & * & * & 6 \end{bmatrix}$$

(b) Resulting bandwidth 2 pattern.

Figure: Graph and sparsity pattern after RCM reordering.

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Definition 8.34

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be graph, then the **degree** of a node $i \in \mathcal{V}$ is given by the number of neighbors in the graph, i.e. the number of edges the node is involved.

Breadth-First-Search (BFS)

Breadth-First Search (BFS) is a graph traversal algorithm used to explore the nodes and edges of a graph. It starts at a selected node (often called the "root" in the context of trees) and explores all of its neighboring nodes at the present depth prior to moving on to nodes at the next depth level.

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

(Reverse) Cuthill-McKee Reordering

Basically, Cuthill-McKee Reordering works in the following way:

1. select a root node, forms the tree that consists of all shortest paths to all other vertices in $\mathcal{G}(A)$
2. perform an **ordered** breadth first search on that tree to fill the permutation vector.
3. **reverse the computed permutation**

In contrast to a standard breadth first search, here the vertices are ordered with respect to their increasing degree.

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

(Reverse) Cuthill-McKee Reordering

Basically, Cuthill-McKee Reordering works in the following way:

1. select a root node, forms the tree that consists of all shortest paths to all other vertices in $\mathcal{G}(A)$
2. perform an **ordered** breadth first search on that tree to fill the permutation vector.
3. **reverse the computed permutation**

In contrast to a standard breadth first search, here the vertices are ordered with respect to their increasing degree.

$\rightarrow$ One observed that without the last step, the permuted matrix has the same bandwidth, but more fill-in.

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Algorithm 8.5: Reverse Cuthill-McKee (RCM) reordering

Input: $A \in \mathbb{R}^{n \times n}$ with $\mathcal{P}(A)$ symmetric

Output: $p \in \mathbb{R}^n$ such that $\tilde{A} = A(p, p)$ has reduced bandwidth

```
1  Q = [], R = [];
2  repeat
3      Select root node P ∉ R;
4      R = [R, P];
5      Q = [Q, nodes adjacent to P ordered by increasing degree];
6      while Q ≠ ∅ do
7          R = [R, Q(1)],
8          Q = [Q(2 : End), nodes adjacent of Q(1) not contained in R by increasing degree];
9  until all nodes are contained in R;
10 p = R(n : −1 : 1);
```

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Definition 8.35

Let p a permutation vector, $p \in \{1, \ldots, n\}^n$, where p maps and index i to p_i. Then the **inverse permutation** $\tilde{p} \in \{1, \ldots, n\}^n$ fulfills

$$\tilde{p}(p) = p(\tilde{p}) = \begin{bmatrix} 1 & 2 & \ldots & n \end{bmatrix}.$$

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Definition 8.35

Let p a permutation vector, $p \in \{1, \ldots, n\}^n$, where p maps and index i to p_i. Then the **inverse permutation** $\tilde{p} \in \{1, \ldots, n\}^n$ fulfills

$$\tilde{p}(p) = p(\tilde{p}) = \begin{bmatrix} 1 & 2 & \ldots & n \end{bmatrix}.$$

The element $\tilde{p}_i$ is given by the index j of p where $p_j = i$.

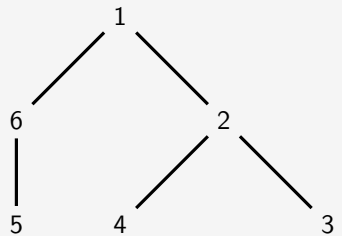
Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Example 8.36

This example shows the importance of the selection of the root node in Step 3 of Algorithm 8.5.

Root-Node: $P = 1$



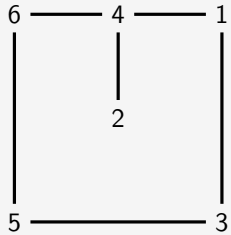
$$
\begin{array}{ll}
R = [1] & Q = [6, 2] \\
R = [1, 6] & Q = [2, 5] \\
R = [1, 6, 2] & Q = [5, 4, 3] \\
R = [1, 6, 2, 5] & Q = [4, 3] \\
R = [1, 6, 2, 5, 4, 3] & Q = [\,]
\end{array}
$$

$$
\begin{aligned}
p_1 &= [3, 4, 5, 2, 6, 1] \\
\tilde{p}_1 &= [6, 4, 1, 2, 3, 5]
\end{aligned}
$$

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Example 8.36



(a) Graph after RCM reordering.

$$\begin{bmatrix} 1 & & * & * & & \\ & 2 & & * & & \\ * & & 3 & & * & \\ * & * & & 4 & & * \\ & & * & & 5 & * \\ & & & * & * & 6 \end{bmatrix}$$

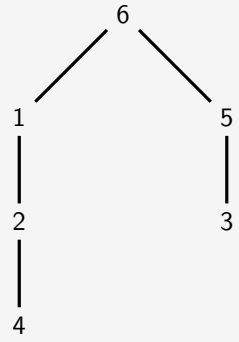
(b) Resulting bandwidth 3 pattern.

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Example 8.36

Root-Node: $P = 6$



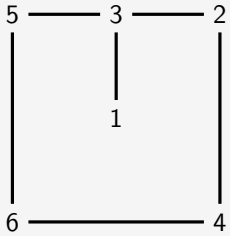
$$
\begin{aligned}
R &= [6] & Q &= [1,5] \\
R &= [6,1] & Q &= [5,2] \\
R &= [6,1,5] & Q &= [2,3] \\
R &= [6,1,5,2] & Q &= [3,4] \\
R &= [6,1,5,2,3,4] & Q &= []
\end{aligned}
$$

$$
\begin{aligned}
p_6 &= [4,3,2,5,1,6] \\
\tilde{p}_6 &= [5,3,2,1,4,6]
\end{aligned}
$$

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Example 8.36



(a) Graph after RCM reordering.

$$\begin{bmatrix} 1 & & * & & & \\ & 2 & * & * & & \\ * & * & 3 & & * & \\ & * & & 4 & & * \\ & & * & & 5 & * \\ & & & * & * & 6 \end{bmatrix}$$

(b) Resulting bandwidth 2 pattern.

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Example 8.36

The p_6 permutation is the one produced by **`symrcm`** implementation in MATLAB and GNU Octave. Note that the matrix A is transformed into the reduced-bandwidth matrix $\tilde{A}$ as $\tilde{A} = A(p, p)$, while the graph uses the inverse permutation.

A bad choice for the root node is $P = 2$.

Direct Solvers for Sparse Symmetric Systems

(Reverse) Cuthill-McKee Reordering (RCM)

Root-Node-Selection

The exampled showed, that $P = 6$ was a good root node, $P = 1$ was not the best, but somehow "ok", but $P = 2$ end in a bad result.

As a rule of thumb, the root node should be chosen such that it has preferably long paths to all other nodes in the graph.

$\rightarrow$ Tracking the depth of each node with respect to the current root node, and start the RCM again with the "deepest" node. Repeat until the maximum depth is no longer increasing.

Direct Solvers for Sparse Symmetric Systems

Local heuristics

We employ the following assumptions:

- $A \in \mathbb{R}^{n \times n}$ sparse symmetric
- $\mathcal{G}(A) = (\mathcal{V}, \mathcal{E})$ the corresponding undirected graph of A
- $m\colon V \to \mathbb{R}$ a metric, such that $m(i) < m(j)$ implies that node i is "better" than node j.

Algorithm 8.5: Generic local strategy

Input: $A \in \mathbb{R}^{n \times n}$ sparse, m a metric on the nodes in $\mathcal{G}(A)$, $p = []$
Output: $p \in \mathbb{R}^n$ such that $\tilde{A} = A(p, p)$ is the reordered matrix

1 **repeat**
2 Select a node P (the pivot element) with minimal metric value $m(P)$: $p = [p, P]$;
3 Update elimination graph erasing P;
4 Update metric for all non-selected nodes;
5 **until** **all nodes selected**;

Remark 11

- Step 4 in Algorithm 8.5 should be restricted to those nodes where m changed due to the graph update.
- The local pivot search allows combination with classic pivot strategies to improve the numerical results.

Direct Solvers for Sparse Symmetric Systems

Local heuristics – Minimum degree idea

- $m(j)$ is the number of neighbors of node j
- $m(i) < m(j)$ means node i has less neighbors than node j.

$\rightarrow$ metric update only required on the neighbors

$\rightarrow$ very local update

$\rightarrow$ the procedure is integrated in the generation of the elimination graph.

Direct Solvers for Sparse Symmetric Systems

Local heuristics

Example 8.37

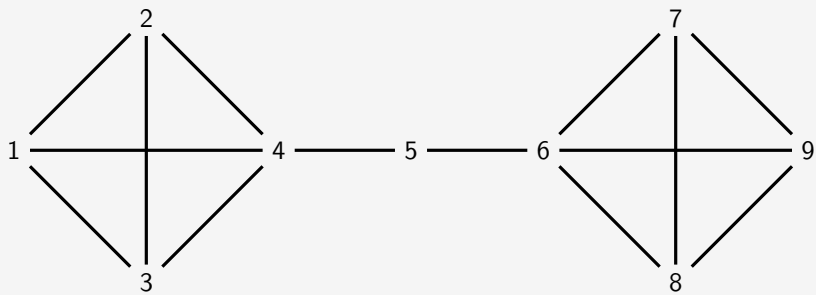
We consider the following matrix $A \in \mathbb{R}^{9\times 9}$

$$A = \begin{bmatrix} 1 & * & * & * & & & & & \\ * & 2 & * & * & & & & & \\ * & * & 3 & * & & & & & \\ * & * & * & 4 & * & & & & \\ & & & * & 5 & * & & & \\ & & & & * & 6 & * & * & * \\ & & & & & * & 7 & * & * \\ & & & & & * & * & 8 & * \\ & & & & & * & * & * & 9 \end{bmatrix},$$

for which, by construction, factorization is possible without fill-in.

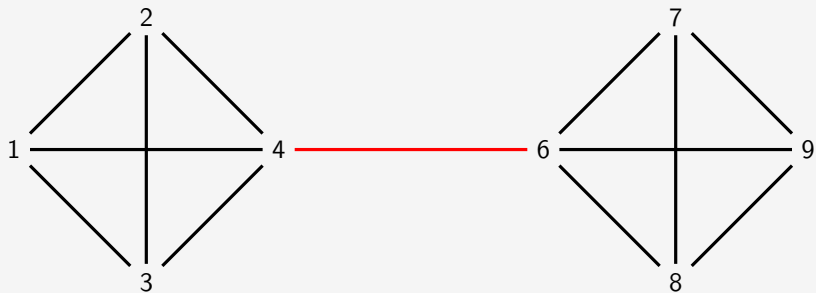
Example 8.37

$\mathcal{G}(A)$ looks like this



Example 8.37

Now the minimum degree metric suggests to choose node 5 (of degree 2) for elimination, which results in:



Example 8.37

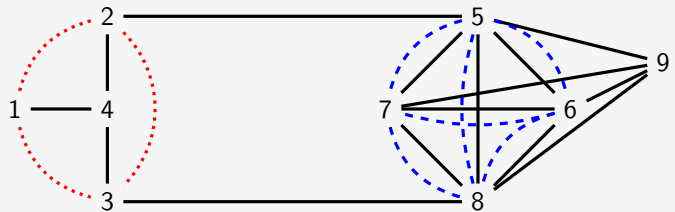
This obviously introduces a new edge from node 4 to node 6, i.e., results in fill-in. This is still better than choosing node 4 or 6 (both degree 3), which would lead to two new edges each, i.e. more fill-in. On the other hand, all other nodes (also degree 3) could obviously be removed without causing additional edges.

Remark 12

All heuristic approaches to the minimum fill problem in general only produce suboptimal solutions. This is however clear, since the optimal solution is usually not accessible since it is the solution to an NP-hard problem.

Example 8.38 (minimum degree metric versus minimum fill metric)

The following simple graph (—— edges) shows the discrepancies between minimum degree and minimum fill as metrics.



- ····· edges indicate the fill resulting from the removal of node 4.
- - - - edges resulting from the removal of node 9, already existing.

Node 4 – degree 3 and the fill measures 3. Node 9 – degree 4, fill measure 0.

Example 8.38 (minimum degree metric versus minimum fill metric)

node	degree metric value	fill metric value
1	1	0
2	2	1
3	2	1
4	3	3
5	5	4
6	4	0
7	4	0
8	5	4
9	4	0

Hybrid method and graph components

Definition 8.39

In an undirected graph $\mathcal{G}$ two vertices u and v are called **connected** if $\mathcal{G}$ contains a path from u to v. Otherwise, they are called **disconnected**.
A **Graph** $\mathcal{G}$ is said to be **connected** if each pair of vertices is connected. A **connected component** is a maximal connected subgraph of $\mathcal{G}$.

Definition 8.39

In an undirected graph $\mathcal{G}$ two vertices u and v are called **connected** if $\mathcal{G}$ contains a path from u to v. Otherwise, they are called **disconnected**.
A **Graph** $\mathcal{G}$ is said to be **connected** if each pair of vertices is connected. A **connected component** is a maximal connected subgraph of $\mathcal{G}$.

- If u, v are vertices in $\mathcal{G}$ from different connected components, then u, v are disconnected. Thus, the corresponding degrees of freedom in the linear system are independent of each other.
- Reordering A corresponding to the connected components leads to a block diagonal matrix. The resulting diagonal blocks can then be treated independently.
- For general non-symmetric matrices **strongly connected components** have to be used. That means, both directed paths between two vertices need to exist.
  $\rightarrow$ not all diagonal blocks decouple completely, since only one direction may exist for a pair of vertices in two components.

Sparse Matrix Vector Products and Reordering

Naively looking at the problem one might think:

Even if the elements in A are scattered all over the row, in the CSR format they are stored one after the other, anyway. This would lead us to the expectation that we get no advantage due to reordering.

Naively looking at the problem one might think:

Even if the elements in A are scattered all over the row, in the CSR format they are stored one after the other, anyway. This would lead us to the expectation that we get no advantage due to reordering.

RCM Reordering

Consider an RCM reordered matrix with small bandwidth. The relevant indices corresponding to the entries are local, as well. Thus, a local portion of x is used. Additionally, the next row has a very similar set of indices containing entries. That means, in the next row product almost the entire portion of x can be reused, which leads to only little cache misses on x.

Related Software

Direct Solvers for Sparse Symmetric Systems

Related Software

SuiteSparse – `https://people.engr.tamu.edu/davis/suitesparse.html`:

- CSparse, basic direct solver library for sparse linear systems, follows the book "Direct solution of sparse linear systems" by Tim Davis.
- UMFPack, unsymmetric multifrontal solver, used for "\" in MATLAB and GNU Octave.
- Cholmod, high performance Cholesky decomposition
- various local heuristic reorderings, like **Approximate Minimum Degree**

SuperLU – `https://portal.nersc.gov/project/sparse/superlu/`:

- Sparse LU decomposition with the Super-Node approach

ITPACK – `https://www.netlib.org/itpack/`:

- Different Sparse Solvers

Direct Solvers for Sparse Symmetric Systems

Related Software