Linear Algebra

Brigitte Bidégaray-Fesquet

Univ. Grenoble Alpes, Laboratoire Jean Kuntzmann, Grenoble

MSIAM, 23–24 September 2015

Overview

Overview

Elementary operations on vectors

$\mathbb{C}^n$ resp. $\mathbb{R}^n$: linear space of vectors with n entries in $\mathbb{C}$ resp. $\mathbb{R}$.
Generically: F^n, where F is a field.

Linear combination of vectors

$\vec{w} = \alpha\vec{u} + \beta\vec{v}$, $\alpha, \beta \in \mathbb{C}$ or $\mathbb{R}$.

```
for i = 1 to n
  w(i) = alpha * u(i) + beta * v(i)
end for
```

Scalar product of 2 vectors

$$\vec{u} \cdot \vec{v} = \sum_{i=1}^{n} u_i v_i$$

```
uv = 0
for i = 1 to n
  uv = uv + u(i) * v(i)
end for
```

ℓ^2 norm of a vector

$$\|\vec{u}\|_2 = \sqrt{\sum_{i=1}^{n} u_i^2}$$

```
uu = 0
for i = 1 to n
  uu = uu + u(i) * u(i)
end for
norm = sqrt(uu)
```

Elementary operations on matrices

$\mathcal{M}_{np}(F)$: linear space of matrices with $n \times p$ entries in F.

Linear combination of matrices

$C = \alpha A + \beta B$, $\alpha, \beta \in F$.

```
for i = 1 to n
  for j = 1 to p
    C(i,j) = alpha * A(i,j) + beta * B(i,j)
  end for
end for
```

Matrix–vector product

$$\vec{w} = A\vec{u},\ w_i = \sum_{j=1}^{p} A_{ij} u_j$$

```
for i = 1 to n
  wi = 0
  for j = 1 to p
    wi = wi + A(i,j) * u(j)
  end for
  w(i) = wi
end for
```

Matrix–matrix product

$$C = AB,\ C_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj}$$

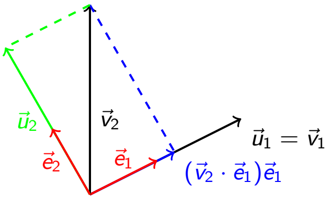
```
for i = 1 to n
  for j = 1 to q
    cij = 0
    for k = 1 to p
      cij = cij + A(i,k) * B(k,j)
    end for
    C(i,j) = cij
  end for
end for
```

Gram–Schmidt orthonormalization

Let $\{\vec{v}_1, \dots, \vec{v}_p\}$ be a free family of vectors.
It generates the vector space E_p with dimension p.
We want to construct $\{\vec{e}_1, \dots, \vec{e}_p\}$, an orthonormal basis of E_p.

Gram–Schmidt algorithm

$$
\begin{aligned}
\vec{u}_1 &= \vec{v}_1 & \vec{e}_1 &= \frac{\vec{u}_1}{\|\vec{u}_1\|_2} \\
\vec{u}_2 &= \vec{v}_2 - (\vec{v}_2 \cdot \vec{e}_1)\vec{e}_1 & \vec{e}_2 &= \frac{\vec{u}_2}{\|\vec{u}_2\|_2} \\
&\dots & &\dots \\
\vec{u}_p &= \vec{v}_p - \sum_{k=1}^{p-1} (\vec{v}_p \cdot \vec{e}_k)\vec{e}_k & \vec{e}_p &= \frac{\vec{u}_p}{\|\vec{u}_p\|_2}
\end{aligned}
$$



Matrix norms

Definition

$$
\begin{array}{ll}
\|A\| \geq 0, & \forall A \in \mathcal{M}_{nn}(F),\ \ F = \mathbb{C} \text{ or } \mathbb{R}. \\
\|A\| = 0 \Leftrightarrow A = 0. & \\
\|\lambda A\| = |\lambda| \|A\|, & \forall A \in \mathcal{M}_{nn}(F),\ \forall \lambda \in F. \\
\|A + B\| \leq \|A\| + \|B\|, & \forall A, B \in \mathcal{M}_{nn}(F) \text{ (triangle inequality)}. \\
\|AB\| \leq \|A\| \|B\|, & \forall A, B \in \mathcal{M}_{nn}(F) \text{ (specific for matrix norms)}.
\end{array}
$$

Matrix norms

Definition

$\|A\| \geq 0$, $\quad \forall A \in \mathcal{M}_{nn}(F)$, $F = \mathbb{C}$ or $\mathbb{R}$.

$\|A\| = 0 \Leftrightarrow A = 0$.

$\|\lambda A\| = |\lambda| \|A\|$, $\quad \forall A \in \mathcal{M}_{nn}(F)$, $\forall \lambda \in F$.

$\|A + B\| \leq \|A\| + \|B\|$, $\quad \forall A, B \in \mathcal{M}_{nn}(F)$ (triangle inequality).

$\|AB\| \leq \|A\| \|B\|$, $\quad \forall A, B \in \mathcal{M}_{nn}(F)$ (specific for matrix norms).

Subordinate matrix norms

$$\|A\|_p = \max_{\|x\|_p \neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p = 1} \|Ax\|_p, \ \forall x \in F^n, \text{ where } \|\vec{x}\|_p = \sqrt[p]{\sum_{i=1}^{n} x_i^p}.$$

in particular: $\|A\|_1 = \max_j \sum_i |A_{ij}|$ and $\|A\|_\infty = \max_i \sum_j |A_{ij}|$.

Matrix norms

Definition

$\|A\| \geq 0, \qquad \forall A \in \mathcal{M}_{nn}(F),\ F = \mathbb{C}$ or $\mathbb{R}$.

$\|A\| = 0 \Leftrightarrow A = 0.$

$\|\lambda A\| = |\lambda| \|A\|, \qquad \forall A \in \mathcal{M}_{nn}(F),\ \forall \lambda \in F.$

$\|A + B\| \leq \|A\| + \|B\|, \qquad \forall A, B \in \mathcal{M}_{nn}(F)$ (triangle inequality).

$\|AB\| \leq \|A\| \|B\|, \qquad \forall A, B \in \mathcal{M}_{nn}(F)$ (specific for matrix norms).

Subordinate matrix norms

$$\|A\|_p = \max_{\|x\|_p \neq 0} \frac{\|Ax\|_p}{\|x\|_p} = \max_{\|x\|_p = 1} \|Ax\|_p,\ \forall x \in F^n, \text{ where } \|\vec{x}\|_p = \sqrt[p]{\sum_{i=1}^{n} x_i^p}.$$

in particular: $\|A\|_1 = \max_j \sum_i |A_{ij}|$ and $\|A\|_\infty = \max_i \sum_j |A_{ij}|$.

Matrix-vector product estimate

$\|A\|_p \geq \frac{\|Ax\|_p}{\|x\|_p}$ and hence $\|Ax\|_p \leq \|A\|_p \|x\|_p$ for all $x \in F^n$.

Matrix conditioning

Definition

$$\mathrm{Cond}(A) = \|A^{-1}\|\|A\|.$$

Matrix conditioning

Definition

$$\mathrm{Cond}(A) = \|A^{-1}\|\|A\|.$$

Properties

$$\mathrm{Cond}(A) \geq 1,$$
$$\mathrm{Cond}(A^{-1}) = \mathrm{Cond}(A),$$
$$\mathrm{Cond}(\alpha A) = \mathrm{Cond}(A).$$

Matrix conditioning

Definition

$$\text{Cond}(A) = \|A^{-1}\|\|A\|.$$

Properties

$$\text{Cond}(A) \geq 1,$$
$$\text{Cond}(A^{-1}) = \text{Cond}(A),$$
$$\text{Cond}(\alpha A) = \text{Cond}(A).$$

For the Euclidian norm

$$\text{Cond}_2(A) = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}.$$

Conditioning and linear systems

Problem

$(S_0)\ A\vec{x} = \vec{b}, \qquad (S_{\text{per}})\ (A + \delta A)(\vec{x} + \delta\vec{x}) = (\vec{b} + \delta\vec{b}).$

$(S_{\text{per}}) - (S_0)$: $A\delta\vec{x} + \delta A(\vec{x} + \delta\vec{x}) = \delta\vec{b}$,

$\delta\vec{x} = A^{-1}\left(\delta\vec{b} - \delta A(\vec{x} + \delta\vec{x})\right)$,

$\|\delta\vec{x}\| \leq \|A^{-1}\| \left\|\delta\vec{b} - \delta A(\vec{x} + \delta\vec{x})\right\|$ (for a subordinate matrix norm),

$\|\delta\vec{x}\| \leq \|A^{-1}\| \left(\|\delta\vec{b}\| + \|\delta A\| \|\vec{x} + \delta\vec{x}\|\right)$,

$$\frac{\|\delta\vec{x}\|}{\|\vec{x} + \delta\vec{x}\|} \leq \|A^{-1}\| \left(\frac{\|\delta\vec{b}\|}{\|\vec{x} + \delta\vec{x}\|} + \|\delta A\|\right).$$

Conditioning and linear systems

Problem

$(S_0)\ A\vec{x} = \vec{b}$, $\quad (S_{\text{per}})\ (A + \delta A)(\vec{x} + \delta\vec{x}) = (\vec{b} + \delta\vec{b})$.

$(S_{\text{per}}) - (S_0)$: $A\delta\vec{x} + \delta A(\vec{x} + \delta\vec{x}) = \delta\vec{b}$,

$\delta\vec{x} = A^{-1}\left(\delta\vec{b} - \delta A(\vec{x} + \delta\vec{x})\right)$,

$\|\delta\vec{x}\| \leq \|A^{-1}\| \left\|\delta\vec{b} - \delta A(\vec{x} + \delta\vec{x})\right\|$ (for a subordinate matrix norm),

$\|\delta\vec{x}\| \leq \|A^{-1}\| \left(\|\delta\vec{b}\| + \|\delta A\|\|\vec{x} + \delta\vec{x}\|\right)$,

$$\frac{\|\delta\vec{x}\|}{\|\vec{x} + \delta\vec{x}\|} \leq \|A^{-1}\| \left(\frac{\|\delta\vec{b}\|}{\|\vec{x} + \delta\vec{x}\|} + \|\delta A\|\right).$$

Result

$$\frac{\|\delta\vec{x}\|}{\|\vec{x} + \delta\vec{x}\|} \leq \text{Cond}(A) \left(\frac{\|\delta\vec{b}\|}{\|A\|\|\vec{x} + \delta\vec{x}\|} + \frac{\|\delta A\|}{\|A\|}\right).$$

relative error on x = Cond(A) (relative error on $\vec{b}$ + relative error on A).

Hermitian, orthogonal...

Transposed matrix: $({}^tA)_{ij} = A_{ji}$.
Adjoint matrix: $(A^*)_{ij} = \overline{A_{ji}}$.

Symmetric matrix

${}^tA = A$.

Hermitian, orthogonal...

Transposed matrix: $({}^tA)_{ij} = A_{ji}$.
Adjoint matrix: $(A^*)_{ij} = \overline{A_{ji}}$.

Symmetric matrix

${}^tA = A$.

Hermitian matrix

$A^* = A$ and hence ${}^tA = \bar{A}$.

Hermitian, orthogonal...

Transposed matrix: $({}^tA)_{ij} = A_{ji}$.
Adjoint matrix: $(A^*)_{ij} = \overline{A_{ji}}$.

Symmetric matrix

${}^tA = A$.

Hermitian matrix

$A^* = A$ and hence ${}^tA = \bar{A}$.

Orthogonal matrix (in $\mathcal{M}_{nn}(\mathbb{R})$)

${}^tAA = I$.

Hermitian, orthogonal...

Transposed matrix: $({}^tA)_{ij} = A_{ji}$.
Adjoint matrix: $(A^*)_{ij} = \overline{A_{ji}}$.

Symmetric matrix

${}^tA = A$.

Hermitian matrix

$A^* = A$ and hence ${}^tA = \bar{A}$.

Orthogonal matrix (in $\mathcal{M}_{nn}(\mathbb{R})$)

${}^tAA = I$.

Unitary matrix (in $\mathcal{M}_{nn}(\mathbb{C})$)

$A^*A = I$.

Hermitian, orthogonal...

Transposed matrix: $({}^tA)_{ij} = A_{ji}$.
Adjoint matrix: $(A^*)_{ij} = \overline{A_{ji}}$.

Symmetric matrix

${}^tA = A$.

Hermitian matrix

$A^* = A$ and hence ${}^tA = \bar{A}$.

Orthogonal matrix (in $\mathcal{M}_{nn}(\mathbb{R})$)

${}^tAA = I$.

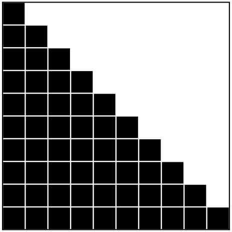
Unitary matrix (in $\mathcal{M}_{nn}(\mathbb{C})$)

$A^*A = I$.

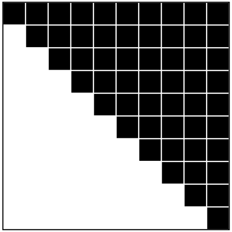
Similar matrices ("semblables" in French)

A and B are similar if $\exists P / B = P^{-1}AP$.

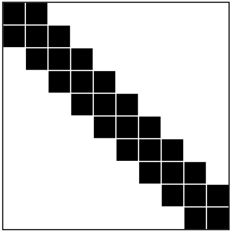
Profiles



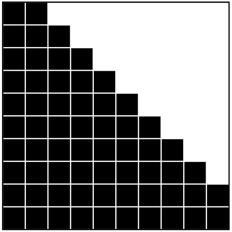
Lower triangular



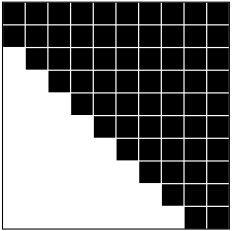
Upper triangular



Tridiagonal



Lower Hessenberg



Upper Hessenberg

Householder matrices

Definition

$$H_{\vec{v}} = I - 2\frac{\vec{v}^t\vec{v}}{\|\vec{v}\|_2^2}$$

Householder matrices

Definition

$$H_{\vec{v}} = I - 2\frac{\vec{v}^t\vec{v}}{\|\vec{v}\|_2^2}$$

Properties

Householder matrices

Definition

$$H_{\vec{v}} = I - 2\frac{\vec{v}^t\vec{v}}{\|\vec{v}\|_2^2}$$

Properties

1. $H_{\vec{v}}$ is orthogonal.

Householder matrices

Definition

$$H_{\vec{v}} = I - 2\frac{\vec{v}^t\vec{v}}{\|\vec{v}\|_2^2}$$

Properties

1. $H_{\vec{v}}$ is orthogonal.
2. If $\vec{v} = \vec{a} - \vec{b} \neq \vec{0}$ and $\|\vec{a}\|_2 = \|\vec{b}\|_2$, then $H_{\vec{v}}\vec{a} = \vec{b}$.

Householder matrices

Definition

$$H_{\vec{v}} = I - 2\frac{\vec{v}^t\vec{v}}{\|\vec{v}\|_2^2}$$

Properties

1. $H_{\vec{v}}$ is orthogonal.
2. If $\vec{v} = \vec{a} - \vec{b} \neq \vec{0}$ and $\|\vec{a}\|_2 = \|\vec{b}\|_2$, then $H_{\vec{v}}\vec{a} = \vec{b}$.

$${}^t\vec{v}\vec{v} = \|\vec{a}\|_2 - 2{}^t\vec{a}\vec{b} + \|\vec{b}\|_2 = 2\|\vec{a}\|_2 - 2{}^t\vec{a}\vec{b} = 2{}^t\vec{a}\vec{v} = 2{}^t\vec{v}\vec{a}$$
$$H_{\vec{v}}\vec{a} = \vec{a} - \frac{2\vec{v}^t\vec{v}\vec{a}}{\|\vec{v}\|_2} = \vec{a} - \vec{v} = \vec{b}.$$

Householder matrices

Linear Algebra

Vectors and matrices
Elementary operations
Gram–Schmidt orthonormalization
Matrix norm
Conditioning
Specific matrices
Tridiagonalisation
LU and QR factorizations
Eigenvalues and eigenvectors
Numerical solution of linear systems
Storage
Bandwidth reduction
References

Definition

$$H_{\vec{v}} = I - 2\frac{\vec{v}\,^t\vec{v}}{\|\vec{v}\|_2^2}$$

Properties

1. $H_{\vec{v}}$ is orthogonal.
2. If $\vec{v} = \vec{a} - \vec{b} \neq \vec{0}$ and $\|\vec{a}\|_2 = \|\vec{b}\|_2$, then $H_{\vec{v}}\vec{a} = \vec{b}$.

$$^t\vec{v}\vec{v} = \|\vec{a}\|_2 - 2\,^t\vec{a}\vec{b} + \|\vec{b}\|_2 = 2\|\vec{a}\|_2 - 2\,^t\vec{a}\vec{b} = 2\,^t\vec{a}\vec{v} = 2\,^t\vec{v}\vec{a}$$

$$H_{\vec{v}}\vec{a} = \vec{a} - \frac{2\vec{v}\,^t\vec{v}\vec{a}}{\|\vec{v}\|_2} = \vec{a} - \vec{v} = \vec{b}.$$

Application

Let $\vec{a} \in K^n$, we look for $H_{\vec{v}}$ such that $H_{\vec{v}}\vec{a} = {}^t(\alpha, 0, \ldots, 0)$.
Solution: take $\vec{b} = {}^t(\alpha, 0, \ldots, 0)$ with $\alpha = \|\vec{a}\|_2$, and $\vec{v} = \vec{a} - \vec{b}$. Then $H_{\vec{v}}\vec{a} = \vec{b}$.

Householder tridiagonalisation

Aim

A: symmetric matrix.

Construct a sequence $A^{(1)} = A, \ldots, A^{(n)}$ tridiagonal and $A^{(n)}n = HA^tH$.

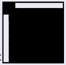
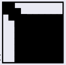
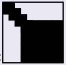
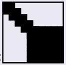
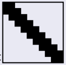
$A^{(2)} =$, $A^{(3)} =$, $A^{(4)} =$, $A^{(5)} =$, $\ldots A^{(n)} =$

Householder tridiagonalisation

Aim

A: symmetric matrix.

Construct a sequence $A^{(1)} = A, \dots, A^{(n)}$ tridiagonal and $A^{(n)} n = HA^t H$.

$A^{(2)} =$  $A^{(3)} =$  $A^{(4)} =$  $A^{(5)} =$  $\dots A^{(n)} =$ 

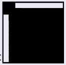
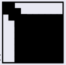
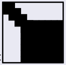
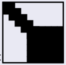
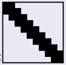
First step

$$A^{(1)} \equiv \begin{pmatrix} A^{(1)}_{11} & {}^t\vec{a}^{(1)}_{12} \\ \vec{a}^{(1)}_{21} & \tilde{A}^{(1)} \end{pmatrix} \quad H^{(1)} \equiv \begin{pmatrix} 1 & {}^t\vec{0} \\ \vec{0} & \tilde{H}^{(1)} \end{pmatrix} \quad A^{(2)} \equiv \begin{pmatrix} A^{(1)}_{11} & {}^t(\tilde{H}^{(1)}\vec{a}^{(1)}_{21}) \\ \tilde{H}^{(1)}\vec{a}^{(1)}_{21} & \tilde{H}^{(1)}\tilde{A}^{(1)}\,{}^t\tilde{H}^{(1)} \end{pmatrix}.$$

Householder tridiagonalisation

Aim

A: symmetric matrix.
Construct a sequence $A^{(1)} = A, \dots, A^{(n)}$ tridiagonal and $A^{(n)} n = HA^t H$.

$A^{(2)} =$  $A^{(3)} =$  $A^{(4)} =$  $A^{(5)} =$  $\dots A^{(n)} =$ 

First step

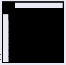
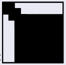
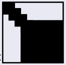
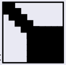
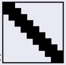
$$A^{(1)} \equiv \begin{pmatrix} A_{11}^{(1)} & {}^t\vec{a}_{12}^{(1)} \\ \vec{a}_{21}^{(1)} & \tilde{A}^{(1)} \end{pmatrix} \quad H^{(1)} \equiv \begin{pmatrix} 1 & {}^t\vec{0} \\ \vec{0} & \tilde{H}^{(1)} \end{pmatrix} \quad A^{(2)} \equiv \begin{pmatrix} A_{11}^{(1)} & {}^t(\tilde{H}^{(1)}\vec{a}_{21}^{(1)}) \\ \tilde{H}^{(1)}\vec{a}_{21}^{(1)} & \tilde{H}^{(1)}\tilde{A}^{(1)}\,{}^t\tilde{H}^{(1)} \end{pmatrix}.$$

Choose $\tilde{H}^{(1)}$ such that $\tilde{H}^{(1)}\vec{a}_{21}^{(1)} = {}^t(\alpha, 0, \dots, 0)_{n-1} = \alpha(\vec{e}_1)_{n-1}$.
$\alpha = \|\vec{a}_{21}^{(1)}\|_2$, $\vec{u}_1 = \vec{a}_{21}^{(1)} - \alpha(\vec{e}_1)_{n-1}$, $\tilde{H}^{(1)} = H_{\vec{u}_1}$.

Householder tridiagonalisation

Aim

A: symmetric matrix.
Construct a sequence $A^{(1)} = A, \ldots, A^{(n)}$ tridiagonal and $A^{(n)} n = HA^t H$.

$A^{(2)} =$  $A^{(3)} =$  $A^{(4)} =$  $A^{(5)} =$  $\ldots A^{(n)} =$ 

First step

$$A^{(1)} \equiv \begin{pmatrix} A^{(1)}_{11} & {}^t\vec{a}^{(1)}_{12} \\ \vec{a}^{(1)}_{21} & \tilde{A}^{(1)} \end{pmatrix} H^{(1)} \equiv \begin{pmatrix} 1 & {}^t\vec{0} \\ \vec{0} & \tilde{H}^{(1)} \end{pmatrix} A^{(2)} \equiv \begin{pmatrix} A^{(1)}_{11} & {}^t(\tilde{H}^{(1)}\vec{a}^{(1)}_{21}) \\ \tilde{H}^{(1)}\vec{a}^{(1)}_{21} & \tilde{H}^{(1)}\tilde{A}^{(1)\,t}\tilde{H}^{(1)} \end{pmatrix}.$$

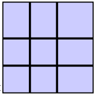
Choose $\tilde{H}^{(1)}$ such that $\tilde{H}^{(1)}\vec{a}^{(1)}_{21} = {}^t(\alpha, 0, \ldots, 0)_{n-1} = \alpha(\vec{e}_1)_{n-1}$.
$\alpha = \|\vec{a}^{(1)}_{21}\|_2$, $\vec{u}_1 = \vec{a}^{(1)}_{21} - \alpha(\vec{e}_1)_{n-1}$, $\tilde{H}^{(1)} = H_{\vec{u}_1}$.

Complexity

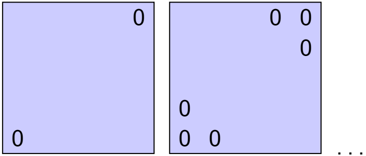
Order $\dfrac{2}{3}n^3$ products.

Givens tridiagonalization

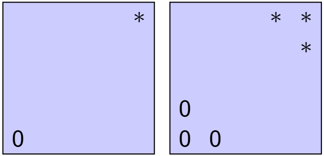
Let $G_{pq}(c,s) = \begin{pmatrix} 1 & & & & & & 0 \\ & 1 & & & & & \\ & & c & & s & & \\ & & & 1 & & & \\ & & -s & & c & & \\ & & & & & 1 & \\ 0 & & & & & & 1 \end{pmatrix}$ with $c^2 + s^2 = 1$.

${}^tGAG =$ 

If A is symmetric:



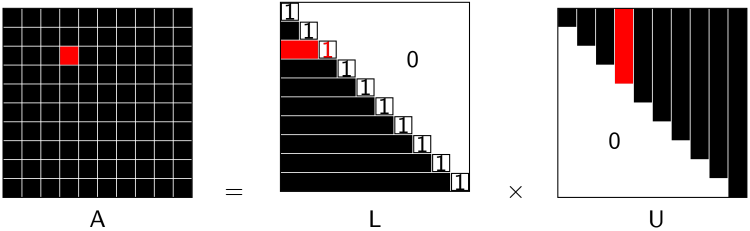
else: leads to Hessenberg matrix



Complexity

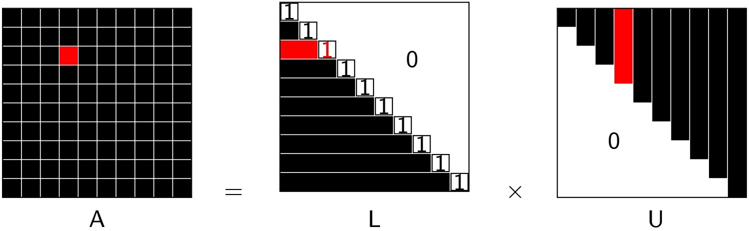
Order $\frac{4}{3}n^3$ products.

Principles of LU factorization



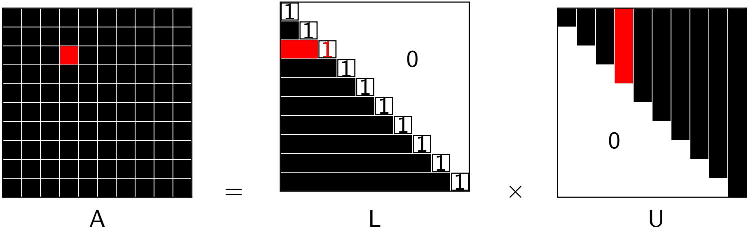
- Some regular matrix (with non-zero determinant) are not LU-transformable, e.g. ([0 1; 1 1]) is not.

Principles of LU factorization



- Some regular matrix (with non-zero determinant) are not LU-transformable, e.g. ([0 1; 1 1]) is not.
- If it exists, the LU decomposition of A is not unique.
  It is unique if A is non-singular.

Principles of LU factorization



- Some regular matrix (with non-zero determinant) are not LU-transformable, e.g. $([0\ 1; 1\ 1])$ is not.
- If it exists, the LU decomposition of A is not unique.
  It is unique if A is non-singular.
- A is non-singular and LU-transformable
  $\Longleftrightarrow$ all the determinants of the fundamental principal minors are non zero (and in this case the decomposition is unique).

Doolittle LU factorization – principle

It proceeds line by line.

$$\left\{\begin{array}{lcl} A_{11} & = & L_{11}U_{11} \\ A_{12} & = & L_{11}U_{12} \\ & \dots & \\ A_{1n} & = & L_{11}U_{1n} \end{array}\right. \qquad L_{11}=1 \quad \Rightarrow \{U_{1j}\}_{j=1,\dots,n}$$

$$\left\{\begin{array}{lcl} A_{21} & = & L_{21}U_{11} \quad \Rightarrow L_{21} \\ A_{22} & = & L_{21}U_{12}+U_{22} \\ & \dots & \\ A_{2n} & = & L_{21}U_{1n}+U_{2n} \end{array}\right. \quad \Rightarrow \{U_{2j}\}_{j=2,\dots,n}$$

$$\left\{\begin{array}{lcl} A_{31} & = & L_{31}U_{11} \quad \Rightarrow L_{31} \\ A_{32} & = & L_{31}U_{12}+L_{32}U_{22} \quad \Rightarrow L_{32} \\ A_{33} & = & L_{31}U_{13}+L_{32}U_{23}+U_{33} \\ & \dots & \\ A_{3n} & = & L_{31}U_{1n}+L_{32}U_{2n}+U_{3n} \end{array}\right. \quad \Rightarrow \{U_{3j}\}_{j=3,\dots,n}$$

$$\dots$$

Doolittle LU factorization – algorithm

Doolittle algorithm

$$L_{ij} = \frac{A_{ij} - \sum_{k=1}^{j-1} L_{ik} U_{kj}}{U_{jj}} \qquad U_{ij} = A_{ij} - \sum_{k=1}^{i-1} L_{ik} U_{kj}$$

```
for i = 1 to n
  for j = 1 to i-1
    sum = 0
    for k=1 to j-1
      sum = sum + L(i,k)*U(k,j)
    end for
    L(i,j) = (A(i,j)-sum)/U(j,j)
  end for
  L(i,i) = 1
  for j = i to n
    sum = 0
    for k = 1 to i-1
      sum = sum + L(i,k)*U(k,j)
    end for
    U(i,j) = A(i,j) - sum
  end for
end for
```

Complexity

Order n^3 products

Cholesky factorization for an Hermitian matrix

Principle

$A = C\,^tC$

Cholesky algorithm

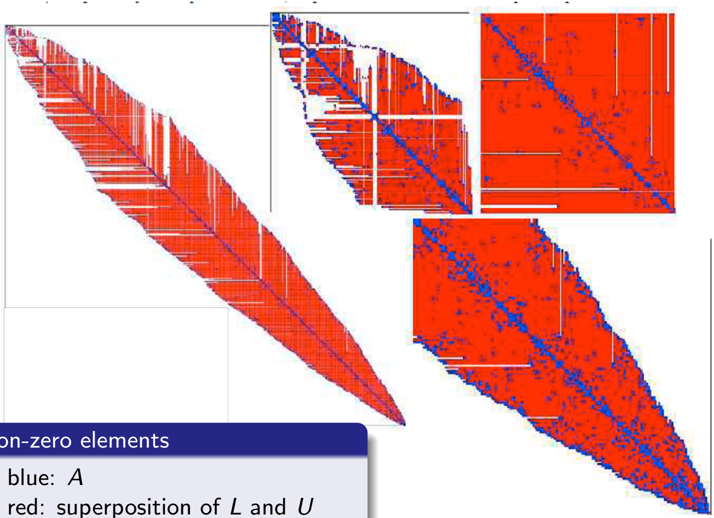
$$C_{ii} = \sqrt{A_{ii} - \sum_{k=1}^{i-1} C_{ik} C_{ik}} \qquad C_{ij} = \frac{A_{ij} - \sum_{k=1}^{j-1} C_{ik} C_{jk}}{C_{jj}},\ j \neq i$$

```
C(1,1) = sqrt(A(1,1))
for i = 2 to n
  for j = 1 to i-1
    sum = 0
    for k = 1 to j-1
      sum = sum + C(i,k)*C(j,k)
    end for
    C(i,j) = (A(i,j)-sum)/C(j,j)
  end for
  sum=0
  for k = 1 to i-1
    sum = sum + C(i,k)*C(i,k)
  end for
  C(i,i) = sqrt(A(i,i) - sum)
end for
```

Complexity

Order n^3 products

LU factorization – profiles



Non-zero elements

In blue: A
In red: superposition of L and U

The interior of the profile is filled!

QR factorization – principle

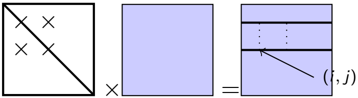
Principle

$A = QR$, where Q orthogonal and R right (upper) triangular.

$G_m \dots G_2 G_1 A = R$,

$$A = \underbrace{{}^tG_1\,{}^tG_2 \dots {}^tG_m}_{Q} R$$

Let $$G_{ij}(c,s) = \begin{pmatrix} 1 & & & & & & 0 \\ & 1 & & & & & \\ & & c & & s & & \\ & & & 1 & & & \\ & & -s & & c & & \\ & & & & & 1 & \\ 0 & & & & & & 1 \end{pmatrix}$$ with $c^2 + s^2 = 1$.



$$(GA)_{ij} = -sA_{jj} + cA_{ij}$$

$$c = \frac{A_{jj}}{\sqrt{A_{ij}^2 + A_{jj}^2}} \quad s = \frac{A_{ij}}{\sqrt{A_{ij}^2 + A_{jj}^2}}$$

QR factorization – algorithm

Algorithm

```
R = A
Q = Id   // size of A
for i = 2 to n
  for j = 1 to i-1
    root = sqrt(R(i,j)*R(i,j)+R(j,j)*R(j,j))
    if root != 0
      c = R(j,j)/root
      s = R(i,j)/root
    else
      c = 1
      s = 0
    end if
    Construct Gji
    R = Gji*R                // matrix product
    Q = Q*transpose(Gji)     // matrix product
  end for
end for
```

Complexity

Order n^3 products

QR factorization – Python example

$$A = \begin{pmatrix} 3 & 2 & 1 & 0 & 0 \\ 4 & 3 & 2 & 1 & 0 \\ 5 & 4 & 3 & 2 & 1 \\ 6 & 5 & 4 & 3 & 2 \\ 7 & 6 & 5 & 4 & 3 \end{pmatrix}$$

$$R = \begin{pmatrix} 11.619 & 9.467 & 7.316 & 5.164 & 3.271 \\ 3.437\ 10^{-16} & 6.086\ 10^{-01} & 1.217 & 1.826 & 1.704 \\ 4.476\ 10^{-17} & 1.989\ 10^{-18} & 2.324\ 10^{-15} & 3.768\ 10^{-15} & -3.775\ 10^{-01} \\ -6.488\ 10^{-16} & 1.082\ 10^{-17} & 0.000 & 1.618\ 10^{-16} & -6.764\ 10^{-02} \\ -6.671\ 10^{-16} & -2.548\ 10^{-17} & 0.000 & -3.082\ 10^{-33} & -5.029\ 10^{-01} \end{pmatrix}$$

$$Q = \begin{pmatrix} 0.2582 & -0.7303 & -0.3775 & -0.0676 & -0.5029 \\ 0.3443 & -0.4260 & -0.0062 & -0.1589 & 0.821 \\ 0.4303 & -0.1217 & 0.5407 & 0.7050 & -0.1030 \\ 0.5164 & 0.1826 & 0.4472 & -0.6627 & -0.2466 \\ 0.6025 & 0.4869 & -0.6042 & 0.1842 & 0.0311 \end{pmatrix}$$

Overview

Power iteration algorithm – Python experiment

$$A = \begin{pmatrix} 10 & 0 \\ -9 & 1 \end{pmatrix}$$

Eigenvalues and eigenvectors:

$$\lambda_1 = 1, \lambda_2 = 10, \vec{v}_1 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \vec{v}_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}.$$

Construct the series

$$\vec{x}^k = A\vec{x}^{k-1}$$

$$\vec{x}^0 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}, \vec{x}^1 = \begin{pmatrix} 20 \\ -17 \end{pmatrix}, \vec{x}^2 = \begin{pmatrix} 200 \\ -197 \end{pmatrix}, \vec{x}^3 = \begin{pmatrix} 2000 \\ -1997 \end{pmatrix} \dots$$

$\vec{x}$ tends to the direction of the eigenvector associated to the higher modulus eigenvalue.

"$\vec{x}^k / \vec{x}^{k-1}$" tends to the higher modulus eigenvalue.

Power iteration algorithm – Algorithm

Computation of the eigenvalue with higher modulus.
A may be diagonalizable or not, the dominant eigenvalue can be unique or not.

Algorithm

```
choose q(0)
for k = 1 to convergence
  x(k) = A * q(k-1)
  q(k) = x(k) / norm(x(k))
end for
lambdamax = x(k)(j)/q(k-1)(j)
```

Attention: good choice of component j.

Power iteration algorithm – Python example

$$A = \begin{pmatrix} 10 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

Rotations:

$$R_1 = \begin{pmatrix} \cos(1) & 0 & \sin(1) \\ 0 & 1 & 0 \\ -\sin(1) & 0 & \cos(1) \end{pmatrix}, R_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(2) & \sin(2) \\ 0 & -\sin(2) & \cos(2) \end{pmatrix}$$

$$B = R_2 R_1 A^t R_1{}^t R_2 = \begin{pmatrix} 4.33541265 & -3.30728724 & 1.51360499 \\ -3.30728724 & 7.20313893 & -1.00828318 \\ 1.51360499 & -1.00828318 & 5.46144841 \end{pmatrix}$$

Eigenvalues and eigenvectors:

$$\lambda_1 = 2, \lambda_2 = 5, \lambda_3 = 10,$$

$$\vec{v}_1 = \begin{pmatrix} -0.8415 \\ -0.4913 \\ 0.2248 \end{pmatrix}, \vec{v}_2 = \begin{pmatrix} 1.365\ 10^{-16} \\ 0.4161 \\ 0.9093 \end{pmatrix}, \vec{v}_3 = \begin{pmatrix} -0.5403 \\ 0.7651 \\ -0.3502 \end{pmatrix}.$$

Power iteration algorithm – Remarks

1. Convergence results depend on the fact that

Power iteration algorithm – Remarks

1. Convergence results depend on the fact that
   - the matrix is diagonalizable or not

Power iteration algorithm – Remarks

1. Convergence results depend on the fact that
    - the matrix is diagonalizable or not
    - the dominant eigenvalue is multiple or not

Power iteration algorithm – Remarks

1. Convergence results depend on the fact that
   - the matrix is diagonalizable or not
   - the dominant eigenvalue is multiple or not
2. The choice of the norm is not explicit: usually max norm or euclidian norm

Power iteration algorithm – Remarks

1. Convergence results depend on the fact that
   - the matrix is diagonalizable or not
   - the dominant eigenvalue is multiple or not
2. The choice of the norm is not explicit: usually max norm or euclidian norm
3. $\vec{q}_0$ should not be orthogonal to the eigen-subspace associated to the dominant eigenvalue.

Inverse iteration algorithm – Algorithm

Computation of the eigenvalue with smallest modulus.
A may be diagonalizable or not, the dominant eigenvalue can be unique or not.
Based on the fact that

$$\lambda_{\min}(A) = \left(\lambda_{\max}(A^{-1})\right)^{-1}.$$

Algorithm

```
choose q(0)
for k = 1 to convergence
  solve A * x(k) = q(k-1)
  q(k) = x(k) / norm(x(k))
end for
lambdamin = q(k-1)(j) / x(k)(j)
```

Inverse iteration algorithm – Python example

$$A = \begin{pmatrix} 10 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 2 \end{pmatrix}$$

Rotations:

$$R_1 = \begin{pmatrix} \cos(1) & 0 & \sin(1) \\ 0 & 1 & 0 \\ -\sin(1) & 0 & \cos(1) \end{pmatrix}, R_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(2) & \sin(2) \\ 0 & -\sin(2) & \cos(2) \end{pmatrix}$$

$$B = R_2 R_1 A^t R_1{}^t R_2 = \begin{pmatrix} 4.33541265 & -3.30728724 & 1.51360499 \\ -3.30728724 & 7.20313893 & -1.00828318 \\ 1.51360499 & -1.00828318 & 5.46144841 \end{pmatrix}$$

Eigenvalues and eigenvectors:

$$\lambda_1 = 2, \lambda_2 = 5, \lambda_3 = 10,$$

$$\vec{v}_1 = \begin{pmatrix} -0.8415 \\ -0.4913 \\ 0.2248 \end{pmatrix}, \vec{v}_2 = \begin{pmatrix} 1.365\ 10^{-16} \\ 0.4161 \\ 0.9093 \end{pmatrix}, \vec{v}_3 = \begin{pmatrix} -0.5403 \\ 0.7651 \\ -0.3502 \end{pmatrix}.$$

Generalized inverse iteration algorithm – Algorithm

Computation of the closest eigenvalue to a given μ.
The eigenvalues of $A - \mu I$ are the $\lambda_i - \mu$,
where λ_i are the eigenvalues of A.
$\Rightarrow$ apply the inverse iteration algorithm to $A - \mu I$.

Algorithm

```
choose q(0)
for k = 1 to convergence
   solve (A-mu*I) * x(k) = q(k-1)
   q(k) = x(k) / norm(x(k))
end for
lambda = q(k-1)(j) / x(k)(j) + mu
```

Generalized inverse iteration algorithm – Python example

$$A = \begin{pmatrix} 10 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 2 \end{pmatrix} \qquad \mu = 4.$$

Rotations:

$$R_1 = \begin{pmatrix} \cos(1) & 0 & \sin(1) \\ 0 & 1 & 0 \\ -\sin(1) & 0 & \cos(1) \end{pmatrix}, R_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(2) & \sin(2) \\ 0 & -\sin(2) & \cos(2) \end{pmatrix}$$

$$B = R_2 R_1 A^t R_1{}^t R_2 = \begin{pmatrix} 4.33541265 & -3.30728724 & 1.51360499 \\ -3.30728724 & 7.20313893 & -1.00828318 \\ 1.51360499 & -1.00828318 & 5.46144841 \end{pmatrix}$$

Eigenvalues and eigenvectors:

$$\lambda_1 = 2, \lambda_2 = 5, \lambda_3 = 10,$$

$$\vec{v}_1 = \begin{pmatrix} -0.8415 \\ -0.4913 \\ 0.2248 \end{pmatrix}, \vec{v}_2 = \begin{pmatrix} 1.365\ 10^{-16} \\ 0.4161 \\ 0.9093 \end{pmatrix}, \vec{v}_3 = \begin{pmatrix} -0.5403 \\ 0.7651 \\ -0.3502 \end{pmatrix}.$$

Deflation – Algorithm and Python example

Computation of all the eigenvalues in modulus decreasing order.

When an eigenelement (λ, q) is found, it is removed from further computation by replacing $A \leftarrow A - \lambda \vec{q}^t \vec{q}$.

Algorithm

```
  for i = 1 to n
    choose q(0)
    for k = 1 to convergence
      x(k) = A * q(k-1)
      q(k) = x(k) / norm(x(k))
    end for
    lambda = x(k)(j) / q(k-1)(j)
    A = A - lambda * q * transpose(q)
// eliminates direction q
  end for
```

Galerkin method – Algorithm

Let H be a subspace of dimension m, generated by the orthonormal basis $(\vec{q}_1, \dots, \vec{q}_m)$.
Construct the rectangular matrix $Q = (\vec{q}_1, \dots, \vec{q}_m)$.
Remark: $Q^* Q = Id_m$

Goal

Look for eigenvectors in H.

If $\vec{u} \in H$, $\vec{u} = \sum_{i=1}^{m} \alpha_i \vec{q}_i$ (unique).
$\vec{u} = Q\vec{U}$, where $\vec{U} = {}^t(\alpha_1, \dots, \alpha_m)$.

$A\vec{u} = \lambda\vec{u} \Leftrightarrow AQ\vec{U} = \lambda Q\vec{U}$.
Project on H: $Q^* AQ\vec{U} = \lambda Q^* Q\vec{U} = \lambda\vec{U}$.

$\Rightarrow$ We look for eigenelements of $B = Q^* AQ$.

Vocabulary:

- $\{\lambda_i, \vec{u}_i\}$ are the Ritz elements,
- B is the Rayleigh matrix.

Jacobi method – Algorithm

Goal

Diagonalize the (real symmetric) matrix.

Until a "reasonably diagonal" matrix is obtained:

Jacobi method – Algorithm

Goal

Diagonalize the (real symmetric) matrix.

Until a "reasonably diagonal" matrix is obtained:

- Choose the largest off-diagonal element (largest modulus)

Jacobi method – Algorithm

Goal

Diagonalize the (real symmetric) matrix.

Until a "reasonably diagonal" matrix is obtained:

- Choose the largest off-diagonal element (largest modulus)
- Construct a rotation matrix that annihilates this term

Jacobi method – Algorithm

Goal

Diagonalize the (real symmetric) matrix.

Until a "reasonably diagonal" matrix is obtained:

- Choose the largest off-diagonal element (largest modulus)
- Construct a rotation matrix that annihilates this term

In the end, the eigenvalues are the diagonal elements.

QR method – Algorithm

Algorithm

```
A(1) = A
for k = 1 to convergence
   [Q(k),R(k)] = QR_factor(A(k))
   A(k+1) = R(k)*Q(k)
end for
```

The eigenvalues are the diagonal elements of the last matrix A_{k+1}.

Properties

- $A_{k+1} = R_k Q_k = Q_k^* Q_k R_k Q_k = Q_k^* A_k Q_k$
  $\Rightarrow$ A_{k+1} and A_k are similar.
- If A_k is tridiagonal or Hessenberg, A_{k+1} also is
  $\Rightarrow$ First restrict to this case keeping similar matrices.

QR method – Convergence and Python example

Theorem

Let V^* be the matrix of left eigenvectors of A ($A^*\vec{u}^* = \lambda\vec{u}^*$).
If

- the principal minors of V are non-zero.
- the eigen-values of A are such that $|\lambda_1| > \cdots > |\lambda_n|$.

Then the QR method converges A_{k+1} tends to an upper triangular form and $(A_k)_{ii}$ tends to λ_i.

Eigenvalues – Summary

We want to know all the eigenvalues

Eigenvalues – Summary

We want to know all the eigenvalues

- QR method — better than Jacobi
  Preprocessing: find a similar tridiagonal or Heisenberg matrix (Householder or Givens algorithm).

Eigenvalues – Summary

We want to know all the eigenvalues

- QR method — better than Jacobi
  Preprocessing: find a similar tridiagonal or Heisenberg matrix (Householder or Givens algorithm).

We only want one eigenvector whose eigenvalue is known (or an approximation)

Eigenvalues – Summary

We want to know all the eigenvalues

- QR method — better than Jacobi
  Preprocessing: find a similar tridiagonal or Heisenberg matrix (Householder or Givens algorithm).

We only want one eigenvector whose eigenvalue is known (or an approximation)

- Power iteration algorithm and variants. . .

Eigenvalues – Summary

We want to know all the eigenvalues

- QR method — better than Jacobi
  Preprocessing: find a similar tridiagonal or Heisenberg matrix (Householder or Givens algorithm).

We only want one eigenvector whose eigenvalue is known (or an approximation)

- Power iteration algorithm and variants. . .

We only want a sub-set of eigenelements

Eigenvalues – Summary

We want to know all the eigenvalues

- QR method — better than Jacobi
  Preprocessing: find a similar tridiagonal or Heisenberg matrix (Householder or Givens algorithm).

We only want one eigenvector whose eigenvalue is known (or an approximation)

- Power iteration algorithm and variants. . .

We only want a sub-set of eigenelements

- We know the eigenvalues and look for eigenvectors: deflation and variants

Eigenvalues – Summary

We want to know all the eigenvalues

- QR method — better than Jacobi
  Preprocessing: find a similar tridiagonal or Heisenberg matrix (Householder or Givens algorithm).

We only want one eigenvector whose eigenvalue is known (or an approximation)

- Power iteration algorithm and variants. . .

We only want a sub-set of eigenelements

- We know the eigenvalues and look for eigenvectors: deflation and variants
- We know the subspace for eigenvectors: Galerkin and variants

Overview

Principles

$$A\vec{x} = \vec{b}$$

Elimination methods

The solution to the system remains unchanged if

Principles

$$A\vec{x} = \vec{b}$$

Elimination methods

The solution to the system remains unchanged if

- lines are permuted,

Principles

$$A\vec{x} = \vec{b}$$

Elimination methods

The solution to the system remains unchanged if

- lines are permuted,
- line i is replaced by a linear combination $\ell_i \leftarrow \sum_{k=1}^{n} \mu_k \ell_k$, with $\mu_i \neq 0$.

Principles

$$A\vec{x} = \vec{b}$$

Elimination methods

The solution to the system remains unchanged if

- lines are permuted,
- line i is replaced by a linear combination $\ell_i \leftarrow \sum_{k=1}^{n} \mu_k \ell_k$, with $\mu_i \neq 0$.

Factorisation methods

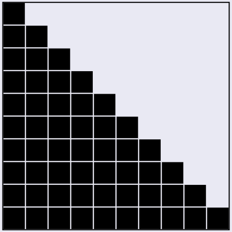
$A = LU$
$LU\vec{x} = \vec{b}$
We solve two triangular systems
$L\vec{y} = \vec{b}$
$U\vec{x} = \vec{y}$.

Lower triangular matrix

$$x_i = \frac{b_i - \sum_{k=1}^{i-1} A_{ik} x_k}{A_{ii}}$$

Algorithm

```
if A(1,1)==0 then stop
x(1) = b(1)/A(1,1)
for i = 2 to n
  if A(i,i)==0 then stop
  ax = 0
  for k = 1 to i-1
    ax = ax + A(i,k)*x(k)
  end for
  x(i) = (b(i)-ax)/A(i,i)
end for
```



Complexity

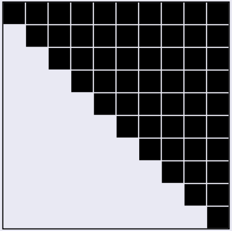
Order $n^2/2$ products.

Upper triangular matrix

$$x_i = \frac{b_i - \sum_{k=i+1}^{n} A_{ik} x_k}{A_{ii}}$$

Algorithm

```
if A(n,n)==0 then stop
x(n) = b(n)/A(n,n)
for i = n-1 to 1
  if A(i,i)==0 then stop
  ax = 0
  for k = i+1 to n
    ax = ax + A(i,k)*x(k)
  end for
  x(i) = (b(i)-ax)/A(i,i)
end for
```



Complexity

Order $n^2/2$ products.

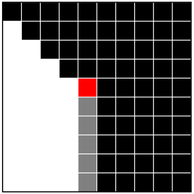
Gauss elimination – Principle

Aim

Transform A to upper triangular matrix.

At rank $p-1$:

$$A_{ij} = 0 \qquad \text{if } i > j,\ j < p.$$



$$\ell_i \leftarrow \ell_i - A_{ip}\frac{\ell_p}{A_{pp}}$$

```
for p = 1 to n
  pivot = A(p,p)
  if pivot == 0 stop
  line(p) = line(p)/pivot
  for i = p+1 to n
    Aip = A(i,p)
    line(i) = line(i) - Aip * line(p)
  end for
end for
x = solve(A,b) // upper triangular
```

Complexity

Still order n^3 products.

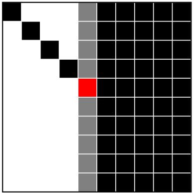
Gauss–Jordan elimination – Principle

Aim

Transform A to identity.

At rank $p-1$:

$$A_{ii} = 1 \qquad \text{if } i < p,$$
$$A_{ij} = 0 \qquad \text{if } i \neq j,\ j < p.$$



$$\ell_i \leftarrow \ell_i - A_{ip}\frac{\ell_p}{A_{pp}}$$

```
for p = 1 to n
  pivot = A(p,p)
  if pivot == 0 stop
  line(p) = line(p)/pivot
  for i = 1 to n, i!=p
    Aip = A(i,p)
    line(i) = line(i) - Aip * line(p)
  end for
end for
x = b
```

Attention

- take into account le right-hand side in the "line".
- what if $A_{pp} = 0$?

Gauss–Jordan elimination – Algorithm


```
// unknown entries numbering
for i = 1 to n
  num(i) = i
end for

for p=1 to n
  // maximal pivot
  pmax = abs(A(p,p))
  imax = p
  jmax = p
  for i = p to n
    for j = p to n
      if abs(A(i,j)) > pmax then
        pmax = abs(A(i,j))
        imax = i
        jmax = j
      end if
    end for
  end for
  // line permutation
  for j = p to n
    permute(A(p,j),A(imax,j))
  end for
  permute(b(p),b(imax))
  // column permutation
  for i = p to n
    permute(A(i,p),A(i,jmax))
  end for
  permute(num(p),num(jmax))
  pivot = A(p,p)
  if pivot == 0 stop, rank(A) = p-1
  for j = p to n
    A(p,j) = A(p,j)/pivot
  end for
  b(p) = b(p)/pivot
  for i = 1 to n, i!=p
    Aip = A(i,p)
    for j = p to n
      A(i,j) = A(i,j) - Aip * A(p,j)
    end for
    b(i) = b(i) - Aip*b(p)
  end for
end for  // loop on p

for i = 1 to n
  x(num(i)) = b(i)
end for
```

Complexity

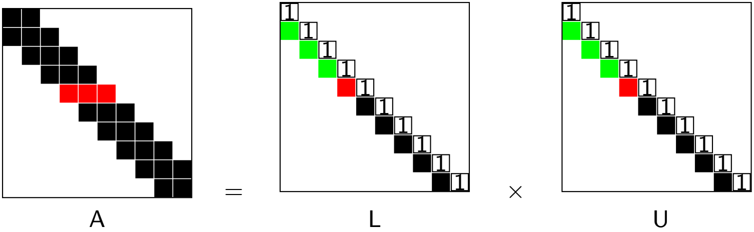
Order n^3 products.

Remark

Also computes the rank of the matrix.

Factorization methods — Thomas algorithm — principle

LU decomposition for tridiagonal matrices.



We suppose that L_{ij} and U_{ij} are known for $i < p$. Then

$$\begin{aligned}
A_{p,p-1} &= L_{p,p-1} U_{p-1,p-1}, \\
A_{p,p} &= L_{p,p-1} U_{p-1,p} + U_{p,p}, \\
A_{p,p+1} &= U_{p,p+1}.
\end{aligned}$$

$\Rightarrow$

$$\begin{aligned}
L_{p,p-1} &= A_{p,p-1} / U_{p-1,p-1}, \\
U_{p,p} &= A_{p,p} - L_{p,p-1} U_{p-1,p} = A_{p,p} - A_{p,p-1} U_{p-1,p} / U_{p-1,p-1}, \\
U_{p,p+1} &= A_{p,p+1}.
\end{aligned}$$

Factorization methods — Thomas algorithm — algorithm

Algorithmm

```
// factorization
U(1,1) = A(1,1)
U(1,2) = A(1,2)
for i = 2 to n
  if U(i-1,i-1) = 0 then stop
  L(i,i-1) = A(i,i-1)/U(i-1,i-1)
  U(i,i) = A(i,i) - L(i,i-1)*U(i-1,i)
  U(i,i+1) = A(i,i+1)
end for
// construction of the solution
y = solve(L,b) // lower triangular
x = solve(U,y) // upper triangular
```

Complexity

Order $5n$ products.

Factorization methods — general matrices

For general matrices:

- Factorize the matrix

Factorization methods — general matrices

For general matrices:

- Factorize the matrix
  - LU algorithm

Factorization methods — general matrices

For general matrices:

- Factorize the matrix
  - LU algorithm
  - Choleski algorithm

Factorization methods — general matrices

For general matrices:

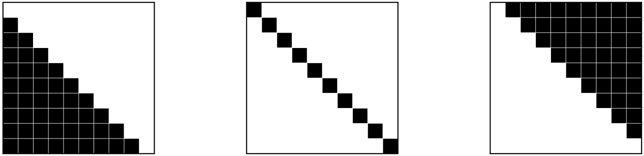
- Factorize the matrix
  - LU algorithm
  - Choleski algorithm
- Solve upper triangular system

Factorization methods — general matrices

For general matrices:

- Factorize the matrix
  - LU algorithm
  - Choleski algorithm
- Solve upper triangular system
- Solve lower triangular system.

Iterative methods — Principle

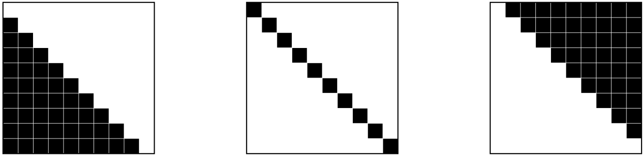


$A = E + D + F$

To solve $A\vec{x} = \vec{b}$, write $A = M - N$
and iterate $M\vec{x}^{k+1} - N\vec{x}^{k} = \vec{b}$, i.e. $\vec{x}^{k+1} = M^{-1}N\vec{x}^{k} + M^{-1}\vec{b}$.

Attention

Iterative methods — Principle



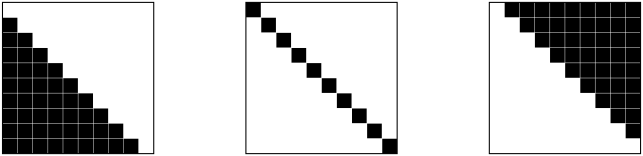
$A = E + D + F$

To solve $A\vec{x} = \vec{b}$, write $A = M - N$
and iterate $M\vec{x}^{k+1} - N\vec{x}^{k} = \vec{b}$, i.e. $\vec{x}^{k+1} = M^{-1}N\vec{x}^{k} + M^{-1}\vec{b}$.

Attention

- M should be easy to invert.

Iterative methods — Principle



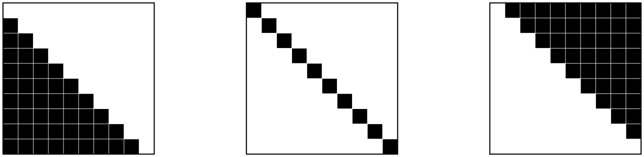
A= E + D + F

To solve $A\vec{x} = \vec{b}$, write $A = M - N$
and iterate $M\vec{x}^{k+1} - N\vec{x}^{k} = \vec{b}$, i.e. $\vec{x}^{k+1} = M^{-1}N\vec{x}^{k} + M^{-1}\vec{b}$.

Attention

- M should be easy to invert.
- $M^{-1}N$ should lead to a stable algorithm.

Iterative methods — Principle



A= E + D + F

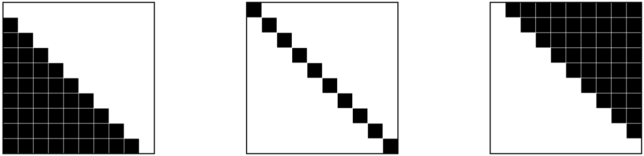
To solve $A\vec{x} = \vec{b}$, write $A = M - N$
and iterate $M\vec{x}^{k+1} - N\vec{x}^{k} = \vec{b}$, i.e. $\vec{x}^{k+1} = M^{-1}N\vec{x}^{k} + M^{-1}\vec{b}$.

Attention

- M should be easy to invert.
- $M^{-1}N$ should lead to a stable algorithm.

Jacobi $M = D$, $N = -(E + F)$,

Iterative methods — Principle



A= E + D + F

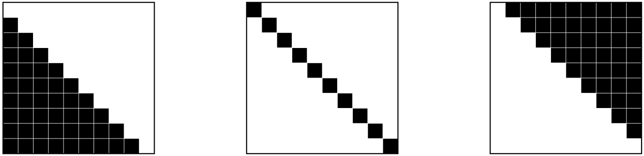
To solve $A\vec{x} = \vec{b}$, write $A = M - N$
and iterate $M\vec{x}^{k+1} - N\vec{x}^{k} = \vec{b}$, i.e. $\vec{x}^{k+1} = M^{-1}N\vec{x}^{k} + M^{-1}\vec{b}$.

Attention

- M should be easy to invert.
- $M^{-1}N$ should lead to a stable algorithm.

Jacobi $M = D$, $N = -(E + F)$,
Gauss–Seidel $M = D + E$, $N = -F$,

Iterative methods — Principle



A= E + D + F

To solve $A\vec{x} = \vec{b}$, write $A = M - N$
and iterate $M\vec{x}^{k+1} - N\vec{x}^{k} = \vec{b}$, i.e. $\vec{x}^{k+1} = M^{-1}N\vec{x}^{k} + M^{-1}\vec{b}$.

Attention

- M should be easy to invert.
- $M^{-1}N$ should lead to a stable algorithm.

Jacobi $M = D,\ N = -(E+F)$,

Gauss–Seidel $M = D + E,\ N = -F$,

Successive Over Relaxation $M = \frac{D}{\omega} + E,\ N = \left(\frac{1}{\omega} - 1\right) D - F$.

Jacobi method

Algorithm

```
choose x(k=0)
for k = 0 to convergence
  for i = 1 to n
    rhs = b(i)
    for j = 1 to n, j!=i
      rhs = rhs - A(i,j)*x(j,k)
    end for
    x(i,k+1) = rhs / A(i,i)
  end for
  test = norm(x(k+1)-x(k))<epsilon
end for (while not test)
```

$$
\begin{aligned}
x_i^{k+1} &= \frac{1}{A_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} A_{ij} x_j^k\right)\\
\vec{x}^{k+1} &= D^{-1}(\vec{b} - (E+F)\vec{x}^k)\\
&= D^{-1}(\vec{b} + (D-A)\vec{x}^k)\\
&= D^{-1}\vec{b} + (I - D^{-1}A)\vec{x}^k.
\end{aligned}
$$

Remarks

- simple,
- two copies of the variable $\vec{x}^{k+1}$ and $\vec{x}^k$,
- insensible to permutations,
- converges if the diagonal is strictly dominant.

Gauss–Seidel method

$$x_i^{k+1} = \frac{1}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij} x_j^k \right)$$

Algorithm

```
choose x(k=0)
for k = 0 to convergence
  for i = 1 to n
    rhs = b(i)
    for j = 1 to i-1
      rhs = rhs - A(i,j)*x(j,k+1)
    end for
    for j = i+1 to n
      rhs = rhs - A(i,j)*x(j,k)
    end for
    x(i,k+1) = rhs / A(i,i)
  end for
  test = norm(x(k+1)-x(k))<epsilon
end for (while not test)
```

Remarks

- still simple,
- one copy of the variable $\vec{x}$,
- sensible to permutations,
- often converges better than Jacobi.

SOR method

$$x_i^{k+1} = \frac{\omega}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij} x_j^{k} \right) + (1-\omega) x_i^k$$

SOR method

$$x_i^{k+1} = \frac{\omega}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij} x_j^k \right) + (1-\omega) x_i^k$$

$$\vec{x}^{k+1} = \left(\frac{D}{\omega} + E \right)^{-1} \vec{b} + \left(\frac{D}{\omega} + E \right)^{-1} \left[\left(\frac{1}{\omega} - 1 \right) D - F \right] \vec{x}^k$$

SOR method

$$x_i^{k+1} = \frac{\omega}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij} x_j^k \right) + (1-\omega) x_i^k$$

$$\vec{x}^{k+1} = \left(\frac{D}{\omega} + E \right)^{-1} \vec{b} + \left(\frac{D}{\omega} + E \right)^{-1} \left[\left(\frac{1}{\omega} - 1 \right) D - F \right] \vec{x}^k$$

Remarks

SOR method

$$x_i^{k+1} = \frac{\omega}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij} x_j^k \right) + (1-\omega) x_i^k$$

$$\vec{x}^{k+1} = \left(\frac{D}{\omega} + E \right)^{-1} \vec{b} + \left(\frac{D}{\omega} + E \right)^{-1} \left[\left(\frac{1}{\omega} - 1 \right) D - F \right] \vec{x}^k$$

Remarks

- still simple,

SOR method

$$x_i^{k+1} = \frac{\omega}{A_{ii}}\left(b_i - \sum_{j=1}^{i-1} A_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij}x_j^{k}\right) + (1-\omega)x_i^k$$

$$\vec{x}^{k+1} = \left(\frac{D}{\omega}+E\right)^{-1}\vec{b} + \left(\frac{D}{\omega}+E\right)^{-1}\left[\left(\frac{1}{\omega}-1\right)D - F\right]\vec{x}^k$$

Remarks

- still simple,
- one copy of the variable $\vec{x}$,

SOR method

$$x_i^{k+1} = \frac{\omega}{A_{ii}} \left(b_i - \sum_{j=1}^{i-1} A_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} A_{ij} x_j^k \right) + (1-\omega) x_i^k$$

$$\vec{x}^{k+1} = \left(\frac{D}{\omega} + E \right)^{-1} \vec{b} + \left(\frac{D}{\omega} + E \right)^{-1} \left[\left(\frac{1}{\omega} - 1 \right) D - F \right] \vec{x}^k$$

Remarks

- still simple,
- one copy of the variable $\vec{x}$,
- Necessary condition for convergence: $0 < \omega < 2$.

Descent method — general principle

For A symmetric definite positive!!

Principle

Construct a series of approximations of the solution to the system

$$\vec{x}^{k+1} = \vec{x}^k + \alpha^k \vec{p}^k,$$

where $\vec{p}^k$ descent direction and α^k to be determined.

The solution $\underline{\vec{x}}$ minimizes the functional $J(\vec{x}) = {}^t\vec{x}A\vec{x} - 2{}^t\vec{b}\vec{x}$.

$$\begin{aligned}
\frac{\partial J}{\partial x_i}(\vec{x}) &= \frac{\partial}{\partial x_i}\left(\sum_{j,k} x_j A_{jk} x_k - 2\sum_j b_j x_j\right) \\
&= \sum_k A_{ik} x_k + \sum_j x_j A_{ji} - 2b_i \\
&= 2\left(A\vec{x} - \vec{b}\right)_i, \\
\frac{\partial J}{\partial x_i}(\underline{\vec{x}}) &= 0.
\end{aligned}$$

Descent method — determining α_k

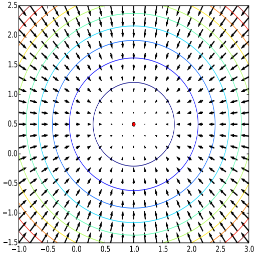
$\underline{\vec{x}}$ also minimizes the functional $E(\vec{x}) = {}^t(\vec{x} - \underline{\vec{x}})A(\vec{x} - \underline{\vec{x}})$, and $E(\underline{\vec{x}}) = 0$.
For a given $\vec{p}^k$, which α minimizes $E(\vec{x}^{k+1})$?

$$
\begin{aligned}
E(\vec{x}^k + \alpha\vec{p}^k) &= {}^t(\vec{x}^k + \alpha\vec{p}^k - \underline{\vec{x}})A(\vec{x}^k + \alpha\vec{p}^k - \underline{\vec{x}}), \\
\frac{\partial}{\partial\alpha}E(\vec{x}^k + \alpha\vec{p}^k) &= {}^t\vec{p}^k A(\vec{x}^k + \alpha\vec{p}^k - \underline{\vec{x}}) + {}^t(\vec{x}^k + \alpha\vec{p}^k - \underline{\vec{x}})A\vec{p}^k \\
&= 2\,{}^t(\vec{x}^k + \alpha\vec{p}^k - \underline{\vec{x}})A\vec{p}^k.
\end{aligned}
$$

$$
\begin{aligned}
&{}^t(\vec{x}^k + \alpha^k\vec{p}^k - \underline{\vec{x}})A\vec{p}^k = 0 \\
&{}^t\vec{x}_k A\vec{p}_k + \alpha_k\,{}^t\vec{p}_k A\vec{p}^k - {}^t\underline{\vec{x}}A\vec{p}^k = 0 \\
&{}^t\vec{p}^k A\vec{x}^k + \alpha^k\,{}^t\vec{p}_k A\vec{p}^k - {}^t\vec{p}^k A\underline{\vec{x}} = 0.
\end{aligned}
$$

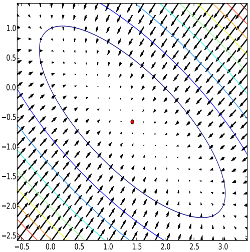
$$
\alpha^k = \frac{{}^t\vec{p}^k A\vec{x}^k - {}^t\vec{p}^k A\underline{\vec{x}}}{{}^t\vec{p}^k A\vec{p}^k}
$$

Descent method — functional profiles (good cases)

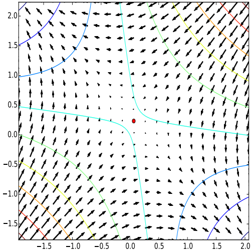


$$A = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}, \quad \vec{b} = A \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$
$$\mathrm{Cond}(A) = 1$$

$$A = \begin{pmatrix} 2 & 1.5 \\ 1.5 & 2 \end{pmatrix}, \quad \vec{b} = A \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$
$$\mathrm{Cond}(A) = 7$$



Descent method — functional profiles (bad cases)

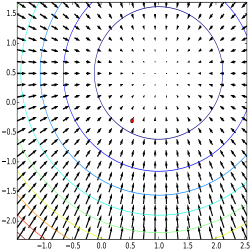


Nonpositive case

$$A = \begin{pmatrix} 2 & 8 \\ 8 & 2 \end{pmatrix}, \ \vec{b} = A \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

Nonsymmetric case

$$A = \begin{pmatrix} 2 & -3 \\ 3 & 2 \end{pmatrix}, \ \vec{b} = A \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$



Descent method — optimal parameter (principle)

Principle

Descent method — optimal parameter (principle)

Principle

- Choose $\vec{p}^k = \vec{r}^k \equiv \vec{b} - A\vec{x}^k$.

Descent method — optimal parameter (principle)

Principle

- Choose $\vec{p}^k = \vec{r}^k \equiv \vec{b} - A\vec{x}^k$.
- Choose α^k is such that $\vec{r}^{k+1}$ is orthogonal to $\vec{p}^k$.

Descent method — optimal parameter (principle)

Principle

- Choose $\vec{p}^k = \vec{r}^k \equiv \vec{b} - A\vec{x}^k$.
- Choose α^k is such that $\vec{r}^{k+1}$ is orthogonal to $\vec{p}^k$.

$$
\begin{aligned}
\vec{r}^{k+1} &= \vec{b} - A\vec{x}^{k+1} = \vec{b} - A(\vec{x}^k + \alpha\vec{p}^k) = \vec{r}^k - \alpha^k A\vec{p}^k, \\
0 &= {}^t\vec{p}^k\vec{r}^{k+1} = {}^t\vec{p}^k\vec{r}^k - \alpha^k\,{}^t\vec{p}^k A\vec{p}^k.
\end{aligned}
$$

Descent method — optimal parameter (principle)

Principle

- Choose $\vec{p}^k = \vec{r}^k \equiv \vec{b} - A\vec{x}^k$.
- Choose α^k is such that $\vec{r}^{k+1}$ is orthogonal to $\vec{p}^k$.

$$\begin{aligned} \vec{r}^{k+1} &= \vec{b} - A\vec{x}^{k+1} = \vec{b} - A(\vec{x}^k + \alpha\vec{p}^k) = \vec{r}^k - \alpha^k A\vec{p}^k, \\ 0 &= {}^t\vec{p}^k\vec{r}^{k+1} = {}^t\vec{p}^k\vec{r}^k - \alpha^k\, {}^t\vec{p}^k A\vec{p}^k. \end{aligned}$$

$$\alpha^k = \frac{{}^t\vec{p}^k\vec{r}^k}{{}^t\vec{p}^k A\vec{p}^k}.$$

Descent method — optimal parameter (principle)

Principle

- Choose $\vec{p}^k = \vec{r}^k \equiv \vec{b} - A\vec{x}^k$.
- Choose α^k is such that $\vec{r}^{k+1}$ is orthogonal to $\vec{p}^k$.

$$\begin{aligned} \vec{r}^{k+1} &= \vec{b} - A\vec{x}^{k+1} = \vec{b} - A(\vec{x}^k + \alpha\vec{p}^k) = \vec{r}^k - \alpha^k A\vec{p}^k, \\ 0 &= {}^t\vec{p}^k\vec{r}^{k+1} = {}^t\vec{p}^k\vec{r}^k - \alpha^k\,{}^t\vec{p}^k A\vec{p}^k. \end{aligned}$$

$$\alpha^k = \frac{{}^t\vec{p}^k\vec{r}^k}{{}^t\vec{p}^k A\vec{p}^k}.$$

$$E(\vec{x}^{k+1}) = (1-\gamma^k)E(\vec{x}^k)$$

$$\text{with } \gamma^k = \frac{({}^t\vec{p}^k\vec{r}^k)^2}{({}^t\vec{p}^k A\vec{p}^k)({}^t\vec{r}^k A^{-1}\vec{r}^k)} \geq \frac{1}{\mathrm{Cond}(A)}\,\frac{|{}^t\vec{p}^k\vec{r}^k|}{\|\vec{p}^k\|\|\vec{r}^k\|}.$$

Descent method — optimal parameter (algorithm)

Algorithm

```
choose x(k=1)
for k = 1 to convergence
  r(k) = b - A * x(k)
  p(k) = r(k)
  alpha(k) = r(k) . p(k) / p(k) . A * p(k)
  x(k+1) = x(k) + alpha(k) * p(k)
end for  //r(k) small
```

Descent method — conjugate gradient (principle)

Principle

- Choose $\vec{p}^k = \vec{r}^k + \beta^k \vec{p}^{k-1}$.
- Choose β^k to minimize the error, i.e. maximize the factor γ^k

Properties

- ${}^t\vec{r}^k \vec{p}^j = 0 \; \forall j < k$,
- $\text{Span}(\vec{r}^1, \vec{r}^2, \ldots, \vec{r}^k) = \text{Span}(\vec{r}^1, A\vec{r}^1, \ldots, A^{k-1}\vec{r}^1)$
- $\text{Span}(\vec{p}^1, \vec{p}^2, \ldots, \vec{p}^k) = \text{Span}(\vec{r}^1, A\vec{r}^1, \ldots, A^{k-1}\vec{r}^1)$
- ${}^t\vec{p}^k A\vec{p}^j = 0 \; \forall j < k$
- ${}^t\vec{r}^k A\vec{p}^j = 0 \; \forall j < k$
- The algorithm converges in at most n iterations.

Descent method — conjugate gradient (algorithm)

Algorithm

```
choose x(k=1)
p(1) = r(1) = b - A*x(1)
for k = 1 to convergence
  alpha(k) = r(k) . p(k) / p(k) . A * p(k)
  x(k+1) = x(k) + alpha(k) * p(k)
  r(k+1) = r(k) - alpha(k) * A * p(k)
  beta(k+1) = r(k+1) . r(k+1) / r(k) . r(k)
  p(k+1) = r(k+1) + beta(k+1) * p(k)
end for   //r(k) small
```

Descent method — GMRES

For generic matrices A

GMRES: General Minimal RESidual method

Descent method — GMRES

For generic matrices A

GMRES: General Minimal RESidual method

- Take a "fair" approximation $\vec{x}^k$ of the solution

Descent method — GMRES

For generic matrices A

GMRES: General Minimal RESidual method

- Take a "fair" approximation $\vec{x}^k$ of the solution
- Construct the m-dimensional set of free vectors

$$\{\vec{r}^k, A\vec{r}^k, \ldots, A^{m-1}\vec{r}^k\}$$

This spans the Krylov space H_m^k.

Descent method — GMRES

For generic matrices A

GMRES: General Minimal RESidual method

- Take a "fair" approximation $\vec{x}^k$ of the solution
- Construct the m-dimensional set of free vectors

$$\{\vec{r}^k, A\vec{r}^k, \ldots, A^{m-1}\vec{r}^k\}$$

  This spans the Krylov space H_m^k.
- Construct an orthonormal basis for H_m^k – e.g. via Gram-Schmidt

$$\{\vec{v}_1, \ldots, \vec{v}_m\}$$

Descent method — GMRES

For generic matrices A

GMRES: General Minimal RESidual method

- Take a "fair" approximation $\vec{x}^k$ of the solution
- Construct the m-dimensional set of free vectors

$$\{\vec{r}^k, A\vec{r}^k, \ldots, A^{m-1}\vec{r}^k\}$$

  This spans the Krylov space H_m^k.
- Construct an orthonormal basis for H_m^k – e.g. via Gram-Schmidt

$$\{\vec{v}_1, \ldots, \vec{v}_m\}$$

- Look for a new approximation $\vec{x}^{k+1} \in H_m^k$:

$$\vec{x}^{k+1} = \sum_{j=1}^{m} X_j \vec{v}_j = [V]\vec{X}$$

Descent method — GMRES

For generic matrices A

GMRES: General Minimal RESidual method

- Take a "fair" approximation $\vec{x}^k$ of the solution
- Construct the m-dimensional set of free vectors

$$\{\vec{r}^k, A\vec{r}^k, \ldots, A^{m-1}\vec{r}^k\}$$

  This spans the Krylov space H_m^k.
- Construct an orthonormal basis for H_m^k – e.g. via Gram-Schmidt

$$\{\vec{v}_1, \ldots, \vec{v}_m\}$$

- Look for a new approximation $\vec{x}^{k+1} \in H_m^k$:

$$\vec{x}^{k+1} = \sum_{j=1}^{m} X_j \vec{v}_j = [V]\vec{X}$$

- We obtain a system of n equations with m unknowns

$$A\vec{x}^{k+1} = A[V]\vec{X} = \vec{b}.$$

Descent method — GMRES (cont'd)

Descent method — GMRES (cont'd)

- Project on H_m^k

$$[{}^t V]A[V]\vec{X} = [{}^t V]\vec{b}.$$

Descent method — GMRES (cont'd)

- Project on H_m^k

$$[{}^tV]A[V]\vec{X} = [{}^tV]\vec{b}.$$

- Solve this system of m equations with m unknowns

Descent method — GMRES (cont'd)

- Project on H_m^k

$$[{}^t V]A[V]\vec{X} = [{}^t V]\vec{b}.$$

- Solve this system of m equations with m unknowns
- $\vec{x}^{k+1} = [V]\vec{X}.$

Descent method — GMRES (cont'd)

- Project on H_m^k

$$[{}^tV]A[V]\vec{X} = [{}^tV]\vec{b}.$$

- Solve this system of m equations with m unknowns
- $\vec{x}^{k+1} = [V]\vec{X}$.
- and so on. . .

Descent method — GMRES (cont'd)

- Project on H_m^k
$$[^tV]A[V]\vec{X} = [^tV]\vec{b}.$$
- Solve this system of m equations with m unknowns
- $\vec{x}^{k+1} = [V]\vec{X}$.
- and so on. . .

To work well GMRES should be preconditioned!

Preconditioning – principle

Principle

Replace system $A\vec{x} = \vec{b}$ by $C^{-1}A\vec{x} = C^{-1}\vec{b}$
where $\mathrm{Cond}(C^{-1}A) \ll \mathrm{Cond}(A)$.

Preconditioning – principle

Principle

Replace system $A\vec{x} = \vec{b}$ by $C^{-1}A\vec{x} = C^{-1}\vec{b}$
where $\mathrm{Cond}(C^{-1}A) \ll \mathrm{Cond}(A)$.

Which matrix C?

C should be easily invertible, typically the product of two triangular matrices.

Preconditioning – principle

Principle

Replace system $A\vec{x} = \vec{b}$ by $C^{-1}A\vec{x} = C^{-1}\vec{b}$ where $\text{Cond}(C^{-1}A) \ll \text{Cond}(A)$.

Which matrix C?

C should be easily invertible, typically the product of two triangular matrices.

- $C = \text{diag}(A)$, simplest but well. . .

Preconditioning – principle

Principle

Replace system $A\vec{x} = \vec{b}$ by $C^{-1}A\vec{x} = C^{-1}\vec{b}$
where $\mathrm{Cond}(C^{-1}A) \ll \mathrm{Cond}(A)$.

Which matrix C?

C should be easily invertible, typically the product of two triangular matrices.

- $C = \mathrm{diag}(A)$, simplest but well. . .
- incomplete Cholesky or LU factorization

Preconditioning – principle

Principle

Replace system $A\vec{x} = \vec{b}$ by $C^{-1}A\vec{x} = C^{-1}\vec{b}$ where $\mathrm{Cond}(C^{-1}A) \ll \mathrm{Cond}(A)$.

Which matrix C?

C should be easily invertible, typically the product of two triangular matrices.

- $C = \mathrm{diag}(A)$, simplest but well...
- incomplete Cholesky or LU factorization
- ...

Preconditioning – symmetry issues

Symmetry

Even if A and C are symmetric, $C^{-1}A$ may not be symmetric.
What if symmetry is needed?

Let $C^{-1/2}$ such that $C^{-1/2}C^{-1/2} = C^{-1}$.
Then $C^{-1/2}AC^{-1/2}$ is similar to $C^{-1}A$.

We consider the system

$$C^{+1/2}(C^{-1}A)C^{-1/2}C^{+1/2}\vec{x} = C^{+1/2}C^{-1}\vec{b}$$

$$(C^{-1/2}AC^{-1/2})C^{+1/2}\vec{x} = C^{-1/2}\vec{b}$$

Preconditioning – symmetry issues

Symmetry

Even if A and C are symmetric, $C^{-1}A$ may not be symmetric.
What if symmetry is needed?

Let $C^{-1/2}$ such that $C^{-1/2}C^{-1/2} = C^{-1}$.
Then $C^{-1/2}AC^{-1/2}$ is similar to $C^{-1}A$.

We consider the system

$$C^{+1/2}(C^{-1}A)C^{-1/2}C^{+1/2}\vec{x} = C^{+1/2}C^{-1}\vec{b}$$

$$(C^{-1/2}AC^{-1/2})C^{+1/2}\vec{x} = C^{-1/2}\vec{b}$$

Solve

$$(C^{-1/2}AC^{-1/2})\vec{y} = C^{-1/2}\vec{b}$$

and then

$$\vec{y} = C^{+1/2}\vec{x}.$$

Preconditioning – preconditioned conjugate gradient

Algorithm

```
choose x(k=1)
r(1) = b - A*x(1)
solve Cz(1) = r(1)
p(1) = r(1)
for k = 1 to convergence
  alpha(k) = r(k) . z(k) / p(k) . A * p(k)
  x(k+1) = x(k) + alpha(k) * p(k)
  r(k+1) = r(k) - alpha(k) * A * p(k)
  solve C z(k+1) = r(k+1)
  beta(k+1) = r(k+1) . z(k+1) / r(k) . z(k)
  p(k+1) = z(k+1) + beta(k+1) * p(k)
end for
```

At each iteration a system $C\vec{z} = \vec{r}$ is solved.

Overview

Storage – main issues

- Problems involve often a large number of variables, of degrees of freedom, say 10^6.

Storage – main issues

- Problems involve often a large number of variables, of degrees of freedom, say 10^6.
- To store a full matrix for a 10^6-order system, 10^{12} real numbers (if real) are needed... In simple precision this necessitates 4 To of memory.

Storage – main issues

- Problems involve often a large number of variables, of degrees of freedom, say 10^6.
- To store a full matrix for a 10^6-order system, 10^{12} real numbers (if real) are needed... In simple precision this necessitates 4 To of memory.
- But high order problems are often very sparse.

Storage – main issues

- Problems involve often a large number of variables, of degrees of freedom, say 10^6.
- To store a full matrix for a 10^6-order system, 10^{12} real numbers (if real) are needed... In simple precision this necessitates 4 To of memory.
- But high order problems are often very sparse.
- We therefore use a storage structure which consists in only storing relevant, non-zero, data.

Storage – main issues

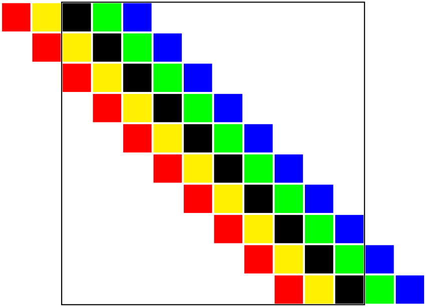
- Problems involve often a large number of variables, of degrees of freedom, say 10^6.
- To store a full matrix for a 10^6-order system, 10^{12} real numbers (if real) are needed... In simple precision this necessitates 4 To of memory.
- But high order problems are often very sparse.
- We therefore use a storage structure which consists in only storing relevant, non-zero, data.
- Access to one element A_{ij} should be very efficient.

CDS: Compressed Diagonal Storage

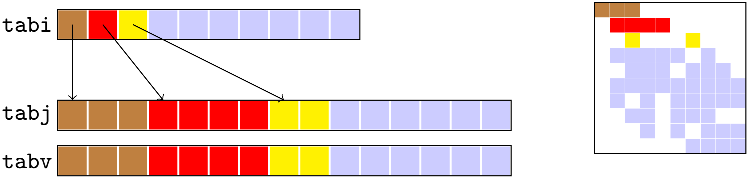
$$L(A) = \max_i L_i(A)$$

where

$$L_i(A) = \max_{j / A_{ij} \neq 0} |i - j|$$

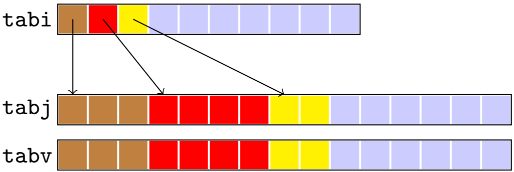


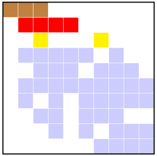
CRS: Compressed Row Storage



- All the non-zero values of the matrix are stored in a table `tab`; they are stored line by line in the increasing order of columns.

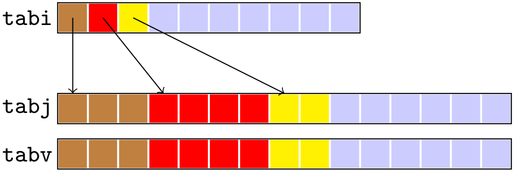
CRS: Compressed Row Storage

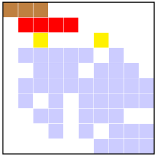




- All the non-zero values of the matrix are stored in a table `tab`; they are stored line by line in the increasing order of columns.
- A table `tabj`, with same size than `tab` stores the column number of the values in `tabv`.

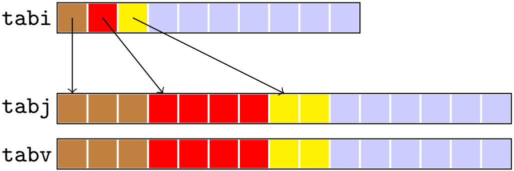
CRS: Compressed Row Storage

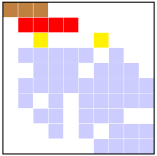




- All the non-zero values of the matrix are stored in a table `tab`; they are stored line by line in the increasing order of columns.
- A table `tabj`, with same size than `tab` stores the column number of the values in `tabv`.
- A table `tabi` with size $n+1$ stores the indices in `tabj` of the first element of each line. The last entry is the size of `tabv`.

CRS: Compressed Row Storage

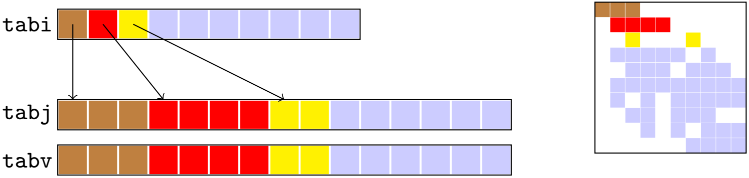




- All the non-zero values of the matrix are stored in a table `tab`; they are stored line by line in the increasing order of columns.
- A table `tabj`, with same size than `tab` stores the column number of the values in `tabv`.
- A table `tabi` with size $n+1$ stores the indices in `tabj` of the first element of each line. The last entry is the size of `tabv`.

CCS: Compressed Column Storage = Harwell Boeing

CRS: Compressed Row Storage



- All the non-zero values of the matrix are stored in a table `tab`; they are stored line by line in the increasing order of columns.
- A table `tabj`, with same size than `tab` stores the column number of the values in `tabv`.
- A table `tabi` with size $n+1$ stores the indices in `tabj` of the first element of each line. The last entry is the size of `tabv`.

CCS: Compressed Column Storage = Harwell Boeing

Generalization to symmetric matrices

CRS: Compressed Row Storage – exercise

Question

$$A = \begin{pmatrix} 0 & 4 & 1 & 6 \\ 2 & 0 & 5 & 0 \\ 0 & 9 & 7 & 0 \\ 0 & 0 & 3 & 8 \end{pmatrix}$$

CRS storage?

CRS: Compressed Row Storage – exercise

Question

$$A = \begin{pmatrix} 0 & 4 & 1 & 6 \\ 2 & 0 & 5 & 0 \\ 0 & 9 & 7 & 0 \\ 0 & 0 & 3 & 8 \end{pmatrix}$$

CRS storage?

Solution

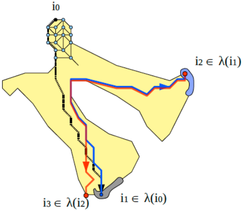
$$\begin{aligned} \texttt{tabi} &= \{1, 4, 6, 8, 10\} \\ \texttt{tabj} &= \{2, 3, 4, 1, 3, 2, 3, 3, 4\} \\ \texttt{tabv} &= \{4, 1, 6, 2, 5, 9, 7, 3, 8\} \end{aligned}$$

Overview

Cuthill–McKee algorithm – construction of a graph

Goal

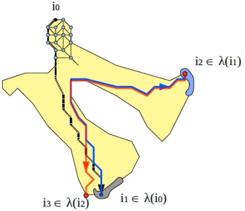
Reduce the bandwidth of a large sparse matrix by renumbering the unknowns.



Cuthill–McKee algorithm – construction of a graph

Goal

Reduce the bandwidth of a large sparse matrix by renumbering the unknowns.

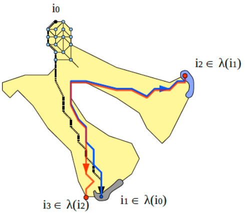


Construction

Cuthill–McKee algorithm – construction of a graph

Goal

Reduce the bandwidth of a large sparse matrix by renumbering the unknowns.



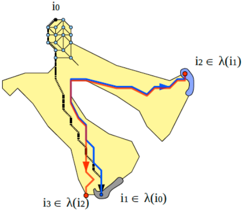
Construction

- The nodes of the graph are the unknowns of the system. They are labelled with a number from 1 to n.

Cuthill–McKee algorithm – construction of a graph

Goal

Reduce the bandwidth of a large sparse matrix by renumbering the unknowns.



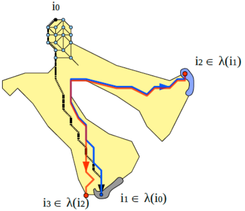
Construction

- The nodes of the graph are the unknowns of the system. They are labelled with a number from 1 to n.
- The edges are the relations between the unknowns. Two unknowns i and j are linked if $A_{ij} \neq 0$.

Cuthill–McKee algorithm – construction of a graph

Goal

Reduce the bandwidth of a large sparse matrix by renumbering the unknowns.



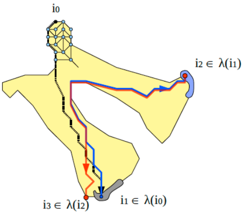
Construction

- The nodes of the graph are the unknowns of the system. They are labelled with a number from 1 to n.
- The edges are the relations between the unknowns. Two unknowns i and j are linked if $A_{ij} \neq 0$.
- The distance $d(i, j)$ between two nodes is the minimal number of edges to follow to join both nodes.

Cuthill–McKee algorithm – construction of a graph

Goal

Reduce the bandwidth of a large sparse matrix by renumbering the unknowns.



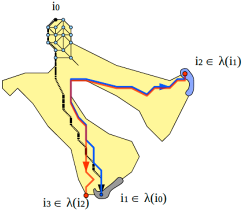
Construction

- The nodes of the graph are the unknowns of the system. They are labelled with a number from 1 to n.
- The edges are the relations between the unknowns. Two unknowns i and j are linked if $A_{ij} \neq 0$.
- The distance $d(i,j)$ between two nodes is the minimal number of edges to follow to join both nodes.
- The excentricity $E(i) = \max_j d(i,j)$

Cuthill–McKee algorithm – construction of a graph

Goal

Reduce the bandwidth of a large sparse matrix by renumbering the unknowns.



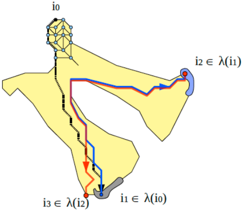
Construction

- The nodes of the graph are the unknowns of the system. They are labelled with a number from 1 to n.
- The edges are the relations between the unknowns. Two unknowns i and j are linked if $A_{ij} \neq 0$.
- The distance $d(i,j)$ between two nodes is the minimal number of edges to follow to join both nodes.
- The excentricity $E(i) = \max_j d(i,j)$
- Far neighbors are $\lambda(i) = \{j / d(i,j) = E(i)\}$

Cuthill–McKee algorithm – construction of a graph

Goal

Reduce the bandwidth of a large sparse matrix by renumbering the unknowns.



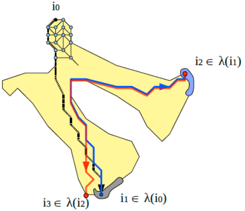
Construction

- The nodes of the graph are the unknowns of the system. They are labelled with a number from 1 to n.
- The edges are the relations between the unknowns. Two unknowns i and j are linked if $A_{ij} \neq 0$.
- The distance $d(i,j)$ between two nodes is the minimal number of edges to follow to join both nodes.
- The excentricity $E(i) = \max_j d(i,j)$
- Far neighbors are $\lambda(i) = \{j / d(i,j) = E(i)\}$
- Graph diameter $D = \max_i E(i)$

Cuthill–McKee algorithm – construction of a graph

Goal

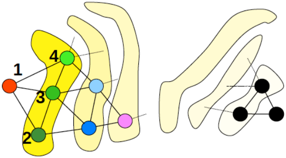
Reduce the bandwidth of a large sparse matrix by renumbering the unknowns.



Construction

- The nodes of the graph are the unknowns of the system. They are labelled with a number from 1 to n.
- The edges are the relations between the unknowns. Two unknowns i and j are linked if $A_{ij} \neq 0$.
- The distance $d(i,j)$ between two nodes is the minimal number of edges to follow to join both nodes.
- The excentricity $E(i) = \max_j d(i,j)$
- Far neighbors are $\lambda(i) = \{j / d(i,j) = E(i)\}$
- Graph diameter $D = \max_i E(i)$
- Peripheral nodes $P = \{j / E(j) = D\}$.

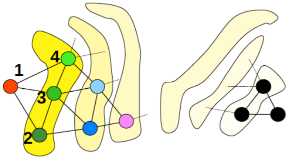
Cuthill–McKee algorithm – bandwidth reduction



This graph is used to renumber the unknonws.

- Choose a first node and label it with 1.

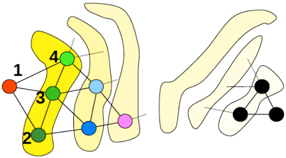
Cuthill–McKee algorithm – bandwidth reduction



This graph is used to renumber the unknonws.

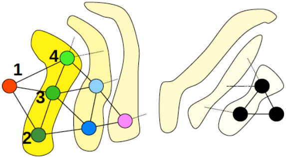
- Choose a first node and label it with 1.
- Attribute the new numbers (2,3,. . .) to the neighbors of node 1 with have the less non-labelled neighbors.

Cuthill–McKee algorithm – bandwidth reduction



This graph is used to renumber the unknonws.

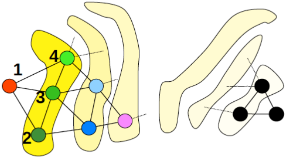
- Choose a first node and label it with 1.
- Attribute the new numbers (2,3,. . .) to the neighbors of node 1 with have the less non-labelled neighbors.
- Label the neighbors of node 2



This graph is used to renumber the unknonws.

- Choose a first node and label it with 1.
- Attribute the new numbers (2,3,. . .) to the neighbors of node 1 with have the less non-labelled neighbors.
- Label the neighbors of node 2
- and so on. . .

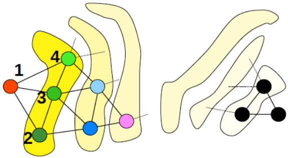
Cuthill–McKee algorithm – bandwidth reduction



This graph is used to renumber the unknonws.

- Choose a first node and label it with 1.
- Attribute the new numbers (2,3,. . .) to the neighbors of node 1 with have the less non-labelled neighbors.
- Label the neighbors of node 2
- and so on. . .
- until all nodes are labelled.

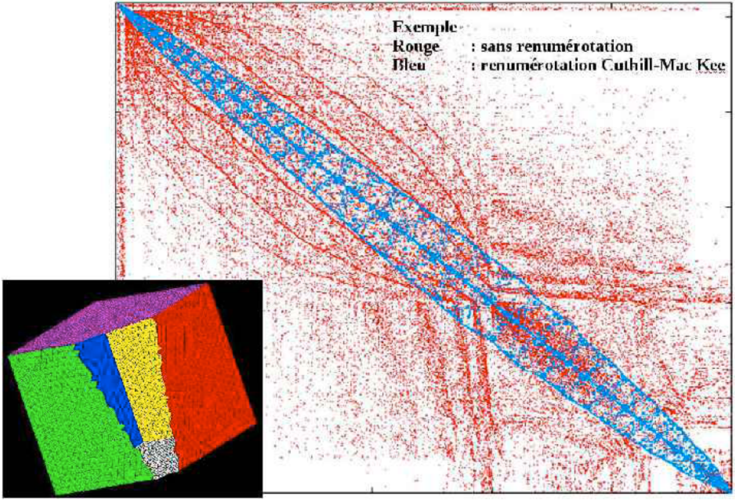
Cuthill–McKee algorithm – bandwidth reduction



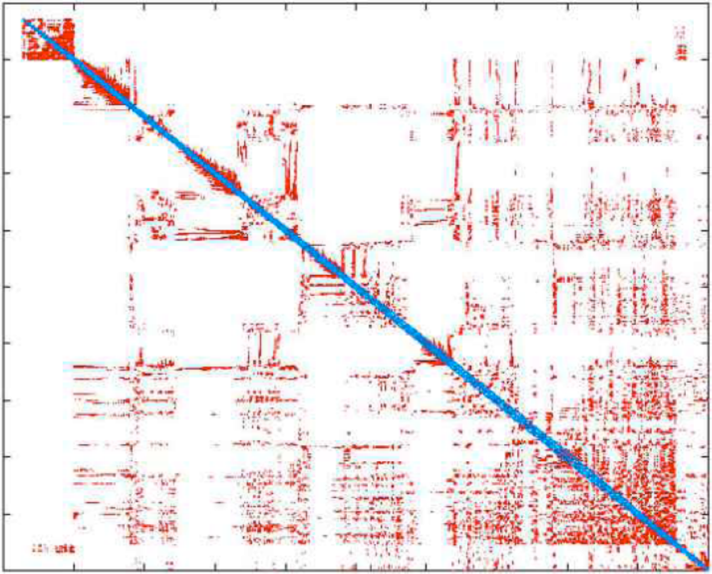
This graph is used to renumber the unknonws.

- Choose a first node and label it with 1.
- Attribute the new numbers (2,3,. . .) to the neighbors of node 1 with have the less non-labelled neighbors.
- Label the neighbors of node 2
- and so on. . .
- until all nodes are labelled.
- once this is done the numbering is reversed: the first become the last.

Cuthill–McKee algorithm – example 1



Cuthill–McKee algorithm – example 2



Overview

Bibliography

P. Lascaux, R. Théodor, *Analyse numérique matricielle appliquée à l'art de l'ingénieur* Volumes 1 and 2, 2ème édition, Masson (1997).

Bibliography

P. Lascaux, R. Théodor, *Analyse numérique matricielle appliquée à l'art de l'ingénieur* Volumes 1 and 2, 2ème édition, Masson (1997).

Gene H. Golub, Charles F. van Loan, *Matrix Computations*, 3rd edition, Johns Hopkins University Press (1996).

The End