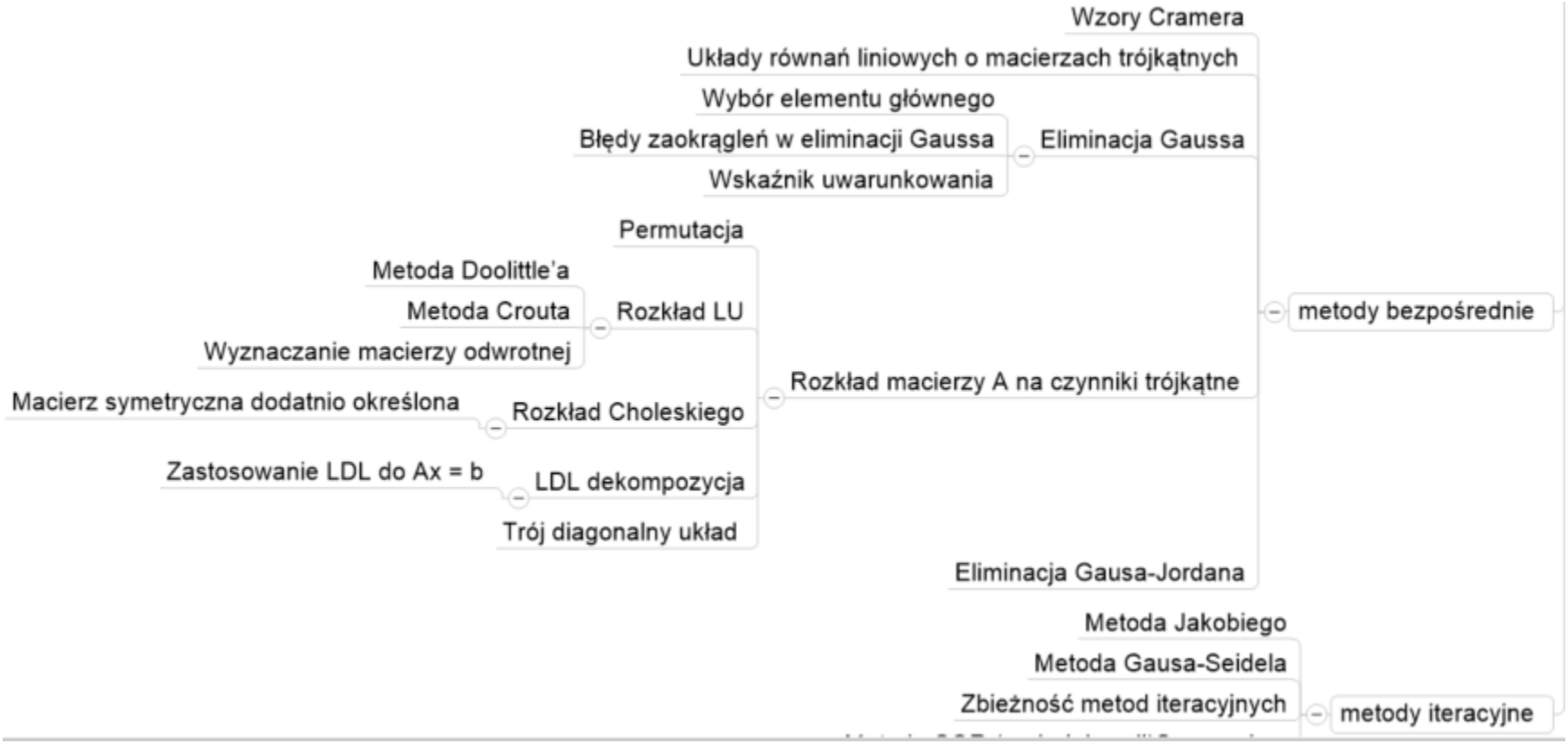
# Metody numeryczne II.
# Układy równań liniowych (cd.)

Oleksandr Sokolov

Wydział Fizyki, Astronomii i Informatyki Stosowanej UMK
http://fizyka.umk.pl/~osokolov/MNII/

# Niektóre metody rozwiązywania układów równań liniowych

# Niektóre metody rozwiązywania układów równań liniowych (cd.)



# Rozkład PA=LU

- **Dla każdej macierzy nieosobliwej** ($\det(A) \neq 0$) rozkład $PA = LU$ jest zawsze możliwy, jeśli stosujemy częściowy wybór elementu głównego.
- **Dla macierzy osobliwych** ($\det(A) = 0$) rozkład $PA = LU$ nie istnieje.

$$PA = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 7 \\ 1 & 5 & 8 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 7 \\ 1 & 5 & 8 \\ 1 & 2 & 3 \end{bmatrix}$$

$$LU = \begin{bmatrix} 1 & 0 & 0 \\ 0.5 & 1 & 0 \\ 0.5 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 4 & 7 \\ 0 & 3 & 4.5 \\ 0 & 0 & -0.5 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 7 \\ 1 & 5 & 8 \\ 1 & 2 & 3 \end{bmatrix}$$

```
>> [L U P]=lu(A)

L =

   1.0000         0         0
   0.5000    1.0000         0
   0.5000         0    1.0000


U =

   2.0000    4.0000    7.0000
        0    3.0000    4.5000
        0         0   -0.5000


P =

     0     1     0
     0     0     1
     1     0     0
```

# Macierz hermitowska

Macierz hermitowska (albo samosprzężona) – macierz kwadratowa

$$A = (a_{ij})$$

równa swojemu sprzężeniu hermitowskiemu, tj. macierz spełniająca warunek

$$(a_{ij}) = (\overline{a_{ji}})$$

Sprzężenie hermitowskie macierzy – złożenie operacji transpozycji i sprzężenia zespolonego macierzy zespolonych.

Sprzężenie zespolone – jednoargumentowe działanie algebraiczne określone na liczbach zespolonych polegające na zmianie znaku części urojonej danej liczby zespolonej.

André-Louis Cholesky
(15 October 1875, in Montguyon – 31 August 1918, in Bagneux)
was a French military officer and mathematician.

# Rozkład Choleskiego

**Definicja** *Macierz zespoloną* $\mathbf{A}$ *wymiaru* $n \times n$, *nazywamy* ***dodatnio określoną****, gdy:*

1. $\mathbf{A} = \mathbf{A}^H$, *tzn.* $\mathbf{A}$ *jest macierzą hermitowską.*
2. *Dla wszystkich* $\mathbf{x} \in C^n, \mathbf{x} \neq 0$ *zachodzi* $\mathbf{x}^H \mathbf{A}\mathbf{x} > 0$.

*Hermitowską macierz nazywamy* ***dodatnio półokreśloną*** *gdy dla wszystkich* $\mathbf{x} \in C^n$ *zachodzi* $\mathbf{x}^H \mathbf{A}\mathbf{x} \geqslant 0$.

**Twierdzenie** *Dla każdej dodatnio określonej macierzy* $\mathbf{A}$ *wymiaru* $n \times n$ *istnieje jednoznaczna trójkątna dolna macierz* $\mathbf{L}$ *wymiaru* $n \times n$, *spełniająca warunki* $l_{kk} > 0,\ k = 1, \ldots, n$, *oraz*

$$\mathbf{A} = \mathbf{L}\mathbf{L}^H.$$

*Jeżeli* $\mathbf{A}$ *jest rzeczywista, to również* $\mathbf{L}$ *jest rzeczywista.*

# Macierz symetryczna dodatnio określona

$$[A]\ n \times n$$

Wszystkie **główne minory** są dodatni

$$I.\ \det(A_{ii}) > 0\ (i = 1, 2, \ldots, n)$$ Kryterium Sylvestera

lub

$$II.\ y^T A y > 0,$$ dla każdego wektora $\forall y$

Symmetrical Positive Definite (SPD)

$$[x]\ n \times 1 \qquad [A][x] = [b] \qquad [b]\ n \times 1$$

# Macierz dodatnio określona

$$\mathbf{A}=\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n-1} & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n-1} & a_{2n} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n-11} & a_{n-12} & \cdots & a_{n-1n-1} & a_{n-1n} \\ a_{n1} & a_{n2} & \cdots & a_{nn-1} & a_{nn} \end{pmatrix},$$

lub $P(\lambda)=0,$

$$P(\lambda)=\det(A-\lambda I)$$

$$\Delta_1=a_{11}>0, \Delta_2=\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}>0, \Delta_{n-1}=\begin{vmatrix} a_{11} & \cdots & a_{1n-1} \\ \cdots & \cdots & \cdots \\ a_{n-11} & \cdots & a_{n-1n-1} \end{vmatrix}>0, \Delta_n=\det\mathbf{A}>0.$$

$$\mathbf{A}=\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}$$

$$\mathbf{A}=\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}.$$

$$P(\lambda)=0,\ \lambda_1=2>0,\ \lambda_2=2-\sqrt{2}>0,\ \lambda_3=2+\sqrt{2}>0.$$

$$\mathbf{A}-\lambda I=\begin{pmatrix} 2-\lambda & -1 & 0 \\ -1 & 2-\lambda & -1 \\ 0 & -1 & 2-\lambda \end{pmatrix}$$

Wszystkie wartości własne są dodatnie

$$\mathbf{A}=\begin{pmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{pmatrix}, \Delta_1=2>0, \Delta_2=\begin{vmatrix} 2 & -1 \\ -1 & 2 \end{vmatrix}=3>0, \Delta_3=\det A=\begin{vmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 2 \end{vmatrix}=4>0.$$

# Przykład

$$[A]=\begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

I sposób

$$\det\left|[A]_{1\times 1}\right| = |2| = 2 > 0 \qquad a_{ij} = a_{ji}$$

$$\det\left|[A]_{2\times 2}\right| = \begin{vmatrix} 2 & -1 \\ -1 & 2 \end{vmatrix} = 3 > 0$$

$$\det\left|[A]_{3\times 3}\right| = \begin{vmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{vmatrix} = 1 > 0$$

A jest SPD

# 2 sposób

$$scalar = y^T A y$$

$$= \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} \begin{bmatrix} 2 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$= \left(2y_1^2 - 2y_1y_2 + 2y_2^2\right) + \left\{y_3^2 - 2y_2y_3\right\}$$

$$= \left(y_1 - y_2\right)^2 + y_1^2 + y_2^2 + \left\{y_3^2 - 2y_2y_3\right\}$$

$$scalar = \left(y_1 - y_2\right)^2 + y_1^2 + \left(y_2 - y_3\right)^2 > 0$$

A jest SPD

# Rozkład Choleskiego (cd.)

jest procedurą rozkładu ***symetrycznej, dodatnio określonej macierzy***

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & \cdots & 0 \\ l_{21} & l_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & 0 \\ l_{n1} & l_{n2} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & \cdots & l_{n1} \\ 0 & l_{22} & \cdots & l_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{nn} \end{bmatrix}$$

$$\begin{aligned} a_{11} = l_{11}^2 &\rightarrow \boxed{l_{11} = \sqrt{a_{11}}} \\ a_{21} = l_{21} l_{11} &\rightarrow l_{21} = \frac{a_{12}}{l_{11}} \\ a_{22} = l_{21}^2 + l_{22}^2 &\rightarrow l_{22} = \sqrt{a_{22} - l_{21}^2} \\ a_{32} = l_{31} l_{21} + l_{32} l_{22} &\rightarrow l_{32} = \frac{a_{23} - l_{31} l_{21}}{l_{22}} \\ &\cdots \end{aligned}$$

$$l_{ii} = \sqrt{\left( a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \right)}$$

$$i = 1, 2, \ldots, n$$

$$l_{ji} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{jk} l_{ik}}{l_{ii}}$$

$$j = i + 1, i + 2, \ldots, n$$

# Przykład

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

$$l_{ii} = \sqrt{\left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2\right)}$$

$$l_{ji} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{jk} l_{ik}}{l_{ii}}$$

znajdujemy:

$i = 1$: $\quad l_{11} = \sqrt{4} = 2, \quad l_{21} = 2/2 = 1, \quad l_{31} = 2/2 = 1$

$i = 2$: $\quad l_{22} = \sqrt{5 - 1 \cdot 1} = 2, \quad l_{32} = (3 - 1 \cdot 1)/2 = 1$

$i = 3$: $\quad l_{33} = \sqrt{6 - 1 \cdot 1 - 1 \cdot 1} = 2$

zatem

$$\begin{bmatrix} 4 & 2 & 2 \\ 2 & 5 & 3 \\ 2 & 3 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ 1 & 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} 2 & 1 & 1 \\ 0 & 2 & 1 \\ 0 & 0 & 2 \end{bmatrix}$$

# Zadanie

Otrzymać rozkład Choleskiego dla macierzy. Sprawdzić wynik.

$$\begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix}$$

$$\begin{aligned} a_{11} = l_{11}^2 &\rightarrow l_{11} = \sqrt{a_{11}} \\ a_{21} = l_{21} l_{11} &\rightarrow l_{21} = \frac{a_{12}}{l_{11}} \\ a_{22} = l_{21}^2 + l_{22}^2 &\rightarrow l_{22} = \sqrt{a_{22} - l_{21}^2} \\ a_{32} = l_{31} l_{21} + l_{32} l_{22} &\rightarrow l_{32} = \frac{a_{23} - l_{31} l_{21}}{l_{22}} \\ &\dots \end{aligned}$$

$$l_{ii} = \sqrt{\left( a_{ii} - \sum_{k=1}^{i-1} l_{ik}^2 \right)}$$

$$l_{ji} = \frac{a_{ij} - \sum_{k=1}^{i-1} l_{jk} l_{ik}}{l_{ii}}$$

# Rozwiązanie

$$\begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 6 & 1 & 0 \\ -8 & 5 & 3 \end{pmatrix} \begin{pmatrix} 2 & 6 & -8 \\ 0 & 1 & 5 \\ 0 & 0 & 3 \end{pmatrix}$$

# Zastosowanie

***Dla symetrycznej, dodatnio określonej macierzy A***

$$Ax = b$$

$$A = LL^*$$

$$LL^*x = b$$

$$Ly = b$$

$$L^*x = y$$

U=L*

# Przykład

$$y_j = \frac{b_j - \sum_{i=1}^{j-1} l_{ij} y_i}{l_{jj}}$$

$$\begin{bmatrix} l_{11} & 0 & 0 \\ l_{12} & l_{22} & 0 \\ l_{13} & l_{23} & l_{33} \end{bmatrix} \begin{Bmatrix} y_1 \\ y_2 \\ y_3 \end{Bmatrix} = \begin{Bmatrix} b_1 \\ b_2 \\ b_3 \end{Bmatrix}$$

$$l_{11} y_1 = b_1 \qquad y_1 = \frac{b_1}{l_{11}}$$

$$l_{12} y_1 + l_{22} y_2 = b_2 \qquad y_2 = b_2 - \frac{l_{12} y_1}{l_{22}}$$

$$y_3 = \frac{b_3 - l_{13} y_1 - l_{23} y_2}{l_{33}}$$

# Przykład (cd.)

$$x_j = \frac{y_j - \sum_{i=j+1}^{n} l_{ji} x_i}{l_{jj}}$$

$$\begin{bmatrix} l_{11} & l_{12} & l_{13} \\ 0 & l_{22} & l_{23} \\ 0 & 0 & l_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$l_{33} x_3 = y_3 \qquad x_3 = \frac{y_3}{l_{33}}$$

$$x_2 = \frac{y_2 - l_{23} x_3}{l_{22}}$$

$$x_1 = \frac{y_1 - l_{12} x_2 - l_{13} x_3}{l_{11}}$$

# Zadanie

```
A =

    3    -2
   -2     3

>> B

B =

     1
     1

>> A^-1*B

ans =

    1.0000
    1.0000
```

Rozwiązać układ przez rozkład Choleskiego

```
>> chol(A)

ans =

    1.7321   -1.1547
         0    1.2910
```

# LDL* dekompozycja

# Ax = b

Macierz A jest **symetryczna**,
**ale nie koniecznie dodatnio określona**

$$A = LDL^T$$

$$A = LDL^*$$
$$LDL^*X = B$$
$$Z = DL^*X$$
$$Y = L^*X$$

$$l_{ii}=1$$

$$LZ = B$$
$$DY = Z$$
$$L^*X = Y$$

$$\begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix}$$

$$\begin{pmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 3 & 1 & 0 \\ -4 & 5 & 1 \end{pmatrix} \begin{pmatrix} 4 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 9 \end{pmatrix} \begin{pmatrix} 1 & 3 & -4 \\ 0 & 1 & 5 \\ 0 & 0 & 1 \end{pmatrix}$$

# LDL* faktoryzacja

$$[A] = [L][D][L]^T$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix} \begin{bmatrix} d_{11} & 0 & 0 \\ 0 & d_{22} & 0 \\ 0 & 0 & d_{33} \end{bmatrix} \begin{bmatrix} 1 & l_{21} & l_{31} \\ 0 & 1 & l_{32} \\ 0 & 0 & 1 \end{bmatrix}$$

$$d_{jj} = a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 d_{kk}$$

$$l_{ij} = \left( a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_{kk} l_{jk} \right) \times \left( \frac{1}{d_{jj}} \right)$$

1

# Przykład

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

$$d_{jj} = a_{jj} - \sum_{k=1}^{j-1} l_{jk}^2 d_{kk}$$

$$l_{ij} = \left( a_{ij} - \sum_{k=1}^{j-1} l_{ik} d_{kk} l_{jk} \right) \times \left( \frac{1}{d_{jj}} \right)$$

$$\begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

# Zastosowanie LDL* do **Ax = b**

$$[A]=[L][D][L]^T$$

$$[L][D][L]^T[x]=[b]$$

$$[L][z]=[b]$$

$$[D][y]=[z]$$

$$[L]^T[x]=[y]$$

$$\begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{bmatrix}\begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix}=\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

$$z_i = b_i - \sum_{k=1}^{i-1} L_{ik} z_k \quad for\ i = 1,2,3,\ldots\ldots,n$$

$$\begin{bmatrix} 1 & l_{21} & l_{31} \\ 0 & 1 & l_{32} \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$x_i = y_i - \sum_{k=i+1}^{n} l_{ki} x_k;\ for\ i = n, n-1,\ldots\ldots,1$$

# Zadanie

**Ax = b**

Rozwiązać układ przez LDL* rozkład

```
A =

    3    -2
   -2     3
```

```
>> B

B =

    1
    1

>> A^-1*B

ans =

    0.0769
   -0.3846
```

# Funkcje w Matlabie

```
>> A=[3 -2; -2 -3]
A =
     3    -2
    -2    -3
>> eig(A)
ans =
   -3.6056
    3.6056
>> [L,D] = ldl(A)
L =
    1.0000         0
   -0.6667    1.0000
D =
    3.0000         0
         0   -4.3333
```

```
>> A=[3 -2; -2 -3]
>> [L,U,P] = lu(A)
L =
    1.0000         0
   -0.6667    1.0000
U =
    3.0000   -2.0000
         0   -4.3333
P =
     1     0
     0     1
>> chol(A)
Error using chol
Matrix must be positive definite.
```

# Macierz trójdiagonalna

$$A = \begin{bmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & 0 & \\ & a_3 & b_3 & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & c_{n-1} \\ & 0 & & & a_n & b_n \end{bmatrix}$$

# LU faktoryzacja macierzy trójdiagonalnej

$$A = \begin{bmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & 0 & \\ & a_3 & b_3 & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \ddots & \ddots & c_{n-1} \\ & & & & a_n & b_n \end{bmatrix} \quad \boldsymbol{L} = \begin{bmatrix} 1 & & & & \\ l_2 & \ddots & & 0 & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & \\ 0 & & & l_n & 1 \end{bmatrix} \quad \boldsymbol{U} = \begin{bmatrix} u_1 & c_1 & & & \\ & \ddots & \ddots & & \\ & & \ddots & \ddots & 0 \\ & & & \ddots & c_{n-1} \\ 0 & & & & u_n \end{bmatrix}$$

Elementy $l_2, \ldots, l_n$ oraz $u_1, \ldots, u_n$ można obliczyć rekurencyjnie ze wzorów

$$\begin{aligned} u_1 &= b_1 \\ l_i &= a_i / u_{i-1} \\ u_i &= b_i - l_i c_{i-1} \end{aligned} \qquad \text{dla} \quad i = 2, 3, \ldots, n$$

# Zadanie

Znaleźć rozkład $LU$ macierzy

$$T = \begin{bmatrix} 2 & 2 & 0 \\ -2 & 0 & 2 \\ 0 & -2 & 0 \end{bmatrix}$$

$$u_1 = b_1$$
$$l_i = a_i / u_{i-1}$$
$$u_i = b_i - l_i c_{i-1} \quad \text{dla} \quad i = 2, 3, \ldots, n$$

# Odpowiedź

```
>> [L,U] = lu([2 2 0;-2 0 2;0 -2 0])

L =

    1    0    0
   -1    1    0
    0   -1    1


U =

    2    2    0
    0    2    2
    0    0    2
```

# Trójdiagonalny układ (cd.)

$$Tx = d$$

$$LUx = d \rightarrow Ly = d$$

$$Ux = y$$

$$y_1 = d_1$$
$$y_i = d_i - l_i y_{i-1} \quad \text{dla} \quad i = 2, 3, \ldots, n$$

$$x_n = y_n / u_n$$
$$x_i = (y_i - c_i x_{i+1}) / u_i$$

$$L = \begin{bmatrix} 1 & & & 0 \\ l_2 & \ddots & & \\ & \ddots & \ddots & \\ 0 & & \ddots & 1 \\ & & l_n & \end{bmatrix} \qquad U = \begin{bmatrix} u_1 & c_1 & & \\ & \ddots & \ddots & 0 \\ & & \ddots & c_{n-1} \\ 0 & & & u_n \end{bmatrix}$$

$$\text{dla} \quad i = n - 1, n - 2, \ldots, 1$$

# Niestabilność metody

$$T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \qquad \det T = -1$$

$$\boxed{\begin{aligned} u_1 &= b_1 \\ l_i &= a_i / u_{i-1} \\ u_i &= b_i - l_i c_{i-1} \end{aligned}}$$

$$A = \begin{bmatrix} b_1 & c_1 & & & & & \\ a_2 & b_2 & c_2 & & & 0 & \\ & a_3 & b_3 & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & c_{n-1} & \\ & 0 & & & a_n & b_n & \end{bmatrix}$$

otrzymujemy $u_1 = 1$, $l_2 = 1$, $u_2 = 0$, a więc niemożliwe jest obliczenie $l_3 = 1/u_2$.

$$A = \begin{bmatrix} b_1 & c_1 & & & & & \\ a_2 & b_2 & c_2 & & & 0 & \\ & a_3 & b_3 & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & c_{n-1} & \\ & 0 & & & a_n & b_n & \end{bmatrix} \quad L = \begin{bmatrix} 1 & & & & & & \\ l_2 & \ddots & & & & 0 & \\ & \ddots & \ddots & & & & \\ & & \ddots & \ddots & & & \\ & & & \ddots & \ddots & & \\ & 0 & & & l_n & 1 & \end{bmatrix} \quad U = \begin{bmatrix} u_1 & c_1 & & & & & \\ & \ddots & \ddots & & & 0 & \\ & & \ddots & \ddots & & & \\ & & & \ddots & \ddots & & \\ & & & & \ddots & c_{n-1} & \\ & 0 & & & & u_n & \end{bmatrix}$$

# Niestabilność metody

$$T = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}, \qquad \det T = -1$$

$$\begin{aligned} u_1 &= b_1 \\ l_i &= a_i / u_{i-1} \\ u_i &= b_i - l_i c_{i-1} \end{aligned}$$

$$A = \begin{bmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & 0 & \\ & a_3 & b_3 & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & 0 & & \ddots & \ddots & c_{n-1} \\ & & & & a_n & b_n \end{bmatrix}$$

otrzymujemy $u_1 = 1,\ l_2 = 1,\ u_2 = 0$, a więc niemożliwe jest obliczenie $l_3 = 1/u_2$.

Możliwe są inne rozwiązania

1

$$L = \begin{bmatrix} l_1 & & & 0 \\ a_2 & \ddots & & \\ & \ddots & \ddots & \\ 0 & & \ddots & \ddots \\ & & a_n & l_n \end{bmatrix}, \qquad U = \begin{bmatrix} 1 & u_1 & & \\ & \ddots & \ddots & 0 \\ & & \ddots & \ddots \\ & & \ddots & \ddots \\ 0 & & & u_{n-1} \\ & & & 1 \end{bmatrix}$$

2 $T = LDU$

3 $T = LDL^T$

```
>> T=[1 1 0;1 1 1;0 1 1]

T =

    1    1    0
    1    1    1
    0    1    1
```

```
>> [L U]=lu(T)

L =

    1.0000         0         0
    1.0000         0    1.0000
         0    1.0000         0

U =

    1.0000    1.0000         0
         0    1.0000    1.0000
         0         0    1.0000
```

```
>> [L U P]=lu(T)

L =

    1.0000         0         0
         0    1.0000         0
    1.0000         0    1.0000

U =

    1.0000    1.0000         0
         0    1.0000    1.0000
         0         0    1.0000

P =

    1.0000         0         0
         0         0    1.0000
         0    1.0000         0
```

```
>> PT=P*T

PT =

    1.0000    1.0000         0
         0    1.0000    1.0000
    1.0000    1.0000    1.0000
```

```
>> [L U]=lu(PT)

L =

    1.0000         0         0
         0    1.0000         0
    1.0000         0    1.0000

U =

    1.0000    1.0000         0
         0    1.0000    1.0000
         0         0    1.0000
```

# Rzadkie macierzy

Macierz rzadka – macierz, w której większość elementów ma wartość zero.

$$\begin{pmatrix} 4 & 1 & 2 & \frac{1}{2} & 2 \\ 1 & \frac{1}{2} & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{5}{8} & 0 \\ 2 & 0 & 0 & 0 & 16 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 7 \\ 3 \\ 7 \\ -4 \\ -4 \end{pmatrix}$$

$$Ly = b \qquad y = \begin{pmatrix} 3.5 \\ 2.5 \\ 6 \\ -2.5 \\ -0.50 \end{pmatrix}$$

$$L^T x = y$$

$$L = \begin{pmatrix} 2 & & & & 0 \\ 0.50 & 0.50 & & & \\ 1 & -1 & 1 & & \\ 0.25 & -0.25 & -0.50 & 0.50 & \\ 1 & -1 & -2 & -3 & 1 \end{pmatrix}$$

$$x = \begin{pmatrix} 2 \\ 2 \\ 1 \\ -8 \\ -0.50 \end{pmatrix}$$

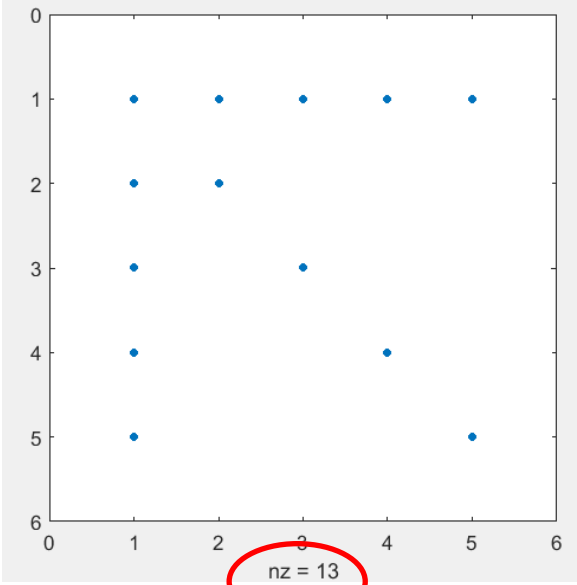
Niezerowe elementy

# Problem wypełnienia (fill-in)

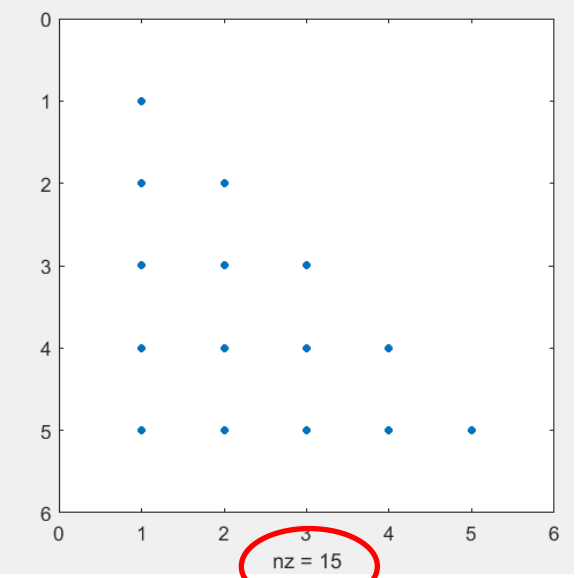
Faktoryzacja LU może **zwiększyć liczbę niezerowych elementów** w porównaniu do oryginalnej macierzy. Zjawisko to nazywa się **wypełnieniem** (fill-in).

```
A=[4 1 2 1/2 2;
   1 1/2 0 0 0;
   2 0 3 0 0;
   1/2 0 0 5/8 0;
   2 0 0 0 16];
L=chol(A)';
```

Spy(A)




Spy(L)



# Przykład

```
A=[4 1 2 1/2 2;
   1 1/2 0 0 0;
   2 0 3 0 0;
   1/2 0 0 5/8 0;
   2 0 0 0 16];
L=chol(A)';
```

```
L =

    2.0000         0         0         0         0
    0.5000    0.5000         0         0         0
    1.0000   -1.0000    1.0000         0         0
    0.2500   -0.2500   -0.5000    0.5000         0
    1.0000   -1.0000   -2.0000   -3.0000    1.0000
```

# Przebudowa macierzy

$$x_i \rightarrow \tilde{x}_{5-i+1}, \quad i = 1, 2, \ldots, 5$$

$$\begin{pmatrix} 4 & 1 & 2 & \frac{1}{2} & 2 \\ 1 & \frac{1}{2} & 0 & 0 & 0 \\ 2 & 0 & 3 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{5}{8} & 0 \\ 2 & 0 & 0 & 0 & 16 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{pmatrix} = \begin{pmatrix} 7 \\ 3 \\ 7 \\ -4 \\ -4 \end{pmatrix}$$

$$\begin{pmatrix} 16 & 0 & 0 & 0 & 2 \\ 0 & \frac{5}{8} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 3 & 0 & 2 \\ 0 & 0 & 0 & \frac{1}{2} & 1 \\ 2 & \frac{1}{2} & 2 & 1 & 4 \end{pmatrix} \begin{pmatrix} \tilde{x}_1 \\ \tilde{x}_2 \\ \tilde{x}_3 \\ \tilde{x}_4 \\ \tilde{x}_5 \end{pmatrix} = \begin{pmatrix} -4 \\ -4 \\ 7 \\ 3 \\ 7 \end{pmatrix}$$

$$L = \begin{pmatrix} 2 & & & & 0 \\ 0.50 & 0.50 & & & \\ 1 & -1 & 1 & & \\ 0.25 & -0.25 & -0.50 & 0.50 & \\ 1 & -1 & -2 & -3 & 1 \end{pmatrix}$$

$$L = \begin{pmatrix} 4 & & & & 0 \\ 0 & 0.791 & & & \\ 0 & 0 & 1.73 & & \\ 0 & 0 & 0 & 0.707 & \\ 0.500 & 0.632 & 1.15 & 1.41 & 0.129 \end{pmatrix}$$

```
A=[4 1 2 1/2 2;
   1 1/2 0 0 0;
   2 0 3 0 0;
   1/2 0 0 5/8 0;
   2 0 0 0 16];
L=chol(A)';
```

```
P=[0 0 0 0 1;
   0 0 0 1 0;
   0 0 1 0 0;
   0 1 0 0 0;
   1 0 0 0 0];
Ap=P*A; %wiersze
As=Ap*P; %kolumny
```

```
>> A

A =

    4.0000    1.0000    2.0000    0.5000    2.0000
    1.0000    0.5000         0         0         0
    2.0000         0    3.0000         0         0
    0.5000         0         0    0.6250         0
    2.0000         0         0         0   16.0000

L =

    2.0000         0         0         0         0
    0.5000    0.5000         0         0         0
    1.0000   -1.0000    1.0000         0         0
    0.2500   -0.2500   -0.5000    0.5000         0
    1.0000   -1.0000   -2.0000   -3.0000    1.0000

As =

   16.0000         0         0         0    2.0000
         0    0.6250         0         0    0.5000
         0         0    3.0000         0    2.0000
         0         0         0    0.5000    1.0000
    2.0000    0.5000    2.0000    1.0000    4.0000

>> chol(As)'

ans =

    4.0000         0         0         0         0
         0    0.7906         0         0         0
         0         0    1.7321         0         0
         0         0         0    0.7071         0
    0.5000    0.6325    1.1547    1.4142    0.1291
```

# Zagadnienie optymalizacji

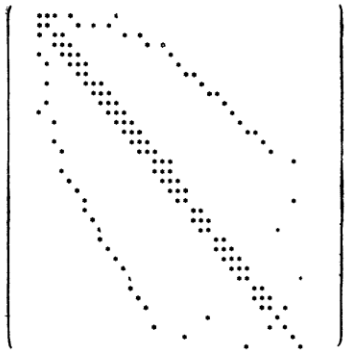
```
>> A

A =

    4.0000    1.0000    2.0000    0.5000    2.0000
    1.0000    0.5000         0         0         0
    2.0000         0    3.0000         0         0
    0.5000         0         0    0.6250         0
    2.0000         0         0         0   16.0000
```

```
P =

     0     0     0     0     1
     0     0     0     1     0
     0     0     1     0     0
     0     1     0     0     0
     1     0     0     0     0
```

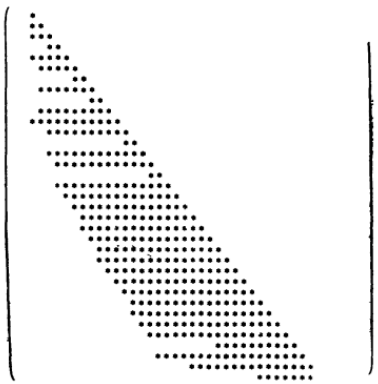
```
>> P*A*P

ans =

   16.0000         0         0         0    2.0000
         0    0.6250         0         0    0.5000
         0         0    3.0000         0    2.0000
         0         0         0    0.5000    1.0000
    2.0000    0.5000    2.0000    1.0000    4.0000
```

# Schematy przebudowy

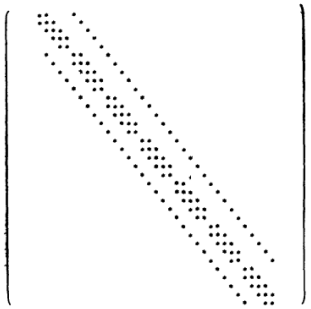
A

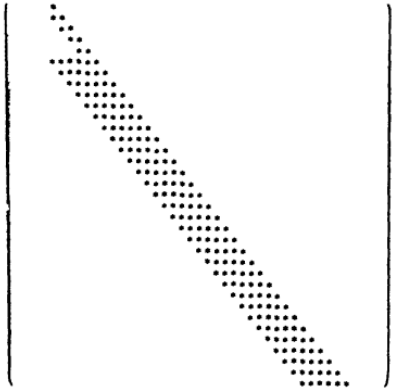


L



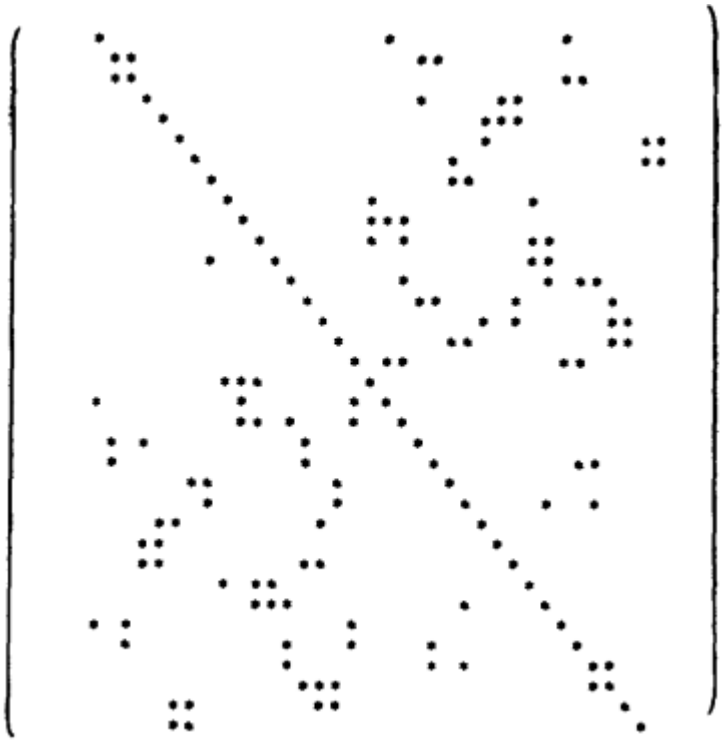
CEL: Otrzymanie stopnia rzadkości **L** takiego, jaki jest stopień rzadkości **A**



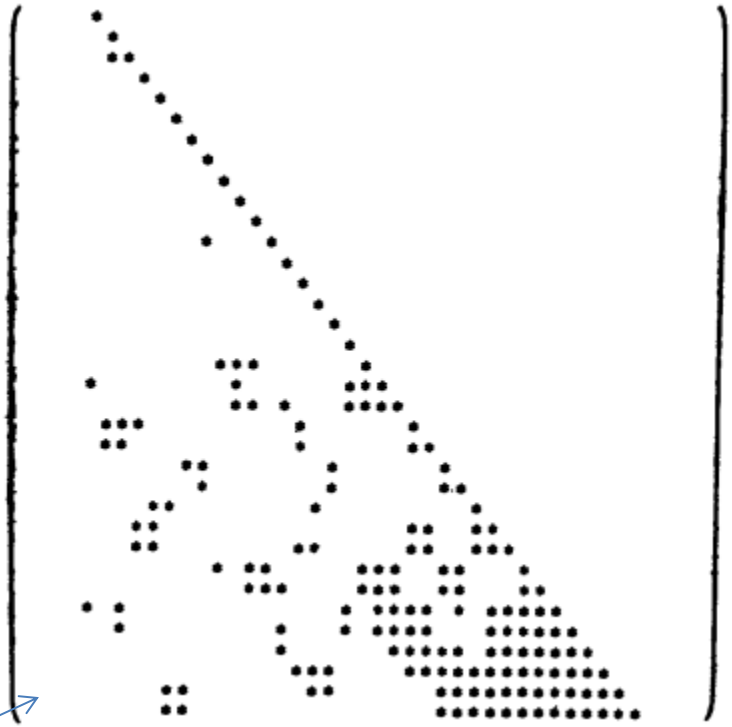


# Problem optymalizacyjny

A



L



Jak przebudować A?

P

A'=?

Macierz L powtarza „wprawie”
kształt dolnej części macierzy A

# Approximate Minimum Degree Permutation (AMD)

**Approximate Minimum Degree (AMD)** to efektywny heurystyczny algorytm do **znajdowania permutacji wierszy i kolumn rzadkiej macierzy**. Jego **celem jest minimalizacja wypełnienia** (liczby niezerowych elementów) **w trakcie dekompozycji** LU, Cholesky lub QR, co prowadzi do oszczędności pamięci i czasu obliczeń.

Dla dużych, rzadkich macierzy dekompozycje takie jak LU, Cholesky czy QR mogą prowadzić do powstawania nowych, niezerowych elementów (tzw. wypełnienia).
**AMD** reorganizuje macierz przez permutację wierszy i kolumn w taki sposób, aby **minimalizować to wypełnienie**, co usprawnia obliczenia.

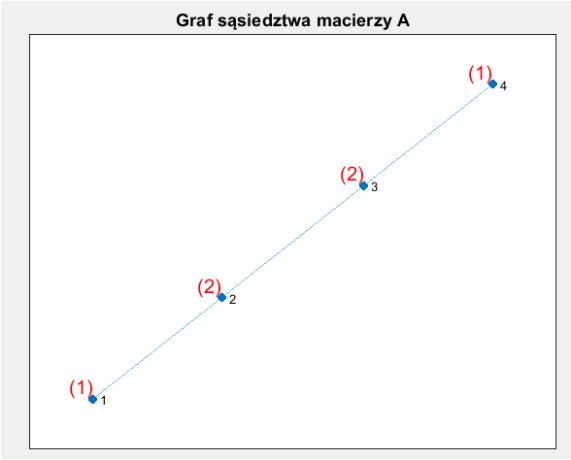
# Graf sąsiedztwa

**Graf sąsiedztwa:**
Każdą macierz rzadką można przedstawić jako graf nieskierowany, gdzie:

- **Wierzchołki odpowiadają wierszom** (lub kolumnom).
- Krawędzie istnieją, jeśli macierz **ma niezerowy element**.
- **Stopień wierzchołka**: Liczba krawędzi łączących dany wierzchołek z innymi odpowiada liczbie niezerowych elementów w danym wierszu lub kolumnie.

```
A = [
0 1 0 0;
1 0 1 0;
0 1 0 1;
0 0 1 0];
```

showGraphDS.m




**Strategia minimalnego stopnia:**
**Minimalizowanie stopnia wierzchołków (liczby połączeń) minimalizuje wypełnienie w trakcie eliminacji Gaussa**

# Algorytm AMD

1. **Inicjalizacja:**
   - Utwórz **graf sąsiedztwa** macierzy (na podstawie jej niezerowych elementów).
   - Oblicz **stopnie wierzchołków** (liczba połączeń każdego wiersza/kolumny).
2. **Wybór wierzchołka o minimalnym stopniu:**
   - Znajdź **wierzchołek o najmniejszym stopniu**.
   - **Dodaj go do permutacji**.
3. **Aktualizacja grafu (schodkowe eliminacje):**
   - Po wybraniu wierzchołka **aktualizuj graf**:
     - Połącz sąsiadów (operacja zwana wypełnieniem).
     - Przelicz stopnie wierzchołków.
4. **Kontynuacja:**
   - Powtarzaj kroki, aż wszystkie **wierzchołki zostaną dodane do permutacji**.
5. **Zakończenie:**
   - **Zwróć permutację**, która minimalizuje wypełnienie.

# Przykład

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

**Krok 1: Reprezentacja macierzy jako grafu**

Macierz $A$ jest macierzą sąsiedztwa grafu nieskierowanego. Każdy wiersz/kolumna odpowiada wierzchołkowi, a niezerowe elementy oznaczają krawędzie między wierzchołkami.

- Wierzchołek 1 jest połączony z wierzchołkiem 2.
- Wierzchołek 2 jest połączony z wierzchołkami 1 i 3.
- Wierzchołek 3 jest połączony z wierzchołkami 2 i 4.
- Wierzchołek 4 jest połączony z wierzchołkiem 3.

Graf wygląda następująco: 1 -2 - 3 - 4

# Przykład

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

### Krok 2: Inicjalizacja

Algorytm AMD działa na zasadzie eliminacji wierzchołków o najmniejszym stopniu (liczbie sąsiadów). W każdym kroku wybieramy wierzchołek o najmniejszym stopniu, eliminujemy go i aktualizujemy stopnie jego sąsiadów.

- **Stopnie wierzchołków**:
  - Wierzchołek 1: stopień 1 (sąsiad: 2)
  - Wierzchołek 2: stopień 2 (sąsiedzi: 1, 3)
  - Wierzchołek 3: stopień 2 (sąsiedzi: 2, 4)
  - Wierzchołek 4: stopień 1 (sąsiad: 3)

# Przykład

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

### Krok 3: Eliminacja wierzchołków

#### Krok 3.1: Eliminacja wierzchołka 1

- Wybieramy wierzchołek o najmniejszym stopniu (wierzchołek 1 lub 4). Załóżmy, że wybieramy wierzchołek 1.
- Eliminujemy wierzchołek 1 i dodajemy go do permutacji: $p = [1]$.
- Aktualizujemy stopnie sąsiadów wierzchołka 1:
  - Wierzchołek 2: stopień zmniejsza się z 2 do 1 (traci sąsiada 1).

# Przykład

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

**Krok 3.2: Eliminacja wierzchołka 4**

- Teraz wierzchołki 2 i 4 mają stopień 1. Wybieramy wierzchołek 4.
- Eliminujemy wierzchołek 4 i dodajemy go do permutacji: $p = [1, 4]$.
- Aktualizujemy stopnie sąsiadów wierzchołka 4:
  - Wierzchołek 3: stopień zmniejsza się z 2 do 1 (traci sąsiada 4).

# Przykład

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

**Krok 3.3: Eliminacja wierzchołka 2**

- Teraz wierzchołki 2 i 3 mają stopień 1. Wybieramy wierzchołek 2.
- Eliminujemy wierzchołek 2 i dodajemy go do permutacji: $p = [1, 4, 2]$.
- Aktualizujemy stopnie sąsiadów wierzchołka 2:
  - Wierzchołek 3: stopień zmniejsza się z 1 do 0 (traci sąsiada 2).

# Przykład

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

**Krok 3.4: Eliminacja wierzchołka 3**

- Pozostał tylko wierzchołek 3.
- Eliminujemy wierzchołek 3 i dodajemy go do permutacji: $p = [1, 4, 2, 3]$.

# Przykład

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

**Krok 4: Permutacja**

Ostateczna permutacja zwrócona przez algorytm AMD to:

$$p = [1, 4, 2, 3]$$

Oznacza to, że wiersze i kolumny macierzy $A$ należy przepermutować w kolejności: 1, 4, 2, 3.

# Przykład

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

**Krok 5: Zastosowanie permutacji do macierzy**

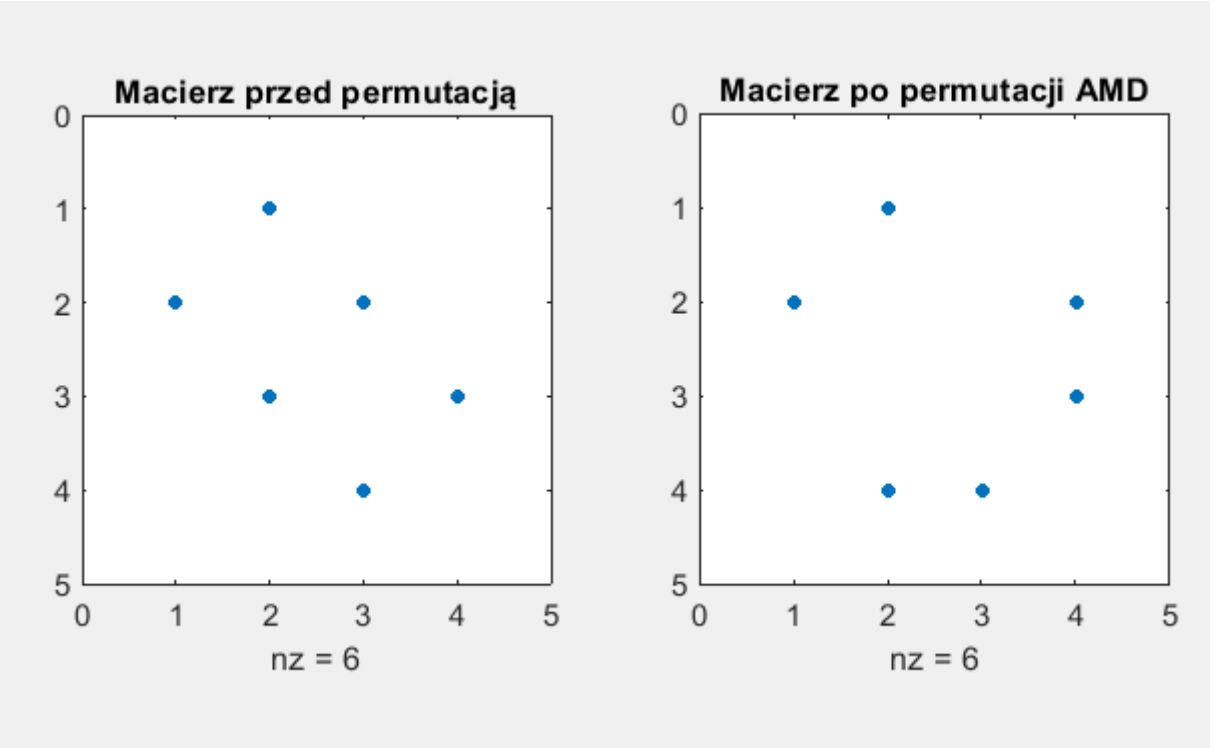
Po zastosowaniu permutacji $p$ do macierzy $A$ otrzymujemy macierz $A_{\text{perm}}$:

$$A_{\text{perm}} = A(p, p) = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

# Przykład

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

przykladAMDprosty.m



# A(P,P)

Wyrażenie `A(p, p)` w MATLAB-ie oznacza **permutację wierszy i kolumn macierzy $A$ zgodnie z wektorem permutacji $p$**. Jest to sposób na zmianę kolejności wierszy i kolumn macierzy, aby uzyskać nową macierz $A_{\text{perm}}$, która ma lepszą strukturę (np. mniejsze wypełnienie podczas faktoryzacji LU).

### Jak działa `A(p, p)`?

1. **Wektor permutacji $p$**:
   - Wektor $p$ określa nową kolejność wierszy i kolumn.
   - Przykład: Jeśli $p = [2, 1, 4, 3]$, oznacza to:
     - Wiersz 1 nowej macierzy będzie wierszem 2 starej macierzy.
     - Wiersz 2 nowej macierzy będzie wierszem 1 starej macierzy.
     - Wiersz 3 nowej macierzy będzie wierszem 4 starej macierzy.
     - Wiersz 4 nowej macierzy będzie wierszem 3 starej macierzy.
   - To samo dotyczy kolumn.

**Składnia `A(p, p)`**:
- `A(p, p)` oznacza:
  - Wybór wierszy macierzy $A$ zgodnie z wektorem $p$.
  - Wybór kolumn macierzy $A$ zgodnie z wektorem $p$.

# Przykład

```
A = [1 2 3;
     4 5 6;
     7 8 9];

p = [2, 3, 1];  % Wektor permutacji

A_perm = A(p, p);  % Permutacja wierszy i kolumn
```

```
Macierz przed permutacją:
     1     2     3
     4     5     6
     7     8     9

Macierz po permutacji:
     5     6     4
     8     9     7
     2     3     1
```

```
>> A([3,2,1],[1,2,3])

ans =

     7     8     9
     4     5     6
     1     2     3
```

# Problem pamięciowy

```
A = eye(10000);
whos A
```

```
  Name          Size                 Bytes  Class     Attributes

  A         10000x10000           800000000  double
```

This matrix uses 800-megabytes of memory.

Convert the matrix to sparse storage.

```
S = sparse(A);
whos S
```

```
  Name          Size                Bytes  Class     Attributes

  S         10000x10000           240008  double    sparse
```

In sparse form, the same matrix uses roughly 0.25-megabytes of memory. In this case, y

# Portret macierzy rzadkiej

$$P_{\mathbf{A}} \stackrel{\text{def}}{=} \{(i,j) \mid a_{ij} \neq 0\}$$ Zbiór indeksów

Macierz z niesymetrycznym portretem $\mathbf{A} \neq \mathbf{A}^T \quad \exists (i,j) \in P_{\mathbf{A}} : \quad (j,i) \notin P_{\mathbf{A}}$

Niesymetryczna macierz z symetrycznym portretem $(i,j) \in P_{\mathbf{A}} \iff (j,i) \in P_{\mathbf{A}} \quad a_{ij} \neq a_{ji};$

Symetryczna macierz $(i,j) \in P_{\mathbf{A}} \iff (j,i) \in P_{\mathbf{A}} \quad a_{ij} = a_{ji}$

# Compressed Sparse Row (CSR,CSC – column)

- **aelem**, który zawiera wszystkie niezerowe elementy macierzy A, wymienione w kolejności wierszy;
- **jptr**, który zawiera taką samą liczbę elementów jak aelem i dla każdego z nich wskazuje, w której kolumnie znajduje się dany element;
- **iptr**, który przechowuje liczbę elementów równą wymiarowi macierzy A powiększonemu o jeden. Jej i-ty element określa pozycję, od której zaczyna się i-ty wiersz macierzy w tablicach aelem i jptr. W związku z tym iptr[i+1]-iptr[i] jest równe liczbie niezerowych elementów w i-tym wierszu. Ostatni element iptr[n+1] jest równy liczbie elementów w aelem plus jeden.

Można zastosować modyfikację **CSR**, w której dane są przechowywane w ten sam sposób i w tych samych tablicach, z tym, że niezerowe elementy macierzy są zestawiane nie wierszami, a kolumnami. Ten wariant jest znany jako **CSC**.

# Przykład

$$\mathbf{A} = \begin{pmatrix} 9 & 0 & 0 & 3 & 1 & 0 & 1 \\ 0 & 11 & 2 & 1 & 0 & 0 & 2 \\ 0 & 1 & 10 & 2 & 0 & 0 & 0 \\ 2 & 1 & 2 & 9 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 12 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 2 & 2 & 0 & 0 & 3 & 0 & 8 \end{pmatrix}$$

$$n = 7$$

$$\texttt{aelem} = [\underbrace{9,3,1,1}_{4(1)},\underbrace{11,2,1,2}_{4(5)},\underbrace{1,10,2}_{3(9)},\underbrace{2,1,2,9,1}_{5(12)},\underbrace{1,1,12,1}_{4(17)},\underbrace{8}_{1(21)},\underbrace{2,2,3,8}_{4(22)}]$$

$$\texttt{jptr} = [\underbrace{1,4,5,7}_{4(1)},\underbrace{2,3,4,7}_{4(5)},\underbrace{2,3,4}_{3(9)},\underbrace{1,2,3,4,5}_{5(12)},\underbrace{1,4,5,7}_{4(17)},\underbrace{6}_{1(21)},\underbrace{1,2,5,7}_{4(22)}]$$

$$\texttt{iptr} = [1,5,9,12,17,21,22,26]$$

# Iloczyn macierzowy $z := \mathbf{A}x$

```
x=[1 2 3 4 5 6 7]';
```

```
Input: x; aelem, jptr, iptr; n
Output: z=Ax
for i= 1 : n {
z[i]:=0
for j = iptr[i] : iptr[i+1]-1 {
z[i]:=z[i]+x[jptr[j]]*aelem[j]
}
}
```

```
>> A*x

ans =

   33
   46
   40
   51
   72
   48
   77
```

49 kroków

```
z =

   33
   46
   40
   51
   72
   48
   77
```

25 kroków

```
> nnz(A)
ans =
   25
```

iloczynAx.m

# CSlR — Compressed Sparse (lower triangle)

Aby przechowywać **macierze z symetrycznym portretem i niezerową przekątną główną**, można zastosować ten sam schemat CSR, z następującymi zmianami:

- elementy głównej przekątnej są przechowywane oddzielnie w tablicy **adiag**;
- zamiast tablicy **aelem** używana jest tablica **altr**, w której w ten sam sposób przechowywane są tylko elementy pod przekątnej macierzy. Tworzone są dla niego tablice **jptr** i **iptr**;
- dla przekątnych elementów macierzy tworzona jest tablica **autr**, która przechowuje elementy nie w rzędach, ale w kolejności kolumn. Dla **autr**, ze względu na symetrię portretu, tablice **jptr** i **iptr** nie są tworzone.

# Przykład

$$\mathbf{A} = \begin{pmatrix} 9 & 0 & 0 & 3 & 1 & 0 & 1 \\ 0 & 11 & 2 & 1 & 0 & 0 & 2 \\ 0 & 1 & 10 & 2 & 0 & 0 & 0 \\ 2 & 1 & 2 & 9 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 12 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 2 & 2 & 0 & 0 & 3 & 0 & 8 \end{pmatrix}$$

```
x=[1 2 3 4 5 6 7]';
```

$$\begin{aligned} \texttt{adiag} &= [9, 11, 10, 9, 12, 8, 8] \\ \texttt{altr} &= [1, \underbrace{2, 1, 2}, \underbrace{1, 1}, \underbrace{2, 2, 3}] \\ \texttt{autr} &= [2, \underbrace{3, 1, 2}, \underbrace{1, 1}, \underbrace{1, 2, 1}] \\ \texttt{jptr} &= [2, \underbrace{1, 2, 3}, \underbrace{1, 4}, \underbrace{1, 2, 5}] \\ \texttt{iptr} &= [1, 1, 1, 2, 5, 7, 7, 10] \end{aligned}$$

# Iloczyn macierzowy II $z := \mathbf{A}x$

```
>> A*x

ans =

   33
   46
   40
   51
   72
   48
   77
```

49 kroków

```
>> z

ans =

   33
   46
   40
   51
   72
   48
   77
```

16 kroków

```
Input: x; adiag, altr, autr, jptr, iptr; n
Output: z=Ax
for i = 1 : n { z[i]:=x[i]*adiag[i] }
for i = 1 : n {
for j=iptr[i] : iptr[i+1]-1 {
z[i]:=z[i]+x[jptr[j]]*altr[j]
z[jptr[j]]:=z[jptr[j]]+x[i]*autr[j]
}
}
```

Jeśli zamienimy referencje na tablice **altr** i **autr**, to zamiast **z := Ax** zostanie obliczony iloczyn **$z := A^T x$**.

Ten sam algorytm można zastosować w przypadku **macierzy symetrycznych**. Jedyną konieczną zmianą jest użycie nie dwóch tablic **altr** i **autr**, ale jednej z nich.

# ILU do rozwiązania układu równań

Incomplete Lower-Upper triangles decomposition

A: adiag, altr, autr, jptr, iptr

L: lltr, ldiag

U: uutr, udiag

lltr-CSR

$$\mathbf{L}z = f$$

$$\mathbf{z}=\mathbf{L}^{-1}\mathbf{f}$$

Input: f; ldiag, lltr, jptr, iptr; n
Output: z=L $^{-1}$ f
Additional effect: changed f
for i=1: n {
for j= iptr[i]:iptr[i+1]-1 {
f[i]:=f[i]-z[jptr[j]]*lltr[j]
}
z[i]:=f[i]/ldiag[i]
}

uutr-CSC

$$\mathbf{U}z = f$$

$$\mathbf{z}=\mathbf{U}^{-1}\mathbf{f}$$

Input: f; udiag, uutr, jptr, iptr; n
Output: z=U $^{-1}$ f
Additional effect: changed f
for i=n: -1:1 {
z[i]:=f[i]/ldiag[i]
for j= iptr[i]:iptr[i+1]-1 {
f[jptr[j]]:=f[jptr[j]]-z[i]*uutr[j]
}
}

# Przykład

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| z = | 0.1111 | 0.1818 | 0.2818 | 0.3369 | 0.3793 | 0.7500 | 0.6595 |
| zf'= | 0.1111 | 0.1818 | 0.2818 | 0.3369 | 0.3793 | 0.7500 | 0.6595 |

9 kroków

49 kroków (LU)

```
f=[1 2 3 4 5 6 7];

L=[
     9     0     0     0     0     0     0
     0    11     0     0     0     0     0
     0     1    10     0     0     0     0
     2     1     2     9     0     0     0
     1     0     0     1    12     0     0
     0     0     0     0     0     8     0
     2     2     0     0     3     0     8
     ];
```

zf=L^-1*f

Aminus1.m

# Przykład

$$\mathbf{A} = \begin{pmatrix} 9 & 0 & 0 & 3 & 1 & 0 & 1 \\ 0 & 11 & 2 & 1 & 0 & 0 & 2 \\ 0 & 1 & 10 & 2 & 0 & 0 & 0 \\ 2 & 1 & 2 & 9 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 12 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 2 & 2 & 0 & 0 & 3 & 0 & 8 \end{pmatrix}$$

$$\mathbf{A} = \mathbf{L_A U_A}$$

$$\mathbf{L_A} = \begin{pmatrix} 1 & & & & & & \\ 0 & 1 & & & & & \\ 0 & 0.091 & 1 & & & & \\ 0.222 & 0.091 & 0.185 & 1 & & & \\ 0.111 & 0 & 0 & 0.085 & 1 & & \\ 0 & 0 & 0 & 0 & 0 & 1 & \\ 0.222 & 0.182 & -0.037 & -0.099 & 0.241 & 0 & 1 \end{pmatrix}$$

$$\mathbf{U_A} = \begin{pmatrix} 9 & 0 & 0 & 3 & 1 & 0 & 1 \\ & 11 & 2 & 1 & 0 & 0 & 2 \\ & & 9.818 & 1.909 & 0 & 0 & -0.182 \\ & & & 7.889 & 0.778 & 0 & -0.37 \\ & & & & 11.823 & 0 & 0.92 \\ & & & & & 8 & 0 \\ & & & & & & 7.149 \end{pmatrix}.$$

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

$$\mathbf{L_A}y = b$$
$$\mathbf{U_A}x = y.$$

Dużo nowych niezerowych elementów

>> [L U]=lu(A)

# Przypadek rzadkiej macierzy - ILU

$$P_{\mathbf{A}} \stackrel{\text{def}}{=} \{(i,j) \mid a_{ij} \neq 0\}$$

$$\mathbf{A} = \mathbf{LU} + \mathbf{R}$$

- $P_{\mathbf{L}} \subset P_{\mathbf{A}}$ i $P_{\mathbf{U}} \subset P_{\mathbf{A}}$; indeksy
- $\forall (i,j) \in P_{\mathbf{A}} : \ [\mathbf{LU}]_{ij} = [\mathbf{A}]_{ij}$;
- $P_{\mathbf{A}} \cap P_{\mathbf{R}} = \emptyset$.

Niekompletna LU -faktoryzacja macierzy – ILU - **incomplete LU factorization**

$$\mathbf{A} \approx \mathbf{LU}$$

# ILU faktoryzacja

W tej metodzie do portretu nie wprowadzano żadnych **nowych niezerowych elementów**!

Aby znaleźć macierze L i U, wygenerujemy je wiersz po wierszu. Załóżmy, że pierwsze (k−1) wiersze zostały już znalezione i trzeba znaleźć k-ty wiersz. Zapiszmy blokami pierwsze k wierszy rozkładu

$$\mathbf{A} = \mathbf{L}\mathbf{U} + \mathbf{R}$$

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{a}_{21}^T & \mathbf{a}_{22}^T \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{11} & \mathbf{0} \\ \mathbf{l}_{21}^T & 1 \end{pmatrix} \begin{pmatrix} \mathbf{U}_{11} & \mathbf{U}_{12} \\ \mathbf{0} & \mathbf{u}_{22}^T \end{pmatrix} + \begin{pmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{r}_{21}^T & \mathbf{r}_{22}^T \end{pmatrix}$$

$\mathbf{l}_{21}, \mathbf{u}_{22}, \mathbf{r}_{21}$ i $\mathbf{r}_{22}$ są wektorami.

Można przepisać to tak:

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{a}_{21}^T & \mathbf{a}_{22}^T \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{11}\mathbf{U}_{11} + \mathbf{R}_{11} & \mathbf{L}_{11}\mathbf{U}_{12} + \mathbf{R}_{12} \\ \mathbf{l}_{21}^T\mathbf{U}_{11} + \mathbf{r}_{21}^T & \mathbf{l}_{21}^T\mathbf{U}_{12} + \mathbf{u}_{22}^T + \mathbf{r}_{22}^T \end{pmatrix}$$

# Metoda ILU faktoryzacji (cd)

Skąd mamy

$$\begin{aligned} \mathbf{l}_{21}^T \mathbf{U}_{11} + \mathbf{r}_{21}^T &= \mathbf{a}_{21}^T ; \\ \mathbf{u}_{22}^T + \mathbf{r}_{22}^T &= \mathbf{a}_{22}^T - \mathbf{l}_{21}^T \mathbf{U}_{12} \end{aligned}$$

Pamiętając, że $l_{kk} \equiv 1$

rozwiązując te układy, możemy znaleźć współczynniki k-tych rzędów macierzy dekompozycji

$$l_{k1}, \ldots, l_{k,k-1}, u_{kk}, \ldots, u_{kn}$$

Obliczymy $l_{kj}$ zakładając, że $l_{k1} \ldots l_{k,j-1}$ zostały już znalezione.
(j<k zawsze dla macierzy L)

Jeżeli $a_{kj} = 0$ to $l_{kj} = 0$

Inaczej

$$\sum_{i=1}^{j} l_{ki} u_{ij} = \sum_{i=1}^{j-1} l_{ki} u_{ij} + l_{kj} u_{jj} = a_{kj}$$

biorąc pod uwagę fakt, że dla i > j elementy górnej macierzy trójkątnej U są równe zeru.

# Metoda ILU faktoryzacji (cd)

Skąd mamy $$l_{kj} = \frac{1}{u_{jj}} \left( a_{kj} - \sum_{i=1}^{j-1} l_{ki} u_{ij} \right)$$

Przypomnijmy, że zakłada się, że rzędy macierzy L i U od pierwszego do k-tego zostały już skonstruowane. Skoro j < k, to wszystkie współczynniki macierzy U zostały już określone.

W sposób podobny pod warunkiem $l_{jj} \equiv 1$ można obliczyć $u_{kj}$

$$u_{kj} = a_{kj} - \sum_{i=1}^{k-1} l_{ki} u_{ij} \quad \text{gdy} \quad a_{kj} \neq 0$$

Teraz dla i > k elementy dolnej trójkątnej macierzy L są równe zeru.

W przypadku gdy $a_{kj} = 0$ mamy $u_{kj} = 0$

# Algorytm ILU

Dla $k = \overline{1, n}$

Dla $j = \overline{1, k-1}$

Jeżeli $(k, j) \notin P_{\mathbf{A}}$

to $l_{kj} := 0$

inaczej $l_{kj} := \frac{1}{u_{jj}} \left( a_{kj} - \sum_{i=1}^{j-1} l_{ki} u_{ij} \right)$

zwiększ $j$

$l_{kk} := 1$

Dla $j = \overline{k, n}$

Jeżeli $(k, j) \notin P_{\mathbf{A}}$

to $u_{kj} := 0$

inaczej $u_{kj} := a_{kj} - \sum_{i=1}^{k-1} l_{ki} u_{ij}$

zwiększ $j$

zwiększ $k$

# Przykład

$$\mathbf{A} = \begin{pmatrix} 9 & 0 & 0 & 3 & 1 & 0 & 1 \\ 0 & 11 & 2 & 1 & 0 & 0 & 2 \\ 0 & 1 & 10 & 2 & 0 & 0 & 0 \\ 2 & 1 & 2 & 9 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 12 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 2 & 2 & 0 & 0 & 3 & 0 & 8 \end{pmatrix}$$

$P_A$={(1,1),(1,4),(1,5),(1,7),(2,2),…}

Dla $k = \overline{1,n}$

  Dla $j = \overline{1,k-1}$

    Jeżeli $(k,j) \notin P_{\mathbf{A}}$

      to $l_{kj} := 0$

      inaczej $l_{kj} := \frac{1}{u_{jj}} \left( a_{kj} - \sum_{i=1}^{j-1} l_{ki} u_{ij} \right)$

  zwiększ $j$

  $l_{kk} := 1$

  Dla $j = \overline{k,n}$

    Jeżeli $(k,j) \notin P_{\mathbf{A}}$

      to $u_{kj} := 0$

      inaczej $u_{kj} := a_{kj} - \sum_{i=1}^{k-1} l_{ki} u_{ij}$

  zwiększ $j$

zwiększ $k$

k=1,j=0
l11=1

j=1
(1,1)∈$P_A$
u11=a11=9
j=2
(1,2) ∉$P_A$
**u12=0**
…
j=7
(1,7)∈$P_A$
u17=a17=1

k=2,j=1
(2,1)∉$P_A$
**l21=0**
l22=1

j=2
(2,2)∈$P_A$
u22=a22-l11*u12=11-1*0=11
…

# Przykład

$$\mathbf{A} = \begin{pmatrix} 9 & 0 & 0 & 3 & 1 & 0 & 1 \\ 0 & 11 & 2 & 1 & 0 & 0 & 2 \\ 0 & 1 & 10 & 2 & 0 & 0 & 0 \\ 2 & 1 & 2 & 9 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 12 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 2 & 2 & 0 & 0 & 3 & 0 & 8 \end{pmatrix}$$

$$\mathbf{L} = \begin{pmatrix} 1 & & & & & & \\ 0 & 1 & & & & & \\ 0 & 0.091 & 1 & & & & \\ 0.222 & 0.091 & 0.185 & 1 & & & \\ 0.111 & 0 & 0 & 0.085 & 1 & & \\ 0 & 0 & 0 & 0 & 0 & 1 & \\ 0.222 & 0.182 & 0 & 0 & 0.235 & 0 & 1 \end{pmatrix};$$

$$\mathbf{U} = \begin{pmatrix} 9 & 0 & 0 & 3 & 1 & 0 & 1 \\ & 11 & 2 & 1 & 0 & 0 & 2 \\ & & 9.818 & 1.909 & 0 & 0 & 0 \\ & & & 7.889 & 0.778 & 0 & 0 \\ & & & & 11.823 & 0 & 0.889 \\ & & & & & 8 & 0 \\ & & & & & & 7.205 \end{pmatrix}.$$

$$\mathbf{R} = \mathbf{A} - \mathbf{LU} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.182 \\ 0 & 0 & 0 & 0 & 0 & 0 & -0.404 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.364 & -0.848 & 0 & 0 & 0 \end{pmatrix}.$$

# Sparse w Matlabie

S=sparse(A);

nnz(A)=25 Number of nonzero matrix elements.

7x7 sparse double

```
val =

   (1,1)        9
   (4,1)        2
   (5,1)        1
   (7,1)        2
   (2,2)       11
   (3,2)        1
   (4,2)        1
   (7,2)        2
   (2,3)        2
   (3,3)       10
   (4,3)        2
   (1,4)        3
   (2,4)        1
   (3,4)        2
```

[L,U,P] = ilu(S);

permutation matrix P

S L

7x7 sparse double

```
val =

   (1,1)        1.0000
   (4,1)        0.2222
   (5,1)        0.1111
   (7,1)        0.2222
   (2,2)        1.0000
   (3,2)        0.0909
   (4,2)        0.0909
   (7,2)        0.1818
   (3,3)        1.0000
   (4,3)        0.1852
   (4,4)        1.0000
   (5,4)        0.0845
   (5,5)        1.0000
   (7,5)        0.2349
   (6,6)        1.0000
   (7,7)        1.0000
```

S L U

7x7 sparse double

```
val =

   (1,1)        9.0000
   (2,2)       11.0000
   (2,3)        2.0000
   (3,3)        9.8182
   (1,4)        3.0000
   (2,4)        1.0000
   (3,4)        1.9091
   (4,4)        7.8889
   (1,5)        1.0000
   (4,5)        0.7778
   (5,5)       11.8232
   (6,6)        8.0000
   (1,7)        1.0000
   (2,7)        2.0000
   (5,7)        0.8889
   (7,7)        7.2053
```

S L U P

7x7 sparse double

```
val =

   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
   (5,5)        1
   (6,6)        1
   (7,7)        1
```

iluA.m

# Sparse to Full

```
Lf=full(L);
Uf=full(U);
Pf=full(P);
R=A-Lf*Uf;
Af=Lf*Uf;
```

$$\mathbf{A} = \begin{pmatrix} 9 & 0 & 0 & 3 & 1 & 0 & 1 \\ 0 & 11 & 2 & 1 & 0 & 0 & 2 \\ 0 & 1 & 10 & 2 & 0 & 0 & 0 \\ 2 & 1 & 2 & 9 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 12 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 8 & 0 \\ 2 & 2 & 0 & 0 & 3 & 0 & 8 \end{pmatrix}$$

S L U P R Lf

7x7 double

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0.0909 | 1 | 0 | 0 | 0 | 0 |
| 4 | 0.2222 | 0.0909 | 0.1852 | 1 | 0 | 0 | 0 |
| 5 | 0.1111 | 0 | 0 | 0.0845 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 7 | 0.2222 | 0.1818 | 0 | 0 | 0.2349 | 0 | 1 |

S L U P R Lf Uf Pf Af

7x7 double

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 9 | 0 | 0 | 3 | 1 | 0 | 1 |
| 2 | 0 | 11 | 2 | 1 | 0 | 0 | 2 |
| 3 | 0 | 1 | 10 | 2 | 0 | 0 | 0.1818 |
| 4 | 2 | 1 | 2 | 9 | 1 | 0 | 0.4040 |
| 5 | 1 | 0 | 0 | 1 | 12 | 0 | 1 |
| 6 | 0 | 0 | 0 | 0 | 0 | 8 | 0 |
| 7 | 2 | 2 | 0.3636 | 0.8485 | 3 | 0 | 8 |

S L U P R Lf Uf

7x7 double

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 9 | 0 | 0 | 3 | 1 | 0 | 1 |
| 2 | 0 | 11 | 2 | 1 | 0 | 0 | 2 |
| 3 | 0 | 0 | 9.8182 | 1.9091 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 7.8889 | 0.7778 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 11.8232 | 0 | 0.8889 |
| 6 | 0 | 0 | 0 | 0 | 0 | 8 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 7.2053 |

S L U P R

7x7 double

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | -0.1818 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | -0.4040 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | -0.3636 | -0.8485 | 0 | 0 | 0 |

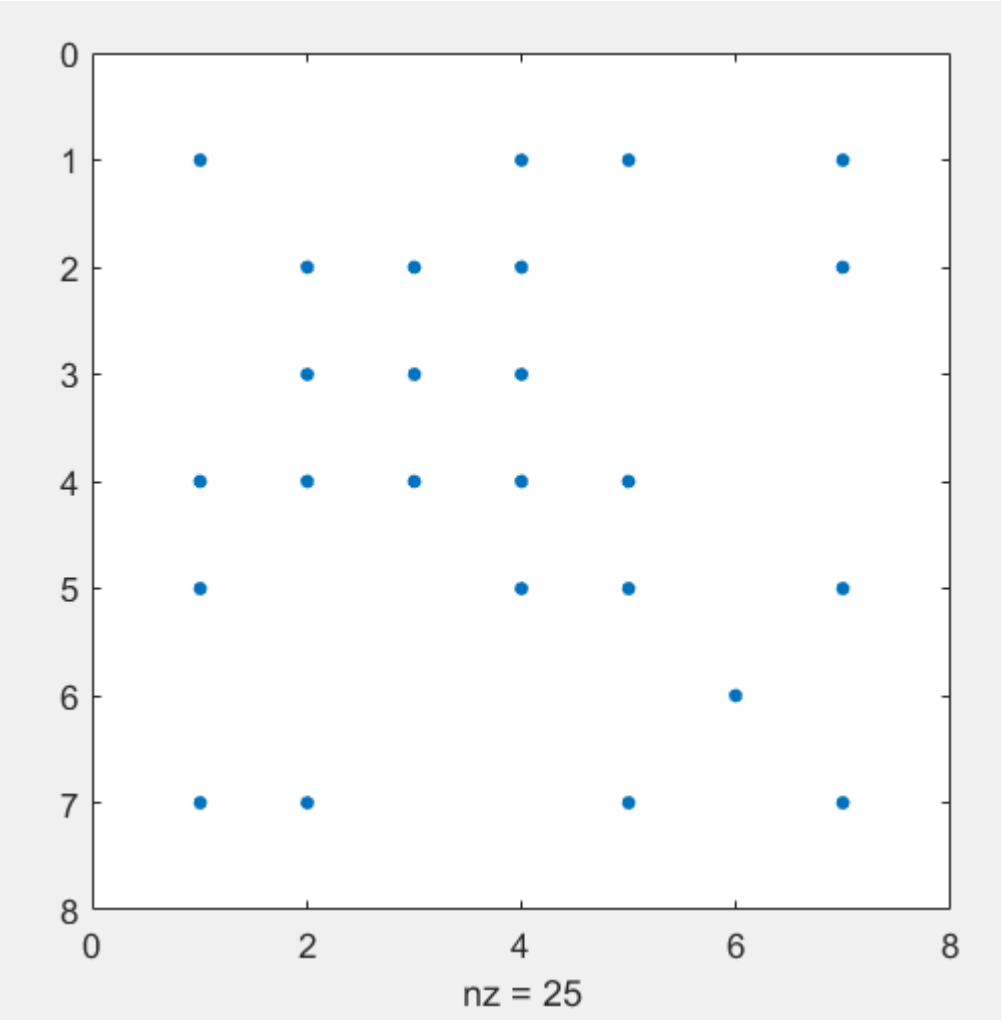
S L U P R Lf Uf Pf

7x7 double

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

# Matlab

spy(A)



# Realizacja algorytmu ILU

iluA.m | ILIOS.m

```
clear all;
PA=[1 1;1 2;2 1;2 2;2 3;3 1;3 3];
A=[1 1 0;2 1 3;0 0 8];
%A=[9 0 0 3 1 0 1;0 11 2 1 0 0 2;0 1 10 2 0 0 0; 2 1 2 9 1
%PA=[1 1;1 4;1 5;1 7;2 2;2 3;2 4;2 7;3 2 ;3 3;3 4;4 1; 4 2
n=3;
%n=7;
%U(1,1)=A(1,1);
%L=eye(n,n);

for k=1:n
    for j=1:k-1
        if isempty(findrow(num2str([k j]),num2str(PA)))
            L(k,j)=0;
        else
            s=0;
            for i=1:j-1
                s=L(k,i)*U(i,j);
            end;
            L(k,j)=1/U(j,j)*(A(k,j)-s);
        end;%if
    end;%j
    L(k,k)=1;
    for j=k:n
        if isempty(findrow(num2str([k j]),num2str(PA)))
            U(k,j)=0;
        else
            s=0;
            for i=1:k-1
                s=L(k,i)*U(i,j);
            end;
            U(k,j)=A(k,j)-s;
        end;%if
    end;%j

Uend;%k
```

$(k,j) \notin P_A$

# Przypadek symetrycznej macierzy

$$\mathbf{A} = \mathbf{LU}$$

$$\mathbf{A} = \mathbf{A}^T \quad \Rightarrow \quad \mathbf{L} = \mathbf{U}^T$$

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T + \mathbf{R} = \mathbf{U}^T\mathbf{U} + \mathbf{R}$$

lub $\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^T + \mathbf{R}$

# Algorytm

Dla $k = \overline{1, n}$

  Dla $j = \overline{1, k-1}$

    Jeżeli $(k, j) \notin P_{\mathbf{A}}$

      to $l_{kj} := 0$

      inaczej $l_{kj} := \frac{1}{d_j}\left(a_{kj} - \sum_{i=1}^{j-1} d_i l_{ik} l_{jk}\right)$

  zwiększ $j$

  $d_k := a_{kk} - \sum_{i=1}^{k-1} l_{ki}^2 d_i^2$

zwiększ $k$

# Przykład

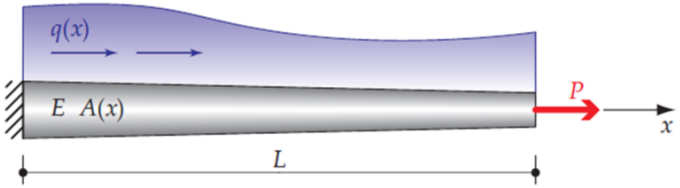
$$\mathbf{A} = \begin{pmatrix} 9 & 0 & 3 & 0 \\ 0 & 8 & 0 & 1 \\ 3 & 0 & 11 & 1 \\ 0 & 1 & 1 & 9 \end{pmatrix}$$

$$\mathbf{L} = \begin{pmatrix} 1 & & & \\ 0 & 1 & & \\ 0.333 & 0 & 1 & \\ 0 & 0.125 & 0.091 & 1 \end{pmatrix}; \qquad \mathbf{D} = \operatorname{diag}(9, 8, 11, 8.784)\,.$$

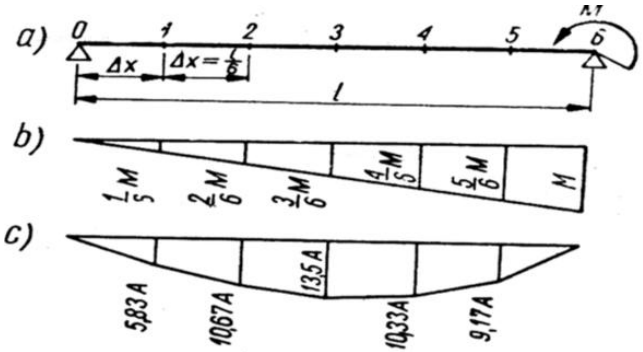
R=0

# Zastosowania

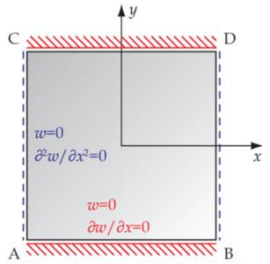
Rozciąganie pręta




Ugięcie belki

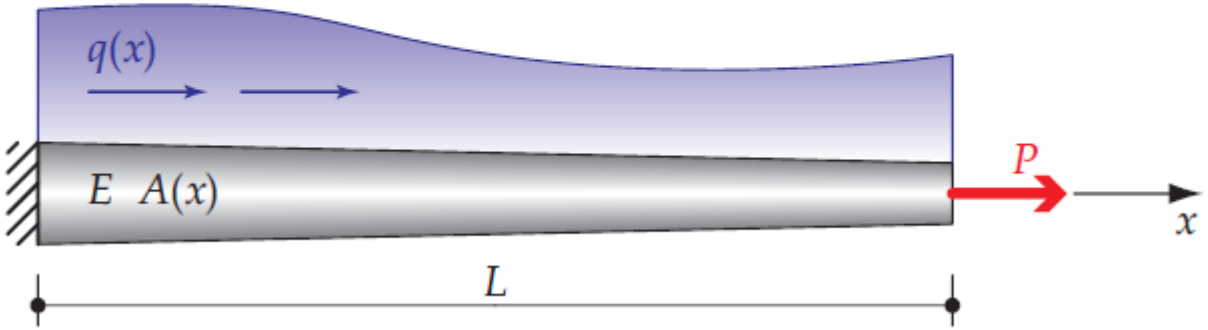


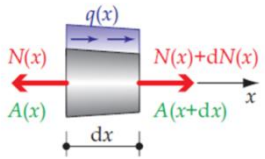

Zginanie cienkiej płyty



# Pręt rozciągany

Rozważmy pręt o powierzchni przekroju poprzecznego $A(x)$ i długości $L$. Pręt zrobiony jest z materiału o module Younga $E$. Pręt jest obciążony obciążeniem ciągłym działającym wzdłuż osi pręta o intensywności $q(x)$ i siłą skupioną $P$.




$N(x)$ i $N(x) + \mathrm{d}N(x)$ są **siłami normalnymi** działającymi w przekrojach $A(x)$ i $A(x + \mathrm{d}x)$. Korzystając z **warunku równowagi** możemy zapisać

$$-N(x) + N(x) + \mathrm{d}N(x) + q(x)\mathrm{d}x = 0.$$

Siłę **normalną** (podłużną) stanowi suma **naprężeń normalnych**

$$N(x) = \sigma_x(x)A(x).$$

Wykorzystując **równanie fizyczne** (konstytutywne, Hooke'a) określające zależność odkształcenia od naprężenia

$$\sigma_x(x) = E\varepsilon_x(x),$$

i **równanie geometryczne** (Cauchy'ego) wiążące ze zobą przemieszczenia i odkształcenia

$$\varepsilon_x(x) = \frac{\mathrm{d}u(x)}{\mathrm{d}x},$$

$$EA\frac{\mathrm{d}^2u(x)}{\mathrm{d}x^2} + q(x) = 0,$$

$$0 < x < L.$$

$$u(x = 0) = 0$$

*u(x)* – przemieszczenie punkowe

# Metoda różnic skończonych

**Co to jest MRS?**

**Metoda różnic skończonych** (MRS) jest przybliżoną metodą dyskretnego rozwiązywania problemów brzegowych, opisywanych zwyczajnymi lub cząstkowymi równaniami różniczkowymi.

Idea metody polega na zamianie **operatorów różniczkowych** na odpowiednie **operatory różnicowe** (ilorazy różnicowe), określone na dyskretnym i regularnym zbiorze punktów, zwanych **siatką**.

# Metoda różnic skończonych

Metodę różnic skończonych zastosujemy do rozwiązania równania

$$\frac{\mathrm{d}^2 u}{\mathrm{d}x^2} = f(x, y)$$

W pierwszym kroku zamienimy operatory różniczkowe na odpowiednie operatory różnicowe, wykorzystamy szereg Taylora dla funkcji $u(x, y)$ w otoczeniu $u(x + h, y)$ i $u(x - h, y)$

$$u(x + h, y) = u(x, y) + \frac{\mathrm{d}u(x, y)}{\mathrm{d}x} h + \frac{1}{2} \frac{\mathrm{d}^2 u(x, y)}{\mathrm{d}x^2} h^2 + \frac{1}{6} \frac{\mathrm{d}^3 u(x, y)}{\mathrm{d}x^3} h^3 + \ldots,$$

$$u(x - h, y) = u(x, y) - \frac{\mathrm{d}u(x, y)}{\mathrm{d}x} h + \frac{1}{2} \frac{\mathrm{d}^2 u(x, y)}{\mathrm{d}x^2} h^2 - \frac{1}{6} \frac{\mathrm{d}^3 u(x, y)}{\mathrm{d}x^3} h^3 + \ldots.$$

# Metoda różnic skończonych

Odejmując stronami i pomijając wyrazy z $h^2$ i wyższe otrzymamy

$$\frac{\mathrm{d}u(x,y)}{\mathrm{d}x} \approx \frac{1}{2h}[u(x+h,y) - u(x-h,y)].$$

Powyższą zależność nazywa się **wzorem różnicowym centralnym pierwszego rzędu**.

W przypadku sumowania stronami rozwinięć w szereg Taylora i po pominięci wyrazów z $h^3$ i wyższych, otrzymamy

$$\frac{\mathrm{d}^2u(x,y)}{\mathrm{d}x^2} \approx \frac{1}{h^2}[u(x+h,y) - 2u(x,y) + u(x-h,y)].$$

Powyższe zależności nazywają się **wzorami różnicowymi drugiego rzędu**.

# Metoda różnic skończonych

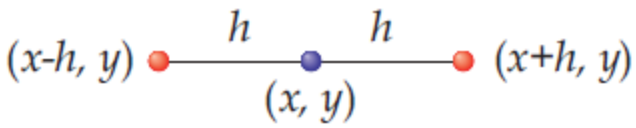
Podstawiając wzory różnicowe do wyjściowego równania, otrzymamy

$$u(x+h,y) - 2u(x,y) + u(x-h,y) = f(x,y),$$

lub

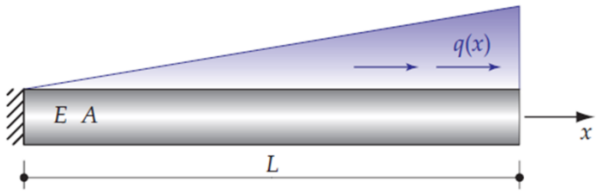
$$u(x_{i+1},y_j) - 2u(x_i,y_j) + u(x_{i-1},y_j) = f(x_i,y_j).$$

Na rysunku poniżej znajduje się punkty występujące we wzorach różnicowych, tworzące tzw. **gwiazdę różnicową**.

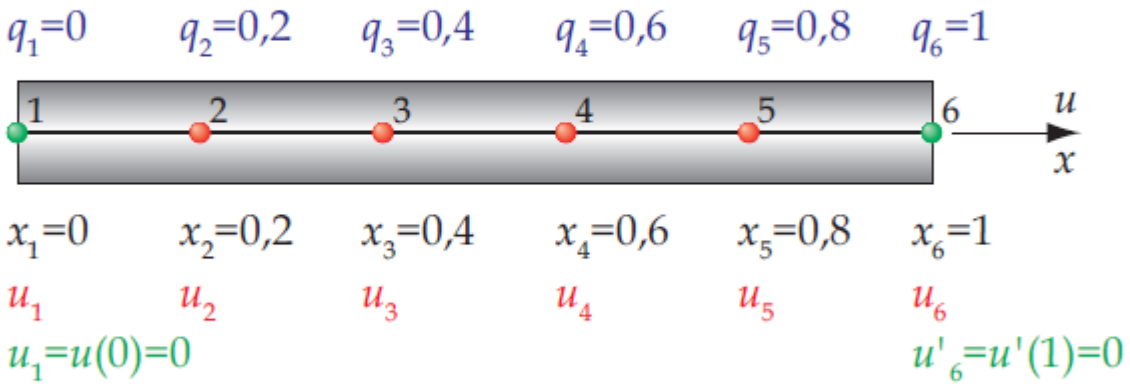


# Metoda różnic skończonych

- sztywność pręta $EA = 1$,
- obciążenie $q(x) = x$,
- długość pręta $L = 1$,




Do dalszych obliczeń, przyjmijmy siatkę 6-węzłową, zatem odległość między węzłami wynosi $h = 0,2$.




*u(x)* – przemieszczenie punkowe

# Metoda różnic skończonych

Zapiszmy układ równań MRS

- węzeł 1: $u_1 = 0,$ podstawowy warunek brzegowy,
- węzeł 2: $\frac{u_1 - 2u_2 + u_3}{(0,2)^2} = -0,2,$ iloraz różnicowy centralny,
- węzeł 3: $\frac{u_2 - 2u_3 + u_4}{(0,2)^2} = -0,4,$ iloraz różnicowy centralny,
- węzeł 4: $\frac{u_3 - 2u_4 + u_5}{(0,2)^2} = -0,6,$ iloraz różnicowy centralny,
- węzeł 5: $\frac{u_4 - 2u_5 + u_6}{(0,2)^2} = -0,8,$ iloraz różnicowy centralny,
- węzeł 6: $\frac{u_6 - u_5}{0,2} = 0,$ naturalny warunek brzegowy, iloraz różnicowy wstecz.

*u(x)* – przemieszczenie punkowe

# Metoda różnic skończonych

Po prostych przekształceniach otrzymamy układ równań

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & -2 & 1 & 0 & 0 & 0 \\ 0 & 1 & -2 & 1 & 0 & 0 \\ 0 & 0 & 1 & -2 & 1 & 0 \\ 0 & 0 & 0 & 1 & -2 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \end{bmatrix} = \begin{bmatrix} 0 \\ -\frac{1}{125} \\ -\frac{2}{125} \\ -\frac{3}{125} \\ -\frac{4}{125} \\ 0 \end{bmatrix},$$

a jego rozwiązanie wynosi

$$u_1 = 0, \quad u_2 = \frac{55}{750}, \quad u_3 = \frac{104}{750}, \quad u_4 = \frac{156}{750}, \quad u_5 = \frac{180}{750}, \quad u_6 = \frac{180}{750}.$$

A

6x6 double

| | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | -2 | 1 | 0 | 0 | 0 |
| 3 | 0 | 1 | -2 | 1 | 0 | 0 |
| 4 | 0 | 0 | 1 | -2 | 1 | 0 |
| 5 | 0 | 0 | 0 | 1 | -2 | 1 |
| 6 | 0 | 0 | 0 | 0 | 1 | 1 |

>> nnz(A)= 15

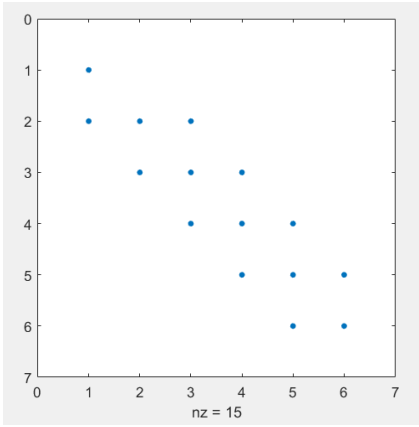
As-sparse(A)

As | S | L

6x6 sparse double

```
val =

   (1,1)        1
   (2,1)        1
   (2,2)       -2
   (3,2)        1
   (2,3)        1
   (3,3)       -2
   (4,3)        1
   (3,4)        1
   (4,4)       -2
   (5,4)        1
   (4,5)        1
   (5,5)       -2
   (6,5)        1
   (5,6)        1
   (6,6)        1
```

Spy(A)




[L U P]=ilu(As)

```
>> [L U P]=lu(As)

L =

   (1,1)        1.0000
   (2,1)        1.0000
   (2,2)        1.0000
   (3,2)       -0.5000
   (3,3)        1.0000
   (4,3)       -0.6667
   (4,4)        1.0000
   (5,4)       -0.7500
   (5,5)        1.0000
   (6,5)       -0.8000
   (6,6)        1.0000

U =

   (1,1)        1.0000
   (2,2)       -2.0000
   (2,3)        1.0000
   (3,3)       -1.5000
   (3,4)        1.0000
   (4,4)       -1.3333
   (4,5)        1.0000
   (5,5)       -1.2500
   (5,6)        1.0000
   (6,6)        1.8000

P =

   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
   (5,5)        1
   (6,6)        1
```

```
>> full(L*U)

ans =

     1     0     0     0     0     0
     1    -2     1     0     0     0
     0     1    -2     1     0     0
     0     0     1    -2     1     0
     0     0     0     1    -2     1
     0     0     0     0     1     1
```

b=[0 -1/125 -2/125 -3/125 -4/125 0];

```
>> A^-1*b'

ans =

         0
    0.0267
    0.0453
    0.0480
    0.0267
   -0.0267
```

```
>> [L U P]=ilu(As);
>> y=U^-1*b';
x=L^-1*y;
>> x

x =

         0
    0.0217
    0.0463
    0.0681
    0.0767
    0.0613
```

# Metody bezpośrednie a metody iteracyjne

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{A} = \mathbf{LU}$$

$$\mathbf{LUx} = \mathbf{b}$$

$$Ax = b \quad \Longleftrightarrow \quad x = Gx + c$$

?

$$x_{k+1} = Gx_k + c$$

**bezpośrednie**

**metody które przy braku błędów zaokrągleń dają dokładne rozwiązanie po skończonej liczbie przekształceń układu wejściowego**

- duża efektywność dla układów o macierzach pełnych
- duże obciążenie pamięci
- możliwa niestabilność ze względu na błędy zaokrągleń i wskaźnika uwarunkowania

**iteracyjne**

**polegają na konstrukcji nieskończonego ciągu wektorów, zbieżnych do szukanego rozwiązania, x(0) -> x(1) -> x(2) ->...**

- efektywne dla macierzy rzadkich, dużych rozmiarów
- stosunkowo nieduże obciążenie pamięci
- problemy ze zbieżnością

# Przykład

$$Ax = b \quad \Longleftrightarrow \quad x = Gx + c$$

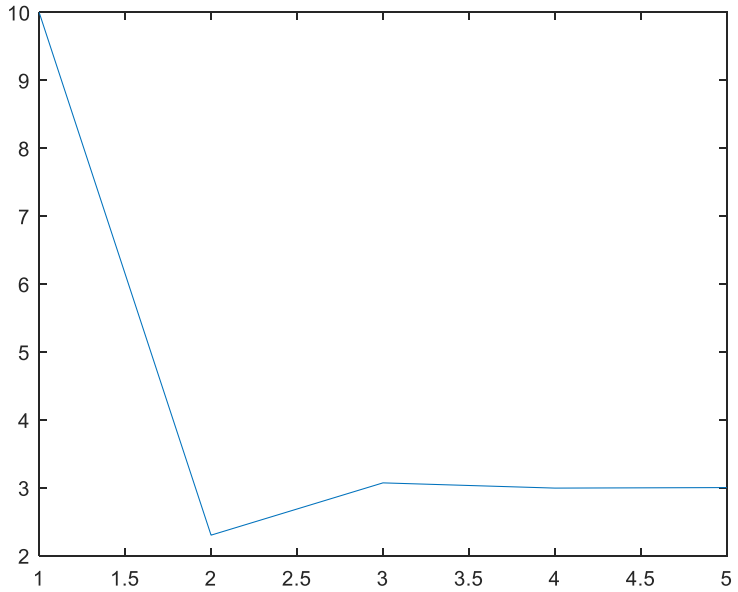
$$x_{k+1} = Gx_k + c$$

$1.1x = 3.3$

$x = -0.1x + 3.3$

$x_{k+1} = -0.1x_k + 3.3$

```
%1.1x=3.3
%x=-0.1x+3.3
x=10;
out=[];
for i=1:100
   out=[out;x];
   x=-0.1*x+3.3;
end;
plot(out);
```



# Przykład 2

```
A=[1.1 0.2;0.3 1.2]

A =

    1.1000    0.2000
    0.3000    1.2000
```

```
>> G=[-0.1 -0.2;-0.3 -0.2]

G =

   -0.1000   -0.2000
   -0.3000   -0.2000
```

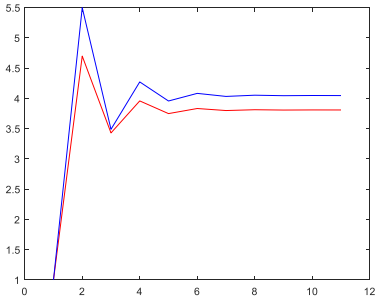
```
>> I-G

ans =

    1.1000    0.2000
    0.3000    1.2000
```

$$Ax = b \quad \Longleftrightarrow \quad x = Gx + c$$

?

$$x_{k+1} = Gx_k + c$$

```
b=[5 6]'

b =

     5
     6
```




```
x=[1 1]';
Out=x';
for i=1:10
   x=G*x+b;
   Out=[Out;x'];
end;
plot(Out(:,1),'r-');
hold on;
plot(Out(:,2),'b-');
```

$$Ax = b$$
$$A = LU$$
$$LUx = b$$

# Metody iteracyjne

$$Ax = b \quad \Longleftrightarrow \quad x = Gx + c$$

$$x_{k+1} = Gx_k + c$$

**Warunek zbieżności: Promień spektralny** $\rho(G) < 1$

Największa co do modułu wartość własna macierzy.

$$A = M - N \quad \rightarrow \quad Mx=Nx+b$$

$$x=M^{-1}Nx+M^{-1}b \qquad G = M^{-1}N \qquad c = M^{-1}b$$

$$x_{k+1} = M^{-1}Nx_k + M^{-1}b$$

$$Mx_{k+1} = Nx_k + b$$

Metody iteracyjne stosowane są głównie do rozwiązywania układów o **dużej liczbie równań** i niewiadomych (nawet rzędu milionów)

# Metoda Jakobiego

$$Ax = b \qquad A = M - N$$

Dodatkowe warunki $M = D, \qquad N = -(L + U)$

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

$$D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix}$$

$$N = \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}$$

$$x = M^{-1}Nx + M^{-1}b$$

$$x^{(k+1)} = D^{-1}\left(b - (L+U)x^{(k)}\right)$$

$$x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}}{a_{ii}}, \quad i = 1, \ldots, n$$

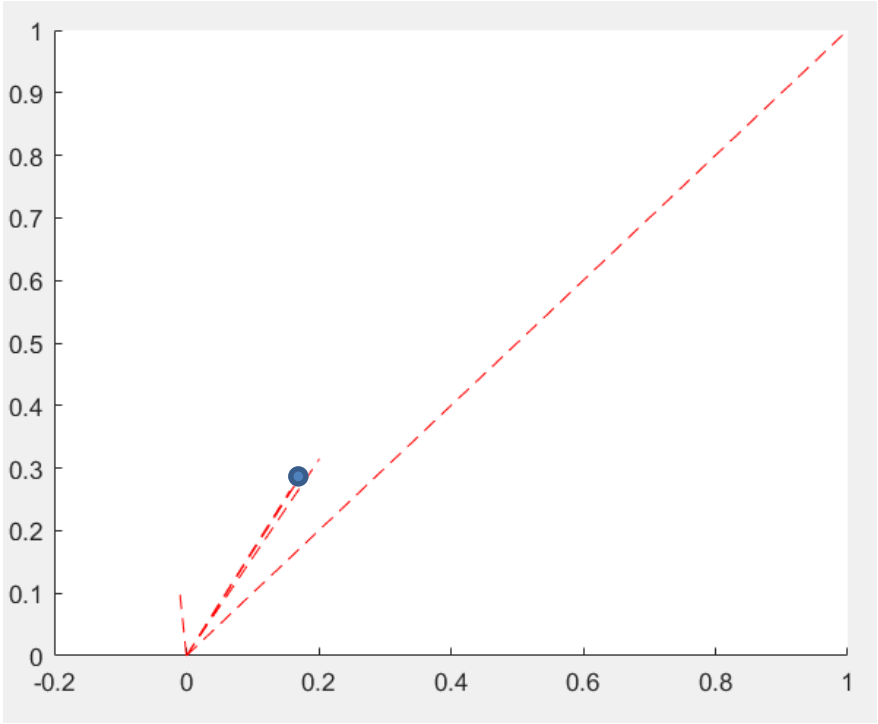
**Dostateczny warunek zbieżności: Macierz dominująca**

$$|a_{ii}| > \sum_{j \neq i} |a_{ij}|$$

Macierz dominująca to macierz, której wartości bezwzględne elementów na głównej przekątnej są większe od **sumy wartości** bezwzględnych pozostałych elementów w wierszach

# Przykład

```
%jakobi 2 x 2
 A=[10 1;1 9];b=[2 3]';
 %A^-1*b
 %0.1685
  %  0.3146
 %[L  D]=ldl(A);
 D=[A(1,1) 0;0 A(2,2)];
 U=[0 A(1,2);0 0];
 L=[0 0;A(2,1) 0];

 U=L';
 M=D;
 N=-(L+U);
 Out=[];
 x=[1 1]';
 Out=x;
  line([0 x(1)],[0 x(2)],'Color','red','LineStyle','--');
  hold on;
 for i=1:5
     x=M^-1*(b+N*x);
     line([0 x(1)],[0 x(2)],'Color','red','LineStyle','--');
     Out=[Out,x];
 end;
```




```
Out =

    1.0000    0.1000    0.1778    0.1678    0.1686    0.1685
    1.0000    0.2222    0.3222    0.3136    0.3147    0.3146
```

# Algorytm

Choose an initial guess $x^{(0)}$ to the solution

$k = 0$

while convergence not reached do

for i := 1 step until n do

$\sigma = 0$

for j := 1 step until n do

if j ≠ i then

$\sigma = \sigma + a_{ij} x_j^{(k)}$

end if

end (j-loop)

$x_i^{(k+1)} = \frac{(b_i - \sigma)}{a_{ii}}$

$$x_i^{(k+1)} = \frac{b_i - \sum_{j \neq i} a_{ij} x_j^{(k)}}{a_{ii}}, \quad i = 1, \ldots, n$$

end (i-loop)

check if convergence is reached

$k = k + 1$

loop (while convergence condition not reached)

# Przykład

$$\begin{aligned} 4x_1 - x_2 - x_3 &= 3 \\ -2x_1 + 6x_2 + x_3 &= 9 \\ -x_1 + x_2 + 7x_3 &= -6 \end{aligned}$$

$$\begin{aligned} 4x_1^{(k+1)} - x_2^{(k)} - x_3^{(k)} &= 3 \\ -2x_1^{(k)} + 6x_2^{(k+1)} + x_3^{(k)} &= 9 \\ -x_1^{(k)} + x_2^{(k)} + 7x_3^{(k+1)} &= -6 \end{aligned}$$

$\boldsymbol{x}^{(0)} = (0, 0, 0)$.

$$\begin{aligned} 4x_1^{(1)} - 0 - 0 &= 3 \\ -2 \cdot 0 + 6x_2^{(1)} + 0 &= 9 \\ -0 + 0 + 7x_3^{(1)} &= -6 \end{aligned} .$$

$\boldsymbol{x}^{(1)} = (x_1^{(1)}, x_2^{(1)}, x_3^{(1)}) = (3/4, 9/6, -6/7) \approx (0.750, 1.500, -0.857)$

# Ocena błędów

| $k$ | $x^{(k)}$ | | | $x^{(k)} - x^{(k-1)}$ | | | $e^{(k)} = x - x^{(k)}$ | | | $\|e^{(k)}\|$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.000 | 0.000 | 0.000 | – | – | – | 0.000 | 0.000 | -1.000 | 2.449 |
| 1 | 0.750 | 1.500 | -0.857 | 0.000 | 0.000 | -0.857 | 0.250 | 0.500 | -0.143 | 0.557 |
| 2 | 0.911 | 1.893 | -0.964 | 0.161 | 0.393 | -0.107 | 0.089 | 0.107 | -0.036 | 0.144 |
| 3 | 0.982 | 1.964 | -0.997 | 0.071 | 0.071 | -0.033 | 0.018 | 0.036 | -0.003 | 0.040 |
| 4 | 0.992 | 1.994 | -0.997 | 0.010 | 0.029 | 0.000 | 0.008 | 0.006 | -0.003 | 0.011 |
| 5 | 0.999 | 1.997 | -1.000 | 0.007 | 0.003 | -0.003 | 0.001 | 0.003 | 0.000 | 0.003 |
| 6 | 0.999 | 2.000 | -1.000 | 0.000 | 0.003 | 0.001 | 0.000 | 0.001 | 0.000 | 0.001 |
| 7 | 1.000 | 2.000 | -1.000 | 0.001 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |
| 8 | 1.000 | 2.000 | -1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

# Zadanie

$$\boldsymbol{Ax = b}$$

$$A = \begin{bmatrix} 2 & 1 \\ 5 & 7 \end{bmatrix}, \; b = \begin{bmatrix} 11 \\ 13 \end{bmatrix}$$

$$x^{(0)} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Obliczyć 2 iteracji przez metodę Jakobiego

$$\begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,2} & \dots & a_{n,n} \end{bmatrix} = \begin{bmatrix} 0 & 0 & \dots & 0 \\ a_{2,1} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ a_{n,1} & a_{n,2} & \dots & 0 \end{bmatrix} + \begin{bmatrix} a_{1,1} & 0 & \dots & 0 \\ 0 & a_{2,2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{n,n} \end{bmatrix} + \begin{bmatrix} 0 & a_{1,2} & \dots & a_{1,n} \\ 0 & 0 & \dots & a_{2,n} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 0 \end{bmatrix}$$

# Rozwiązanie

$$D^{-1} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1/7 \end{bmatrix}, \; L = \begin{bmatrix} 0 & 0 \\ 5 & 0 \end{bmatrix} \quad \text{and} \quad U = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}.$$

$$\boldsymbol{x}^{(k+1)} = \boldsymbol{D}^{-1}(\boldsymbol{b} - (\boldsymbol{L} + \boldsymbol{U})\boldsymbol{x}^{(k)})$$

# Przykład

```
A=[10 2 3;1 8  4;1 2 7 ];
D=[10 0 0;0 8 0;0 0 7];
L=[0 0 0;1 0 0;1 2 0];
U=[0 2 3;0 0 4;0 0 0];
b=[6 7 8]';
%metoda bezposrednia
%ldl - dla macierzy symetrycznej
%lu(A) - funkcja inv(A)
%help inv
%help \ - A\b

x=inv(A)*b;
x1=A\b;

%metoda iteracyjna
x=[1 1 1]';
x=inv(D)*(b-(L+U)*x)
```

# Metoda Gausa-Seidla

$Ax = b$ $\xrightarrow{A = M - N}$ ***Mx=Nx+b***

Metoda Jakobiego

$$M = D, \qquad N = -(L + U)$$

***$Mx_{k+1}=Nx_k+b$***

$$x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)})$$

Niech teraz $$M = D + L, \qquad N = -U$$

***$Dx_{k+1}=-Lx_{k+1}-Ux_k+b$***

***$(D+L)x_{k+1}=-Ux_k+b$***

$$x^{(k+1)} = D^{-1}\left(b - Lx^{(k+1)} - Ux^{(k)}\right)$$

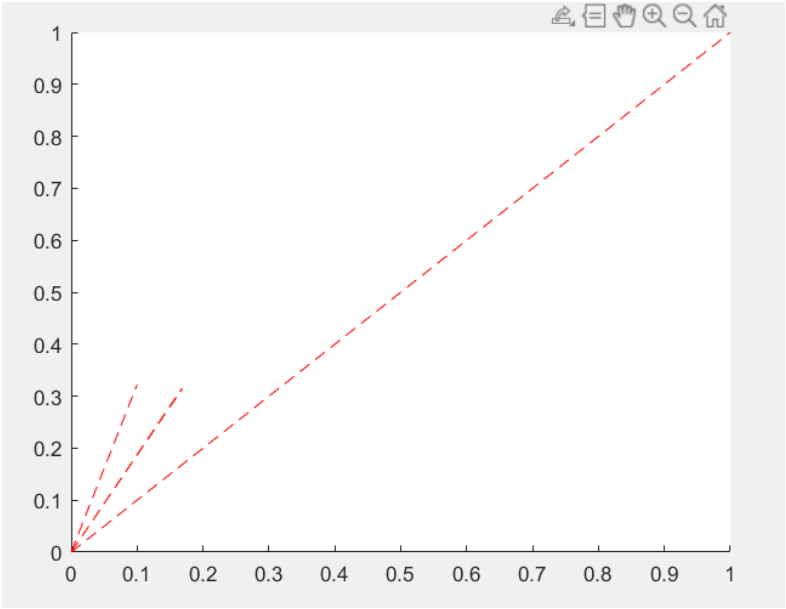
$$x^{(k+1)} = (D + L)^{-1}\left(b - Ux^{(k)}\right)$$

**Dostateczny warunek zbieżności:**
**Macierz dominująca**

$$|a_{ii}| \geq \sum_{j \neq i} |a_{ij}|$$

# Przykład

```
%gauss seidel 2 x 2
 A=[10 1;1 9];b=[2 3]';
 %A^-1*b
 %0.1685
  %  0.3146
 %[L  D]=ldl(A);
  D=[A(1,1) 0;0 A(2,2)];
 U=[0 A(1,2);0 0];
 L=[0 0;A(2,1) 0];
 U=L';
 M=D;
 N=-(L+U);
 Out=[];
 x=[1 1]';
 Out=x;
  line([0 x(1)],[0 x(2)],'Color','red','LineStyle','--');
  hold on;
for i=1:5
    x=(D+L)^-1*(b-U*x);
    line([0 x(1)],[0 x(2)],'Color','red','LineStyle','--');
    Out=[Out,x];
end;
```




GS

```
Out =

    1.0000    0.1000    0.1678    0.1685    0.1685    0.1685
    1.0000    0.3222    0.3147    0.3146    0.3146    0.3146
```

Jakobi

```
Out =

    1.0000    0.1000    0.1778    0.1678    0.1686    0.1685
    1.0000    0.2222    0.3222    0.3136    0.3147    0.3146
```

# Algorytm

*Wybierz początkowe przybliżenie* $\mathbf{x}^{(0)}$

**for** k := 1 **step** 1 **until** *oczekiwane przybliżenie* **do**

**for** i := 1 **step** 1 **until** n **do**

$\sigma = 0$

**for** j := 1 **step** 1 **until** i-1 **do**

$\sigma = \sigma + a_{ij} x_j^{(k)}$

**end** (j-for)

**for** j := i+1 **step** 1 **until** n **do**

$\sigma = \sigma + a_{ij} x_j^{(k-1)}$

**end** (j-for)

$x_i^{(k)} = \frac{(b_i - \sigma)}{a_{ii}}$

**end** (i-for)

*sprawdź, czy osiągnięto oczekiwane przybliżenie*

**end** (k-for)

$\mathbf{x} \approx \mathbf{x}^{(k)}$

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}\left(\mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)}\right)$$

# Metoda Gausa-Seidla (cd.)

1 krok
2 krok
...

$$
\begin{aligned}
a_{11}x_1^{(k+1)} &+ a_{12}x_2^{(k)} + \cdots + a_{1n}x_n^{(k)} = b_1 \\
a_{21}x_1^{(k+1)} &+ a_{22}x_2^{(k+1)} + \cdots + a_{2n}x_n^{(k)} = b_2 \\
&\vdots \qquad\quad \vdots \qquad\quad \ddots \qquad\quad \vdots \qquad\quad \vdots \\
a_{n1}x_1^{(k+1)} &+ a_{n2}x_2^{(k+1)} + \cdots + a_{nn}x_n^{(k+1)} = b_n
\end{aligned}
$$

$$
\begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ a_{21} & a_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ a_{n1} & \cdots & a_{n\,n-1} & a_{nn} \end{bmatrix}
\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{bmatrix}
+
\begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ 0 & 0 & \ddots & \vdots \\ \vdots & \ddots & \ddots & a_{n-1\,n} \\ 0 & \cdots & 0 & 0 \end{bmatrix}
\begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_n^{(k)} \end{bmatrix}
=
\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}.
$$

$$(L+D)\boldsymbol{x}^{(k+1)} + U\boldsymbol{x}^{(k)} = \boldsymbol{b}$$

$$\boldsymbol{x}^{(k+1)} = (L+D)^{-1}[-U\boldsymbol{x}^{(k)} + \boldsymbol{b}].$$

# Przykład

$$
\begin{aligned}
4x_1 - x_2 - x_3 &= 3 \\
-2x_1 + 6x_2 + x_3 &= 9 \\
-x_1 + x_2 + 7x_3 &= -6
\end{aligned}
$$

$$
\begin{aligned}
4x_1^{(k+1)} - x_2^{(k)} - x_3^{(k)} &= 3 \\
-2x_1^{(k+1)} + 6x_2^{(k+1)} + x_3^{(k)} &= 9 \\
-x_1^{(k+1)} + x_2^{(k+1)} + 7x_3^{(k+1)} &= -6
\end{aligned}
$$

$\boldsymbol{x}^{(0)} = (0, 0, 0)$.

$$
\begin{aligned}
4x_1^{(1)} - 0 - 0 &= 3 \\
-2x_1^{(1)} + 6x_2^{(1)} + 0 &= 9 \\
-x_1^{(1)} + x_2^{(1)} + 7x_3^{(1)} &= -6
\end{aligned}
$$

$x_1^{(1)} = 3/4 = 0.750.$

$x_2^{(1)} = [9 + 2(0.750)] / 6 = 1.750$

$x_3^{(1)} = [-6 + 0.750 - 1.750] / 7 = -1.000$

$\mathbf{x}^{(1)} = (x1(1), x2(1), x3(1)) = (0.750, 1.750, -1.000)$

# Ocena błędów

| $k$ | $x^{(k)}$ | | | $x^{(k)} - x^{(k-1)}$ | | | $e^{(k)} = x - x^{(k)}$ | | | $\|e^{(k)}\|$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.000 | 0.000 | 0.000 | – | – | – | 0.000 | 0.000 | -1.000 | 2.449 |
| 1 | 0.750 | 1.750 | -1.000 | 0.000 | 0.000 | -1.000 | 0.250 | 0.250 | 0.000 | 0.354 |
| 2 | 0.938 | 1.979 | -1.006 | 0.188 | 0.229 | -0.006 | 0.063 | 0.021 | 0.006 | 0.066 |
| 3 | 0.993 | 1.999 | -1.001 | 0.056 | 0.020 | 0.005 | 0.007 | 0.001 | 0.001 | 0.007 |
| 4 | 0.999 | 2.000 | -1.000 | 0.006 | 0.001 | 0.001 | 0.001 | 0.000 | 0.000 | 0.001 |
| 5 | 1.000 | 2.000 | -1.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 | 0.000 |

# Zadanie

Rozwiązać układ za pomocą metody Gausa-Seidla

$$\begin{cases} x_1 + 6x_2 + x_3 = 9 \\ 4x_1 - x_2 + x_3 = 4 \\ -x_1 + 2x_2 + 5x_3 = 2 \end{cases}$$

# Rozwiązanie

$$x_1 + 6x_2 + x_3 = 9$$
$$4x_1 - x_2 + x_3 = 4$$
$$-x_1 + 2x_2 + 5x_3 = 2$$

$$4x_1 - x_2 + x_3 = 4$$
$$x_1 + 6x_2 + x_3 = 9$$
$$-x_1 + 2x_2 + 5x_3 = 2$$

Macierz dominująca

$$x_1 = \frac{1}{4}(4 + x_2 - x_3)$$
$$x_2 = \frac{1}{6}(9 - x_1 - x_3)$$
$$x_3 = \frac{1}{5}(2 + x_1 - 2x_2)$$

$x_2 = 0$ i $x_3 = 0$

$$x_1 = \frac{1}{4}(4 + 0 - 0) = 1$$
$$x_2 = \frac{1}{6}(9 - 1 - 0) = 4/3 \approx 1.333$$
$$x_3 = \frac{1}{5}(2 + 1 - 2 \times 4/3) = 1/15 \approx 0.0667$$
$$x_1 = \frac{1}{4}(4 + 4/3 - 1/15) = 79/60 \approx 1.317$$
$$x_2 = \frac{1}{6}(9 - 79/60 - 1/15) = 457/360 \approx 1.269$$
$$x_3 = \frac{1}{5}(2 + 79/60 - 2 \times 457/360) = 7/45 \approx 0.1556$$

$x_1 = 23/18 \approx 1.278$, $x_2 = 53/42 \approx 1.262$, $x_3 = 19/126 \approx 0.1508$

# Zbieżność metod iteracyjnych

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

$$Mx^{(k+1)} = Nx^{(k)} + b$$

$$x^{(k+1)} = M^{-1}Nx^{(k)} + M^{-1}b$$

$$x^{(k+1)} = Bx^{(k)} + \tilde{b}$$

Dokładne rozwiązanie

$$B = M^{-1}N \text{ oraz } \tilde{b} = M^{-1}b$$

$$x = Bx + \tilde{b}$$

$$x - x^{(k+1)} = B\left(x - x^{(k)}\right)$$

$\boldsymbol{e}^{(k)} = \boldsymbol{x} - \boldsymbol{x}^{(k)}$

$$e^{(k)} = Be^{(k-1)} = B\left(Be^{(k-2)}\right) = B^2e^{(k-2)} = \mathrm{L} = B^ke^{(0)}$$

$||B|| < 1$

# Porównywanie

Metoda Jakobiego

$$x^{(k+1)} = D^{-1}(b - (L + U)x^{(k)})$$

```
>> eig(M^-1*N)

ans =

    0.1054
   -0.1054
```

Metoda Gausa-Seidla

$$x^{(k+1)} = (D + L)^{-1}(b - Ux^{(k)}),$$

```
>> eig((D+L)^-1*U)

ans =

         0
   -0.0111
```

# Porównywanie (cd.)

$$\boldsymbol{x}^{(k+1)} = B\boldsymbol{x}^{(k)} + \widetilde{\boldsymbol{b}},$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

| Method | $B$ |
|---|---|
| Jacobi | $D^{-1}(-L-U) = \begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix}^{-1} \begin{bmatrix} 0 & -a_{12} \\ -a_{21} & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 \end{bmatrix}$ |
| GS | $(L+D)^{-1}(-U) = \begin{bmatrix} a_{11} & 0 \\ a_{21} & a_{22} \end{bmatrix}^{-1} \begin{bmatrix} 0 & -a_{12} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} \\ 0 & \frac{a_{12}a_{21}}{a_{11}a_{22}} \end{bmatrix}$ |

| Method | Eigenvalues | Eigenvectors |
|---|---|---|
| Jacobi | $\lambda_1 = \sqrt{\frac{a_{12}a_{21}}{a_{11}a_{22}}},\ \lambda_2 = -\sqrt{\frac{a_{12}a_{21}}{a_{11}a_{22}}}$ | $\boldsymbol{v}_1 = \begin{bmatrix} 1 \\ -\sqrt{\frac{a_{11}a_{21}}{a_{12}a_{22}}} \end{bmatrix},\ \boldsymbol{v}_1 = \begin{bmatrix} 1 \\ \sqrt{\frac{a_{11}a_{21}}{a_{12}a_{22}}} \end{bmatrix}$ |
| GS | $\lambda_1 = 0,\ \lambda_2 = -\frac{a_{12}a_{21}}{a_{11}a_{22}}$ | $\boldsymbol{v}_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix},\ \boldsymbol{v}_2 = \begin{bmatrix} 1 \\ -\frac{a_{21}}{a_{22}} \end{bmatrix}$ |

$$||B_{GS}|| = ||B_{Jacobi}||^2$$

# GS a SOR

### 1. **Krytyka klasycznej metody Gaussa-Seidela**

Metoda Gaussa-Seidela (GS) jest zbieżna dla macierzy diagonalnie dominujących lub symetrycznych dodatnio określonych, ale często **zbiega wolno**.

- **Problem**: W GS nowe przybliżenie $x^{(k+1)}$ jest całkowicie zastępowane przez wartość z aktualnej iteracji, co może prowadzić do "przesuwania się" rozwiązania małymi krokami.
- **Rozwiązanie**: Wprowadzenie **kontrolowanego kroku relaksacji**, który przyspiesza lub spowalnia zmianę zmiennych.

---

### 2. **Inspiracja: Metoda gradientu prostego**

Metoda SOR jest analogią do **optymalnego doboru kroku w metodzie gradientu prostego**:

$$x^{(k+1)} = x^{(k)} + \alpha \cdot \text{kierunek poprawy}$$

- W SOR "kierunkiem poprawy" jest różnica między iteracją GS a poprzednim przybliżeniem:

$$\text{kierunek} = (x_{\text{GS}}^{(k+1)} - x^{(k)})$$

- Parametr $\omega$ pełni rolę **optymalnego kroku** $\alpha$.

# Metoda SOR (nad relaksacji)
# Successive Overrelaxation (SOR)

Metoda Gausa-Seidla

$$x^{(k+1)} = D^{-1}\left[b - Lx^{(k+1)} - Ux^{(k)}\right]$$

$$Dx^{(k+1)} = b - Lx^{(k+1)} - Ux^{(k)}$$

$$\Rightarrow \quad x^{(k+1)} - x^{(k)} = D^{-1}\left[b - Lx^{(k+1)} - Dx^{(k)} - Ux^{(k)}\right]$$

Korekcja:

Zbieżność metody Gaussa–Seidela można przyspieszyć, wprowadzając parametr relaksacji i kolejne współrzędne nowego przybliżenia wyznaczać, kombinując ze sobą poprzednie przybliżenie oraz współrzędną nowego przybliżenia uzyskanego metodą Gaussa–Seidela:

$$x_i^{(k+1)} = (1-\omega)x_i^{(k)} + \omega\tilde{x}_i^{k+1}$$

$$x^{(k+1)} = x^{(k)} + \omega\left(x^{(k+1)} - x^{(k)}\right)_{GS}$$

$1 < \omega < 2$

Gdy *ω=1 mamy GS*

$$\left(x^{(k+1)} - x^{(k)}\right)_{GS} = D^{-1}\left[b - Lx^{(k+1)} - Dx^{(k)} - Ux^{(k)}\right]$$

$$\Rightarrow \quad x^{(k+1)} = x^{(k)} + \omega D^{-1}\left[b - Lx^{(k+1)} - Dx^{(k)} - Ux^{(k)}\right]$$

# Zbieżność

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

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

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

```
>> eig(M^-1*N)

ans =

    0.1054
   -0.1054
```

$$\left\| \left( L + \frac{1}{\omega} D \right)^{-1} \left( \frac{1}{\omega} D - D - U \right) \right\|$$

$Jacobi \quad D^{-1}(-L-U)$

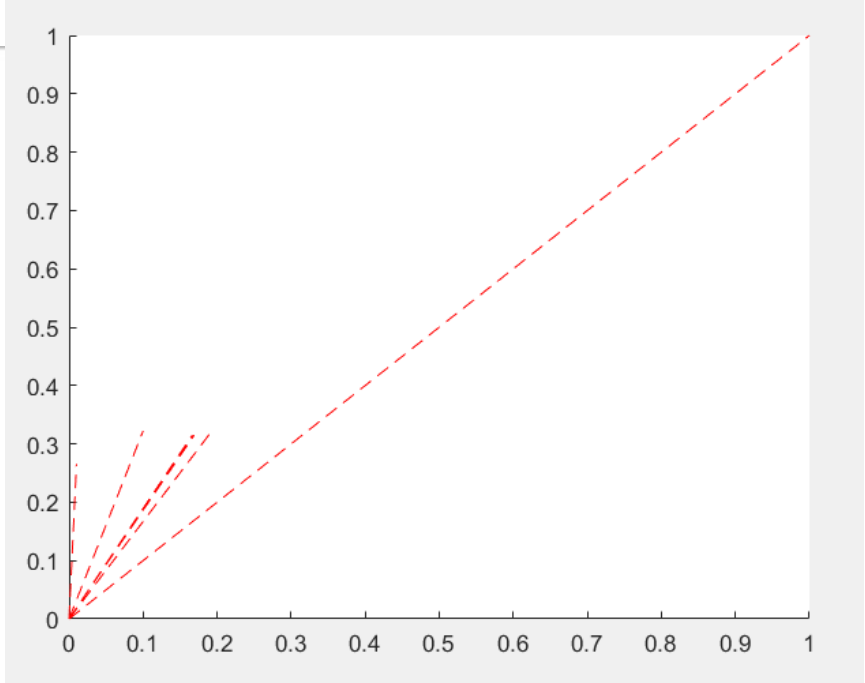
$GS \quad (L+D)^{-1}(-U)$

```
>> eig((D+L)^-1*U)

ans =

         0
   -0.0111
```

```
>> w=1.01

w =

    1.0100

>> eig((1/w*D+L)^-1*(1/w*D-D-U))

ans =

  -0.0043 + 0.0090i
  -0.0043 - 0.0090i
```

# Przykład

```
%SOR 2 x 2
A=[10 1;1 9];b=[2 3]';
%A^-1*b
%0.1685
 %  0.3146
%[L  D]=ldl(A);
  D=[A(1,1) 0;0 A(2,2)];
U=[0 A(1,2);0 0];
L=[0 0;A(2,1) 0];

U=L';
M=D;
N=-(L+U);
w=1.1;
Out=[];
x=[1 1]';
Out=x;
 line([0 x(1)],[0 x(2)],'Color','red','LineStyle','--');
 hold on;
for i=1:5
    x=(1/w*D+L)^-1*(b+(1/w*D-D-U)*x);

    line([0 x(1)],[0 x(2)],'Color','red','LineStyle','--');
    Out=[Out,x];
end;
```




```
Out =

    1.0000    0.0100    0.1898    0.1662    0.1688    0.1685
    1.0000    0.2654    0.3169    0.3147    0.3146    0.3146
```

# Porównywanie

Metoda Jakobiego

$$x^{(k+1)} = D^{-1}\left(b - (L+U)x^{(k)}\right)$$

Metoda Gausa-Seidla

$$x^{(k+1)} = (D+L)^{-1}\left(b - Ux^{(k)}\right)$$

Metoda SOR

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

# Tabela algorytmów iteracyjnych

Metoda Jakobiego

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

Metoda Gausa-Seidla

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

Metoda SOR

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

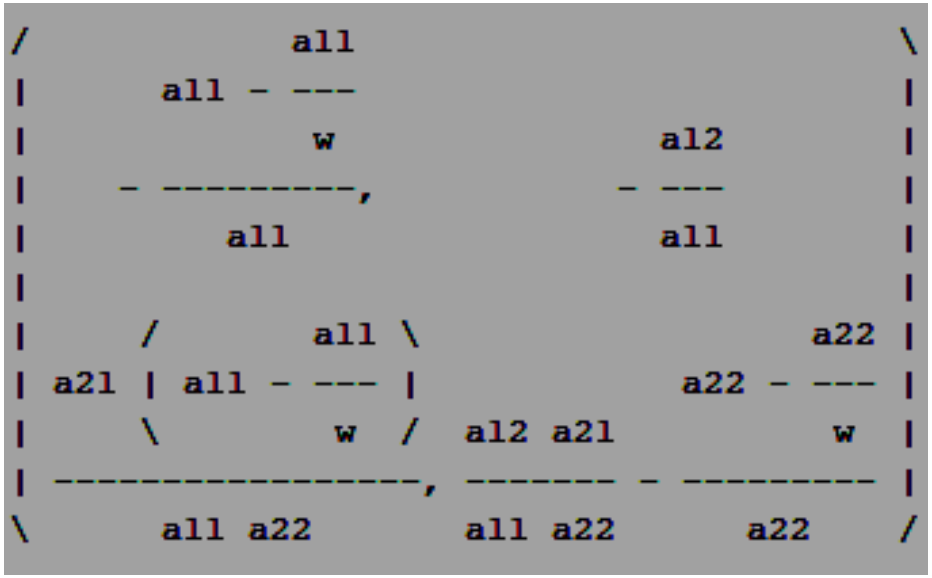
# Zbieżność

$$\boldsymbol{x}^{(k+1)} = B\boldsymbol{x}^{(k)} + \widetilde{\boldsymbol{b}}\,,$$

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

| Method | $B$ |
|---|---|
| Jacobi | $D^{-1}(-L-U) = \begin{bmatrix} a_{11} & 0 \\ 0 & a_{22} \end{bmatrix}^{-1} \begin{bmatrix} 0 & -a_{12} \\ -a_{21} & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 \end{bmatrix}$ |
| GS | $(L+D)^{-1}(-U) = \begin{bmatrix} a_{11} & 0 \\ a_{21} & a_{22} \end{bmatrix}^{-1} \begin{bmatrix} 0 & -a_{12} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} \\ 0 & \frac{a_{12}a_{21}}{a_{11}a_{22}} \end{bmatrix}$ |

SOR $\left(L+\frac{1}{\omega}D\right)^{-1}\left(\frac{1}{\omega}D-D-U\right)=$



```
syms a11 a21 a22 a12 w
D=[a11 0;0 a22]
L=[0 0;a21 0]
U=[0 a12; 0 0]
SOR=(L+D)^-1*(1/w*D-D-U)
simplify(SOR)
pretty(SOR)
[Vs Ls]=eig(SOR)
pretty(Ls)
pretty(Vs)
```

# Przykład

$$\begin{cases} 4x_1 - x_2 = 2 \\ -x_1 + 4x_2 - x_3 = 6 \\ -x_2 + 4x_3 = 2 \end{cases}$$

$\mathbf{x}=[1, 2, 1]$

$\mathbf{x}^{(1)} = \mathbf{0}$

# Przykład (cd.)

Metoda Jakobiego

$$\mathbf{x}^{(i+1)} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{x}^{(i)} + \mathbf{D}^{-1}\mathbf{b}$$

$$\mathbf{x}^{(1)} = \mathbf{0}$$

$$\mathbf{x}^{(2)} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{x}^{(1)} + \mathbf{D}^{-1}\mathbf{b} =$$

$$= -\begin{bmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} 2 \\ 6 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 1.5 \\ 0.5 \end{bmatrix},$$

$$\mathbf{x}^{(3)} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{x}^{(2)} + \mathbf{D}^{-1}\mathbf{b} =$$

$$= -\begin{bmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{2} \\ \frac{3}{2} \\ \frac{1}{2} \end{bmatrix} + \begin{bmatrix} \frac{1}{4} & 0 & 0 \\ 0 & \frac{1}{4} & 0 \\ 0 & 0 & \frac{1}{4} \end{bmatrix} \begin{bmatrix} 2 \\ 6 \\ 2 \end{bmatrix} = \begin{bmatrix} 0.875 \\ 1.75 \\ 0.875 \end{bmatrix},$$

$$\mathbf{x}^{(4)} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{x}^{(3)} + \mathbf{D}^{-1}\mathbf{b} = [0.9375,\ 1.9375,\ 0.9375],$$

$$\mathbf{x}^{(5)} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U})\mathbf{x}^{(4)} + \mathbf{D}^{-1}\mathbf{b} = [0.9844,1.9688,0.9844].$$

# Przykład (cd.)

Metoda Gaussa-Seidela

$$\mathbf{x}^{(i+1)} = -(\mathbf{D}+\mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(i)} + (\mathbf{D}+\mathbf{L})^{-1}\mathbf{b}$$

$$\mathbf{x}^{(1)} = \mathbf{0}$$

$$\mathbf{x}^{(2)} = -(\mathbf{D}+\mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(1)} + (\mathbf{D}+\mathbf{L})^{-1}\mathbf{b} = [0.5000,\ 1.6250,\ 0.9062],$$

$$\mathbf{x}^{(3)} = -(\mathbf{D}+\mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(2)} + (\mathbf{D}+\mathbf{L})^{-1}\mathbf{b} = [0.9062,\ 1.9531,\ 0.9883],$$

$$\mathbf{x}^{(4)} = -(\mathbf{D}+\mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(3)} + (\mathbf{D}+\mathbf{L})^{-1}\mathbf{b} = [0.9883,\ 1.9941,\ 0.9985],$$

$$\mathbf{x}^{(5)} = -(\mathbf{D}+\mathbf{L})^{-1}\mathbf{U}\mathbf{x}^{(4)} + (\mathbf{D}+\mathbf{L})^{-1}\mathbf{b} = [0.9885,\ 1.9993,\ 0.9998].$$

# Przykład (cd.)

SOR

$$\mathbf{x}^{(i+1)} == (\mathbf{D}+\omega\mathbf{L})^{-1}((1-\omega)\mathbf{D}-\omega\mathbf{U})\mathbf{x}^{(i)} + \omega(\mathbf{D}+\omega\mathbf{L})^{-1}\mathbf{b}$$

**ω=1.2** otrzymujemy kolejno:

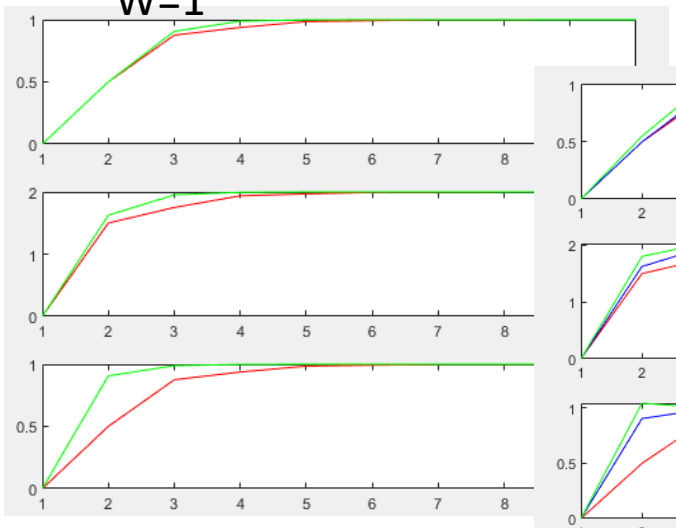
$\mathbf{x}^{(2)}$= [0.6000, 1.9800, 1.1940],
$\mathbf{x}^{(3)}$= [1.0740, 2.0844, 0.9865],
$\mathbf{x}^{(4)}$= [1.0105, 1.9822, 0.9974],
$\mathbf{x}^{(5)}$= [0.9926, 2.0005, 1.0007],

natomiast dla **ω=1.1** mamy:

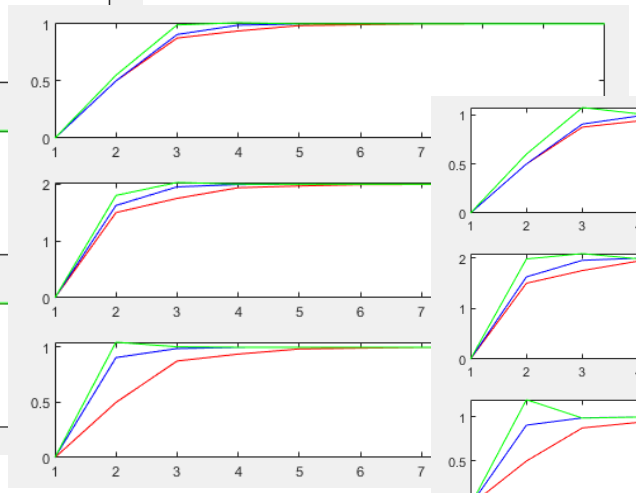
$\mathbf{x}^{(2)}$= [0.5500, 1.8013, 1.0453],
$\mathbf{x}^{(3)}$= [0.9903, 2.0297, 1.0036],
$\mathbf{x}^{(4)}$= [1.0091, 2.0005, 0.9998],
$\mathbf{x}^{(5)}$= [0.9998, 1.9997, 0.9999].

# Iter2.m

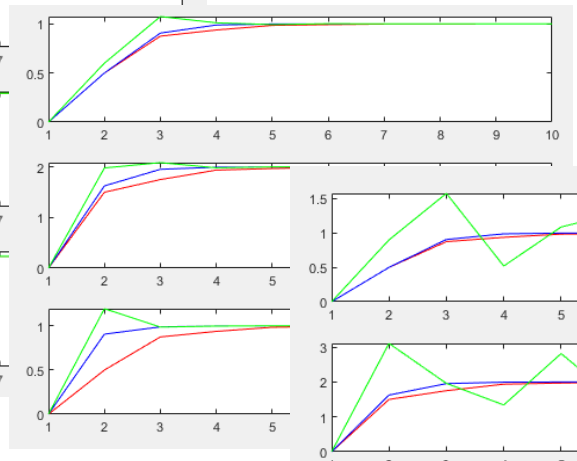
W=1



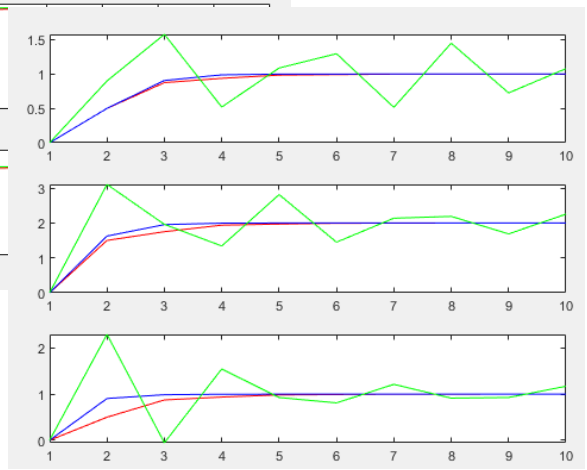
W=1.1



W=1.2



W=1.8



Rozważmy prosty układ 2×2:

$$\begin{cases} 4x + y = 7 \\ x + 3y = 5 \end{cases}$$

Macierz tego układu jest **diagonalnie dominująca**, więc zarówno Gauss-Seidel (GS), jak i SOR będą zbieżne.

**Rozwiązanie dokładne (analityczne):**

$$x = 1.6, \quad y = 1.1$$

### Krok 1: Metoda Gaussa-Seidela (GS)

Wzory iteracyjne GS:

$$x^{(k+1)} = \frac{7 - y^{(k)}}{4}, \quad y^{(k+1)} = \frac{5 - x^{(k+1)}}{3}$$

**Start:** $x^{(0)} = 0, y^{(0)} = 0$

**Wyniki po 5 iteracjach:**

| Iteracja (k) | $x^{(k)}$ | $y^{(k)}$ | Błąd ($\|x^{(k)} - 1.6\|$) |
|---|---|---|---|
| 1 | 1.750 | 1.083 | 0.150 |
| 2 | 1.479 | 1.174 | 0.121 |
| 3 | 1.706 | 1.098 | 0.106 |
| 4 | 1.525 | 1.158 | 0.075 |
| 5 | 1.660 | 1.113 | 0.060 |

Zbieżność jest monotoniczna, ale wolna (błąd maleje liniowo).

### Krok 2: Metoda SOR z różnymi $\omega$

Wzór SOR dla $x$:

$$x^{(k+1)} = x^{(k)} + \omega \left( \frac{7 - y^{(k)}}{4} - x^{(k)} \right)$$

Analogicznie dla $y$.

**Przypadek 1: Nadrelaksacja ($\omega = 1.2$)**

**Wyniki po 5 iteracjach:**

| Iteracja (k) | $x^{(k)}$ | $y^{(k)}$ | Błąd ($\|x^{(k)} - 1.6\|$) |
|---|---|---|---|
| 1 | 2.100 | 0.967 | 0.500 |
| 2 | 1.629 | 1.124 | 0.029 |
| 3 | 1.598 | 1.134 | 0.002 |
| 4 | 1.600 | 1.133 | 0.000 |
| 5 | 1.600 | 1.133 | 0.000 |

**Obserwacja:**

- Zbieżność **znacznie szybsza** niż GS – już po 3 iteracjach błąd jest minimalny!
- Dla $\omega = 1.2$ metoda osiąga rozwiązanie z dokładnością do 3 miejsc dziesiętnych w **4 iteracjach**, podczas gdy GS potrzebuje >10 iteracji.

**Przypadek 2: Podrelaksacja ($\omega = 0.8$)**

**Wyniki po 5 iteracjach:**

| Iteracja (k) | $x^{(k)}$ | $y^{(k)}$ | Błąd ($\|x^{(k)} - 1.6\|$) |
|---|---|---|---|
| 1 | 1.400 | 1.200 | 0.200 |
| 2 | 1.520 | 1.160 | 0.080 |
| 3 | 1.568 | 1.144 | 0.032 |
| 4 | 1.587 | 1.138 | 0.013 |
| 5 | 1.595 | 1.135 | 0.005 |

**Obserwacja:**

- Zbieżność **wolniejsza niż GS**, ale stabilniejsza (mniejsze "przeskoki" wartości).
- Przydatna np. dla układów źle uwarunkowanych, gdzie GS może oscylować.

### Krok 3: Optymalny wybór $\omega$

Dla macierzy diagonalnie dominujących **optymalne** $\omega$ można oszacować teoretycznie. W tym przypadku:

$$\omega_{\text{opt}} \approx \frac{2}{1 + \sqrt{1 - \rho(J)^2}}$$

gdzie $\rho(J)$ to promień spektralny macierzy iteracji Jacobiego. Dla naszego układu $\rho(J) \approx 0.25$, więc:

$$\omega_{\text{opt}} \approx 1.05$$

**Test dla $\omega = 1.05$:**

- Rozwiązanie osiągane z dokładnością do 5 miejsc dziesiętnych w **3 iteracjach**.

## Wnioski

1. **Przyspieszenie zbieżności**:
   - Dla $\omega = 1.2$ SOR osiąga błąd < 0.01 w **3 iteracjach**, podczas gdy GS potrzebuje **8 iteracji**.
2. **Dokładność**:
   - SOR z optymalnym $\omega$ redukuje błędy szybciej, minimalizując kumulację błędów zaokrągleń.
3. **Ryzyko**:
   - $\omega > 1.5$ może prowadzić do oscylacji lub rozbieżności.

# Metoda gradientowa

Rozpatrzmy rozwiązania poniższego układu równań:

Ax = b,

gdzie macierz A (n x n) jest **symetryczna, rzeczywista i dodatnio określona.**

Oznaczmy rozwiązanie tego układu przez x*.

$$A\mathbf{x} - \mathbf{b} = 0$$

$$F(\mathbf{x}) = \mathbf{x}^T A\mathbf{x} - 2\mathbf{x}^T\mathbf{b} \qquad F(\mathbf{x}) = \|A\mathbf{x} - \mathbf{b}\|^2$$

$$\nabla F(\mathbf{x}) = 2(A\mathbf{x} - \mathbf{b}) \qquad \nabla F(\mathbf{x}) = 2A^T(A\mathbf{x} - \mathbf{b})$$

# Metoda gradientowa (cd)

$$Ax^* = b$$

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

$$\nabla f(x) = Ax - b = 0$$

# Metoda gradientowa (cd)

$$p_k = -\nabla f(x_k) = r_k$$

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

$$\min_\alpha f(x_k + \alpha p_k)$$

# Metoda gradientowa (cd)

$$f(x_k + \alpha p_k) = \frac{1}{2}(x_k + \alpha p_k)^T A(x_k + \alpha p_k) - (x_k + \alpha p_k)^T b$$

$$= \frac{1}{2}\alpha^2 p_k^T A p_k + \alpha p_k^T A x_k + -\alpha p_k^T b + \text{constant}$$

$$\frac{\partial f}{\partial \alpha} = 0$$

$$p_k^T A x_k + \alpha p_k^T A p_k - p_k^T b = 0$$

$$\alpha = -\frac{p_k^T (A x_k - b)}{p_k^T A p_k} = \frac{p_k^T r_k}{p_k^T A p_k}$$

$$\alpha_k = \frac{p_k^T r_k}{p_k^T A p_k} = \frac{r_k^T r_k}{p_k^T A p_k}$$

# Algorytm

repeat in the loop:

$\mathbf{r} := \mathbf{b} - \mathbf{A}\mathbf{x}$

$\gamma := \mathbf{r}^\mathsf{T}\mathbf{r} / \mathbf{r}^\mathsf{T}\mathbf{A}\mathbf{r}$

$\mathbf{x} := \mathbf{x} + \gamma\mathbf{r}$

if $\mathbf{r}^\mathsf{T}\mathbf{r}$ is sufficiently small, then exit loop

end repeat loop

return $\mathbf{x}$ as the result

# Zbieżność

$$E(x) = \frac{1}{2}(x - x^*)^T A(x - x^*)$$

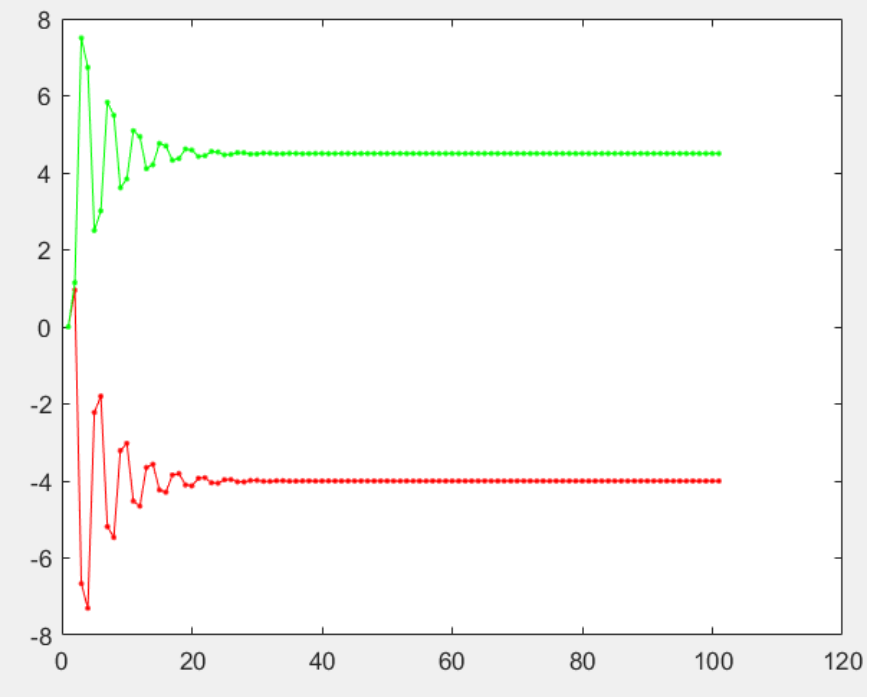
$$E(x_k) \leq \left( \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} \right)^{2k} E(x_0)$$

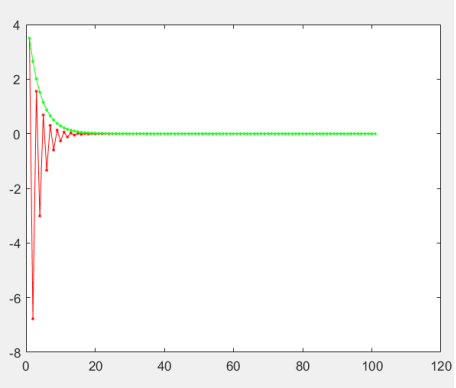
$\lambda_{max}$ and $\lambda_{min}$ are the largest and smallest eigenvalues of A

# Przykład

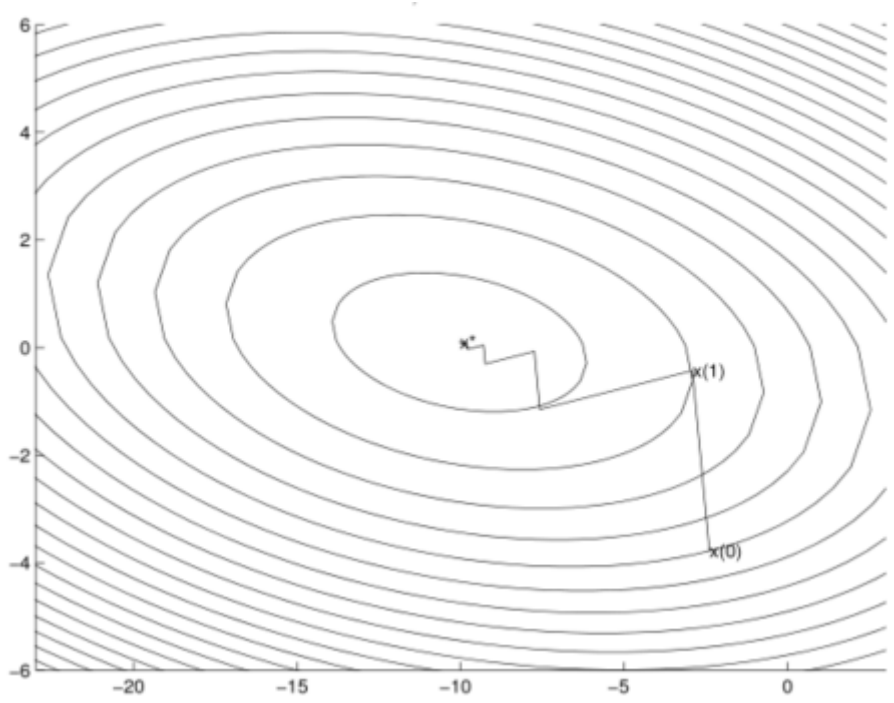
gradAxB.m

```
A=[1 2;3 4];
b=[5 6]';
%A^-1*b'
%   -4.0000
%    4.5000
xg=A\b;
x=[0 0]';
Ex0=1/2*(x-xg)'*A*(x-xg);
L=eig(A);
Lmax=L(2);
Lmin=L(1);
GranicaBledu=Ex0;
Out=[x' Ex0 GranicaBledu];
for i=1:100
    r=b-A*x;
    gamma=r'*r/(r'*A*r);
    x=x+gamma*r;
    Ex=1/2*(x-xg)'*A*(x-xg);
%GranicaBledu=((Lmax-Lmin)/(Lmax+Lmin))^(2*i)*Ex0;
GranicaBledu=((Lmax+Lmin)/(Lmax-Lmin))^(2*i)*Ex0;

    Out=[Out;x' Ex GranicaBledu];
end;
plot(Out(:,1),'.-r');
hold on;
plot(Out(:,2),'.-g');
figure;
plot(Out(:,3),'.-r');
hold on;
plot(Out(:,4),'.-g');
```



# Wizualizacja



# Metoda gradientu sprzężonego

Mówimy, że dwa niezerowe wektory ***u*** i **v** są sprzężone (względem ***A***), jeśli

$$\mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{v} = \mathbf{0}.$$

Ponieważ A jest symetryczna i dodatnio określona

$$\langle \mathbf{u}, \mathbf{v} \rangle_{\mathbf{A}} := \langle \mathbf{A}^{\mathrm{T}}\mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{A}\mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{u}, \mathbf{A}\mathbf{v} \rangle = \mathbf{u}^{\mathrm{T}}\mathbf{A}\mathbf{v}$$

Więc, dwa wektory są sprzężone jeśli są ortogonalne względem tego iloczynu skalarnego. Sprzężoność jest relacją symetryczną.

Przypuśćmy, że **{pk}** jest ciągiem n wzajemnie sprzężonych kierunków.

Dowolny wektor można wyrazić jako ważona suma liniowo niezależnych wektorów

$$\mathbf{x}_* = \sum_{i=1}^{n} \alpha_i \mathbf{p}_i$$

# Metoda gradientu sprzężonego (cd)

$$\mathbf{A}\mathbf{x}_* = \sum_{i=1}^{n} \alpha_i \mathbf{A}\mathbf{p}_i = \mathbf{b},$$

$$\mathbf{p}_k^{\mathrm{T}} \mathbf{A}\mathbf{x}_* = \sum_{i=1}^{n} \alpha_i \mathbf{p}_k^{\mathrm{T}} \mathbf{A}\mathbf{p}_i = \mathbf{p}_k^{\mathrm{T}} \mathbf{b},$$

(wg liniowej niezależności)

$$\alpha_k = \frac{\mathbf{p}_k^{\mathrm{T}} \mathbf{b}}{\mathbf{p}_k^{\mathrm{T}} \mathbf{A}\mathbf{p}_k} = \frac{\langle \mathbf{p}_k, \mathbf{b} \rangle}{\langle \mathbf{p}_k, \mathbf{p}_k \rangle_{\mathbf{A}}} = \frac{\langle \mathbf{p}_k, \mathbf{b} \rangle}{\|\mathbf{p}_k\|_{\mathbf{A}}^2}$$

Co daje nam następującą metodę rozwiązywania równania **Ax = b**. Najpierw znajdujemy ciąg n sprzężonych kierunków, następnie obliczamy współczynniki $\alpha_k$

# Metoda iteracyjna

$$f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x} - \mathbf{x}^{\mathrm{T}}\mathbf{b}, \quad \mathbf{x} \in \mathbf{R}^n \quad \mathbf{x}_0 = 0$$

$$\nabla f(x) = Ax - b = 0$$

jako pierwszy wektor bazowy **p1** wybrać gradient f w $x = x_0$, który wynosi $\mathbf{Ax_0 - b}$, a ponieważ wybraliśmy $x_0 = 0$, otrzymujemy **−b**. **Pozostałe wektory w bazie będą sprzężone do gradientu** (stąd nazwa metoda gradientu sprzężonego).

$$\mathbf{r}_k = \mathbf{b} - \mathbf{A}\mathbf{x}_k$$

Zauważmy, że $r_k$ jest przeciwny do gradientu f w $x = x_k$, więc metoda gradientu prostego nakazywałaby ruch w kierunku $r_k$. Tutaj jednak założyliśmy wzajemną sprzężoność kierunków $p_k$, więc wybieramy kierunek najbliższy do $r_k$ pod warunkiem sprzężoności. Co wyraża się wzorem:

$$\mathbf{p}_{k+1} = \mathbf{r}_k - \frac{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{r}_k}{\mathbf{p}_k^{\mathrm{T}}\mathbf{A}\mathbf{p}_k}\mathbf{p}_k$$

Ortogonalizacja Grama-Shcmidta

# Ortogonalizacja Grama-Schmidta

Ortogonalizacja Grama-Schmidta – przekształcenie układu liniowo niezależnych wektorów przestrzeni unitarnej w układ wektorów ortogonalnych.

A= $\mathbf{a}_1, \ldots, \mathbf{a}_N$

B= $\mathbf{b}_1, \ldots, \mathbf{b}_N$ $\qquad\qquad$ $\mathbf{e}_1, \ldots, \mathbf{e}_N$

$$\operatorname{proj}_{\mathbf{b}} \mathbf{a} = \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\langle \mathbf{b}, \mathbf{b} \rangle} \mathbf{b}$$

$$\begin{aligned}
\mathbf{b}_1 &= \mathbf{a}_1 && (1)\\
\mathbf{b}_2 &= \mathbf{a}_2 - \operatorname{proj}_{\mathbf{b}_1} \mathbf{a}_2 && (2)\\
\mathbf{b}_3 &= \mathbf{a}_3 - \operatorname{proj}_{\mathbf{b}_1} \mathbf{a}_3 - \operatorname{proj}_{\mathbf{b}_2} \mathbf{a}_3 && (3)\\
\mathbf{b}_4 &= \mathbf{a}_4 - \operatorname{proj}_{\mathbf{b}_1} \mathbf{a}_4 - \operatorname{proj}_{\mathbf{b}_2} \mathbf{a}_4 - \operatorname{proj}_{\mathbf{b}_3} \mathbf{a}_4 && (4)\\
&\vdots\\
\mathbf{b}_N &= \mathbf{a}_N - \sum_{j=1}^{N-1} \operatorname{proj}_{\mathbf{b}_j} \mathbf{a}_N && (N)
\end{aligned}$$

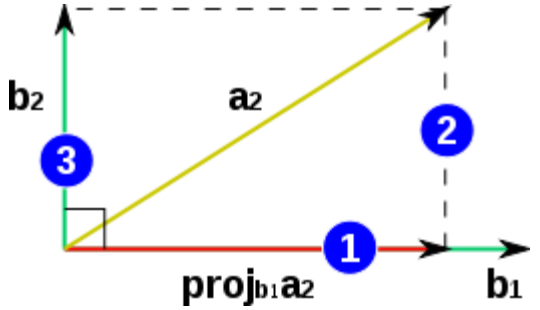
$$\mathbf{e}_j = \frac{\mathbf{b}_j}{\|\mathbf{b}_j\|}$$

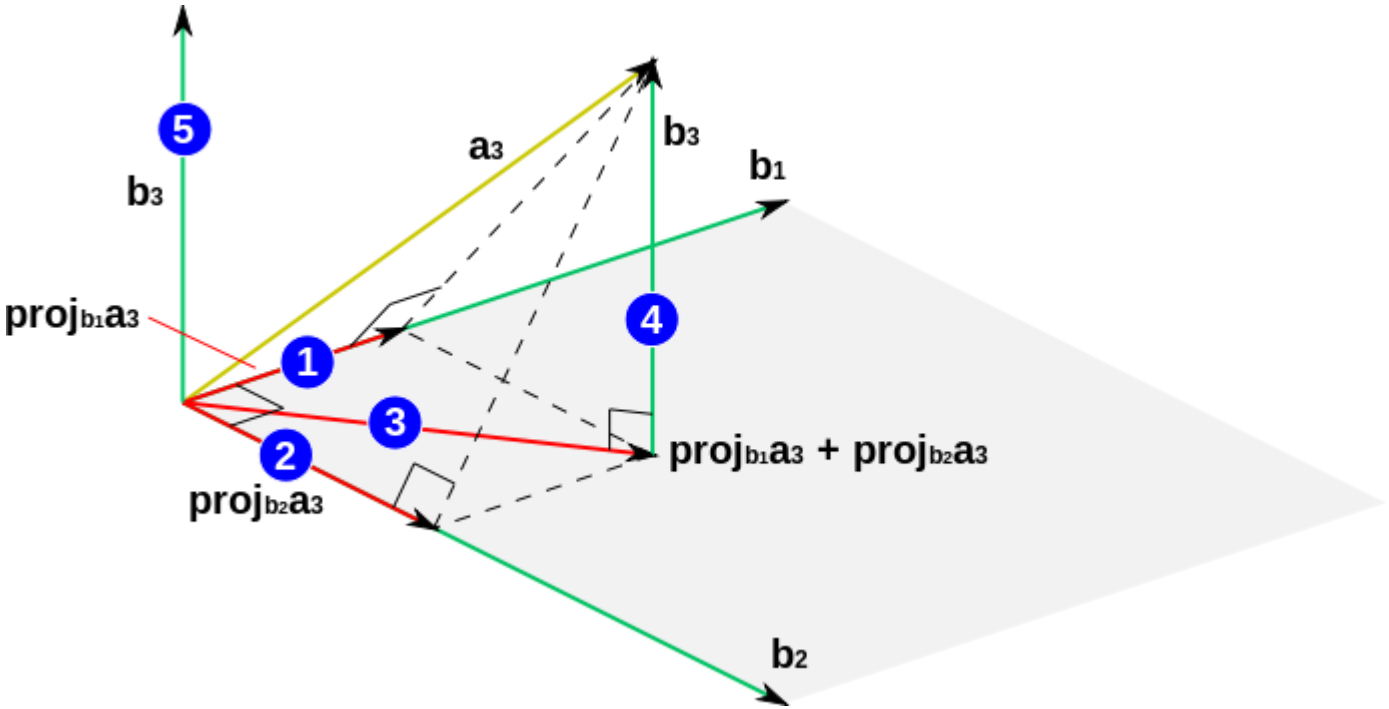
# Interpretacja geometryczna

$$\mathbf{b}_1 = \mathbf{a}_1$$

$$\mathbf{b}_2 = \mathbf{a}_2 - \mathrm{proj}_{\mathbf{b}_1}\,\mathbf{a}_2$$

$$\mathbf{b}_3 = \mathbf{a}_3 - (\mathrm{proj}_{\mathbf{b}_1}\mathbf{a}_3 + \mathrm{proj}_{\mathbf{b}_2}\mathbf{a}_3)$$



# Algorytm

$r_0 := b - Ax_0$

$p_0 := r_0$

$k := 0$

**repeat**

$$\alpha_k := \frac{r_k^\top r_k}{p_k^\top A p_k}$$

$x_{k+1} := x_k + \alpha_k p_k$

$r_{k+1} := r_k - \alpha_k A p_k$

**if** $r_{k+1}$ jest "wystarczająco mały" **then** exit loop **end if**

$$\beta_k := \frac{r_{k+1}^\top r_{k+1}}{r_k^\top r_k}$$

$p_{k+1} := r_{k+1} + \beta_k p_k$

$k := k + 1$

**end repeat**

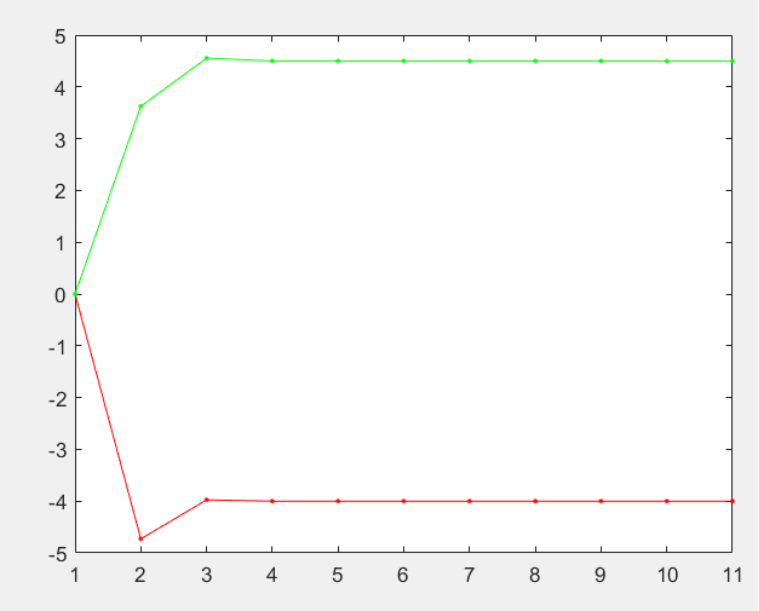
Wynikiem jest $x_{k+1}$

# Zbieżność algorytmu

$$E(x_k) \leq \left( \frac{1 - \sqrt{\kappa^{-1}}}{1 + \sqrt{\kappa^{-1}}} \right)^{2k} E(x_0)$$

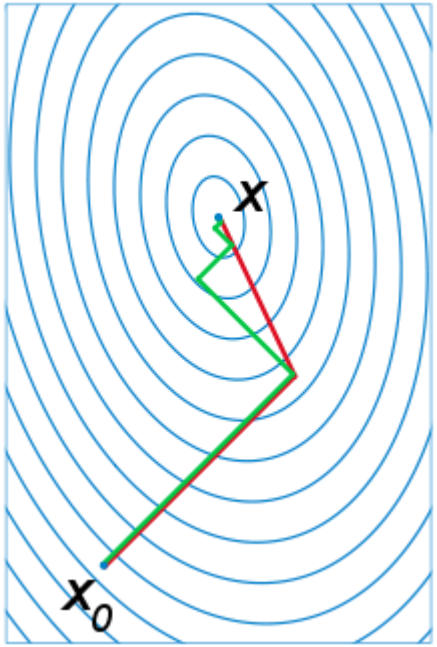
$$\kappa = \lambda_{max} / \lambda_{min}$$

```
function x = conjgrad(A, b, x)
    r = b - A * x;
    p = r;
    rsold = r' * r;

    for i = 1:length(b)
        Ap = A * p;
        alpha = rsold / (p' * Ap);
        x = x + alpha * p;
        r = r - alpha * Ap;
        rsnew = r' * r;
        if sqrt(rsnew) < 1e-10
            break
        end
        p = r + (rsnew / rsold) * p;
        rsold = rsnew;
    end
end
```




conjgradOS

# Porównanie zbieżności




Porównanie zbieżności metody gradientu prostego (na zielono) i sprzężonego gradientu (na czerwono) dla minimalizacji formy kwadratowej powiązanej z zadanym układem równań liniowych. Metoda sprzężonego gradientu zbiega w co najwyżej n krokach, gdzie n jest rozmiarem macierzy zadanego układu (tutaj n=2).

# Macierz blokowa

- Dowolną macierz A można przedstawić jako macierz blokową, zbudowaną z pewnej liczby macierzy o mniejszych wymiarach:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \cdots & \mathbf{A}_{1n} \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \cdots & \mathbf{A}_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ \mathbf{A}_{m1} & \mathbf{A}_{m2} & \cdots & \mathbf{A}_{mn} \end{bmatrix}$$

$\mathbf{A}_{ij}$ Ma wymiar $p_i \mathrm{x} q_j$

# Macierz blokowo-przekątniowa

$$A = \begin{bmatrix} B_1 & 0 & \cdots & 0 \\ 0 & B_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & B_N \end{bmatrix} \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_N \end{bmatrix} \qquad \xi = \begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \end{bmatrix}$$

gdzie $\xi_i$ to rozwiązanie układu $B_i \xi_i = b_i$.

$$x = \bigoplus_{i=1}^{N} \xi_i \quad \text{(symbol } \oplus \text{ oznacza łączenie wektorów)}$$

$$x = \text{vertcat}(\xi_1, \xi_2, \ldots, \xi_N)$$

# Przykład

Block.m

```
% Przykładowe bloki macierzy B_i
B1 = [4, 1;
      1, 3];
B2 = [5, 2;
      2, 6];
B3 = [7];

% Wektory prawej strony b_i
b1 = [10; 8];
b2 = [12; 9];
b3 = [14];

% Rozwiązanie każdego podukładu
x1 = B1 \ b1;  % Lub: x1 = inv(B1) * b1;
x2 = B2 \ b2;
x3 = B3 \ b3;

% Połączenie wyników
x = [x1; x2; x3];
```

```
% Zbuduj pełną macierz blokowo-przekątniową A
A = blkdiag(B1, B2, B3);

% Oblicz residuum r = b - A*x
b = [b1; b2; b3];
r = b - A * x;
```

```
Rozwiązanie x =
    2.0000
    2.0000
    2.0769
    0.8077
    2.0000

Residuum r = b - A*x =
     0
     0
     0
     0
     0

Macierz A =
     4     1     0     0     0
     1     3     0     0     0
     0     0     5     2     0
     0     0     2     6     0
     0     0     0     0     7
```

# Macierz blokowa ogólna

$$A = \begin{pmatrix} A_{11} & A_{12} & A_{13} & \cdots & A_{1p} \\ A_{21} & A_{22} & A_{23} & \cdots & A_{2p} \\ A_{31} & A_{32} & A_{33} & \cdots & A_{3p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{p1} & A_{p2} & \cdots & \cdots & A_{pp} \end{pmatrix}, \; x = \begin{pmatrix} \xi_1 \\ \xi_2 \\ \xi_3 \\ \vdots \\ \xi_p \end{pmatrix}, \; b = \begin{pmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \beta_p \end{pmatrix}$$

$$(Ax)_i = \sum_{j=1}^{p} A_{ij}\xi_j$$

**wektor x** jest jawnie oznaczony jako kolumna o blokowych składowych x=(ξ1,ξ2,…,ξpξ1,ξ2,…,ξp)

# Metoda blokowa Jakobiego

$$A = D - E - F$$

$$D = \begin{pmatrix} A_{11} & & & \\ & A_{22} & & \\ & & \ddots & \\ & & & A_{pp} \end{pmatrix},$$

$$E = -\begin{pmatrix} O & & & \\ A_{21} & O & & \\ \vdots & \vdots & \ddots & \\ A_{p1} & A_{p2} & \cdots & O \end{pmatrix}, \quad F = -\begin{pmatrix} O & A_{12} & \cdots & A_{1p} \\ & O & \cdots & A_{2p} \\ & & \ddots & \vdots \\ & & & O \end{pmatrix}$$

$$A_{ii}\xi_i^{(k+1)} = ((E+F)x_k)_i + \beta_i$$

$$\xi_i^{(k+1)} = A_{ii}^{-1}\left((E+F)x_k\right)_i + A_{ii}^{-1}\beta_i, \quad i = 1, \ldots, p,$$

Indeks $i$ "wycina" z wektora $(E+F)x_k$ tylko tę część, która wpływa na aktualizację $\xi_i^{(k+1)}$.

# Przykład

$$A = \begin{pmatrix} 4 & 1 & 0 & 2 \\ 1 & 3 & 0 & 0 \\ 1 & 0 & 5 & 0 \\ 0 & 1 & 0 & 6 \end{pmatrix} \qquad A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}$$

$$A_{11} = \begin{pmatrix} 4 & 1 \\ 1 & 3 \end{pmatrix}, \quad A_{12} = \begin{pmatrix} 0 & 2 \\ 0 & 0 \end{pmatrix}, \quad A_{21} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad A_{22} = \begin{pmatrix} 5 & 0 \\ 0 & 6 \end{pmatrix}$$

$$A_{11} = \begin{pmatrix} 4 & 1 \\ 1 & 3 \end{pmatrix},$$

$$A_{12} = \begin{pmatrix} 0 & 2 \\ 0 & 0 \end{pmatrix},$$

$$A_{21} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix},$$

$$A_{22} = \begin{pmatrix} 5 & 0 \\ 0 & 6 \end{pmatrix}.$$

$$D = \begin{pmatrix} 4 & 1 & 0 & 0 \\ 1 & 3 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 6 \end{pmatrix}, \quad E = -\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}, \quad F = -\begin{pmatrix} 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$D = \begin{pmatrix} A_{11} & 0 \\ 0 & A_{22} \end{pmatrix},$$

$$E = -\begin{pmatrix} 0 & 0 \\ A_{21} & 0 \end{pmatrix},$$

$$F = -\begin{pmatrix} 0 & A_{12} \\ 0 & 0 \end{pmatrix}.$$

# Przykład

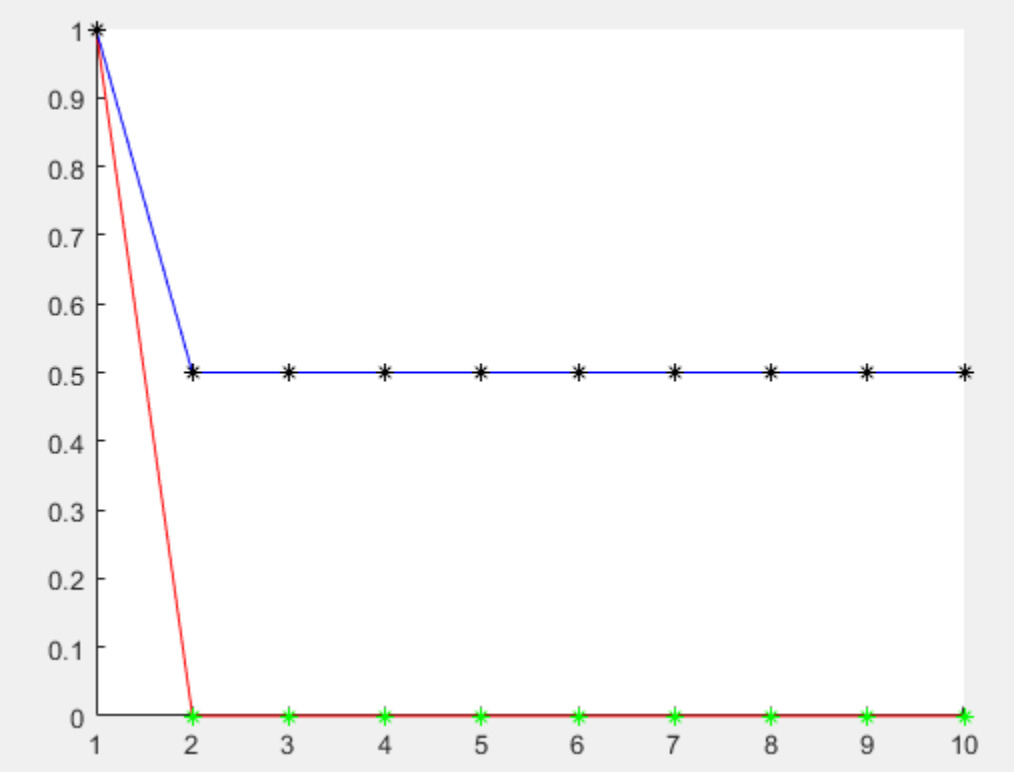
blockOgolny.m

```
% Definicja bloków macierzy A
A11 = [4, 1; 1, 3];
A12 = [0, 2; 0, 0];
A21 = [1, 0; 0, 1];
A22 = [5, 0; 0, 6];

% Pełna macierz A (do weryfikacji)
A = [A11, A12; A21, A22];

% Wektor prawej strony b (przykładowy)
b = [10; 8; 12; 9];

% Inicjalizacja rozwiązania x (startowe przybliżenie)
x = zeros(4, 1);

% Macierz diagonalna D (tylko bloki A11 i A22)
D = blkdiag(A11, A22);

% Macierze E i F (dolna i górna trójkątna część blokowa)
E = -[zeros(2), zeros(2); A21, zeros(2)];
F = -[zeros(2), A12; zeros(2), zeros(2)];
```

```
% Parametry iteracji
max_iter = 100;       % Maksymalna liczba iteracji
tolerance = 1e-6;     % Tolerancja zbieżności

for k = 1:max_iter
    x_old = x;

    % Oblicz (E + F) * x_old
    EFx = (E + F) * x_old;

    % Rozwiąż D * x_new = (E + F) * x_old + b (blokowo)
    % Blok 1: A11 * x1 = [EFx(1); EFx(2)] + [b(1); b(2)]
    x(1:2) = A11 \ (EFx(1:2) + b(1:2));

    % Blok 2: A22 * x2 = [EFx(3); EFx(4)] + [b(3); b(4)]
    x(3:4) = A22 \ (EFx(3:4) + b(3:4));

    % Sprawdź zbieżność
    if norm(x - x_old) < tolerance
        fprintf('Zbieżność osiągnięta po %d iteracjach.\n', k);
        break;
    end
end
```

# IterBlock.m

```
A11=[1 2;3 4];
A22=[5 6;7 8];
Z=[0 0;0 0];
A12=Z;A21=Z;
A=[A11 A12;A21 A22];
%B=blkdiag(A11,A22);
b=[1 2 3 4]';
E=-[Z Z;A21 Z];
F=-[Z A12;Z Z];
D=[A11 Z;Z A22];
x=[1 1 1 1]';
OutJ=[];
for i=1:10
      OutJ=[OutJ;x'];
   x=inv(D)*(b+(E+F)*x);
end;
 hold on;
plot(OutJ(:,1),'r');
plot(OutJ(:,2),'b');
plot(OutJ(:,3),'g*');
plot(OutJ(:,4),'k*');
```



# IterBlock2

**Dostateczny warunek zbieżności:**
**Macierz dominująca**

```
A11=[10 2;3 40];
A22=[50 6;7 80];
Z=[0 0;0 0];
A12=[3 2;5 4];A21=[7 5;4 1];
A=[A11 A12;A21 A22];
%B=blkdiag(A11,A22);
b=[1 2 3 4]';
E=-[Z Z;A21 Z];
F=-[Z A12;Z Z];
D=[A11 Z;Z A22];

x=[1 1 1 1]';
OutJ=[];

ksi1=[1 1]';
ksi2=[2 3]';
x=[ksi1; ksi2];
for i=1:10
      OutJ=[OutJ;ksi1' ksi2'];
    %x=inv(D)*(b+(E+F)*x);
     xx=(E+F)*x;
    ksi1=inv(A11)*(xx(1:2)+b(1:2));
    ksi2=inv(A22)*(xx(3:4)+b(3:4));
    x=[ksi1;ksi2];

end;
```

```
A =

    10     2     3     2
     3    40     5     4
     7     5    50     6
     4     1     7    80
```

# Regularyzacja Tichonowa

Rozważmy układ równań typu:

$$A\mathbf{x} = \mathbf{b},$$

gdzie $A$ to macierz układu (macierz przekształcenia), $\mathbf{x}$ wektor niewiadomych, $\mathbf{b}$ wektor wyjściowy. Jeśli zagadnienie to jest źle postawione (np. rozwiązanie nie istnieje, lub jest niejednoznaczne) wtedy standardowym podejściem jest rozwiązanie w sensie najmniejszych kwadratów. Oznacza to poszukiwanie minimum następującego funkcjonału $\Pi$[3]:

$$\Pi = \{A\mathbf{x} - \mathbf{b}\}^T \{A\mathbf{x} - \mathbf{b}\}.$$

Aby poprawić właściwości funkcjonału $\Pi$ (np. żeby wyeliminować niefizyczne oscylacje w rozwiązaniu) do powyższej postaci dodaje się człon regularyzacyjny $\alpha\mathbf{x}^T\mathbf{x}$:

$$\Pi = \{A\mathbf{x} - \mathbf{b}\}^T \{A\mathbf{x} - \mathbf{b}\} + \alpha\mathbf{x}^T\mathbf{x}.$$

Wyrażenie $\mathbf{x}^T\mathbf{x}$ to kwadrat długości wektora $\mathbf{x}$, zaś $\alpha$ to parametr regularyzacyjny zwany również współczynnikiem tłumienia.

Szukanie minimum funkcjonału $\Pi$ oznacza rozwiązanie równania:

$$\frac{\partial \Pi}{\partial x} = 0,$$

co daje:

$$-2A^T\{\mathbf{b} - A\mathbf{x}\} + 2\alpha\mathbf{x} = 0$$

i po przekształceniu:

$$A^T\mathbf{b} = \left[A^TA + \alpha\mathbf{I}\right]\mathbf{x},$$

gdzie $\mathbf{I}$ to macierz jednostkowa.

Macierz stojąca po prawej stronie powyższego równania jest macierzą symetryczną, ponadto dzięki obecności członu $\alpha\mathbf{I}$ jest również odwracalna. Poza tym parametr regularyzacyjny wpływa na określoność macierzy i poprawia jej uwarunkowanie. W związku z tym rozwiązanie wyjściowego równania może być wyznaczone za pomocą:

$$\mathbf{x} = \left[A^TA + \alpha\mathbf{I}\right]^{-1} A^T\mathbf{b}.$$

# Przykład

Rozważmy równanie:

$$ax = b$$

**Problem pojawia się, gdy:**

1. **Małe wartości** $a$ – Dzielimy przez liczbę bliską zera, co prowadzi do dużych wartości $x$.
2. **Zakłócenia w** $b$ – Mała zmiana w $b$ powoduje ogromną zmianę w $x$.

👉 **Przykład:** Jeśli:

$$a = 0.01, \quad b = 1$$

Rozwiązanie to:

$$x = \frac{b}{a} = \frac{1}{0.01} = 100$$

Teraz dodajmy małe zakłócenie do $b$:

$$b = 1.0001 \Rightarrow x = \frac{1.0001}{0.01} = 100.01$$

Niewielka zmiana w $b$ o 0.01% spowodowała dużą różnicę w $x$.

# Przykład

Dodajemy dodatkowy człon regularyzacyjny:

$$\min_x \left((ax - b)^2 + \lambda x^2\right)$$

Gdzie:

- $\lambda > 0$ – parametr regularyzacji (kontroluje karę za duże wartości $x$)

Aby znaleźć minimum, różniczkujemy po $x$:

$$\frac{d}{dx}\left(a^2x - 2ab + b^2 + \lambda x^2\right) = 0$$

$$2a^2x + 2\lambda x = 2ab$$

Po przekształceniu:

$$x = \frac{ab}{a^2 + \lambda}$$

- **Bez regularyzacji**: $\lambda = 0 \Rightarrow x = \frac{b}{a}$
- **Z regularyzacją**: $\lambda > 0$ – rozwiązanie jest mniejsze i stabilniejsze, nawet gdy $a$ jest bliskie zera

# Przykład

Załóżmy:

$$a = 0.01, \quad b = 1, \quad \lambda = 0.1$$

Bez regularyzacji:

$$x = \frac{1}{0.01} = 100$$

Z regularyzacją:

$$x = \frac{0.01 \times 1}{0.01^2 + 0.1} = \frac{0.01}{0.0001 + 0.1} = \frac{0.01}{0.1001} \approx 0.0999$$

```
a = 0.01;      % skalar
b = 1;         % wynik
lambda = 0.1;  % parametr regularyzacji

x = (a * b) / (a^2 + lambda);
disp(x);
a
a = 0.01;      % skalar
b = 1;         % wynik

% Różne wartości lambda
lambda_values = [0, 0.0001, 0.01, 0.1, 1];

% Obliczamy x dla każdej lambda
for lambda = lambda_values
    x = (a * b) / (a^2 + lambda);
    fprintf('Lambda: %.4f, x: %.4f\n', lambda, x);
end
```

Lambda: 0.0000, x: 100.0000
Lambda: 0.0001, x: 50.0000
Lambda: 0.0100, x: 0.9901
Lambda: 0.1000, x: 0.0999
Lambda: 1.0000, x: 0.0100

Załóżmy, że losowy szum wynosił:

$$b_{\text{noisy}} = [1.01, 0.98, 1.02, 0.95, 1.03]$$

| $\lambda$ | Rozwiązanie $x$ dla różnych zaszumionych $b$ | Charakterystyka |
|---|---|---|
| 0 | [101, 98, 102, 95, 103] | **Niestabilne** – wynik "eksploduje". |
| 0.0001 | [50.50, 49.00, 51.00, 47.50, 51.50] | Nadal duże wahania. |
| 0.01 | [0.995, 0.980, 1.010, 0.955, 1.030] | **Stabilne i dokładne** – dobry kompromis. |
| 0.1 | [0.0995, 0.0980, 0.1010, 0.0955, 0.1030] | Bardzo stabilne, ale **zaniżone**. |
| 1.0 | [0.0100, 0.0098, 0.0102, 0.0095, 0.0103] | **Przesadna regularyzacja** – tłumi wynik. |

# Regularyzacja Tichonowa

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

$A$ – macierz współczynników (rozmiaru $m \times n$)
$x$ – wektor niewiadomych (rozmiaru $n \times 1$)
$b$ – wektor wynikowy (rozmiaru $m \times 1$)

Dotichonov_reg.m

Zamiast rozwiązywać układ bezpośrednio, minimalizujemy funkcję celu:

$$\|A\mathbf{x} - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x}\|_2^2$$

gdzie:

- $\| \cdot \|_2$ – norma 2 (euklidesowa)
- $\lambda > 0$ – parametr regularyzacji (kontroluje kompromis między zgodnością z danymi a gładkością rozwiązania)

## Postać rozwiązania

Rozwiązanie układu z regularyzacją Tichonowa można zapisać jako:

$$\mathbf{x} = (A^T A + \lambda I)^{-1} A^T \mathbf{b}$$

gdzie:

- $A^T$ – macierz transponowana do $A$
- $I$ – macierz jednostkowa o odpowiednich wymiarach
- $\lambda$ – parametr regularyzacji (im większy, tym bardziej "wygładzone" rozwiązanie)

# Optymalizacja parametru regularyzacji

**Metoda L-krzywej**

Metoda L-krzywej pomaga znaleźć optymalny parametr regularyzacji $\lambda$ w problemach typu $\min_x \|Ax - b\|^2 + \lambda\|x\|^2$.

Krzywa ta wizualizuje kompromis między **normą residuum** ($\|Ax - b\|$) a **normą rozwiązania** ($\|x\|$) dla różnych $\lambda$.

### 1. Algorytm metody L-krzywej

1. **Oblicz rozwiązania regularyzowane** dla różnych wartości $\lambda$.
2. **Dla każdego** $\lambda$ zapisz:
   - Normę residuum: $\rho(\lambda) = \|Ax_\lambda - b\|$,
   - Normę rozwiązania: $\eta(\lambda) = \|x_\lambda\|$.
3. **Narysuj log-log wykres** $\eta(\lambda)$ vs. $\rho(\lambda)$.
4. **Wybierz** $\lambda$ **w "kącie" krzywej** – punkt optymalnego balansu między dopasowaniem do danych a stabilnością rozwiązania.

Krzywizna parametryczna w dwóch wymiarach (dla krzywej (x(t),y(t)) jest obliczana według wzoru:

$$\kappa = \frac{|x'(t) \cdot y''(t) - y'(t) \cdot x''(t)|}{(x'(t)^2 + y'(t)^2)^{3/2}}$$

# Przykład

$$x = \frac{ab}{a^2 + \lambda}$$

Ax=b

a=0.01,b=1,lambda=9.326e-05

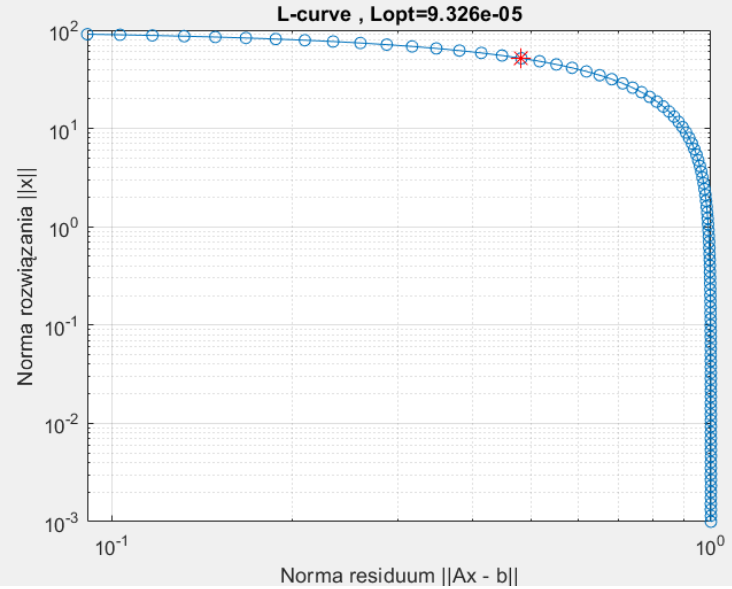
x=

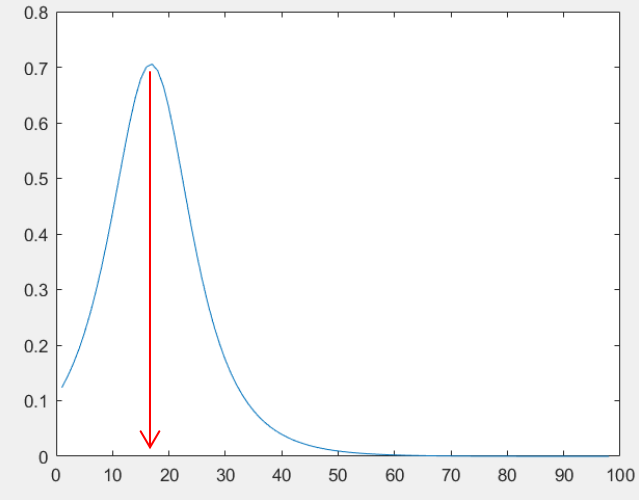
x=a*b/(a^2+lambda)

x = 51.7438

LcurveFindLambda.m

Optymalne lambda: 9.326033e-05 (indeks: 17)



# Dodatkowe materiały

# Metoda gradientu sprzężonego

Liniowo niezależni wektory względem A $p_k,\ k = 0, 1, \ldots, n-1$ $p_k^T A p_j = 0\,, k \neq j$

Dowolny wektor można wyrazić jako ważona suma liniowo niezależnych wektorów

$$x^* - x_0 = \sum_{j=0}^{n-1} \alpha_j p_j$$

Pomnożymy na $p_k^T A$

$$p_k^T A(x^* - x_0) = p_k^T (b - Ax_0) = p_k^T r_0 \qquad p_k^T A \sum_{j=0}^{n-1} \alpha_j p_j = \alpha_k p_k^T A p_k$$

(wg liniowej niezależności)

Więc mamy $p_k^T r_0 = \alpha_k p_k^T A p_k$

Oraz

$$\alpha_k = \frac{p_k^T r_0}{p_k^T A p_k}$$

# Algorytm

Najpierw znajdujemy ciąg n sprzężonych kierunków, następnie obliczamy współczynniki $\alpha$

Dla dowolnych liniowo niezależnych wektorów $v_k,\ k = 0, 1, \ldots, n-1$

1. Let $p_0 = v_0$ .
2. For $k = 0, 1, \ldots, n-2$

$$p_{k+1} = v_{k+1} - \sum_{j=0}^{k} \frac{p_j^T A v_{k+1}}{p_j^T A p_j} p_j$$

End For.

Ortogonalizacja Grama-Shcmidta

jako pierwszy wektor bazowy p1 wybrać gradient f w $\boldsymbol{x = x_0}$, który wynosi $Ax_0 - b$, a ponieważ wybraliśmy $x_0 = 0$, otrzymujemy $-b$. **Pozostałe wektory w bazie będą sprzężone do gradientu** (stąd nazwa metoda gradientu sprzężonego).

# Algorytm (cd)

1. Pick $x_0$ and $A$-conjugate directions $p_k$, $k = 0, 1, \ldots, n-1$.
2. For $k = 0, 1, \ldots, n-1$

   (a) Set

   $$\alpha_k = \frac{p_k^T r_0}{p_k^T A p_k}.$$

   (b) Let $x_{k+1} = x_k + \alpha_k p_k$.

   End For.

# Algorytm gradientu sprężonego

1. Let $x_0$ be an initial guess.
   Let $r_0 = b - Ax_0$ and $p_0 = r_0$.

2. For $k = 0, 1, 2, \ldots,$ until convergence,

   (a) Compute the search parameter $\alpha_k$ and the new iterate and residual

$$\begin{aligned} \alpha_k &= \frac{r_k^T r_k}{p_k^T A p_k}, \\ x_{k+1} &= x_k + \alpha_k p_k, \\ r_{k+1} &= r_k - \alpha_k A p_k, \end{aligned}$$

   (b) Compute the new search direction $p_{k+1}$ by Gram-Schmidt on $r_{k+1}$ and the previous $p$ vectors to make $p_{k+1}$ $A$-conjugate to the previous directions.

   End For.

# Algorytm gradientu sprężonego (w całości)

1. Let $x_0$ be an initial guess.
   Let $r_0 = b - Ax_0$ and $p_0 = r_0$.

2. For $k = 0, 1, 2, \ldots,$ until convergence,

   (a) Compute the search parameter $\alpha_k$ and the new iterate and residual

$$\begin{aligned} \alpha_k &= \frac{p_k^T r_k}{p_k^T A p_k}, (\text{or, equivalently, } \frac{r_k^T r_k}{p_k^T A p_k}) \\ x_{k+1} &= x_k + \alpha_k p_k, \\ r_{k+1} &= r_k - \alpha_k A p_k, \end{aligned}$$

   (b) Compute the new search direction

$$\begin{aligned} \beta_k &= -\frac{p_k^T A r_{k+1}}{p_k^T A p_k}, (\text{or, equivalently, } \frac{{r_{k+1}}^T r_{k+1}}{r_k^T r_k}), \\ p_{k+1} &= r_{k+1} + \beta_k p_k, \end{aligned}$$

   End For.