Pokročilá algoritmizace

hledání kořenu funkce, řešení soustav lineárních rovnic, výpočet inverzní matice

Jiří Vyskočil, Marko Genyg-Berezovskyj

2009

Hledání kořenu funkce

- Mějme funkci $f : \mathbb{R} \to \mathbb{R}$.
- Chceme najít tzv. *kořen* $r \in \mathbb{R}$ tak, aby platilo

$$f(r) = 0$$

- Ve většině případů lze kořen najít pouze užitím numerických metod (pozn. většina reálný čísel nelze ani jednoduše symbolicky vyjádřit).
- To znamená, že nezískáme přesnou hodnotu kořene, ale pouze jeho přibližnou hodnotu (aproximaci).
- Iterativní numerické metody nám umožňují nalézt řešení s libovolnou požadovanou přesností (přesnost je samozřejmě ještě závislá na reprezentaci čísel).

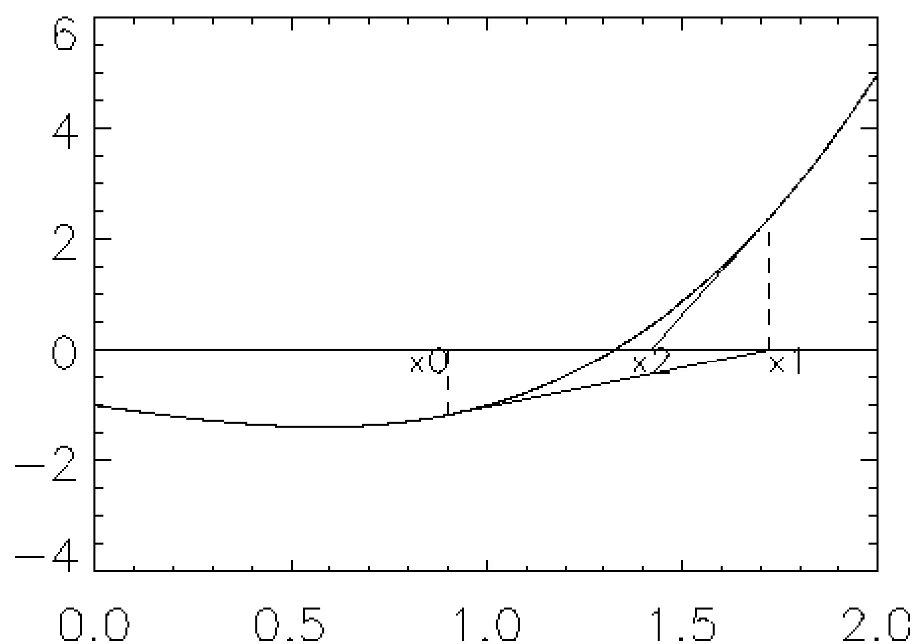
Hledání kořenu funkce

- numerické metody
  - **stabilita algoritmu**
    - Algoritmus je *stabilní*, pokud malé změny počátečních dat způsobí malé změny v celkovém výsledku. Pokud malé změny počátečních dat způsobí velké změny výsledku, označujeme takový numerický algoritmus jako *nestabilní*.
  - **kritérium zastavení** (stopping criterion)
    - *Kritérium zastavení* je ukončovací podmínka v numerických algoritmech (nejčastěji iterativních). Po této podmínce obvykle požadujeme, aby algoritmus skončil po dosažení požadované přesnosti nebo aby skončil v případě, že došlo k zacyklení nebo divergenci.
  - **absolutní chyba**
    - Nechť a je přená hodnota a nechť $\tilde{a}$ je její aproximace, potom *absolutní chyba* ε je definována vztahem $a = \tilde{a} + \varepsilon \quad \Rightarrow \quad \varepsilon = a - \tilde{a}$
  - **relativní chyba**
    - Nechť a je přená hodnota a nechť $\tilde{a}$ je její aproximace, potom *relativní chyba* ε_r je definována vztahem

$$\varepsilon_r = \frac{a - \tilde{a}}{a}$$

Hledání kořenu funkce

- Newtonova metoda (grafické řešení)
  1) $iterace = 0$;
  2) $x_{iterace}$ = počáteční odhad kořenu funkce;
  3) **while not** kritérium_zastavení **do** {
  4) zkonstruuj tangentu k funkci f v bodě $(x_{iterace}, f(x_{iterace}))$;
  5) $iterace$++;
  6) $x_{iterace}$ = x-ová hodnota průsečíku tangenty a osy x;
  7) }
  8) $r = x_{iterace}$;



Hledání kořenu funkce

- tangenta (tečna)
  - Rovnice tangenty v bodě x_0 je

$$y - y_0 = m \cdot (x - x_0)$$

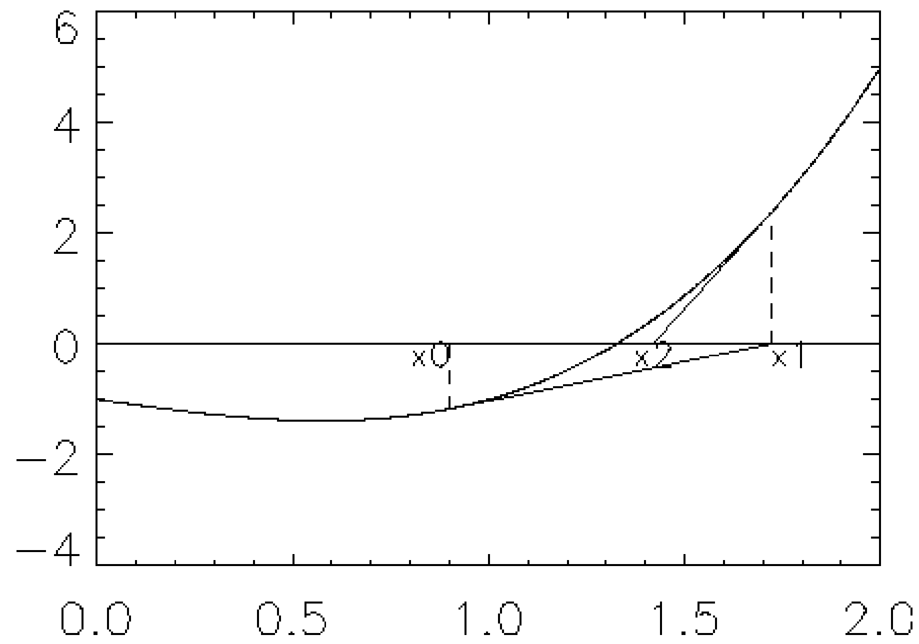
kde $y_0 = f(x_0)$ a $m = f'(x_0)$

  - Průsečík tangenty s osou x (tedy y=0)

$$\left(x_0 - \frac{f(x_0)}{f'(x_0)}, 0\right)$$

  - Z toho můžeme odvodit výpočet pro x_{n+1}

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



Hledání kořenu funkce

- kritérium zastavení Newtonovy metody
  1. $|f(x_n)|$ je dostatečně malé.
     - Hledáme hodnotu x, takovou aby $f(x)=0$. To znamená, že pokud je $|f(x_n)|$ dostatečně malé, jsme blízko kořene.
  2. $|x_{n+1} - x_n|$ je dostatečně malé.
     - Při výpočtu je dobré pracovat s přesností alespoň o dva řády vyšší než je požadovaná cílová přesnost. Pokud se výsledné hodnoty x v posledních dvou iteracích po zaokrouhlení na požadovanou přesnost neliší, nemá smysl pokračovat v iteraci.
  3. Bylo provedeno určité maximální množství iterací.
     - Tato podmínka má smysl, pokud k řešení splňující podmínky 1. a 2. konvergujeme příliš pomalu nebo pokud nekonvergujeme vůbec a dostáváme se do nekonečného cyklu (technicky se zacyklení velmi špatně detekuje).
  4. $f'(x_n)=0$
     - Tento stav je velmi nepravděpodobný, ale pokud nastane způsobí chybu při dělení 0. Takovýto stav lze ošetřit přičtením nebo odečtením malé konstanty (V takovém případě je však třeba ošetřit zacyklení).

Hledání kořenu funkce

- odhad chyby Newtonovy metody
  - Po provedení n iterací označíme chybu ε_n tak aby platilo

    $$r = x_n + \varepsilon_n$$

    kde r je přesná hodnota kořene.
  - Podobně získáme závislost chyby po n+1 iteracích

    $$r = x_{n+1} + \varepsilon_{n+1} \quad \Rightarrow \quad x_{n+1} = r - \varepsilon_{n+1}$$

  - Nyní nahradíme chybu v n –té iteraci chybou v n+1 iteraci

    $$x_{n+1} = r - \varepsilon_{n+1} = r - \varepsilon_n - \frac{f(r - \varepsilon_n)}{f'(r - \varepsilon_n)}$$

  - To lze přepsat na

    $$\varepsilon_{n+1} = \varepsilon_n + \frac{f(r - \varepsilon_n)}{f'(r - \varepsilon_n)}$$

  - Pro další pokračování přepokládejme, že chyba každé iterace je dostatečně malá tak, že funkci f můžeme aproximovat Taylorovým polynomem.

Hledání kořenu funkce

- Taylorova řada
  - Za určitých předpokladů o funkci $f(x)$ v okolí bodu a lze tuto funkci vyjádřit jako *Taylorovu řadu*

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots = \sum_{k=0}^{\infty} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

- Taylorův polynom
  - Pro přibližné vyjádření hodnot funkce není nutné vyjadřovat všechny členy Taylorovy řady, ale můžeme zanedbat členy s vyššími derivacemi. Získáme tím tzv. *Taylorův polynom* řádu n v bodě a.

$$T_n(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \frac{f^{(k)}(a)}{k!}(x-a)^k$$

Hledání kořenu funkce

- odhad chyby Newtonovy metody
  - Vyjádříme funkci f a její derivaci pomocí Taylorovi řady v bodě r

$$f(r-\varepsilon_n) = 0 - \varepsilon_n f'(r) + \frac{{\varepsilon_n}^2}{2} f''(r) + \cdots$$

$$f'(r-\varepsilon_n) = f'(r) - \varepsilon_n f''(r) + \frac{{\varepsilon_n}^2}{2} f'''(r) + \cdots$$

  - Nyní stačí dosadit do předchozího vyjádření ε_n a upravit

$$\begin{aligned}
\varepsilon_{n+1} &= \varepsilon_n + \frac{f(r-\varepsilon_n)}{f'(r-\varepsilon_n)} \\
&= \varepsilon_n + \frac{-\varepsilon_n f'(r) + \frac{{\varepsilon_n}^2}{2} f''(r) + \cdots}{f'(r) - \varepsilon_n f''(r) + \cdots} \\
&= \frac{\varepsilon_n f'(r) - {\varepsilon_n}^2 f''(r) + \cdots - \varepsilon_n f'(r) + \frac{{\varepsilon_n}^2}{2} f''(r) + \cdots}{f'(r) - \varepsilon_n f''(r) + \cdots} \\
&= \frac{-{\varepsilon_n}^2 f''(r)/2 + \cdots}{f'(r) - \varepsilon_n f''(r) + \cdots} = -\frac{{\varepsilon_n}^2 f''(r) + \cdots}{2f'(r) + \cdots} \approx -\frac{{\varepsilon_n}^2 f''(r)}{2f'(r)}
\end{aligned}$$

Hledání kořenu funkce

- odhad chyby Newtonovy metody
  - Výsledný vztah $\varepsilon_{n+1} \approx -\frac{{\varepsilon_n}^2 f''(r)}{2f'(r)}$ ukazuje závislost chyby v iteraci $n+1$ na chybě v n-té iteraci.
  - Jinými slovy tento vztah ukazuje rychlost konvergence algoritmu. Pokud bude například chyba v nějaké iteraci 10^{-2}. V další iteraci bude chyba úměrná 10^{-4} a v další iteraci 10^{-8} atd.
  - Chyba tedy klesá poměrně rychle.

Hledání kořenu funkce

- nevýhody Newtonovy metody
  - Z popisu algoritmu není jasné jak se má spočítat derivace funkce v bodě (obecně to může být obtížné). Jeden z jednoduchých způsobů jak derivaci spočítat je
    $$f'(x) = \frac{f(x+\Delta) - f(x)}{\Delta}$$
    kde Δ je co nejmenší (pro $\Delta \to 0$ limitně jde přímo o definici derivace).
  - Pokud má funkce více kořenů je obtížné tyto kořeny touto metodou nalézt všechny.
  - Problémy mohou nastat v případě, že funkce nemá ve všech bodech nenulovou derivaci (můžeme pokračovat na špatné straně).
  - Další problémy mohou nastat v případě, že funkce není spojitá nebo dokonce není všude definovaná.

LUP dekompozice

- Zopakování maticových definic:
  - *Matice* (matrix) je dvojrozměrné pole čísel. Například matice A o rozměrech 2 × 3

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

  - *Transponovaná matice* vznikne výměnou všech řádků a sloupců.

$$A^{\mathrm{T}} = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

$$x = \begin{pmatrix} 2 \\ 3 \\ 5 \end{pmatrix}$$

  - *Vektor* je matice s druhým rozměrem vždy 1.
  - *Jednotkový vektor* e_{i} je vektor, kde i-tý prvek je 1 a ostatní jsou nuly..
  - *Čtvercová matice* je matice $n \times n$.
  - *Diagonální matice* je čtvercová matice, kde $a_{\mathrm{ij}} = 0$ pro $\mathrm{i} \neq \mathrm{j}$.

$$\operatorname{diag}(a_{11}, a_{22}, \ldots, a_{nn}) = \begin{pmatrix} a_{11} & 0 & \ldots & 0 \\ 0 & a_{22} & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & a_{nn} \end{pmatrix}$$

  - *Jednotková matice* I_n rozměrů $n \times n$ je diagonální matice s jedničkami na úhlopříčce. Sloupce jsou jednotkové vektory e_{i}.

LUP dekompozice

- Zopakování maticových definic:
  - *Horní trojúhelníková matice* U má $u_{ij}=0$ pro i>j .
    Horní jednotková trojúhelníková matice má navíc na diagonále jedničky.
  - *Dolní trojúhelníková matice* L má $l_{ij}=0$ pro i<j .
    Dolní jednotková trojúhelníková matice má navíc na diagonále jedničky.
  - *Permutační matice* P má právě jednu 1 v každém řádku a sloupci a 0 jinde.
  - *Inverzní matice* k $n \times n$ matici A je matice rozměrů $n \times n$, označovaná A^{-1} (pokud existuje) taková, že platí $A A^{-1} = I_n = A^{-1} A$

$$U = \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \dots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & u_{nn} \end{pmatrix}$$

$$L = \begin{pmatrix} l_{11} & 0 & \dots & 0 \\ l_{21} & l_{22} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \dots & l_{nn} \end{pmatrix}$$

$$P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

LUP dekompozice

- **Řešení soustav lineárních rovnic pomocí LUP dekompozice**
  - Mějme soustavu rovnic $Ax = b$, tj. pro $A = (a_{ij})$, $x = (x_j)$ a $b = (b_i)$

$$a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2$$

...

$$a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n$$

  - Pro dané A a b hledáme řešení x soustavy. Řešení může být i několik (málo určená soustava) nebo žádné (přeurčená soustava).
  - Pokud je A není singulární, existuje A^{-1} a $x = A^{-1}b$, protože
    $x = A^{-1} A x = A^{-1}b$.
    Řešení x je potom jediné.

LUP dekompozice

- **Řešení soustav lineárních rovnic pomocí LUP dekompozice**
  - Možná metoda rešení:
    - spočítáme A^{-1} a následě x. Tento postup je ale numericky nestabilní, tj. zaokrouhlovací chyby se kumulují při práci s počítačovou reprezentací reálných čísel.
  - Metoda LUP:
    - pro A najdeme tři matice L, U, P rozměru $n \times n$, tzv. LUP dekompozici tak, že

$$PA = LU$$

kde

- L je jednotková dolní trojúhelníková matice
- U je horní trojúhelníková matice
- P je permutační matice

LUP dekompozice

- **Řešení soustav lineárních rovnic se znalostí LUP dekompozice**
  - Soustava $PAx = Pb$ je permutované původní řešení (odpovídá pouze přehození rovnic díky provedené permutaci).
  - Použitím rovnosti dekompozice $PA=LU$ máme $LUx = Pb$ a řešíme trojúhelníkové soustavy.
  - Označme $y = Ux$.
  - Řešíme $Ly = Pb$ pro neznámý vektor y metodou *dopředné substituce* a potom pro známé y řešíme $Ux = y$ pro x metodou *zpětné substituce*.
  - Vektor x je hledané řešení, protože P je invertovatelná a

    $Ax = P^{-1}LUx = P^{-1}Pb = b$.

LUP dekompozice

- **dopředná substituce**
  - řeší dolní trojúhelníkovou soustavu v čase $\Theta(n^2)$ pro dané L, P a b.
  - Označme $c = Pb$ permutaci vektoru b (tedy $c_i = b_{\pi(i)}$).
    Řešená soustava $Ly = Pb$ je soustava rovnic

$$
\begin{array}{ccccccccc}
y_1 & & & & & & & = & c_1 \\
l_{21}y_1 & + & y_2 & & & & & = & c_2 \\
l_{31}y_1 & + & l_{32}y_2 & + & y_3 & & & = & c_3 \\
\vdots & & & & & \ddots & & & \vdots \\
l_{n1}y_1 & + & l_{n2}y_2 & + & l_{n3}y_3 & +\cdots+ & y_n & = & c_n
\end{array}
$$

  - Hodnotu y_1 známe z první rovnice a můžeme ji dosadit do druhé.
  - Dostáváme $y_2 = c_2 - l_{21}y_1$
  - Obecně, dosadíme $y_1, y_2, \ldots, y_{i-1}$ "dopředu" do i-té rovnice a dostaneme

$$y_i = c_i - \sum_{j=1}^{i-1} l_{ij} y_j$$

LUP dekompozice

- **zpětná substituce**
  - Je podobná *dopředné substituci* a řeší horní trojúhelníkovou soustavu v čase $\Theta(n^2)$ pro dané U a y.
  - Řešená soustava $Ux = y$ je soustava rovnic

$$
\begin{array}{ccccccccc}
u_{11}x_1 & + & u_{12}x_2 & +\cdots+ & u_{1,n-1}x_{n-1} & + & u_{1n}x_n & = & y_1 \\
 & & u_{22}x_2 & +\cdots+ & u_{2,n-1}x_{n-1} & + & u_{2n}x_n & = & y_2 \\
 & & & \ddots & & & & \vdots & \\
 & & & & u_{n-1,n-1}x_{n-1} & + & u_{n-1,n}x_n & = & y_{n-1} \\
 & & & & & & u_{nn}x_n & = & y_n
\end{array}
$$

  - Hodnotu x_n vypočteme nejprve z poslední rovnice $x_n = y_n / u_{nn}$ a pak ji můžeme dosadit do předposlední rovnice a dostáváme

$$x_{n\text{-}1} = (y_{n\text{-}1} - u_{n\text{-}1,n}x_n) / u_{n\text{-}1,n\text{-}1}$$

  - Obecně, dosadíme $x_n, x_{n\text{-}1}, \ldots, x_{i+1}$ "zpětně" do i-té rovnice a dostaneme

$$x_i = \left(y_i - \sum_{j=i+1}^{n} u_{ij}\, x_j\right) \Big/ u_{ii}$$

LUP dekompozice

- **Řešení soustav lineárních rovnic se znalostí LUP dekompozice**
  - Permutační matice P je reprezentovaná polem $\pi[1..n]$,

    kde $\pi[i]=j$ znamená, že i-tý řádek P obsahuje 1 v j-tém sloupci.
  - Celková složitost algoritmu na řešení soustav lineárních rovnic se znalostí L, U a π je $\Theta(n^2)$ (použili jsme pouze dopřednou a zpětnou substituci).
  - Nyní nám zbývá ukázat provedení LUP dekompozice.
  - Pro jednoduchost začneme nejprve provedením LU dekompozice.

LUP dekompozice

Výpočet LU dekompozice

- Nejprve jednodušší případ, když matice P chybí (tj. $P = I_n$).
- Idea metody:
  - Gaussova eliminace, při které vhodné násobky prvního řádku přičítáme k dalším řádkům tak, abychom odstranili x_1 z dalších rovnic (koeficienty u x_1 v prvním sloupci budou nulové). Potom pokračujeme (rekurzivně) v dalších sloupcích, až vznikne horní trojúhelníková matice, tj. U. Matice L vzniká z koeficientů, kterými jsme násobili řádky.
- Z matice A oddělíme první řádek a sloupec, A' je matice $(n-1) \times (n-1)$, v sloupcový vektor a w^{T} řádkový vektor.

$$A = \left(\begin{array}{c|cccc} a_{11} & a_{12} & \cdots & a_{1n} \\ \hline a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{array}\right) = \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ v & A' \end{pmatrix} .$$

LUP dekompozice

- **Výpočet LU dekompozice**
  - Dále matici rozložíme na součin

$$
\begin{aligned}
A &= \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ v & A' \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & A' - vw^{\mathrm{T}}/a_{11} \end{pmatrix} .
\end{aligned}
$$

  - Podmatice A'-vw^{T}/a_{11} rozměrů $(n-1) \times (n-1)$ se nazývá *Schurův komplement* A vzhledem k a_{11}.
  - Nyní najdeme rekuzivně LU rozklad Schurova komplementu, nechť je roven $L'U'$.
  - S využitím maticové algebry odvodíme:
  - Matice L a U jsou požadované trojúhelníkové matice, protože L' a U' jsou také požadovaného tvaru.

$$
\begin{aligned}
A &= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & A' - vw^{\mathrm{T}}/a_{11} \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ v/a_{11} & I_{n-1} \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & L'U' \end{pmatrix} \\
&= \begin{pmatrix} 1 & 0 \\ v/a_{11} & L' \end{pmatrix} \begin{pmatrix} a_{11} & w^{\mathrm{T}} \\ 0 & U' \end{pmatrix} \\
&= LU ,
\end{aligned}
$$

LUP dekompozice

- **Výpočet LU dekompozice** (nerekurzivně)

Procedure LU-DECOMPOSITION(matrix A)
$n = rows[A]$;
for $k = 1$ **to** n **do** {
 $u_{kk} = a_{kk}$;
 for $i = k + 1$ **to** n **do** {
  $l_{ik} = a_{ik}/u_{kk}$; // l_{ik} představuje v_i
  $u_{ki} = a_{ki}$; // u_{ki} představuje $w^{T}{}_{i}$
 }
 for $i = k + 1$ **to** n **do**
  for $j = k + 1$ **to** n **do**
   $a_{ij} = a_{ij} - l_{ik}u_{kj}$;
}
return L and U

- Složitost je $\Theta(n^3)$.

LUP dekompozice

- **Výpočet LU dekompozice** (příklad)

$$\begin{array}{cccc} 2 & 3 & 1 & 5 \\ 6 & 13 & 5 & 19 \\ 2 & 19 & 10 & 23 \\ 4 & 10 & 11 & 31 \end{array} \qquad \begin{array}{c|ccc} 2 & 3 & 1 & 5 \\ \hline 3 & 4 & 2 & 4 \\ 1 & 16 & 9 & 18 \\ 2 & 4 & 9 & 21 \end{array} \qquad \begin{array}{cc|cc} 2 & 3 & 1 & 5 \\ \hline 3 & 4 & 2 & 4 \\ \hline 1 & 4 & 1 & 2 \\ 2 & 1 & 7 & 17 \end{array} \qquad \begin{array}{ccc|c} 2 & 3 & 1 & 5 \\ 3 & 4 & 2 & 4 \\ \hline 1 & 4 & 1 & 2 \\ \hline 2 & 1 & 7 & 3 \end{array}$$

(a) (b) (c) (d)

$$\underbrace{\begin{pmatrix} 2 & 3 & 1 & 5 \\ 6 & 13 & 5 & 19 \\ 2 & 19 & 10 & 23 \\ 4 & 10 & 11 & 31 \end{pmatrix}}_{A} = \underbrace{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 3 & 1 & 0 & 0 \\ 1 & 4 & 1 & 0 \\ 2 & 1 & 7 & 1 \end{pmatrix}}_{L} \underbrace{\begin{pmatrix} 2 & 3 & 1 & 5 \\ 0 & 4 & 2 & 4 \\ 0 & 0 & 1 & 2 \\ 0 & 0 & 0 & 3 \end{pmatrix}}_{U}$$

(e)

  - Metoda nefunguje pokud u aktuálně zpracovávané podmatice A' platí $a'_{11} = 0$.
  - Metoda může generovat velké chyby pokud se $|a'_{11}|$ blíží k 0.

LUP dekompozice

- **Výpočet LUP dekompozice** (nerekurzivně)

```
Procedure LUP-DECOMPOSITION(matrix A)
n = rows[A];
for i = 1 to n do π[i] = i ;
for k = 1 to n do {              // hlavní cyklus
    p = 0 ;                      // nulování pivota
    for i = k to n do {          // výběr pivota
        if |a_ik| > p then {
            p = |a_ik| ;
            k' = i ;             // pozice pivota
        }
    if p = 0 then error "singular matrix“;
    exchange π[k] ↔ π[k'] ;
    for i = 1 to n do exchange a_ki ↔ a_k'i ;
    for i = k + 1 to n do {
        a_ik = a_ik/a_kk ;       // k-tý sloupec L
        for j = k + 1 to n do  a_ij = a_ij - a_ik a_kj ;   // změna U
    }
}
```

- Složitost je $\Theta(n^3)$.
- Výsledné matice L a U jsou obsaženy v pozměněné matici A následujícím způsobem

$$a_{ij} = \begin{cases} l_{ij} \text{ pokud } i > j \\ u_{ij} \text{ pokud } i \leq j \end{cases}$$

- LUP dekompozice odstraňuje nevýhody LU dekompozice.
  - Pokud matice A není singulární vždy vybereme největší prvek pro volbu a_{11}.
  - Tímto zabráníme vzniku výpočetních chyb a zajistíme nalezení řešení pro všechny matice A, které nejsou singulární.

LUP dekompozice

- **Výpočet LUP dekompozice** (příklad)

$$\begin{array}{|c|cccc}1 & 2 & 0 & 2 & 0.6\\2 & 3 & 3 & 4 & -2\\3 & 5 & 5 & 4 & 2\\4 & -1 & -2 & 3.4 & -1\end{array}$$

(a)

$$\begin{array}{|c|cccc}3 & 5 & 5 & 4 & 2\\2 & 3 & 3 & 4 & -2\\1 & 2 & 0 & 2 & 0.6\\4 & -1 & -2 & 3.4 & -1\end{array}$$

(b)

$$\begin{array}{|c|cccc}3 & 5 & 5 & 4 & 2\\2 & 0.6 & 0 & 1.6 & -3.2\\1 & 0.4 & -2 & 0.4 & -.2\\4 & -0.2 & -1 & 4.2 & -0.6\end{array}$$

(c)

$$\begin{array}{|c|cccc}3 & 5 & 5 & 4 & 2\\2 & 0.6 & 0 & 1.6 & -3.2\\1 & 0.4 & -2 & 0.4 & -0.2\\4 & -0.2 & -1 & 4.2 & -0.6\end{array}$$

(d)

$$\begin{array}{|c|cccc}3 & 5 & 5 & 4 & 2\\1 & 0.4 & -2 & 0.4 & -0.2\\2 & 0.6 & 0 & 1.6 & -3.2\\4 & -0.2 & -1 & 4.2 & -0.6\end{array}$$

(e)

$$\begin{array}{|c|cccc}3 & 5 & 5 & 4 & 2\\1 & 0.4 & -2 & 0.4 & -0.2\\2 & 0.6 & 0 & 1.6 & -3.2\\4 & -0.2 & 0.5 & 4 & -0.5\end{array}$$

(f)

$$\begin{array}{|c|cccc}3 & 5 & 5 & 4 & 2\\1 & 0.4 & -2 & 0.4 & -0.2\\2 & 0.6 & 0 & 1.6 & -3.2\\4 & -0.2 & 0.5 & 4 & -0.5\end{array}$$

(g)

$$\begin{array}{|c|cccc}3 & 5 & 5 & 4 & 2\\1 & 0.4 & -2 & 0.4 & -0.2\\4 & -0.2 & 0.5 & 4 & -0.5\\2 & 0.6 & 0 & 1.6 & -3.2\end{array}$$

(h)

$$\begin{array}{|c|cccc}3 & 5 & 5 & 4 & 2\\1 & 0.4 & -2 & 0.4 & -0.2\\4 & -0.2 & 0.5 & 4 & -0.5\\2 & 0.6 & 0 & 0.4 & -3\end{array}$$

(i)

$$\underbrace{\begin{pmatrix}0 & 0 & 1 & 0\\1 & 0 & 0 & 0\\0 & 0 & 0 & 1\\0 & 1 & 0 & 0\end{pmatrix}}_{P}\underbrace{\begin{pmatrix}2 & 0 & 2 & 0.6\\3 & 3 & 4 & -2\\5 & 5 & 4 & 2\\-1 & -2 & 3.4 & -1\end{pmatrix}}_{A}=\underbrace{\begin{pmatrix}1 & 0 & 0 & 0\\0.4 & 1 & 0 & 0\\-0.2 & 0.5 & 1 & 0\\0.6 & 0 & 0.4 & 1\end{pmatrix}}_{L}\underbrace{\begin{pmatrix}5 & 5 & 4 & 2\\0 & -2 & 0.4 & -0.2\\0 & 0 & 4 & -0.5\\0 & 0 & 0 & -3\end{pmatrix}}_{U}$$

(j)

Výpočet inverzní matice

- **Výpočet inverzní matice pomocí LUP dekompozice**
  - Pomocí dekomponované matice A na L, U a P jsme již z předchozího popisu schopni vyřešit úlohu $Ax = b$ (pozn. LUP dekompozice není závislá na b).
  - Stejným postupem jsme schopni vyřešit i úlohu $Ax = e_i$ pro i od 1 do n (za předpokladu, že matice A má rozměr $n \times n$) kde e_i je jednotkový vektor.
  - Pokud spojíme n vektorů e_i pro i od 1 do n dohromady, dostáváme I_n (jednotkovou matici).
  - Úloha nalezení inverzní matice X k A představuje vyřešení $AX = I$.
  - Postupným spojením řešení x z $Ax = e_i$ od 1 do n dostáváme hledanou matici X z $AX = I$.
  - Složitost algoritmu nalezení inverzní matice k A je tedy

    LUP dekompozice v $\Theta(n^3)$ + (n × *dopředná* a *zpětná substituce* v $\Theta(n^2)$) = $\Theta(n^3)$