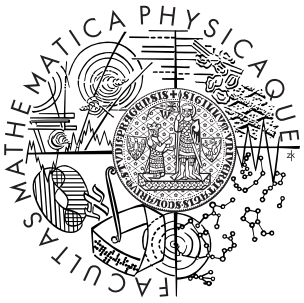
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Lukáš Trojek

Moderní metody LU rozkladu řídkých matic

Katedra numerické matematiky

Vedoucí bakalářské práce: Dipl.-Math. Jurjen Duintjer Tebbens, PhD., Ústav informatiky AV ČR

Studijní program: Obecná matematika

2010

Rád bych poděkoval mému vedoucímu Dipl.-Math. Jurjen Duintjer Tebbens, PhD. za ochotu, odborné vedení a pomoc podrobně porozumět problémům mé práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

V Praze 23.5. Lukáš Trojek

Obsah

Název práce: Moderní metody LU rozkladu řídkých matic
Autor: Lukáš Trojek
Katedra: Katedra numerické matematiky
Vedoucí bakalářské práce: Dipl.-Math. Jurjen Duintjer Tebbens, PhD., Ústav informatiky AV ČR
e-mail vedoucího: tebbens@cs.cas.cz

V této práci poskytneme stručný přehled některých moderních technik využívaných v LU rozkladu pro nalezení řešení velkých řídkých lineárních systémů. Zabýváme se řešením řídké trojúhelníkové soustavy, kterou také použijeme k popisu pojmu eliminační strom. Definujeme pojem supernode a popíšeme multifrontální metodu. Probereme několik metod minimalizace zaplnění. Práci uzavřeme seznamem některých populárních softwareových balíčků.

Klíčová slova: LU rozklad, řídká přímá metoda, eliminační strom, supernode, multifrontální metoda, minimální zaplnění

Title: Modern LU-factorization methods for sparse matrices
Author: Lukáš Trojek
Department:Department of Numerical Mathematics
Supervisor: Dipl.-Math. Jurjen Duintjer Tebbens, PhD., Institute of Computer Science AS CR
Supervisor's e-mail address: tebbens@cs.cas.cz

In the present work we give a brief survey of some modern techniques exploited in the LU-decomposition for the solution of large sparse linear systems. We consider sparse triangular solves which we use to describe the notion of elimination tree, we define supernodes and we address multifrontal implementation. We discuss several fill-minimizing strategies and we conclude with a list of some popular software packages.

Keywords: LU-decomposition, sparse direct method, elimination tree, supernode, multifrontal LU, minimal fill-in.

Úvod

V této práci budeme popisovat některé základní moderní techniky řešení řídkých lineárních systémů pomocí LU rozkladu. Důraz je kladen na využití řídkosti systému.

LU rozklad matice A definujeme jako rozklad

$$A = LU,$$

kde L je dolní trojúhelníková matice s jednotkovou hlavní diagonálou a U je horní trojúhelníková matice. LU rozklad je nejstarší způsob rozkladu matice založený na Gaussově eliminaci. Gaussova eliminace je dodnes jednou z nejdůležitějších součástí algoritmu pro řešení soustav lineárních rovnic, které počítají v konečném počtu kroků.

Nechť matice A je reálná symetrická pozitivně definitní. Pak rozklad

$$A = LL^T,$$

kde L je dolní trojúhelníková matice, nazýváme Choleského rozklad A.

V celém textu budeme o matici A předpokládat, že je čtvercová regulární dimenze n. Díky regularitě A existuje permutace, která zajišťuje proveditelnost LU rozkladu.

Pro snazší pochopení a jednoduchost textu předpokládáme symetrický nenulový vzor matice A, tj. prvek a_{ij} je nenulový, pravě když prvek a_{ji} je nenulový. Dále prvky, jež jsou nulové důsledkem numerického vynulování, nebudeme v textu považovat za nulové.

Hlavním vodítkem pro výběr jednotlivých technik popsaných v této práci byla kniha Davise [3]. Z důvodu omezeného rozsahu práce zde nejsou popsány důležité otázky jako jsou například: stabilita, pivotace na základě velikosti prvku, iterační zpřesnění, postordering, bloková verze LU rozkladu a jiné.

V textu se vyskytují některé termíny v angličtině, jelikož neexistuje buď žádný nebo jednotný překlad v češtině.

V první kapitole znázorníme některé popsané jevy pomocí experimentů v prostředí Matlab[1]. Jelikož naměřené časy v Matlabu mohou záviset na mnoha (i externích) faktorech, proto jsme pro maximální eliminaci těchto závislostí volili jednoduché akademické příklady. Napsané subroutiny prezentujeme v příloze této práce, kde zároveň popíšeme použitý formát pro uložení řídkých matic.

První kapitola je věnována řešení trojúhelníkové soustavy $Lx = b$, což je nezbytné pro vyřešení soustavy $Ax = b$ po aplikaci LU rozkladu či Choleského rozkladu. Právě pro řešení trojúhelníkových soustav, jež vznikly z těchto rozkladů, ukážeme jak sestrojit a použít eliminační strom, který vede k efektivnějšímu řešení. V druhé kapitole popíšeme dvě nejpoužívanější implementace LU rozkladu, a to left-looking LU a right-looking LU. Dále zde vysvětlíme základní princip multifrontální metody a supernodální implementaci. Ve třetí kapitole se zabýváme úlohou minimalizace zaplnění a stručně zde popisujeme některé metody pokoušející se nalézt minimální zaplnění, jmenovitě: redukce šířky pásu matice, blokově trojúhelníkový tvar, pořadí podle nejnižšího stupně a nested dissection. Čtvrtá kapitola obsahuje přehled několika populárních softwareových balíčků a jejich vlastností.

Lukáš Trojek, 23.5. 2010 v Praze

[1] www.mathworks.com

Kapitola 1

Řešení trojúhelníkové soustavy a eliminační strom

Řešení soustavy s dolní trojúhelníkovou maticí, $Lx = b$, je pro naši práci klíčové. Řešení $Lx = b$ je nezbytné k vyřešení $Ax = b$ pomocí LU rozkladu či Choleského rozkladu matice A.

1.1 Řídká pravá strana

Budeme řešit soustavu $Lx = b$ s řídkou maticí L a s řídkou pravou stranou b. Standardní postup řešení této soustavy, tedy přímý chod, je následující:

function $x = troj\_r(L, b)$
  for $i = 1 : n$ **do**
    $x_i = b_i$
    for $j < i$ **do**
      $x_i = x_i - l_{ij}x_j$
    end
    $x_i = x_i / l_{ii}$
  end

V předchozím algoritmu jsme k matici L přistupovali po řádcích, nyní budeme přistupovat po sloupcích. V $troj\_r$ je nultým krokem pro výpočet x_i přiřazení hodnoty b_i a dalším j-tým krokem je odečítání prvku $l_{ij}x_j$, pokud x_j je nenulové. Pro jednotlivá j můžeme tento odečet provádět současně u

všech x_i, jakmile je známa konečná hodnota pro x_j. Tedy využíváme právě všechny nenulové prvky j-tého sloupce matice L. Algoritmus s přístupem k matici po sloupcích bude vypadat takto:

```
function x = troj_s(L, b)
    x = b
    for j = 1 : n do
        if x_j ≠ 0
            for i, i > j ∧ l_ij ≠ 0 do
                x_i = x_i − l_ij x_j
            end
        end
    end
```

Tento algoritmus využívá řídkosti pravé strany, resp. řešení x.

Definice 1.1 (nenulový vzor)**.** *Nenulový vzor X vektoru x, je množina takových indexů j, pro které je x_j nenulové, tedy $X = \{j \mid x_j \neq 0\}$.*

Implementujeme-li algoritmus podle předchozího pseudokódu, získáme náklady $O(n)+O(|B|)+O(f)$, kde f je počet operací vykonaných v prostředí floating-points a $|B|$ je počet prvků nenulového vzoru B vektoru b. Ve většině případů je $|B| \ll f$, a proto jsou náklady v podstatě $O(n) + O(f)$. Ačkoli by se mohlo zdát, že jsme dosáhli dobrého výsledku, není tomu tak, n může vysoce převyšovat f. Například mějme vektor b nulový až na poslední prvek, b_n. Pak f je $O(1)$, ale celková složitost je $O(n)$.

Závislost nákladů na n je způsobena první **for** smyčkou. Algoritmus bychom vylepšili, kdybychom začínali se seznamem indexů, pro které je x_j nenulové, tedy množinou X. Tímto bychom mohli tuto smyčku vypustit a odstranili bychom přímou závislost algoritmu na n. Dosáhli bychom nákladů $O(|B|) + O(f)$. Mějme prozatím X vzestupně seřazenou. Algoritmus potom bude vypadat takto:

```
function x = troj_sX(L, b)
    x = b
    for  j ∈ X do
            for  i, i > j ∧ l_ij ≠ 0  do
                x_i = x_i − l_ij x_j
            end
    end
```

Jak určit množinu X s využitím teorie grafu říká následující věta.

Věta 1.1. *Definujme orientovaný graf $G_L = (V, H)$ s vrcholy $V = \{1...n\}$ a s hranami $H = \{(j, i)|l_{ij} \neq 0\}$. Označme $D(B)$ množinu všech dosažitelných vrcholů z množiny vrcholů B v grafu G_L, kde $B = \{i|b_i \neq 0\}$. Pak platí $X = D(B)$.*

Důkaz. Prvky vektoru x se stanou nenulovými právě ve dvou místech algoritmu $troj\_sX$, v prvním a posledním příkazu předchozího pseudokódu. Podmínky pro nenulovost prvku x_i můžeme zapsat takto:

$$b_i \neq 0 \Rightarrow x_i \neq 0 \tag{1.1}$$

$$(x_j \neq 0 \ \wedge \ \exists i, l_{ij} \neq 0) \Rightarrow x_i \neq 0 \tag{1.2}$$

Tyto podmínky lze přeformulovat do teorie grafu. Podle první části podmínky označíme vrcholy i takové, pro které je $b_i \neq 0$. Druhá podmínka říká: je-li označen vrchol j a existuje-li hrana z j do nějakého vrcholu i, pak označme vrchol i. Množina označených vrcholů v grafu G_L je onou množinou X. Tedy X je množina všech dosažitelných vrcholů G_L z množiny $B = \{i|b_i \neq 0\}$, což jsme chtěli dokázat. □

Množinu X můžeme získat pomocí algoritmu prohledávání do hloubky grafu G_L startujícím ve vrcholech z B.

Definice 1.2 (prohledávání do hloubky)**.** *Prohledávání do hloubky je grafový algoritmus pro průchod grafu. Postupuje stále dál od počátečního vrcholu dosud neprozkoumaným směrem. Pokud narazí na vrchol, ze kterého už nelze dále pokračovat (nemá žádné následníky nebo byli všichni navštíveni), vrací se zpět do nejbližšího vrcholu, odkud lze pokračovat. Algoritmus končí, jakmile navštíví všechny dosažitelné vrcholy.*

Výsledkem algoritmu prohledávání do hloubky je seznam vrcholů s topologickým uspořádáním.

Definice 1.3 (topologické uspořádaní)**.** *Buď $G = (V, H)$ orientovaný graf s vrcholy V a hranami H. Topologické uspořádání vrcholů grafu G je taková posloupnost vrcholů grafu v_1, v_2, ..., $v_{|V|}$, že kdykoliv vede cesta z v_i do v_j, platí $i < j$.*

Další možností jak získat X je prohledáváním do šířky. Tento algoritmus však uspořádá vrcholy podle nejkratší vzdálenosti od počátečního vrcholu.

Definice 1.4 (prohledávání do šířky)**.** *Prohledávání do šířky je grafový algoritmus pro průchod grafu. V prvním kroku prozkoumává všechny sousední vrcholy počátečního vrcholu. V dalším kroku postupně prozkoumá všechny sousední vrcholy těchto sousedních vrcholů. Algoritmus se takto opakuje, dokud nejsou navštíveny všechny dosažitelné vrcholy.*

Při výpočtu x v soustavě $Lx = b$ může být uskutečněn výpočet $x_i = x_i - l_{ij}x_j$, jakmile je známo x_j. První se musí vypočítat ty x_k, ze kterých vrcholů k vede hrana do j, protože právě tyto x_k se podílejí na výpočtu x_j. Toto zajišťuje topologické uspořádání. A proto pro určení množiny X volíme algoritmus prohledávání do hloubky.

Na příkladu matice L z (1.3) názorně ukážeme, že topologické uspořádání vrcholů stačí.

$$L = \begin{pmatrix} \bullet & & & & & \\ & \bullet & & & & \\ * & * & \bullet & & & \\ * & & * & \bullet & & \\ & & & & \bullet & \\ * & & * & * & * & \bullet \end{pmatrix} \tag{1.3}$$

V (1.3) symbol $*$ značí nenulový prvek L a symbol $\bullet$ označuje diagonální prvek L.

Mějme pravou stranu b takovou, že $B = \{1, 5\}$. Pomocí prohledávání do hloubky v G_L vypíšeme dosahy každého z těchto vrcholů:

- $D(1) = \{1, 3, 4, 6\}$
- $D(5) = \{5, 6\}$

Při prohledávání do hloubky nezáleží, ve kterém vrcholu začneme hledat, zvolme vrchol 1. První získáme celou množinu $D(1)$. Pak dále na první

pozici této množiny postupně přidáme vrcholy z $D(5)$, které se v množině ještě nevyskytují. Vznikne $D(\{1,5\}) = \{5,1,3,4,6\}$. Rozdíl uspořádaní této množiny oproti úplnému uspořádaní je ve vrcholu 5. Výpočet $x_i = x_i - l_{i5}x_5$ však můžeme provést nezávisle na hodnotách x_1, x_3 a x_4 pro všechny indexy $i > 5$ (v našem příkladě $i = 6$).

Následující funkce *dosah* vypočte množinu X pomoci funkce *pruchod_h*, což je průchod do hloubky zapsaný rekurzivně, nenavštívené vrcholy budeme v pomocném poli C značit 0, navštívené 1:

```
function X = dosah(L, B)
    C = zeros
    for i, b_i ≠ 0 do
        if C_i ≠ 1
            pruchod_h(i)
        end
    end

function pruchod_h(j)
    C_j = 1
    for i, l_ij ≠ 0 do
        if C_i ≠ 1
            pruchod_h(i)
        end
    end
    X = {j, X}
```

Náklady prohledávání do hloubky jsou úměrně součtu počtu startovacích vrcholů a počtu prošlých hran. To znamená, že náklady funkce *dosah* jsou $O(|B|) + O(f)$, stejně jako tomu bylo pro funkci *troj_sX*. Tedy celkové náklady řešení úlohy $Lx = b$ s použitím prohledávání do hloubky jsou $O(|B|) + O(f)$.

1.2 Eliminační strom

Ve speciálním případě kdy je potřeba nejen řešit $Lx = b$, ale i najít faktorizaci $A = LU$ respektive $A = LL^T$, můžeme k nalezení X použít tzv. eliminační strom. Pojem eliminační strom v následujícím definujeme pomocí Choleského rozkladu.

Definice 1.5 (eliminační strom)**.** *Nechť L je Choleského faktor matice A. Označme $V = \{1, ..., n\}$ a $H = \{(j, i) \mid j = \min\{k > i \mid l_{ki} \neq 0\}\}$. Pak graf $S = (V, H)$ se nazývá eliminační strom matice A.*

Poznámka 1.1 (vlastnosti eliminačního stromu)**.** *Nechť $S = (V, H)$ je eliminační strom, pak platí:*

- *vrchol i je otec vrcholu j, pokud první nenulový mimodiagonální prvek j-tého sloupce má řádkový index i*
- *vrchol j je kořenem S, pokud j-tý sloupec nemá žádný nenulový mimodiagonální prvek, tedy nemá otce*
- *eliminační strom může být také les*

Eliminační strom je v určitém smyslu minimální graf vypovídající o nenulových prvcích Choleského faktoru L. Fakta, že lze vybrané nenulové prvky z grafu G_L vypustit a že eliminační strom stačí k nalezení nenulového vzoru X, vyplývají z následujících dvou výsledků.

Lemma 1.1. *Nechť $A = LL^T$ je Choleského rozklad. Nechť $i < j < k$ a prvky l_{ji} a l_{ki} jsou nenulové, pak l_{kj} je nenulový.*

Důkaz. Uvažujme 2×2 blokový rozklad $LL^T = A$

$$\begin{pmatrix} L_{n-1} & \\ l_n^T & l_{nn} \end{pmatrix} \begin{pmatrix} L_{n-1}^T & l_n \\ & l_{nn} \end{pmatrix} = \begin{pmatrix} A_{n-1} & a_n \\ a_n^T & a_{nn} \end{pmatrix},$$

kde L_{n-1} a A_{n-1} jsou submatice velikosti $(n-1) \times (n-1)$. Tímto nám vznikli tři rovnice: $L_{n-1}L_{n-1}^T = A_{n-1}$, $L_{n-1}l_n = a_n$ a $l_n^T l_n + l_{nn}^2 = a_{nn}$. Budeme-li rovnici $L_{n-1}L_{n-1}^T = A_{n-1}$ rozkládat stejným způsobem jako $LL^T = A$, pak se po $n - k - 1$ krocích dostaneme k blokovému rozdělení

$$\begin{pmatrix} L_{k-1} & \\ l_k^T & l_{kk} \end{pmatrix} \begin{pmatrix} L_{k-1}^T & l_k \\ & l_{kk} \end{pmatrix} = \begin{pmatrix} A_{k-1} & a_k \\ a_k^T & a_{kk} \end{pmatrix},$$

a rovnici

$$L_{k-1}l_k = a_k,$$

kde L_{k-1} je submatice L velikosti $(k-1)\times(k-1)$, a_k a l_k je prvních $k-1$ prvků k-tých sloupců příslušných matic. Prvek l_{ki} je i-tá složka řádku l_k^T. Protože l_k^T není nic jiného než transponovaný l_k, je hodnota prvku l_{ki} rovna hodnotě prvku l_{ik}, což je i-tá složka sloupce l_k. Z předpokladu $l_{ki} \neq 0$ plyne $l_{ik} \neq 0$. Z podmínky (1.2) z důkazu Věty 1.1 dostáváme: jelikož i-tý prvek l_k je nenulový a l_{ji} je nenulový, proto j-tá složka sloupce l_k je nenulová, tedy $l_{jk} \neq 0$. Po transpozici sloupce l_k dostáváme řádek l_k^T s nenulovou j-tou složkou, což je k-tý řádek matice L, a proto platí $l_{kj} \neq 0$. □

Poznámka 1.2. *Blokové rozdělení $LL^T = A$ v předchozím důkazu dává návod na možnost výpočtu Choleského rozkladu. Rekurzivně provádíme rozklady soustav $LL^T = A, L_{n-1}L_{n-1}^T = A_{n-1}^T, ...L_1L_1^T = A_1$.*

Věta 1.2. *Nechť $S = (V, H)$ je eliminační strom matice A a nechť L je Choleského faktor A. Existovala-li cesta z i do k v grafu matice L, kde $i, k \in V$, pak existuje i ve stromu S.*

Důkaz. Nechť cesta z i do k je hrana (i,k) v G_L , tj. $l_{ki} \neq 0$. Pokud $k = \min\{m > i | l_{mi} \neq 0\}$, pak jsme hotovi, neboť z definice eliminačního stromu je hrana (i,k) obsažena přímo v S. Pokud je toto minimum rovno $j < k$, pak podle Lemmatu 1.1 platí $l_{kj} \neq 0$. Proto rozeberme umístění prvku l_{kj} v j-tém sloupci. Je-li $k = \min\{m > j | l_{mj} \neq 0\}$, pak hrana (j,k) je obsažena v S, to víme i o hraně (i,j), a tedy cesta z i do k existuje. Pokud $j_2 = \min\{m > j | l_{mj} \neq 0\}$, kde $j_2 < k$, pak pro l_{kj_2} provedeme stejný proces jako pro l_{kj}. Byla-li potřeba tento proces provést p krát, pak jsme získali cestu postupně navštěvující vrcholy $i, j, j_2, ..., j_p, k$ a tím cestu z i do k.

Vede-li cesta z i do k přes více než jednu hranu, pak existuje posloupnost vrcholů $i = k_1, k_2, ..., k_q = k$, kde $q > 2$, takových, že $l_{k_{j+1}k_j} \neq 0$, pro $j = 1, ..., q-1$, a pro všechna taková $l_{k_{j+1}k_j}$ použijeme předchozí část důkazu. □

Je vidět, že eliminační strom má obecně méně hran než graf matice L a přesto víme, že zachovává cesty mezi jednotlivými vrcholy. Díky těmto vlastnostem je výhodnější k nalezení množiny X použít eliminační strom a tím urychlit algoritmus pro řešení $Lx = b$.

Poznamenejme, že Lemma 1.1 platí jen pro Choleského faktor; pro obecnou dolní trojúhelníkovou matici L musíme k nalezení X použít strategii ze sekce 1.1.

Na základě následující věty jsme schopni sestrojit eliminační strom S matice A bez znalosti struktury L.

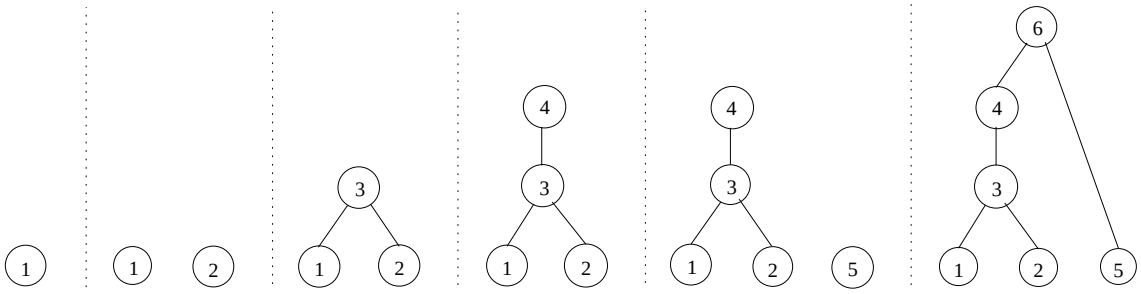
Věta 1.3. *Nechť $A = LL^T$ je Choleského rozklad. Nechť $a_{ij} \neq 0$ a $i > j$, pak vrchol j je potomek vrcholu i v eliminačním stromě S, ekvivalentně cesta z j do i v S existuje.*

Důkaz. Důkaz této věty nalezneme v knize [9]. □

Označme S_k eliminační strom submatice L_k, prvních k řádků a sloupců L. Eliminační podstrom S_1 je vrchol. Předpokládejme, že S_{i-1} již známe, $i \in \{2, ..., n-1\}$. Pro získání S_i z S_{i-1} je potřeba přiřadit vrchol i k podstromu S_{i-1}, což spočívá v nalezení synů vrcholu i. Pro všechna $j < i$ taková, že $a_{ij} \neq 0$ podle Věty 1.3 platí, že cesta z j do i v S existuje, a tedy nutně i v S_i. Tato cesta vede skrz S_{i-1} dokud nenarazí na kořen tohoto podstromu. Tento kořen pak musí být synem i v důsledku věty. Tímto přiřazením získáváme S_i, a tedy postupně $S = S_n$. Náklady postavení tohoto stromu jsou $O|A|$ [3, Strana 41].

Na Obrázku 1.1 názorně ukážeme, jak vypadají jednotlivé podstromy pro matici A z (1.4). Symboly v A značí totéž jako v matici (1.3).

$$A = \begin{pmatrix} \bullet & & * & * & & * \\ & \bullet & * & & & \\ * & * & \bullet & * & & * \\ * & & * & \bullet & & * \\ & & & & \bullet & * \\ * & & * & * & * & \bullet \end{pmatrix} \tag{1.4}$$



Obrázek 1.1: Postupná výstavba eliminačního stromu S matice A.

n	100	1000	10000	100000	1000000
troj_s	0.00012	0.0008	0.008	0.09	0.84
troj_sX	0.0013	0.0013	0.0013	0.0015	0.0015
troj_SXE	0.0011	0.0011	0.0012	0.0013	0.0013

Tabulka 1.1: Naměřené časy funkcí *troj_s*, *troj_sX* a *troj_sXE* pro jednotlivé dimenze matice

Tvrzení o nákladech řešení řídké trojúhelníkové soustavy, resp. o závislosti nákladů na dimenzi matice, demonstrujeme na funkcích *troj_s*, *troj_sX* a *troj_sXE* naprogramovaných v Matlabu. Funkce *troj_s* odpovídá stejnojmennému pseudokódu, funkce *troj_sX* a *troj_sXE* odpovídají pseudokódu *troj_sX*, kde funkce *troj_sX* nalezne množinu X pomocí prohledávaní do hloubky grafu G_L a funkce *troj_sXE* pomocí eliminačního stromu. Subroutiny těchto funkcí nalezneme v příloze této práce. Experimenty byly prováděny na matici L, jejíž prvních šest řádků a sloupců tvořila dolní trojúhelníková část z matice A z (1.4) a zbylou nenulovou částí byla pouze jednotková hlavní diagonála. Jako nenulové prvky pravé strany jsme volili prvky na 1. a 5. pozici. Nenulovým prvkům L a b jsme přiřadili hodnotu 1. Výsledky těchto experimentů jsou zapsány v Tabulce 1.1 (naměřené časy jsou uvedeny v sekundách).

Kapitola 2

Zápisy LU rozkladu a multifrontální metoda

Algoritmus Gaussovy eliminace používá tři cykly, jeden vnější od 1 do n a dva vnitřní pro řádkové a sloupcové indexy. Různá uspořádání jednotlivých cyklů vedou k různým implementacím s odlišnými vlastnostmi, zejména vzhledem k přístupu k datům. Zde jsou popsaný dvě třídy implementací, a to left-looking LU a right-looking LU. Zároveň vysvětlíme základní principy supernodálních a multifrontálních implementací. Up-looking LU, jakožto nejméně používanou implementaci, zde nepopíšeme. Stručný popis byl již uveden v Poznámce 1.2.

2.1 Left-looking LU a supernodální implementace

Algoritmus left-looking LU rozklad počítá sloupce matic L a U současně. V k-tém kroku algoritmu pracuje s prvními $k-1$ sloupci matic L a U, z nichž získá k-tý sloupec L a k-tý sloupec U. Tedy sloupce jsou vypočteny postupně zleva doprava. Algoritmus můžeme odvodit na základě následujícího 3×3 blokově rozděleného rozkladu

$$\begin{pmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & l_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & u_{12} & U_{13} \\ & u_{22} & u_{23} \\ & & U_{33} \end{pmatrix} = \begin{pmatrix} A_{11} & a_{12} & A_{13} \\ a_{21} & a_{22} & a_{23} \\ A_{31} & a_{32} & A_{33} \end{pmatrix}. \qquad (2.1)$$

Matice L_{11} je submatice velikosti $(k-1) \times (k-1)$, l_{21} je prvních $k-1$ prvků k-tého řádku L. Vektor l_{32} je k-tý sloupec L s prvky $k+1$ až n. Matice L_{31} a L_{33} jsou příslušné části $(k+1)$-ního až n-tého řádku L. Prvky matic A a U jsou definovány analogicky. Předpokládejme, že známe prvních $k-1$ sloupců matic L a U, pak k-té sloupce těchto matic získáme z následujících tří rovnic:

- $L_{11}u_{12} = a_{12}$
- $l_{21}u_{12} + u_{22} = a_{22}$
- $L_{31}u_{12} + l_{32}u_{22} = a_{32}$

Z první rovnice získáme u_{12} jako řešení trojúhelníkové soustavy rovnic. Z druhé dostaneme prvek u_{22}. Nakonec za využití předchozích výsledků vypočteme l_{32}. Předchozí rovnice můžeme přepsat tak, že tyto výsledky dostaneme jako řešení řídké trojúhelníkové soustavy

$$\begin{pmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & 0 & I \end{pmatrix} \begin{pmatrix} u_{12} \\ u_{22} \\ l_{32}u_{22} \end{pmatrix} = \begin{pmatrix} a_{12} \\ a_{22} \\ a_{32} \end{pmatrix}.$$

V tzv. supernodální implementaci prostřední sloupce rozkladu (2.1) představují sloupcové bloky. Reprezentaci pomocí bloků volíme, pokud vrcholy v tomto bloku splňují následující definici

Definice 2.1 (supernode pro left-looking LU)**.** *Supernode matice U k vrcholu i velikosti t je množina sousedních vrcholů $i, ..., i+t-1$, jejíž sloupce mají od pozice 1 do pozice $i-1$ stejný nenulový vzor a od pozice i do pozice $i+t-1$ pouze nenulové prvky.*

Sloupcový blok příslušný k vrcholům supernode matice U k vrcholu i velikosti t může být uložen jako hustá matice velikosti $|U_{i+t-1}| \times t$, což je původní blok s vynechanými nulovými řádky, kde U_{i+t-1} je nenulový vzor $(i+t-1)$-tého sloupce U.

Výhodou supernodální implementace je, že umožňuje využít optimalizované strategie pro husté matice. Například můžeme použít balíček BLAS (Basic Linear Algebra Subprograms) s doposud nejefektivnějšími algoritmy.

Využijeme-li supernodální implementaci, pak z blokového rozdělení (2.1) dostaneme rovnice:

- $L_{11}U_{12} = A_{12}$, jež řešíme jako trojúhelníkovou soustavu s více pravými stranami se stejným nenulovým vzorem
- $L_{21}U_{12} + L_{22}U_{22} = A_{22}$, kde $L_{21}U_{12}$ vypočteme jako součin hustých matic, a pak vypočteme $L_{22}U_{22}$ jako hustý LU rozklad matice $A_{22} - L_{21}U_{12}$
- $L_{31}U_{12} + L_{32}U_{22} = A_{32}$, zde $L_{31}U_{12}$ vypočteme jako součin redukované řídké matice a husté matice, a L_{32} získáme jako řešení husté trojúhelníkové soustavy s více pravými stranami, $U_{22}^T L_{32}^T = (A_{32} - L_{31}U_{12})^T$

2.2 Right-looking LU a multifrontální metoda

Pro odvození metody right-looking LU uvažujme blokové rozdělení (2.2)

$$\begin{pmatrix} 1 & \\ l_{21} & L_{22} \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} \\ & U_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & A_{22} \end{pmatrix}. \tag{2.2}$$

vektor l_{21} je první sloupec L bez prvního prvku a matice L_{22} je matice L bez prvního řádku a sloupce. Matice U a A jsou rozděleny analogicky. Z tohoto vztahu dostáváme čtyři rovnice

- $u_{11} = a_{11}$
- $u_{12} = a_{12}$
- $l_{21}u_{11} = a_{21}$
- $l_{21}u_{12} + L_{22}U_{22} = A_{22}$ (2.3)

Z těchto rovnic přímo vypočteme první sloupec L a první řádek U. Zbylé matice L_{22} a U_{22} vypočteme rekurzivně. Je vidět, že v dalším kroku pracujeme pouze se zbylými pravými částmi matic, a že v k-tém kroku vypočteme k-tý sloupec L a k-tý řádek U. Předchozí rovnice vedou k rekurzivnímu algoritmu, jež zapíšeme následovně:

```
function LU_right(A)
    for k = 1 : n do
        for j = k : n do
            u_kj = a_kj
        end
        for i = k + 1 : n do
            l_ik = a_ik/u_kk
            for j = k + 1 : n do
                a_ij = a_ij − l_ik u_kj
            end
        end
    end
```

Nyní vysvětlíme, jak funguje multifrontální metoda založena na right-looking LU algoritmu. Multifrontální metoda dává návod, jak a kdy odečítat 1-dimensionální vnější součin v rovnici

$$A_{22} = A_{22} - l_{21}u_{12}$$

odpovídající (2.3) nebo poslednímu příkazu algoritmu *LU_right*. Namísto vytvoření iterované matice A_{22} se vytvoří frontální matice obsahující 1-dimensionální vnější součin a odečet vnějšího součinu je pak odložen na iteraci, kdy je to nezbytné.

Definice 2.2 (frontální matice)**.** *Nechť $l^{(k)}$ je vektor všech nenulových pod-diagonálních prvků k-tého sloupce matice L v tomtéž pořadí, nechť $u^{(k)}$ je vektor všech nenulových prvků zprava od diagonály k-tého řádku matice U v tomtéž pořadí a nechť u_{kk} je k-tý diagonální prvek U. Frontální matice $F^{(k)}$ vrcholu k je hustá matice, jejíž první sloupec je*

$$\begin{pmatrix} u_{kk} \\ l^{(k)}u_{kk} \end{pmatrix},$$

první řádek je

$$\begin{pmatrix} u_{kk}, & u^{(k)} \end{pmatrix}$$

a zbylá submatice je vnější součin $l^{(k)}u^{(k)}$, tj.

$$F^{(k)} = \begin{pmatrix} u_{kk}, & u^{(k)} \\ l^{(k)}u_{kk} & l^{(k)}u^{(k)} \end{pmatrix}.$$

Poznamenejme, že vektor $l^{(k)}u_{kk}$ v prvním sloupci frontální matice $F^{(k)}$ odpovídá prvkům l_{ik} v pseudokódu *LU_right* před škálováním $l_{ik} = a_{ik}/u_{kk}$, kde $k = 1, ..., n; i = k + 1, ..., n$. Vnější součin $l^{(k)}u^{(k)}$ však toto škálování již obsahuje a odpovídá proto přímo součinu $l_{ik}u_{kj}$ z posledního příkazu algoritmu *LU_right*.

Jak efektivně spočítat vnější součin $l^{(k)}u^{(k)}$ z $F^{(k)}$ říká následující věta.

Věta 2.1. *Nechť matice A má blokovou strukturu*

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix},$$

kde $A_{11} \in \mathbb{R}^{t\times t}$, $A_{12} \in \mathbb{R}^{(n-t)\times t}$, $A_{21} \in \mathbb{R}^{t\times(n-t)}$ a $A_{22} \in \mathbb{R}^{(n-t)\times(n-t)}$. Pak po t krocích Gaussovy eliminace máme rozklad

$$A = \begin{pmatrix} L_{11} & \\ A_{21}U_{11}^{-1} & I_{n-t} \end{pmatrix} \begin{pmatrix} U_{11} & L_{11}^{-1}A_{12} \\ & C \end{pmatrix},$$

kde $L_{11}U_{11}$ je LU rozklad matice A_{11}, I_{n-t} je jednotková matice dimenze $n-t$ a C je Schurův komplement $C = A_{22} - A_{21}A_{11}^{-1}A_{12}$.

Důkaz. Po t krocích Gaussovy eliminace bude LU rozklad A vypadat následovně

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} = \begin{pmatrix} L_{11} & \\ L_{21} & I_{n-t} \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & U_{22} \end{pmatrix},$$

Toto blokové rozdělení vede na čtyři rovnice:

- $L_{11}U_{11} = A_{11}$
- $L_{21}U_{11} = A_{21} \Rightarrow L_{21} = A_{21}U_{11}^{-1}$
- $L_{11}U_{12} = A_{12} \Rightarrow U_{12} = L_{11}^{-1}A_{12}$
- $L_{21}U_{12} + U_{22} = A_{22} \Rightarrow U_{22} = A_{22} - L_{21}U_{12}$

Do upravené čtvrté rovnice dosadíme vztahy pro L_{21} a U_{12} z předchozích dvou rovnic a dále dostáváme

$$\underline{U_{22}} = A_{22} - (A_{21}U_{11}^{-1})(L_{11}^{-1}A_{12}) = A_{22} - A_{21}(U_{11}^{-1}L_{11}^{-1})A_{12} =$$

$$A_{22} - A_{21}(L_{11}U_{11})^{-1}A_{12} = \underline{A_{22} - A_{21}A_{11}^{-1}A_{12}},$$

kde poslední rovnost plyne z první rovnice. □

Vnější součin $l^{(k)}u^{(k)}$ v matici $F^{(k)}$ je podle předchozí věty s volbou $t = 1$ Schurův komplement, který vzniká při aplikaci jednoho kroku Gaussovy eliminace na matici

$$\begin{pmatrix} u_{kk} & u^{(k)} \\ l^{(k)}u_{kk} & 0 \end{pmatrix}.$$

Kroky Gaussovy eliminace husté matice provedeme velice efektivně pomocí balíčku BLAS. Věta je formulovaná pro obecný počet kroků Gaussovy eliminace. Více kroků je potřeba provádět, jak nyní vysvětlíme, v případě supernodální implementace.

Definice 2.3 (supernode pro right-looking LU)**.** *Supernode $SN_i^{(t)}$ matice L vrcholu i velikosti t je množina sousedních vrcholů $i, ..., i+t-1$, jejíž sloupce mají od pozice i do pozice $i+t-1$ pouze nenulové prvky a od pozice $i+t$ mají stejný nenulový vzor.*

Sloupcový (řádkový) blok supernode $SN_i^{(t)}$ jsou sloupce (řádky) s prvky od i-té pozice odpovídající vrcholům supernode $SN_i^{(t)}$.

Definici 2.2 můžeme zobecnit pro případ, kdy multifrontální metoda používá, analogicky jako left-looking LU, supernodální implementaci.

Definice 2.4 (supernodální frontální matice)**.** *Nechť $SN_k^{(t)}$ je supernode matice L. Nechť $L^{(k)}$ je sloupcový blok supernode $SN_k^{(t)}$ matice L s vynechanými nulovými řádky a bez diagonálního bloku označeného D_{kk}. Nechť $L_{kk}U_{kk} = D_{kk}$ je LU rozklad husté matice D_{kk} a nechť $U^{(k)}$ je řádkový blok supernode $SN_k^{(t)}$ matice U s vynechanými nulovými sloupci bez diagonálního bloku D_{kk}. Frontální matice $F^{(k,k+t-1)}$ vrcholů $k, ..., k+t-1$ je hustá matice, jejíž prvních t sloupců je*

$$\begin{pmatrix} D_{kk} \\ L^{(k)}U_{kk} \end{pmatrix},$$

jejíž prvních t řádků je

$$\begin{pmatrix} D_{kk} & L_{kk}U^{(k)} \end{pmatrix}$$

a zbylá submatice je vnější součin $L^{(k)}U^{(k)}$, tj.

$$F^{(k,k+t-1)} = \begin{pmatrix} D_{kk} & L_{kk}U^{(k)} \\ L^{(k)}U_{kk} & L^{(k)}U^{(k)} \end{pmatrix}.$$

Vnější součin $L^{(k)}U^{(k)}$ v matici $F^{(k,k+t-1)}$ je podle předchozí věty Schurův komplement, který vzniká při aplikaci t kroků Gaussovy eliminace na matici

$$\begin{pmatrix} D_{kk} & L_{kk}U^{(k)} \\ L^{(k)}U_{kk} & 0 \end{pmatrix}.$$

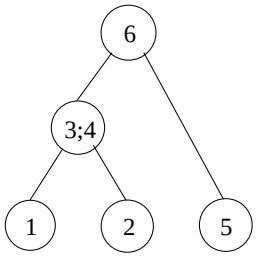
Provedení t kroků Gaussovy eliminace nám zároveň dává LU rozklad hustého bloku D_{kk} a konečný tvar sloupců $L^{(k)}$ a řádků $U^{(k)}$. Podobně jako v případě $t = 1$ nám Věta 2.1 zaručí, že vnější součin $L^{(k)}U^{(k)}$ odpovídá součinu v posledním příkazu algoritmu LU_right (resp. jeho supernodální verze).

Vzájemné vztahy frontálních matic jsou popsány v tzv. assembly tree, což je pozměněný eliminační strom S. Vrcholy představují frontální matice a některé vrcholy zanikly v důsledku vytvoření supernodálních frontálních matic.

Vrcholy v assembly tree, které mají stejného otce, si navzájem neovlivňují prvky frontální matice, a tak se jejich frontální matice mohou počítat nezávisle, tedy paralelně. To znamená, že odečtení příslušných vnějších součinů můžeme odložit. Tento postup výrazně snižuje výpočetní čas.

Nyní ukážeme, jak konkrétně funguje multifrontální metoda na příkladu matice A z (1.4). Její assembly tree $\widehat{S}$ nalezneme na Obrázku 2.1.

$$A = \begin{pmatrix} \bullet & & * & * & & * \\ & \bullet & * & & & \\ * & * & \bullet & * & & * \\ * & & * & \bullet & & * \\ & & & & \bullet & * \\ * & & * & * & * & \bullet \end{pmatrix}$$



Obrázek 2.1: Assembly tree $\widehat{S}$ matice A.

Postupujme podle algoritmu funkce $LU\_right(A)$ psaného výše. Kroky, které se přímo netykají multifrontální metody, nebudeme v textu vypisovat,

tedy budeme vždy pro jednotlivá k mluvit o posledním příkazu algoritmu, $a_{ij} = a_{ij} - l_{ik}u_{kj}$, kde $i, j = k+1, ..., n$.

Pro $k = 1$ vznikne $F^{(1)}$, jejíž první sloupec má tvar

$$\begin{pmatrix} u_{11} \\ l^{(1)}u_{11} \end{pmatrix} \equiv \begin{pmatrix} a_{11} \\ a_{31} \\ a_{41} \\ a_{61} \end{pmatrix},$$

a první řádek má tvar

$$\begin{pmatrix} u_{11} & u^{(1)} \end{pmatrix} \equiv \begin{pmatrix} a_{11} & a_{13} & a_{14} & a_{16} \end{pmatrix}.$$

Po jednom kroku Gaussovy eliminace získáváme vnější součin $l^{(1)}u^{(1)}$. Podíváme-li se na assembly tree $\widehat{S}$ na Obrázku 2.1, vidíme, že odečet $l^{(1)}u^{(1)}$ můžeme odložit až do kroku $k = 3$.

Pro $k = 2$ je první sloupec a řádek druhé frontální matice roven

$$\begin{pmatrix} u_{22} \\ l^{(2)}u_{22} \end{pmatrix} \equiv \begin{pmatrix} a_{22} \\ a_{32} \end{pmatrix}, \quad \begin{pmatrix} u_{22} & u^{(2)} \end{pmatrix} \equiv \begin{pmatrix} a_{22} & a_{23} \end{pmatrix}.$$

Jedním krokem Gaussovy eliminace $F^{(2)}$ získáme $l^{(2)}u^{(2)}$.

Vrchol číslo 3 v $\widehat{S}$ již nemá stejného otce jako předchozí vrchol, a tak uskutečníme odečet vnějších součinů obsažených v $F^{(1)}$ a $F^{(2)}$,

$$\begin{pmatrix} a_{33} & a_{34} & a_{36} \\ a_{43} & a_{44} & a_{46} \\ a_{63} & a_{64} & a_{66} \end{pmatrix} = \begin{pmatrix} a_{33} & a_{34} & a_{36} \\ a_{43} & a_{44} & a_{46} \\ a_{63} & a_{64} & a_{66} \end{pmatrix} - l^{(1)}u^{(1)},$$

$$a_{33} = a_{33} - l^{(2)}u^{(2)}.$$

Matice $F^{(1)}$ a $F^{(2)}$ již dále nepotřebujeme.

Dále si všimněme, že vrcholy 3 a 4 tvoří supernode. Proto kroky $k = 3$ a $k = 4$ sloučíme do jednoho. Vytvoříme supernodální frontální matici $F^{(3,4)}$, jejichž první dva sloupce budou

$$\begin{pmatrix} D_{33} \\ L^{(3)}U_{33} \end{pmatrix} \equiv \begin{pmatrix} a_{33} & a_{34} \\ a_{43} & a_{44} \\ a_{63} & a_{64} \end{pmatrix}$$

a první dva řádky budou

$$\begin{pmatrix} D_{33} & L_{33}U^{(3)} \end{pmatrix} \equiv \begin{pmatrix} a_{33} & a_{34} & a_{36} \\ a_{43} & a_{44} & a_{46} \end{pmatrix}.$$

Zde je D_{33} čtvercová matice dimenze 2×2. Dvěma kroky Gaussovy eliminace matice $F^{(3,4)}$ dostaneme vnější součin $L^{(3)}U^{(3)}$, zároveň dostaneme LU rozklad matice D_{33} a konečný tvar prvků v $L^{(3)}$ a $U^{(3)}$. Ze struktury $\widehat{S}$ vyčteme, že vrcholy (3,4) a 5 mají stejného otce vrchol 6, a tak odečet vnějšího součinu obsaženého v $F^{(3,4)}$ provedeme pak společně s odečtením vnějšího součinu v $F^{(5)}$.

V kroku $k = 5$ je první sloupec a řádek matice $F^{(5)}$ následující

$$\begin{pmatrix} u_{55} \\ l^{(5)}u_{55} \end{pmatrix} \equiv \begin{pmatrix} a_{55} \\ a_{65} \end{pmatrix}, \qquad \begin{pmatrix} u_{55} & u^{(5)} \end{pmatrix} \equiv \begin{pmatrix} a_{55} & a_{56} \end{pmatrix}.$$

Jedním krokem Gaussovy eliminace $F^{(5)}$ získáme vnější součin $l^{(5)}u^{(5)}$, a pak provedeme odečet vnějších součinů z $F^{(3,4)}$ a $F^(5)$:

$$a_{66} = a_{66} - L^{(3)}U^{(3)} - l^{(5)}u^{(5)}.$$

V posledním kroku $k = 6$ výpočet LU rozkladu ukončíme přiřazením $u_{66} = a_{66}$.

Poznamenejme, že v tomto příkladu nevzniká žádné zaplnění. Příklady se zaplněním vypadají koncepčně stejně.

Kapitola 3

Minimalizace zaplnění

Hledat minimální zaplnění matice je velmi důležité jak kvůli potencionálnímu nedostatečnému místu v paměti, tak k dosáhnutí co nejnižších nákladů při řešení rovnic $Lx = b$ a $Uz = x$. Pro danou matici A můžeme zkusit nalézt sloupcové permutace Q a řádkové permutace P tak, aby počet nenulových prvků v LU rozkladu matice PAQ byl minimální. Permutace P také slouží k nalezení pivotního prvku, což probíhá v průběhu algoritmu. Dosáhnout minimálního zaplnění však obecně nelze v rozumném čase, a tak se používají různé heuristiky. V následujících podkapitolách velmi stručně popíšeme některé metody pokoušející se o minimalizaci zaplnění matice.

3.1 Redukce šířky pásu matice

Šířku pásu matice můžeme měřit následujícími způsoby:

Definice 3.1 (profile a bandwidth)**.** *Nechť A má nenulové prvky na diagonále. Profile matice A je roven*

$$\sum_{j=1}^{n}(j - \min\ A_j)$$

a bandwidth je

$$\max_{j=1,..,n}(j - \min\ A_j),$$

kde A_j je nenulový vzor j-tého sloupce A.

Permutace P, jež sníží hodnotu profile nebo bandwidth matice A, často zajišťuje dobré uspořádání pro LU rozklad matice PAP^T. Například je

známo, že profile Choleského rozkladu L je identický s profilem původní matice $A = LL^T$ [11, Strana 57].Nalezení nejnižšího profile či bandwidth je velice náročné, a proto se k tomuto účelu využívá heuristik, např. reverse Cuthill-McKee algoritmus jež nalezneme v [2] .

3.2 Blokově trojúhelníkový tvar

Matice A má vlastnost Hall property, pokud každá množina k sloupců má nenulové prvky aspoň v k řádcích pro všechna $k \in \{1, ..., n\}$. Nechť matici A má vlastnost Hall property, pak ji můžeme permutovat na tvar

$$PAQ = \begin{pmatrix} A_{11} & \dots & A_{1k} \\ & \ddots & \vdots \\ & & A_{kk} \end{pmatrix},$$

kde diagonální bloky jsou čtvercové a mají nenulovou diagonálu. Tento tvar je jednoznačně určen až na pořadí bloků; existence vyplývá z triviálního příkladu pro který $k = 1$. Řešíme-li tuto permutovanou soustavu LU rozkladem, pak je potřeba faktorizovat pouze diagonální bloky matice. Při řešení trojúhelníkové soustavy zpětným chodem mimodiagonální bloky postupně upravují pravé strany jednotlivých soustav s diagonálními bloky. Při těchto výpočtech nevzniká zaplnění mimodiagonálních bloků. Pro názornost mějme soustavu (3.1), kde $z = Q^T x$ a $c = Pb$.

$$\begin{pmatrix} A_{11} & & \dots & A_{1k} \\ & \ddots & & \vdots \\ & & A_{(k-1)(k-1)} & A_{(k-1)k} \\ & & & A_{kk} \end{pmatrix} \begin{pmatrix} z_1 \\ \vdots \\ z_{k-1} \\ z_k \end{pmatrix} = \begin{pmatrix} c_1 \\ \vdots \\ c_{k-1} \\ c_k \end{pmatrix} \tag{3.1}$$

Zde vzniknou rovnice $A_{kk}z_k = c_k$, $A_{(k-1)(k-1)}z_{k-1} + A_{(k-1)k}z_k = c_{k-1}$ atd. Z první rovnice vyřešíme pomocí LU rozkladu diagonálního bloku A_{kk} vektor z_k. Po aplikaci LU rozkladu na $A_{(k-1)(k-1)}$ pak z upravené druhé rovnice $A_{(k-1)(k-1)}z_{k-1} = c_{k-1} - A_{(k-1)k}z_k$ získáme z_{k-1}. Tímto způsobem se postupuje i v dalších krocích.

3.3 Pořadí podle nejnižšího stupně

Vezměme opět jako v multifrontální metodě poslední příkaz k-tého kroku algoritmu *LU_right*, který zde zapíšeme ve tvaru

$$A_{22} = A_{22} - l_{21}u_{12},$$

což odpovídá vztahu (2.3). V tomto kroku od A_{22} odečítáme 1-dimensionální vnější součin $l_{21}u_{12}$, kde A_{22} značí k krát upravenou matici A algoritmem *LU_right*. Množinu vrcholů grafu matice $l_{21}u_{12}$ označme jako element $el(k)$ vrcholu k.

Definice 3.2 (stupeň)**.** *Stupeň d_i vrcholu i je velikost množiny*

$$\left(A_i \ \cup \ \bigcup_{j \in E_i} el^{(i)}(j) \right) / \{i\},$$

kde E_i je množina vrcholů v jejichž elementech se vyskytuje i, $el^{(i)}(j) = \{l \in el(j) | l \geqq i\}$ a A_i je nenulový vzor i-tého řádku A.

Algoritmy založené na minimalizaci stupně pracují se seznamem stupňů jednotlivých vrcholů, který se postupně aktualizuje. Cílem je vybrat jako pivotní vrchol ten, jež má minimální stupeň. Aktualizace seznamu stupňů je poměrně drahá. Existují různé techniky, jako jsou například cmd, amd, colamd.

3.4 Nested dissection

Definice 3.3 (množina rozdělovacích vrcholů)**.** *Nechť G je neorientovaný graf A. Množina rozdělovacích vrcholů $R = \{r_1, ..., r_k\}$ je množina vrcholů, které rozdělí graf G na dva přibližně stejně velké podgrafy, které již nejsou spojeny žádnými hranami a neobsahují vrcholy z R.*

Metoda nested dissection hledá množinu rozdělovacích vrcholů. Příslušné podgrafy jsou pak dále rozdělovány další množinou rozdělovacích vrcholů, dokud není vhodné použít poměrně drahou metodu nejnižšího stupně. Po nalezení k množin rozdělovacích vrcholů bude mít matice tvar

$$\begin{pmatrix} A_{11} & & & A_{1(k+1)} \\ & \ddots & & \vdots \\ & & A_{kk} & A_{k(k+1)} \\ A_{1(k+1)}^T & \dots & A_{k(k+1)}^T & A_{(k+1)(k+1)} \end{pmatrix}.$$

Zde pro jednoduchost předpokládáme, že matice A je symetrická. Submatice A_{ii}, pro $i = 1, ..., k$, jsou matice jednotlivých podgrafů. Submatice

$$\begin{pmatrix} A_{1(k+1)}^T & \dots & A_{k(k+1)}^T & A_{(k+1)(k+1)} \end{pmatrix} a \begin{pmatrix} A_{1(k+1)} \\ \vdots \\ A_{k(k+1)} \\ A_{(k+1)(k+1)} \end{pmatrix}$$

jsou řádky a sloupce příslušné k jednotlivým vrcholům z $k-1$ rozdělovacích množin.

Na následujícím akademickém příkladě názorně ukážeme, proč je tento tvar výhodný. Mějme matici

$$A = \begin{pmatrix} \bullet & * & * & * & * & * \\ * & \bullet & & & & \\ * & & \bullet & & & \\ * & & & \bullet & & \\ * & & & & \bullet & \\ * & & & & & \bullet \end{pmatrix}$$

a matice

$$B = \begin{pmatrix} \bullet & & & & & * \\ & \bullet & & & & * \\ & & \bullet & & & * \\ & & & \bullet & & * \\ & & & & \bullet & * \\ * & * & * & * & * & \bullet \end{pmatrix}$$

přičemž platí $B = PAP^T$, a dané symboly značí totéž jako dříve. Z matice A se po jednom kroku Gaussovy eliminace stane obecně hustá matice. Ale u matice B nedochází k žádnému zaplnění pokud vybereme jako pivotní prvky, prvky diagonální.

Kapitola 4

Přehled softwaru

Na závěr této práce popíšeme několik populárních softwarových balíčků pro řešení řídkých lineárních soustav pomocí LU rozkladu. Pro přehlednost a orientaci vypíšeme vlastnosti jednotlivých balíčků heslovitě.

GPLU[7](Gilbert Peierls LU, Gilbert a Peierls, 1988) - první LU rozklad, který využívá řídké pravé strany a prohledávání do hloubky, patří do třídy left-looking, má v sobě obsaženo řešení řídké trojúhelníkové soustavy, GPLU je obsaženo v Matlabu, kde se vyvolává se příkazem $[L, U, P] = lu(A)$.

MA48[6](pouze označení subroutine, Duff a Reid, 1993) - patří do oxfordské sbírky balíčků HSL, do třídy left-looking, řeší také matice s nesymetrickým nenulovým vzorem a obdélníkové matice, používá úplnou pivotaci a permutaci na blokově trojúhelníkový tvar, navíc také počítá chybu a provádí iterační zpřesnění.

UMFPACK[4](Unsymmetric Frontal Package, Davis a Duff, 1997) - první software využívající multifrontální metodu pro obecně nesymetrické matice, minimalizuje zaplnění pomocí metody nejnižšího stupně, UMFPACK je nejrychlejší přímý řešič v Matlabu, je volán ve tvaru $[L, U, P, Q] = lu(A)$.

SuperLU[5](Supernodal LU, Demmel, Eisenstat, Gilbert a Li, 1999) - patří do třídy left-looking založené na supernode, pro minimalizaci zaplnění používá metodu nejnižšího stupně, kromě vlastních metod minimálního zaplnění je možno připojit tyto metody z vnější, umožňuje paralelní implementaci, rozkládá také obdélníkové matice.

balíček	minimalizace zaplnění	metoda
GPLU	žádná	left-looking
MA48	nejnižší stupeň, blokově trojúhelníková	left-looking
MUMPS	nejnižší stupeň, nested dissection	multifrontal
PARDISO	nejnižší stupeň, nested dissection	left/right supernodal
SuperLU	nejnižší stupeň	left-looking supernodal
UMFPACK	nejnižší stupeň	multifrontal
WSMP	nejnižší stupeň, nested diss., blokově trojúhel.	multifrontal

Tabulka 4.1: Balíčky a jejich vlastnosti.

MUMPS[1](Multifrontal Massively Parallel Solver, Amestoy, Duff, Guermouche, Koster, L'Excellent, 2000) - volně dostupný balíček, založený na multifrontální metodě, provádí paralelní přímý a zpětný chod, sám si při rozkladu rozdělí soustavu za účelem paralelizace, nebo ji může z vnější rozdělit uživatel, pro minimální zaplnění používá metodu nejnižšího stupně a nested dissection.

PARDISO[10](Parallel Direct Solvers, Schenk, Gärtner a Fichtner, 2000) - založen na strategii left-looking a na shared memory multiprocessors, umožňuje kombinaci iteračních metod s přímou metodou pro řešení lineárních systémů, dále využívá supernodální implementaci, paralelní chod a iterační zpřesnění

WSMP[8](Watson Sparse Matrix Package, Gupta, 2002) - založen na multifrontální metodě, pro permutace používá metody nejnižšího stupně, nested dissection, blokově trojúhelníkový tvar, uživatel si může vybrat, zda-li výpočty budou probíhat sekvenčně či paralelně.

Klíčové vlastnosti jednotlivých balíčků jsou shrnuty v Tabulce 4.1.

Literatura

[1] P. R. Amestey, I. S. Duff a J.-Y. L'Excellent: *Multifrontal parallel distributed symmetric and unsymmetric solvers*, Comput. Methods Appl. Mech. Engrg., 2000, strany 501–520.

[2] E. Cuthill a J. McKee: *Reducing the bandwidth of sparse symmetric matrices*, Brandon Press, Princeton, 1969, strany 157–172.

[3] T. A. Davis: *Direct Methods for Sparse Linear Systems*, SIAM Series on the Fundamentals of Algorithms, Philadelphia, 2006.¨

[4] T. A. Davis a I. S. Duff: *An unsymmetric-pattern multifrontal method for sparse LU factorization*, SIAM J. Matrix Anal. Appl., 1997, strany 140–158.

[5] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li a J. W. H. Liu: *A supernodal approach to sparse partial pivoting*, SIAM J. Matrix Anal. Appl., 1999, strany 720–755.

[6] I. S. Duff a J. K. Reid: *The design of MA48: A code for the direct solution of sparse unsymmetric linear systems of equations*, ACM Trans. Math. Software, 1996, strany 187–226.

[7] J. R. Gilbert a T. Peierls: *Sparse partial pivoting in time proportional to arithmetic operations*, SIAM J. Sci. Statist. Comput, 1988, strany 862–874.

[8] A. Gupta: *Highly scalable parallel algorithms for sparse matrix factorization*, IEEE Trans. Parallel Distrib. Systems, 1997, strany 502–520.

[9] J. W. H. Lui: *A compact row storage scheme for Cholesky factors using elimination tree*, ACM Trans. Math. Software, 1986, strany 127–148.

[10] P. Schenk, K. Gärtner a W. Fichtner: *Efficient sparse LU factorization with left-looking strategy on shared memory multiprocessors*, BIT, 2000, strany 158–176.

[11] David S. Watkins: *Fundamentals of Matrix Computations*, John Wiley & Sons, Inc., New York, 2002.

Příloha

Uložení řídké matice

K uložení matice o m řádcích a n sloupcích potřebujeme místo v paměti o velikosti $m \times n$; je zde $m \times n$ obecně nenulových prvků. Ale pokud ukládáme řídkou matici, můžeme využít toho, že počet nulových prvku je podstatně vyšší než počet nenulových. V našich aplikacích pouze nenulové prvky nesou informaci. Zde popíšeme tři způsoby uložení řídkých matic: souřadnicový zápis, komprimovaný sloupcový řídký formát a komprimovaný řádkový řídký formát.

V souřadnicovém zápise ukládáme k-tý nenulový prvek jako trojici $[x_k, i_k, j_k]$, kde x_k značí hodnotu prvku, i_k řádkovou souřadnici, j_k sloupcovou souřadnici. Tedy každému nenulovému prvku vyhrazujeme 3 místa paměti. Z tohoto plyne, že uložení matice souřadnicovým zápisem zabere $3 \times nnz$ místa paměti, kde nnz značí počet nenulových prvků matice. Toto číslo je v případě řídké matice podstatně nižší než m×n.

Uložíme-li matici v komprimovaném sloupcovém řídkém formátu (v angličtině se používá termín compressed column sparse format, a proto jej označíme CCS), dostaneme dokonce lepší výsledek. Budeme matici procházet po sloupcích. K ukládání použijeme tři pole. Narazíme-li v průchodu maticí na k-tý nenulový prvek, uložíme jeho hodnotu do prvního pole označeného x, konkrétně do x_k. Jeho řádkovou souřadnici zaznamenáme do druhého pole označeného i, tedy do i_k. Hodnoty třetího pole c určíme následujícím způsobem: do l-té pozice pole c je zapsáno číslo pořadí, v průchodu maticí, prvního prvku l-tého sloupce. Pole x a i mají délku nnz a pole c je délky $n+1$. Sečteme-li délky těchto polí, dostaneme velikost paměti, která je potřebná pro uložení formátu CCS, tedy $2 \times nnz + n + 1$ místa paměti.

Popis uložení komprimovaným řádkový řídkým formátem (compressed

row sparse format, značeno CRS) se liší pouze v přístupu k matici, je po řádcích. Do k-té pozice pole x se zapíše hodnota k-tého načteného nenulového prvku. Do k-té pozice pole j je zapsána sloupcová souřadnice téhož prvku. A konečně do l-té pozice třetího pole r je uloženo číslo pořadí prvního prvku l-tého řádku. Tento zápis vyžaduje $2 \times nnz + m + 1$ místa paměti.

Předpokládejme, že máme souřadnicový zápis matice s přístupem po sloupcích (ten je také používán v Matlabu). Ukážeme, jak jej převést do formátu CCS, tedy jak vytvořit pole x, i a c. Do pole x vložíme všechny hodnoty x_k, kde x_k je z trojice $[x_k, i_k, j_k]$ souřadnicového zápisu, pro $k = 1, ..., nnz$. Do pole i vložíme všechny řádkové souřadnice i_k. Vkládáme ve stejném pořadí. Jak vytvořit pole c ukážeme na l-té pozici pole c. Číslo na pozici c_l je rovno počtu indexu k, pro které je j_k rovno l, k tomuto navíc přičteme číslo na předchozí pozici pole, tedy c_{l-1}. Pozice c_1 je inicializována jako 1.

Při experimentech pro uložení matice používáme pozměněný CCS formát, neukládáme hlavní diagonálu. Neukládáme jí, protože náklady na uložení v Matlabu jsou nepřirozeně velké ve srovnání s náklady provedených operací. Jelikož předpokládáme jednotkovou hlavní diagonálu L, tak se toto vypuštění v experimentech nijak negativně neodrazí.

Subroutiny

```
function [x]=troj_s(L,b)
      n=length(b);
      [dia,jsl,ira]=ccs(L-speye(n));
      tic
      x=full(b);
      for j=1:n
            if x(j)  = 0
                  for i=jsl(j):jsl(j+1)-1
                        x(ira(i))=x(ira(i))-x(j)*dia(i);
                  end
            end
      end
      toc
```

```
function [x]=troj_sX(L,b)
      [dia,jsl,ira] = ccs(L-speye(length(b)));
      b = sparse(b);
      tic
      x=b;
      [X]=reach(jsl,ira,b);
      for j=1:length(X)
            temp = X(j);
            for i=jsl(temp):jsl(temp+1)-1
                  x(ira(i))=x(ira(i))-dia(i)*x(temp);
            end
      end
      toc
```

```
function [X]=reach(jslS,iraS,b)
    X=zeros(1,0);
    [b_poz]=find(b);
    for i=1:length(b_poz)
        pom = b_poz(i)-X;
        if nnz(pom)==length(X)
            [X]=dfs(b_poz(i),X,jslS,iraS);
        end
    end
    [X]=X([length(X):-1:1])';
```

```
function [X]=dfs(j,X,jslS,iraS)
    for i=jslS(j):jslS(j+1)-1
        prom = iraS(i);
        if prom  = j
            pom = prom-X;
            if nnz(pom)==length(X)
                [X]=dfs(prom,X,jslS,iraS);
            end
        end
    end
    X(length(X)+1) = j;
```

```
function [x]=troj_sXE(L,b)
    [dia,jsl,ira]=ccs(L-speye(length(b)));
    E = etree(L+L');
    b = sparse(b);
    tic;
    x=b;
    [X]=reachE(E,b);
    for j=1:length(X)
        temp = X(j);
        for i=jsl(temp):jsl(temp+1)-1
            x(ira(i))=x(ira(i))-dia(i)*x(temp);
        end
    end
    toc
```

```
function [X]=reachE(E,b)
    X=zeros(1,0);
    [b_poz]=find(b);
    for i=1:length(b_poz)
        pom = b_poz(i)-X;
        if nnz(pom)==length(X)
            [X]=dfsE(b_poz(i),X,E);
        end
    end
    [X]=X([length(X):-1:1])';
```

```
function [X]=dfsE(j,X,E)
    prom = E(j);
    if prom  = 0
        pom = prom-X;
        if nnz(pom)==length(X)
            [X]=dfsE(E(j),X,E);
        end
    end
    X(length(X)+1) = j;



function [dia,jsl,ira]=ccs(A)
    As=sparse(A);
    [r,s,p]=find(As);
    [s,I] = sort(s);
    p = p(I);
    r = r(I);
    nnz = length(r);
    n = r(nnz);
    ira = zeros(1,nnz); dia = ira;
    jsl = ones(1,n);jsl(1) = 1;co = 1;
    for i=2:nnz
        if s(i)  = s(i-1)
            co = co + 1;
            jsl(co) = i;
            [ira(jsl(co-1):jsl(co)-1),I2] = sort(r(I(jsl(co-1):jsl(co)-1)));
            dia(jsl(co-1):jsl(co)-1)=p(jsl(co-1)+I2-1);
        end
    end
    jsl(co+1) = nnz+1;
    if co+1 < n+1
        for i=co+2:n+1
            jsl(i) = jsl(co+1);
        end
    end
    [ira(jsl(co):jsl(co+1)-1),I2] = sort(r(I(jsl(co):jsl(co+1)-1)));
    dia(jsl(co):jsl(co+1)-1)=p(jsl(co)-1+I2);
```