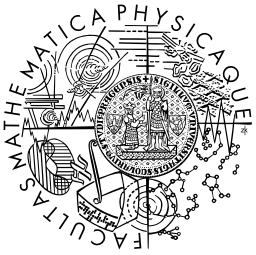


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

DIPLOMOVÁ PRÁCE

Bc. Ilona Riegerová

Řešení problému nejmenších čtverců s maticemi o proměnlivé hustotě nenulových prvků

Katedra numerické matematiky

Vedoucí diplomové práce: prof. Ing. Miroslav Tůma, CSc.
Studijní program: Matematika
Studijní obor: MNVM

Praha 2020

Prohlašuji, že jsem tuto diplomovou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Tato práce nebyla využita k získání jiného nebo stejného titulu.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ....

Podpis autora

Za odborné vedení mé diplomové práce, velkou míru trpělivosti a ochoty, čas a také za cenné a velmi podnětné rady při zpracovávání práce děkuji vedoucímu prof. Tůmovi. Poděkování patří i mé rodině a přátelům, kteří mě podporovali po celou délku studia.

Název práce: Řešení problému nejmenších čtverců s maticemi o proměnlivé hustotě nenulových prvků

Autor: Bc. Ilona Riegerová

Katedra: Katedra numerické matematiky

Vedoucí diplomové práce: prof. Ing. Miroslav Tůma, CSc., Katedra numerické matematiky

Abstrakt: Problém nejmenších čtverců (dále jen LS problém) je aproximační úloha řešení soustav lineárních algebraických rovnic, které jsou z nějakého důvodu zatíženy chybami. Existence a jednoznačnost řešení a metody řešení jsou známé pro různé typy matic, kterými tyto soustavy reprezentujeme. Typicky jsou matice řídké a obrovských dimenzí, ale velmi často dostáváme z praxe i úlohy s maticemi o proměnlivé hustotě nenulových prvků. Těmi se myslí řídké matice s jedním nebo více hustými řádky. Zde rozebíráme metody řešení tohoto LS problému. Obvykle jsou založeny na rozdělení úlohy na hustou a řídkou část, které řeší odděleně. Tak pro řídkou část může přestat platit předpoklad plné sloupcové hodnosti, který je potřebný pro většinu metod. Proto se zde speciálně zabýváme postupy, které tento problém řeší.

Klíčová slova: lineární problém nejmenších čtverců, iterační metody, předpodmínění, rozsáhlé soustavy lineárních algebraických rovnic

Title: Least-squares problems with sparse-dense matrices

Author: Bc. Ilona Riegerová

Department: Department of Numerical Mathematics

Supervisor: prof. Ing. Miroslav Tůma, CSc., Department of Numerical Mathematics

Abstract: The Least-Squares problem (LS problem) is an approximation method for solving a system of linear algebraic equations which are burdened with errors for many reasons. The existence and uniqueness of solutions and LS methods for those solutions are available for different types of matrices that represent these systems. Matrices are typically huge and sparse, however, many practical applications generate sparse-dense matrices - known as sparse matricies with one or more dense rows. We focus on LS methods for this type of LS problem. These are usually based on splitting matrix to sparse and dense part and dealing with both of them separately. Therefore basic assumption of full column rank of the sparse part which is needed for most LS methods doesn't have to hold. We specifically address the procedures that solve this problem with non-regularity.

Keywords: linear least-squares problems, iterative methods, preconditioning, large sparse linear equations

Obsah

Úvod **2**

1 Úvod do problému nejmenších čtverců **3**
1.1 Existence a jednoznačnost LS řešení 4
1.2 Standardní metody řešení LS problému 4
1.2.1 Metody založené na Gaussově eliminaci 5
1.2.2 Metoda založená na QR rozkladu 7
1.2.3 Metoda založená na Choleského rozkladu 9
1.2.4 Metoda založená na SVD rozkladu 10
1.3 Metody řešení řídkého LS problému 11
1.3.1 Řídké přímé metody 11
1.3.2 Řídké iterační metody 16

2 Kombinovaný řídký-hustý LS problém **22**
2.1 Aktualizace řešení LS problému 24
2.2 Metoda kombinace předpodmínění řídké a husté části 25
2.3 Metoda Schurova doplňku 26
2.4 Stretching 28
2.4.1 Řádkový stretching 28
2.4.2 Řídký stretching 30

3 Problém nulových sloupců matice A_s **32**
3.1 Metody řešení LS problému s A_s singulární 32
3.1.1 Metoda Dulmage-Mendelsohnova rozkladu 33
3.1.2 Varianta řídké QR faktorizace 33
3.2 Metody řešení LS problému s nulovými sloupci 34
3.2.1 Blokově předpodmíněná metoda s částečným stretchingem 34
3.2.2 Regularizace 35
3.2.3 Kombinace částečných řešení 36

4 Numerické experimenty **38**
4.1 Regularizace 40
4.2 Částečný stretching 42
4.3 Srovnání metod řešení LS problému s nulovými sloupci 46

Závěr **51**

Seznam použité literatury **54**

Seznam obrázků **57**

Seznam tabulek **57**

Přílohy - Kódy pro Matlab **58**

Úvod

V praxi se často setkáváme s potřebou řešit soustavu lineárních algebraických rovnic, která je, např. důsledkem nepřesností v měření, zatížená chybami. Tuto úlohu nazýváme problém nejmenších čtverců (též LS problém, z anglického termínu *Least Squares Problem*) a reprezentujeme ji maticově ve tvaru

$$\min_x \|Ax - b\|_2, \; kde \quad m, n \in \mathbb{N}, m \geq n, \; A \in \mathbb{R}^{m \times n}, \; b \in \mathbb{R}^m, \; x \in \mathbb{R}^n.$$

Díky vyspělejším moderním technikám dokážeme pracovat s obrovským množstvím dat, a proto jsou stále častější úlohy s rozsáhlou maticí A. Pro zaručení efektivity výpočtu řešení musíme zohledňovat strukturu matice. Typicky se setkáváme s řídkými maticemi. Protože ale tato klasifikace není striktně daná nebo jí vyhovuje pouze část soustavy, jsou běžné i matice o proměnlivé hustotě nenulových prvků. Těmi se myslí řídké matice s jedním nebo více hustými řádky. Úlohu s takovými maticemi nazýváme kombinovaný řídký-hustý LS problém a zapisujeme ji jako

$$\min_x \left\| \begin{pmatrix} A_s \\ A_d \end{pmatrix} x - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2, \; kde \quad A = \begin{pmatrix} A_s \\ A_d \end{pmatrix}, \; A_s \in \mathbb{R}^{m_s \times n}, \; A_d \in \mathbb{R}^{m_d \times n},$$
$$b = \begin{pmatrix} b_s \\ b_d \end{pmatrix}, \; b_s \in \mathbb{R}^{m_s}, \; b_d \in \mathbb{R}^{m_d},$$
$$m = m_s + m_d, \; m_s \geq n, \; m_d \geq 1.$$

Tato práce se zabývá řešením kombinovaného řídko-hustého LS problému. Nejdřív však čtenáře seznámí s existencí a jednoznačností řešení LS problému a s přímými metodami LS řešení s malou a hustou maticí. Důraz je kladen spíše na numerické vlastnosti a aplikaci metod než na detailní odvození. Následně text popisuje modifikace přímých a iterační metody pro LS problém s velkou a řídkou maticí. Ukazuje se, že optimální je vhodná kombinace přímé a iterační metody, typicky pak iterační metoda předpodmíněná nějakou neúplnou faktorizací.

V kapitole 2 jsou popsány tři speciální metody řešení kombinovaného řídko-hustého LS problému, tj. metoda Aktualizace řešení, metoda kombinace předpodmínění řídké a husté části a metoda Schurova doplňku. Navíc je zde rozvedena známá technika stretchingu (z anglického termínu *stretching*), která je velmi používaným nástrojem pro zacházení s nežádoucími hustými řádky. Spočívá v přeuspořádání nenulových prvků husté části A_d tak, aby došlo ke zřídnutí a zvětšení celé matice.

Speciální pozornost je pak věnována případu, kdy matice řídké části A_s nesplňuje předpoklad plné sloupcové hodnosti kvůli přítomnosti nulových sloupců. Ty způsobí potíže v průběhu každé z doposud zmíněných metod, a proto je potřeba buď do metod zahrnout nějaký další ošetřovací krok nebo aplikovat úplně jiný postup. Takové modifikace jsou shrnuty v kapitole 3, kde jsou představeny blokově předpodmíněná metoda s částečným stretchingem, metoda Schurova doplňku s regularizací a metoda Kombinace částečných řešení. Numerické experimenty práce se věnují právě těmto metodám a jejich závislosti na úloze, volbě parametrů a nulových sloupcích.

1. Úvod do problému nejmenších čtverců

Jednou ze základních úloh numerické lineární algebry je řešení soustavy lineárních algebraických rovnic. Takovou soustavu lze maticově reprezentovat ve tvaru $Ax = b$, kde bez újmy na obecnosti buďte $m, n \in \mathbb{N}$, $m \geq n$ a $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$, $x \in \mathbb{R}^n$. Je-li matice A čtvercová a regulární, tedy s plnou sloupcovou hodností, existuje řešení této soustavy, které je navíc jednoznačné.

Problém nejmenších čtverců (dále jako LS problém, z anglického termínu *Least Squares Problem*) je matematicko-statistická metoda zabývající se aproximační úlohou $Ax \approx b$. Ve statistice je tato úloha, představující aproximaci daných hodnot přímkou, nazývána *lineární regrese*, ale s její obdobou či modifikací je možné se setkat i v jiných aplikacích, kde má své specifické názvy a postupy řešení. LS problém obecně vyvstává z praxe z potřeby adaptace matematického modelu odpovídajícího matici A na naměřené hodnoty vektoru pravé strany b, které zpravidla obsahují chyby způsobené nepřesnostmi v měření nebo výpočty v konečné aritmetice. Běžně se vliv těchto chyb snižuje pomocí většího počtu pozorování než je neznámých proměnných. Potom tedy platí $m > n$ a úloha se nazývá přeurčená. Vektor x, který minimalizuje normu rezidua r definovaného předpisem $r = b - Ax$, se nazývá řešení úlohy ve smyslu nejmenších čtverců.

Definice 1 (Problém nejmenších čtverců)**.** *Buďte $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$. Problémem nejmenších čtverců je nazývána úloha určení vektoru $x \in \mathbb{R}^n$ takového, že platí*

$$\min_{x,f} \|f\| \qquad \textit{za podmínky} \qquad Ax = b + f.$$

Často se setkáme s formulací LS problému jako úlohy určení vektoru $x \in \mathbb{R}^n$ splňujícího v Eukleidovské normě

$$\min_x \|Ax - b\|_2 . \tag{1.1}$$

Mnoho metod, které LS problém řeší, vychází z ekvivalentní charakterizace pomocí soustavy normálních rovnic. Toto shrnuje následující věta, jejíž důkaz je k nalezení v [10].

Věta 1. *Nechť $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$. Potom x je řešením problému nejmenších čtverců právě tehdy, když je řešením soustavy normálních rovnic,*

$$A^T Ax = A^T b. \tag{1.2}$$

Poznámka. LS problém lze ekvivalentně definovat v oboru komplexních čísel, tím se ale práce zabývat nebude. Je vhodné také zmínit, že existuje i obecnější varianta LS problému, tzv. problém úplných nejmenších čtverců (dále jako TLS problém, z anglického termínu *Total Least Squares Problem*), který je v mnoha ohledech velmi odlišný od LS problému a jeho podrobný popis je nad rámec této práce. TLS problém uvažuje chyby nejen ve vektoru pravé strany b, ale i v samotném modelu, tj. v matici soustavy A. Ve statistice se taková úloha nazývá *ortogonální regrese*. Více se jím zabývá např. [24] nebo [25].

V následujících podkapitolách jsou shrnuty teoretické vlastnosti maticového LS problému (1.1). Především tedy existence a jednoznačnost řešení a metody řešení. Nejdříve bude diskutován případ, kdy je matice soustavy A relativně malá a je možné o ní uvažovat jako o matici husté. Později bude uvedena teorie uvažující matici A, která je velká a řídká. Takto se práce postupně dostane k maticím s proměnlivou hustotou nenulových prvků v řádcích, což je hlavním tématem. Nebude-li výslovně uvedeno jinak, předpokládejme od nynějška, že matice A má plnou sloupcovou hodnost, tj. nechť $rank(A) = n$.

1.1 Existence a jednoznačnost LS řešení

Existence LS řešení (dále též jako řešení LS problému) se dá ukázat pomocí rozkladu pravé strany $b = Ax + r$ do dvou ortogonálních komponent

$$Ax \in \mathcal{R}(A), \qquad r \in \mathcal{N}(A^T)$$

a využitím faktu, že nejlepší aproximace vektoru pravé strany b je jednoznačně dána ortogonální projekcí $b|_{\mathcal{R}(A)}$ do prostoru oboru hodnot $\mathcal{R}(A)$. Přesné odvození lze nalézt v knize [10], odkud jsou převzaty i následující dvě věty.

Věta 2 (Existence LS řešení)**.** *Nechť $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$. Vektor x je řešením problému nejmenších čtverců právě tehdy, když*

$$Ax = b|_{\mathcal{R}(A)}, \qquad \|b - Ax\| = \|b|_{\mathcal{N}(A^T)}\|.$$

Pokud matice A nemá plnou sloupcovou hodnost, potom LS řešení není jednoznačné. V takovém případě se za řešení uvažuje to minimální v normě, o kterém lze dokázat, že už existuje pouze jedno.

Věta 3 (Jednoznačnost LS řešení)**.** *Nechť $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$. Potom existuje právě jedno řešení x problému nejmenších čtverců minimální v normě, které je dáno vztahy*

$$Ax = b|_{\mathcal{R}(A)} \qquad a \qquad x \in \mathcal{R}(A^T).$$

1.2 Standardní metody řešení LS problému

Metody řešení LS problému se obvykle dělí do dvou základních skupin, přímé a iterační metody. Každá z nich obsahuje v praxi více i méně používané řešiče, které se aplikují na různé typy LS problému. Volba mezi jednotlivými skupinami a jejich metodami je primárně založena na vlastnostech matice A. Nelze však ale zanedbat ani vliv počítačové architektury, která může např. nabízet možnost paralelního výpočtu. Dále je volba ovlivněna původem a strukturou matice, nebo třeba i její další aplikací a zpracováním. Zde budou stručně popsány některé vybrané příklady. Důraz je kladen spíše na chování, numerické vlastnosti a rozdíly

mezi metodami než na jejich odvození a detailní popis.

Přímé metody jsou založeny na rozkladu matice A a na jeho aplikaci, obvykle ve formě substitucí. Obecně platí, že tyto metody jsou vhodnějším nástrojem pro malé a husté úlohy. Proto jsou i běžné v blokových variantách klasických algoritmů. K hlavním příkladům patří metody založené na QR rozkladu, Choleského rozkladu, nebo Gaussově eliminaci, např. Peters-Wilkinsonova metoda, výpočet pomocí pseudoinverze z LU rozkladu nebo pomocí rozšířené matice soustavy. V obecném případě, kdy nemusí být splněn předpoklad plné sloupcové hodnosti, se často používá SVD rozklad. Všechny tento text později probere. Nutno zmínit, že existují tzv. řídké varianty některých těchto metod, které se používají jako řešiče LS problému s velkou a řídkou maticí nebo k předpodmínění. Toto bude více rozebráno v podkapitole 1.3.

Iterační metody, které aproximují přesné LS řešení díky postupnému vylepšování aktuálně spočteného řešení, jsou běžnějším nástrojem pro řešení LS problému. A to nejen proto, že mohou rychleji poskytnout alespoň nějaký výsledek, ale i kvůli samotné práci s maticí. Ta nemusí být dána explicitně svou strukturou, ale například jen svým působením na vektor, tedy jako operátor. Do této skupiny se řadí níže rozebírané metody CGLS, LSQR a LSMR, ale i další.

Rozdělení přímých a iteračních metod podle jejich aplikací není striktní. Ačkoliv iterační metoda může zpřesnit výsledek poskytnutý přímou metodou, je velmi časté, že se přímá metoda použije k zefektivnění té iterační. Toto vede k všeobecně rozšířené myšlence, že v jistém smyslu nejlepším LS řešičem je kombinace přímé a iterační metody.

1.2.1 Metody založené na Gaussově eliminaci

V první řadě se nabízí řešit LS problém pomocí Gaussovy eliminace, neboť právě tak se řeší klasické soustavy algebraických rovnic. Předpokládáme zde, že čtenář je obeznámen s obecnými vztahy propojující Gaussovu eliminaci s LU rozkladem či jeho speciálními variantami. Zde budou více rozebrány tři varianty tohoto postupu, tj. Peters-Wilkinsonova metoda, výpočet pomocí pseudoinverze z LU rozkladu a transformace na rozšířenou matici soustavy.

Definice 2 (LU rozklad)**.** *Nechť $A \in \mathbb{R}^{n\times n}$ je regulární matice. Rozklad tvaru*

$$A = LU,$$

kde L je dolní trojúhelníková matice s jednotkovou diagonálou a U je horní trojúhelníková matice, je nazýván LU rozkladem matice A.

1.2.1.1 Peters-Wilkinsonova metoda

Peters-Wilkinsonova metoda je založena na LU rozkladu prováděného Gaussovou eliminací s částečnou pivotací, tj.

$$\Pi_1 A \Pi_2 = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} = LU = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix} U,$$

kde $L_1 \in \mathbb{R}^{n\times n}$ je jednotková dolní trojúhelníková, $U \in \mathbb{R}^{n\times n}$ je nesingulární (regulární) horní trojúhelníková a matice Π_1, Π_2 reprezentují částečné pivotace.

Částečná pivotace řádků není nutná, ale může se tak získat dobře podmíněný faktor L. To bývá pro stabilitu výpočtu a řešení podstatnější než výpočetní náročnost samotné pivotace. LS problém (1.1) pak přejde na tvar

$$\min_y \left\| Ly - \hat{b} \right\|_2, \qquad U\hat{x} = y,$$

kde $\hat{x} = \Pi_2^T x$, $\hat{b} = \Pi_1 b$, viz [4]. Řešení tohoto problému lze potom již snadno a bez význačné ztráty na přesnosti obdržet ze soustavy normálních rovnic

$$L^T L y = L^T \hat{b}.$$

Lze ukázat, že tento postup je stabilnější než řešení soustavy normálních rovnic (1.2) právě díky vlastnostem LU rozkladu. Navíc pokud nebude nutné, nebudou zde diskutovány konkrétní perturbační a stabilitní odhady, týkající se těchto problémů a příslušných algoritmů.

1.2.1.2 Pseudoinverze

LS řešení lze explicitně vyjádřit pomocí tzv. pseudoinverze. Tento pojem je definován více způsoby, ale nejběžněji se používá v následující smyslu.

Definice 3 (Pseudoinverze)**.** *Pseudoinverze obecné obdélníkové matice $Y \in \mathbb{R}^{m\times n}$ se definuje jako matice $X \in \mathbb{R}^{n\times m}$ taková, že*

$$XYX = X, \qquad YXY = Y,$$

kde matice YX, XY jsou ortogonální.

Pro regulární matici Y, zjevně platí $X = Y^{-1}$. Zde proto bude pseudoinverzí myšlena matice X taková, že LS řešení je dáno jako Xb, což pro regulární matici soustavy odpovídá vektoru $A^{-1}b$.

Buď stále $A \in \mathbb{R}^{n\times n}$ čtvercová regulární matice. Pak ji lze obecně rozložit na tvar $A = BC$, kde $B, C \in \mathbb{R}^{n\times n}$ a $rank(B) = rank(C) = n$. Z definice LS řešení minimalizuje reziduum, a tedy platí ortogonalita

$$A^T(b - Ax) = 0, \tag{1.3}$$

ze které přímo vychází charakterizace LS problému pomocí soustavy normálních rovnic (1.2). Pak následujícími úpravami plyne z (1.3) explicitní vyjádření LS řešení x.

$$\begin{aligned} A^T Ax = A^T b \quad &\Leftrightarrow \quad C^T B^T BCx = C^T B^T b, \\ &\Leftrightarrow \quad Cx = (B^T B)^{-1} B^T b. \end{aligned}$$

Odtud plyne, že LS řešení lze zapsat ve tvaru $x = C^T(CC^T)^{-1}(B^T B)^{-1}B^T b$. Toto odvození je převzato z [27], kde je i dokázáno, že vektor x je LS řešení minimální v normě a je jednoznačné. Matice pseudoinverze je potom tvaru $X = C^T(CC^T)^{-1}(B^T B)^{-1}B^T$.

Článek [27] navíc ukazuje, že matice X je nezávislá na volbě matic B, C, a proto lze vhodně položit $B = L$, $C = U$, kde L,U jsou faktory z LU rozkladu. Při této volbě jsou běžně nutné řádkové pivotace, díky kterým je možné zaručit,

aby matice L byla dobře podmíněná. Možná špatná podmíněnost matice A se tedy přenáší pouze do matice U. Díky předpokladu plné sloupcové hodnosti se ale dají výrazy $U^T(UU^T)^{-1}$ a $(L^TL)^{-1}L^T$ zredukovat. Explicitní vyjádření LS řešení tak přejde na jednodušší tvar, tj.

$$x = U^T(L^TLUU^T)^{-1}L^Tb \quad \Leftrightarrow \quad x = U^{-1}L^{-1}b.$$

Vliv špatné podmíněnosti úlohy se díky této redukci nepropaguje. LU rozklad matice prováděný Gaussovou eliminací s částečnou pivotací je podmíněně zpětně stabilní a vyžaduje $\frac{2}{3}n^3$ operací. Je tedy obecně stabilnější, ale dražší než např. Choleského faktorizace též používaná pro řešení LS problému. Ta je více diskutována v sekci 1.2.3.

1.2.1.3 Rozšířená matice soustavy

LS problém (1.1) lze charakterizovat pomocí rozšířené matice systému, viz [4], jako

$$\begin{pmatrix} I & A \\ A^T & 0 \end{pmatrix} \begin{pmatrix} y \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}. \tag{1.4}$$

Tato ekvivalence je snadno vidět po jednom kroku blokové eliminace, která (1.4) převádí na tvar

$$\begin{pmatrix} I & A \\ 0 & -A^TA \end{pmatrix} \begin{pmatrix} y \\ x \end{pmatrix} = \begin{pmatrix} b \\ -A^Tb \end{pmatrix}.$$

Soustava (1.4) je čtvercová, symetrická a regulární. Pokud $A \neq 0$, pak je soustava navíc indefinitní. Řešení soustavy je závislé na vhodné pivotaci, ale i při použití tzv. Bunch-Kaufmann pivotace, která se běžně aplikuje, bývá dosažení výsledku značně náročné, časově i výpočetně. Soustava je obtížně řešitelná, i pokud je matice A faktorizována QR rozkladem, viz sekce 1.2.2. LS řešení jde potom explicitně vyjádřit pomocí

$$\begin{pmatrix} d_1 \\ d_2 \end{pmatrix} = Q^Tb, \qquad Rx = d_1, \qquad r = Q \begin{pmatrix} 0 \\ d_2 \end{pmatrix},$$

kde $r = y = b - Ax$. Tento postup může být atraktivní, pokud je obecný indefinitní řešič k dispozici.

1.2.2 Metoda založená na QR rozkladu

LS problém je ortogonálně invariantní úloha díky Eukleidovské normě, a je proto možné ho vhodně reformulovat za použití ortogonální matice Q z QR rozkladu.

Definice 4 (QR rozklad)**.** *Buď $A \in \mathbb{R}^{m\times n}$ obecná obdélníková matice. Rozklad tvaru*

$$A = QR,$$

kde $Q \in \mathbb{R}^{m\times m}$ je matice s ortonormálními sloupci, tj. $Q^TQ = I$, a $R \in \mathbb{R}^{m\times n}$ má všechny prvky pod hlavní diagonálou nulové, tj. $R = [r_{i,j}], r_{i,j} = 0$ pro $i > j$, se nazývá QR rozklad matice A.

QR rozklad existuje pro libovolnou matici a předpoklad plné sloupcové hodnosti zajišťuje jeho jednoznačnost v případě zafixování znamének diagonálních prvků matice R. V případě, že $rank(A) < n$ nebo se A takové matici blíží, jsou nasnadě jisté řádkové permutace aplikované na matici A, pomocí kterých lze QR rozklad také určit jednoznačně.

Pro tento odstavec buď LS problém (1.1) uvažován v matematicky ekvivalentním tvaru

$$\min_x \left\| Q^T(Ax - b) \right\|_2 ,$$

kde $Q^T \in \mathbb{R}^{m\times m}$ je ortogonální matice. Následující úpravy transformují (1.1) na úlohu

$$\min_x \|b - Ax\| = \min_x \|b - QRx\| = \min_x \left\| Q^T b - Rx \right\| = \left\| (Q_m^{\perp})^T b \right\|, \qquad (1.5)$$

kde $Q = [q_1, \ldots, q_m, q_{m+1}, \ldots, q_n] = [Q_m, Q_m^{\perp}]$, viz [10]. Platí, že násobení ortogonální maticí lze provádět pomocí zpětně stabilního algoritmu a navíc vzniklá úloha (1.5) by měla být snadněji řešitelná.

Tuto faktorizaci lze aplikovat vždy, má-li matice A dobře určenou numerickou sloupcovou hodnost, tj. je-li splněno

$$\frac{\sigma_1(A)}{\sigma_n(A)} \ll \frac{1}{\varepsilon},$$

kde $\sigma_1(A)$, resp. $\sigma_n(A)$, značí největší, resp. nejmenší, singulární číslo matice A a ε strojovou přesnost, viz [10].

Numerické vlastnosti výpočtu QR rozkladu jsou klíčové pro celkové řešení LS problému. Z definice rozkladu platí, že faktor R má stejná singulární čísla jako matice A, čehož se dá dobře využít při numerické analýze nebo dalších výpočtech. Důkaz existence tohoto rozkladu, k nalezení v knize [4], zase ukazuje, že výpočet je prováděn pomocí ortogonálních transformací. Aplikací těchto transformací na matici A lze získat numericky stabilním způsobem matici v předem zvoleném tvaru. Zde tedy horní trojúhelníkovou matici R. Jedním z nejranějších zástupců těchto ortogonálních transformací jsou Householderovy reflexe, které spočívají v zrcadlení vektoru podle nadroviny. Jejich odvození lze najít například v [20] nebo [4]. Ve stejnou dobu byly také představeny Givensovy rotace, viz [15], které z geometrického pohledu představují otočení vektoru o nějaký úhel. Výpočet QR rozkladu pomocí některé z těchto transformací je založen na postupné aplikaci reflexe nebo rotace na matici A, tedy

$$P_n \cdots P_2 P_1 A = \begin{pmatrix} R \\ 0 \end{pmatrix}, \qquad Q = (P_n \cdots P_2 P_1)^T = P_1 P_2 \cdots P_n,$$

kde každá z matic P_i, $i \in \{1, \ldots, n\}$, nuluje určitou část příslušného vektoru.

Výpočet pomocí Householderových reflexí vyžaduje pro obdélníkovou matici $n^2(m - \frac{1}{3}n)$ operací, výpočet pomocí Givensových rotací vyžaduje dvakrát tolik. Oba přístupy jsou tedy srovnatelné s metodou založenou na Choleského rozkladu, rozebranou v sekci 1.2.3. Pokud však platí, že $m \gg n$, pak je QR rozklad dvojnásobně výpočetně dražší.

Posledním zmíněným způsobem výpočtu QR rozkladu je Gram-Schmidtův ortogonalizační proces vyžadující mn^2 operací. Hlavní nevýhodou tohoto procesu může být značná ztráta ortogonality v aritmetice konečné přesnosti, tj.

$\left\|Q^TQ - I\right\|_F \cong \kappa^2\varepsilon$. U Householderových reflexí i Givensových rotací se ztráta ortogonality drží na úrovni strojové přesnosti ε. Není však pravda, že je Gram-Schmidtův proces kvůli tomuto nepoužitelný. Platí ale, že klasický algoritmus není vhodný pro řešení LS problému, ačkoliv je lépe paralelizovatelný. Jeho modifikovaná rekurzivní varianta (dále jako MGS, z anglického termínu *Modified Gram-Schmidt algorithm*) aplikovaná na matici (A, b) je numericky zpětně stabilní algoritmus, a tedy nepodléhá tolik vlivu zaokrouhlovacích chyb. Pro MGS lze ukázat, že $\left\|Q^TQ - I\right\|_F \cong \kappa\varepsilon$, a dokonce bylo experimentálně ukázáno, že dává o něco přesnější výsledky než jiné ortogonální metody. Spekuluje se, že to může být způsobeno necitlivostí MGS na řádkové permutace často potřebné při výpočtech rozkladu.

Ačkoliv existují novější implementace těchto algoritmů vhodně využívající počítačovou architekturu a paralelismus, tedy i blokové a jiné hybridní varianty, stále obecně platí, že QR rozklad patří ke stabilním, ale výpočetně i časově náročnějším řešičům.

1.2.3 Metoda založená na Choleského rozkladu

Charakterizace LS problému (1.1) pomocí soustavy normálních rovnic z Věty 1 dobře poslouží k odvození další skupiny metod, které jsou vhodné pro soustavy se symetrickou pozitivně definitní maticí. Charakterizace (1.2) je snadnými úpravami převeditelná na ekvivalentní tvar (1.5) a řešení lze vyjádřit explicitně rovností

$$x = (A^TA)^{-1}A^Tb.$$

Velkou výhodou tohoto postupu může být zmenšení celé úlohy, pokud $m \gg n$ a n je malé. V opačném případě je možné násobit transponovanou maticí A^T zprava. Pokud je i přesto matice soustavy velká, využívá se blokových algoritmů. Navíc ze symetrie stačí ukládat pouze horní nebo dolní trojúhelník matice A^TA. Tímto se paměťová náročnost výpočtu výrazně snižuje. Matici A^TA a novou pravou stranu A^Tb lze obdržet v $\frac{1}{2}n(n+1)m + mn$ krocích. Časová náročnost je ovlivněna sloupcovou nebo řádkovou reprezentací matice A^TA a vektoru A^Tb. Je nutné mít na paměti architekturu počítače, aby nedocházelo k opakovanému procházení sloupců nebo případnému nahlížení do sekundárního úložiště. Již bylo zmíněno, že matice nemusejí být nutně ukládány po prvcích, ale jako nějaké funkce. V takových případech je procházení o to dražší. V praxi se této teorie využívá pro vývoj iteračních metod, např. CGLS, CGNR nebo CGNE, které matici soustavy A^TA explicitně nekonstruují.

Choleského rozklad je typickým nástrojem pro přímou metodu řešící LS problém ve tvaru (1.2). Jde o rozklad tvaru $A = LL^T$, který je roven LU rozkladu pro symetrické a pozitivně definitní matice. Jeho odvození a důkaz následující věty je k nalezení v [10].

Věta 4 (Choleského rozklad)**.** *Pro každou symetrickou pozitivně definitní matici A existuje jednoznačný rozklad*

$$A = LL^T,$$

kde L je dolní trojúhelníková matice s kladnými prvky na diagonále.

Získat takový rozklad lze bezpodmínečně zpětně stabilním algoritmem bez potřeby pivotací, ovšem pouze má-li matice A plnou sloupcovou hodnost. Choleského rozklad vyžaduje $\frac{1}{2}mn^2 + \frac{1}{6}n^3 + O(mn)$ operací, a řadí se tak k méně časově náročným metodám. Výpočet LS řešení potom probíhá ve dvou krocích, jež v každém z nich se řeší trojúhelníková soustava, konkrétně

$$L^T z = A^T b \qquad a \qquad Lx = z. \tag{1.6}$$

Numerické řešení je značně ovlivněno faktem, že podmíněnost soustavy může růst kvadraticky vůči původní, neboť

$$\kappa(A^T A) = [\kappa(A)]^2,$$

a tedy propagace zaokrouhlovacích chyb může vést k znehodnocení výsledku. Někdy je možné zvolit parametrizaci, a tím se vyhnout špatně podmíněné úloze. Obecně ale tento postup není vhodný pro špatně podmíněné matice a bezpečně ho lze aplikovat pouze, pokud $\kappa^2(A) \ll \frac{1}{\varepsilon}$. Proto není tato metoda zpětně stabilní.

1.2.4 Metoda založená na SVD rozkladu

Ačkoliv je singulární rozklad matice bez diskuze nejmocnějším a nejvíce říkajícím rozkladem, nebude vždy vhodným nástrojem pro řešení LS problému. Jeho hlavní síla spočívá v aplikovatelnosti na matice, které nesplňují zatím tak potřebný předpoklad plné sloupcové hodnosti nebo jsou špatně podmíněné. Obrovskou nevýhodou je však jeho výpočetní náročnost, a tak je příhodným pro řešení LS problému s malou, hustou maticí soutavy. Následující věta, převzatá z [4], popisuje řešení obecného LS problému pomocí SVD rozkladu.

Věta 5. *Buď následující předpis obecný LS problém*

$$\min_{x \in \mathcal{S}} \|x\|_2, \qquad \mathcal{S} = \{x \in \mathbb{R}^n \,|\, \|b - Ax\|_2 = \min\},$$

kde $A \in \mathbb{R}^{m \times n}$ a $rank(A) = r \leq \min(m,n)$. Tato úloha má vždy jednoznačné řešení, které za použití ekonomického SVD rozkladu matice $A = U_r \Sigma_r V_r^T$ má tvar

$$x = V_r \begin{pmatrix} \Sigma_r^{-1} & 0 \\ 0 & 0 \end{pmatrix} U_r^T b = A^\dagger b,$$

kde $A^\dagger$ je Mooreova-Penroseova pseudoinverze.

Stabilní algoritmus pro SVD rozklad vychází z převedení matice A na horní bidiagonální tvar např. Lanczosovým procesem. Efektivnějším je však algoritmus od dvojice Demmel-Kahan, což je v principu spojení tzv. posunutého QR algoritmu (z anglického termínu *Shifted QR algorithm*) a stabilní implementace s nulovým posunutím, díky čemuž jsou singulární čísla bidiagonální matice spočtena velmi přesně a rychle. Dříve se SVD rozklad počítal metodami Jacobiho typu. Ty stále patří k nejpřesnějším, ale pomalým metodám, viz [4]. Obecně je jakýkoliv postup výpočtu SVD rozkladu velmi pracný, a je proto často nahrazován méně výpočetně náročným QR rozkladem se sloupcovou pivotací. V sekci 1.2.2 jsou shrnuty numerické vlastnosti QR rozkladu a lze tvrdit, že se může jednat také o velmi mocný řešič.

1.3 Metody řešení řídkého LS problému

V podkapitole 1.2 byly rozebrány varianty metod, které jsou vhodné pro řešení LS problému s malými hustými maticemi. V současné praxi se ale díky potřebám aplikační praxe i vyspělejším technikám počítá s obrovským množstvím dat, a tak při jejich zpracovávání vznikají velké a z principu nutně i řídké matice. Proto musejí algoritmy zohledňovat strukturu matice. V následujících odstavcích jsou rozebrány modifikace přímých metod a iterační metody vhodnější pro řídké výpočty. Práce s řídkými maticemi obecně vyžaduje využití grafových reprezentací a dalších nástrojů pro efektivní výpočty. Některé z nich jsou v práci rozepsány.

Pojem řídké matice byl už několikrát zmíněn a zatím vždy bez definice, ačkoliv zní dosti intuitivně. Jedná se o takové matice, jejichž velká část prvků je nulových. To je velmi vágní a nematematická formulace. Konkrétněji se vyjádřil J. H. Wilkinson v [37], který řídkou matici definoval jako libovolnou matici s dostatečným počtem nulových prvků, takže se toho vyplatí využít. Zohledněním řídkosti se zvyšuje efektivita výpočtů, neboť se ukládají pouze nenulové prvky a pouze s nimi se také počítá. Odtud pak také pramení snaha vyhnout se zaplnění matice, tj. vzniku nových nenulových prvků.

Stále buď v následujících úvahách v platnosti předpoklad plné sloupcové hodnosti. Pro tuto podkapitolu buď LS problém

$$\min_x \|Ax - b\|_2 , \tag{1.7}$$

kde $A \in \mathbb{R}^{m \times n}$ je velká a řídká, $m \geq n$, nazýván *řídký LS problém*.

Již bylo zmíněno, že pro řídké matice jsou ukládány pouze nenulové prvky. K efektivnímu uložení maticové struktury do paměti počítače se obvykle používají grafové reprezentace. V mnoha algoritmech lze na matici pohlížet nejprve symbolicky (jako na eliminační strom) a analyzovat tak problém dříve než nastane krok numerického rozkladu (dále též jako numerické faktorizace) a samotné řešení úlohy.

Zásadním krokem pro řešení řídkého LS problému je nalezení vhodné permutace takové, aby nedocházelo k zaplnění, resp. jenom k minimálnímu možnému zaplnění matice. Tento krok je součástí tzv. symbolické faktorizace a začíná jím jakékoliv řešení řídkého LS problému nezávisle na zvolené další metodě. V praxi jsou permutace založeny na přibližných metodách (heuristikách), neboť jejich nalezení pro úlohu reprezentovanou jako rozhodovací kombinatorický problém, představuje *NP-úplný problém*. Tedy s největší pravděpodobností neexistuje sofistikovaný robustní přístup, který úlohu vyřeší v polynomiálním čase přesně. Mezi nejznámější algoritmy používané k přeuspořádání sloupců ΠA, resp. symetricky řádků a sloupců matice $A^T A$, patří algoritmus minimálního stupně a variace metody vnořených řezů, více o nich je k nalezení např. v [2] nebo [4].

I v řídkém případě bude platit předpoklad plné sloupcové hodnosti, pokud nebude řečeno jinak.

1.3.1 Řídké přímé metody

Jako první se nabízí řešit řídký LS problém obdobně jako ten hustý, a to přímými metodami. Obecný princip je zachován. Tedy výpočet spočívá ve faktorizaci matice A a následném řešení většinou trojúhelníkových soustav pomocí

substituce, viz podkapitola 1.2. Avšak jsou nutné jisté modifikace standardních metod tak, aby jejich aplikování na řídký LS problém bylo co nejoptimálnější. Tyto řídké variace tří přímých metod, které jsou popsány v podkapitole 1.2, budou shrnuty v následujících sekcích 1.3.1.1, 1.3.1.2 a 1.3.1.3. Zde buď jen stručně shrnuto, že řešení řídkého LS problému pomocí QR rozkladu je úzce spjato s řešením řídké pozitivně definitní soustavy Choleského rozkladem. A varianta řešení LS problému pomocí rozšířené soustavy je založena na řídkém rozkladu indefinitních symetrických matic. Varianty zmíněných metod a jejich srovnání jsou k nalezení v článku [8] nebo [18].

1.3.1.1 Řídká LU faktorizace

Účinnou metodou pro řídký LS problém, obzvláště pak se špatně podmíněnou maticí, je použití LU rozkladu na soustavu normálních rovnic (1.2). Tato metoda, která je modifikací Peters-Wilkinsonovy metody popsané v sekci 1.2.1.1, byla poprvé sepsána v [5] a využívá následujícího dělení faktoru L tvaru

$$L = \begin{pmatrix} L_1 \\ L_2 \end{pmatrix}, \qquad \Pi_1 b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{1.8}$$

pro permutovanou matici soustavy $\Pi_1 A \Pi_2 = LU$.

Po zadefinování dvou pomocných vektorů c, d

$$L_1 c = b_1, \qquad d = b_2 - L_2 c, \tag{1.9}$$

platí rovnosti

$$\Pi_1 (b - Ax) = \begin{pmatrix} 0 \\ d \end{pmatrix} - Lz, \qquad z = U\Pi_2^T x - c.$$

Proto pokud z řeší minimalizační problém

$$\min_z \left\| \begin{pmatrix} 0 \\ d \end{pmatrix} - Lz \right\|_2, \ kde \quad U\Pi_2^T x_1 = c, \quad U\Pi_2^T x_2 = z,$$

potom vektor $x = x_1 + x_2$ je řešením úlohy (1.7). Pokud je norma $\|b - Ax_1\|_2 = \|d\|_2$ dostatečně malá, vektor x_1 může být navíc prohlášen za řešení a výpočet vektorů x_2 a z vůbec nemusí proběhnout. Šetří se tak výpočetní náročnost.

Je vidět, že se tento případ efektivně vyhýbá možnému kvadratickému nárůstu čísla podmíněnosti. Permutační matice Π_1, Π_2 totiž kromě zachovávání řídkosti slouží i k izolování vlivu možné špatné podmíněnosti matic A a U na podmíněnost faktoru L. Tak matice Π_1, Π_2 přispívají k numerické stabilitě metody. Obecně je ale stabilita stále přímo závislá na podmíněnosti matice L. Proto platí, že metoda založená na QR rozkladu bude stejně stabilnější. Na druhou stranu si umí LU faktorizace lépe poradit se zaplněním. Taky proto bude jen pro lehce přeurčené problémy jedna z nejlepších vůbec.

Z Algoritmu 1, převzatého z [18], je vidět, že matice U je potřeba pouze při zpětné substituci. Proto může být uložena v sekundárním úložišti a nezhoršuje tak paměťovou náročnost výpočtu.

Zde stojí za zmínku, že metodu používající rozšířené matice soustavy (1.4) lze také využít pro řídký LS problém. Z numerického pohledu je sice výpočetně náročnější, ale stabilitou si stojí mezi QR rozkladem a Choleského faktorizací. Stejně jako výše je tento postup velmi efektivní, je-li k dispozici příslušný indefinitní řešič s řídkou variantou výše zmíněné Bunch-Kaufmann pivotace.

Algoritmus 1: Řídký LS problém pomocí LU faktorizace.

Data: $A \in R^{n \times n},\ b \in R^n$

Výpočet faktorizace $\Pi_1 A \Pi_2 = LU$, kde jsou permutační matice vhodně voleny tak, aby zachovaly řídký faktor U a L a faktor L byl navíc dobře podmíněn.

Výpočet $L_1 c = b_1$ z rovnosti (1.9), viz značení z (1.8).

Výpočet $U \Pi_2^T x_1 = c$.

Výpočet $d = b_2 - L_2 c$ z rovnosti (1.9), viz značení z (1.8).

Pokud je velikost normy $\|d\|_2$ dostatečně malá, $x = x_1$ a výpočet končí.

Výpočet $L^T L z = L^T \begin{pmatrix} 0 \\ d \end{pmatrix}$, např. pomocí Algoritmu 3.

Výpočet $U \Pi_2^T x_2 = z$.

Výpočet $x = x_1 + x_2$.

1.3.1.2 Řídká QR faktorizace

Z numerických vlastností popsaných v sekci 1.2.2 plyne, že použití QR rozkladu vede na stabilní výpočet. Na druhou stranu je však QR rozklad časově náročný a navíc kvůli užitým ortogonálním transformacím v něm dochází k tak nechtěnému zaplnění. Dále je pro úlohy s velmi různě škálovanými řádky velmi nepřesný. Toto samotné by zdánlivě stačilo k zavržení metody QR rozkladu pro řešení řídkého LS problému (1.7). Je nutné mít ale na paměti, že vzniklé zaplnění je odhadováno a pro některé matice se tento odhad může dramaticky lišit od skutečnosti. Konkrétně tomu tak je pro matice, které nemají silnou Hallovu vlastnost viz [7], kde bude odhad zaplnění značně nadhodnocen. Protože je QR rozklad tak hojně využívaným nástrojem pro hustý LS problém, byly v minulosti navrženy jeho řídké modifikace. Znovu jsou aplikovány permutace v symbolické faktorizaci za účelem zachování řídkosti. To sice nemá vliv na výsledný faktor R, ale bude tím ovlivněna výpočetní náročnost rozkladu. Proto se strategie přeuspořádání volí a aplikují s ohledem na skutečný numerický výpočet a je možné jimi náročnost alespoň částečně snížit. Dalším argumentem pro QR rozklad byla skutečnost, že se hodí i pro úlohy s maticí, která nemá plnou sloupcovou hodnost. Toto přetrvává i pro řídký případ.

V článku [14] je popsána numericky stabilní metoda řešení řídkého LS problému založená na QR rozkladu prováděného Givensovými rotacemi, která se snaží převzít pozitiva z metody Choleského rozkladu v sekci 1.2.3. V zásadě se proto jedná o speciální případ této metody. Řídká QR faktorizace se však potýká s větší výpočetní náročností.

Celá metoda je odvozena z ekvivalence mezi metodou založenou na QR rozkladu a charakterizací LS problému ve tvaru normálních rovnic (1.2).

$$\begin{aligned} A^T A x = A^T b &\Leftrightarrow R^T R x = R^T Q^T b \\ &\Leftrightarrow R^T R x = R^T Q_m^T b \\ &\Leftrightarrow R x = Q_m^T b \end{aligned}$$

Odtud se dokazuje rovnost mezi Choleského faktorem matice $A^T A$ a maticí R z QR rozkladu.

Tedy řídký LS problém je v tomto případě převeden na soustavu normálních rovnic, ale místo Choleského faktorizace a následného řešení soustav (1.6), je aplikována stabilní QR faktorizace na permutovanou matici ΠA. Při řídkých výpočtech bývaly často preferovány Givensovy rotace, neboť nejsou na rozdíl od Householderových reflexí nebo Gram-Schmidtova ortogonalizačního procesu doprovázeny tak velkým zaplněním v průběhu výpočtu. U Householderových reflexí a Gram-Schmidtova ortogonalizačního procesu se může vzniklé zaplnění ke konci výpočtu zase redukovat, ale není možné to nějak zaručit. Navíc pro Givensovy rotace není potřeba mít k dispozici celé sloupce matice A, neboť je A zpracovávána postupně po řádcích. Proto lze v průběhu výpočtu přidávat řádky, čehož se dá využít při tzv. aktualizovacích technikách, které budou rozebrány později. Dále lze každou Givensovu rotaci reprezentovat jedním číslem a je tak paměťová náročnost nižší než u jiných ortogonálních transformací. Algoritmus 2, který schématicky popisuje průběh výpočtu, je detailněji rozebrán v [18].

Algoritmus 2: Řídký LS problém pomocí QR faktorizace.
Data: $A \in R^{n \times n}$, $b \in R^n$
Vyhodnocení struktury matice $A^T A$.
Nalezení permutační matice Π takové, že $\Pi^T A^T A \Pi$ má řídký Choleského faktor L.
Symbolická faktorizace $\Pi^T A^T A \Pi$ a odhad struktury faktoru L.
Numerický výpočet faktoru L a $c = \Pi^T A^T b$ pomocí Givensových rotací.
Výpočet $Ry = c$.
Výpočet $x = \Pi y$.

Jednou z nových efektivnějších variant QR rozkladu pro řídký LS problém je tzv. Multifrontální QR rozklad (z anglického termínu *Multifrontal QR Decomposition*), představen v [9], který se ale potýká s paměťovou náročností. Tento rozklad funguje na principu rozdělení matice na malé husté podmatice a postupném aplikování Householderových reflexí. Tím se může částečně vyhnout zaplnění v průběhu výpočtu a lze jej dobře paralelizovat. V implementaci metody narážíme na koncept eliminačních stromů, které reprezentují celý postup. Detailnější popis metody je v knize [4].

1.3.1.3 Řídká Choleského faktorizace

Než přijde na řadu samotný numerický výpočet Choleského faktorizace, je potřeba mít v řídkém případě k dispozici odhad struktury matice $A^T A$ a volit pak permutační matici Π takovou, že matice $\Pi^T A^T A \Pi$ má řídký Choleského faktor. Jeho struktura je pak prakticky daná a může se tak dopředu alokovat místo v paměti. K tomuto odhadu je nutný tzv. *předpoklad o nevyrušení*, který říká, že součet nebo rozdíl dvou nenulových prvků je též nenulový prvek. Protože se struktury řídkých matic reprezentují grafy, nebývají odhady struktur složité na vyhodnocení. Struktura Choleského faktoru se pak celkem snadno získá aplikací grafového modelu Gaussovy eliminace na graf matice $\Pi^T A^T A \Pi$. Celkový proces faktorizace je tímto zachycen v eliminačních grafech, ve kterých je podchyceno i zaplnění. Zde je záhodno zmínit, že pokud řídká matice A obsahuje

snad jenom jeden hustý řádek, bude nutně matice A^TA hustá a přímá aplikace řídkých metod pak ztrácí na významu. V takovém případě se může hustý řádek vypustit, výpočet se provede pro zbylou řídkou matici a na konci se využije již zmíněných technik aktualizace podobně jako u výpočtů, kdy v průběhu přibývají řádky matice soustavy. Tento problém je ale jako celek komplexnější a komplikovanější, a proto se mu text věnuje později.

Jak již bylo ukázáno, faktor R z QR rozkladu odpovídá Choleskému faktoru L matice A^TA. Ale pokud by se struktura faktoru R měla odhadovat stejně jako struktura L, pak bude značně nadhodnocena. Příkladem buď diagonální matice s prvním hustým řádkem. Pak $R = A$, ale matice A^TA je hustá, a tedy i její faktor L by byl odhadnut hustý. Toto představuje strukturální vyrušení prvků, které chce mít na paměti i přes platnost předpokladu o nevyrušení. Struktura faktoru R bývá z pravidla určena během symbolické fáze faktorizace z Givensových rotací nebo Houselholderových reflexí, ale i ty mohou strukturu nadhodnotit. Bezpečnou predikci struktury R lze zatím získat pouze pro matice se silnou Hallovou vlastností, jak je shrnuto v následující větě, viz [4].

Věta 6. *Buď $A \in \mathbb{R}^{m\times n}$, $m \geq n$, matice se silnou Hallovou vlastností. Potom je možné ze struktury A^TA správně vyvodit strukturu faktoru R kromě numerických vyrušení.*

Protože Choleského faktorizace nevyžaduje pivotace pro udržení stability výpočtu, mohou být voleny tak, aby se zachovala co největší řídkost. Choleského faktorizace tak může na rozdíl od QR rozkladu poskytovat velmi dobré neúplné rozklady, které slouží k předpodmiňování. Navíc pořadí řádků nemá vliv na faktorizaci. To v důsledku znamená, že je lze libovolně permutovat a faktorizovat postupně. Matice proto nemusí být nikdy uložená celá v rychle přístupné paměti.

Obdobně jako v hustém případě dává metoda pomocí soustavy normálních rovnic uspokojivé výsledky s dostatečnou přesností avšak pouze pro dobře podmíněné matice. Pro špatně podmíněné matice je vhodnější využít QR rozkladu nebo metodu vylepšovat iterativním zpřesněním. Dále nelze tuto metodu snadno adaptovat na obecnější úlohy, například pro matice nesplňující předpoklad plné sloupcové hodnosti. Toto představuje hlavní úskalí dalšího směřování tohoto textu. Schématický Algoritmus 3 níže, který popisuje řešení LS problému pomocí Choleského faktorizace, je převzat z [18].

Algoritmus 3: Řídký LS problém pomocí Choleského faktorizace.
Data: $A \in R^{n\times n},\ b \in R^n$
Vyhodnocení struktury matice A^TA.
Nalezení permutační matice Π takové, že $\Pi^TA^TA\Pi$ má řídký Choleského faktor L.
Symbolická faktorizace $\Pi^TA^TA\Pi$ a odhad struktury faktoru L.
Numerický výpočet A^TA a A^Tb.
Numerická faktorizace $\Pi^TA^TA\Pi = L^TL$.
Výpočet $L^Tz = \Pi^TA^Tb$.
Výpočet $Ly = z$.
Výpočet $x = \Pi y$.

1.3.2 Řídké iterační metody

Doposud rozebírané metody počítaly LS řešení v konečném a předem odhadnutelném počtu kroků. Přesnost získaného řešení se tedy dá zlepšit pouze zopakováním celého výpočtu. Tomuto se vyhýbají iterační metody. Ty aproximují přesné řešení postupně, iteračně, a jsou navrženy tak, aby iterace byly co nejméně náročné. Tak je možné získat řešení s předem danou tolerancí v kratším čase než za použití přímé metody. Zjevně jsou vhodné pro výpočty, kdy postačí velmi orientační aproximované řešení. Na druhou stranu, pokud je cílem se k přesnému řešení přiblížit co nejvíce, znovu budou voleny iterační metody. Obecně mohou být levnější a rychlejší než klasické rozklady v přímých metodách, ale nejsou tolik robustní. Tyto metody nebyly zařazeny do podkapitoly 1.2, protože nabývají smyslu a svého potenciálu v plné míře až pro LS prolém s velkou řídkou maticí.

Primárním kritériem pro výběr iterační metody je konvergence. Ideální iterační metoda bude konvergovat vždy a v co nejmenším počtu kroků. To je odvislé od samotné metody, její implementace i předpodmínění. Nejrozšířenější jsou metody pro řešení symetrických pozitivně definitních systémů, které se aplikují na LS problém v ekvivalentním tvaru normálních rovnic (1.2). Jak již bylo zmíněno na začátku podkapitoly 1.2, iterační metody nepotřebují mít k dispozici matici A explicitně, a tedy ani matice A^TA nemusí být explicitně konstruována. Tímto se vyhýbají značné nevýhodě této ekvivalentní formulace.

Iterační metody jsou rozděleny do dvou skupin, které se principiálně liší. Stacionární iterační metody, které bývají často označovány za klasické, štěpí matici A na dvě vhodně zvolené matice, kde z pravidla alespoň jedna je regulární a dobře invertovatelná. Řešená soustava pak splňuje

$$\begin{aligned}(M+N)x &= b\\ x &= M^{-1}(b-Nx) = x + M^{-1}(b-Ax),\end{aligned}$$

odkud je odvozen obecný iterační krok metody

$$x_k = (I - M^{-1}A)x_{k-1} + M^{-1}b.$$

Na soustavu s normální maticí bývají typicky aplikovány stacionární metody tvaru

$$Mx^{(k+1)} = Nx^{(k)} + b \qquad k = 0,1,\cdots .$$

K hlavním zástupcům takových metod patří Gaussova-Seidelova metoda, Jacobiho metoda, metoda SOR (z anglického termínu *Successive Overrelaxation method*) nebo SSOR (z anglického termínu *Symetric Successive Overelaxation method*). Detailnímu rozboru metod se text věnovat nebude, ale jsou k nalezení např. v knize [10].

Významnější a druhou skupinou iteračních metod jsou tzv. projekční iterační metody. Ty konstruují posloupnosti aproximujících řešení $x_k \in x_0 + \mathcal{S}_k$, kde $\mathcal{S}_k \subseteq \mathbb{R}^n$ je k-dimenzionální podprostor nazýván *prostor aproximace* (z anglického termínu *Search space*). Tedy tyto metody hledají řešení na menším ale velmi blízkém podprostoru, kam je úloha projektována tak, aby byla lépe a snadněji řešitelná. Reziduum $r_k \perp \mathcal{C}_k$, kde $\mathcal{C}_k \subseteq \mathbb{R}^n$ je k-dimenzionální podprostor nazýván *prostor podmínek* (z anglického termínu *Constraint space*), a splňuje tzv. *Petrov-Galerkinovu podmínku*. Ta popisuje tradiční volbu předpokladu kolmosti rezidua

a v kombinaci s podmínkou růstu dimenze prostorů $\mathcal{S}_k$ a $\mathcal{C}_k$ dává, že potom projekční metoda nalezne řešení původní úlohy nejpozději v n krocích. Nejpodstatnějšími projekčními metodami jsou Krylovské iterační metody, které využívají Krylovových podprostorů. Rozdíl mezi jednotlivými metodami pak činí princip generování vhodné báze Krylovova podprostoru. To je typicky prováděno pomocí Lanczosova algoritmu pro symetrické nebo Arnoldiho algoritmu pro nesymetrické matice.

Nejznámější Krylovskou metodou je zaručeně metoda sdružených gradientů. Té se věnuje sekce 1.3.2.1. Dále jsou rozebrány metody LSQR a LSMR. Jakákoliv iterační metoda odvozená pro symetrickou pozitivně definitní soustavu rovnic může být aplikována na ekvivalentní charakterizaci LS problému ve tvaru normálních rovnic (1.2). Kvůli možnému většímu zaplnění a citlivosti řešení na perturbace je však vhodnější uvažovat soustavu ve faktorovém tvaru

$$A^T(Ax - b) = 0.$$

Další ekvivalentní formulace je pomocí rozšířené matice soustavy. Tu lze také řešit iteračními metodami, ale pouze těmi, které jsou vhodné pro symetrické indefinitní soustavy. Takových však není mnoho.

1.3.2.1 Metoda sdružených gradientů pro řídký LS problém

Tato sekce shrnuje nejzákladnější poznatky o CGLS (z anglického termínu *Conjugate Gradients for Least Squares*), což je iterační metoda sdružených gradientů pro řešení řídkého LS problému. Stejně jako pro originální metodu sdružených gradientů platí i pro CGLS, že v přesné aritmetice nalezne řešení nejpozději v n krocích. Proto byla dříve metoda sdružených gradientů mylně označována za přímou metodu.

Základní ideou CGLS je minimalizování chybového funkcionálu na určité varietě Krylovova prostoru. Buď $x = A^\dagger b$ přesné řešení dané pseudoinverzí $A^\dagger = (A^TA)^{-1}A^T$ a $r = b - Ax$ buď příslušné reziduum. Potom vektor aproximace $x^{(k)}$ minimalizuje chybový funkcionál

$$E(x^{(k)}) = (x - x^{(k)})^T(A^TA)(x - x^{(k)})$$

na prostoru $x^{(k)} \in x^{(0)} + \mathcal{K}_k(A^TA, A^T(b - Ax^{(0)}))$. Platí, že normy $\|r - r^{(k)}\|$ a $\|x - x^{(k)}\|$ monotónně klesají. A pokud je matice A špatně podmíněná, pak norma $\|A^Tr^{(k)}\|$ může výrazně oscilovat. Matice soustavy musí být nejen regulární, ale navíc symetrická a pozitivně definitní. To je zde zaručeno aplikací metody na soustavu normálních rovnic A^TA.

Konvergence CGLS je přímo závislá na rozložení vlastních čísel matice A a na normách projekcí počátečního rezidua, jak je popsáno v následující větě. Pro důkladnější studium je vhodná kniha [21].

Věta 7 (Konvergence CG a závislost na rozložení vlastních čísel)**.** *Buď $A \in \mathbb{R}^{n\times n}$ symetrická pozitivně definitní matice. Potom metoda sdružených gradientů CG pro soustavu $Ax = b$ konverguje nejpozději v n-té iteraci. Navíc pro A-normu k-té chyby $e^{(k)}$ platí*

$$\|e^{(k)}\|_A = \|r_0\| \min_{p\in\pi_k} \left(\sum_{j=1}^{n} \frac{|\omega_j|^2}{\lambda_j} p(\lambda_j)^2 \right)^{1/2},$$

a tedy odhad relativní A-normy chyby je tvaru

$$\frac{\|e^{(k)}\|_A}{\|e^{(0)}\|_A} \leq \min_{p\in\pi_k} \max_{i=1,\ldots,n} |p(\lambda_i)|.$$

Kvůli stabilitě je CGLS aplikována na ekvivalentní charakterizaci LS problému ve tvaru (1.2). Obecně je metoda paměťově nenáročná, protože v každé iteraci je nutno uchovávat pouze čtyři vektory $x, p \in \mathbb{R}^n$, a $r, q \in \mathbb{R}^m$. Navíc v každé iteraci dochází nanejvýš k násobení matice s vektorem Ap_k a $A^T r_k$. A tedy pokud je toto násobení implementováno efektivně pro řídké matice, jedná se též o výpočetně velmi nenáročný algoritmus. Implementace násobení vektoru a matice je ovlivněna počítačovou architekturou i samotným ukládáním vektorů a matic.

Algoritmus CGLS, který byl prvně odvozen M. R. Hestenesem a E. Stiefelem v [19], se neaplikuje přímo na soustavu normálních rovnic, neboť by to mohlo vést na numericky nestabilní algoritmus. Tato skutečnost je důkladně rozebrána v [6]. Jak je vidět z Algoritmu 4, počet potřebných operací v každé iteraci odpovídá $2nz(A) + 3n + 2m$, kde $nz(A)$ je počet nenulových prvků matice A. Zde je uveden algoritmus PCGLS, který má v sobě navíc zapracováno předpodmínění.

Algoritmus 4: PCGLS

`/* Inicializace. */`
$r^{(0)} = b - Ax^{(0)}$;
$s^{(0)} = p^{(0)} = M^{-T}(A^T r^{(0)})$;
$\gamma_0 = \| s^{(0)} \|^2$;

for $k = 0, 1, 2 \ldots$ **do**
  while $\gamma_k > tol$ **do**
    $t^{(k)} = M^{-1} p^{(k)}$;
    $q^{(k)} = At^{(k)}$;
    $\alpha_k = \gamma_k / \left\| q^{(k)} \right\|^2$;
    $x^{(k+1)} = x^{(k)} + \alpha_k t^{(k)}$;
    $r^{(k+1)} = r^{(k)} - \alpha_k q^{(k)}$;
    $s^{(k+1)} = M^{-T}(A^T r^{(k)})$;
    $\gamma_{k+1} = \left\| s^{(k+1)} \right\|^2$;
    $\beta_k = \gamma_{k+1} / \gamma_k$;
    $p^{(k+1)} = s^{(k+1)} + \beta_k p^{(k)}$;
  end
end

Existují další dvě modifikace této metody, CGNR a CGNE, které hledají aproximované řešení na stejném Krylovově podprostoru, ale s rozdílnou minimalizační podmínkou. CGNR spočítá aproximované řešení s nejmenší možnou normou rezidua, zatímco CGNE spočítá aproximované řešení s co nejmenší možnou normou chyby. Algoritmické zápisy je možné dohledat v knize [29].

1.3.2.2 LSQR

Velmi častou alternativou pro CGLS je metoda LSQR. Ta je matematicky ekvivalentní, protože generuje stejnou posloupnost aproximací řešení x_k, resp. stejnou alespoň v případě přesné aritmetiky. Odtud plyne, že pro LSQR platí obdobné výsledky, závěry analýz chování a odhady o konvergenci jako pro CGLS. LSQR ve svém principu tedy minimalizuje normy $\|r_k\|$. Hlavním rozdílem je však způsob, jakým se generuje posloupnost aproximací x_k. LSQR využívá tzv. Golub-Kahanovy bidiagonalizace, pomocí které souběžně počítá dvě ortogonální báze.

Celý proces bidiagonalizace spočívá v následujících rekurencích. Pro $j = 1, 2, \ldots$

$$\begin{aligned}
\beta_1 u_1 &= b, \\
\alpha_1 v_1 &= A^T u_1, \\
\beta_{j+1} u_{j+1} &= A v_j - \alpha_j u_j, \\
\alpha_{j+1} v_{j+1} &= A^T u_{j+1} - \beta_{j+1} v_j,
\end{aligned}$$

kde $\alpha_{j+1}, \beta_{j+1} \geq 0$ takové, že $\|u_{j+1}\| = \|v_{j+1}\| = 1$. V k-tém kroku buďte $V_k = (v_1, \ldots, v_k)$, $U_k = (u_1, \ldots, u_k)$ a

$$B_k = \begin{pmatrix} \alpha_1 & & & \\ \beta_2 & \alpha_2 & & \\ & \beta_3 & \ddots & \\ & & \ddots & \alpha_k \\ & & & \beta_{k+1} \end{pmatrix} \in \mathbb{R}^{(k+1)\times k}.$$

Sloupce matice V_k tvoří ortonormální bázi Krylovova podprostoru $\mathcal{K}_k(A^T A, s^{(0)})$, kde $s^{(0)} = A^T(b - Ax^{(0)})$, a navíc jsou matice V_k a U_{k+1} ortogonální, maticově zapsáno tedy platí

$$V_k^T V_k = I_k, \qquad U_{k+1}^T U_{k+1} = I_{k+1}.$$

Odtud, pokud $x^{(k)} = V_k y^{(k)}$, pak platí

$$\min_{x_k} \|b - Ax_k\| = \min_{y^{(k)}} \|r_k\| = \min_{y^{(k)}} \left\| B_k y^{(k)} - \beta_1 e_1 \right\|.$$

Řešení $y^{(k)}$ je možné získat např. aplikací Givensových rotací, tedy QR rozkladem. Odtud si získává metoda svůj název i numerickou stabilitu. Platí, že LSQR je numericky spolehlivější než CGLS, pokud je zapotřebí více iterací a matice A je špatně podmíněná. Obdobně jako u CGLS, se v každé iteraci násobí matice s vektorem. Znovu tedy implementace tohoto násobení značně ovlivňuje efektivitu algoritmu.

Algoritmus 5, vyvinut Ch. C. Paigem a M. Saundersem v [26], vyžaduje $3m + 5n$ násobení, tedy o $2n$ více násobení než CGLS. V paměti drží dva m-složkové a tři n-složkové vektory, tedy o jeden vektor více než CGLS. Klíčové slovo *givrot* v Algoritmu 5 reprezentuje funkci vracející Givensovu rotaci. Pro vstupní parametry α, β tedy *givrot* vrátí c, s, ρ takové, že

$$-s\alpha + c\beta = 0 \quad a \quad \rho = \sqrt{(\alpha^2 + \beta^2)}.$$

Celý cyklus algoritmu končí, když je dosaženo konvergence. To je rozhodnuto na základě zvoleného zastavovacího kritéria, mezi které typicky patří zkoumání některé z norem $\|r_k\|$, $\|x_k\|$, $\|s_k\|$, $\|A\|$ nebo $\left\|A^\dagger\right\|$, resp. jejich odhadů. V [26] je ukázáno, že vyhodnocení těchto zastavovacích kritérií není náročné a v porovnání se zastavovacím kritériem u CGLS dokonce levnější.

Algoritmus 5: LSQR

`/* Inicializace. */`
$x^{(0)} = 0$;
$\beta_1 u_1 = b$; $\quad \alpha_1 v_1 = A^T u_1$;
$w_1 = v_1$;
$\hat{\phi}_1 = \beta_1$;
$\hat{\rho}_1 = \alpha_1$;

for $k = 0, 1, 2 \ldots$ **do**

`/* Bidiagonalizace. */`
$\beta_{k+1} u_{k+1} = A v_k - \alpha_k u_k$;
$\alpha_{k+1} v_{k+1} = A^T u_{k+1} - \beta_{k+1} v_k$;

`/* QR rozklad: Konstrukce a aplikace rotace. */`
$[c_k, s_k, \rho_k] = givrot(\hat{\rho}_k, \beta_{k+1})$;
$\theta_k = s_k \alpha_{k+1}$;
$\hat{\rho}_{k+1} = c_k \alpha_{k+1}$;
$\phi_k = c_k \hat{\phi}_k$;
$\hat{\phi}_{k+1} = -s_k \hat{\phi}_k$;

`/* Aktualizace` x_k `a` w_{k+1}`. */`
$x_k = x_{k-1} + (\phi_k / \rho_k) w_k$;
$w_{k+1} = v_{k+1} - (\theta_{k+1} / \rho_k) w_k$;

end

1.3.2.3 LSMR

Poslední zde srovnávanou metodou je LSMR. Zjednodušeno do jedné věty jde o MINRES aplikované na LS problém ve tvaru soustavy normálních rovnic (1.2). Odtud je možné pozorovat jistou paralelu s LSQR, které je ekvivalentní sdruženým gradientům též aplikovaným na (1.2). Narozdíl však od LSQR metoda LSMR ve svém principu minimalizuje normy $\left\|A^T r_k\right\|$. Buď zachováno značení z předchozí sekce 1.3.2.2 a buď $\hat{\beta}_k = \alpha_k \beta_k$. S pomocí Golub-Kahanovy bidiagonalizace lze zapsat LS problém jako

$$\min_{y^{(k)}} \left\|A^T r_k\right\| = \min_{y^{(k)}} \left\| \hat{\beta}_1 e_1 - \begin{pmatrix} B_k^T B_k \\ \hat{\beta}_{k+1} e_k^T \end{pmatrix} y_k \right\|.$$

Tento LS podproblém se vyřeší pomocí dvou po sobě jdoucích QR rozkladů realizovaných Givensovými rotacemi, což je jádrem LSMR metody. Detailnější odvození je k nalezení v článku [13].

Komplexita výpočtu LSMR je srovnatelná s LSQR, ale LSMR lépe konverguje pro určité druhy LS problému. Tím je myšleno, že v případě úloh s omezenou pamětí nebo takových úloh, které musejí být zastaveny dříve, je LSMR vhodnější metodou. Navíc bylo experimentálně dokázáno, že čím blíže je aproximace řešení $x^{(k)}$ přesnému řešení, tím levněji je možné získat odhad zpětné chyby, na kterém je postaveno zastavovací kritérium algoritmu. Následuje Algoritmus 6, který schématicky popisuje průběh metody.

Algoritmus 6: LSMR

`/* Inicializace. */`

$x^{(0)} = 0;$

$\beta_1 u_1 = b; \qquad \alpha_1 v_1 = A^T u_1;$

$\hat{\alpha}_1 = \alpha_1; \qquad \hat{\zeta}_1 = \alpha_1 \beta_1;$

$\rho_0 = 1; \qquad \hat{\rho}_0 = 1;$

$\hat{c}_0 = 1; \qquad \hat{s}_0 = 0;$

$h_1 = v_1; \qquad \hat{h}_0 = 0;$

for $k = 1, 2, 3 \ldots$ **do**

`/* Bidiagonalizace. */`

$\beta_{k+1} u_{k+1} = A v_k - \alpha_k u_k;$

$\alpha_{k+1} v_{k+1} = A^T u_{k+1} - \beta_{k+1} v_k;$

`/* QR rozklad: konstrukce a aplikace rotace` P_k`. */`

$\rho_k = \sqrt{(\hat{\alpha_k}^2 + \beta_{k+1}^2)}; \qquad c_k = \hat{\alpha_k}/\rho_k; \qquad s_k = \beta_{k+1}/\rho_k;$

$\theta_{k+1} = s_k \alpha_{k+1}; \qquad \hat{\alpha}_{k+1} = c_k \alpha_{k+1};$

`/* QR rozklad: konstrukce a aplikace rotace` P_k`. */`

$\hat{\theta}_k = \hat{s}_{k-1} \rho_k; \qquad \hat{\rho}_k = \sqrt{(\hat{c}_{k-1} \rho_k)^2 + \theta_{k+1}^2};$

$\hat{c}_k = \hat{c}_{k-1} \rho_k / \hat{\rho}_k; \qquad \hat{s}_k = \theta_{k+1} / \hat{\rho}_k;$

$\zeta_k = \hat{c}_k \hat{\zeta}_k; \qquad \hat{\zeta}_{k+1} = -\hat{s}_k \hat{\zeta}_k;$

`/* Aktualizace` h, $\hat{h}$ `a` x_k`. */`

$\hat{h}_k = h_k - (\hat{\theta}_k \rho_k / (\rho_{k-1} \hat{\rho}_{k-1})) \hat{h}_k - 1;$

$x_k = x_{k-1} + (\zeta_k / (\rho_k \hat{\rho}_k)) \hat{h}_k;$

$h_{k+1} = v_{k+1} - (\theta_{k+1} / \rho_k) h_k;$

end

2. Kombinovaný řídký-hustý LS problém

V minulé kapitole byl rozebrán LS problém s hustou i řídkou maticí a možnosti jeho řešení. S ohledem na pragmatičnost definice řídkosti je ale obecně složité rozhodnout, kdy má být matice považována za řídkou. Často je tato klasifikace závislá na konkrétním problému. Matice, která je pro jednu úlohu řídká, může být pro jinou hustá. O to více, pokud má matice proměnlivou hustotu nenulových prvků v řádcích. To znamená, že rozhodovací kritérium řídkosti určí část matice jako řídkou a zbytek jako hustou. Pak není zjevné, které metody a řešiče aplikovat. Na tyto specifické matice je proto vhodné speciálně nahlížet jako na řídké matice s jedním nebo více hustými řádky, např. Obrázek 2.1. Pro korektnější zavedení pojmu proměnlivé hustoty nenulových prvků v řádcích poslouží následující definice.

Definice 5. *Matice $A \in \mathbb{R}^{m\times n}$ má proměnlivou hustotu nenulových prvků, jestliže je řídká, ale obsahuje m_d hustých řádků, kde $1 \leq m_d \ll m$.*

Hustým řádkem se v tomto kontextu bude rozumět řádek s výrazně více nenulovými prvky než ostatní řádky, ačkoliv jich pořád může být podstatně méně než je dimenze n. Přirozeně pak lze takovou matici rozdělit do dvou částí, řídké a husté. Stejně tak přirozeně jsou odtud i odvozovány jednotlivé přístupy k řešení příslušného LS problému, které s těmito oddělenými částmi vhodně pracují, aby docílili co možná nejpřesnějšího, robustního a efektivního postupu řešení.

$$\begin{pmatrix}
 & & * & & & * & & \\
 & * & & & & & & \\
 & & * & & * & & & \\
 & & & & & & & * \\
* & & * & & & & & * \\
 & & & * & & & * & \\
 & & & & * & & & \\
* & * & * & * & * & * & * & *
\end{pmatrix}$$

Obrázek 2.1: Schéma řídké matice s jedním hustým řádkem.

Buď v této kapitole *kombinovaným řídko-hustým LS problémem* označován takový LS problém, jehož matice $A \in \mathbb{R}^{m\times n}$ má proměnlivou hustotu nenulových prvků v řádcích. Nechť je proto možné ji a příslušnou pravou stranu $b \in \mathbb{R}^m$ rozdělit do dvou odpovídajících částí

$$A = \begin{pmatrix} A_s \\ A_d \end{pmatrix}, \quad A_s \in \mathbb{R}^{m_s\times n}, \quad A_d \in \mathbb{R}^{m_d\times n};$$
$$b = \begin{pmatrix} b_s \\ b_d \end{pmatrix}, \quad b_s \in \mathbb{R}^{m_s}, \quad b_d \in \mathbb{R}^{m_d},$$

kde $m = m_s + m_d$, $m_s \geq n$, $m_d \geq 1$. Potom příslušný LS problém je úloha minimalizace normy

$$\min_x \left\| \begin{pmatrix} A_s \\ A_d \end{pmatrix} x - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2 . \tag{2.1}$$

Evidentním úskalím řešení LS problému (2.1) je zaplnění v matici $A^T A$, ke kterému nutně dojde kvůli přítomnosti A_d. A to i kdyby se A_d skládala pouze z jednoho hustého řádku. Matice $A^T A$ pak bude nutně hustá. Konkrétně bude celá hustá, pokud je v platnosti předpoklad o nevyrušení. Pokud tomu tak není, mohou se některé prvky vynulovat, ale v matici i přesto dojde k významnému zaplnění. Není tedy vhodné a kolikrát vůbec možné použít přímo řídké varianty metod řešení LS problému. Charakterizace LS problému ve tvaru normálních rovnic (1.2), která se doposud zdála jako jeden z nejlepších přístupů, minimálně pro předpodmínění, tedy prakticky ihned ztrácí na svých výhodách.

K zaplnění může dojít i jinak než kvůli samotnému součinu $A^T A$, a tudíž to není jediným potenciálním problémem této úlohy. Řídké varianty metod pro řešení LS problému nemusí být totiž aplikovatelné. Kvůli zaplnění vznikajícímu v průběhu celého výpočtu by se metody měly zcela vyhýbat úplným faktorizacím v jejich klasických podobách. Dokonce i neúplné varianty faktorizací, kterými se často předpodmiňuje, nemusejí být vhodným nástrojem. Při snaze udržet zaplnění pod stanovenou mírou, je snadné volit neúplnou faktorizaci velmi nepřesnou, a proto neefektivní. Všechny tyto obtíže může způsobit už jenom jeden hustý řádek. Odtud plyne jasná motivace zabývat se metodami, které jsou specifické pro řešení LS problému ve tvaru (2.1).

V zásadě existují tři možnosti, jak si s LS problémem (2.1) poradit. Husté řádky lze ve výpočtu nejdříve úplně ignorovat a následně jimi získané řešení chytře aktualizovat. Tento postup aktualizace (z anglického termínu *updating*) je častý v úlohách, kde přibývají řádky matice A v průběhu výpočtu. Typicky se při aktualizaci aplikují přímé řídké metody, což pro velké matice přináší dříve diskutované nevýhody v efektivitě a paměťové i výpočetní náročnosti. S konkrétními příklady takových metod se lze seznámit například v [14], [5] nebo [3], které využívají QR rozkladu. Nutno poznamenat, že pouze LS řešení je aktualizováno, nikoliv faktorizace použitá ve výpočtu. Ta je zde obvykle přítomná pouze implicitně. Více se aktualizaci věnuje podkapitola 2.1.

Matici soustavy lze vhodně předpodmínit a minimalizovat tak dopad hustých řádků na zaplnění. Použitelných a používaných předpodmínění je více, např. pomocí Schurova doplňku nebo specifickým předpodmíněním sdružených gradientů, jak popisují podkapitoly 2.3 a 2.2. Volba mezi předpodmíněním je primárně ovlivněna velikostí, strukturou a reálnou aplikací modelovaného problému.

Dále se využívá techniky zředění matice A_d a zvětšení tím matice soustavy, tzv. stretchingu (z anglického termínu *stretching*). Ten spočívá v roztrhnutí hustého řádku do dvou či více řádků, které již mohou být považovány za řídké. Takto se uměle LS problém zvětšuje. Rozhodnutí o rozdělení řádku je ale velmi složité. V praktických situacích může být rozhodovací kritérium založeno např. na odhadu efektivity řídkého řešiče. To je ale jen jedna z více možností. V následujícím textu bude tato technika popsána blíže v podkapitole 2.4.

2.1 Aktualizace řešení LS problému

Buď x_s LS řešení podproblému s řídkou maticí A_s

$$\min_x \|A_s x - b_s\|_2 . \tag{2.2}$$

Metoda, která řeší LS problém (2.1) pomocí aktualizace LS řešení x_s, modifikuje tento mezivýsledek pomocí hustých dříve opomenutých řádků. Zjevně je tedy aktualizace aplikovaná pouze na LS řešení, nikoliv na použitou faktorizaci či metodu. Proto je tento postup výhodný pro repetitivní úlohy nebo úlohy, kde přibývají řádky matice postupně během výpočtu.

Jednou z možností, jak získat LS řešení, je použití Woodburyho formule na rozvoj inverze C^{-1} ve tvaru

$$C^{-1} = (C_s + A_d^T A_d)^{-1} = C_s^{-1} - C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1} A_d C_s^{-1},$$

kde $C_s = A_s^T A_s$. Pak lze LS řešení explicitně vyjádřit jako

$$\begin{aligned} x &= x_s + C_s^{-1} A_d^T (I_{m_d} + A_d C_s^{-1} A_d^T)^{-1} (b_d - A_d x_s), \\ x_s &= (A_s A_s^T)^{-1} A_s^T b_s. \end{aligned} \tag{2.3}$$

Tuto soustavu už snadno řeší Peters-Wilkinsonova metoda, viz sekce 1.2.1.1, nebo metody shrnuté v [5]. Každý použitý řešič by se však měl snažit vyhnout explicitnímu vyhodnocování faktorizace matice C_s. Tento celý postup lze zařadit do první kategorie jako příklad techniky aktualizace, ačkoliv má vazbu i na metody využívající speciálního předpodmínění. Inverze matice $C = C_s + A_d^T A_d$, která je rozvinuta pomocí Woodburyho vztahu, může totiž aproximovat inverzi matice předpodmínění. Pak ji lze snadno kombinovat s iteračními metodami jako jsou sdružené gradienty, LSQR, LSMR atd.

Buď zde nyní popsána metoda Aktualizace LS řešení, kterou poprvé popisují články [5] a [14]. Buď $x = x_s + z$ LS řešení původní úlohy. Aktualizace řešení bude označena vektorem z. Pro rezidua odpovídající LS problému (2.1) platí tyto rovnosti

$$r_s(x_s) = b_s - A_s x_s, \qquad r_d(x_s) = b_d - A_d x_s.$$

Aktualizace by ideálně měla tyto dvě rezidua minimalizovat a to ve smyslu

$$\min \|r_s(x_s) - A_s z\|_2^2 + \|r_d(x_s) - A_d z\|_2^2 . \tag{2.4}$$

Díky rovnosti $A_s^T r_s(x_s) = 0$ platí následující ekvivalence s (2.4)

$$\begin{aligned} \min_z \{\|A_s z\|_2^2 + \|A_d z - r_d(x_s)\|_2^2\} &\Leftrightarrow \min_u \left\| \begin{pmatrix} B_d \\ I_n \end{pmatrix} u - \begin{pmatrix} r_d(x_s) \\ 0 \end{pmatrix} \right\|_2^2 \\ &\Leftrightarrow \min \left\| \begin{pmatrix} u \\ v \end{pmatrix} \right\|_2 \; kde \; \begin{pmatrix} B_d & I_{m_2} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = r_d(x_s), \end{aligned}$$

kde $u = R_s z$, $B_d = A_d R_s^{-1}$, $v = r_d(x_s) - B_d u$. Poslední ekvivalentní problém je pak snadno řešitelný QR rozkladem a zpětnou substitucí, viz [14]. Podstatnou nevýhodou této metody je ale potřeba explicitní formulace faktoru R_s, což bývá problematické.

Dělení matice A na hustou a řídkou část má své problémy. Může se stát, že nebude dobře navrženo pro konkrétní matici soustavy a hustý řádek se tak vyhodnotí jako řídký. Toto zjevně představuje komplikaci pro metodu Aktualizace. Dále je možné, že matice A_s je podstatně hůře podmíněná než původní matice A. Pak dochází ke ztrátě přesnosti získaného řešení. Vždy je proto nutno sledovat stabilitu výpočtu, neboť aktualizace nemusí být stabilní pro všechny úlohy. Obzvláště pak pro ty špatně podmíněné.

2.2 Metoda kombinace předpodmínění řídké a husté části

S řešením LS problému (2.1) je možné si poradit aplikováním předpodmíněných sdružených gradientů na řídkou část matice v kombinaci s faktorizací hustých řádků. Tento přístup je představen v článku [32], kde je též experimentálně ukázána vetší efektivita metody v porovnání s předpodmíněnými sdruženými gradienty, které nevěnují hustým řádkům speciální pozornost. Použitá metoda sdružených gradientů ve variantě vhodné pro LS problém, tedy CGLS, je předpodmíněna pomocí neúplné Choleského faktorizace matice $C_s = L_s L_s^T$, zatímco hustá část matice je faktorizována úplným Choleského rozkladem, tj. řádu m_d.

Článek [32] rozebírá případ, kdy faktorizace matice C_s není efektivní, ač už z důvodu malé řídkosti nebo celkové dimenze matice. Pak se řešení LS problému získá jako součet $x = x_s + y$, kde x_s je aproximace řešení pro řídký LS podproblém (2.2) a y je vektor korekce, který lze explicitně vyjádřit. Odtud je vidět úzká vazba na metodu aktualizace z předchozí podkapitoly 2.1.

LS problém (2.1) lze převést aplikací Choleského faktorizace $C_s = L_s L_s^T$ na ekvivalentní tvar

$$\min_{L_s^T y} \left\| \begin{pmatrix} A_s L_s^{-T} \\ A_d L_s^{-T} \end{pmatrix} L_s^T y - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2 \quad \Leftrightarrow \quad \min_z \left\| \begin{pmatrix} B_s \\ B_d \end{pmatrix} z - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2, \tag{2.5}$$

kde přesný tvar korekce y formuluje následující lemma. Důkaz Lemmatu 8 je k nalezení v [32]. V článku je též korekce podrobněji odvozena.

Lemma 8 (Formulace vektoru korekce)**.** *LS řešení problému (2.5) lze vyjádřit jako $z = \tilde{x}_1 + y_1$, kde $\tilde{x}_1$ je libovolná aproximace řešení LS problému (2.5) a*

$$y_1 = B_s^T \rho_s + B_d^T (I_{m_d} + B_d B_d^T)^{-1} (\rho_d - B_d B_s^T \rho_s),$$

kde $B_d = A_d L_s^{-T}$, $\rho_s = b_s - B_s \tilde{x}_1$ a $\rho_d = b_d - B_d \tilde{x}_1$.

Metoda však naráží na možný problém s vyhodnocením korekce y, což má zjevně velký vliv na efektivitu i přesnost řešení. Obecně totiž matice B_s může být hustá, což způsobí vysokou paměťovou náročnost. Získat počáteční aproximaci $\tilde{x}_1$ bývá problematické a v konečném důsledku to může vést na velmi drahé předpodmínění. Proto se v praxi volí korekce, jejíž vyhodnocení je efektivnější, např.

$$y_1 = B_d^T (I_{m_d} + B_d B_d^T)^{-1} \rho_d.$$

Obě varianty vektoru korekce y_1 obsahují společný člen $B_d^T(I_{m_d} + B_d B_d^T)^{-1}$. Jeho vyhodnocení je podstatným krokem celé metody a může mít velký vliv na její chování. V algoritmu této metody, který je použit v experimentech kapitoly 4, je vyhodnocení tohoto členu zapracováno pomocí Choleského faktorizace

$$I_{m_d} + B_d B_d^T = L_d L_d^T.$$

Článek [32] uvádí i možnost využití LQ rozkladu jako alternativu.

Kvůli vypuštění hustých řádků nemusí mít A_s plnou sloupcovou hodnost. Potom je nutná buď úprava matice řešené soustavy regularizací, např. $A_s^T A_s + \alpha I$, nebo aplikace metody vhodné pro práci se singulárními maticemi a maticemi, které nemají plnou sloupcovou hodnost. Více se takovým metodám věnuje text v kapitolách 3 a 4.

Bylo experimentálně ukázáno, že tato metoda předpodmíněných sdružených gradientů konverguje rychleji a je i méně časově náročná než metody, které nevyhodnocují husté řádky speciálně. A dokonce existují úlohy, které nedokáží řešit ostatní metody a tato konverguje v rozumném počtu iterací.

Stojí za zmínku, že popsané předpodmínění může být aplikované i na jinou iterační metodu. Nejen tedy na sdružené gradienty, ale i např. na LSMR nebo LSQR. Konkrétní algoritmus nebude uveden zde, ale je k nalezení s detailním popisem v [32] a v přílohách této práce, neboť byl použit pro experimenty v kapitole 4.

2.3 Metoda Schurova doplňku

Další zde rozebraná metoda pro řešení LS problému (2.1) využívá Schurova doplňku a je odvozena v článku [33]. V prvním kroku se úloha převede na tzv. *redukovaný tvar s rozšířenou maticí soustavy* o velikosti $(n + m_d) \times (n + m_d)$, tj.

$$\begin{pmatrix} -C_s & A_d^T \\ A_d & I_{m_d} \end{pmatrix} \begin{pmatrix} x \\ r_d \end{pmatrix} = \begin{pmatrix} -A_s^T b_s \\ b_d \end{pmatrix}, \tag{2.6}$$

kde reziduum splňuje

$$r = \begin{pmatrix} r_s \\ r_d \end{pmatrix} = \begin{pmatrix} b_s \\ b_d \end{pmatrix} - \begin{pmatrix} A_s \\ A_d \end{pmatrix} x.$$

Ačkoliv se jedná o symetrickou, kvazi-definitní soustavu [36], která je řešitelná řídkými metodami pro indefinitní systémy, je výhodnější aplikovat metodu Schurova doplňku. Ta totiž bude mít lepší numerické vlastnosti a lépe si poradí se stále velkou maticí. Během výpočtu není potřeba tolik místa v paměti jako u indefinitních metod, ale hlavně metoda Schurova doplňku bere ohled na blokovou strukturu soustavy. Takto je metoda ospravedlněna a zároveň jsou shrnuty její hlavní pozitiva.

Buď pro účely tohoto odstavce LS problém (2.1) charakterizován ve tvaru normálních rovnic

$$A^T A x = A^T b \quad \Leftrightarrow \quad (A_s^T A_s + A_d^T A_d)x = (A_s^T + A_d^T)b.$$

Potom platí následující ekvivalence

$$\begin{aligned}(C_s + A_d^T A_d)x = Cx = A^T b = A_s^T b_s + A_d^T b_d = c \\ 0 = A_d x - A_d x \\ \Leftrightarrow \quad \begin{pmatrix} C_s & A_d^T \\ A_d & -I \end{pmatrix} \begin{pmatrix} x \\ A_d x \end{pmatrix} = \begin{pmatrix} c \\ 0 \end{pmatrix}.\end{aligned}$$

Pak lze matici soustavy rozepsat jako

$$\begin{pmatrix} C_s & A_d^T \\ A_d & -I \end{pmatrix} = \begin{pmatrix} L_s & \\ B_d & L_d \end{pmatrix} \begin{pmatrix} I & \\ & -I \end{pmatrix} \begin{pmatrix} L_s^T & B_d^T \\ & L_d^T \end{pmatrix}, \tag{2.7}$$

kde matice B_d je taková, že platí rovnosti

$$L_s B_d^T = A_d^T, \qquad I + B_d B_d^T = L_d L_d^T.$$

Odtud lze jednoduše demonstrovat souvislost redukovaného tvaru LS problému (2.6) s Woodburyho vztahem a explicitním předpisem pro LS řešení (2.3), viz [35]. V metodě Schurova doplňku je však tento rozklad aplikován na matici soustavy (2.6). Pomocí Choleského faktorizace $C_s = L_s L_s^T$ a matice Schurova doplňku $S_d \in \mathbb{R}^{m_d \times m_d}$, která je řídká díky řídkosti faktoru L_s, tedy platí

$$K = \begin{pmatrix} -C_s & A_d^T \\ A_d & I_{m_d} \end{pmatrix} = \begin{pmatrix} L_s & \\ B_d & I_{m_d} \end{pmatrix} \begin{pmatrix} -I_n & \\ & S_d \end{pmatrix} \begin{pmatrix} L_s^T & B_d^T \\ & I_{m_d} \end{pmatrix},$$

kde

$$L_s B_d^T = -A_d^T, \qquad S_d = I_{m_d} + B_d B_d^T.$$

Protože Choleského faktor L_s je dolní trojúhelníková matice, lze z první rovnice získat snadnou substitucí matici B_d^T, která je řídká. Následně už jen zbývá vyřešit dvě soustavy

$$\begin{pmatrix} -L_s & \\ -B_d & S_d \end{pmatrix} \begin{pmatrix} y_s \\ y_d \end{pmatrix} = \begin{pmatrix} -A_s^T b_s \\ b_d \end{pmatrix}, \qquad \begin{pmatrix} L_s^T & B_d^T \\ & I_{m_d} \end{pmatrix} \begin{pmatrix} x \\ r_d \end{pmatrix} = \begin{pmatrix} y_s \\ y_d \end{pmatrix}.$$

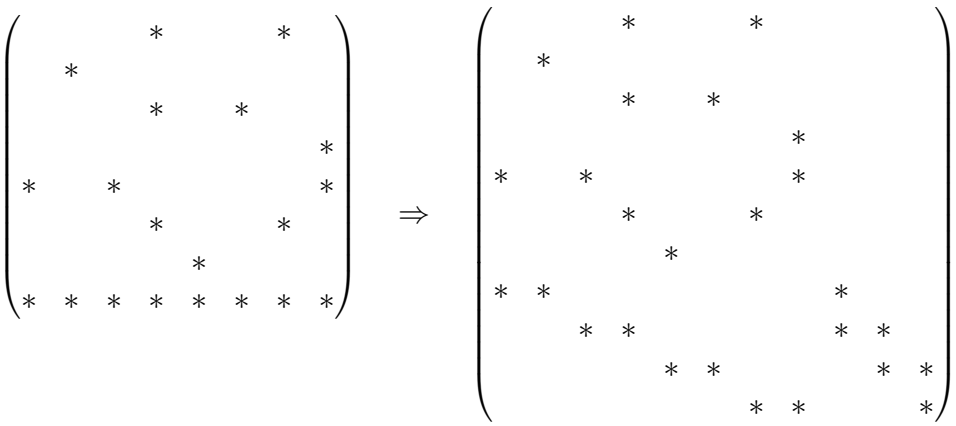
Poslední rovnice dává $r_d = y_d$, čímž se soustava ještě redukuje. Schurův doplněk S_d je nutno buď znovu rozložit pomocí Choleského faktorizace nebo je možné ho vyhodnotit implicitně. Obecně jsou tyto dvě soustavy trojúhelníkové, a proto snadno a hlavně levně řešitelné.

Stabilita Choleského rozkladu, diskutovaná v sekci 1.2.3, zaručuje, že se jedná o efektivní metodu nevyžadující pivotaci. Navíc je velmi snadná na implementaci. Stejně ale jako u metody PCGLS v podkapitole 2.2, je nutné hlídat zaplnění, které spolu s velikostí úlohy způsobuje této metodě největší potíže. Proto bývá paměťová náročnost přední nevýhodou metody.

Kvůli Choleského rozkladům, na kterých je metoda založena, je nutný předpoklad plné sloupcové hodnosti matice A_s. I když má A plnou sloupcovou hodnost, není nijak zaručeno, že tomu tak bude i pro $A_s^T A_s$. Nulové sloupce totiž mohou vzniknout při rozdělení matice A na hustou a řídkou část. Takovou situaci lze řešit např. odstraněním nulových sloupců matice A_s, aplikací regularizačního parametru nebo pomocí Věty 10. Těmto postupům se blíže věnuje kapitola 3.

2.4 Stretching

J. F. Grcar navrhl ve svém článku [16] maticovou metodu nazývanou stretching. Sám stretching se popisuje jako technika, která odstraní husté řádky, které tolik vadí řídkým metodám a efektivitě výpočtu. Odstraněním se zde ale nemyslí celkové opomenutí. Jde o jisté přeuspořádání nenulových prvků tak, aby došlo ke zředění a zvětšení matice, viz Obrázek 2.2. Přitom se z LS problému s takto rozšířenou maticí snadno získá řešení původního problému. Matice soustavy by potom měla být regulární a lépe rozložitelná. Případně se stretching používá na sloupce. To je ale v jiných aplikacích, kde sloupce znamenají řádky a soustava normálních rovnic je tvaru AA^T. Ideální stretching by vůbec neměl zvýšit číslo podmíněnosti nebo změnit spektrální vlastnosti problému A^TA, protože pak jeho aplikací není narušena stabilita výpočtu. V praxi se toho však docílit nedá. Proto se za dobrý stretching považuje ten, který mění tyto vlastnosti úlohy co možná nejméně. Grcar ukazuje přednosti stretchingu na soustavách se šípovou maticí, kde je LS řešení získáno pomocí LU rozkladu a transformace na trojúhelníkovou matici. Stretching se tak ukazuje jako vhodný nástroj pro zefektivnění řešení LS problému (2.1).



Obrázek 2.2: Příklad stretchingu jednoho hustého řádku.

Pro stretching je kritické rozhodnutí, do kolika řádků bude hustá část A_d rozdělena. Čím více vznikne nových řádků, tím řidší bude matice soustavy, což je velmi ceněné. Ale také to může výrazně ovlivnit úlohu jako takovou, zvýšit její číslo podmíněnosti a nezaručuje to ani řídkost Choleského faktoru, a tedy ani možnost využití řídkých přímých řešičů. Článek [34] navrhuje kombinatorický postup, který rozhodne, do kolika částí a jakým způsobem budou husté řádky rozděleny. Rozhoduje tak na základě vazby mezi řídkou částí A_s a rozdělovanými hustými řádky, viz sekce 2.4.2.

Následují dva konkrétní příklady stretchingu. Obecný řádkový stretching, který je detailně rozebraný v článku [1] i se svojí speciální binární variantou, a řídký stretching.

2.4.1 Řádkový stretching

Binární řádkový stretching je nejjednodušší varianta této techniky. Ta rozdělí jeden hustý řádek do dvou řádků vyhodnocených již jako řídké. Bez újmy

na obecnosti lze předpokládat, že hustý řádek matice $A \in \mathbb{R}^{m\times n}$ je právě ten poslední m-tý. Rozdělení posledního řádku soustavy bude tedy tvaru

$$a_m^T x = a_{m1}^T x_1 + a_{m2}^T x_2 = b_m - r_m,$$

kde b_m představuje poslední prvek pravé strany b a r_m poslední prvek rezidua $r = b - Ax$. Nechť je k LS problému (2.1) přidána rovnice

$$\sqrt{2}s + a_{m1}^T x_1 - a_{m2}^T x_2 = 0.$$

Aplikací Givensovy rotace tvaru

$$G = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix}$$

na poslední dva řádky rozšířeného LS problému jsou získány finální dva řádky stretchingu a LS problém přejde na tvar

$$\begin{pmatrix} — & A_s & — \\ 1 & \sqrt{2}a_{m1}^T & \\ \text{-1} & & \sqrt{2}a_{m2}^T \end{pmatrix} \begin{pmatrix} x_s \\ s \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_s + r_s \\ \frac{1}{\sqrt{2}}(b_m + r_m) \\ \frac{1}{\sqrt{2}}(b_m + r_m) \end{pmatrix}. \tag{2.8}$$

Protože Givensovy rotace jsou ortogonální transformace a LS problém je ortogonálně invariantní úloha, platí, že množiny řešení LS problému (2.1) a nově získaného rozšířeného LS problému (2.8) jsou totožné ve svých původních komponentách. Spojovací pomocný prvek s a příslušná konstanta $\sqrt{2}$ jsou vhodně volené tak, aby aplikovaná Givensova rotace dala dobrý výsledek ve smyslu nulového posledního prvku rezidua příslušného LS problému (2.8). Z popsaného principu stretchingu je zjevné, že celý proces lze aplikovat rekurzivně a tak hustý řádek dělit do více, dokud není dosažena žádoucí řídkost.

Zobecněním binárního řádkového stretchingu lze docílit rozdělení hustého řádku rovnou do více řídkých bez nutné rekurze. Buď proto pro účely tohoto odstavce hustá část A_d tvořena pouze jedním posledním řádkem matice A označeným $a_m^T = d^T$. Obecný řádkový stretching rozdělí d^T do k různých řádků $D^T \in \mathbb{R}^{k\times n}$ obsahujících vždy část hustého řádku d^T. Tak LS problém (2.1) přejde na tvar

$$\min_x \left\| \hat{A}z - \hat{b} \right\|_2, \quad \hat{A} = \begin{pmatrix} A_s & 0 \\ D^T & S \end{pmatrix}, \quad z = \begin{pmatrix} z \\ s \end{pmatrix}, \quad \hat{b} = \begin{pmatrix} b_s \\ \frac{b_d}{\sqrt{k}} e \end{pmatrix}, \tag{2.9}$$

kde $s \in \mathbb{R}^{k-1}$, jednotkový vektor $e \in \mathbb{R}^k$ a matice

$$S = \begin{pmatrix} 1 & & & \\ -1 & 1 & & \\ & -1 & \ddots & \\ & & \ddots & 1 \\ & & & -1 \end{pmatrix} \in \mathbb{R}^{k\times(k-1)}.$$

Pro ilustraci dobře poslouží Obrázek 2.2, který odpovídá strukturou matici $\hat{A}$ z (2.9). Matice vazby S je regulární a splňuje rovnost $S^T e = 0$. Protože pro pseudoinverzi $S^\dagger$ platí

$$S^T(I - S(S^T S)^{-1} S^T)x = 0,$$

viz [10] a [1], lze explicitně vyjádřit ortogonální projekci do nulového prostoru S^T. Tak se získá předpis pro prvních k rovnic soustavy (2.9) jako součet ortogonálních komponent prostorů $\mathcal{R}(S)$ a $\mathcal{N}(S^T)$

$$[Ss + (I - \frac{1}{k}ee^T)D^T x] + \frac{1}{k}ee^T D^T x = \frac{1}{\sqrt{k}}b_d e. \tag{2.10}$$

Norma rezidua je potom minimalizována pro $s = S^\dagger D^T x$.

Numerická stabilita tohoto postupu je podrobně odvozena v článku [1] a shrnuta do věty, která je v [34] doplněna. Následuje její korektní znění bez důkazu, který lze najít v citovaných článcích. Věta je odvozena pro řádkový stretching aplikovaný na první řádek matice namísto posledního. Tato řádková permutace je ale bez újmy na obecnosti.

Věta 9 (Odhad čísla podmíněnosti rozšířené matice)**.** *Horní odhad čísla podmíněnosti rozšířené matice*

$$\tilde{A} = \begin{pmatrix} \gamma S & D^T \\ 0 & A_s \end{pmatrix}$$

s parametrem $\gamma = \frac{1}{2}\sqrt{pk}\,\|A_d\|_2$*, kde* p *značí počet hustých řádků, je dán předpisem*

$$\kappa^2(\tilde{A}) \leq \kappa^2(A)k\left(1 + \frac{2pk\,\|A_d\|_2^2}{\|A\|_2^2}\right)\left(k + 1 + \frac{\sigma_{min}^2}{\|A_d\|_2^2}\right),$$

kde σ_{min}^2 *je nejmenší singulární číslo matice* A*.*

2.4.2 Řídký stretching

Řádkový stretching transformuje LS problém (2.1) na celkově řídký, avšak nijak nezohledňuje strukturu Choleského faktoru L matice A. Teto faktor může být totiž i přesto hustý, neboť jeho řídkost je závislá na počtu stretchingem generovaných řádků a struktuře řídké části A_s. Možným důvodem pro vznik zaplnění ve faktoru může být fakt, že řádkový stretching rozdělí hustý řádek do řádků o rovnocenném počtu nenulových prvků, navíc typicky za sebou seřazených. To nemusí být kompatibilní se zbytkem matice. Toto je demonstrováno na třech jednoduchých příkladech v článku [34], kde je z tohoto důvodu navržena jiná varianta tzv. řídkého stretchingu, která se snaží minimalizovat zaplnění v Choleského faktoru.

Nechť je zachováno značení z předchozí sekce 2.4.1. Buď tedy d^T hustý řádek matice A_d, D^T buď z d^T stretchingem generovaná řídká matice a

$$\tilde{A} = \begin{pmatrix} A_s \\ D^T \end{pmatrix}$$

buď celá matice soustavy po aplikaci stretchingu. Pro názornost buď tedy Choleského faktor soustavy normálních rovnic rozepsán ve tvaru

$$C = A^T A = \begin{pmatrix} A_s^T & D \\ & S^T \end{pmatrix}\begin{pmatrix} A_s & \\ D^T & S \end{pmatrix} = \begin{pmatrix} A_s^T A_s + DD^T & DS \\ S^T D^T & S^T S \end{pmatrix}.$$

Zjevně je pro zaplnění určující blok $A_s^T A_s + DD^T$. Tedy právě zde je soustředěna snaha ho minimalizovat. Pozorování převzatá z článku [34] ukazují, jakým způsobem je nutné analyzovat strukturu a navádějí tak na lepší modifikaci metody stretchingu.

Pozorování 1. *Matice S^TS je třídiagonální, a tudíž počet nenulových prvků v (2,2) bloku Choleského faktoru $\tilde{L}$ závisí pouze na počtu $k \in \mathbb{N}$ stretchingem generovaných řádků. Proto by mělo být k malé.*

Definice 6 (Dominance struktury)**.** *Nechť u, v jsou řídké vektory a Struct(u), Struct(v) množiny indexů nenulových prvků ve vektorech u, v. Potom struktura u dominuje v, pokud*

$$Struct(v) \subseteq Struct(u).$$

Pozorování 2. *Zaplnění v řídícím bloku $A_s^TA_s + DD^T$ Choleského faktoru $\tilde{L}$ způsobené hustým řádkem bude minimální, pokud je struktura normální matice generovaných řádků DD^T obsažena ve struktuře matice $A_s^TA_s$.*

Pro názornost je definice dominance struktury znázorněna Obrázkem 2.3, kde struktura vektoru u dominuje vektorům v a w, ale nedominuje vektoru z.

$$\begin{pmatrix} u^T \\ v^T \\ w^T \\ z^T \end{pmatrix} = \begin{pmatrix} * & & * & * \\ * & & & * \\ & & * & \\ * & * & * & \end{pmatrix}$$

Obrázek 2.3: Příklad dominance struktury.

Na základě Pozorování 1 a 2 se řídký stretching snaží najít rozdělení nenulových prvků řádku d^T mezi co nejméně řídkých strukturálně disjunktních řádků D^T, které jsou dominovány strukturou řídké části A_s. Postup je velmi blízký úloze *minimálního pokrytí množiny* (z anglického termínu *Minimal Set Cover Problem*). Jedná se o NP-úplný problém nalezení minimálního počtu podmnožin nějakého souboru množin, který pokrývá dané univerzum. Právě kvůli složitosti této úlohy je v praxi řídký stretching, jakožto kombinatorická úloha, řešen aproximačně.

Binární řádkový stretching šlo zobecnit pro více řádků bez užití rekurze. Řídký stretching je odvozen pro rozhození jednoho hustého řádku do k řídkých. Tedy pokud má A_d více řádků než jeden lze stejný algoritmus opakovaně aplikovat na jednotlivé husté řádky. Ačkoliv to lze dělat do jisté míry paralelně, stejně řídký stretching naráží na výpočetní náročnost. Stretching bude mít vždy jistý negativní vliv na celkovou časovou a paměťovou náročnost. To je zjevné z principu rozdělení řádků do více a tím často velmi podstatným zvětšením úlohy. Proto sami autoři článku [34] poznamenávají, že by bylo záhodno zobecnit řídký stretching na blokovou variantu.

Navíc lze použít této techniky nejen pro úplné rozklady ale i pro neúplné faktorizace jako předpodmínění pro iterační metodu. Tomu se věnuje podkapitola 3.2.1. Stretching by mohl být též aplikován na úlohy, kde A_s nemá plnou sloupcovou hodnost. Husté řádky A_d by po stretchingu mohly být zpět vráceny k A_s a zajistit tak plnou sloupcovou hodnost.

3. Problém nulových sloupců matice A_S

Úvodní rozdělení matice A na řídkou a hustou část může vytvořit z regulární matice dvě matice, které takové nejsou, např. jsou singulární nebo obsahují nulové sloupce. Příklad takového rozdělení matice je ilustrován Obrázkem 3.1. Hlavně pro A_s to vyžaduje speciální pozornost při volbě a aplikaci řídké metody.

$$\left.\begin{pmatrix} & & 0 & * & & 0 & * & \\ & * & 0 & & & 0 & & \\ & & 0 & * & & 0 & & \\ & & 0 & & & 0 & & * \\ * & & 0 & & & 0 & & * \\ & & 0 & * & & 0 & * & \\ & & 0 & & * & 0 & & \end{pmatrix}\right\} A_s$$
$$\left.\begin{pmatrix} * & * & * & * & * & * & * & * \end{pmatrix}\right\} A_d$$

Obrázek 3.1: Příklad rozdělení matice nesplňující pro A_s předpoklad plné sloupcové hodnosti.

3.1 Metody řešení LS problému s A_s singulární

Nechť nyní matice soustavy nesplňují předpoklad plné sloupcové hodnosti, tedy platí $rank(A) = k \leq \min(m,n)$. Zde bude však jen zběžně nastíněno, jak tento případ řešit, neboť se jedná o obsáhlý problém, který je nad rámec této práce.

Už jen samotné určení hodnosti matice je často komplikované, neboť může záviset na zvolené metodě, může se měnit díky perturbacím matice a přesné vyhodnocení nemusí být často ani možné. Proto se sloupcová hodnost musí někdy odhadovat.

Pokud $k < n$, neexistuje jednoznačné řešení LS problému a za LS řešení se bere to minimální v normě. Takové řešení poskytuje Moore-Penrosova pseudoinverze, kterou lze získat např. metodou SVD rozkladu, jak je shrnuto v 1.2.4. Takový postup je obvykle neefektivní pro matice, které nejsou malé a husté. Především proto, že matice rozkladu U a V jsou obecně husté a během transformace matice A na diagonální tvar dochází k velkému zaplnění. Dobrou volbou pro řešení LS problému s maticí, která nemá plnou sloupcovou hodnost, se ukazuje metoda založená na QR rozkladu, proto se o ní text více rozepisuje v sekci 3.1.2. Pravděpodobně vhodnějším postupem je ale využití *Dulmage-Mendelsohnova rozkladu*, viz sekce 3.1.1.

3.1.1 Metoda Dulmage-Mendelsohnova rozkladu

Dulmage-Mendelsohnův rozklad je kanonický rozklad bipartitního grafu reprezentující obecnou obdélníkovou matici pomocí množin jejích řádků, sloupců a nenulových prvků. Rozklad převede čtvercovou i obdélníkovou, nesymetrickou matici do blokově trojúhelníkového tvaru. Algoritmus výpočtu tohoto rozkladu je založen na párování bipartitního grafu nebo vrcholových pokrytí. Teorie Dulmage-Mendelsohnova rozkladu byla poprvé popsána v článku [11].

Zde bude velmi stručně shrnuta aplikace Dulmage-Mendelsohnova rozkladu na LS problém, viz [28]. Buď pro účely tohoto odstavce matice A převedena na blokově dolní trojúhelníkový tvar

$$\begin{pmatrix} A_v & & \\ X & A_s & \\ X & X & A_h \end{pmatrix},$$

kde A_v je přeurčená matice, A_s je čtvercová, A_h je nedourčená a matice X symbolizuje obecnou matici, která může být triviální nebo dokonce nulová. Všechny tři podmatice mají silnou Hallovu vlastnost. Nutno podotknout, že A_s nemá s označením řídké části matice A používaným výše nic společného a jde jen o duplicitní, ale konvenční, značení.

Potom lze řešení LS problému s takovou maticí A určit jen prací s podmaticemi A_h, A_s a A_v. Typicky se matice faktorizují QR rozkladem, kde je díky silné Hallově vlastnosti zachován předpoklad o nevyrušení a tak lze předpovědět strukturu faktoru R velmi přesně. Tím se šetří na paměti i výpočetní náročnosti. Metoda řešení LS problému pomocí Dulmage-Mendelsohnova rozkladu tedy transformuje matici a využije metody QR rozkladu na jednotlivé bloky. Ty lze totiž klasifikovat jako malé a husté, a proto je úplný QR rozklad vhodným nástrojem, viz sekce 1.2.2.

3.1.2 Varianta řídké QR faktorizace

Již bylo zmíněno, že na úlohy s maticí nesplňující předpoklad plné sloupcové hodnosti lze také aplikovat metodu založenou na QR rozkladu, neboť rozklad existuje a lze jej získat stabilním způsobem. Každopádně i tak jsou nutné modifikace, viz [17]. Faktor R z QR rozkladu bude singulární matice, a nelze proto rovnou použít pro řešení příslušného LS problému (2.1). Modifikace metody spočívají ve speciálním přeuspořádání sloupců během faktorizace tak, že v každém kroku je redukován sloupec s největší normou. Tedy rozklad je tvaru

$$QAP = \begin{pmatrix} R & U \\ 0 & T \end{pmatrix},$$

kde $R \in \mathbb{R}^{k\times k}$ je horní trojúhelníková matice a P je matice prováděných permutací. Za předpokladu přesné aritmetiky bude R singulární a $T = 0$. V praxi se nulovost T musí většinou vynucovat pomocí nějakého kritéria, často založeného na velikosti normy T nebo čísle podmíněnosti faktoru R. Potom výpočet LS problému vede na rovnici

$$\begin{pmatrix} R & U \\ 0 & T \end{pmatrix} P^T \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} c \\ d \end{pmatrix},$$

kde pravá strana odpovídá vektoru Qb. Minimálního řešení se pak zaručí ortogonálními transformacemi matice U. Tedy metoda řeší soustavu

$$QAVV^T x = Qb,$$

kde $R \in \mathbb{R}^{k \times k}$ je aktualizovaná, regulární, horní trojúhelníková matice a V je ortogonální matice zohledňující permutace P. Potom pro výsledné LS řešení platí

$$x = V \begin{pmatrix} y \\ 0 \end{pmatrix}, \; kde \; Ry = c$$

Už jen kvůli sloupcovému přístupu je tento postup nekonsistentní s řádkově orientovaným přístupem řídké QR faktorizace pro řešení LS problému, která je rozevedena v sekci 1.3.1.2. To je jeden z důvodů, proč tento postup není úplně vhodný pro řídký LS problém. Článek [17] však tuto problematiku rozkrývá a navrhuje jak vhodněji upravit výpočet pro řídkou úlohu. Metoda v podstatě sekvenčně promítne pravou stranu b do oboru hodnot matice A a výsledek znovu promítne na nulový prostor matice A. V článku je uveden schématický algoritmus pro řešení řídkého LS problému se singulární maticí, který bere v potaz zaplnění.

3.2 Metody řešení LS problému s nulovými sloupci

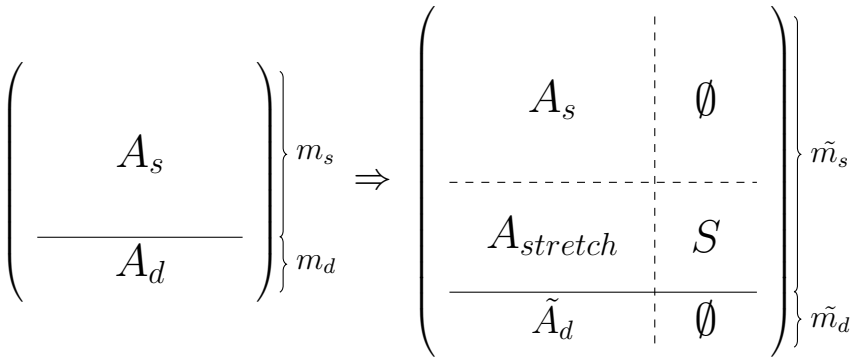
Nechť nyní matice A_s není regulární, protože obsahuje nulové sloupce. Jedná se tak o speciální případ, kdy matice A_s nesplňuje předpoklad plné sloupcové hodnosti. V následujících sekcích jsou shrnuty postupy a možné modifikace výše popsaných metod, kterými lze problém nulových sloupců obejít nebo efektivně řešit. Mezi ně patří blokově předpodmíněná metoda s částečným stretchingem, technika regularizace a metoda kombinace částečných řešení.

3.2.1 Blokově předpodmíněná metoda s částečným stretchingem

Hlavním nebezpečím stretchingu je sice paměťová náročnost úlohy, ale zároveň se může stát, že matice soustavy nemá plnou sloupcovou hodnost. Částečný stretching, ilustrován Obrázkem 3.2, bere v potaz oba tyto problémy. V podstatě jde o řídký stretching, který není aplikovaný na celou hustou část A_d, ale pouze na vybranou podmatici. Podmatice by měla obsahovat takové husté řádky, které po aplikaci stretchingu způsobí zánik nulových sloupců v nově vzniklé řídké části.

Částečný stretching tedy generuje další řídkou část $A_{stretch}$, matici vazby S a zbylou hustou část $\tilde{A}_d$. Přeznačením pak lze získat novou řídkou a hustou část matice soustavy. Stejného postupu lze využít i v případě, že A_s neobsahuje nulové sloupce, ale i přesto nemá plnou sloupcovou hodnost.

Metoda řešení LS problému (2.1) s maticí A_s obsahující nulové sloupce, která je zde představena, kombinuje řídký stretching a předpodmíněnou iterační metodu. Nejprve využije modifikovaného řídkého stretchingu za účelem získání řídké



Obrázek 3.2: Schéma částečného stretchingu.

soustavy splňující předpoklad plné sloupcové hodnosti. Potom se na soustavu aplikuje předpodmínění pomocí blokové faktorizace tak, aby úloha byla dobře řešitelná iterační metodou.

Metoda vychází z (2.7), kde z důvodu výpočetní náročnosti buď nyní $C_s \approx \tilde{L}_s \tilde{L}_s^T$ pouze neúplná Choleského faktorizace matice $A_s^T A_s$. Metoda využívá navíc blokového přístupu, který navádí k použití symetrického pozitivně definitního předpodmínění vhodného pro metodu CGLS. Tedy

$$\begin{pmatrix} C_s & A_d^T \\ A_d & -I \end{pmatrix} \approx \begin{pmatrix} \tilde{L}_s & \\ \tilde{B}_d & \tilde{L}_d \end{pmatrix} \begin{pmatrix} I & \\ & -I \end{pmatrix} \begin{pmatrix} \tilde{L}_s^T & \tilde{B}_d^T \\ & \tilde{L}_d^T \end{pmatrix}, \tag{3.1}$$

kde $\tilde{L}_s \tilde{B}_d^T = A_d^T$ a $I + \tilde{B}_d \tilde{B}_d^T = \tilde{L}_d \tilde{L}_d^T$. Dosazením rovnosti

$$(C_s + A_d^T A_d)^{-1} = \begin{pmatrix} I & 0 \end{pmatrix} \begin{pmatrix} C_s & A_d^T \\ A_d & -I \end{pmatrix}^{-1} \begin{pmatrix} I \\ 0 \end{pmatrix}.$$

do rozkladu (3.1) lze získat symetrické pozitivně definitní předpodmínění vhodné např. pro iterační metody PCGLS, LSQR nebo LSMR. Algoritmus faktorizace s detailním popisem implementace do zmíněných iteračních metod je k nalezení v [35]. Zároveň je zpracován v kódu v přílohách 4.3.

Rozhodovacích kritérií a postupů, podle kterých se určí hustá podmatice pro částečný stretching, je více a rozdíly mezi nimi nebyly zatím důkladně studované. Vhodnou volbu lze proto určit experimentálně. Jednou z možností je jeden po druhém brát ty husté řádky, které mají jako první nenulový prvek v jinak nulovém sloupci A_s. V článku [35] je zvolena varianta LU rozkladu s řádkovou pivotací. Na matici A_d je aplikováno tolik iterací LU rozkladu, kolik má A_s nulových řádků a sloupců. Vzniklé pivotované řádky pak tvoří hledanou podmatici.

3.2.2 Regularizace

Protože $rank(A_s) = n_s < n$ způsobí, že příslušná C_s je pozitivně semidefinitní matice, není možné řešit LS problém pomocí klasické Choleského faktorizace. V tomto případě se totiž rozklad zastaví v kroku, ve kterém je nalezen nulový pivot. Proto metody využívající Choleského rozkladu musí přistoupit k dalším variacím. Jednou možností je úprava matice C_s, nazývaná regularizace, jak je popsáno v [33]. V principu jde o aplikaci posuvu αI_n na C_s ve tvaru

$$C_s(\alpha) = A_s^T A_s + \alpha I_n,$$

kde $\alpha > 0$ je *Tichonovův regularizační parametr*, a $I_n \in \mathbb{R}^{n\times n}$ značí matici identity. Takto posunutý Choleského rozklad už proběhne bez dalších problémů a lze ho aplikovat jako předpodmínění pro iterační metodu. Parametr α je vhodně volen tak, aby Choleského faktorizace nahrazené matice $C_s(\alpha)$ proběhla úspěšně a zároveň α bylo co nejmenší. V praxi se parametr hledá inkrementálně.

Popsaná regularizace má globální vliv, neboť posun αI_n je aplikován na celou diagonálu. To může být již na první pohled zbytečně silný zásah. Proto lze regularizaci lokalizovat a posuvat pouze problematické prvky, které způsobují vznik negativních pivotů v Choleského faktorizaci. Posun αI_n pak může mít různé váhy pro různé diagonální prvky. V článku [33] je ale uvedeno, že ačkoliv lokalizace regularizace není složitá, je globální přístup v případě Choleského faktorizace vhodnější. Tento postup je poprvé navržen v článku [30].

3.2.2.1 Varianta metody Schurova doplňku

Metoda Schurova doplňku může být v případě, že A_s nemá plnou sloupcovou hodnost, regularizována následujícím způsobem.

$$M = \begin{pmatrix} -C_s(\alpha) & A_d^T \\ A_d & I_{m_d} \end{pmatrix} = \begin{pmatrix} \hat{L}_s & \\ \hat{B}_d & I_{m_d} \end{pmatrix} \begin{pmatrix} -I_n & \\ & \hat{S}_d \end{pmatrix} \begin{pmatrix} \hat{L_s^T} & \hat{B_d^T} \\ & I_{m_d} \end{pmatrix}$$

nechť zde značí pravé indefinitní předpodmínění, kde $C_s(\alpha) = \hat{L}_s\hat{L_s^T}$ je Choleského rozklad regularizované matice $C_s(\alpha)$. Úlohu (2.6) lze pak snadnou substitucí převést na tvar

$$K \begin{pmatrix} x \\ r_d \end{pmatrix} = \begin{pmatrix} -A_s^T b_s \\ b_d \end{pmatrix} \quad \Leftrightarrow \quad KMM^{-1} \begin{pmatrix} x \\ r_d \end{pmatrix} = \begin{pmatrix} -A_s^T b_s \\ b_d \end{pmatrix} \tag{3.2}$$

$$\Leftrightarrow \quad KM^{-1} \begin{pmatrix} w_s \\ w_d \end{pmatrix} = \begin{pmatrix} -A_s^T b_s \\ b_d \end{pmatrix}, \; M \begin{pmatrix} x \\ r_d \end{pmatrix} = \begin{pmatrix} w_s \\ w_d \end{pmatrix}. \tag{3.3}$$

První soustavu (3.2) lze řešit pomocí iterační metody. Protože soustava je nesymetrická a předpodmínění je indefinitní, vhodnou metodou je např. GMRES, která byla použita i v experimentech kapitoly 4. Ale lze použít i jiných iteračních metod, např. LSMR. Následně lze na druhou soustavu (3.3) aplikovat analogický postup jako v podkapitole 2.3.

3.2.3 Kombinace částečných řešení

Další možnost, jak si poradit s případem, kdy A_s nemá plnou sloupcovou hodnost, protože obsahuje nulové sloupce, je shrnuta v [32]. Buď proto $rank(A) = n$, ale $rank(A_s) = n_s \ll n$. Potom lze matici A rozdělit na řídkou a hustou část, kde je bez újmy na obecnosti n_2 nulových sloupců matice A_s přeuspořádáno na konec, takto

$$A = \begin{pmatrix} A_s \\ A_d \end{pmatrix} = \begin{pmatrix} A_{s_1} & 0 \\ A_{d_1} & A_{d_2} \end{pmatrix} = \begin{pmatrix} A_1 & A_2 \end{pmatrix}. \tag{3.4}$$

Potom platí, že LS řešení problému (2.1) lze získat kombinací dvou částečných řešení, jak je ukázáno v následující větě. Její důkaz je přímočarý a k nalezení v [32].

Věta 10. *Buďte $z \in \mathbb{R}^{n_1}$, $W \in \mathbb{R}^{n_1 \times n_2}$ řešení minimalizačních úloh*

$$\min_z \|A_1 z - b\|_2 , \qquad \min_W \|A_1 W - A_2\|_F .$$

Potom je $x = (x_1 \quad x_2)^T$ LS řešení problému (2.1), kde $x_1 \in \mathbb{R}^{n_1}$ a $x_2 \in \mathbb{R}^{n_2}$, dáno předpisem

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} z - W x_2 \\ x_2 \end{pmatrix}, \qquad x_2 = (A_2^T (A_2 - A_1 W))^{-1} A_2^T (b - A_1 z).$$

Metoda kombinace částečných řešení je založena na Větě 10. Částečná LS řešení z, W minimalizačních úloh lze vypočítat některou z výše popsaných metod, např. pomocí metody kombinace předpodmínění řídké a husté části popsané v podkapitole 2.2. Protože získat částečné řešení W obnáší řešení n_2 LS problémů dimenze m, je tato metoda časově i výpočetně značně náročná. Platí ale, že se tyto LS řešení dají získat pomocí blokově iterační metody nebo paralelně, a tak výrazně zefektivnit výpočet. V experimentech této práce se ale LS řešení W vyhodnocuje sekvenčně, a proto je nutné zohledňovat spíše průměrný čas jedné iterace než celkovou dobu výpočtu. I přesto tato metoda obstojně konkuruje ostatním. O to více, pokud se do časové náročnosti metod zahrne i stretching a případné jiné kroky předcházející aplikaci metody.

Výpočetní náročnost tohoto postupu také závisí ještě na vyhodnocení matice $A_2^T(A_2 - A_1 W)$. To je zjevně levnější, čím menší je n_2. Protože v praxi je hustá část A_2 typicky malá, je n_2 malé, a tedy vyhodnocení matice $A_2^T(A_2 - A_1 W)$ je levné.

4. Numerické experimenty

V předchozích kapitolách jsou shrnuty vlastnosti a chování různých postupů řešení LS problému. Prvním rozlišovacím faktorem těchto postupů je charakter matice soustavy, a proto byl LS problém nejdříve definován pro malou hustou matici, následně velkou a řídkou a nakonec pro soustavu s proměnlivou hustotou nenulových prvků v řádcích. Experimenty této kapitoly se budou věnovat kombinovanému řídko-hustému LS problému v podobě, v jaké je definován v (2.1), tedy

$$\min_x \left\| \begin{pmatrix} A_s \\ A_d \end{pmatrix} x - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2, \ kde \quad A = \begin{pmatrix} A_s \\ A_d \end{pmatrix}, \ A_s \in \mathbb{R}^{m_s \times n}, \ A_d \in \mathbb{R}^{m_d \times n},$$
$$b = \begin{pmatrix} b_s \\ b_d \end{pmatrix}, \ b_s \in \mathbb{R}^{m_s}, \ b_d \in \mathbb{R}^{m_d},$$
$$m = m_s + m_d, \ m_s \geq n, \ m_d \geq 1.$$

Dříve v textu byly popsány tři metody vhodné pro řešení tohoto LS problému. Byly to metoda Aktualizace 2.1, metoda kombinace předpodmínění řídké a husté části 2.2 (dále už jen jako PCGLS) a metoda Schurova doplňku 2.3. Jak už bylo zmíněno, může se hned několika způsoby stát, že řídká část A_s kombinovaného řídko-hustého LS problému nesplňuje předpoklad plné sloupcové hodnosti a dokonce, že obsahuje nulové sloupce. Protože pak nelze aplikovat klasické varianty známých metod, je nutné do postupu řešení přidat krok, který tuto situaci řeší nebo šikovně obchází. V takovém případě se text rozepisuje o PCGLS s částečným stretchingem 3.2.1, Schurově metodě s regularizací 3.2.2 a Kombinaci částečných řešení 3.2.3. Všechny zmíněné metody byly pro účely této práce naprogramované a použité v experimentech této kapitoly.

Ačkoliv jsou tyto navržené postupy známé a často používané, neexistují žádná pozorování ani vzájemné porovnání při aplikaci na LS problém s proměnlivou hustotou nenulových prvků. Je tedy jejich potenciální přínos zatím nevyhodnocen. Z tohoto důvodu zde budou představeny experimenty, které demonstrují chování těchto přístupů pro různé konkrétní úlohy. Bude zde zkoumána závislost metod na druhu úlohy, na počtu hustých řádků, na počtu nulových sloupců, ale i závislost na parametrech samotných řešičů. Tabulky vypisující výsledky jednotlivých experimentů jsou uspořádány vzestupně podle počtu nenulových prvků v matici, aby se i tento parametr dal snadněji zohlednit při analýze výsledků metod.

Specifikujme zde základní proměnné experimentů. Hustou část soustavy A_d jsme uměle generovali po řádcích. Její dimenze je určena parametrem m_art (z anglického termínu *artificial*). Abychom udrželi celý výpočet *in-cache* a mohli tak lépe porovnávat časovou náročnost jednotlivých metod, uvažovali jsme m_art relativně malé v řádu nízkých desítek. Praktické úlohy se ale někdy potýkají s výrazně dominantnější hustou částí úlohy. To je však nad rámec našich experimentů. Počet hustých řádků, na které byl aplikován stretching, značíme $m\_stretch$, a tedy platí $m\_art \geqq m\_stretch$. Poslední podstatnou proměnnou je počet nulových sloupců, značen n_nulls. Z praktických úloh víme, že takových sloupců se v reálných maticích vyskytuje málo, ale stačí přítomnost jednoho, aby byla celá úloha pro velké dimenze neřešitelná klasickými metodami. V následujících experimentech jsme nulové sloupce uměle generovali, tzn. nulovali jsme posledních

n_nulls sloupců matice A_s. I proto musí být n_nulls relativně malé, abychom výrazně nezměnili matici. Zásadnější narušení struktury totiž může vést na triviální experimenty či zcela bezvýznamnou úlohu.

Matice použité v experimentech byly brány z *SuiteSparse Matrix Collection* (`https://sparse.tamu.edu/`) a jejich abecední výčet je uveden v Tabulce 4.1. Procedura na doplnění matice hustými řádky a stretching řádků je napsána ve Fortranu a byla převzata od vedoucího práce prof. Tůmy. Veškeré zbylé procedury jsou vlastním příspěvkem a jsou přiložené v přílohách 4.3. Tyto procedury jsou programované pro MATLAB, který je vhodným numerickým nástrojem nejen na výpočty, ale i vizualizaci výsledků. Experimenty jsme spouštěli na počítači s procesorem Intel(R) Core(TM) i7-7600U CPU $2.80GHz$ a vnitřní pamětí $16.0GB$. Vlastní kódy jsou programované pro MATLAB R2019b verze 9.7.0.1216025.

Matice	m	n	nz	SPD	rank(A) = n
Meszaros/aircraft	3,754	7,517	20,267	×	✓
Meszaros/cep1	1,521	4,769	8,233	×	✓
Meszaros/co5	5,774	12,325	57,993	×	✓
Meszaros/deter3	7,647	21,777	44,547	×	✓
Meszaros/deter8	3,831	10,905	22,299	×	✓
Meszaros/fxm2-6	1,520	2,845	12,812	×	✓
Meszaros/gams60am	714	1,071	2,607	×	✓
HB/illc1033	1,033	320	4,719	×	✓
LPnetlib/lpi_cplex1	3,005	5,224	10,947	×	✓
LPnetlib/lp-d2q06c	2,171	5,831	33,081	×	✓
LPnetlib/lp_agg	488	615	2,862	×	✓
LPnetlib/lp_pds_02	2,953	7,716	16,571	×	✓
LPnetlib/lp_stocfor2	2,157	3,045	9,357	×	✓
Meszaros/rosen2	1,032	3,080	47,536	×	✓
Meszaros/rosen8	520	1,544	16,058	×	✓
Meszaros/scagr7-2b	9,743	13,847	35,885	×	✓
Meszaros/scagr7-2c	2,447	3,479	9,005	×	✓
HB/well1850	1,850	712	8,755	×	✓

Tabulka 4.1: Matice použité v experimentech a jejich základní vlastnosti. Parametry m, n a nz značí počet řádků, sloupců a nenulových prvků matice. Ve sloupcích SPD a $rank(A) = n$ je vyneseno, zdali je matice symetrická, pozitivně definitní, a jestli má plnou sloupcovou hodnost.

Pokud pro použitou matici A platilo $m < n$, potom jsme matici transponovali, abychom řešili LS problém s přeurčenou úlohou. V některých experimentech, např. regularizaci 4.1 nebo u částečného stretchingu 4.2, bylo potřeba matici A škálovat. V experimentu regularizace tedy místo matice A probíhal výpočet s maticí $\frac{1}{a_{max}}A$, kde a_{max} značí největší prvek matice A. V experimentech se stretchingem je matice zprava přeškálována diagonální maticí D na AD, kde $(D_{ii})^2 = \frac{1}{\|Ae_i\|}$ pro e_i nulový vektor s jedničkou na i-té pozici. Vektor pravé strany b je pro jednoduchost volen jako vektor samých jedniček.

Zastavovací kritérium bylo založeno na velikosti podílu

$$\frac{\left\|A^T r_k\right\|}{\|r_0\|} < TOL,$$

kde r_0 značí startovací reziduum, r_k reziduum v k-té iteraci a TOL je pevně zvolená tolerance $TOL = 10^{-8}$. Pro detailnější studium zastavovacích kritérií jsou vhodné např. kniha [21] nebo článek [23]. Navíc jsme omezili ještě počet iterací provedených zvolenou iterační metodou na $max\_iter = 200$.

Jako předpodmínění soustavy normálních rovnic $A^T A$ jsme v experimentech zvolili neúplnou Choleského faktorizaci. Ta funguje relativně spolehlivě a je populárním nástrojem. Navíc právě proto, že Choleského faktorizace lze úspěšně použít jenom na symetrické pozitivně definitní matice, jsou v našem případě zapotřebí postupy vhodné pro matice s nulovými sloupci. Konkurentem by mohla být neúplná QR faktorizace. Tu však zatím nelze počítat efektivně a robustně, proto ji nebylo možné použít. Hlavní aplikací QR rozkladu jsou stále úlohy s malou a hustou maticí, kde se počítají úplné faktorizace. Většímu rozboru chování a vlastností těchto faktorizací se věnují sekce 1.3.1.2 a 1.3.1.3. Novější články věnující se předpodmiňování ukazují, že neúplnou Choleského faktorizaci lze výrazně vylepšit nebo nahradit jiným lepším předpodmiňovačem, viz [31] nebo [12]. To je ale nad rámec této práce.

4.1 Regularizace

Věnujme se v první řadě technice regularizace a porovnejme její chování a výkon pro všechny tři výše zmíněné metody Aktualizace, PCGLS a Schurova doplňku. Zde pokud by v nějakém kroku výpočtu nebyla možná Choleského faktorizace, protože narazí na nulový pivot a tedy tzv. *breakdown*, počítejme s Choleského faktorizací regularizované matice ve tvaru

$$A(\alpha) = A + \alpha I,$$

kde I značí matici identity. Regularizační parametr α se v praktických aplikacích hledá inkrementálně. Zjevně čím menší je parametr α, tím menší je dopad regularizace, a tedy i menší odchylka od původní matice A, resp. A_s, což je žádoucí. Zároveň ale může být regularizovaná matice stále blízko maticím nesplňující předpoklad plné sloupcové hodnosti. To pak může vést k celkové nestabilitě výpočtu. Tedy volba příliš malého α nemusí zajistit konvergenci metody k LS řešení nebo jen negativně ovlivní počet iterací. Optimální hodnota parametru α je navíc závislá i na velikosti prvků matice. Proto je záhodno před začátkem výpočtu matici škálovat. Pro hlubší analýzu volby parametru α doporučujeme článek [22], který uvádí na námi použitou volbu parametru $\alpha = 10^{-3}, 10^{-5}$. Software MATLAB nabízí možnost regularizace pro neúplnou Choleského faktorizaci. Příkazem

```
L = ichol(A, struct('diagcomp',alpha));
```

lze konstruovat Choleského faktor L takový, že $L^T L \approx M$ splňující

```
alpha = max(sum(abs(A),2) ./ diag(A))-2;
M = A + alpha*diag(diag(A));
```

Tedy L je neúplný Choleského faktor diagonálně dominantní matice, která vznikla z A posunutím o `alpha*diag(diag(A))`. Tato varianta regularizace, ač se zdá mnohem sofistikovanější, nezlepšuje průběh zvolené metody výrazněji než námi aplikovaný posun diagonály.

Sledujme nyní rozdíl mezi metodou Aktualizace řešení LS problému, PCGLS a metodou Schurova doplňku. Porovnání těchto metod je založeno na počtu iterací a výpočetním času v sekundách, které jsou potřeba k získání stejně přesného LS řešení. Protože počet hustých řádků m_art nemá na tento druh experimentu významný vliv, byl též pevně zvolen $m\_art = 1$.

Pokud matice A_s neobsahuje žádné nulové sloupce, lze volit regularizační parametr $\alpha = 0$. Pro srovnání jsme i tento případ zahrnuli do experimentu. Metoda Schurova doplňku je v takovém případě přímá metoda, a nemá proto smysl porovnávat pro ni počet iterací. Výpočetní čas je ovlivněn architekturou použitého počítače a souběžně běžících procesů. Nevypisujeme zde přesné hodnoty, ale jenom srovnání pořadí, ve kterém jednotlivé metody skončily. Poslední sloupec Tabulky 4.2 vyznačuje, která z metod byla z hlediska výpočetního času nejrychlejší.

Metoda Aktualizace aplikuje sdružené gradienty předpodmíněné Choleského faktorizací na regularizovanou matici a následně řešení opraví pomocí spočtené korekce. Metoda PCGLS aplikuje na LS problém sdružené gradienty, které jsou vhodně předpodmíněné neúplnou Choleského faktorizací s nulovým zaplněním, aby byl minimalizován dopad existence hustých řádků. Schurova metoda transformuje LS problém na indefinitní úlohu a aplikuje GMRES iterační metodu předpodmíněnou Choleského faktorizací. Velmi zjednodušeně tedy sledujeme vliv regularizace a nulových sloupců na varianty metod sdružených gradientů a GMRES.

Výsledky ukazují, že metoda Aktualizace a PCGLS jsou na počet iterací prakticky stejné. Metoda Aktualizace vyžaduje obecně o trochu méně iterací, ale musíme brát v úvahu, že po skončení iteračního procesu musí LS řešení x_s ještě aktualizovat. Čas potřebný na samotnou aktualizaci je krátký, neboť potřebujeme řešit dvě typicky velmi malé soustavy. Přičemž QR rozklad, pomocí kterého řešíme soustavy, je možné počítat paralelně s iterační metodou pro řešení x_s. Metoda Schurova doplňku se ukazuje pro větší úlohy méně náročná na počet provedených iterací. Avšak čas potřebný na iteraci je vyšší než u sdružených gradientů, a proto vychází PCGLS lépe.

Zřejmě platí úvaha, čím více obsahuje matice A_s nulových sloupců, tj. čím větší n_nulls, tím větší musí být i α, abychom zajistili konvergenci. Ukazujeme, že na počet nulových sloupců n_nulls je nejméně závislá metoda Aktualizace. Ostatní vyžadují více iterací pro větší n_nulls a i větší regularizaci, tj. větší α. Nejméně citlivou metodou na volbu parametru α je metoda Schurova doplňku. Je tomu tak ale v porovnání se sdruženými gradienty. Zajímavé ale je, že pro efektivitu metody PCGLS stačí velmi jednoduché předpodmínění bez zaplnění. V rámci experimentu jsme nezkoumali použití dalších metod, jako jsou třeba LSMR nebo LSQR a zde je jistě prostor pro další analýzu.

Matice	n_nulls	Aktualizace			PCGLS			Schurova m.		Nejrychlejší metoda
		α			α			α		
		0	10^{-3}	10^{-5}	0	10^{-3}	10^{-5}	10^{-3}	10^{-5}	
gams60am	1	7	7	7	8	8	8	4	3	Schur. m.
	10	7	7	7	8	8	8	4	3	
lp_agg	1	5	58	14	5	61	16	62	15	Schur. m.
	10	5	48	15	5	61	16	52	16	
cep1	1	3	5	5	6	6	6	4	3	PCGLS
	10	3	3	3	6	6	5	4	3	
scagr7-2c	1	41	41	41	43	44	43	6	4	Schur. m.
	10	41	41	41	43	44	43	7	4	
lpi_cplex1	1	7	24	9	6	27	10	25	7	Schur. m.
	10	7	24	9	6	28	10	25	7	
rosen8	1	11	11	11	12	12	12	5	3	PCGLS
	10	11	11	11	12	12	12	6	4	
aircraft	1	1	8	7	5	9	8	8	5	PCGLS
	10	1	8	7	5	11	9	10	6	
deter8	1	73	78	78	78	81	81	6	4	PCGLS
	10	73	78	78	78	84	84	7	4	
lp_d2q06c	1	30	200	200	32	200	200	200	200	PCGLS
	10	30	200	200	32	200	200	200	200	
deter3	1	78	83	89	83	88	88	6	4	PCGLS
	10	78	83	83	83	90	90	8	4	
rosen2	1	12	12	12	12	12	12	4	3	PCGLS
	10	12	12	12	12	13	12	5	4	

Tabulka 4.2: Porovnání počtu iterací metod Aktualizace, PCGLS a Schurova doplňku v závislosti na parametru regularizace α a počtu nulových sloupců n_nulls. Poslední sloupec vypisuje nejrychlejší metodu z pohledu výpočetního času v sekundách.

4.2 Částečný stretching

Dalším přístupem k řešení kombinovaného řídko-hustého LS problému s nulovými sloupci v matici A_s, se kterým budeme experimentovat, je částečný stretching. Teorie k tomuto postupu je shrnuta v sekci 3.2.1. Pro připomenutí se jedná o využití techniky stretchingu na $m\_stretch$ vybraných řádků husté části A_d tak, aby jejich spojením s řídkou částí A_s zanikly nulové sloupce, viz Obrázek 3.2. Tímto způsobem je možné získat $C_s = A_s^T A_s$ jako symetrickou a pozitivně definitní matici. Protože stretching výrazně zvětšuje dimenzi matice úlohy, může být částečný stretching, který rozvine pouze některé z hustých řádků m_art, lepší strategií. Částečný stretching tak může pozitivně ovlivnit paměťovou, ale i výpočetní náročnost celého výpočtu.

První otázkou je, jak vybírat husté řádky vhodné ke stretchingu. Lze je např. postupně iterativně přibírat jeden po druhém, dokud není matice C_s symetrická a pozitivně definitní. To je velmi jednoduchý a účinný přístup, který jsme v tomto experimentu zvolili. Další možností je volit postupně husté řádky, které

mají nenulové prvky ve sloupcích, které jsou pro A_s nulové. Toto může ušetřit nějaké místo v paměti. Na rozdíl od prvního přístupu iterativního příbírání řádků tento postup není závislý na pořadí řádků v matici, resp. jejich permutaci. Typicky však nebude takový rozdíl mezi tímto částečných stretchingem a tím s iterativním přibírání řádků. Stojí tedy za zvážení, jestli hledání potřebných řádků není příliš časově náročné. Poslední zde uvedenou a nejsofistikovanější možností je využití LU rozkladu s řádkovou pivotací k určení vhodných hustých řádků, kterou popisuje a ve svých experimentech využívá článek [35].

Kromě výrazného zvětšení úlohy má částečný stretching ještě jednu nevýhodu. I pokud se zbaví nulových sloupců v matici A_s, nemusí být splněn předpoklad plné sloupcové hodnosti, kterou ale předpokládáme u celé matice. Neúplná Choleského faktorizace, která je jedním z hlavních nástrojů pro výpočet, potom nemůže být aplikována, protože nutně narazí na nulový pivot. V takovém případě je možné aplikovat stretching na další husté řádky, dokud nezískáme pozitivně definitní matici soustavy. Často se také využívá kombinace s regularizací, která nezpůsobuje další zvětšování úlohy.

Následující experiment sleduje chování metod Aktualizace, PCGLS a Schurovy metody s využitím částečného stretchingu a jejich závislost na počtu nulových sloupců *n_nulls*, uměle generovaných hustých řádků *m_art* a hustých řádků *m_stretch* vyhrazených pro stretching. Pro úplnost jsme pozorovali, jak počet iterací, nutných k dosažení kýžené přesnosti, tak i výpočetní čas. Čas měříme v MATLABU pomocí příkazů `tic, toc` umístěných v kódu těsně před a těsně po volání funkce metody, viz kódy v přílohách 4.3. Výsledný čas tedy zahrnuje i řešení pomocných systémů a rozkladů a neúplných faktorizací, které mohou výpočetní čas výrazně navyšovat, přičemž by bylo možné je řešit efektivněji.

Přejdeme nyní k výsledkům a pozorování plynoucích z experimentu. Počet hustých řádků *m_art*, které jsou k maticím uměle generovány, negativně ovlivňují výkon všech zkoumaných metod. Tedy čím větší *m_art*, tím větší je časová náročnost výpočtu. Ukazuje se, že počet iterací zůstává ale relativně stabilní. To může být způsobeno neefektivitou rozkladů a neúplných faktorizací, které je nutno počítat v každém volání metody. Odtud je zjevné, že nejen numerické vlastnosti faktorizace, které jsou studovány v kapitole 1, ale i jejich implementace má podstatný význam a dopad na konečný výkon metod.

Čím více je matice úlohy vzdálena od regulárních matic, tj. čím větší je počet nulových sloupců *n_nulls* matice A_s, tím více je potřeba regularizovat. To se úměrně projevuje na počtu hustých řádků *m_stretch*, které jsou vstupem pro částečný stretching. Proto s rostoucím *n_nulls*, roste *m_stretch*, čímž se zvětšují dimenze m_s, n_s matice A_s. Toto vše má za důsledek prodloužení výpočetního času, viz Tabulka 4.4. Navíc obecně platí, že již pouze pro jeden nulový sloupec je zapotřebí aplikovat stretching na poměrně velké množství hustých řádků *m_stretch*. Nejvíce toto demonstrují např. matice *illc1033* a *lp_pds_02*. Časová náročnost je též negativně ovlivněna celkovým počtem nenulových prvků, tedy nejen rozložením, strukturou matice a její dimenzí.

Podívejme se teď na srovnání jednotlivých metod Aktualizace, PCGLS a Schurova doplňku. První dvě metody jsou iterační, a tedy je pro ně možné srovnávat počet iterací, viz Tabulka 4.3. Jak jsme předpokládali, obě metody Aktualizace i PCGLS konvergují v obdobném počtu iterací, neboť se v obojím v základu jedná o sdružené gradienty. Rozhodujícím rozdílem mezi nimi však je hustá

Matice	*m_art*	*m_stretch*	*n_nulls*	metoda Aktualizace	PCGLS
gams60am	10	3	1	2	2
	50	30	5	3	3
	50	40	10	3	3
lpagg	30	10	3	4	4
	50	40	10	3	3
illc1033	40	30	1	200	200
	40	30	10	200	200
cep1	30	10	1	3	3
	30	10	3	3	3
	50	30	10	3	3
well1850	10	3	1	7	7
	40	3	1	7	5
	40	35	10	200	200
scagr7-2c	30	10	1	4	4
	50	30	5	4	5
	50	30	10	4	5
lp_stocfor2	30	10	1	6	8
	30	10	3	6	8
	50	30	10	6	7
lpi_cplex1	30	10	1	5	5
	30	10	3	5	5
	50	40	10	6	6
fxm2-6	10	3	1	5	6
	30	10	3	5	5
	50	30	10	4	5
rosen8	30	10	1	2	2
	30	10	5	30	29
	50	30	10	2	2
lp_pds_02	40	30	1	4	5
	50	30	5	4	4
deter8	30	10	1	5	5
	30	10	5	5	5
lp_d2q06c	30	10	1	5	5
	30	10	3	6	6
	50	30	3	6	6
scagr7-2b	10	3	1	7	7
deter3	10	3	1	6	6
rosen2	10	3	1	2	2
	30	10	2	2	2
	50	30	10	2	2
co5	30	10	1	6	6

Tabulka 4.3: Porovnání počtu iterací metod Aktualizace a PCGLS v závislosti na počtu hustých řádků *m_art*, řádků *m_stretch* určených k stretchingu a nulových sloupců *n_nulls*

Matice	*m_art*	*m_stretch*	*n_nulls*	metoda Aktualizace	PCGLS	Schurova metoda
gams60am	10	3	1	0.173	0.144	0.0962
	50	30	5	24.935	24.5	11.601
	50	40	10	55.336	52.906	21.496
lpagg	30	10	3	0.485	0.401	0.195
	50	40	10	5.100	4.233	1.724
illc1033	40	30	1	11.019	10.569	1.565
	40	30	10	10.495	10.45	1.666
cep1	30	10	1	21.83	19.755	6.755
	30	10	3	19.627	16.572	6.734
	50	30	10	101.14	83.745	32.934
well1850	10	3	1	1.181	1.655	0.388
	40	3	1	1.150	1.127	0.404
	40	35	10	30.921	18.252	18.151
scagr7-2c	30	10	1	131.09	164.62	51.254
	50	30	5	1021.8	1110.4	291.1
	50	30	10	1095.9	1297.5	350.76
lp_stocfor2	30	10	1	83.291	128.56	31.667
	30	10	3	83.575	124.7	30.371
	50	30	10	844.8	1087.8	280.21
lpi_cplex1	30	10	1	429.76	372.47	83.024
	30	10	3	407.9	324.18	80.404
	50	40	10	3302.6	2935.7	619.33
fxm2-6	10	3	1	5.952	8.916	2.399
	30	10	3	28.99	34.835	11.294
	50	30	10	283.06	359.21	151.42
rosen8	30	10	1	2.178	1.731	0.795
	30	10	5	0.893	0.364	0.793
	50	30	10	13.86	9.986	4.452
lp_pds_02	40	30	1	2740.9	3251.9	875.97
	50	30	5	2603.8	2648.5	865.33
deter8	30	10	1	519.96	614.71	249.94
	30	10	5	491.55	542.27	226
lp_d2q06c	30	10	1	89.982	104.48	39.956
	30	10	3	98.476	120.74	32.842
	50	30	3	887.67	903.43	228.93
scagr7-2b	10	3	1	2720.9	2489.9	577.53
deter3	10	3	1	1114.8	1481.3	405.05
rosen2	10	3	1	6.505	4.855	2.158
	30	10	2	13.921	9.763	4.16
	50	30	10	60.052	46.582	20.043
co5	30	10	1	2112.9	2456.2	781.17

Tabulka 4.4: Porovnání výpočetního času v sekundách metod Aktualizace, PCGLS a Schurovy metody v závislosti na počtu hustých řádků *m_art*, řádků *m_stretch* určených k stretchingu a nulových sloupců *n_nulls*.

část A_d. Platí, že pro rostoucí počet hustých řádků, na které nebyl aplikován stretching, je rychlejším přístupem metoda Aktualizace. Tedy pokud je rozdíl $m\_art - m\_stretch$ velký a pro výpočet je podstatnější časová náročnost, tím vhodnější je využít metodu Aktualizace oproti PCGLS. Z výsledků v Tabulce 4.4 je také evidentní, že celkově nejrychlejší je metoda Schurova doplňku a to v některých případech až násobně. Podstatné je ale vysvětlit proč. Zde znovu hrají neopomenutelnou roli neúplné faktorizace a rozklady, které mohou být drahé, jak na paměť tak na čas. Metoda Schurova doplňku pracuje pouze s úplným Choleského rozkladem matice C_s, který MATLAB řeší rychleji než neúplnou Choleského faktorizaci využívanou pro předpodmínění metod Aktualizace a PCGLS. Ani druh neúplné Choleského faktorizace, ať s nulovým zaplněním nebo se stanovenou tolerancí na nulování nenulových prvků, nepředčí efektivitu úplného Choleského rozkladu pro velké matice. Na druhou stranu byly vyvinuty velmi efektivní implementace předpodmínění založených na neúplné Choleského faktorizaci, které by mohly časovou náročnost metod Aktualizace a PCGLS v našem experimentu velmi snížit. Proto použití funkcí již implementovaných v MATLABU nezaručuje optimalitu výpočtu.

4.3 Srovnání metod řešení LS problému s nulovými sloupci

Následující odstavce budou věnovány srovnání metod řešení LS problému s nulovými sloupci tak, jak jsou popsány v podkapitole 3.2. Mezi sebou jsme porovnávali Schurovu metodu s regularizací, předpodmíněnou metodu s částečným stretchingem, konkrétně PCGLS, a metodu kombinace částečných řešení. Sledovali jsme počet iterací, celkový čas v sekundách potřebný k výpočtu, ale i průměrný výpočetní čas na jednu iteraci. Pro každou testovanou matici jsme měnili počet hustých řádků m_art, počet $m\_stretch$ řádků určených pro stretching a počet nulových sloupců n_nulls.

Omezení na přesnost, resp. tolerance pro zastavovací kritérium, a maximální počet provedených iterací jsou stále stejná jako v předchozích experimentech, tedy $TOL = 10^{-8}$ a $max\_iter = 200$. Regularizační parametr pro Schurovu metodu byl volen pevně $\alpha = 10^{-3}$. Protože částečný stretching nezaručuje úspěšný průběh Choleského faktorizace, a protože tak výrazně roste dimenze úlohy, aplikovali jsme v některých případech regularizaci stejně jako u Schurovy metody i u PCGLS. V Tabulkách 4.5, 4.6 a 4.7 je použití regularizace navíc vyznačeno symbolem *.

Nejdříve shrneme výsledky pro jednotlivé metody a potom přejdeme k jejich vzájemnému porovnání. Schurova metoda s regularizací je v počtu iterací opět celkem nezávislá na parametrech m_art, $m\_stretch$ a n_nulls. Časová náročnost metody lehce roste s celkovou dimenzí úlohy, ale na průměrný čas na iteraci to nemá výrazný vliv. Z předchozích experimentů víme, že metoda PCGLS s částečným stretchingem je velmi ovlivněna n_nulls a úměrně tedy i $m\_stretch$. Využitím regularizace v případech, kdy byl počet stretchingem generovaných řídkých řádků příliš vysoký, jsme mohli kontrolovat časovou náročnost výpočtu. Ukazuje se, že PCGLS je citlivá na n_nulls právě kvůli časové náročnosti. Po-

čet iterací této metody zůstává relativně srovnatelný, ale čas potřebný je jednu iteraci výrazně roste, viz Tabulka 4.7. Do počtu iterací ani výpočetního času ale není zahrnut stretching, který metodě předchází. Proto je potřeba srovnávat výsledky pro PCGLS tak, že počítáme s nějakým časem navíc. Metoda kombinace částečných řešení je nejméně předvídatelná metoda. Z experimentů plyne, že počet iterací je ovlivněn spíše *n_nulls*, zatímco čas na výpočet spíše hustými řádky *m_art*. U několika matic je dokonce větší úloha rychlejší než menší se stejným počtem nulových sloupců. Nutno ale podotknout, že se často potýká s problémem singularity matice A_1. Pokud je A_1 blízká singulárním maticím nebo je špatně škálovaná částečné řešení z může být velmi nepřesné a způsobit tak nestabilitu celého postupu. Dokládají to tak např. výsledky pro matice *illc1033*, *lp_pds_02* nebo *well1850*.

Celkově se ukazuje, že Schurova metoda s regularizací je všeobecně spolehlivá a velmi rychlá metoda. Nejen na počet iterací, ale i celkovou časovou náročnost převládá nad PCGLS s částečným stretchingem a metodu kombinace částečných řešení. Nutno zmínit, že částečný stretching je skvělý nástroj, avšak vyžaduje specifičtější aplikaci a není tak robustní jako regularizace. Navíc nejen že je úloha závislá na *m_stretch*, ale i na provedení samotného stretchingu. Zde je na druhou stranu možnost značného vylepšení a dalších analýz. Kombinace částečných řešení je co do počtu iterací nejnáročnější ze všech tří zkoumaných metod. To plyne přímo z definice. Experimenty však dokazují, že i tak může snadno konkurovat ostatním metodám, neboť jde o velmi rychlou metodu, která nevyžaduje žádných předcházejících kroků.

Matice	*m_art*	*m_stretch*	*n_nulls*	Schurova metoda s regul.	PCGLS s část. stretch.	Kombinace částečných řešení
gams60am	10	3	1	3	2	4
	10	3	3	4	4*	206
	30	10	3	4	5*	8
lp_agg	10	3	1	3	4*	4
	10	3	3	3	5*	8
	30	10	3	3	4	8
illc1033	10	3	1	38	110*	202
	10	3	3	35	115*	206
	30	10	3	47	59*	8
well1850	10	3	1	9	4	4
	10	3	3	8	9*	206
	30	10	3	6	7*	8
scagr7-2c	10	3	1	5	5*	5
	10	3	3	5	6*	207
	30	10	3	5	6*	9
lp_stocfor2	10	3	1	3	10*	13
	10	3	3	4	10*	25
	30	10	3	3	8	25
lpi_cplex1	10	3	1	4	5*	6
	10	3	3	4	7*	12
	30	10	3	3	5	12
fxm2-6	10	3	1	5	6	9
	10	3	3	5	11*	18
	30	10	3	4	5	17
rosen8	10	3	1	2	3	2
	10	3	3	3	4*	4
	30	10	3	3	2*	4
lp_pds_02	10	3	1	8	10*	201
	10	3	3	6	8*	203
	30	10	3	5	6*	×
lp_d2q06c	10	3	1	6	13*	8
	10	3	3	6	13*	16
	30	10	3	6	6	16
deter3	10	3	1	6	6	3
	10	3	3	7	9*	5
	30	10	3	6	8	5
rosen2	10	3	1	2	2	4
	10	3	3	2	3*	206
	30	10	3	3	3*	8
co5	10	3	1	3	7*	11
	10	3	3	4	8*	21
	30	10	3	3	6	21

Tabulka 4.5: Porovnání počtu iterací Schurovy metody, PCGLS s částečným stretchingem a Kombinace částečných řešení v závislosti na počtu hustých řádků *m_art*, řádků *m_stretch* určených k stretchingu a nulových sloupců *n_nulls*.

Matice	*m_art*	*m_stretch*	*n_nulls*	Schurova metoda s regul.	PCGLS s část. stretch.	Kombinace částečných řešení
gams60am	10	3	1	0.1971	0.1389	0.0504
	10	3	3	0.1372	0.2247*	1.1188
	30	10	3	0.1334	2.4149*	0.0843
lp_agg	10	3	1	0.1975	0.1517*	0.0963
	10	3	3	0.0866	0.1331*	0.1527
	30	10	3	0.0720	0.3490	0.1590
illc1033	10	3	1	0.0323	1.9739*	1.326
	10	3	3	0.0328	2.2021*	1.3705
	30	10	3	0.0261	5.2782*	0.1363
well1850	10	3	1	0.1316	0.9955	0.4690
	10	3	3	0.1365	2.0038*	16.114
	30	10	3	0.1455	5.8266*	0.9209
scagr7-2c	10	3	1	4.3727	51.242*	17.146
	10	3	3	3.4294	54.159*	491.09
	30	10	3	3.8081	179.54*	31.941
lp_stocfor2	10	3	1	2.2814	38.162*	19.192
	10	3	3	2.7662	38.833*	37.52
	30	10	3	2.6193	124.38	37.259
lpi_cplex1	10	3	1	9.6226	158.05*	156.44
	10	3	3	10.9	209.82*	313.46
	30	10	3	9.2142	353.75	313.28
fxm2-6	10	3	1	1.0357	9.2085	4.5453
	10	3	3	1.0231	16.679*	8.7867
	30	10	3	1.0344	38.15	9.2121
rosen8	10	3	1	0.0789	0.8989	0.7446
	10	3	3	0.0799	1.368*	1.2412
	30	10	3	0.0906	1.7459*	1.1729
lp_pds_02	10	3	1	5.1865	46.427*	61.518
	10	3	3	6.1302	40.212*	68.819
	30	10	3	5.8088	277.13*	×
lp_d2q06c	10	3	1	2.6676	65.137*	22.002
	10	3	3	2.5905	64.533*	44.025
	30	10	3	2.6267	114.26	39.357
deter3	10	3	1	91.038	1356.5	328.62
	10	3	3	99.98	1908*	536.53
	30	10	3	113.73	6208.7	595.48
rosen2	10	3	1	0.3577	4.6112	6.567
	10	3	3	0.3517	6.3218*	219.93
	30	10	3	0.3969	13.23*	14.585
co5	10	3	1	40.605	687.14*	501.16
	10	3	3	39.798	776.88*	853.78
	30	10	3	41.262	2425.1	895.73

Tabulka 4.6: Porovnání výpočetního času v sekundách Schurovy metody, PCGLS s částečným stretchingem a Kombinace částečných řešení v závislosti na počtu hustých řádků *m_art*, řádků *m_stretch* určených k stretchingu a nulových sloupců *n_nulls*.

Matice	*m_art*	*m_stretch*	*n_nulls*	Schurova metoda s regul.	PCGLS s část. stretch.	Kombinace částečných řešení
gams60am	10	3	1	0.0657	0.0695	0.0126
	10	3	3	0.0343	0.0562*	0.0054
	30	10	3	0.0333	0.4830*	0.0105
lp_agg	10	3	1	0.0658	0.3793*	0.0241
	10	3	3	0.0289	0.0266*	0.0191
	30	10	3	0.0240	0.0873*	0.0199
illc1033	10	3	1	0.0009	0.0179*	0.0066
	10	3	3	0.0009	0.0191*	0.0067
	30	10	3	0.0006	0.0895*	0.0170
well1850	10	3	1	0.0146	0.2489	0.1173
	10	3	3	0.0171	0.2227*	0.0782
	30	10	3	0.0243	0.8324*	0.1151
scagr7-2c	10	3	1	0.8745	10.248*	3.4293
	10	3	3	0.6859	9.0265*	2.3724
	30	10	3	0.7616	29.924*	3.549
lp_stocfor2	10	3	1	0.7605	3.8162*	1.4763
	10	3	3	0.6916	3.8833*	1.5008
	30	10	3	0.8731	15.547	1.4904
lpi_cplex1	10	3	1	2.4057	31.609*	26.073
	10	3	3	2.7251	29.975*	26.121
	30	10	3	3.0714	70.751	26.107
fxm2-6	10	3	1	0.2071	1.5348	0.505
	10	3	3	0.2046	1.5162*	0.4882
	30	10	3	0.2586	7.63	0.5419
rosen8	10	3	1	0.0395	0.2996	0.3723
	10	3	3	0.0266	0.3420*	0.3103
	30	10	3	0.0302	0.8730*	0.2932
lp_pds_02	10	3	1	0.6483	4.6427*	0.3061
	10	3	3	1.0217	5.0265*	0.3390
	30	10	3	1.1618	46.189*	×
lp_d2q06c	10	3	1	0.4446	5.0105*	2.7502
	10	3	3	0.4317	4.964*	2.7516
	30	10	3	0.4378	19.043	2.4598
deter3	10	3	1	15.173	226.09	109.54
	10	3	3	14.283	212*	107.31
	30	10	3	18.955	776.08	119.1
rosen2	10	3	1	0.1789	2.3056	1.6417
	10	3	3	0.1759	2.1073*	1.0676
	30	10	3	0.1323	4.41*	1.8231
co5	10	3	1	13.535	98.1628*	45.56
	10	3	3	9.9495	97.11*	40.6562
	30	10	3	13.754	404.1833	42.6538

Tabulka 4.7: Porovnání podílu času v sekundách na počet iterací Schurovy metody, PCGLS s částečným stretchingem a Kombinace částečných řešení v závislosti na počtu hustých řádků *m_art*, řádků *m_stretch* určených k stretchingu a nulových sloupců *n_nulls*.

Závěr

Předmětem této diplomové práce byl problém nejmenších čtverců a jeho speciální aplikace na soustavy s maticí o proměnlivé hustotě nenulových prvků v řádcích. Tuto úlohu nazýváme kombinovaný řídký-hustý LS problém. Nejdříve se text zabýval definováním úlohy, existenci a jednoznačnosti LS řešení a metodám LS řešení. V první řadě byly rozvedeny přímé metody vhodné pro malé a husté soustavy. Potom byly rozepsány jejich zobecnění a iterační metody pro soustavy s velkou a řídkou maticí. Uvedené metody jsou ale specializované tak, aby efektivně řešili buď hustou nebo řídkou úlohu. Ukazuje se však, že efektivním řešením kombinovaného řídko-hustého LS problému bude nejspíše jistá kombinace přímé a iterační metody.

Předpokládali jsme, že matice soustavy je velká, její značná část je považována za řídkou a také, že počet řádků, které jsou husté, je malý. Takové úlohy totiž plynou z mnoha praktických aplikací. Maticově zapsáno tedy kombinovaný řídký-hustý LS problém odpovídá

$$\min_x \left\| \begin{pmatrix} A_s \\ A_d \end{pmatrix} x - \begin{pmatrix} b_s \\ b_d \end{pmatrix} \right\|_2, \ kde \quad A = \begin{pmatrix} A_s \\ A_d \end{pmatrix}, \ A_s \in \mathbb{R}^{m_s \times n}, \ A_d \in \mathbb{R}^{m_d \times n},$$
$$b = \begin{pmatrix} b_s \\ b_d \end{pmatrix}, \ b_s \in \mathbb{R}^{m_s}, \ b_d \in \mathbb{R}^{m_d},$$
$$m = m_s + m_d, \ m_s \geq n, \ m_d \geq 1.$$

V práci jsme popsali metody řešení tohoto LS problému, které vhodně zacházejí s přirozeným rozdělením úlohy na hustou a řídkou část a tak se efektivně snaží najít LS řešení. První metodou byla Aktualizace řešení LS problému, která získá LS řešení na základě řídké části, většinou aplikací předpodmíněné iterační metody, a následně toto řešení aktualizuje pomocí zbylých hustých řádků. Z experimentů plyne, že se jedná o elegantní a rychlou metodu, která lze snadno paralelizovat a dobře využít, pokud je hustých řádků málo nebo postupně přibývají v průběhu výpočtu.

Přítomnost hustých řádků komplikuje výpočet primárně kvůli vznikajícímu zaplnění, když formulujeme úlohu ve tvaru normálních rovnic. S tím je možné si ale poradit aplikací iterační metody se speciálně navrženým předpodmíněním, které minimalizuje nepříznivý dopad hustých řádků. Jako příklad takové metody jsme uvedli metodu kombinace předpodmínění řídké a husté časti, která využívá metodu sdružených gradientů a Choleského faktorizaci. Celkem předvídatelně se tento postup ukazuje jako velmi užitečná metoda. Díky numerickým vlastnostem sdružených gradientů postačí jednoduché předpodmínění pro docílení rychlé a časově nenáročné metody. Obdobných výsledků se dostává i pro metodu Schurova doplňku. Ta transformuje úlohu na symetrickou, kvazi-definitní soustavu, kterou pak řeší s ohledem na blokovou strukturu, a tedy úsporněji než jiné indefinitní metody. V experimentech je ukázáno, že pro mnohé matice je toto nejrychlejší metoda, jak na celkový čas, tak i na počet iterací.

Husté řádky je tedy možno nejdříve ignorovat a zohlednit až na konci výpočtu, nebo s nimi vhodně pracovat již na začátku. Variantou řešení kombinovaného řídko-hustého LS problému, která zachází s hustými řádky již před samotným výpočtem, je stretching. Tato technika rozdělí husté řádky do více již řídkých. Tedy

je možné následně využít efektivní řídké metody. Nevýhodou stretchingu je ale nárůst dimenze, který bývá velký. Proto existuje více postupů a implementací stretchingu, každý z nich vhodnější na jiné matice. Částečný stretching, který byl využit v experimentech práce, je speciální varianta stretchingu aplikovananá pouze na vybranou podmnožinu hustých řádků. Tím je sice regulován nárůst dimenze matice soustavy, je ale zapotřebí využít některé z předchozích metod.

Rozdělení úlohy na dvě části, které umožňuje efektivně řešit úlohu pomocí výše zmíněných metod, se potýká s jednou zásadní překážkou. Ačkoliv má celá soustava plnou sloupcovou hodnost, není to ničím zaručeno pro řídkou nebo hustou část. Proto je velmi pravděpodobné, že metody LS řešení, které předpokládají regulární matici, není možné použít. My jsme se v této práci věnovali právě tomuto případu, kdy speciálně řídká část matice obsahuje nulové sloupce. Všechny doposud zmíněné metody obsahovaly někde ve výpočtu Choleského faktorizaci matice $A_s^T A_s$, která nutně narazí na nulový pivot kvůli přítomnosti nulových sloupců. Je proto nutné přidat kroky, které tuto situaci obchází nebo řeší.

Jednou z možností je využití částečného stretchingu. Tedy pokud řídká část obsahuje nulové sloupce, je možné aplikovat stretching na vybrané vhodné husté řádky, přidat je k řídké části a tak zajistit regularitu matice. Experimenty ukazují, že je zapotřebí relativně sofistikovaného rozhodovacího kritéria na volbu hustých řádků pro stretching. I v případě, že řídká část obsahuje pouze jeden nulový sloupec, je potřeba aplikovat stretching na relativně velké množství hustých řádků. Tak se ale zvětšuje úloha a to má negativní vliv na časovou náročnost. To lze ale balancovat metodou LS řešení. Metoda Schurova doplňku se ukazuje jako velmi dobrý řešič jak z časového, ale i výpočetního pohledu. Je to však přímo ovlivněno efektivitou implementace Choleského faktorizace při předpodmínění. Metoda Schurova doplňku pracuje s úplnou Choleského faktorizací, zatímco metody Aktualizace či PCGLS počítají s neúplnou Choleského faktorizací. Pokud je tedy některá z faktrizací navržena lépe, efektivněji, pak bude i příslušná metoda dávat lepší výsledky. Z výsledků kapitoly 4 také vyplývá, že metoda Aktualizace je časově méně náročná než metoda PCGLS, pokud i po stretchingu zůstává velké množství hustých řádků.

Jak již bylo řečeno, nulové sloupce způsobují breakdown Choleského faktorizace, neboť se v průběhu rozkladu vyskytne nulový pivot. Tuto skutečnost lze ale řešit malou perturbací matice, na kterou je faktorizace aplikována. Tato technika se nazývá regularizace. Ačkoliv očekáváme, že z časového i výpočetního hlediska bude metoda PCGLS nejlepší z porovnávaných metod, ukazuje se, že metoda Schurova doplňku je velmi dobrý konkurent. Ta není tolik závislá na volbě regularizačního parametru a nevyžaduje proto takovou míru zkušenosti při řešení některých úloh. Regularizace obstojně funguje i pro metodu Aktualizace, která z porovnání vychází jako vhodnější pro vysoký počet nulových sloupců.

Pokud soustavu rozdělíme podle hustoty nenulových prvků v řádcích a ještě navíc podle nulových sloupců, jak je uvedeno v (3.4), je možné použít metodu kombinace částečných řešení. Tato metoda tedy přeuspořádá nulové sloupce na konec matice a řeší kombinovaný řídký-hustý LS podproblém a speciální soustavu s nulovým blokem. K získání LS řešení je tedy nutné vyhodnotit celkem tolik řídko-hustých LS problémů, kolik má matice nulových sloupců. Časová a výpočetní náročnost je tedy typicky násobná oproti ostatním srovnávaným metodám. To lze však řešit pomocí blokových implementací a paralelního výpočtu.

I přes celkovou náročnost metody je ale čas potřebný na jednu iteraci metody často nižší a někdy dokonce i výrazně.

V práci jsme ukázali, jak postupovat při řešení kombinovaného řídko-hustého LS problému. Představili jsme různé metody, shrnuli jejich numerické vlastnosti a nastínili jsme jejich aplikovatelnost. V praktické části jsme porovnali chování vybraných metod v závislosti na druhu úlohy. Zaměřili jsme se na problém nulových sloupců v řídké části matice, který se může velmi často objevit a způsobit tak komplikace. Snažíme se tak upozornit na novější přístupy řešení, které se ukazují být v mnohých případech lepšími řešiči než běžně využívané předpodmíněné iterační metody. Proto jsme se věnovali nejen studiu chování jednotlivých metod, ale i vzájemnému porovnání mezi nimi. Jednotlivé kroky metod by bylo možné sledovat důkladněji a detailnější analýzou je i vylepšit. V přední řadě jsme počítali s neúplnou Choleského faktorizací jako předpodmínění pro iterační metodu PCGLS, která lze aplikovat lépe nebo zcela nahradit lepším předpodmíněním. Dále jsme porovnávali metody využívající pouze sdružených gradientů, které jsou osvědčenou a velmi efektivní metodou, avšak by bylo zajímavé srovnávat i jiné iterační metody, např. LSMR nebo LSQR. Časová náročnost metod by také mohla být zrychlena využitím architektury počítače, paralelním počítáním nebo blokovými přístupy. U všech těchto připomínek je prostor k další práci, analýze a experimentům.

Seznam použité literatury

[1] M. Adlers and Å. Björck. Matrix stretching for sparse least squares problems. *Numerical linear algebra with applications*, 7(2):51–65, 2000.

[2] C. Ashcraft. Compressed graphs and the minimum degree algorithm. *SIAM Journal on Scientific Computing*, 16(6):1404–1411, 1995.

[3] Å. Björck. A general updating algorithm for constrained linear least squares problems. *SIAM Journal on Scientific and Statistical Computing*, 5(2):394–402, 1984.

[4] Å. Björck. *Numerical methods for Least Squares Problems*. SIAM, Philadelphia, 1996.

[5] Å. Björck and I. S. Duff. A direct method for the solution of sparse linear least squares problems. *Linear Algebra and its Applications*, 34:43–67, 1980.

[6] Å. Björck, T. Elfving, and Z. Strakoš. Stability of conjugate gradient and lanczos methods for linear least squares problems. *SIAM Journal on Matrix Analysis and Applications*, 19(3):720–736, 1998.

[7] R. Brualdi and B. Shader. Strong hall matrices. *Siam Journal on Matrix Analysis and Applications*, 15, 1994.

[8] I. S. Duff and J. K. Reid. A comparison of some methods for the solution of sparse overdetermined systems of linear equations. *IMA Journal of Applied Mathematics*, 17(3):267–280, 1976.

[9] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear. *ACM Transactions on Mathematical Software*, 9(3):302–325, 1983.

[10] J. Duintjer Tebbens, I. Hnětynková, M. Plešinger, Z. Strakoš, and P. Tichý. *Analýza metod pro maticové výpočty. Základní metody*. Vydání první. Matfyzpress, MFF UK, 2012.

[11] A. L. Dulmage and N. S. Mendelsohn. Coverings of bipartite graphs. *Canadian Journal of Mathematics*, 10:517–534, 1958.

[12] F. Durastante and D. Bertaccini. *Iterative Methods and Preconditioning for Large and Sparse Linear Systems with Applications*. CRC Press, 2018.

[13] D. Fong and M. Saunders. LSMR: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, 33:2950–2971, 2011.

[14] A. George and M. T. Heath. Solution of sparse linear least squares problems using givens rotations. *Linear Algebra and its Applications*, 34:69–83, 1980.

[15] W. Givens. Computation of plain unitary rotations transforming a general matrix to triangular form. *Siam Journal on Matrix Analysis and Applications*, 6(1):26–50, 1958.

[16] J. F. Grcar. Matrix stretching for linear equations. *arXiv preprint arXiv:1203.2377*, 2012.

[17] M. T. Heath. Some extensions of an algorithm for sparse linear least squares problems. *SIAM Journal on Scientific and Statistical Computing*, 3(2):223–237, 1982.

[18] M. T. Heath. Numerical methods for large sparse linear least squares problems. *SIAM Journal on Scientific and Statistical Computing*, 5(3):497–513, 1984.

[19] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49:409–435, 1952.

[20] A. S. Householder. Unitary triangularization of a nonsymmetric matrix. *ACM Transactions on Mathematical Software*, 5(4):339–342, 1958.

[21] J. Liesen and Z. Strakoš. *Krylov subspace methods*. Numerical Mathematics and Scientific Computation. Oxford University Press, Oxford, 2013. Principles and analysis.

[22] C. J. Lin and J. J. Moré. Incomplete cholesky factorizations with limited memory. *SIAM Journal on Scientific Computing*, 21(1):24–45, 1999.

[23] G. Meurant and P. Tichý. Approximating the extreme ritz values and upper bounds for the a-norm of the error in cg. *Numerical Algorithms*, 2018.

[24] C. Paige and Z. Strakoš. Scaled total least squares fundamentals. *Numerische Mathematik*, 91(1):117–146, 2002.

[25] C. Paige and Z. Strakoš. Core problems in linear algebraic systems. *SIAM Journal on Matrix Analysis and Applications*, 27(3):861–875, 2005.

[26] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, 1982.

[27] G. Peters and J. H. Wilkinson. The least squares problem and pseudo-inverses. *The Computer Journal*, 29:309–316, 1970.

[28] A. Pothen and C. J. Fan. Computing the block triangular form of a sparse matrix. *ACM Transactions on Mathematical Software*, 16(4):303–324, 1990.

[29] Y. Saad. *Iterative Methods for Sparse Linear Systems*, volume 82. Society for Industrial and Applied Mathematics, second edition, 2003.

[30] J. Scott. On using cholesky-based factorizations and regularization for solving rank-deficient sparse linear least-squares problems. *SIAM Journal on Scientific Computing*, 39(4):C319–C339, 2017.

[31] J. Scott and M. Tůma. On positive semidefinite modification schemes for incomplete cholesky factorization. *SIAM Journal on Scientific Computing*, 36, 2014.

[32] J. Scott and M. Tůma. Solving mixed sparse-dense linear least squares problems by preconditioned iterative methods. *SIAM Journal on Scientific Computing*, 39:A2422–A2437, 2017.

[33] J. Scott and M. Tůma. A schur complement approach to preconditioning sparse linear least squares problems with some dense rows. *Numerical Algorithms*, 79(4):1147–1168, 2018.

[34] J. Scott and M. Tůma. Sparse stretching for solving sparse-dense linear least-squares problems. *SIAM Journal on Scientific Computing*, 41(3):A1604–A1625, 2019.

[35] J. Scott and M. Tůma. Strengths and limitations of stretching for least-squares problems with some dense rows. *ACM Transactions on Mathematical Software*, 2019.

[36] R. Vanderbei. Symmetric quasidefinite matrices. *Siam Journal on Optimization*, 5, 1995.

[37] J. Wilkinson and C. Reinsch. *Handbook for Automatic Computation*, volume II: Linear Algebra. Springer, 1971.

Seznam obrázků

2.1 Schéma řídké matice s jedním hustým řádkem. 22
2.2 Příklad stretchingu jednoho hustého řádku. 28
2.3 Příklad dominance struktury. 31

3.1 Příklad rozdělení matice nesplňující pro A_s předpoklad plné sloupcové hodnosti. 32
3.2 Schéma částečného stretchingu. 35

Seznam tabulek

4.1 Matice použité v experimentech a jejich základní vlastnosti. Parametry m, n a nz značí počet řádků, sloupců a nenulových prvků matice. Ve sloupcích SPD a $rank(A) = n$ je vyneseno, zdali je matice symetrická, pozitivně definitní, a jestli má plnou sloupcovou hodnost. 39
4.2 Porovnání počtu iterací metod Aktualizace, PCGLS a Schurova doplňku v závislosti na parametru regularizace α a počtu nulových sloupců n_nulls. Poslední sloupec vypisuje nejrychlejší metodu z pohledu výpočetního času v sekundách. 42
4.3 Porovnání počtu iterací metod Aktualizace a PCGLS v závislosti na počtu hustých řádků m_art, řádků $m\_stretch$ určených k stretchingu a nulových sloupců n_nulls 44
4.4 Porovnání výpočetního času v sekundách metod Aktualizace, PCGLS a Schurovy metody v závislosti na počtu hustých řádků m_art, řádků $m\_stretch$ určených k stretchingu a nulových sloupců n_nulls. 45
4.5 Porovnání počtu iterací Schurovy metody, PCGLS s částečným stretchingem a Kombinace částečných řešení v závislosti na počtu hustých řádků m_art, řádků $m\_stretch$ určených k stretchingu a nulových sloupců n_nulls. 48
4.6 Porovnání výpočetního času v sekundách Schurovy metody, PCGLS s částečným stretchingem a Kombinace částečných řešení v závislosti na počtu hustých řádků m_art, řádků $m\_stretch$ určených k stretchingu a nulových sloupců n_nulls. 49
4.7 Porovnání podílu času v sekundách na počet iterací Schurovy metody, PCGLS s částečným stretchingem a Kombinace částečných řešení v závislosti na počtu hustých řádků m_art, řádků $m\_stretch$ určených k stretchingu a nulových sloupců n_nulls. 50

Přílohy - Kódy pro Matlab

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%% - PCGLS - %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% VSTUP
% A_s        % ridka cast A
% A_d        % husta cast A
% b_s        % ridka cast prave strany b
% b_d        % husta cast prave strany b
% x0         % pocatecni aproximace reseni
% max_iter   % maximalni pocet iteraci
             % zastavovaci kriterium
% tol        % tolerance pro podil norem rnorm0 / rnorm
             % zastavovaci kriterium
% facts      % typ faktorizace pro A_s'*A_s;
             % 0 - uplna Chol. fakt.
             % 1 - neuplna Chol. faktorizace s nulovym zaplnenim
             % 2 - neuplna Chol. faktorizace s toleranci
% alpha      % regularizacni parameter

%% VYSTUP
% x          % reseni
% iter       % pocet iteraci
% resvec     % vektor norem rezidua v kazde iteraci

%%
function [x, iter, resvec] = precond_CGLS_ST (A_s, A_d, b_s, ...
    b_d, x0, max_iter, tol, facts, alpha)

[m_s, n_s] = size(A_s);
[m_d, n_d] = size(A_d);

A = A_s;
A(m_s+1 : m_s+m_d, 1:n_s) = A_d;
[m,n] = size(A);
b = b_s;
b(m_s+1 : m_s+m_d, 1) = b_d;

%% CGLS
% inicializace
x = x0;
r = b - A*x;           % pocatecni reziduum;
r_s = r(1:m_s);
r_d = r(m_s+1:m);
w = A'*r;              % reziduum normalnich rovnic;
                       % <=> w = w_s + w_d = (A_s' * r_s) + ...
                           (A_d' * r_d);
w_s = (A_s' * r_s);
rnorm0 = norm(r);      % pocatecni norma rezidua
beta_aux = 1;

% faktorizace A_s
I = eye(n_s);
```

```
C_s = sparse((A_s'*A_s)+alpha*I);
if (facts == 0)
    L_s = chol(C_s, 'lower');
elseif (facts == 1)
    L_s = ichol(C_s, struct('type','nofill','shape','lower'));
elseif (facts == 2)
    L_s = ichol(C_s, struct('type','ict','droptol',3e-05,
                'michol','off','diagcomp',0,'shape','lower'));
end

B_d = A_d * inv(L_s');
B_s = A_s * inv(L_s');

for i = 1:max_iter
% predpodmineni
  L_d = chol(B_d*B_d' + eye(m_d),'lower'); % rozklad ...
      pseudoinverze
  x1 = inv(L_s)*w_s;              % transformace rezidua
  rho_d = r_d - (B_d*x1);         % = r_d - A_d*v
                                  % = r_d - A_d*inv(L_s')*x1
                                  % = r_d - ...
                                     A_d*inv(L_s')*inv(L_s)*w_s
                                  % = r_d - A_d*inv(C_s)*w_s
                                  % = r_d - A_d*inv(C_s)*A_s'*r_s
  u = B_d'*inv(L_d*L_d')*rho_d; % = ...
      (A_d*inv(L_s'))'*inv(C_d')*rho_d
                                  % = inv(L_s)*A_d'*inv(C_d')*rho_d
  z = inv(L_s')*(x1 + u(1:n));

  beta_aux0 = beta_aux;
  beta_aux = z'*w;

  if (i==1)
    p = z;
  else
    beta = beta_aux / beta_aux0;
    p = z + (beta*p);
  end

  q = A*p;                        % q_s = A_s*p; q_d = A_d*p;
  alpha = (z'*w) / (q'*q);
  x = x + (alpha*p);              % aktualizace aproximace
  r = r - (alpha*q);              % aktualizace rezidua
  r_s = r(1:m_s);
  r_d = r(m_s+1:m);
  resvec(i,1) = norm(r);

  w = A'*r;
  w_s = A_s'*r_s;                 % w_d = A_d'*r_d;
                                  % w = w_s+w_d;
  rnorm = norm(w);
  iter = i;

  if (rnorm/rnorm0 < tol)
    break;
  end

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% - Metoda Aktualizace - %%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% VSTUP
% A_s       % ridka cast A
% A_d       % husta cast A
% b_s       % ridka cast prave strany b
% b_d       % husta cast prave strany b
% x0        % pocatecni aproximace
% max_iter  % maximalni pocet iteraci
            % zastavovaci kriterium
% tol       % tolerance pro podil norem rnorm0 / rnorm
            % zastavovaci kriterium
% alpha     % regularizacni parameter

%% VYSTUP
% x         % reseni
% iter_xs   % pocet iteraci PCGLS pro reseni ridke casti
% resvec_xs % vektor norem rezidua v kazde iteraci PCGLS

%%
function [x, iter_xs, resvec_xs] = updating (A_s, A_d, b_s, ...
    b_d, x0, max_iter, tol, alpha)

% inicializace
[m_s, n_s] = size(A_s);
[m_d, n_d] = size(A_d);

% Vyres min|| A_s x - b_s || pomoci PCGLS.
% x_s reseni
x0_s = x0(1:n_s,1);
I = eye(n_s);
C_s = sparse((A_s'*A_s)+alpha*I);
M = ichol(C_s, struct('shape','upper'));
[x_s, relres_xs, iter_xs, resvec_xs] = pcgls (A_s, b_s, x0_s, ...
    M, tol, max_iter);

% Vyres (u) = Q_c (inv(R_c)'*r_d) pro u
%       (v) =     ( 0           )
% kde Q_c R_c = C' = (B_d I_md)'; B_d = A_d*inv(R_s), Q_s R_s ...
    = A_s.
r_d = b_d - (A_d * x_s);
R_s = chol(C_s, 'upper');
B_d = A_d*inv(R_s)';
I_md = eye(m_d);
C = [B_d I_md];
[Q_c, R_c] = qr(C', 0); % ekonomicky QR rozklad
uv = Q_c * inv(R_c)' * r_d;
u = uv(1:n_s, 1);

% Vyres R_s z = u pro z.
z = inv(R_s)*u;

% Aktualizace x_s.
x = x_s + z;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% - Metoda Schurova doplnku - %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% VSTUP
% A_s       % ridka cast A
% A_d       % husta cast A
% b_s       % ridka cast prave strany b
% b_d       % husta cast prave strany b

%% VYSTUP
% x             % reseni

%%
function [x] = schur_method (A_s, A_d, b_s, b_d)

% inicializace
C_s = A_s'*A_s;
L_s = chol(C_s,'lower');
[m_d,n_d] = size(A_d);

% Vyres L_s*B_d' = -A_d' pro B_d'
B_d_trans = -1*inv(L_s)*A_d';
B_d = B_d_trans';                   % B_d = husta matice
% Schuruv doplnek
I_md = eye(m_d);                    % S_d = husta, SPD matice
S_d = I_md + (B_d*B_d');

% Vyres -L_s * y_s = -A_s' * b_s pro y_s.
y_s = inv(L_s)*A_s' * b_s;
% Vyres -B_d*y_s + S_d*r_d = b_d pro r_d.
L = chol(S_d, 'lower');
z = inv(L)*(b_d + B_d*y_s);
r_d = inv(L')*z;

% Vyres L_s' * x = y_s - (B_d' * r_d) pro x.
x = inv(L_s')*(y_s - (B_d'*r_d));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% - Metoda Schurova doplnku - %%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%% - s regularizaci - %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% VSTUP
% A_s        % ridka cast A
% A_d        % husta cast A
% b_s        % ridka cast prave strany b
% b_d        % husta cast prave strany b
% x0         % pocatecni aproximace
% max_iter   % maximalni pocet iteraci
             % zastavovaci kriterium
% tol        % tolerance pro podil norem rnorm0 / rnorm
             % zastavovaci kriterium
% alpha      % regularizacni parameter
% restart    % pocet iteraci pro restart GMRES metody

%% VYSTUP
% x              % reseni
% iter_gmres     % pocet iteraci GMRES metody
% resvec_gmres   % vektor norem rezidua v kazde iteraci
% flag_gmres     % indikator prubehu GMRES metody

%%
function [x, iter_gmres, resvec_gmres, flag_gmres] = ...
    schur_method_nullclms (A_s, A_d, b_s, b_d, tol, alpha, ...
    max_iter, restart)

% inicializace
[m_s, n_s] = size(A_s);
[m_d,n_d] = size(A_d);
I_n = eye(n_s);
I_md = eye(m_d);
C_s = A_s'*A_s;

K = [-C_s A_d'; A_d I_md];
b = [-A_s'*b_s; b_d];

L_s = chol((A_s'*A_s + alpha*I_n),'lower');
B_d_trans = -inv(L_s)*A_d';
B_d = B_d_trans';
S_d = I_md + (B_d*B_d');

N1 = zeros(n_s, m_d);
M1 = [L_s N1; B_d I_md];
N2 = zeros(m_d,n_s);
M2 = [-I_n N1; N2 S_d];
M3 = [L_s' B_d'; N2 I_md];
M = M1 * M2 * M3;

% Vyres K*inv(M)*w = b pro w pomoci GMRES.
[w_gmres,flag_gmres,relres_gmres,iter_gmres,resvec_gmres] = ...
    gmres(K*inv(M),b,restart,tol,max_iter);

% Vyres M*(x r_d) = (w_s w_d) pro (x r_d).
% ekvivalentni s: Vyres y = (y_s y_d) = inv(M)*z
```

```
55 z_s = w_gmres(1:n_s,1);
56 z_d = w_gmres(n_s+1:n_s+m_d,1);
57
58 % Vyres L_s*u_s = -z_s pro u_s.
59 u_s = -inv(L_s)*z_s;
60 % Vyres u_d = z_d + B_d*u_s.
61 u_d = z_d + (B_d*u_s);
62 % S pomoci Choleskeho faktoru matice S_d vyres S_d*y_d = u_d ...
       pro y_d.
63 L = chol(S_d, 'lower');
64 y_pom = inv(L)*u_d;
65 y_d = inv(L')*y_pom;
66 % Zformuluj u_s = u_s - B_d'*y_d.
67 u_s = u_s - (B_d'*y_d);
68 % Vyres L_s'*y_s = u_s pro y_s.
69 y_s = inv(L_s')*u_s;
70
71
72 x = y_s;
73 r_d = y_d;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%% - Metoda kombinace castecnych reseni - %%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% VSTUP

% A_s       % ridka cast matice A
% A_d       % husta cast matice A
% b_s       % ridka cast prave strany b
% b_d       % husta cast prave strany b
% n_nulls   % pocet nulovych sloupcu A_s
% max_iter  % maximalni pocet iteraci
            % zastavovaci kriterium
% tol       % tolerance pro podil norem rnorm0 / rnorm
            % zastavovaci kriterium

%% VYSTUP

% x         % reseni
% iter      % pocet iteraci

%%
function [x, iter] = comb_part_sols (A_s, A_d, b_s, b_d, ...
    n_nulls, max_iter, tol)

% inicializace
[m_s, n_s] = size(A_s);
[m_d, n_d] = size(A_d);

A_s1 = A_s;
A_d1 = A_d(:, 1:(n_d-n_nulls));
A_d2 = A_d(:, (n_d-n_nulls+1): n_d);

A_1 = [A_s1; A_d1];
[m_1, n_1] = size(A_1);

A_2 = [zeros([m_s n_nulls]); A_d2];
[m_2, n_2] = size(A_2);

b = [b_s; b_d];


% Vyres min|| A_1*z - b ||_2 pro z pomoci PCGLS.
z0 = zeros(n_1, 1);
facts = 2;
alpha = 0;
[z, iter_z, resvec_z] = precond_CGLS_ST (A_s1, A_d1, b_s, ...
    b_d, z0, max_iter, tol, facts, alpha);

% Vyres min|| A_1*W - A_2 ||_F pro W pomoci PCGLS.
w0 = zeros(n_1, 1);
facts = 2;
alpha = 0;
iter_W_sum = 0;
for i = 1 : n_2
    [w, iter_w, resvec_w] = precond_CGLS_ST (A_s1, A_d1, ...
        A_2(1:n_s,i), A_2(m_s+1:m_2,i), w0, max_iter, tol, ...
```

```
          facts, alpha);
55      W(:,i) = w;
56      iter_W_sum = iter_W_sum + iter_w;
57  end
58
59  % Zformuluj x kombinaci castecnych reseni.
60  x_2 = inv(A_2'*(A_2 - (A_1*W)))*A_2'*(b - (A_1*z));
61  x_1 = z - (W*x_2);
62
63  x = [x_1; x_2];
64  iter = iter_z + iter_W_sum;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% - Numericky experiment 1 - %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
clear;
clc;

%% Nacteni matic a pravych stran.

FileName    = 'matrix_org';
FolderName  = 'E:\diplomova_prace\matlab\matice\exp01\';
File        = fullfile(FolderName, FileName);
load(File);
FileRead    = mmread(File);
A_sd        = spconvert(FileRead);

FileName    = 'rhs_org';
File        = fullfile(FolderName, FileName);
FileRead    = fopen(File);
b_sd        = fscanf(FileRead,'%f');

%% Skalovani.

M = max(A_sd(:));
A_sd(:) = A_sd(:)/M;
b_sd(:) = b_sd/M;

%% inicializace

m_art = 10;
n_nulls = 1;

[m_sd,n_sd] = size(A_sd);

A_org = A_sd(1 : (m_sd - m_art), :);  % puvodni matice A
b_org = b_sd(1 : (m_sd - m_art), :);  % puvodni prava strana b
[m_org,n_org] = size(A_org);

A_s = A_org;        % ridka cast matice A
b_s = b_org;        % ridka cast prave strany b
[m_s,n_s] = size(A_s);
A_d = A_sd((m_sd-m_art+1) : m_sd, :); % husta cast matice A
b_d = b_sd( (m_sd-m_art+1) : m_sd, :);% husta cast prave ...
    strany b
[m_d,n_d] = size(A_d);

A_srd = A_s;        % A_s obsahujici nulove sloupce
for i = 1 : m_s
    for j = (n_s - n_nulls + 1) : n_s
        A_srd(i,j) = 0;
    end
end
[m_srd,n_srd] = size(A_srd);

%% Matlab regularizace

```

```
C = A_srd'*A_srd;
D = diag(diag(C));
format shortG;
E = norm(eye(size(D))-D)
alpha_mat = max(sum(abs(C),2)./diag(C))-2

%% Reseni Ax ¬ b pomoci metody Aktualizace.

alpha0 = 0;           % pokud A_s regularni
alpha_reg = 1e-3;     % pocatecni regul. parametr alpha_reg
nreg = 2;             % pocet for-cyklu regularizaci
alpha_desc = 1e-2;    % pokles dalsiho alpha_reg

problem = 'Reseni Ax ¬ b pomoci metody Aktualizace.'
for i = 1:nreg+1
% set parameter alpha
    if (i == 1)
        alpha = alpha0;
    elseif (i == 2)
        alpha = alpha_reg;
    else
        alpha = alpha*alpha_desc;
    end
% Aktualizace
x0 = zeros(n_sd, 1);     % pocatecni aproximace
max_iter = 200;          % maximalni pocet iteraci
tol = 1e-8;              % tolerance zastavovaciho kriteria
alphas(i,1) = alpha;
facts = 1;               % parametr volby predpodmineni

tic;
if (i == 1)
    [x_up, iter_up, resvec_xs] = updating (A_s, A_d, b_s, ...
       b_d, x0, max_iter, tol, facts, alpha);
else
    [x_up, iter_up, resvec_xs] = updating (A_srd, A_d, b_s, ...
       b_d, x0, max_iter, tol, facts, alpha);
end
resvec_up = resvec_xs;
resvec_up(iter_up+1,1) = norm(b_sd - (A_sd*x_up));
time = toc;

format shortG;
results = 'alpha, norm_x, norm_r, n_iter, time'
values = [alpha,norm(x_up),resvec_up(iter_up+1,1),iter_up,time]
end

%% Reseni Ax ¬ b pomoci PCGLS.

alpha0 = 0;           % pokud A_s regularni
alpha_reg = 1e-3;     % pocatecni regul. parametr alpha_reg
nreg = 2;             % pocet for-cyklu regularizaci
alpha_desc = 1e-2;    % pokles dalsiho alpha_reg

problem = 'Reseni Ax ¬ b pomoci PCGLS.'

for i = 1:nreg+1
% set parameter alpha
```

```
113     if (i == 1)
114         alpha = alpha0;
115     elseif (i == 2)
116         alpha = alpha_reg;
117     else
118         alpha = alpha*alpha_desc;
119     end
120 % PCGLS
121 x0 = zeros(n_sd, 1);     % pocatecni aproximace
122 max_iter = 200;          % maximalni pocet iteraci
123 tol = 1e-8;              % tolerance zastavovaciho kriteria
124 facts = 1;               % parametr volby predpodmineni
125 alphas(i,1) = alpha;
126
127 tic;
128 if (i == 1)
129     [x_reg, iter_reg, resvec_reg] = precond_CGLS_ST (A_s, ...
            A_d, b_s, b_d, x0, max_iter, tol, facts, alpha);
130 else
131     [x_reg, iter_reg, resvec_reg] = precond_CGLS_ST (A_srd, ...
            A_d, b_s, b_d, x0, max_iter, tol, facts, alpha);
132 end
133 time = toc;
134
135 format shortG;
136 results = 'alpha, norm_x, norm_r, n_iter, time'
137 values = [alpha,norm(x_reg),resvec_reg(iter_reg,1),iter_reg,time]
138 end
139
140 %% Reseni Ax ¬ b pomoci metody Schurova doplnku.
141
142 alpha0 = 0;              % pokud A_s regularni
143 alpha_reg = 1e-3;        % pocatecni regul. parametr alpha_reg
144 nreg = 2;                % pocet for-cyklu regularizaci
145 alpha_desc = 1e-2;       % pokles dalsiho alpha_reg
146
147 problem = 'Reseni Ax ¬ b pomoci metody Schurova doplnku s GMRES.'
148
149 for i = 1:nreg+1
150 % set parameter alpha
151     if (i == 1)
152         alpha = alpha0;
153     elseif (i == 2)
154         alpha = alpha_reg;
155     else
156         alpha = alpha*alpha_desc;
157     end
158 % metoda Schurova doplnku s GMRES
159 x0 = zeros(n_sd, 1);     % pocatecni aproximace
160 max_iter = 200;          % maximalni pocet iteraci
161 restart = 50;            % pocet iteraci pro restart GMRES
162 tol = 1e-8;              % tolerance zastavovaciho kriteria
163 alphas(i,1) = alpha;
164
165 tic;
166 if (i == 1)
167     [x_schur] = schur_method (A_s, A_d, b_s, b_d);
168     resvec_gmres = zeros(2,1);
```

```
169     iter_gmres = [1,1];
170 else
171     [x_schur, iter_gmres, resvec_gmres, flag_gmres] =
172         schur_method_nullclms (A_srd, A_d, b_s, b_d, tol, ...
            alpha, max_iter, restart);
173 end
174 time = toc;
175
176 format shortG;
177 results = 'alpha, norm_x, norm_r, n_iter, time'
178 iter_gmres(2) = ((iter_gmres(1)-1)*restart)+iter_gmres(2)+1;
179 values = [alpha,norm(x_schur),resvec_gmres(iter_gmres(2),1), ...
    iter_gmres(1),iter_gmres(2),time]
180 end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% - Numericky experiment 2 - %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
clear;
clc;

%% Inicializace ulohy.

matrix = 'lpagg'
m_art = 10
m_stretch = 3
n_nulls = 1

%% Nacteni matic a pravych stran.

FolderPath = 'E:\diplomka\matlab\matrices\for_exp\';
FolderName = [FolderPath matrix '_' num2str(m_art)]

FileName    = 'matrix_org';
File        = fullfile(FolderName, FileName);
load(File);
FileRead    = mmread(File);
A_sd        = spconvert(FileRead);

FileName    = 'rhs_org';
File        = fullfile(FolderName, FileName);
FileRead    = fopen(File);
b_sd      = fscanf(FileRead,'%f');

FolderName = [FolderPath matrix '_' num2str(m_stretch)]

FileName    = 'matrix_stretched';
File        = fullfile(FolderName, FileName);
load(File);
FileRead    = mmread(File);
A_stretch   = spconvert(FileRead);

FileName    = 'rhs_stretched';
File        = fullfile(FolderName, FileName);
FileRead    = fopen(File);
b_stretch   = fscanf(FileRead,'%f');

%% Inicializace.

[m_sd,n_sd] = size(A_sd);
A_org = A_sd(1:(m_sd - m_art), :);    % puvodni matice A
b_org = b_sd(1:(m_sd - m_art), :);    % puvodni prava strana b
[m_org,n_org] = size(A_org)

A_nulls = A_org;                      % A s nulovymi sloupci
for i = 1 : m_org
    for j = (n_org - n_nulls + 1) : n_org
        A_nulls(i,j) = 0;
    end
end
```

```
58
59 A_s = A_stretch;                % ridka cast po stretchingu
60 for i = 1 : m_org
61     for j = (n_org - n_nulls + 1) : n_org
62         A_s(i,j) = 0;
63     end
64 end
65 b_s = b_stretch;                % ridka prava strana po ...
       stretchingu
66 [m_s,n_s] = size(A_s)
67
68 A_d = A_sd((m_sd-(m_art-m_stretch)+1):m_sd, :);  % husta cast ...
       matice A
69 b_d = b_sd((m_sd-(m_art - m_stretch)+1):m_sd, :);% husta cast ...
       prave strany b
70 [m_d, n_d] = size(A_d)
71 A_d(:,n_d+1:n_s) = 0;
72
73 A_sd = [A_s; A_d];
74 [m_sd,n_sd] = size(A_sd);
75 b_sd = [b_s; b_d];
76
77 alpha = 0;               % regularizacni parametr
78 x0 = zeros(n_sd, 1);     % pocatecni aproximace
79 max_iter = 200;          % maximalni pocet iteraci
80 tol = 1e-8;              % tolerance zastavovaciho kriteria
81 facts = 2;               % parametr volby predpodmineni
82
83 %% Kontrola SPD.
84
85 try ichol(C_s, struct('type','nofill','shape','lower'));
86     disp('Neuplna Chol. fakt. s nul. zapl. pro C_s je v ...
          poradku.')
87 catch ME;
88     disp('Neuplna Chol. fakt. s nul. zapl. pro C_s neni v ...
          poradku.')
89     facts = 2;
90 end
91
92 try ichol(C_s, struct('type','ict','droptol',1e-8,'michol',
93                       'off','diagcomp',0,'shape','lower'));
94     disp('Neuplna Chol. fakt. s tol. pro C_s je v poradku.')
95 catch ME;
96     disp('Neuplna Chol. fakt. s tol. pro C_s neni v poradku.')
97     alpha = 1e-7;
98     facts = 1;
99 end
100
101 facts
102
103 %% Metoda Aktualizace.
104
105 problem = 'Reseni Ax ¬ b pomoci metody Aktualizace.'
106
107 tic;
108 [x_up, iter_up, resvec_up] = updating (A_s, A_d, b_s, b_d, ...
       x0, max_iter, tol, facts, alpha);
109 time_up = toc;
```

```
110
111 format shortG;
112 results = 'norm_x, norm_r, n_iter, time'
113 values = [norm(x_up), resvec_up(iter_up,1), iter_up, time_up]
114
115 %% PCGLS.
116
117 problem = 'Reseni Ax ¬ b pomoci PCGLS.'
118
119 tic;
120 [x_cg, iter_cg, resvec_cg] = precond_CGLS_ST (A_s, A_d, b_s, ...
       b_d, x0, max_iter, tol, facts, alpha);
121 time_cg = toc;
122
123 format shortG;
124 results = 'norm_x, norm_r, n_iter, time'
125 values = [norm(x_cg), resvec_cg(iter_cg,1), iter_cg, time_cg]
126
127 %% Metoda Schurova doplnku.
128
129 problem = 'Reseni Ax ¬ b pomoci metody Schurova doplnku.'
130
131 tic;
132 [x_schur] = schur_method (A_s, A_d, b_s, b_d);
133 time_schur = toc;
134
135 format shortG;
136 results = 'norm_x, norm_r, time'
137 values = [norm(x_schur), norm((A_sd*x_schur) - b_sd),time_schur]
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%% - Numericky experiment 3 - %%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%
clear;
clc;

%% Inicializace ulohy.

matrix = 'lpagg'
m_art = 10
m_stretch = 3
n_nulls = 1

%% Nacteni matic a pravych stran.

FolderPath = 'E:\diplomka\matlab\matrices\for_exp\';
FolderName = [FolderPath matrix '_' num2str(m_art)]

FileName    = 'matrix_org';
File        = fullfile(FolderName, FileName);
load(File);
FileRead    = mmread(File);
A_sd        = spconvert(FileRead);

FileName    = 'rhs_org';
File        = fullfile(FolderName, FileName);
FileRead    = fopen(File);
b_sd      = fscanf(FileRead,'%f');

FolderName = [FolderPath matrix '_' num2str(m_stretch)]

FileName    = 'matrix_stretched';
File        = fullfile(FolderName, FileName);
load(File);
FileRead    = mmread(File);
A_stretch   = spconvert(FileRead);

FileName    = 'rhs_stretched';
File        = fullfile(FolderName, FileName);
FileRead    = fopen(File);
b_stretch   = fscanf(FileRead,'%f');

%% Inicializace.

[m_sd,n_sd] = size(A_sd);
A_org = A_sd(1:(m_sd - m_art), :);    % puvodni matice A
b_org = b_sd(1:(m_sd - m_art), :);    % puvodni prava strana b
[m_org,n_org] = size(A_org)

A_s_nulls = A_org;                    % ridka cast A s nulovymi ...
    sloupci
for i = 1 : m_org
    for j = (n_org - n_nulls + 1) : n_org
        A_s_nulls(i,j) = 0;
    end
```

```
end
[m_s_nulls,n_s_nulls] = size(A_s_nulls)

A_s = A_org(:, 1:(n_org-n_nulls));  % ridka cast A
b_s = b_org;                        % ridka cast b
[m_s,n_s] = size(A_s)

A_s_stretch = A_stretch;            % ridka cast A po ... stretchingu s nulovymi sloupci
for i = 1 : m_org
    for j = (n_org - n_nulls + 1) : n_org
        A_s_stretch(i,j) = 0;
    end
end
b_s_stretch = b_stretch;            % ridka cast b po stretchingu
[m_s_stretch,n_s_stretch] = size(A_stretch)

A_d_full = A_sd((m_sd-m_art+1) : m_sd, :);   % husta cast A
b_d_full = b_sd((m_sd-m_art+1) : m_sd, :);   % husta cast b
[m_d_full,n_d_full] = size(A_d_full);

A_d = A_sd((m_sd-(m_art-m_stretch)+1):m_sd, :);  % husta cast ... A po stretchingu
b_d = b_sd((m_sd-(m_art-m_stretch)+1):m_sd, :);  % husta cast ... b po stretchingu
[m_d, n_d] = size(A_d)

A_d_nulls = A_d;                            % doplneni A_d nulami
A_d_nulls(:,n_d+1:n_s_stretch) = 0;

max_iter = 200;     % maximalni pocet iteraci
tol = 1e-8;         % tolerance zastavovaciho kriteria

%% Metoda Schurova doplnku.

problem = 'Reseni Ax ¬ b pomoci metody Schurova doplnku s ... regularizaci.'
alpha = 1e-3;
restart = 50;

tic;
[x_schur, iter_schur, resvec_schur, flag_gmres] = ...
    schur_method_nullclms (A_s_nulls, A_d_full, b_s, b_d_full, ...
    tol, alpha, max_iter, restart);
time_schur = toc;

format shortG;
results = 'norm_x, n_iter, time, time_per_iters'
values = [norm(x_schur), iter_schur, time_schur, ...
    time_schur/iter_schur(2)]

%% PCGLS.

problem = 'Reseni Ax ¬ b pomoci PCGLS s castecnym stretchingem.'
alpha = 0;      % alpha = 1e-3; pokud je potreba regularizovat
x0 = zeros(n_s_stretch, 1); % pocatecni aproximace
facts = 2;      % parametr volby predpodmineni

```

```
108 tic;
109 [x_pcgls, iter_pcgls, resvec_pcgls] = precond_CGLS_ST ...
       (A_s_stretch, A_d_nulls, b_s_stretch, b_d, x0, max_iter, ...
       tol, facts, alpha);
110 time_pcgls = toc;
111
112 format shortG;
113 results = 'norm_x, n_iter, time, time_per_iters'
114 values = [norm(x_pcgls), iter_pcgls, time_pcgls, ...
       time_pcgls/iter_pcgls]
115
116 %% Kombinace castecnych reseni.
117
118 problem = 'Reseni Ax ¬ b pomoci kombinace castecnych reseni.'
119
120 tic;
121 [x_part_sols, iter_part_sols] = comb_part_sols (A_s, ...
       A_d_full, b_s, b_d_full, n_nulls, max_iter, tol);
122 time_part_sols = toc;
123
124 format shortG;
125 results = 'norm_x, n_iter, time, time_per_iters'
126 values = [norm(x_part_sols),iter_part_sols,time_part_sols, ...
       time_part_sols/iter_part_sols]
```