

UNIVERZA V LJUBLJANI  
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Matej Klančar

**Algoritmi za reševanje problema  
matričnih napolnitev**

DIPLOMSKO DELO

UNIVERZITETNI ŠTUDIJSKI PROGRAM  
PRVE STOPNJE  
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Aljaž Zalar

Ljubljana, 2023

To delo je ponujeno pod licenco *Creative Commons Priznanje avtorstva-Deljenje pod enakimi pogoji 2.5 Slovenija* (ali novejšo različico). To pomeni, da se tako besedilo, slike, grafi in druge sestavnine dela kot tudi rezultati diplomskega dela lahko prosto distribuirajo, reproducirajo, uporabljajo, priobčujejo javnosti in predelujejo, pod pogojem, da se jasno in vidno navede avtorja in naslov tega dela in da se v primeru spremembe, preoblikovanja ali uporabe tega dela v svojem delu, lahko distribuira predelava le pod licenco, ki je enaka tej. Podrobnosti licence so dostopne na spletni strani creativecommons.si ali na Inštitutu za intelektualno lastnino, Strelška 1, 1000 Ljubljana.



Izvorna koda diplomskega dela, njeni rezultati in v ta namen razvita programska oprema je ponujena pod licenco GNU General Public License, različica 3 (ali novejša). To pomeni, da se lahko prosto distribuira in/ali predeluje pod njenimi pogoji. Podrobnosti licence so dostopne na spletni strani <http://www.gnu.org/licenses/>.

*Besedilo je oblikovano z urejevalnikom besedil L<sup>A</sup>T<sub>E</sub>X.*

**Kandidat:** Matej Klančar

**Naslov:** Naslov diplomskega dela

**Vrsta naloge:** Diplomska naloga na univerzitetnem programu prve stopnje  
Računalništvo in informatika

**Mentor:** doc. dr. Aljaž Zalar

**Opis:**

Besedilo teme diplomskega dela študent prepiše iz študijskega informacijskega sistema, kamor ga je vnesel mentor. V nekaj stawkih bo opisal, kaj pričakuje od kandidatovega diplomskega dela. Kaj so cilji, kakšne metode naj uporabi, morda bo zapisal tudi ključno literaturo.

**Title:** Algorithms for solving matrix completion problem

**Description:**

opis diplome v angleščini



*Na tem mestu zapišite, komu se zahvaljujete za pomoč pri izdelavi diplomske naloge oziroma pri vašem študiju nasploh. Pazite, da ne boste koga pozabili. Utegnil vam bo zameriti. Temu se da izogniti tako, da celotno zahvalo izpustite.*



# Kazalo

## Povzetek

## Abstract

<b>1</b>	<b>Uvod</b>	<b>1</b>
1.1	Motivacija . . . . .	1
1.2	Cilji . . . . .	2
1.3	Struktura diplomskega dela . . . . .	2
<b>2</b>	<b>Pregled področja</b>	<b>3</b>
<b>3</b>	<b>Algoritmi</b>	<b>5</b>
3.1	Definicije in oznake . . . . .	5
3.2	Algoritem minimizacije nuklearne norme . . . . .	7
3.3	Algoritem praga singularnih vrednosti . . . . .	9
3.4	Algoritem minimizacije prirezane nuklearne norme . . . . .	13
3.5	Izmenjajoč gradientni spust . . . . .	18
3.6	LMaFit . . . . .	20
<b>4</b>	<b>Rezultati</b>	<b>23</b>
4.1	Velika črno-bela slika . . . . .	24
4.2	Vpliv kompleksnosti slik na rekonstrukcijo . . . . .	28
4.3	Rekonstrukcija barvnih slik . . . . .	34
4.4	Vpliv podatka o rangu na rezultate . . . . .	36
4.5	Rekonstrukcija slike z besedilom . . . . .	38

4.6	Primerjava rezultatov z algoritmom za reševanje Poissonovih enačb . . . . .	40
4.7	Povzetek ugotovitev . . . . .	44
<b>5</b>	<b>Zaključek</b>	<b>47</b>

# Seznam uporabljenih kratic

kratica	angleško	slovensko
<b>SVT</b>	Singular Value Thresholding	Prag singularnih vrednosti
<b>NNM</b>	Nuclear norm minimization	Minimizacija nuklearne norme
<b>TNNM</b>	Truncated nuclear norm minimization	Minimizacija prirezane nuklearne norme
<b>ASD</b>	Alternating Steepest Descent	Izmenjajoč gradientni spust
<b>NP</b>	Nondeterministic polynomial time	Nedeterministični polinomni čas
<b>SDP</b>	Semidefinite programming	Semidefinitno programiranje
<b>OZP</b>		Odstotek znanih podatkov



# Povzetek

**Naslov:** Algoritmi za reševanje problema matričnih napolnitev

**Avtor:** Matej Klančar

V vzorcu je predstavljen postopek priprave diplomskega dela z uporabo okolja L<sup>A</sup>T<sub>E</sub>X. Vaš povzetek mora sicer vsebovati približno 100 besed, ta tukaj je odločno prekratek. Dober povzetek vključuje: (1) kratek opis obravnavanega problema, (2) kratek opis vašega pristopa za reševanje tega problema in (3) (najbolj uspešen) rezultat ali prispevek diplomske naloge.

**Ključne besede:** računalnik, računalnik, računalnik.



# Abstract

**Title:** Diploma thesis template

**Author:** Matej Klančar

This sample document presents an approach to typesetting your BSc thesis using L<sup>A</sup>T<sub>E</sub>X. A proper abstract should contain around 100 words which makes this one way too short.

**Keywords:** computer, computer, computer.



# Poglavlje 1

## Uvod

### 1.1 Motivacija

Problem matričnih napolnitev sprejme matriko, največkat označeno z  $M$ , pri kateri so nekateri elementi označeni kot neznani. Problem nato sprašuje po vrednostih, ki jih lahko vstavimo v neznane vrednosti, tako da bo rang matrike najmanjši možen. Gre za NP-poln problem, zato ga poskušamo posostaviti, ter reševati lažje probleme, ki vrnejo dovolj dobre, a ne optimalne rešitve.

Problem je v zadnjih letih postal zelo popularen, z njim pa se ukvarjajo tako številni matematiki kot računalničarji. Zaradi splošnosti problema so metode za reševanje uporabne na številnih področjih. Literatura navaja številne načine uporabe [10], saj lahko algoritmom uporabljam v priporočilnih sistemih, kjer generiramo predvidene ocene uporabnikov kot tudi programih za računanje razdalj med napravami.

Algoritme testiramo in primerjamo na problemih razreševanja neznanih pikslov v slikah. Videli bomo, da poznamo boljše algoritme, ki rešujejo ta problem. Ker pa potrebujemo za izračun napake nezašumljeno matriko, je ta vrsta podatkov primerna. Tudi literatura za namene testiranja uporablja slike ali pa naključno generirane podatke. Ker pa lahko slike tudi vizualno opazujemo, se v diplomskem delu osredotočamo na te.

V diplomi bomo predstavili par algoritmov, ki rešujejo omenjen problem ter pokazali in razložili njihove ideje. Algoritmi so bili izbrani glede na njihovo popularnost in priznanost v literaturi. Prav tako poskrbimo, da so algoritmi primerno različni in temeljijo na drugačnih principih.

## 1.2 Cilji

Cilj diplomske naloge je predstaviti uporabnost algoritmov, ter razložiti zakaj delujejo. Ideje različnih algoritmov ter njihove pristope do problema poskušamo kar se da jasno predstaviti in interpretirati.

Medtem ko obstaja veliko člankov o samih metodah reševanja, je literatura, ki različne metode primerja in odgovarja na razlike med njimi maloštevilna. To delo zato poskuša analizirati rezultate algoritmov na različnih realnih problemih, ter ugotoviti, kako se rezultati algoritmov v različnih primerih razlikujejo. Ponovno poskušamo rezultate tudi interpretirati in povezati s samo implementacijo algoritma.

Glavni prispevki te diplomske naloge so implementacija vseh omenjenih algoritmov, testiranje algoritmov na različnih primerih ter odgovori na vprašanja o uporabnosti metod, ki se nam med implementacijo porodijo. Podamo tudi vizualen prikaz rekonstruiranih slik, kot tudi grafičen prikaz napak in časov do konvergencije programov.

## 1.3 Struktura diplomskega dela

Poglavlje 2 pregleda že napisana dela o samem reševanju matričnih napolnitve in v kratkem predstavi njihove ugotovitve. Poglavlje 3 predstavi algoritme in njihove ideje. V poglavju 4 predstavimo rezultate algoritmov na zašumljenih slikah. Poglavlje je razdeljeno na razdelke, v katerih poskušamo odgovoriti na različna vprašanja. V poglavju 5 diplomsko delo strnemo in podamo ideje, kako bi lahko delo nadaljevali.

# Poglavlje 2

## Pregled področja

Področje matričnih napolnitev je trenutno zelo aktivno, s številnimi raziskovalci, ki na danem področju raziskujejo in iščejo nove načine reševanja problema. Algoritmi, ki problem rešujejo, so zaradi splošnosti problema zelo uporabni, kar pojasnjuje motivacijo po iskanju učinkovitih algoritmov.

Eno prvih del, ki sam problem opisuje je [9]. Delo opisuje sam problem in njegovo naravo, vendar se ne osredotoča zgolj na napolnjevanje s ciljem minimalnega ranga. Delo opisuje tudi ideje za napolnitve matrik tako, da je napolnjena matrika pozitivno semidefinitna, kot tudi maksimizacijo determinante napolnjene matrike.

Doktorska disertacija [6] podaja pomembne opise pretvorbe problema minimizacije ranga v problem iz področja semidefinitnega programiranja. Ideje in dokazi tega dela služijo kot temelj za algoritme, ki so se razvili v zadnjih letih.

Članki [2, 8, 12, 13] opisujejo določene algoritme, ter predstavijo nadgradnjo del in idej pred njimi. Glavne ugotovitve člankov so v tem delu povzete v njihovih razdelkih poglavja 3. Ta dela so bila ključna za samo implementacijo algoritmov.

Vodilna literatura tekom pisanja diplomske naloge je bil članek [10], ki opisuje problem, ter mnoge algoritme. Medtem ko opisi pogosto niso bili dovolj podrobni, da bi lahko začel algoritme implementirati, je članek ponujal

dobro razumljive opise algoritmov, kot tudi navedel vire, ki so pomagali pri implementaciji. Prav tako je članek podal pomembno primerjavo rezultatov različnih algoritmov.

# Poglavlje 3

## Algoritmi

V tem poglavju predstavimo teoretično ozadje algoritmov za reševanje problema matričnih napolnitev. V posameznih razdelkih predstavimo različne algoritme, povzamemo idejo algoritma za iskanje rešitve, ter razložimo, zakaj deluje. Vrstni red predstavitev algoritmov je postavljen tako, da kjer algoritom razširja idejo prejšnjega, tega predstavimo kasneje.

### 3.1 Definicije in oznake

V tem razdelku uvedemo definicije in oznake, ki se bodo pojavljale v preostanku dela.

1. Z  $\mathbb{R}^{n_1 \times n_2}$  označujemo množico  $n_1 \times n_2$  realnih matrik. Matriko  $A \in \mathbb{R}^{n_1 \times n_2}$  pišemo kot  $A = [a_{ij}]_{i,j}$ .
2.  $\Omega$  je podmnožica množice vseh urejenih parov  $\{(i, j) : i = 1, \dots, n_1, j = 1, \dots, n_2\}$ . V delu je imenujemo **množica znanih vrednosti**.
3. Z  $\mathcal{P}_\Omega : \mathbb{R}^{n_1 \times n_2} \rightarrow \mathbb{R}^{n_1 \times n_2}$  označimo preslikavo, definirano kot

$$[\mathcal{P}_\Omega(A)]_{i,j} = \begin{cases} a_{ij}, & (i, j) \in \Omega, \\ 0, & \text{sicer.} \end{cases}$$

Z besedami, preslikava  $\mathcal{P}_\Omega$  ohrani vrednosti matrike na mestih iz množice  $\Omega$ , ostale pa postavi na 0.

4. Naj bo  $\tau > 0$  pozitivno realno število. **Operator praga** ([2])  $\mathcal{D}_\tau$  :  $\mathbb{R}^{n_1 \times n_2} \rightarrow \mathbb{R}^{n_1 \times n_2}$  je definiran kot

$$\begin{aligned}\mathcal{D}_\tau(A) &:= U\mathcal{D}_\tau(\Sigma)V^T, \\ \mathcal{D}_\tau(\Sigma) &= \text{diag} \left( \max(\sigma_1 - \tau, 0), \max(\sigma_2 - \tau, 0), \dots, \right. \\ &\quad \left. \max(\sigma_{\min(n_1, n_2)} - \tau, 0) \right),\end{aligned}\tag{3.1}$$

kjer je  $A = U\Sigma V^T$  singularni razcep matrike  $A$ , pri čemer sta  $U \in \mathbb{R}^{n_1 \times n_1}$  in  $V \in \mathbb{R}^{n_2 \times n_2}$  ortogonalni matriki,

$$\Sigma = \text{diag}(\sigma_1, \sigma_2, \dots, \sigma_{\min(n_1, n_2)}) \in \mathbb{R}^{n_1 \times n_2}$$

pa je diagonalna matrika singularnih vrednosti  $\sigma_i$  matrike  $A$ .

5. Matriko  $M \in \mathbb{R}^{n_1 \times n_2}$ , ki ima določene samo nekatere elemente, imenujemo **delno določena matrika**.
6. **Sled** kvadratne matrike  $A = [a_{ij}]_{i,j} \in \mathbb{R}^{n \times n}$  je vsota njenih diagonalnih elementov oz. s formulo

$$\text{Tr}(A) = \sum_{i=1}^n a_{ii}.$$

7. Skalarni produkt  $\langle \cdot, \cdot \rangle$  na vektorskem prostoru pravokotnih  $n_1 \times n_2$  matrik je definiran kot

$$\langle A, B \rangle = \text{Tr}(AB^T).$$

8. **Frobeniusova norma** matrike  $A \in \mathbb{R}^{n_1 \times n_2}$  izhaja iz zgornjega skalarnega produkta in je enaka

$$\|A\|_F = \sqrt{\langle A, A \rangle} = \sqrt{\sum_{i=1}^{n_1} \sum_{j=1}^{n_2} a_{ij}^2}.$$

9. **Nuklearna norma** matrike  $A \in \mathbb{R}^{n_1 \times n_2}$  je definirana kot

$$\|A\|_* = \sum_{i=1}^n \sigma_i(A),$$

pri čemer  $\sigma_i(A)$  označuje  $i$ -to največjo singularno vrednost matrike  $A$ . Premisliti, zakaj je to res norma, tj. zakaj zadošča trikotniški neenakosti. To ni čisto očitno in bi lahko kdo v komisiji vprašal. Če je dokaz kratek, lahko dodaš.

10. Matrika  $A \in \mathbb{R}^{n \times n}$  je **pozitivno semidefinitna**, če velja  $x^T A x \geq 0$  za vsak  $x \in \mathbb{R}^n$ . To lastnost matrike  $A$  označimo z  $A \succeq 0$ . Spomnimo se še, da je  $A \succeq 0$  ekvivalentno dejstvu, da so vse lastne vrednosti matrike  $A$  nenegativne.
11. Vektor  $c \in \mathbb{R}^n$  je **subgradient** konveksne funkcije  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  v točki  $x_0$ , če velja

$$\forall x \in \mathbb{R}^n : f(x) - f(x_0) \geq c^T(x - x_0).$$

Z besedami:  $c$  je subgradient funkcije  $f$  v točki  $x_0$ , če premica skozi točko  $(x_0, f(x_0))$  s smernim koeficientom  $c$  leži pod grafom funkcije  $f$  (in se ga dotika v točki  $(x_0, f(x_0))$ ).

12. Z oznako  $\arg \min_x f(x)$  označimo tisto točko iz domene funkcije  $f$ , za katero je vrednost funkcije najmanjša možna.

## 3.2 Algoritem minimizacije nuklearne norme

Ker je minimizacija ranga matrike NP-poln problem [6], algoritmi matričnih napolnitev temeljijo na metodah, ki rešujejo sorodne, hkrati pa bistveno enostavnejše probleme. **Algoritem minimizacije nuklearne norme (NNM)** uporabi idejo, da je rang matrike tesno povezan z njeno nuklearno normo. Izkaže se [6], da je nuklearna norma konveksna ovojnica funkcije ranga. **Konveksna ovojnica funkcije**  $f : \mathbb{R}^{n_1 \times n_2} \rightarrow \mathbb{R}$  je definirana kot

$$\tilde{f}(A) = \sup\{g(A) \mid g : \mathbb{R}^{n_1 \times n_2} \rightarrow \mathbb{R} \text{ je konveksna funkcija in } \forall A \in \mathbb{R}^{n_1 \times n_2} : g(A) \leq f(A)\}.$$

poglej razliko convex hull in lower envelope

Problem minimizacije nuklearne norme je možno pretvoriti v optimizacijski problem s področja **semidefinitnega programiranja (SDP)**, ki ga lahko rešujemo z učinkovitimi orodji, npr. knjižnico SeDuMi [11] v Matlabu.

Naj bodo  $C, A_1, \dots, A_\ell \in \mathbb{R}^{n \times n}$  dane matrike in  $b_1, \dots, b_\ell \in \mathbb{R}$  dana števila. **Standardna oblika semidefinitnega programa** je

$$\begin{aligned} & \min_{Y \in \mathbb{R}^{n \times n}} \quad \langle C, Y \rangle, \\ & \text{pri pogojih} \quad \langle A_k, Y \rangle = b_k, \quad k = 1, \dots, \ell, \\ & \quad Y \succeq 0 \end{aligned} \tag{3.2}$$

Vrnimo se k našemu problemu. Naj bo  $M \in \mathbb{R}^{n_1 \times n_2}$  zašumljena matrika (tj. matrika z nekaterimi neznanimi vhodi),  $\Omega$  pa množica urejenih parov, ki predstavljajo mesta, kjer vrednosti matrike poznamo. Problem minimizacije nuklearne norme lahko zapišemo kot

$$\begin{aligned} & \min_{\substack{X \in \mathbb{R}^{n_1 \times n_2}, \\ t \in \mathbb{R}}} \quad t, \\ & \text{pri pogojih} \quad \|X\|_* \leq t, \\ & \quad \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M), \end{aligned} \tag{3.3}$$

Hitro se da videti [6], da za matriko  $X \in \mathbb{R}^{n_1 \times n_2}$  in  $t \in \mathbb{R}$  velja

$$\begin{aligned} \|X\|_* \leq t & \iff \exists W_1 \in \mathbb{R}^{n_1 \times n_1}, W_2 \in \mathbb{R}^{n_2 \times n_2} : \\ & Y = \begin{bmatrix} W_1 & X \\ X^T & W_2 \end{bmatrix}, \quad Y \succeq 0, \quad \text{Tr}(Y) \leq 2t. \end{aligned} \tag{3.4}$$

Minimizacijski problem (3.3) lahko z uporabo (3.4) zapišemo kot

$$\begin{aligned} & \min_{\substack{Y \in \mathbb{R}^{(n_1+n_2) \times (n_1+n_2)}, \\ t \in \mathbb{R}}} \quad 2t, \\ & \text{pri pogojih} \quad \text{Tr}(Y) \leq 2t, \\ & \quad Y \succeq 0, \\ & \quad \langle Y, A_{ab} \rangle = M_{ab}, \quad (a, b) \in \Omega, \end{aligned} \tag{3.5}$$

kjer so  $A_{ab} \in \mathbb{R}^{(n_1+n_2) \times (n_1+n_2)}$  matrike, ki imajo vse elemente ničelne, razen tistega na mestu  $(a, n_1+b)$ , ki ima vrednost 1. Programi za reševanje SDP-jev pa lahko sprejmejo in rešujejo probleme v obliki (3.5) [10].

### 3.3 Algoritem praga singularnih vrednosti

**Algoritem praga singularnih vrednosti (SVT)** [2], uporabi idejo, da imajo matrike z majhnim rangom nekaj velikih singularnih vrednosti, ostale pa 0 ali pa vsaj blizu 0. Ključna parametra v SVT-ju sta *izbira premika* in *izbira praga*, algoritem pa temelji na iteraciji

$$X^{(k)} = \mathcal{D}_\tau(Y^{(k-1)}), \quad (3.6)$$

$$Y^{(k)} = Y^{(k-1)} + \delta_k \mathcal{P}_\Omega(M - X^{(k)}), \quad (3.7)$$

kjer so  $\tau > 0$  izbran prag,  $\delta_k$  izbran premik,  $X^{(0)} = 0 \in \mathbb{R}^{n_1 \times n_2}$  in  $Y^{(0)} = 0 \in \mathbb{R}^{n_1 \times n_2}$ . [2]

V nadaljevanju bomo opisali glavno idejo zgornje iteracije. V grobem pa temelji na uporabi metode za iskanje vezanih ekstremov, kjer elementi matrik  $Y^{(k)}$  predstavljajo Lagrangove množitelje.

Uvedimo funkcijo

$$f_\tau(X) = \tau \|X\|_* + \frac{1}{2} \|X\|_F^2 \quad (3.8)$$

in optimizacijski problem

$$\begin{aligned} \min_{X \in \mathbb{R}^{n_1 \times n_2}} \quad & f_\tau(X), \\ \text{pri pogojih} \quad & \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M). \end{aligned} \quad (3.9)$$

Opazimo lahko, da za velike vrednosti  $\tau$  velja  $f_\tau(X) \approx \tau \|X\|_*$ , kar pomeni, da bo s primerno izbranim  $\tau$ , optimizacijski problem minimiziral nuklearno normo.

Denimo, da želimo poiskati minimum funkcije  $f(x)$  pri pogojih  $g_1(x) = g_2(x) = \dots = g_k(x) = 0$ . V teoriji vezanih ekstremov se za tovrstne probleme uvede **Lagrangeovo funkcijo**

$$\mathcal{L}(x, \lambda_1, \lambda_2, \dots, \lambda_k) = f(x) + \lambda_1 g_1(x) + \lambda_2 g_2(x) + \dots + \lambda_k g_k(x),$$

nato pa išče ekstreme med njenimi stacionarnimi točkami. Problemu (3.9) lahko priredimo Lagrangeovo funkcijo

$$\mathcal{L}(X, Y) = f_\tau(X) + \langle Y, \mathcal{P}_\Omega(M - X) \rangle,$$

nato pa iščemo njene stacionarne točke. Zaradi velikega števila parametrov pa ta pristop navadno ni izvedljiv, zato se v SVT-ju uporabi za iskanje ekstremov  $\mathcal{L}(X, Y)$  t.i. *Uzawa algoritem* [2]. Ta ekstreme išče prek iterativnega postopka:

$$X^{(k)} = \arg \min_X \mathcal{L}(X^{(k)}, Y^{(k-1)}), \quad (3.10)$$

$$Y^{(k)} = Y^{(k-1)} + \delta_k \mathcal{P}_\Omega(M - X^{(k)}) \quad (3.11)$$

Še pri stilu trditve bo potrebno poenotiti označevanje, tako kot pri izrekih. To lahko na koncu dorečeva, saj se samo spremeni v preambuli.

Izkaže se, da je rešitev (3.10) enaka  $\mathcal{D}_\tau(Y^{k-1})$ . To spodaj dokažemo, še prej pa izpeljimo pomožen rezultat.

**Trditev 3.0.1.** *Velja:*

$$\arg \min_X \mathcal{L}(X, Y) = \arg \min_X \left( \tau \|X\|_* + \frac{1}{2} \|X - Y\|_F^2 \right) \quad (3.12)$$

*Dokaz.* Trditev sledi iz krajšega računa:

$$\begin{aligned} & \arg \min_X \left( \tau \|X\|_* + \frac{1}{2} \|X - Y\|_F^2 \right) \\ &= \arg \min_X \left( \tau \|X\|_* + \frac{1}{2} \langle X - Y, X - Y \rangle \right) \\ &= \arg \min_X \left( \tau \|X\|_* + \frac{1}{2} (\|X\|_F^2 - 2 \langle X, Y \rangle + \|Y\|_F^2) \right) \\ &= \arg \min_X \left( \tau \|X\|_* + \frac{1}{2} \|X\|_F^2 - \langle X, \mathcal{P}_\Omega(Y) \rangle \right) \\ &= \arg \min_X \left( \tau \|X\|_* + \frac{1}{2} \|X\|_F^2 + \text{Tr}(-\mathcal{P}_\Omega(X)Y^T) + \text{Tr}(\mathcal{P}_\Omega(M)Y^T) \right) \\ &= \arg \min_X \left( \tau \|X\|_* + \frac{1}{2} \|X\|_F^2 + \text{Tr}(\mathcal{P}_\Omega(M - X)Y^T) \right) \\ &= \arg \min_X \left( \tau \|X\|_* + \frac{1}{2} \|X\|_F^2 + \langle Y, \mathcal{P}_\Omega(M - X) \rangle \right) \\ &= \arg \min_X \mathcal{L}(X, Y) \end{aligned}$$

kjer smo v prvi enakosti uporabili definicijo Frobeniusove norme, v drugi bilinearnost skalarnega produkta, v tretji smo ignorirali konstanto  $\|Y\|_F^2$ , saj

ne vpliva na rezultat, in upoštevali  $\mathcal{P}_\Omega(Y^{(k)}) = Y^{(k)}$  za vse  $k \in \mathbb{N}$ . Zadnje dejstvo sledi iz definicije (3.11) in  $Y^{(0)} = 0$ . V četrti enakosti smo upoštevali  $\langle X, \mathcal{P}_\Omega(Y) \rangle = \langle \mathcal{P}_\Omega(X), Y \rangle$ , kar je lahko videti, ko se spomnimo, da za skalarni produkt  $\langle A, B \rangle$  matrik  $A, B \in \mathbb{R}^{n_1 \times n_2}$  velja

$$\langle A, B \rangle = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} a_{ij} b_{ij}.$$

Prišteli smo tudi konstanto  $\text{Tr}(\mathcal{P}_\Omega(M)Y^T)$ , ki ne vpliva na rezultat. V peti enakosti smo upoštevali linearnost sledi in aditivnost operatorja  $P_\Omega$ , v šesti definicijo skalarnega produkta in v zadnji definicijo Lagrangeove funkcije.  $\square$

**Izrek 1.** Za matriki  $X \in \mathbb{R}^{n_1 \times n_2}$  in  $Y \in \mathbb{R}^{n_1 \times n_2}$  velja:

$$\mathcal{D}_\tau(Y) = \arg \min_X \left( \frac{1}{2} \|X - Y\|_F^2 + \tau \|X\|_* \right) \quad (3.13)$$

*Dokaz.* Najprej se spomnimo definicije konveksne funkcije. Funkcija  $f$  je konveksna, če za katerikoli dve točki  $x_1, x_2$  v domeni funkcije  $f$  velja, da je premica čez ti dve točki na odseku med temi dvema točkama nad grafom funkcije  $f$ . Funkcija  $h(X) := \frac{1}{2} \|X - Y\|_F^2 + \tau \|X\|_*$  je konveksna funkcija, saj potreben pogoj trikotniške neenakosti matričnih norm zagotavlja, da je matrična norma konveksna funkcija. Vsota konveksnih funkcij pa je prav tako konveksna funkcija. Zaradi konveksnosti funkcije  $f$  je subgradient v vsaki točki iz domene dobro definiran. **Tega smo definirali v razdelku 3.1.** Matrika  $Z \in \mathbb{R}^{n_1 \times n_2}$  je subgradient funkcije  $f$  v točki  $X_0 \in \mathbb{R}^{n_1 \times n_2}$ , če velja

$$\forall X \in \mathbb{R}^{n_1 \times n_2} : f(X) \geq f(X_0) + \langle Z, X - X_0 \rangle.$$

Iz definicije subgradiента sledi, da bo imela funkcija  $f$  minimum v točki  $X'$  natanko tedaj, ko bo ničelna matrika  $\mathbf{0}$  eden izmed subgradientov funkcije  $f$  v točki  $X_0$ .

Izkaže se [2], da je množica subgradientov nuklearne norme v točki  $X$  enaka

$$\partial\|X\|_* = \{UV^* + W : W \in \mathbb{R}^{n_1 \times n_2}, U^*W = \mathbf{0}, WV = \mathbf{0}, \|W\|_2 \leq 1\}.$$

kjer  $U\Sigma V^T$  predstavlja SVD razcep matrike  $X$ . S krajšim računom lahko preverimo, da za vsak par  $(i, j)$  velja  $\frac{\partial}{\partial X_{ij}} \|X - Y\|_F^2 = 2(X_{ij} - Y_{ij})$ . Če te parcialne odvode zložimo v matriko, dobimo  $2(X - Y)$ . Od tod sledi, da je  $X - Y$  eden od subgradientov funkcije  $h_1(X) := \frac{1}{2}\|X - Y\|_F^2$  v točki  $X$ . Iz teh dveh premislekov sledi, da bo  $X'$  minimum  $h$  natanko tedaj, ko velja

$$\mathbf{0} \in X' - Y + \tau \partial \|X'\|_*$$

Trditev izreka bo sledila, če pokažemo, da velja  $X' = \mathcal{D}_\tau(Y)$ . Najprej razčlenimo SVD razcep matrike  $Y$  kot

$$Y = U_0 \Sigma_0 V_0^T + U_1 \Sigma_1 V_1^T$$

kjer  $U_0, \Sigma_0$  in  $V_0$  predstavljajo singularne vrednosti in pripadajoče singularne vektorje večje od  $\tau$ ,  $U_1, \Sigma_1$  in  $V_1$  pa tiste manjše od  $\tau$ . Pokazati želimo, da velja

$$X' = U_0(\Sigma_0 - \tau I)V_0^T.$$

Če zapis vstavimo v prejšnji podan pogoj, dobimo

$$\begin{aligned} \mathbf{0} &= X' - Y + \tau \partial \|X'\|_* \\ Y - X' &= \tau(U_0 V_0^T + W), \end{aligned}$$

primerna izbira za  $W$  pa je  $W = \tau^{-1}U_1 \Sigma_1 V_1^T$ . Res,

$$\begin{aligned} Y - X' &= U_0 \Sigma_0 V_0^T + U_1 \Sigma_1 V_1^T - U_0(\Sigma_0 - \tau I)V_0^T \\ &= U_0 V_0^T (\Sigma_0 - \Sigma_0 + \tau I) + U_1 \Sigma_1 V_1^T \\ &= \tau U_0 V_0^T + U_1 \Sigma_1 V_1^T \end{aligned}$$

$U_0(\Sigma_0 - \Sigma_0 + \tau I)V_0^T$  in

$$\begin{aligned} \tau(U_0 V_0^T + W) &= \tau(U_0 V_0^T + \tau^{-1}U_1 \Sigma_1 V_1^T) \\ &= \tau U_0 V_0^T + U_1 \Sigma_1 V_1^T \end{aligned}$$

Sedaj je zgolj potrebno pokazati, da veljajo potrebne lastnosti matrike  $W$ . Po sami definiciji SVD vemo, da so vsi stolpci matrik U in V ortogonalni.

Torej velja  $U_0^T W = 0$  in  $WV_0 = 0$ . Ker pa ima matrika  $\Sigma_1$  vse elemente manjše od  $\tau$  velja tudi  $\|W\|_2 \leq 1$ .  $\|A\|_2$  je namreč definirana kot največja singularna vrednost matrike  $A$ . S tem smo pokazali, da  $Y - X' \in \tau \partial \|X'\|_*$ .

□

Z uporabo (3.10), (3.11) in izreka 1 res pridemo do iteracije (3.6), ki jo uporablja algoritom SVT.

### 3.3.1 Nastavljanje parametrov $\tau$ in $\delta$

Opazimo lahko, da algoritom SVT potrebuje dva parametra,  $\tau$  in  $\delta$ , ki ju moramo izbrati že pred vstopom v algoritom.

Po priporočilih [2] sta primerni izbiri za  $\delta$  in  $\tau$  enaki

$$\delta = 1.2 \frac{n_1 n_2}{m} \quad \text{in} \quad \tau = 5n,$$

pri čemer je  $\tau$  naveden za kvadratne  $n \times n$  matrike. V poglavju z rezultati smo prvotno uporabljali ti dve konstanti, pri čemer smo zaradi pravokotnosti  $n_1 \times n_2$  matrik uporabljali  $\tau = 5 \frac{n_1 + n_2}{2}$ . V naših testiranjih se je izkazalo, da sta taka parametra dobra za večje matrike. V razdelku 4.2 pa bomo videli, da moramo pri manjših matrikah pogosto zmanjšati premik in povečati prag.

## 3.4 Algoritem minimizacije prirezane nuklearne norme

Kot nam že samo ime pove, je **algoritem minimizacije prirezane nuklearne norme (TNNM)** [8] soroden algoritmu NNM. Dodatna informacija, ki pa jo uporabi TNNM, je rang  $r$  originalne, nezašumljene matrike.

Osrednjo vlogo v algoritmu ima  **$r$ -prirezana nuklearna norma**, ki za dano matriko  $X \in \mathbb{R}^{n_1 \times n_2}$  vrne vsoto njenih  $\min(n_1, n_2) - r$  najmanjših singularnih vrednosti:

$$\|X\|_r = \sum_{i=r+1}^{\min(n_1, n_2)} \sigma_i(X).$$

TNNM rešuje optimizacijski problem

$$\min_{X \in \mathbb{R}^{n_1 \times n_2}} \|X\|_r, \quad (3.14)$$

pri pogoju  $\mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M)$ .

Cilj algoritma je torej čim bolj zmanjšati najmanjše singularne vrednosti, medtem ko velikih ne omejujemo. S tem problem minimizacije omilimo.

Problem (3.14) lahko zapišemo v ekvivalentni obliki

$$\min_{X \in \mathbb{R}^{n_1 \times n_2}} \|X\|_* - \sum_{i=1}^r \sigma_i(X), \quad (3.15)$$

pri pogojih  $\mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M)$

V izpeljavah, ki sledijo, bomo potrebovali naslednji izrek.

**Izrek 2.** Za matrike  $X \in \mathbb{R}^{n_1 \times n_2}$ ,  $A \in \mathbb{R}^{r \times n_1}$  in  $B \in \mathbb{R}^{r \times n_2}$ , ter naravno število  $r \in \mathbb{N}$ , ki zadoščajo pogojem  $r \leq \min(n_1, n_2)$ ,  $AA^T = I_r$  in  $BB^T = I_r$ , velja neenakost

$$\mathrm{Tr}(AXB^T) \leq \sum_{i=1}^r \sigma_i(X) \quad (3.16)$$

*Dokaz.* Velja

$$\mathrm{Tr}(AXB^T) = \mathrm{Tr}(XB^TA) \leq \sum_{i=1}^{\min(n_1, n_2)} \sigma_i(X)\sigma_i(B^TA), \quad (3.17)$$

kjer smo v enakosti uporabili komutativnost  $\mathrm{Tr}(ZW) = \mathrm{Tr}(WZ)$  sledi, v neenakosti pa von Neumannovo neenakost za sled [8].

Po definiciji so singularne vrednosti matrike  $Y$  enake korenom lastnih vrednosti matrike  $Y^TY$ . Torej so singularne vrednosti matrike  $B^TA$  enake korenom lastnih vrednosti matrike  $A^TBB^TA = A^TI_rA = A^TA$ . Ker imata matriki  $XY$  in  $YX$  enake neničelne lastne vrednosti in po predpostavki velja  $AA^T = I_r$ , lahko od tod sklepamo, da ima produkt  $B^TA$   $r$  singularnih vrednosti enakih 1. Zato velja

$$\sum_{i=1}^{\min(n_1, n_2)} \sigma_i(X)\sigma_i(B^TA) = \sum_{i=1}^r \sigma_i(X),$$

kar skupaj z (3.17) dokaza neenakost (3.16) v izreku.  $\square$

ali tukaj  
tudi oklepaje

**Izrek 3.** Naj bo  $X = U\Sigma V^T$  SVD razcep matrike  $X$ . Naj bosta  $A$  in  $B$  matriki, sestavljeni iz prvih  $r$  stolpcev matrik  $U$  in  $V$ . Velja

$$\mathrm{Tr}(AXB^T) = \sum_{i=1}^r \sigma_i(X).$$

*Dokaz.* Označimo matriki  $A$  in  $B$  z  $A = (u_1, \dots, u_r)^T$  in  $B = (v_1, \dots, v_r)^T$ , kjer je  $u_i$   $i$ -ti stolpec matrike  $U$  ter  $v_i$   $i$ -ti stolpec matrike  $V$ .

$$\begin{aligned} \mathrm{Tr}(AXB^T) &= \mathrm{Tr}((u_1, \dots, u_r)^T X (u_1, \dots, u_r)^T) \\ &= \mathrm{Tr}((u_1, \dots, u_r)^T U \Sigma V^T (u_1, \dots, u_r)^T) \\ &= \mathrm{Tr}\left(\begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} \Sigma \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix}\right) \\ &= \mathrm{Tr}\left(\begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_r & 0 \\ & & & \ddots \\ & & & & 0 \end{bmatrix}\right) \\ &= \sum_{i=1}^r \sigma_i(X), \end{aligned}$$

kar dokaže trditev izreka. □

Po izrekih 2 in 3 velja

$$\max_{\substack{AA^T=I, \\ BB^T=I}} \mathrm{Tr}(AXB^T) = \sum_{i=1}^r \sigma_i(X)$$

Torej je optimizacijski problem (3.15) ekvivalenten problemu

$$\min_{X \in \mathbb{R}^{n_1 \times n_2}} \|X\|_* - \max_{\substack{AA^T=I, \\ BB^T=I}} \mathrm{Tr}(AXB^T),$$

pri pogoju  $\mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M)$ .

Sedaj lahko opišemo idejo algoritma TNNM:

1. Izračunamo  $X^{(0)} = \mathcal{P}_\Omega(M)$ .

2. Za  $k = 0, 1, 2, \dots$  ponavljamo iteracijo:

- (a) Izračunamo SVD razcep  $X^{(k)} = U^{(k)}\Sigma^{(k)}(V^{(k)})^T$ .
- (b)  $A^{(k)}$  definiramo kot prvih  $r$  stolpcev matrike  $U^{(k)}$ ,  $B^{(k)}$  pa kot prvih  $r$  stolpcev matrike  $V^{(k)}$ .
- (c)  $X^{(k+1)}$  je enak rešitvi optimizacijskega problema

$$\min_{X \in \mathbb{R}^{n_1 \times n_2}} \|X\|_* - \text{Tr}(A^{(k)} X (B^{(k)})^T), \quad (3.18)$$

pri pogoju  $\mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M)$ .

Z nadaljnjam preoblikovanjem problema (3.18) v ekvivalentnega

$$\min_{X \in \mathbb{R}^{n_1 \times n_2}} \|X\|_* - \text{Tr}(A^{(k)} W (B^{(k)})^T), \quad (3.19)$$

pri pogojih  $W = X$ ,  $\mathcal{P}_\Omega(W) = \mathcal{P}_\Omega(M)$ ,

lahko za reševanje uporabimo **algoritem ADMM** [8].

Gre za reševanje problema vezanih ekstremov, ki ga lahko zapišemo s pomočjo Lagrangeove funkcije, pri čemer algoritem ADMM doda še člen, pomnožen z *regularizacijskim parametrom*  $\beta$ .

$$\mathcal{L}(X, Y, W, \beta) = \|X\|_* - \text{Tr}(AWB^T) + \frac{\beta}{2}\|X - W\|_F^2 + \text{Tr}(Y^T(X - W)).$$

Matriki  $A$  in  $B$  sta v okviru algoritma ADMM konstantni. Ti na začetku nastavimo na vrednost  $A^{(k)}$  in  $B^{(k)}$  iz prejšnje iteracije. Opazimo lahko, da funkcija upošteva zgolj pogoj  $X = W$ . Videli bomo, da algoritem pogoj  $\mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M)$  definira posredno, s popravljanjem znanih vrednosti (korak (3.20) spodaj).

Matriko  $X^{(k+1)}$  definiramo kot

$$X^{(k+1)} = \arg \min_X \mathcal{L}(X, Y^{(k)}, W^{(k)}, \beta)$$

Z ignoriranjem konstantnih členov, pa lahko zapišemo

$$\begin{aligned} X^{(k+1)} &= \arg \min_X \left( \|X\|_* + \frac{\beta}{2} \langle X - W^{(k)}, X - W^{(k)} \rangle + \frac{\beta}{2} \langle \frac{2}{\beta} Y, X \rangle \right) \\ &= \arg \min_X \left( \|X\|_* + \frac{\beta}{2} \text{Tr} \left( X^T X - X^T W^{(k)} - \right. \right. \\ &\quad \left. \left. (W^{(k)})^T X + (W^{(k)})^T W^{(k)} + \frac{2}{\beta} (Y^{(k)})^T X \right) \right). \end{aligned}$$

iz kje pride  
izracun  
 $Y^{i+1}$

Z namenom faktorizacije dodamo konstantne člene

$$-(W^{(k)})^T \frac{1}{\beta} (Y^{(k)}) + \left( \frac{1}{\beta} (Y^{(k)}) \right)^T (-W^{(k)} + \frac{1}{\beta} (Y^{(k)}))$$

in dobimo

$$\begin{aligned} & \arg \min_X \left( \|X\|_* + \frac{\beta}{2} \operatorname{Tr} \left( X^T (X - W^{(k)} + \frac{1}{\beta} Y^{(k)}) - (W^{(k)})^T (X - W^{(k)} + \frac{1}{\beta} Y^{(k)}) \right. \right. \\ & \quad \left. \left. + \frac{1}{\beta} (Y^{(k)})^T (X - W^{(k)} + \frac{1}{\beta} Y^{(k)}) \right) \right) \\ & = \arg \min_X \left( \|X\|_* + \frac{\beta}{2} \|X - W^{(k)} + \frac{1}{\beta} Y^{(k)}\|_F^2 \right) \\ & = \arg \min_X \left( \frac{1}{\beta} \|X\|_* + \frac{1}{2} \|X - (W^{(k)} - \frac{1}{\beta} Y^{(k)})\|_F^2 \right) \end{aligned}$$

Po izreku 1 uporabljenem za  $X = X$ ,  $Y = W - \frac{1}{\beta} Y$ ,  $\tau = \frac{1}{\beta}$  pa sledi

$$X^{(k+1)} = \mathcal{D}_{\frac{1}{\beta}}(W^{(k)} - \frac{1}{\beta} Y^{(k)}).$$

Matriko  $W^{(k+1)}$  podobno izračunamo kot

$$\begin{aligned} W^{(k+1)} &= \arg \min_W \mathcal{L}(X^{(k+1)}, Y^{(k)}, W, \beta) \\ &= \arg \min_W \frac{\beta}{2} \|W - (X^{(k+1)} + \frac{1}{\beta} (A^T B + Y^{(k)}))\|_F^2 \end{aligned}$$

Očitno je

$$W^{(k+1)} = X^{(k+1)} + \frac{1}{\beta} (A^T B + Y^{(k)})$$

Sedaj uporabimo še pogoj  $\mathcal{P}_\Omega(W^{(k+1)}) = \mathcal{P}_\Omega(M)$  in tiste elemente, ki jih poznamo, popravimo.

$$W^{(k+1)} = W^{(k+1)} + \mathcal{P}_\Omega(M - W^{(k+1)}) \quad (3.20)$$

Z besedami, predpis (3.20) preprosto spremeni mesta  $W^{(k+1)}$ , kjer vrednosti poznamo, na znane vrednosti, ostalih pa ne spremeni.

### 3.5 Izmenjajoč gradientni spust

Računanje SVD razcepa je zahtevna operacija, saj ima časovno zahtevnost  $O(n^3)$ , kar je lahko za veliko matrike zelo počasno. Zato je bilo izpeljanih nekaj algoritmov, ki za svoje delovanje ne potrebujejo SVD-ja. **Algoritem izmenjajočega gradientnega spusta (ASD)** [12] temelji na izračunu gradijeta in premiku v smeri tega za nek korak. Ideja algoritma je poiskati matriki  $X \in \mathbb{R}^{n_1 \times r}$  ter  $Y \in \mathbb{R}^{r \times n_2}$ , tako da velja  $\mathcal{P}_\Omega(M) = \mathcal{P}_\Omega(XY)$ . Tudi tu potrebujemo informacijo o rangu matrike, ki jo rekonstruiramo. Ker imata  $X$  in  $Y$  rang največ  $r$ , tudi njun produkt  $XY$  ne bo imel ranga večjega od  $r$ .

Cilj algoritma je rešiti optimizacijski problem

$$\min_{\substack{X \in \mathbb{R}^{n_1 \times r}, \\ Y \in \mathbb{R}^{r \times n_2}}} \frac{1}{2} \|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(XY)\|_F^2$$

Algoritem problem minimizacije razdeli na dva optimizacijska podproblema, ki ju rešuje izmenično s pomočjo iterativnega postopka

$$X^{(k+1)} = \arg \min_X \|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(XY^{(k)})\|_F^2, \quad (3.21)$$

$$Y^{(k+1)} = \arg \min_Y \|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(X^{(k+1)}Y)\|_F^2 \quad (3.22)$$

Za uporabo algoritma ASD potrebujemo gradijeta funkcije  $f(X, Y) = \frac{1}{2} \|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(XY)\|_F^2$  po obeh spremenljivkah, ki ju izračunamo za vsak element posebej. Za reševanje (3.21) so spremenljivke elementi matrike  $X$ , za reševanje (3.22) pa elementi matrike  $Y$ . Zaradi enostavnosti definiramo dve pomožni funkciji. Funkcija  $f_Y(X)$  je enaka funkciji  $f(X, Y)$ , le da vrednost  $Y$  jemljemo kot konstanto. Simetrično definiramo  $f_X(Y)$ , kjer jemljemo vrednost  $X$  funkcije  $f(X, Y)$  za konstantno. Sedaj preprosto poiščemo oba gradijeta pomožnih funkcij, označena z  $\nabla f_Y(X)$  in  $\nabla f_X(Y)$  ter najboljša premika po gradientih, označena s  $t_x$  in  $t_y$ . Sam algoritem se potem preprosto premika po gradientih za določen premik. Koraka algoritma bomo v vsaki iteraciji definirali kot

$$X^{(k+1)} = X^{(k)} - t_{x(k)} \nabla f_{Y^{(k)}}(X^{(k)}), \quad (3.23)$$

$$Y^{(k+1)} = Y^{(k)} - t_{y(k)} \nabla f_{X^{(k+1)}}(Y^{(k)}).$$

**Izrek 4.** *Velja*

$$\begin{aligned}\nabla f_Y(X) &= -(\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(XY))Y^T \\ \nabla f_X(Y) &= -X^T(\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(XY)) \\ t_x &= \frac{\|\nabla f_Y(X)\|_F^2}{\|\mathcal{P}_\Omega(\nabla f_Y(X)Y)\|_F^2} \\ t_y &= \frac{\|\nabla f_X(Y)\|_F^2}{\|\mathcal{P}_\Omega(X\nabla f_X(Y))\|_F^2}\end{aligned}$$

*Dokaz.* Pri tem dokazu nam bo prav prišla definicija  $\delta_{i,j}$ , definirana kot

$$\delta_{i,j} = \begin{cases} 1, & (i,j) \in \Omega, \\ 0, & (i,j) \notin \Omega. \end{cases}$$

Za gradient funkcije  $f_Y(X)$  po spremenljivki X, označen z  $\nabla f_Y(X)$ , velja

$$\begin{aligned}\nabla f_Y(X)_{a,b} &= \frac{\partial}{\partial x_{a,b}} \frac{1}{2} \|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(XY)\|_F^2 \\ &= \frac{\partial}{\partial x_{a,b}} \frac{1}{2} \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} (\delta_{i,j} m_{i,j} - \delta_{i,j} \sum_{k=1}^r (x_{i,k} y_{k,j}))^2 \\ &= \sum_{j=1}^{n_2} (\delta_{a,j} m_{a,j} - \delta_{a,j} \sum_{k=1}^r (x_{a,k} y_{k,j})) (-y_{b,j}) \\ \implies \nabla f_Y(X) &= -(\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(XY))Y^T\end{aligned}$$

Na podoben način lahko pokažemo še

$$\nabla f_X(Y) = -X^T(\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(XY))$$

Poščimo še najboljši korak gradientnega spusta. Cilj je pokazati, da je premik po gradientu s korakom  $t_x$  najboljši. Sam premik smo zgoraj (3.23) definirali kot

$$X^{(k+1)} = X^{(k)} - t_{x^{(k)}} \nabla f_{Y^{(k)}}(X^{(k)})$$

Ker želimo, da bodo znane vrednosti produkta  $X^{(k+1)}Y^{(k)}$  kar se da podobne znanim vrednostim matrike  $M$ , nastavimo  $t_x$  kot

$$t_x = \arg \min_t g(t)$$

kjer je

$$g(t) = \frac{1}{2} \|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega((X - t\nabla f_Y(X))Y)\|_F^2$$

Kot v [12] se izkaže, da velja

$$g'(t) = -\|\nabla f_Y(X)\|_F^2 + t\|\mathcal{P}_\Omega(f_Y(X)Y)\|_F^2.$$

Od tod sklepamo, da funkcija  $g(t)$  doseže minimum pri

$$t_x = \frac{\|\nabla f_Y(X)\|_F^2}{\|\mathcal{P}_\Omega(\nabla f_Y(X)Y)\|_F^2}$$

Podobno velja za korak v smeri gradientnega spusta matrike  $Y$ , kjer

$$t_y = \frac{\|\nabla f_X(Y)\|_F^2}{\|\mathcal{P}_\Omega(X\nabla f_X(Y))\|_F^2}$$

□

### 3.6 LMaFit

**LMaFit** je algoritem, ki podobno kot ASD, rešuje problem matričnih napolnitev z uporabo dveh manjših matrik  $X^{n_1 \times r}$  in  $Y^{n_1 \times r}$ . Optimizacijski problem, ki ga rešuje LMaFit je

$$\min_{X,Y,Z} \quad \frac{1}{2} \|XY - Z\|_F^2,$$

pri pogoju  $\mathcal{P}_\Omega(M) = \mathcal{P}_\Omega(Z).$

LMaFit za reševanje tega problema uporabi Moore-Penroseov (MP) inverz. V nadaljevanju bomo MP inverz matrike  $A$  označevali z  $A^\dagger$ .

Očitno bo imela funkcija  $f(X, Y, Z) = \frac{1}{2} \|XY - Z\|_F^2$  najmanjšo vrednost natanko tedaj, ko bo  $XY = Z$ . LMaFit izmenično posodablja matrike  $X$ ,  $Y$  in  $Z$ , pri čemer eno spreminja, preostali dve pa fiksira. Matriko  $X$  posodobi po formuli

$$X^{i+1}Y^i = Z^i \quad \Rightarrow \quad X^{i+1} = Z^i(Y^i)^\dagger,$$

pri čemer uporablja dejstvo, da množenje z MP inverzom z desne da najboljši rezultat po metodi najmanjših kvadratov Lahko dodaš citat. Podobno posodobimo matriko  $Y$ .

$$X^{i+1}Y^{i+1} = Z^i \quad \Rightarrow \quad Y^{i+1} = (X^{i+1})^\dagger Z^i.$$

Na koncu le še posodobimo matriko  $Z$ , kot

$$Z^{i+1} = X^{i+1}Y^{i+1} + \mathcal{P}_\Omega(M - X^{i+1}Y^{i+1}).$$

Torej podobno kot prej poskušamo doseči  $XY = Z$ , vendar zaradi omejitve znanih vrednosti matrike  $M$ , na tistem mestu vrednosti popravimo.



# Poglavlje 4

## Rezultati

V tem poglavju opišemo rezultate, ki smo jih pridobili med testiranjem algoritmov. Kot smo omenili v uvodu, bomo predstavljene algoritme za matrične napolnitve testirali na slikah z manjkajočimi pikslji. V razdelku 4.6 bomo videli, da metoda matričnih napolnitev ni najboljša za rekonstrukcijo slik. Za testiranje na slikah smo se odločili, ker lahko problem lažje vizualiziramo, kot če bi testirali na naključno generiranih podatkih. Pri surovih podatkih pomena numeričnih vrednosti ni tako enostavno interpretirati, zaradi česar težje ovrednotimo koristnost algoritmov.

Poleg točnosti rezultatov različnih metod merimo tudi čas njihovega izvajanja. Probleme zaženemo tudi na različnih vrstah podatkov, npr. podatkih, ki so zašumljeni enakomerno, kot tudi slikah, na katerih odstranjujemo besedilo. Pri interpretaciji slike razdelimo v več skupin in za vsako skupino razložimo ugotovitve.

V tem odstavku opišemo strukturo poglavja. V razdelku 4.1 primerjamo algoritme na problemu rekonstrukcije zašumljene velike črno-bele slike. V razdelku 4.2 raziščemo vpliv kompleksnosti motivov na slikah na rezultate algoritmov. V razdelku 4.3 preizkusimo in primerjamo dva načina rekonstrukcije podatkov, kjer so nekateri podatki med seboj neodvisni. Iz poglavja 3 vemo, da nekateri algoritmi (TNNM, ASD, LMaFit) za svoje delovanje potrebujejo informacijo o rangu nezašumljenih podatkov. V razdelku 4.4

raziščemo vpliv informacije o rangu na rezultate. V razdelku 4.5 preverjamo delovanje algoritmov na slikah, kjer so neznani podatki zgoščeni. Zgoščen šum je v našem primeru dodano besedilo na sliki, ki ga želimo odstraniti. Razdelek 4.6 pa primerja algoritme z drugačno metodo rekonstrukcije slik, ki neznane piksle določa prek reševanja Poissonove parcialne diferencialne enačbe. Razdelek 4.6 strne ugotovitve, predstavljene v ostalih razdelkih.

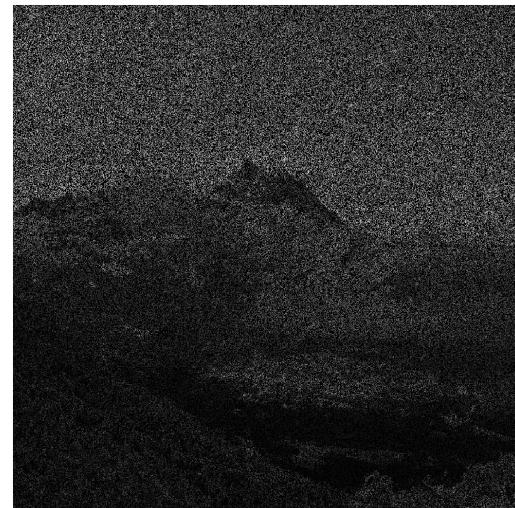
Vse slike si je mogoče v boljši kakovosti ogledati na povezavi <https://cutt.ly/matrice-napolnitve>.

## 4.1 Velika črno-bela slika

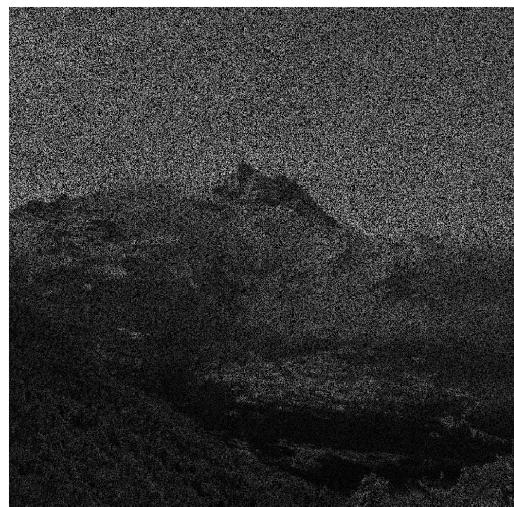
Algoritme najprej testiramo na veliki, črno-beli sliki, velikosti  $1000 \times 1000$  pikslov. Velika slika je bila izbrana, ker omogoča lažje vizualno ocenjevanje delovanja in pravilnosti algoritmov. Medtem ko se v naslednjih razdelkih osredotočamo na specifična vprašanja o rekonstrukciji, je cilj te faze zgolj raziskati, kako dobro algoritmi delujejo, ter katere je v naslednjih fazah sploh smiselno testirati. Zaradi velike časovne zahtevnosti pri velikih slikah, testiranja v ostalih razdelkih opravljamo na manjših slikah. Algoritme preizkušamo trikrat, na podatkih z deleži znanih vrednosti 0.35, 0.45 in 0.60.



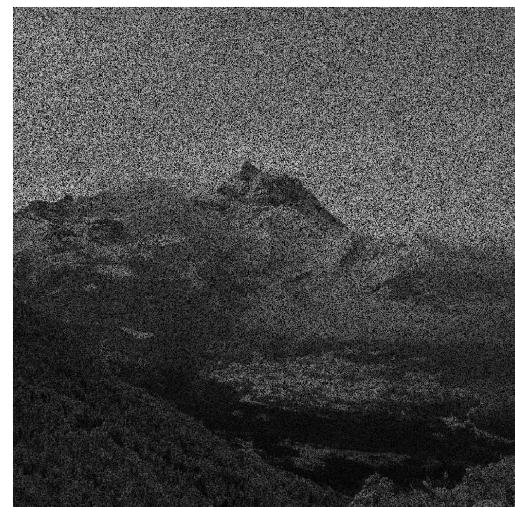
(a) Originalna slika.



(b) Slika s 35% znanimi podatki.

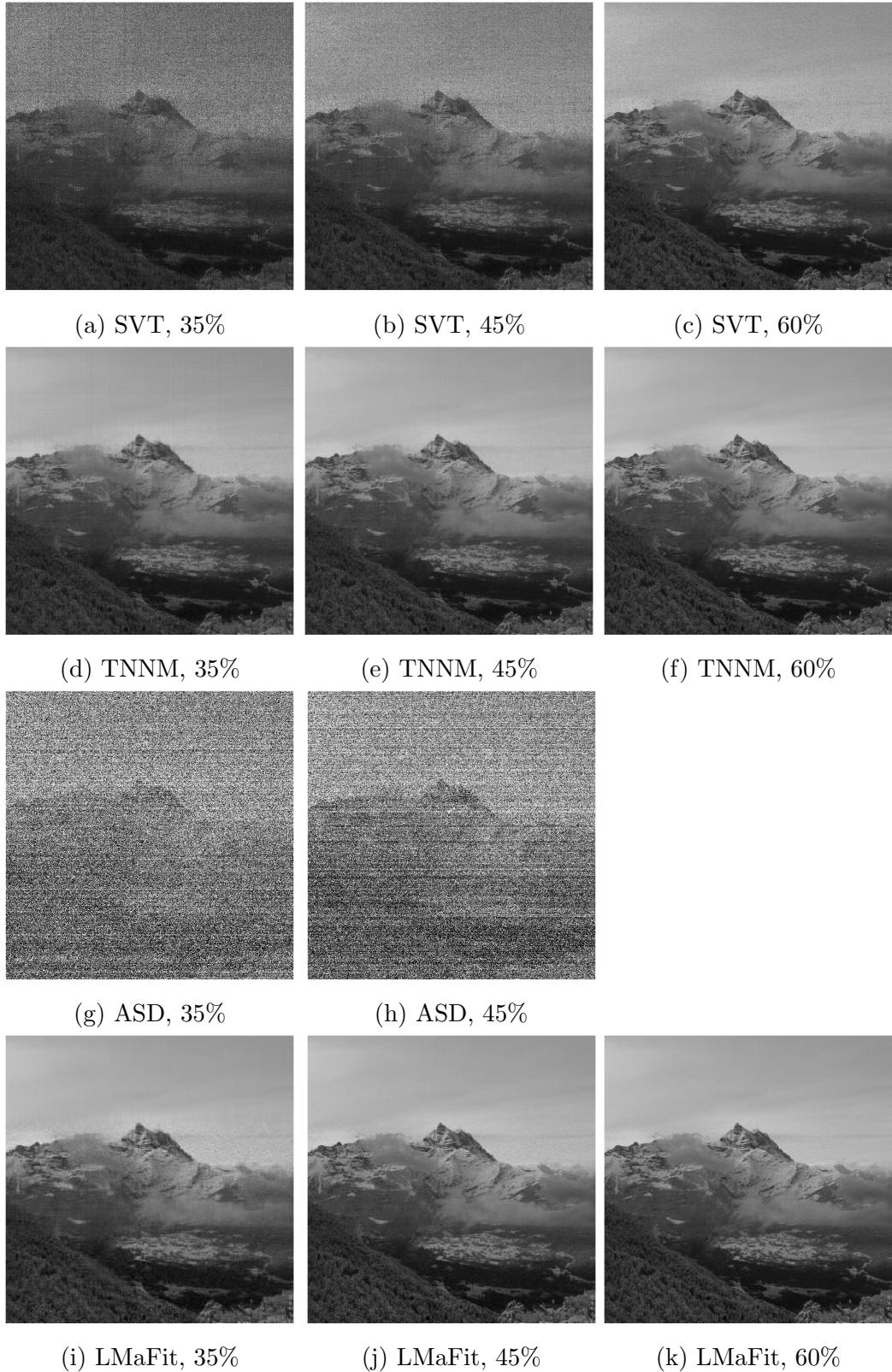


(c) Slika s 45% znanimi podatki.



(d) Slika s 60% znanimi podatki.

Slika 4.1: Slika uporabljena za rekonstrukcijo. Vir slike: [1]



Slika 4.2: Rekonstrukcija zašumljenih slik z uporabo različnih algoritmov in pri različnih odstotkih znanih vrednosti. Kratica pod sliko označuje uporabljen algoritem, odstotek za vejico pa odstotek znanih vrednosti.

Opazimo lahko, da je med rezultati velika razlika. Že na pogled lahko rečemo, da algoritma TNNM in LMaFit delujeta najbolje, SVT nekoliko slabše, medtem ko ASD vrne slabe rezultate. Te si lahko interpretiramo kot posledico lastnosti, da lahko algoritem konča v lokalnem minimumu. Prav tako algoritem ASD ni našel rešitve, ko je imel znanih 60% podatkov. Zato je ta algoritma smiselnouporabljati, kadar imamo manj poznanih vrednosti in dober začetni približek matrik  $X$  in  $Y$ .

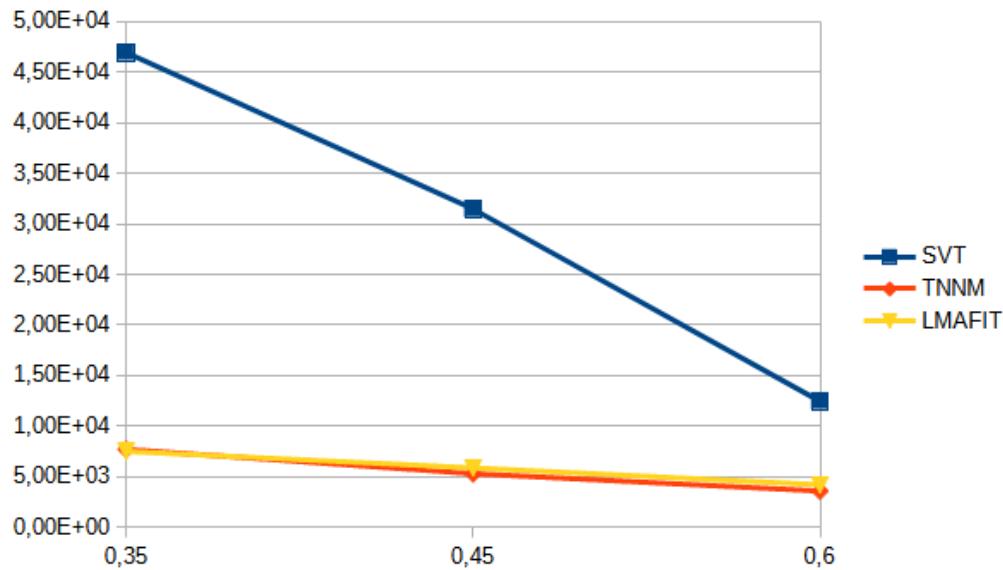
Ali je spodnji rob premajhen pri prejšnji sliki

Algoritem NNM smo med rezultati izpustili, saj je zaradi velikega števila matrik, potrebnih za definicijo omejitev, algoritem preveč prostorsko kompleksen. Ta algoritem je zato smiselnouporabljati, kadar imamo matrike manjših dimenzij [3] (algoritem lahko še rešuje probleme, za matrike velikosti  $100 \times 100$ ). Prednost tega algoritma je, da vrne točno tisto rešitev, kjer je nuklearna norma najmanjša. Zaradi konveksne ovojnice (omenjene v razdelku 3.2), bi moral torej ta algoritem vrniti najboljše rezultate. Zaradi teh opazk se v naslednjih razdelkih osredotočamo na algoritme SVT, TNNM in LMaFit.

OZP \ Algoritem	SVT	TNNM	LMAFIT	ASD
35%	$4.69 \times 10^4$	$7.70 \times 10^3$	$7.50 \times 10^3$	$3.9743 \times 10^7$
45%	$3.15 \times 10^4$	$5.30 \times 10^3$	$5.87 \times 10^3$	$6.0910 \times 10^7$
60%	$1.25 \times 10^4$	$3.58 \times 10^3$	$4.20 \times 10^3$	-

Tabela 4.1: Napake algoritmov izračunane v Frobeniusovi normi. Kratica OZP stoji za odstotek znanih podatkov

Medtem ko bomo čase izvajanja algoritmov podrobnejše analizirali v naslednjem razdelku, lahko že sedaj opišemo par ugotovitev. Vidimo lahko, da je algoritem SVT veliko počasnejši, kadar je delež znanih vrednosti večji, medtem ko se čas izvajanja ostalih algoritmov z dodajanjem pikslov manjša. Prav tako časovna kompleksnost ni linearна, saj lahko opazimo, da je algoritmu SVT čas izvajanja veliko hitreje naraščal v preskoku iz 45% na 60%



Slika 4.3: Napake algoritmov v Frobeniusovi normi. Na abscisni osi so deleži znanih podatkov slik.

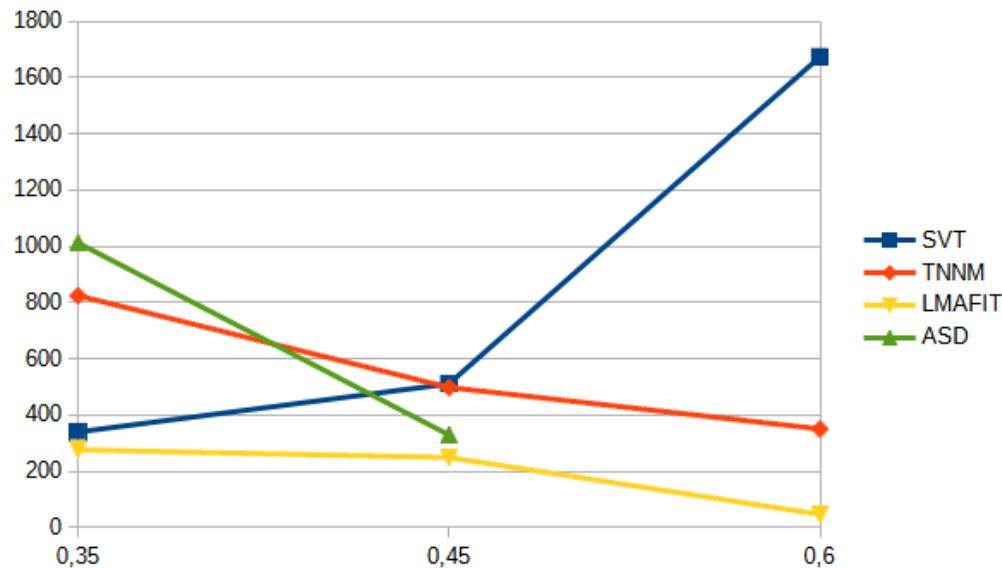
OZP \ Algoritem	SVT	TNNM	LMAFIT	ASD
35%	338s	824s	275s	1012s
45%	510s	498s	248s	328s
60%	1674s	350s	45.6s	-

Tabela 4.2: Časi do dosega zaustavitvenega pogoja.

kot pri preskoku iz 35% na 45%.

## 4.2 Vpliv kompleksnosti slik na rekonstrukcijo

Pomembno vprašanje pri študiju algoritmov matričnih napolnitev za rekonstrukcijo slik je vpliv kompleksnosti slik na točnost rezultatov. Ker slika sestavljena iz naključnih vrednosti pikslov vizualno ni smiselna, domevamo,



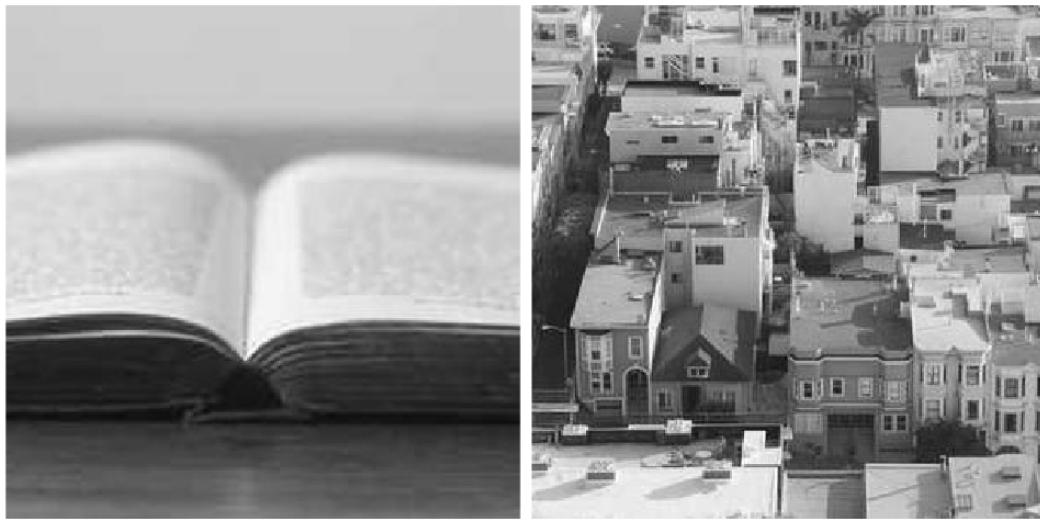
Slika 4.4: Časi izvajanja algoritmov. Na abscisni osi so deleži znanih podatkov slik.

da bodo slike s preprostimi motivi napolnjene bolje. Za namene testiranja je torej smiselno izbrati eno preprosto sliko in eno sliko z veliko različnimi motivi. V naših testiranjih uporabljamo sliko knjige in sliko mesta. Slike sta velikosti  $300 \times 300$  pikslov.

Kot smo pričakovali, so rezultati rekonstrukcije slike s preprostim motivom boljše. Prav tako lahko opazimo, da ima delež znanih vrednosti večji vpliv pri sliki s kompleksnim motivom. Napake z dodajanjem informacij torej hitreje padajo pri matrikah večjega ranga. Spomnimo se, da algoritma LMaFit in TNNM za svoje delovanje potrebujeta informacijo o rangu. Pri testiranju je bilo zato potrebno kompleksni slike podati večjo vrednost ranga, da sta lahko algoritma prišla do dobrih rezultatov.

Točnost algoritmov pa ostaja zelo podobna rekonstrukciji velike slike, torej z najboljšimi rezultati pridobljenimi z algoritmom TNNM, nato LMaFit in z najslabšimi rezultati algoritom SVT.

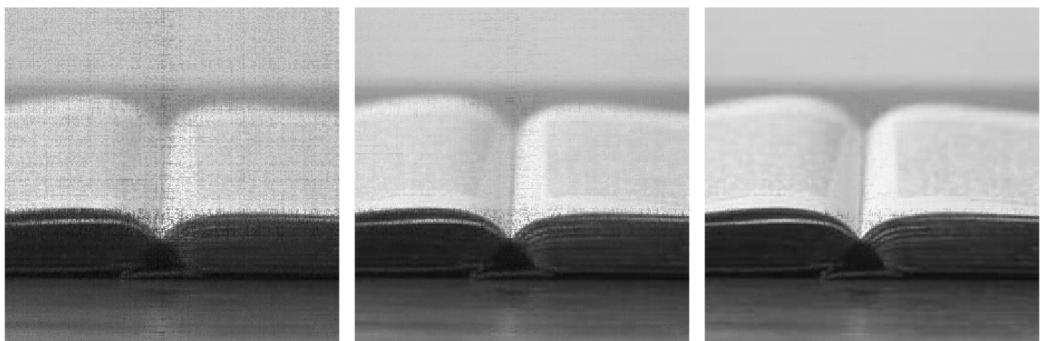
Na koncu preveriti,  
da so slike na smiselnem mestu



(a) Slika s preprostim motivom.

(b) Slika s kompleksnim motivom.

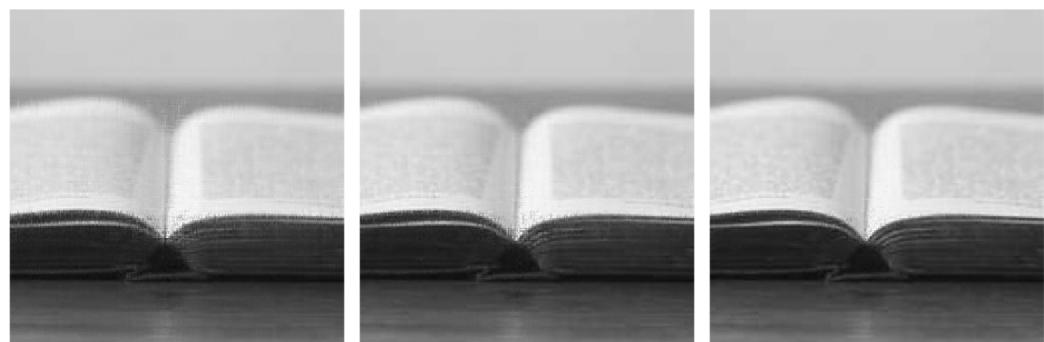
Slika 4.5: Vira slik: [5, 7].



Slika 4.6: Rekonstrukcija preprostega motiva z algoritmom SVT. Znane vrednosti slik so bile: 35% (leva), 45% (sredinska), 60% (desna).



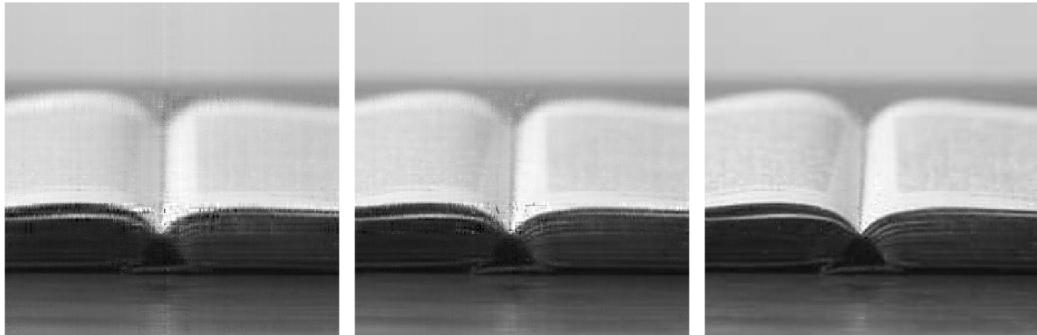
Slika 4.7: Rekonstrukcija kompleksnega motiva z algoritmom SVT. Znane vrednosti slik so bile: 35% (leva), 45% (sredinska), 60% (desna).



Slika 4.8: Rekonstrukcija preprostega motiva z algoritmom TNNM. Znane vrednosti slik so bile: 35% (leva), 45% (sredinska), 60% (desna).



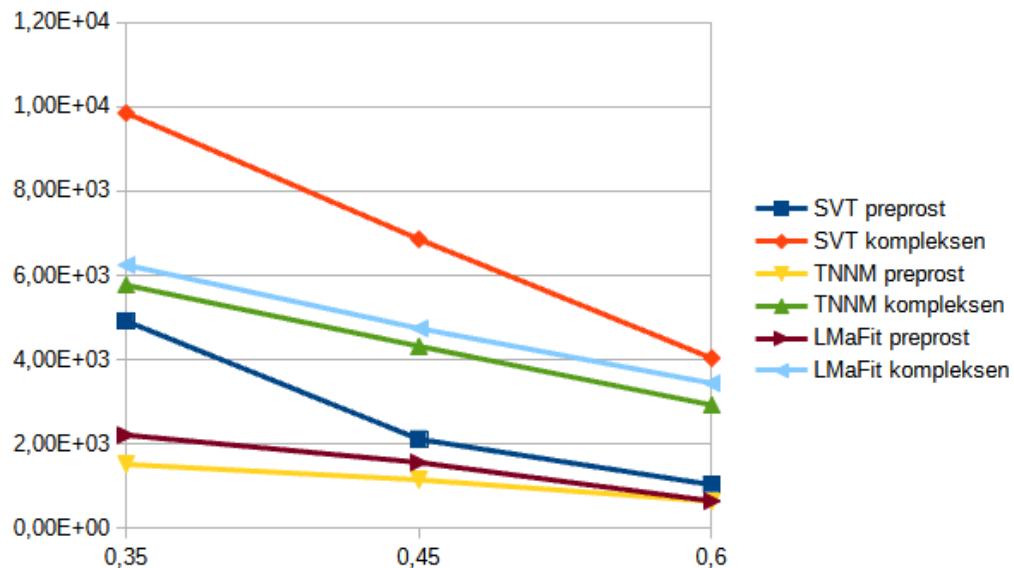
Slika 4.9: Rekonstrukcija kompleksnega motiva z algoritmom TNNM. Znane vrednosti slik so bile: 35% (leva), 45% (sredinska), 60% (desna).



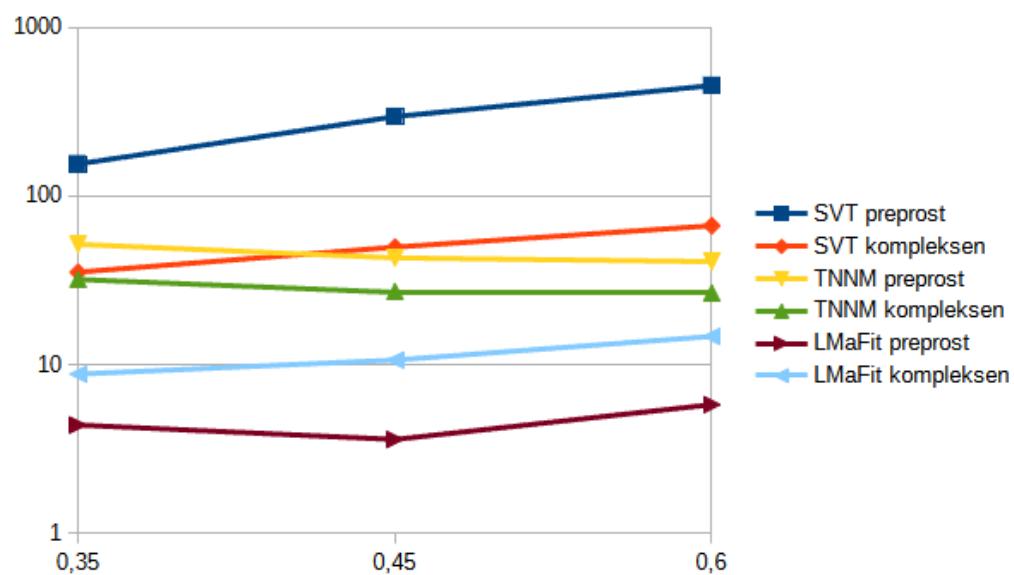
Slika 4.10: Rekonstrukcija preprostega motiva z algoritmom LMaFit. Znane vrednosti slik so bile: 35% (leva), 45% (sredinska), 60% (desna).



Slika 4.11: Rekonstrukcija preprostega motiva z algoritmom LMaFit. Znane vrednosti slik so bile: 35% (leva), 45% (sredinska), 60% (desna).



Slika 4.12: Graf napak algoritmov v Frobeniusovi normi. Na abscisni osi so deleži znanih podatkov slik.



Slika 4.13: Graf časov izvajanja algoritmov na logaritmični skali. Na abscisni osi so deleži znanih podatkov slik.

Časi izvajanja pa tu niso tako intuitivni. Prva glavna ugotovitev je, da algoritom SVT potrebuje veliko več časa pri preprostem motivu kot pri kompleksnem. V razdelku 3.3.1 smo omenili priporočen prag [2], vendar je v tem primeru ta vrnil slabe rezultate. S preizkušanjem smo prag večali in prišli do vrednosti 7200, ki je dala primerljive rezultate z ostalimi algoritmi. Prav tako je bilo za slike z manj znanimi vrednostmi potrebno korak zmanjšati, sicer algoritom ni konvergiral. V primeru preprostega motiva lahko pričakujemo, da bomo imeli manj zelo velikih singularnih vrednosti v primerjavi s kompleksnim. Zaradi tega se algoritom počasneje premika in dolgo išče rešitev. Iz tega sledi ugotovitev, da je algoritom SVT bolj smiselno uporabljati za kompleksne motive. Seveda pa bi lahko, zaradi manjše napake pri preprostem motivu prag nastavili nižje. S tem bi algoritom pospešili, vendar s tem dobili slabši rezultat.

Naslednja pomembna ugotovitev je, da delež znanih vrednosti različno vpliva na čas izvajanja. Tudi ta faktor je torej lahko pomemben pri izbiri algoritma za reševanje problema. Algoritom SVT potrebuje za rekonstrukcijo več časa, kadar ima poznanih več vrednosti, medtem ko se algoritmu TNNM z deležem znanih vrednosti čas izvajanja manjša. Algoritmu LMaFit težko določimo pravilo, saj je njegovo obnašanje odvisno od primera do primera. To je razvidno že iz slike 4.13, kjer je kompleksen motiv potreboval najmanj časa za rekonstrukcijo slike s 35% znanih podatkov, preprost pa s 45%. Ker pa algoritmom LMaFit začne iteracije z naključnima matrikama  $X$  in  $Y$ , je sam čas izvajanja lahko zelo različen med različnimi zagoni. Smiselno bi bilo razmisliiti o implementaciji algoritma, kjer bi začeli z več pari matrik  $X$  in  $Y$ , ter po nekaj iteracijah na vseh parih, algoritom nadaljevali zgolj na paru, ki najhitreje konvergira.

### 4.3 Rekonstrukcija barvnih slik

Naslednje smiselno vprašanje je, kako dobro algoritmi delujejo za rekonstrukcijo barvnih slik. Barvne slike so podane kot kombinacija barvnih kanalov

rdeče, zelene in modre barve, pri čemer je vsak kanal predstavljen z matriko vrednosti pikslov. V tem razdelku nas bo zanimalo, ali je bolje napolnjevati matrike vsakega barvnega kanala posebej, ali je bolje matrike kanalov združiti v večjo pravokotno matriko in napolniti to večjo matriko. V prvem primeru algoritom uporabimo trikrat, medtem ko v drugem definiramo veliko matriko sestavljenou kot

$$A = \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

kjer  $R$  predstavlja matriko z vrednostmi rdečega kanala,  $G$  matriko vrednosti zelenega kanala in  $B$  matriko vrednosti modrega kanala. Vsi testi v tej fazi so bili izvedeni na podatkih, kjer imamo poznanih 35% informacij.

Tip rekonstrukcije	Algoritem		
	SVT	TNNM	LMAFIT
Enojna rekonstrukcija	$1.50 \times 10^4$	$9.60 \times 10^3$	$1.10 \times 10^4$
Trojna rekonstrukcija	$1.66 \times 10^4$	$9.79 \times 10^3$	$1.06 \times 10^4$

Tabela 4.3: Napake algoritmov izračunane v Frobeniusovi normi.

Tip rekonstrukcije	Algoritem		
	SVT	TNNM	LMAFIT
Enojna rekonstrukcija	352s	124s	66s
Trojna rekonstrukcija	112s	100s	46s

Tabela 4.4: Časi do dosega zaustavitvenega pogoja.

Iz rezultatov v tabelah 4.3 in 4.4 opazimo, da obe metodi vrnete približno enako dobre rezultate, vendar pa je rekonstrukcija pri vseh testiranih algoritmih hitrejša, če ločene barvne kanale rekonstruiramo posebej. **Prav tako vizualno med rezultati težko opazimo razlike.** Zaključimo lahko, da je smiselno med sabo neodvisne podatke ločiti, ter jih obravnavati samostojno.

Rezultat je smiseln, saj nam iskanje podobnosti med nepodobnimi podatki poveča količino dela.

## 4.4 Vpliv podatka o rangu na rezultate

Spomnimo se, da algoritma TNNM in LMaFit za svoje izvajanje potrebuje informacijo o rangu (v nadaljevanju jo bomo imenovali *parameter*). V tem podpoglavlju bomo poskušali odgovoriti na vprašanje, kako pri obeh algoritmih ta informacija vpliva na rezultate in hitrost izvajanja. Za namene testiranja smo algoritom na isti sliki pognali večkrat, pri čemer smo postopoma povečevali rang. Ponovno smo teste izvajali na slikah mesta, z znanim deležem podatkov nastavljenim na 0.35.

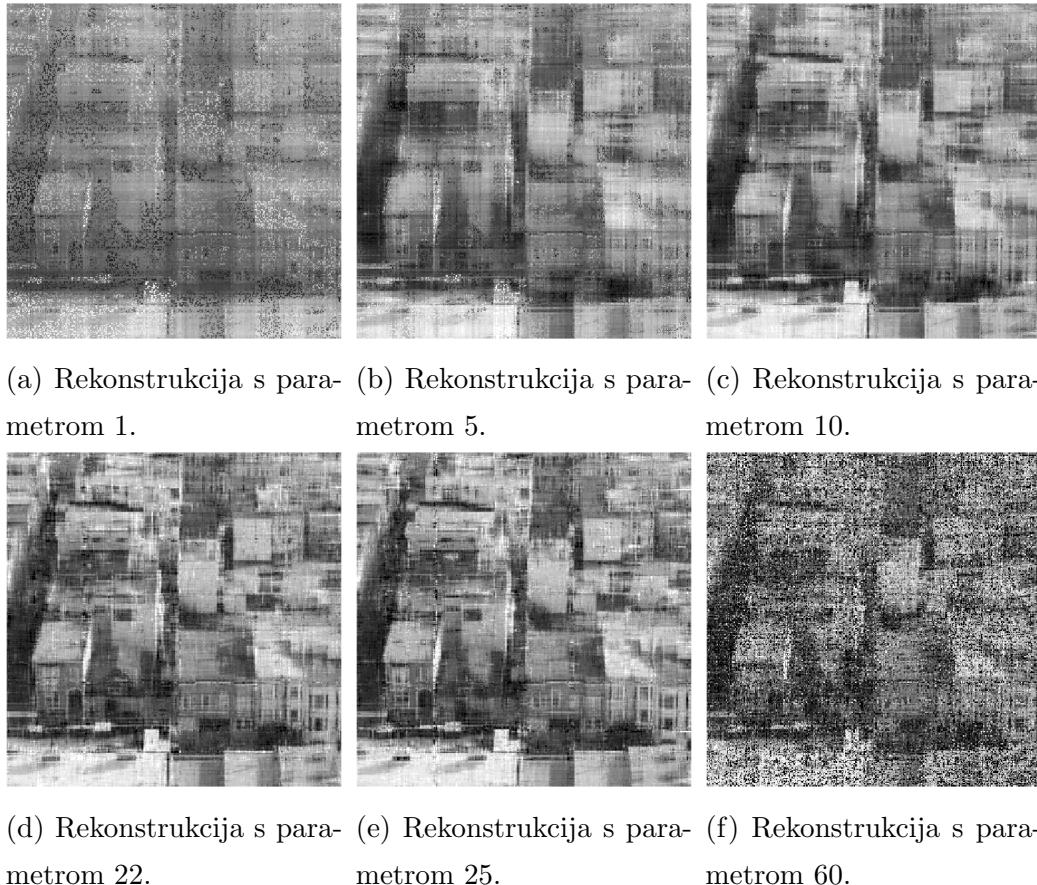
### 4.4.1 LMaFit

Algoritmom LMaFit smo testirali šestkrat, z rangom rezultata določenega na 1, 5, 10, 22, 25 in 60. Vrednost 22 je bila izbrana, ker je ob večkratnem zagonu programa pri različnih parametrih vrnila najboljši rezultat. Posledično sta bili vrednosti 25 in 60 izbrani z namenom opazovanja, kakšne rezultate pridobimo z nadaljnjam povečanjem ranga rekonstruirane matrike.

Parameter	Napaka ( $\ \cdot\ _F$ )	Čas izvajanja
1	$1.11 \times 10^4$	0.04s
5	$8.05 \times 10^3$	0.41s
10	$6.61 \times 10^3$	0.42s
22	$6.25 \times 10^3$	8.78s
25	$6.57 \times 10^3$	41.26s
60	$2.40 \times 10^4$	325.31s

Tabela 4.5: Rezultati rekonstrukcije algoritma LMaFit za različne parametre.

Na rekonstruiranih slikah lahko opazimo izboljševanje podrobnosti slike vse do parametra 22. Vidimo pa lahko tudi, da pride pri prevelikem pa-



rametru do preobrata. Rezultati se ponovno slabšajo, algoritem pa postane počasnejši. Od tod lahko zaključimo, da je pravilna izbira parametra ključna. Vredno je omeniti še, da za nekatere izbire ranga, algoritem ne konvergira. V primeru zgornje slike za izbiro ranga 30, algoritem LMaFit ni našel rešitve.

#### 4.4.2 TNNM

V tem razdelku predstavimo rezultate vpliva parametrov na delovanje algoritma TNNM. Zaradi večje časovne zahtevnosti, algoritem TNNM testiramo zgolj trikrat, na parametrih 1, 5 in 12. Za večje parametre je algoritem konvergiral zelo počasi, zaradi česar se testiranju teh izognemo. Tukaj se je pomembno spomniti, da parameter pri algoritmu TNNM ne določa samega ranga rezultata, vendar zgolj vpliva nanj. V algoritmu namreč omejimo samo

ranga nastopajočih matrik  $A_l$  in  $B_l$ , prek katerih dobimo rezultat (Algoritem 3.19). Zaradi tega je težje določiti pravilo, kateri parameter je najboljši.



(a) Rekonstrukcija s parametrom 1. (b) Rekonstrukcija s parametrom 5. (c) Rekonstrukcija s parametrom 12.

Parameter	Napaka ( $\ \cdot\ _F$ )	Čas izvajanja
1	$5.70 \times 10^3$	35.9s
5	$5.46 \times 10^3$	481s
12	$5.27 \times 10^3$	930s

Slika 4.16: Rezultati rekonstrukcije algoritma TNNM za različne parametre.

Sami rezultati vizualno med seboj delujejo podobni. Numerični izračun napak sicer pokaže, da se napaka počasi zmanjšuje s povečevanjem ranga, vendar pa se čas reševanja zelo hitro povečuje. Zato se je potrebno odločiti, kako dober rezultat potrebujemo in ali smo točnost pripravljeni kompenzirati z bistveno večjo časovno zahtevnostjo rekonstrukcije. Seveda pa velja povedati, da je že pri parametru 1 TNNM dosegel najboljše rezultate izmed vseh algoritmov, obravnavanih v tej diplomskej nalogi.

## 4.5 Rekonstrukcija slike z besedilom

V tem razdelku bomo preizkušali učinkovitost algoritmov na slikah, kjer se želimo znebiti nekega besedila na sliki. Gre za drugačno vrsto šuma, kjer

so namesto enakomerne razporeditve neznani podatki zgoščeni na določenem delu slike. V našem primeru je bil delež znanih podatkov enak 92%, kar je bistveno več kot v primerih, ki smo si jih ogledali doslej.



Slika 4.17: Slika z besedilom.



(a) Rekonstrukcija z algoritmom SVT. (b) Rekonstrukcija z algoritmom TNNM. (c) Rekonstrukcija z algoritmom LMaFit.

	Napaka (v $\ \cdot\ _F$ )	Čas izvajanja (s)
SVT	$4.64 \times 10^3$	674s
TNNM	$2.64 \times 10^3$	83.6s
LMaFit	$3.17 \times 10^3$	0.95s

Tabela 4.6: Rezultati rekonstrukcije različnih algoritmov.

Ponovno lahko opazimo, da je najboljši rezultat vrnil algoritmom TNNM. Pri algoritmu SVT lahko opazimo sledi besedila, zaradi česar je tudi sama

napaka pri tem algoritmu večja. Algoritem LMaFit ni skonvergiral za vrednosti parametra ranga večje od 16, zaradi česar je sam rezultat produkta matrik  $X$  in  $Y$  slab. Opazimo lahko, da po rekonstrukciji manjkajočih vrednosti večina slike izgleda pravilno, še vedno pa je mogoče opaziti obrise besedila. Rezultati teh testov nam pokažejo pomembnost vrste šuma, saj kljub velikemu deležu znanih podatkov, algoritmi večine podatkov ne morejo kakovostno uporabiti.

## 4.6 Primerjava rezultatov z algoritmom za reševanje Poissonovih enačb

Diplomsko delo [4] opisuje ter preizkusi postopek rekonstrukcije slik z reševanjem sistemov Poissonovih enačb. Ta način rekonstrukcije je v praksi pogosto uporabljen, sploh na zašumljenih slikah ter odstranjevanju motivov s slik. Smiselno je primerjati rezultate naših metod s sistemi za reševanje Poissonovih enačb. Poissonova enačba je definirana kot

$$-\frac{\partial^2 v(x, y)}{x^2} - \frac{\partial^2 v(x, y)}{y^2} = f(x, y).$$

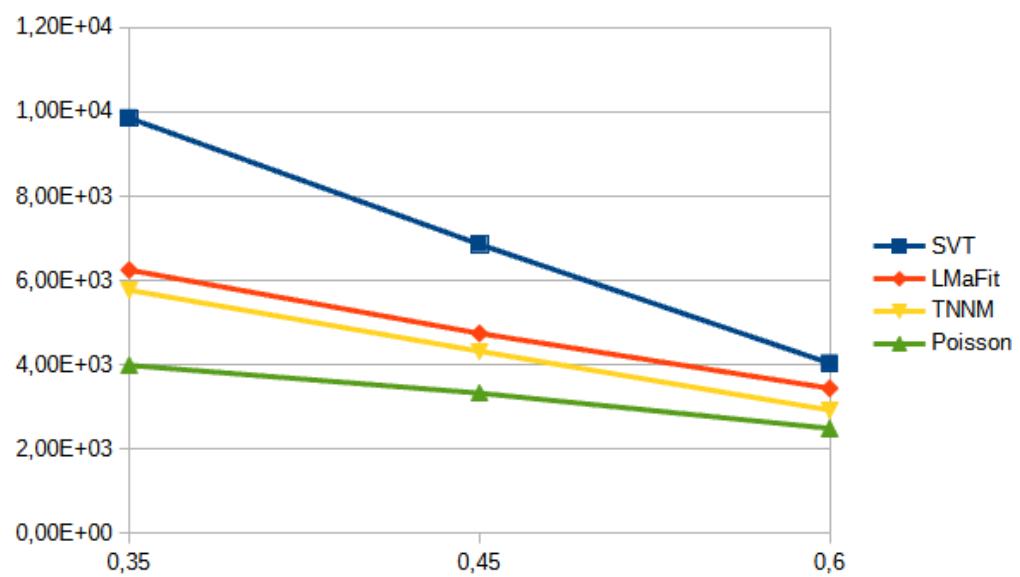
kjer je  $v(x, y)$  vrednost piksla v točki  $(x, y)$ . To je parcialna diferencialna enačba, ki se jo običajno rešuje z uporabo metode *končnih differenc*. Pri tej metodi odvode aproksimiramo z uporabo diferenčnega kvocienta kot

$$\begin{aligned} -\frac{\partial^2 v(x, y)}{x^2} &\approx \frac{2v_{i,j} - v_{i-1,j} - v_{i+1,j}}{h^2}, \\ -\frac{\partial^2 v(x, y)}{y^2} &\approx \frac{2v_{i,j} - v_{i,j-1} - v_{i,j+1}}{h^2}. \end{aligned}$$

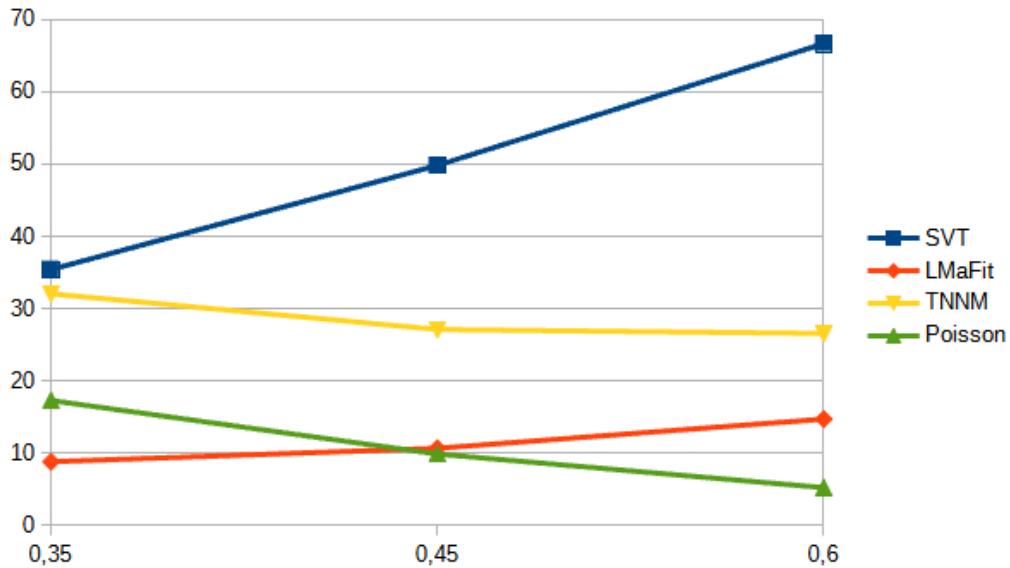
Z uporabo *Jacobijeve iteracije* lahko problem rešujemo iterativno, tako da neznane vrednosti v vsaki iteraciji posodobimo po formuli

$$u_{i,j}^{(k+1)} = \frac{1}{4}(u_{i-1,j}^{(k)} + u_{i,j-1}^{(k)} + u_{i+1,j}^{(k)} + u_{i,j+1}^{(k)})$$

Zaradi lastnosti vrstične diagonalne dominantnosti matrike Jacobijeve iteracije velja, da bo iteracija konvergirala. Spodaj si lahko ogledamo rezultate rekonstrukcij, zašumljene slike mesta.

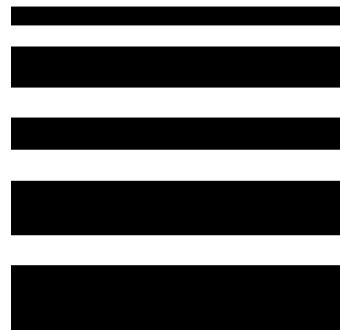


Slika 4.20: Napaka rekonstrukcij slike mesta v Frobeniusovi normi. Na abscisni osi so deleži znanih podatkov slik.



Slika 4.21: Čas izvajanja rekonstrukcije slike mesta. Na abscisni osi so deleži znanih podatkov slik.

Vidimo, da je algoritem tako hitrejši, kot bolj točen. Vendar je pri primerjavi potrebno upoštevati, da se tak algoritem zanaša na lokalno podobnost podatkov, tj. bližnje točke imajo podobne vrednosti barvnih kanalov. Pri problemu minimizacije ranga matrik pa se na take podobnosti ne moremo zanašati. Omenili smo že, da lahko algoritem uporabljamo v priporočilnih sistemih. V takem primeru ne moremo uporabljati sosednosti, saj imata lahko uporabnika v sosednjih vrsticah povsem različne preference. Prav tako si je lahko zamisliti sliko, kjer bi reševanje Poissonove enačbe vrnilo slab rezultat. Tak primer je lahko preprosta dvobarvna slika, sestavljena iz več pasov. Očitno je, da ima originalna slika rang 1.

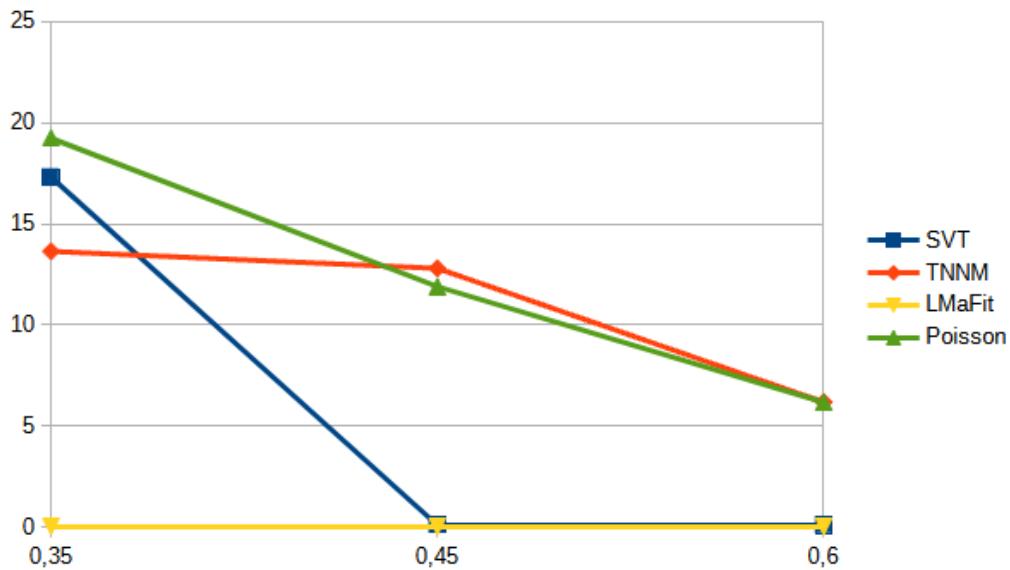


Slika 4.22: Dvobarvna slika.



(a) Rekonstrukcija na 35% znanih podatkih. (b) Rekonstrukcija na 45% znanih podatkih. (c) Rekonstrukcija na 60% znanih podatkih.

Slika 4.23: Rekonstrukcija z uporabo algoritma za reševanje Poissonovih enačb



Slika 4.24: Čas izvajanja rekonstrukcije dvobarvne slike. Na abscisni osi so deleži znanih podatkov slik.

Medtem ko so algoritmi SVT, TNNM in LMaFit sliko rekonstruirali točno, je algoritem za reševanje Poissonove enačbe, kot pričakovano, tu imel več težav. Prav tako je algoritem za reševanje v večini primerov potreboval več časa.

## 4.7 Povzetek ugotovitev

Tekom poglavja smo predstavili in interpretirali rezultate številnih testov algoritmov ter odgovarjali na različna vprašanja o rekonstrukciji slike. Postopoma smo predstavljali ugotovitve, ki jih v tem razdelku povzamemo.

Najprej smo algoritme testirali na veliki sliki, ter ugotovili, da algoritem ASD na zašumljenih slikah ne deluje najbolje. Testiranju tega algoritma smo se v nadaljnjih razdelkih zato izogibali. V razdelku 4.2 smo testirali razliko v rekonstrukciji preproste in kompleksne slike. Ugotovili smo, da je rekonstrukcija preprostega motiva pri vseh algoritmih točnejša. Videli smo tudi, da je pri algoritmu SVT pomembna izbira praga, saj je ob istem pragu

preprost motiv potreboval veliko več časa do konvergencije. V tem razdelku smo opisali tudi, kako odstotek znanih podatkov vpliva na čas izvajanja. Videli smo, da algoritom TNNM konvergira hitreje, kadar ima več poznanih podatkov, medtem ko se algoritom SVT upočasni. Omenili smo tudi, da je algoritmu LMaFit težko določiti pravilo o času izvajanja. Ta ugotovitev je pomembna, kadar izbiramo algoritem za rekonstrukcijo.

V razdelku 4.3 smo preverili, kako rekonstruirati barvne slike. Ugotovili smo, da je zaradi časa izvajanja bolje ločiti neodvisne podatke (različne barvne kanale) in jih rekonstruirati posebej. Preverili smo tudi pomembnost parametra, povezanega z rangom algoritmov (razdelek 4.4). Ta parameter uporabljalata algoritom TNNM in LMaFit. Videli smo, da je za dobre rezultate dobra izbira parametra ključna. Prednost algoritma SVT je torej tudi, da takega parametra ne potrebuje. Če o rangu nezašumljene matrike ne vemo ničesar, je torej smiselno razmisljiti o uporabi tega algoritma. V okviru razdelka 4.5 smo preizkusili še drugačno vrsto šuma, saj smo iz slike odstranjevali besedilo. Videli smo, da kljub velikem številu znanih podatkov, rezultati niso bili preveč dobri, saj je del besedila po rekonstrukciji še vedno viden. S tem smo pokazali, da tudi sama vrsta šuma vpliva na rekonstrukcijo.

Poglavlje smo zaključili z razdelkom 4.6, kjer smo primerjali rekonstrukcijo naših algoritmov z algoritmom za reševanje Poissonovih enačb. Opazili smo, da je ta metoda boljša, vendar pokazali tudi primer, kjer zanašanje na lokalno podobnost vrne slab rezultat. Rekonstrukcija slik je bila v okviru diplomskega dela izbrana, ker smo lahko rezultate ocenili tako numerično kot vizualno. Seveda pa so opisani algoritmi bolj uporabni na drugih področjih, npr. priporočilnih sistemih, kjer so lahko podatki v sosednjih vrsticah povsem neodvisni.



# Poglavlje 5

## Zaključek

Tekom diplomskega dela smo si ogledali 5 različnih algoritmov, za reševanje matričnih napolnitev. Komentirali smo njihovo delovanje ter se spoznali z definicijami, ki so pomembne in pogoste na tem področju. Nato smo pregledali in primerjali rezultate algoritmov, ter jih interpretirali, glede na delovanje algoritmov. Poskušali smo odgovoriti na vprašanja, ki so pri rekonstrukciji pomembna. Prav tako smo algoritme primerjali z drugim popularnim algoritmom za rekonstrukcijo, predstavili razliko v njegovem delovanju, ter podali primer, kjer algoritmi matričnih napolnitev vrnejo boljši rezultat.

Ker je področje matričnih napolnitev široko, bi lahko delo diplomske naloge še razširili. Algoritmi TNNM, LMaFit in ASD imajo še druge, bolj napredne različice implementacije, ki bi jih lahko preizkusili. Seveda pa obstajajo tudi drugi algoritmi, katerih se v tem delu nismo dotaknili. V razdelku 4.2 smo omenili, da bi lahko algoritom LMaFit razširili tako, da bi začeli z več pari matrik  $X$  in  $Y$ . Prav tako pa bi bilo algoritme smiselno preizkusiti še na drugih podatkih, ki niso slike. Ker imajo slike lokalne podatke podobne, na kar pa se pri problemu matričnih napolnitvah ne moremo zanašati, obstajajo boljši algoritmi za rekonstrukcijo slik. Literatura pogosto testira algoritme na naključno generiranih podatkih.



# Literatura

- [1] Jonathan Bean. *Unsplash*. URL: <https://unsplash.com/photos/5ulmc8IHdLc> (pridobljeno 9. 2. 2023).
- [2] Jian-Feng Cai, Emmanuel J. Candès in Zuowei Shen. *A Singular Value Thresholding Algorithm for Matrix Completion*. 2008. arXiv: 0810.3286 [math.OC].
- [3] Emmanuel J. Candès in Benjamin Recht. “Exact Matrix Completion via Convex Optimization”. V: *Foundations of Computational Mathematics* 9.6 (dec. 2009), str. 717–772. ISSN: 1615-3383. DOI: 10.1007/s10208-009-9045-5. URL: <https://doi.org/10.1007/s10208-009-9045-5>.
- [4] Blaž Erzar. “Inkrementalna rekonstrukcija slike iz razpršenih podatkov”. Diplomsko delo. Univerza v Ljubljani, 2023. URL: <https://repozitorij.uni-lj.si/IzpisGradiva.php?lang=slv&id=144588>.
- [5] Alejandro Escamilla. *Unsplash*. URL: <https://unsplash.com/photos/cZhUxIQjILg> (pridobljeno 9. 2. 2023).
- [6] Maryam Fazel. “Matrix rank minimization with applications”. Doktorska disertacija. Stanford, CA: Stanford University, mar. 2002.
- [7] Gabe. *Unsplash*. URL: [https://unsplash.com/photos/bIZrEK-Z\\_cI](https://unsplash.com/photos/bIZrEK-Z_cI) (pridobljeno 9. 2. 2023).

- [8] Yao Hu in sod. “Fast and Accurate Matrix Completion via Truncated Nuclear Norm Regularization”. V: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.9 (2013), str. 2117–2130. DOI: 10.1109/TPAMI.2012.271.
- [9] Charles R Johnson. “Matrix completion problems: a survey”. V: *Matrix theory and applications*. Zv. 40. 1990, str. 171–198.
- [10] Luong Trung Nguyen, Junhan Kim in Byonghyo Shim. “Low-Rank Matrix Completion: A Contemporary Survey”. V: *IEEE Access* 7 (2019), str. 94215–94237. DOI: 10.1109/ACCESS.2019.2928130.
- [11] Jos F. Sturm. “Using SeDuMi 1.02, A MATLAB toolbox for optimization over symmetric cones”. V: *Optimization Methods and Software* 11.1-4 (1999), str. 625–653. DOI: 10.1080/10556789908805766. URL: <https://doi.org/10.1080/10556789908805766>.
- [12] Jared Tanner in Ke Wei. “Low rank matrix completion by alternating steepest descent methods”. V: *Applied and Computational Harmonic Analysis* 40.2 (2016), str. 417–429. ISSN: 1063-5203. DOI: <https://doi.org/10.1016/j.acha.2015.08.003>. URL: <https://www.sciencedirect.com/science/article/pii/S1063520315001062>.
- [13] Zaiwen Wen, Wotao Yin in Yin Zhang. “Solving a low-rank factorization model for matrix completion by a nonlinear successive over-relaxation algorithm”. V: *Mathematical Programming Computation* 4 (dec. 2012). DOI: 10.1007/s12532-012-0044-1.