

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 . . . . .	17
3.6	LMaFit . . . . .	19
<b>4</b>	<b>Rezultati</b>	<b>21</b>
4.1	Velika črno-bela slika . . . . .	22
4.2	Vpliv kompleksnosti slik na rekonstrukcijo . . . . .	26
4.3	Rekonstrukcija barvnih slik . . . . .	32
4.4	Vpliv podatka o rangu na rezultate . . . . .	33
4.5	Rekonstrukcija slike z besedilom . . . . .	36

4.6 Primerjava rezultatov z algoritmom za reševanje Poissonovih enačb . . . . .	37
<b>5 Zaključek</b>	<b>43</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 ponostaviti, 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. V tej diplomski nalogi se osredotočamo na problem razreševanja neznanih pikslov v slikah.

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 konvergence programov.

## 1.3 Struktura diplomskega dela

Poglavlje 2 pregleda že napisana dela o samem reševanju matričnih napolnitev 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 teoretične ozadje algoritmov, ki rešujejo problem 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 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  $\mathcal{D}_\tau : \mathbb{R}^{n_1 \times n_2} \rightarrow \mathbb{R}^{n_1 \times n_2}$  definiran kot

$$\mathcal{D}_\tau(A) := U\mathcal{D}_\tau(\Sigma)V^T, \quad \mathcal{D}_\tau(\Sigma) = \text{diag}(\max(\sigma_i - \tau, 0)), \quad (3.1)$$

imenujemo **operator praga**.[2]

5. Matriko  $M \in \mathbb{R}^{n_1 \times n_2}$ , ki ima določene samo nekatere elemente, imenujemo **delno določena matrika**.

6. Sled matrike  $A \in \mathbb{R}^{n \times n}$  definiramo kot

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

ozziroma z besedami, kot vsoto diagonalnih elementov kvadratne matrike.

7. Skalarni produkt  $\langle A, B \rangle$  je definiran kot

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

8. **Frobeniusova norma** je za matriko  $A \in \mathbb{R}^{n_1 \times n_2}$  definirana kot

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

Definicija je ekvivalentna zapisu

$$\|A\|_F = \sqrt{\langle A, A \rangle}$$

9. **Nuklearna norma** je definirana kot

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

pri čemer  $\sigma_i(A)$  označuje  $i$ -to največjo singularno vrednost matrike  $A$ .

10. Oznaka  $A \succeq 0$  označuje, da je matrika  $A \in \mathbb{R}^{n \times n}$  pozitivno semidefinitna. Velja  $A \succeq 0 \iff x^T A x \geq 0$  za vsak  $x \in \mathbb{R}^n$ . Z drugimi besedami, matrika  $A$  je pozitivno semidefinitna, če so vse njene lastne vrednosti nenegativne.
11. Za subgradient  $c$  v točki  $x_0$  konveksne funkcije  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  velja

$$\forall x : f(x) - f(x_0) \geq c^T(x - x_0),$$

kjer  $c \in \mathbb{R}^n$ . Z drugimi besedami,  $c$  je subgradient na točko  $x_0$ , če je premica na točko  $x_0$  s smernim koeficientom  $c$  vedno pod ali na funkciji  $f$ .

### 3.2 Algoritem minimizacije nuklearne norme

Ker je minimizacija ranga matrike NP-poln problem [6], so se razvile druge metode, ki samo kompleksnost problema zmanjšujejo. Minimizacija nuklearne norme (NNM) uporabi idejo, da je rang matrike povezan z **nuklearno normo** matrike.

Dokazano je bilo [6], da nuklearna norma predstavlja konveksno ovojnico funkcije ranga. **Konveksna ovojnica funkcije  $f : \mathbb{R}^{n_1 \times n_2} \rightarrow \mathbb{R}$  je definirana kot**

$$\tilde{f}(x) = \sup\{g(x) \mid g \text{ je konveksna in } \forall x : g(x) \leq f(x)\}.$$

poglej razliko convex hull in lower envelope

Problem minimizacije nuklearne norme je možno pretvoriti v optimizacijski problem iz področja semidefinitnega programiranja (SDP), za katere obstajajo različna orodja za reševanje, na primer SeDuMi [11].

**Standardna oblika semidefinitnega programa** je podana kot

$$\begin{aligned} & \min_Y \quad \langle C, Y \rangle \\ & \text{pri pogojih} \quad \langle A_k, Y \rangle = b_k, \quad k = 1, \dots, l \\ & \quad Y \succeq 0 \end{aligned} \tag{3.2}$$

Kjer je  $C \in \mathbb{R}^{n_1 \times n_2}$  dana matrika,  $\{A_k \in \mathbb{R}^{n_1 \times n_2}\}$  in  $\{b_k\}$  pa množici matrik in števil.

Problem minimizacije nuklearne norme lahko zapišemo kot

$$\begin{aligned} & \min_{X,t} \quad t \\ & \text{pri pogojih} \quad \|X\|_* \leq t, \\ & \quad \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M), \end{aligned} \tag{3.3}$$

kjer je  $M \in \mathbb{R}^{n_1 \times n_2}$  zašumljena matrika,  $\Omega$  pa množica urejenih parov, ki predstavljajo mesta, kjer vrednosti matrike poznamo. Dokazano je bilo [6], da za matriko  $X \in \mathbb{R}^{n_1 \times n_2}$  in  $t \in \mathbb{R}$  velja  $\|X\|_* \leq t \iff \exists W_1 \in \mathbb{R}^{n_1 \times n_1}, W_2 \in \mathbb{R}^{n_2 \times n_2}$  tako da

$$\begin{aligned} Y &= \begin{bmatrix} W_1 & X \\ X^T & W_2 \end{bmatrix}, \\ Y &\succeq 0, \quad \text{Tr}(Y) \leq 2t \end{aligned}$$

Minimizacijski problem (3.3) lahko tako redefiniramo kot

$$\begin{aligned} & \min_{Y,t} \quad 2t \\ & \text{tako da} \quad \text{Tr}(Y) \leq 2t, \\ & \quad Y \succeq 0, \\ & \quad \langle Y, A_{ab} \rangle = c_{ab} \end{aligned}$$

kjer je  $A_{ab} \in \mathbb{R}^{(n_1+n_2) \times (n_1+n_2)}$  matrika v množici  $A$ . Velja  $A_{ab} \in A \iff (a, b) \in \Omega$ . Matrika  $A_{ab}$  ima vse elemente ničelne, razen na mestu  $(a, n_1 + b)$ , kjer ima vrednost 1. Podobno je definiran tudi  $c_{ab} \in \mathbb{R}$ , ki ima vrednost  $M_{ab}$ .

Ker velja

$$\langle A, B \rangle = \sum_i^{n_1} \sum_j^{n_2} a_{ij} b_{ij}$$

je lahko videti, da je tak pogoj smiselen. Programi za reševanje SDP pa lahko tako obliko že sprejmejo. [10]

### 3.3 Algoritem praga singularnih vrednosti

Algoritem praga singularnih vrednosti, oziroma v nadaljevanju SVT uporabi idejo, da imamo pri matrikah z majhnim rangom nekaj velikih singularnih vrednosti, ostale pa blizu 0. Za svoje delovanje uvede dva nova pomembna koncepta, prvi je premik, drugi pa prag, potreben za uporabo operatorja  $\mathcal{D}_\tau$  (3.1). Algoritem temelji na iteraciji

$$\begin{cases} X^k = \mathcal{D}_\tau(Y^{k-1}) \\ Y^k = Y^{k-1} + \delta_k \mathcal{P}_\Omega(M - X^k) \end{cases}$$

kjer je  $\tau > 0$  izbran prag,  $\delta_k$  izbran premik,  $X^0 = 0 \in \mathbb{R}^{n_1 \times n_2}$  ter  $Y^0 = 0 \in \mathbb{R}^{n_1 \times n_2}$ . [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

$$f_\tau(X) = \tau \|X\|_* + \frac{1}{2} \|X\|_F^2 \quad (3.4)$$

Problem lahko tako zapišemo kot

$$\begin{array}{ll} \min & f_\tau(X) \\ \text{pri pogojih} & \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M) \end{array} \quad (3.5)$$

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

Definirajmo Lagrangeovo funkcijo

$$\mathcal{L}(x, \lambda) = f(x) + \lambda \cdot g(x),$$

kjer je  $f(x)$  (3.5) funkcija, ki jo minimiziramo, pod pogojem, da velja  $g(x) = 0$ . Naš problem tako prevedemo v

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

S pomočjo tako imenovanega Uzawoega algoritma [2], pa lahko problem pretvorimo v iterativni algoritem.

$$X^k = \arg \min_X \mathcal{L}(X^k, Y^{k-1}) \quad (3.6)$$

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

Izkaže se, da je rešitev (3.6) enaka  $\mathcal{D}_\tau(Y^{k-1})$ .

**Izrek 1.**

$$\arg \min_X \mathcal{L}(X, Y) = \arg \min_X \tau \|X\|_* + \frac{1}{2} \|X - Y\|_F^2 \quad (3.8)$$

*Dokaz.*

$$\begin{aligned} & \arg \min_X \tau \|X\|_* + \frac{1}{2} \|X - Y\|_F^2 \\ & \arg \min_X \tau \|X\|_* + \frac{1}{2} \langle X - Y, X - Y \rangle \\ & \arg \min_X \tau \|X\|_* + \frac{1}{2} (\|X\|_F^2 - 2 \langle X, Y \rangle + \|Y\|_F^2) \\ & \arg \min_X \tau \|X\|_* + \frac{1}{2} \|X\|_F^2 - \langle X, \mathcal{P}_\Omega(Y) \rangle \\ & \arg \min_X \tau \|X\|_* + \frac{1}{2} \|X\|_F^2 + \text{Tr}(-\mathcal{P}_\Omega(X)Y^T) + \text{Tr}(\mathcal{P}_\Omega(M)Y^T) \\ & \arg \min_X \tau \|X\|_* + \frac{1}{2} \|X\|_F^2 + \text{Tr}(\mathcal{P}_\Omega(M - X)Y^T) \\ & \arg \min_X \tau \|X\|_* + \frac{1}{2} \|X\|_F^2 + \langle Y, \mathcal{P}_\Omega(M - X) \rangle = \arg \min_X \mathcal{L}(X, Y) \end{aligned} \quad (3.9)$$

kjer je korak (3.9) dovoljen, saj velja  $\mathcal{P}_\Omega(Y^k) = Y^k$  za vse  $k \in \mathbb{N}$ . To je razvidno iz koraka (3.7), velja pa  $Y^0 = 0$ . Prav tako velja  $\langle X, \mathcal{P}_\Omega(Y) \rangle = \langle \mathcal{P}_\Omega(X), Y \rangle$ , 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

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

□

**Izrek 2.**

$$\mathcal{D}_\tau(Y) = \arg \min_X \left\{ \frac{1}{2} \|X - Y\|_F^2 + \tau \|X\|_* \right\} \quad (3.10)$$

*Dokaz.* Ker je  $h(X) := \frac{1}{2}\|X - Y\|_F^2 + \tau\|X\|_*$  strogo konveksna funkcija, lahko za subgradient  $Z$  v točki  $X_0$  rečemo, da velja  $\forall X : f(X) \geq f(X_0) + \langle Z, X - X_0 \rangle$ . Ali drugače povedano, premica na točko  $X_0$  s smernim koeficientom  $Z$  je povsod na ali pod funkcijo  $h(X)$ . To velja po sami definiciji, saj je to zahtevan pogoj subgradienta v neki točki.

Pri iskanju minimuma torej iščemo tako točko  $X'$ , da bo subgradient v točki  $X'$  enak 0. Problem sedaj zapišemo kot  $0 \in X' - Y + \tau\partial\|X'\|_*$ . Izkaže se, da je množica subgradientov nuklearne norme definirana kot

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

kjer  $U\Sigma V^T$  predstavlja SVD razcep matrike  $X$ . [2]

Cilj dokaza je pokazati, da velja  $X' = \mathcal{D}_\tau(Y)$ . Najprej razčlenimo SVD razcep matrike  $Y$  kot

$$Y = U_0\Sigma_0V_0^T + U_1\Sigma_1V_1^T$$

kjer  $U_0, \Sigma_0$  in  $V_0$  predstavljajo lastne vrednosti ter njihove lastne vektorje večje od  $\tau$ ,  $U_1, \Sigma_1$  in  $V_1$  pa tiste manjše od  $\tau$ . Ustrezno je torej pokazati, da velja

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

Gre preprosto za drugačen zapis operatorja  $\mathcal{D}_\tau(Y)$ . Če zapis vstavimo v prejšnji podan pogoj dobimo

$$\begin{aligned} 0 &= X' - Y + \tau\partial\|X'\|_* \\ Y - X' &= \tau(U_0V_0^T + W) \end{aligned}$$

primerna izbira za  $W = \tau^{-1}U_1\Sigma_1V_1^T$ , saj

$$\begin{aligned} Y - X' &= U_0\Sigma_0V_0^T + U_1\Sigma_1V_1^T - U_0(\Sigma_0 - \tau I)V_0^T \\ &= U_0V_0^T(\Sigma_0 - \Sigma_0 + \tau I) + U_1\Sigma_1V_1^T \\ &= \tau U_0V_0^T + U_1\Sigma_1V_1^T \end{aligned}$$

in

$$\begin{aligned} \tau(U_0V_0^T + W) &= \tau(U_0V_0^T + \tau^{-1}U_1\Sigma_1V_1^T) \\ &= \tau U_0V_0^T + U_1\Sigma_1V_1^T \end{aligned}$$

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  $\|W\|_2 \leq 1$ . S tem smo pokazali, da  $Y - X' \in \tau\partial\|X'\|_*$ .  $\square$

Po trditvi lahko sedaj zapišemo algoritom (3.6) - (3.7) kot [2]

$$\begin{cases} X^k = \mathcal{D}_\tau(Y^{k-1}) \\ Y^k = Y^{k-1} + \delta_k \mathcal{P}_\Omega(M - X^k) \end{cases}$$

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

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

Medtem, ko so koraki v samem algoritmu definirani kot množica korakov, smo v okviru rezultatov diplomske naloge, prvotno za premik uporabljali konstanto, ter korak nastavili na

$$\delta = 1.2 \frac{n_1 n_2}{m}$$

po priporočilih [2].

Prav tako članek [2] navaja, da je za matrike velikosti  $\mathbb{R}^{n \times n}$  smiselno nastaviti  $\tau = 5n$ , vendar sem v moji implementaciji zaradi posploševanja na nekvadratne matrike, za matrike velikosti  $\mathbb{R}^{n_1 \times n_2}$  parameter nastavil na

$$\tau = 5 \frac{n_1 + n_2}{2}$$

Medtem, ko so taki parametri dobri za večje matrike, bomo v razdelku 4.2 videli, da moramo pri manjših matrikah pogosto zmanjšati premik in povečati prag.

### 3.4 Algoritem minimizacije prirezane nuklearne norme

Že samo ime nam pove, da bo algoritem minimizacije prirezane nuklearne norme, oziroma TNNM podoben algoritmu NNM. Ideja tega algoritma pa je, da uporabimo dodatno informacijo  $r \in \mathbb{N}$  o rangu originalne, nezašumljene matrike.

Sam algoritem uvede tako imenovano  **$r$ -prirezano nuklearno normo**, ki je za matriko  $X \in \mathbb{R}^{n_1 \times n_2}$  definirana kot vsota  $\min(n_1, n_2) - r$  najmanjših singularnih vrednosti

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

TNNM rešuje problem [8]

$$\begin{aligned} \min_X \quad & \|X\|_r \\ \text{pri pogojih} \quad & \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M) \end{aligned} \tag{3.11}$$

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

Problem (3.11) je ekvivalenten

$$\begin{aligned} \min_X \quad & \|X\|_* - \sum_{i=1}^r \sigma_i \\ \text{pri pogojih} \quad & \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M) \end{aligned}$$

Za nadaljnje korake bomo potrebovali naslednji izrek

**Izrek 3.** Za  $X \in \mathbb{R}^{n_1 \times n_2}$ ,  $A \in \mathbb{R}^{r \times n_1}$ ,  $B \in \mathbb{R}^{r \times n_2}$  in  $r \in \mathbb{N}$ , pri pogojih  $r \leq \min(n_1, n_2)$ ,  $AA^T = I_r$ ,  $BB^T = I_r$  velja:

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

*Dokaz.* Za dokaz uporabljamo von Neumannovo neenakost sledi [8], s katero lahko zapišemo

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

Enakost  $\mathrm{Tr}(AXB^T) = \mathrm{Tr}(XB^TA)$  sledi iz dejstva, da je sled produkta matrik invariantna pod cikličnimi permutacijami.

Po definiciji lahko singularne vrednosti matrike  $Y$  najdemo tako, da najdemo korene nenegativnih lastnih vrednosti matrike  $Y^TY$ . Tako lahko povemo, da so singularne vrednosti matrike  $B^TA$  enake lastnim vrednostim matrike  $A^TBB^TA$ . Izraz razpišemo v

$$A^TBB^TA = A^TI_rA = A^TA$$

Ker pa velja, da imata matriki  $XY$  in  $YX$  enake neničelne lastne vrednosti, ter vemo da  $AA^T = I_r$ , lahko povemo, da ima produkt  $B^TA$   $r$  singularnih vrednosti enakih 1, saj ima  $I_n$   $n$  lastnih vrednosti enakih 1. Tako lahko sedaj razpišemo izraz

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

Ugotovili smo, da velja

$$\mathrm{Tr}(AXB^T) \leq \sum_{i=1}^r \sigma_i(X) \tag{3.12}$$

□

**Izrek 4.** Za SVD razcep matrike  $X = U\Sigma V^T$ , ter matriki  $A = (u_1, \dots, u_r)^T$  in  $B = (v_1, \dots, v_r)^T$ , kjer je  $u_i$   $i$ -ti stolpec matrike  $U$  ter  $v_i$   $i$ -ti stolpec matrike  $V$ , velja

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

*Dokaz.*

$$\begin{aligned}
 \text{Tr}(AXB^T) &= \text{Tr}((u_1, \dots, u_r)^T X (u_1, \dots, u_r)^T) \\
 &= \text{Tr}((u_1, \dots, u_r)^T U \Sigma V^T (u_1, \dots, u_r)^T) \\
 &= \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} \Sigma \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} \\
 &= \text{Tr}\left(\begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_r & 0 \\ & & & \ddots & 0 \end{bmatrix}\right) = \sum_{i=1}^r \sigma_i(X) \quad (3.13)
 \end{aligned}$$

□

Z združitvijo dokazov (3.12) in (3.13) zapišemo

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

Torej je optimizacijski problem ekvivalenten

$$\begin{aligned}
 \min_X \quad & \|X\|_* - \max_{AA^T=I, BB^T=I} \text{Tr}(AXB^T) \\
 \text{pri pogojih} \quad & \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M)
 \end{aligned}$$

Glede na vse ugotovitve, nato nastavimo iterativni algoritmom, tako da, izračunamo  $X^0 = \mathcal{P}_\Omega(M)$ . V  $i$ -ti iteraciji izračunamo  $A^i$  in  $B^i$ , tako da izračunamo SVD razcep  $X^i = U \Sigma V^T$ , ter  $A$  nastavimo kot prvih  $r$  stolpcov matrike  $U$ ,  $B$  pa kot prvih  $r$  stolpcov matrike  $V$ .  $X^{i+1}$  lahko sedaj izračunamo kot [8]

$$\begin{aligned}
 \min_X \quad & \|X\|_* - \text{Tr}(A^i X (B^i)^T) \\
 \text{tako da} \quad & \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M)
 \end{aligned}$$

Če minimizacijo še malo obrnemo, pa lahko problem rešujemo z uporabo algoritma ADMM.

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

$$\min_X \|X\|_* - \text{Tr}(A^i W (B^i)^T) \quad (3.14)$$

tako da  $W = X$ ,  $\mathcal{P}_\Omega(W) = \mathcal{P}_\Omega(M)$

Ta problem ponovno zapišemo s pomočjo Lagrangeove funkcije, le da algoritmom ADMM definira še regularizacijski parameter  $\beta$ . [8]

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

Opazimo lahko, da funkcija definira zgolj pogoj  $X = W$ . Videli bomo, da algoritmom pogoj  $\mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M)$  definira posredno, s popravljanjem znanih vrednosti (korak (3.15) spodaj).

Matriko  $X^{k+1}$  ponovno definiramo kot

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

Z ignoriranjem konstantnih členov, pa lahko tako zapišemo

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

Z namenom faktorizacije dodamo konstantne člene.

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

Po enakkosti v izreku ... ali po izreku ... Po izreku 2 pa tako lahko zapišemo

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

Matriko  $W$  podobno izračunamo kot

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

Lahko je videti, da velja

$$W^{k+1} = X^{k+1} + \frac{1}{\beta}(A_l^T B_l + Y_k)$$

Zaradi pogoja  $\mathcal{P}_\Omega(W) = \mathcal{P}_\Omega(M)$  pa tiste elemente, ki poznamo, popravimo.

[8]

$$W_{k+1} = W_{k+1} + \mathcal{P}_\Omega(M - W_{k+1}) \quad (3.15)$$

malo spremenjeno iz clanca

Korak preprosto spremeni mesta, kjer vrednosti poznamo na znane vrednosti, ostale pa pusti, kakršne so.

### 3.5 Izmenjajoč gradientni spust

Ker je računanje SVD razcepa zahtevna operacija, saj ima časovno zahtevost  $O(n^3)$ , je bilo predlaganih nekaj algoritmov, ki za svoje delovanje ne potrebujejo SVD-ja. Algoritem Izmenjajočega gradientnega spusta, oziroma v nadaljevnu ASD sloni na računanju gradienata in premikanja po njem. Ideja algoritma je najti dve matriki  $X \in \mathbb{R}^{n_1 \times r}$  ter  $Y \in \mathbb{R}^{r \times n_2}$ , tako da velja  $\mathcal{P}_\Omega(M) = \mathcal{P}_\Omega(XY)$ . Vidimo lahko, da ponovno potrebujemo informacijo o rangu matrike, ki jo rekonstruiramo. Ker imata tako  $X$  in  $Y$  kvečjemu rang  $r$ , vemo, da tudi njun produkt  $XY$  ne bo imel ranga večjega od  $r$ .

Cilj algoritma je minimizirati

$$\min_{X,Y} \quad \frac{1}{2} \|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(XY)\|_F^2$$

Algoritem minimizacijo razdeli na dve, nato pa izmenično rešuje eno in nato drugo kot [12]

$$\begin{aligned} X_{i+1} &= \arg \min_X \|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(XY_i)\|_F^2 \\ Y_{i+1} &= \arg \min_Y \|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(X_{i+1}Y)\|_F^2 \end{aligned}$$

Za uporabo algoritma ASD potrebujemo odvoda funkcije  $f(X, Y) =$

$\frac{1}{2} \|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega(XY)\|_F^2$ , ki ga izračunamo za vsak element posebej.

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

kjer

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

Na podoben način bi lahko pokazali tudi

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

Poščimo še najboljši korak gradientnega spusta. Cilj je pokazati, da je premik po gradientu s korakom  $t_x$  najboljši. Najprej definirajmo sam premik, kot

$$X^{k+1} = X^k - t_X \nabla f_Y(X)$$

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

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

Kot v [12] se izkaže

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

vidimo, 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 napolnitvev z uporabo dveh manjših matrik  $X^{n_1 \times r}$  in  $Y^{n_1 \times r}$ . Podobno kot v prejšnjih algoritmih rešujemo problem

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

$$\text{pri pogojih} \quad \mathcal{P}_\Omega(M) = \mathcal{P}_\Omega(Z)$$

le da sam problem rešujemo s pomočjo Moore-Penrose inverza (Označen z  $\dagger$ ).

Lahko je videti, da bo funkcija  $f(X, Y, Z) = \frac{1}{2}\|XY - Z\|_F^2$  imela najmanjšo vrednost, ko bo  $XY = Z$ . Zato uvedemo iterativni algoritem, ki izmenično posodablja  $X$ ,  $Y$  in  $Z$  tako, da fiksira dve izmed matrik. Minimizacijo matrik  $X$  in  $Y$  torej dosežemo na naslednji način.

$$\begin{aligned} X^{i+1}Y^i &= Z^i \\ X^{i+1} &= Z^i(Y^i)^\dagger \end{aligned}$$

Kjer se zanašamo na dejstvo, da nam množenje z Moore-Penrose inverzom z desne da najboljši možen rezultat. Podobno posodobimo matriko  $Y$ .

$$\begin{aligned} X^{i+1}Y^{i+1} &= Z^i \\ Y^{i+1} &= (X^{i+1})^\dagger Z^i \end{aligned}$$

Sedaj 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. [13]

# 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. Gre za problem, ki ga lahko lepo vizualiziramo. Pri surovih podatkih pomena numeričnih vrednosti ni tako enostavno interpretirati, tako da je težje ovrednotiti 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 sliku, 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.

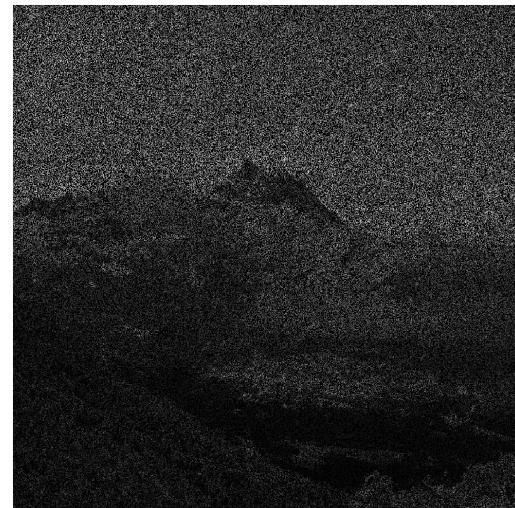
Vse slike si je mogoče v boljši kakovosti ogledati na povezavi <https://tinyurl.com/yb7cjdv7>.

## 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.

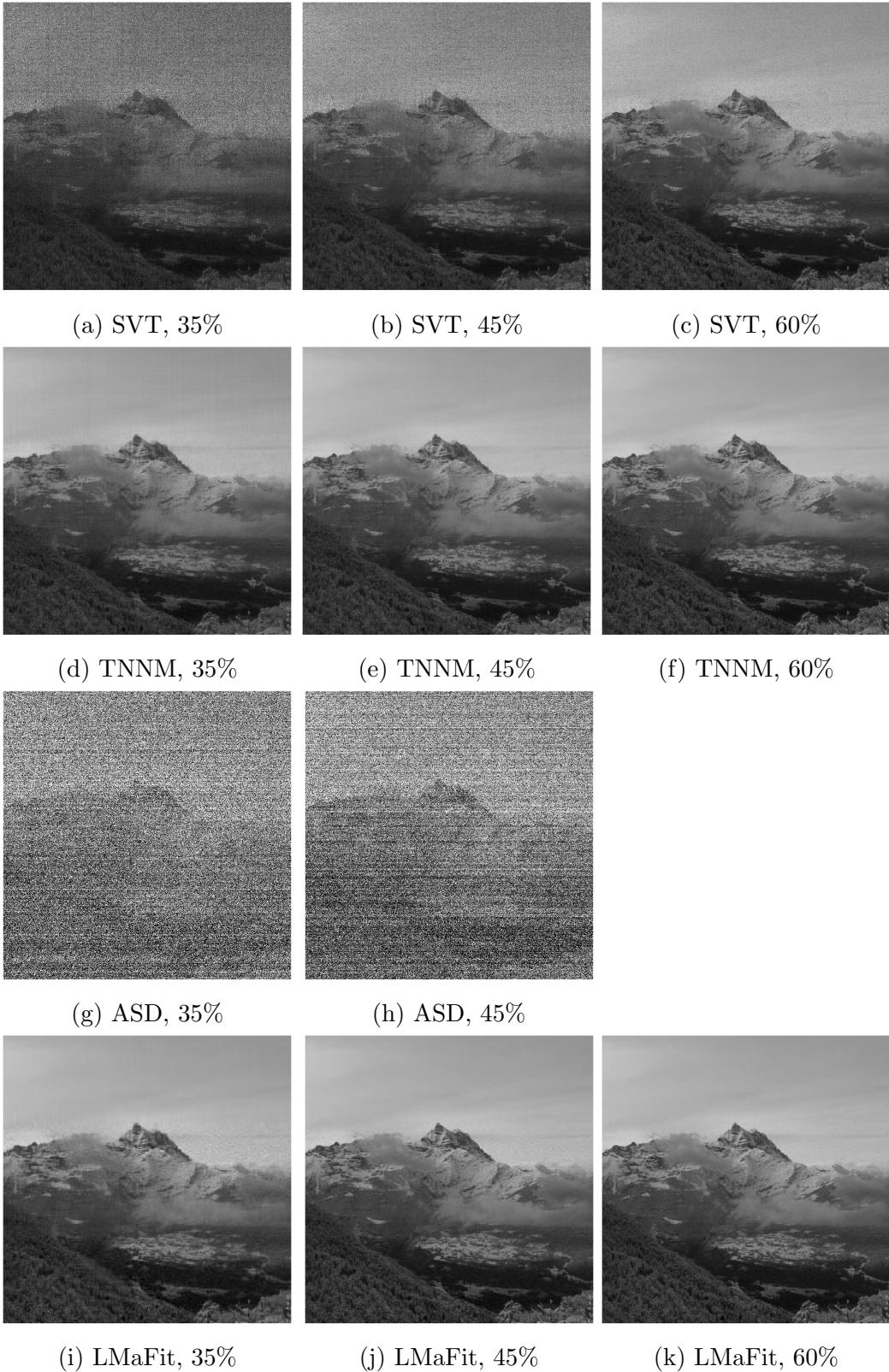


(c) Slika s 45% znanimi podatki.



(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 algoritom konča v lokalnem minimumu. Prav tako algoritom 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

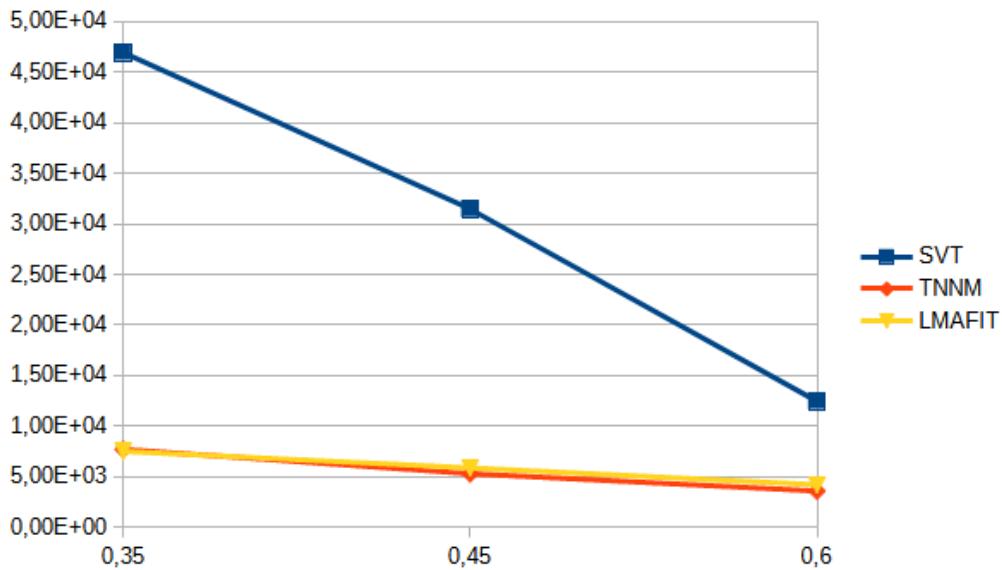
Algoritom NNM smo med rezultati izpustili, saj je zaradi velikega števila matrik, potrebnih za definicijo omejitev, algoritom preveč prostorsko kompleksen. Ta algoritom je zato smiselnouporabljati, kadar imamo matrike manjših dimenzij [3] (algoritom lahko še rešuje probleme, za matrike velikosti  $100 \times 100$ ). 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 algoritom vrniti najboljše rezultate. Zaradi teh opazk se v naslednjih razdelkih osredotočamo na algoritme SVT, TNNM in LMaFit.

OZP \ Algoritom	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.

Medtem ko bomo čase izvajanja algoritmov podrobnejše analizirali v naslednjem razdelku, lahko že sedaj opišemo par ugotovitev. Vidimo lahko, da je algoritom 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% kot pri preskoku iz 35% na 45%.

Ali popravim deleže tudi na grafih?



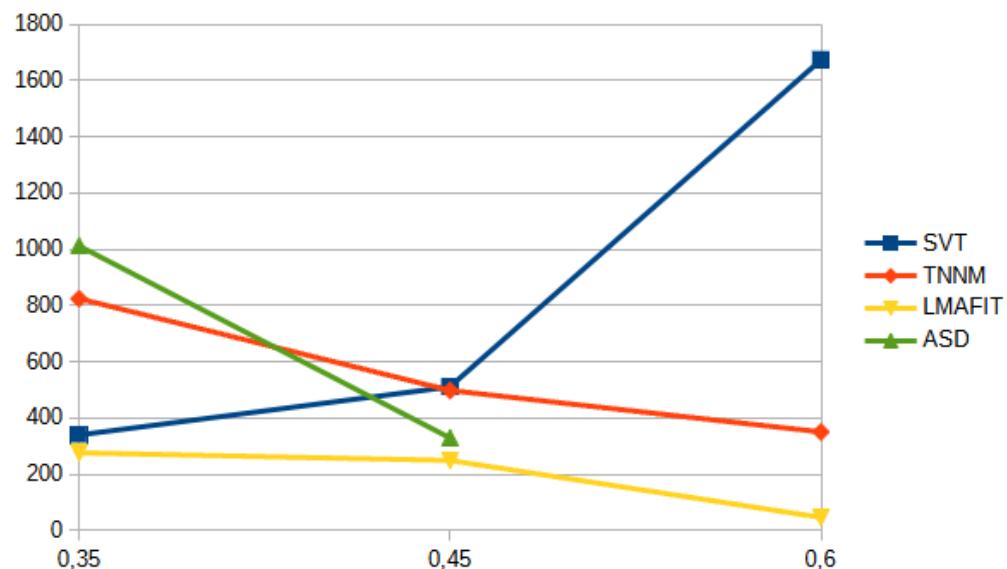
Slika 4.3: Napake algoritmov v Frobeniusovi normi. Na abscisni osi so deleži znanih podatkov slik.

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.

## 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, domnevamo, 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



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

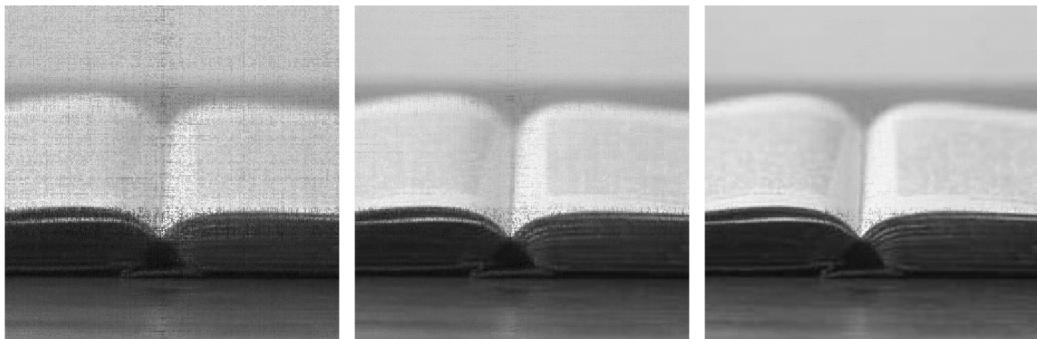
velikosti  $300 \times 300$  pikslov.



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

Kot smo pričakovali, so rezultati rekonstrukcije slike s preprostim moti-

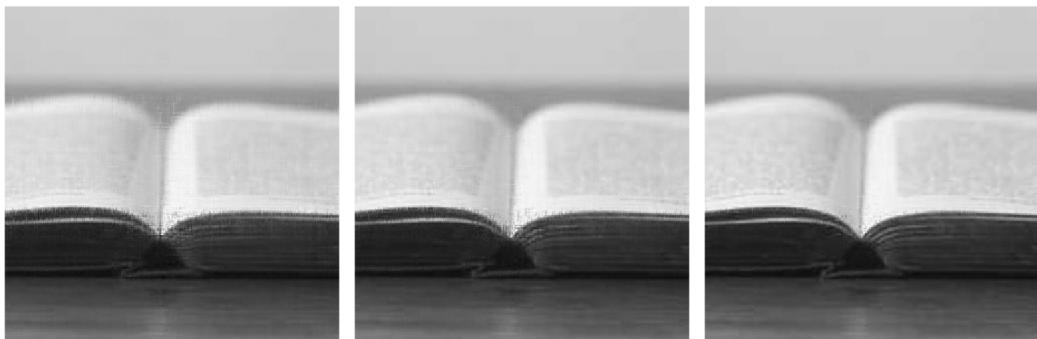
Na koncu  
preveriti,  
da so slike  
na smisel-  
nem mestu



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



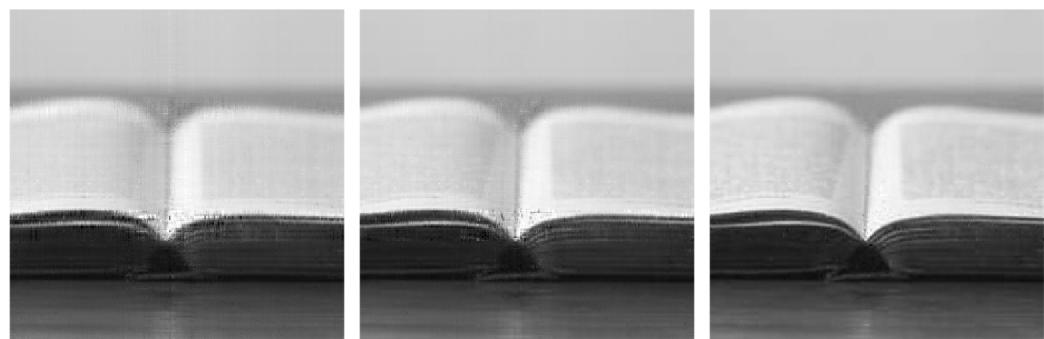
Slika 4.7: Rekonstrukcija kompleksnega motiva z algoritmom SVT. Znane vrednosti slik so bile: 35% (leva), 45% (sredinska), 60% (desna).



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



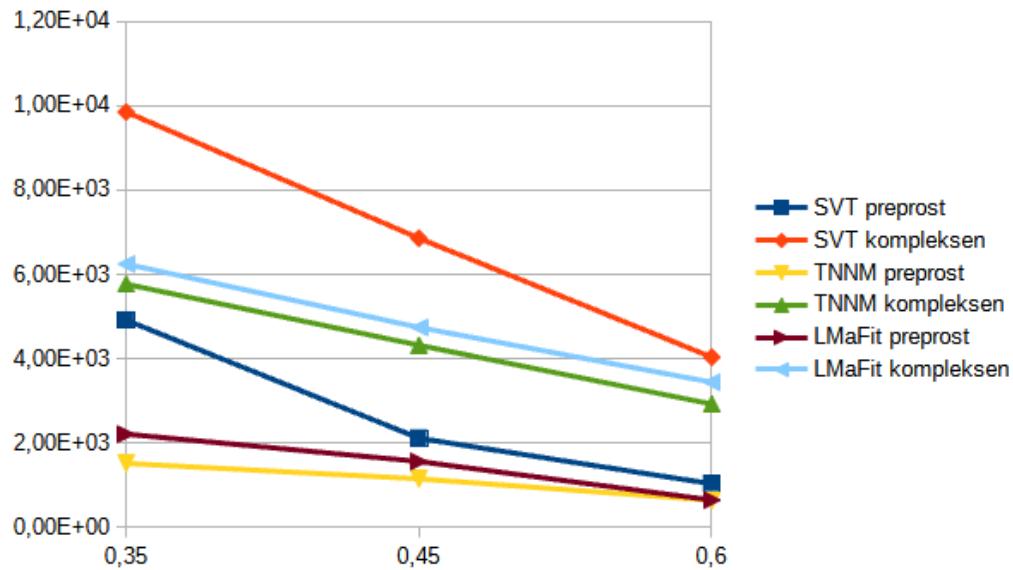
Slika 4.9: Rekonstrukcija kompleksnega motiva z algoritmom TNNM. Znane vrednosti slik so bile: 35% (leva), 45% (sredinska), 60% (desna).



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



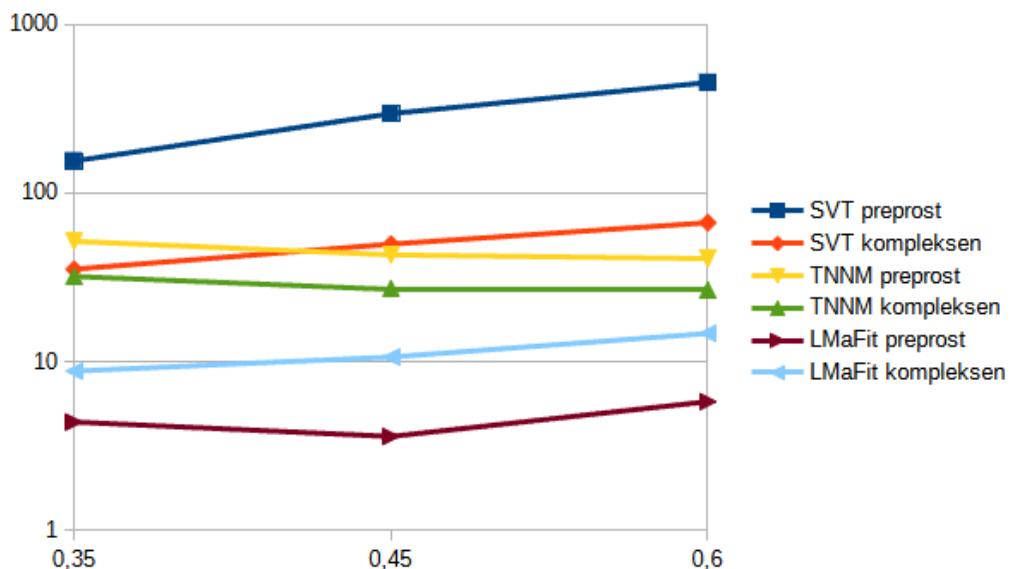
Slika 4.11: Rekonstrukcija preprostega motiva z algoritmom LMaFit. Znane vrednosti slik so bile: 35% (leva), 45% (sredinska), 60% (desna).



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

vom 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 slikai 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.



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 algoritmom 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 algoritmom 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 algoritmom SVT bolj smiselno uporabljati za kompleksne motive. Seveda pa bi lahko, zaradi manjše napake pri preprostem motivu prag nastavili nižje. S tem bi algoritmom 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. Algoritmom 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 algoritom LMaFit začne iteracije z naključnima matrikama  $X$  in  $Y$ , je sam čas izvajanja lahko zelo različen med različnimi zagoni. Smiselno bi bilo razmisiliti 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 sestavljeno 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.

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

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.

smiselno med sabo neodvisne podatke ločiti, ter jih obravnavati samostojno. Rezultat je smiseln, saj nam iskanje podobnosti med nepodobnimi podatki poveča količino dela.

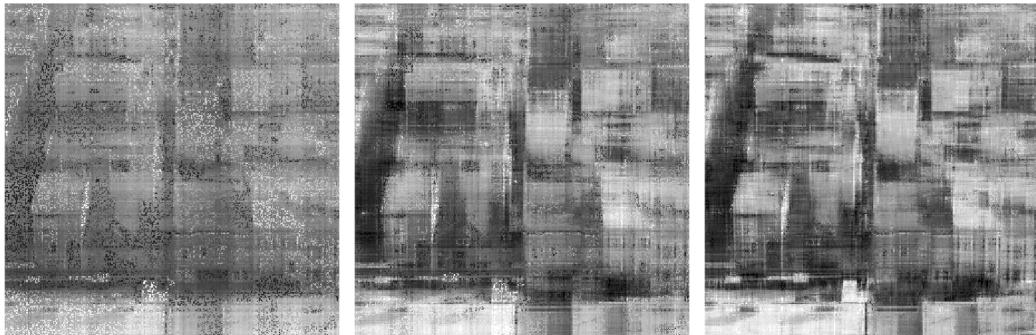
## 4.4 Vpliv podatka o rangu na rezultate

Spomnimo se, da algoritma TNNM in LMaFit za svoje izvajanje potrebujejo informacijo o rangu (v nadaljevanju jo bomo imenovali *parameter*). V tem podoglavlju 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 sliki mesta, z znanim deležem podatkov nastavljenim na 0.35.

### 4.4.1 LMaFit

Algoritem 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 zagoru 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.



(a) Rekonstrukcija s para-  
metrom 1.

(b) Rekonstrukcija s para-  
metrom 5.

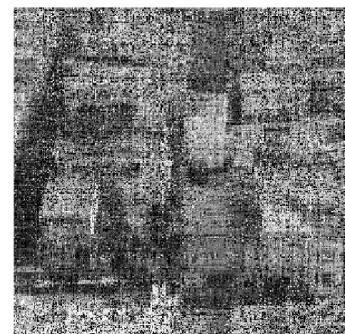
(c) Rekonstrukcija s para-  
metrom 10.



(d) Rekonstrukcija s para-  
metrom 22.



(e) Rekonstrukcija s para-  
metrom 25.



(f) Rekonstrukcija s para-  
metrom 60.

V prvem odstavku govorиш o rangu in različnih parametrih algoritma. Na slikah rang imenuješ parameter. Predlagam, da v imenih slik parameter spremeniš v rang.

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

Parameter	Napaka (v $\ \cdot\ _F$ )	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.

#### 4.4.2 TNNM

V tem razdelku predstavimo rezultate vpliva parametrov na delovanje algoritma TNNM. Zaradi večje časovne zahtevnosti, algoritom TNNM testiramo zgolj trikrat, na rangih parametrov 1, 5 in 12. Za večje parameterje je algoritom 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 3.14). Zaradi tega je težje določiti pravilo, kateri parameter je najboljši.



(a) Rekonstrukcija s parame- (b) Rekonstrukcija s para- (c) Rekonstrukcija s para-  
metrom 1. metrom 5. metrom 12.

Parameter	Napaka ( $v \ \cdot\ _F$ )	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 rangu 1](#) TNNM dosegel najboljše rezultate izmed vseh algoritmov, obravnavanih v tej diplomski 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 sledila, zaradi česar je tudi sama napaka pri tem algoritmu večja. Algoritom 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

Preveri

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.



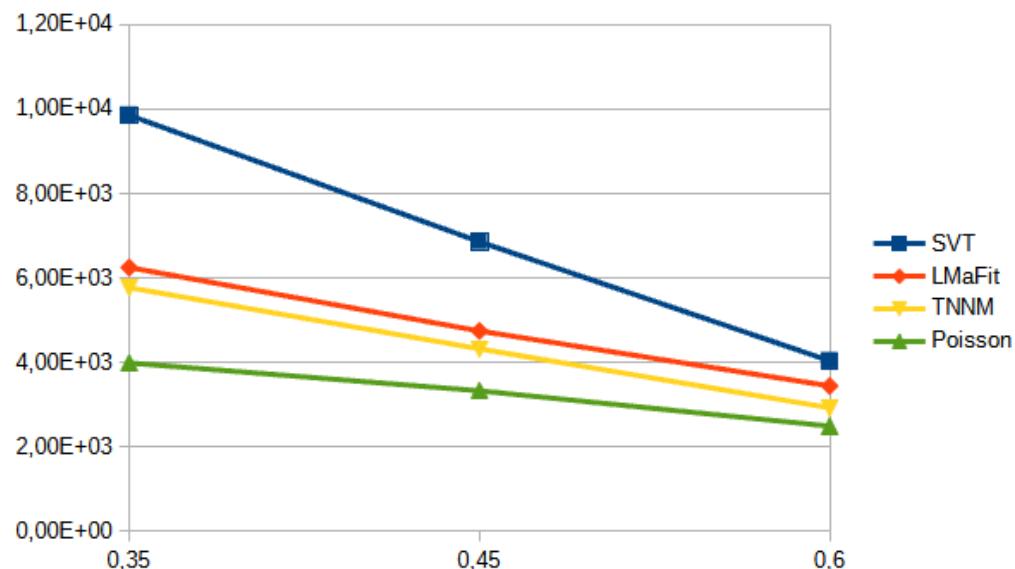
(a) Rekonstrukcija na 0.35  
znanih podatkih.



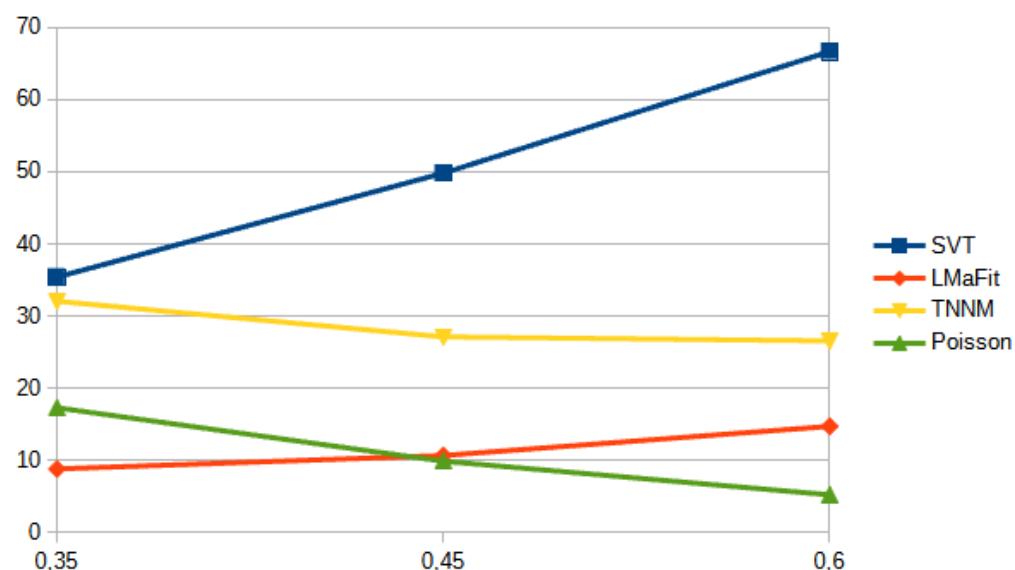
(b) Rekonstrukcija na 0.45  
znanih podatkih.



(c) Rekonstrukcija na 0.60  
znanih podatkih.



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

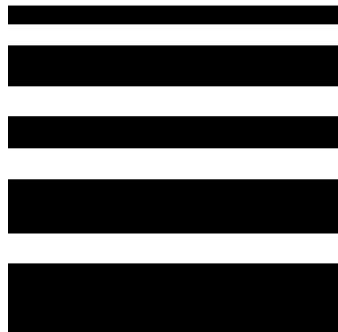


Slika 4.21: Čas izvajanja rekonstrukcije slike mesta. Na abscisni osi so deleži znanih podatkov slik.

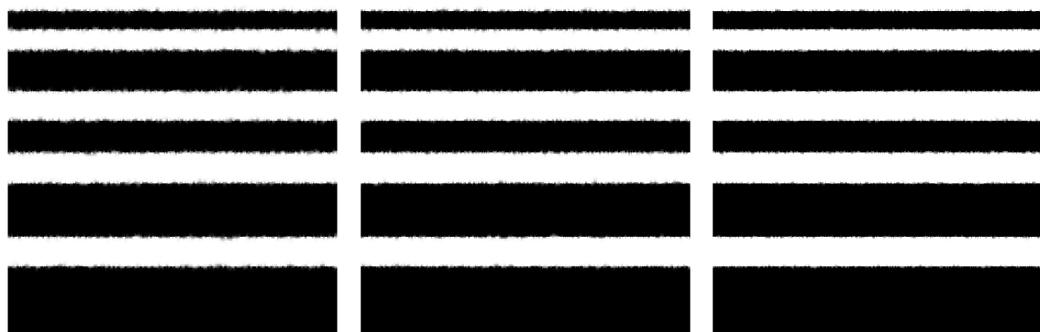
Vidimo, da je algoritem tako hitrejši, kot bolj točen. Vendar je pri primer-

javi potrebno upoštevati, da se tak algoritom 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 algoritmom 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.

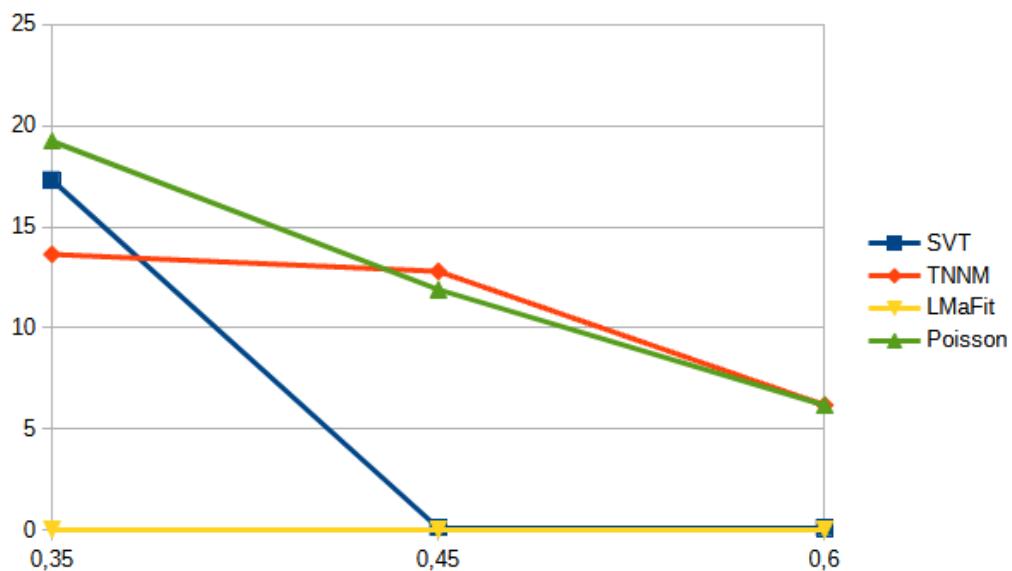


Slika 4.22: Dvobarvna slika.



(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.



# 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, npr. priporočilnih sistemih.

PHD diser-  
tacija

# 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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/10.1007/s12532-012-0044-1).