

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.*



Svoji dragi Alenčici.



# 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	Pomembe definicije . . . . .	5
3.2	Minimizacija nuklearne norme . . . . .	6
3.3	Prag singularnih vrednosti . . . . .	7
3.4	Minimizacija prirezane nuklearne norme . . . . .	10
3.5	Izmenjajoč gradientni spust . . . . .	14
3.6	LMaFit . . . . .	16
<b>4</b>	<b>Rezultati</b>	<b>19</b>
4.1	Velika črno-bela slika . . . . .	19
4.2	Vpliv kompleksnosti slik na napolnjevanje . . . . .	23
4.3	Rekonstrukcija barvnih slik . . . . .	28
4.4	Vpliv podatka o rangu na rezultate . . . . .	29
4.5	Rekonstrukcija slike z besedilom . . . . .	31

4.6 Primerjava rezultatov z algoritmom za reševanje Poissonove enačbe . . . . .	33
<b>5 Zaključek</b>	<b>37</b>

# Seznam uporabljenih kratic

kratica	angleško	slovensko
<b>CA</b>	classification accuracy	klasifikacijska točnost
<b>DBMS</b>	database management system	sistem za upravljanje podatkovnih baz
<b>SVM</b>	support vector machine	metoda podpornih vektorjev



# 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 poskusiti, ter reševati lažje probleme, ki vrnejo dovolj dobre, a ne optimalne rešitve.

Problem je v zadnjih letih zelo popularen, z njim pa se ukvarjajo tako številni matematiki kot računalničarji. Njegova splošnost naredi reševanje problema na številnih področjih, sam pa se v diplomske nalogi osredotočim na razreševanje neznanih pikslov v slikah. Prav tako omenjam in preizkusim algoritem na priporočilnih sistemih. Rezultate teh predstavim v poglavju X.

V tej diplomske nalogi bom predstavil par algoritmov, ki rešujejo omenjen problem. Algoritmi so bili izbrani glede na njihovo popularnost in priznanost v literaturi. Prav tako sem poskrbel, da so algoritmi primerno različni in temeljijo na drugačnih principih. Algoritme sem tudi implementiral, nato pa še napravil analizo ter opisal ugotovitve v poglavju X.

## 1.2 Cilji

Nekaj o tem, kaj želimo narediti.

Glavni prispevki tega diplomskega dela so ...

## 1.3 Struktura diplomskega dela

V poglavju 2 „„, v poglavju 3 ....

# Poglavlje 2

## Pregled področja

Področje je trenutno zelo aktivno, z številnimi raziskovalci, ki se specializirajo na dano področje. Eden vodilnih raziskovalcev je Emmanuel J. Candès, na čigar dela se tekom diplomske naloge večkrat sklicujem.

Vodilna literatura tekom pisanja diplomske naloge je bil članek [**survey**], 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. V kasnejših poglavjih se tam, kjer sem članke potreboval, nanje sklicujem.



# Poglavlje 3

## Algoritmi

Na koncu napisati kratek odstavek o tem, kaj je vsebina poglavja.

### 3.1 Pomembe definicije

Nekatere definicije so uporabljeni čez več algoritmov. Z namenom preglednosti, te opisujem v tem poglavju

1.  $\Omega$  je definirana množica znanih vrednosti

2.

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

3. Operator  $\mathcal{D}_\tau$  kot

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

[1]

4. Z oznako  $M \in \mathbb{R}^{n_1 \times n_2}$  označujemo vhodno matriko, torej tisto, ki ima nekatere podatke neznane.

## 3.2 Minimizacija nuklearne norme

Ker je minimizacija ranga matrike NP-poln problem, so se razvile druge metode, ki samo kompleksnost problema zmanjšujejo. Minimizacija nuklearne norme (Nuclear Norm Minimization) se zanaša na dejstvo, da je rang matrike povezan z nuklearno normo matrike. Ta je definirana kot

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

se lahko sklicujem?

Dokazano je bilo, da nuklearna norma predstavlja konveksno ovojnico ranga.

[2] Konveksna ovojnica funkcije  $f : \mathcal{C} \rightarrow \mathbb{R}$  je največja konveksna funkcija  $g$  tako da velja  $f(x) \geq g(x)$  za vse  $x \in \mathcal{C}$ . [4]

Minimizacijo nuklearne norme je možno pretvoriti v semidefinitni problem, ki ga lahko rešujemo z različnimi pripomočki, na primer SeDuMi [5].

Standardna oblika semidefinitnega programa je

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

Kjer je  $C$  dana matrika,  $\{A_k\}$  in  $\{b_k\}$  pa množici matrik in vektorjev.

Problem minimizacije nuklearne norme lahko zapišemo kot

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

Trdimo, 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 [2]

$$Y = \begin{bmatrix} W_1 & X \\ X^T & W_2 \end{bmatrix},$$

$$Y \succeq 0, \quad \text{Tr}(Y) \leq 2t$$

se lahko sklicujem?

Minimizacijski problem lahko tako redefiniramo kot

$$\begin{aligned} \min_{Y,t} \quad & 2t \\ \text{tako da} \quad & Y \succeq 0, \\ & \langle Y, A_{a,b} \rangle = b_{a,b} \end{aligned}$$

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

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

je lahko videti, da je tak pogoj smiselen. Programi za reševanje SDP pa lahko tako obliko že sprejmejo. [4]

### 3.3 Prag singularnih vrednosti

Algoritem praga singularnih vrednosti, oziroma v nadaljevanju SVT (Singular Value Thresholding) sloni na dejstvu, da imamo pri matrikah z majhnim rangom nekaj velikih singularnih vrednosti, ostale pa blizu ničli. Za svoje delovanje uvede dva nova pomembna koncepta, prvi je premik, drugi pa prag, potreben za uporabo operatorja  $\mathcal{D}_\tau$ . Algoritem lahko na kratko povzamemo z zapisom

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

kjer  $\tau > 0$  predstavlja prag,  $\delta_k$  predstavlja  $k$ -ti premik,  $X \in \mathbb{R}^{n_1 \times n_2}$  ter  $Y^0 = 0 \in \mathbb{R}^{n_1 \times n_2}$ . [1]

Smiselnost algoritma lahko pokažemo s pomočjo Lagrangeovega multiplikatorja. Ponovno rešujemo minimizacijski problem, le da dodamo dodatne parametre in s tem minimizacijo omilimo.

Uvedimo funkcijo  $f_\tau(X) = \tau\|X\|_* + \frac{1}{2}\|X\|_F^2$ . Problem lahko tako zapišemo kot

$$\begin{aligned} \min & \quad f_\tau(X) \\ \text{tako da} & \quad \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M) \end{aligned} \tag{3.1}$$

Lahko je videti, da za velike vrednosti  $\tau$  velja  $f_\tau(X) \approx \tau\|X\|_*$ , kar pokaže, da bo s primerno velikim  $\tau$ , algoritmom res skoraj minimiziral nuklearno normo.

Lagrangeov multiplikator je definiran kot  $\mathcal{L}(x, \lambda) = f(x) + \lambda \cdot g(x)$ , kjer je  $f(x)$  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, pa lahko problem pretvrimo v iterativni algoritmom. [1]

$$\begin{cases} X^k = \arg \min_X \mathcal{L}(X^k, Y^{k-1}) \\ Y^k = Y^{k-1} + \delta_k \mathcal{P}_\Omega(M - X^k) \end{cases}$$

Za reševanje minimizacijskega problema dokažimo teorem

$$\mathcal{D}_\tau(Y) = \arg \min_X \left\{ \frac{1}{2} \|X - Y\|_F^2 + \tau\|X\|_* \right\} \tag{3.2}$$

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, vse točke na vseh tangentah na funkcijo  $h(X)$  bodo pod ali na funkciji  $h(X)$ . 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. [1]

Cilj dokaza je pokazati, 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 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

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

primerna izbira za  $W = \tau^{-1} U_1 \Sigma_1 V_1^T$ , saj

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

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  $W V_0 = 0$ . Ker pa ima matrika  $\Sigma_1$  vse elemente manjše od  $\tau$  velja tudi  $||W||_2 \leq 1$ . S tem smo pokazali, da  $Y - X' \in \tau \partial ||X'||_*$ .

Prav tako lahko sedaj zapišemo algoritom kot [1]

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

### 3.3.1 Nastavljanje parametrov

Medtem, ko so koraki v samem algoritmu definirani kot množica korakov, sem sam za premik uporabljal konstanto, ter korak nastavil na

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

po priporočilih [1].

Prav tako članek 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, jih ne smemo uporabljati kot definitivno najboljše vrednosti. Med mojimi testiranji sem ugotovil, da je vrednost premika velikokrat treba zmanjšati, posebno za manjše matrike z več neznanimi vrednostmi, saj drugače program ni konvergiral. Prav tako, se je večkrat zgodilo, da je bil pridobljen rezultat še vedno zelo zašumljen. Takrat je bilo smiselno prag  $\tau$  povečati. To je sicer upočasnilo program, vendar izboljšalo rezultat.

## 3.4 Minimizacija prirezane nuklearne norme

Že samo ime nam pove, da bo algoritem minimizacije prirezane nuklearne norme, oziroma TNNM (Truncated Nuclear Norm Minimization) podoben algoritmu NNM. Vendar tu privzamemo, da imamo o samem algoritmu še neko dodatno informacijo  $r \in \mathbb{N}$ , ki je povezana z rangom originalne, nezašumljene matrike.

Sam algoritem uvede tako imenovano skrajšano nuklearno normo, ki je za matriko  $X \in \mathbb{R}^{n_1 \times n_2}$  definirana kot

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

torej vsoto  $\min(n_1, n_2) - r$  najmanjših singularnih vrednosti, ter z njeno pomočjo definira problem [3]

$$\begin{aligned} \min_X & \quad \|X\|_r \\ \text{tako da} & \quad \mathcal{P}_\Omega(X) = \mathcal{P}_\Omega(M) \end{aligned}$$

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

Za reševanje minimizacije bomo uporabljali algoritmom ADMM, vendar moramo prej sam problem nekoliko spremeniti. Razpišimo problem kot

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

Za nadaljne korake bomo potrebovali teorem

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

kjer velja  $X \in \mathbb{R}^{n_1 \times n_2}$ ,  $A \in \mathbb{R}^{r \times n_1}$ ,  $B \in \mathbb{R}^{r \times n_2}$  in  $r \in \mathbb{N}$ ,  $r \leq \min(n_1, n_2)$ , kot tudi  $AA^T = I_r$ ,  $BB^T = I_r$ .

Za dokaz uporabljamo Von Neumannovo neenakost sledi, 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 poveemo, 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  $B^T A$   $r$  singularnih vrednosti enakih 1, saj ima  $I_n$   $n$  lastnih vrednosti enakih 1. Tako lahko sedaj razpišemo izraz

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

Ugotovili smo, da velja

$$\text{Tr}(AXB^T) \leq \sum_{i=1}^r \sigma_i(Xs) \quad (3.3)$$

Če definiramo SVD razcep  $X = U\Sigma V^T$ , 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  $V$ , lahko pokažemo da velja  $\text{Tr}(AXB^T) = \sum_{i=1}^r \sigma_i(X)$ .

$$\begin{aligned} \text{Tr}(AXB^T) &= \text{Tr}((u_1, \dots, u_r)^T X (u_1, \dots, u_r)^T) = \\ &= \text{Tr}((u_1, \dots, u_r)^T U \Sigma V^T (u_1, \dots, u_r)^T) = \\ &= \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} \Sigma \begin{bmatrix} I_r & 0 \\ 0 & 0 \end{bmatrix} = \\ &= \text{Tr}\left(\begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \sigma_r & \\ & & & 0 & \ddots & \\ & & & & & 0 \end{bmatrix}\right) = \sum_{i=1}^r \sigma_i(X) \quad (3.4) \end{aligned}$$

Z združitvijo dokazov 3.3 in 3.4 zapišemo

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

Torej je problem, ki ga minimizarmo lahko podan kot

$$\min_X \|X\|_* - \max_{AA^T=I, BB^T=I} \text{Tr}(AXB^T)$$

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

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 [3]

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

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

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

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

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

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

Ta problem ponovno zapišemo s pomočjo Lagrangeove funkcije, le da algoritmom ADMM definira še kazenski parameter  $\beta$ . [3]

kazenski  
parameter?

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

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. 3.5

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

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

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 teoremu 3.2 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.

malo spremenjeno iz clanca

[3]

$$W_{k+1} = W_{k+1} + \mathcal{P}_\Omega(M - W_{k+1}) \quad (3.5)$$

## 3.5 Izmenjajoč gradientni spust

Ker je računanje SVD razcepa zahtevna operacija, saj ima časovno zahtevnost  $O(n^3)$ , je bilo predlaganih nekaj algoritmov, ki za svoje delovanje ne potrebujejo SVD-ja. Algoritem Izmenjajočega gradientnega spusta, oziroma v nadaljevnu ASD (Alternating Steepest Descent) sloni na računanju gradienta in premikanju po njem. Glavni cilj algoritma je, najti dve matriki  $X \in \mathbb{R}^{n_1 \times r}$  ter  $Y \in \mathbb{R}^{r \times n_2}$ , tako da bo veljalo  $M = XY$ . 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

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

[6]

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.

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

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

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

Poiskimo 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

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

kjer

$$\begin{aligned} g(t) &= \frac{1}{2} \|\mathcal{P}_\Omega(M) - \mathcal{P}_\Omega((X - t\nabla f_Y(X))Y)\|_F^2 \\ g(t) &= \frac{1}{2} \|\mathcal{P}_\Omega(M - XY)\|_F^2 - t \text{Tr}(\mathcal{P}_\Omega(M - XY)\mathcal{P}_\Omega(\nabla f_Y(X)Y)^T) + \frac{t^2}{2} \|\mathcal{P}_\Omega(\nabla f_Y(X)Y)\|_F^2 \end{aligned}$$

ali lahko  
tako?

Ker je [6]

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

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

$$\begin{aligned} \min_{X,Y,Z} \quad & \frac{1}{2} \|XY - Z\|_F^2 \\ \text{tako da} \quad & \mathcal{P}_\Omega(M) = \mathcal{P}_\Omega(Z) \end{aligned}$$

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. [7]



# Poglavlje 4

## Rezultati

V tem poglavju bom opisal rezultate, ki sem jih pridobil. Kot sem že omenil, bo večji del preizkušanja programa opravljen na slikah, kjer bodo nekateri piksli manjkali. Gre za problem, ki ga je moč lepo vizualizirati, saj pogosto pri surovih podatkih ni lahko definirati njihovo točnost, zaradi česar težko interpretiramo, kako koristen je sam algoritem.

Prav tako bom opisal točnost rezulatov različnih metod kot tudi čas izvajanja posameznih metod. Probleme bom zagnal tudi na različnih vrst podatkov, npr. podatkih ki so generirani normalno kot tudi enakomerno porazdeljeno. Zaradi interpretacije, slike razdelimo v več skupin, za katere opišemo ugotovitve.

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

### 4.1 Velika črno-bela slika

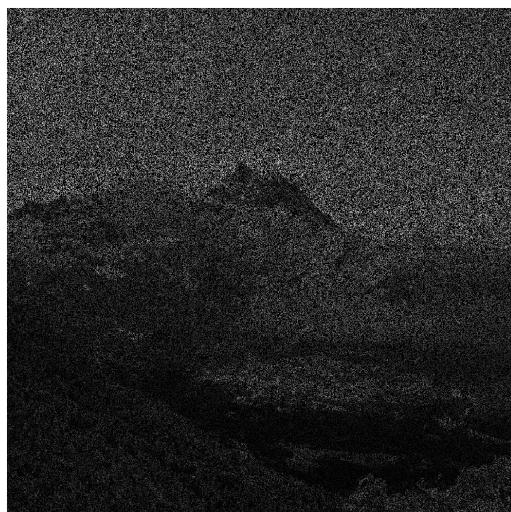
Same teste algoritmov najprej poženemo na veliki, črno-beli sliki, velikosti  $1000 \times 1000$  pikslov. Taka izbira je smiselna, iz vidika, da imamo dovolj podatkov, potrebnih za rekonstrukcijo. Ker je časovna zahtevnost algoritmov pri večjih slikah, kot bomo videli v nadaljevanju že precej velika, nam ta faza testiranja služi kot preverjanje delovanja samih algoritmov. Same podrobno-

sti razlik med rezultati si bomo zato podrobneje pogledali na manjših slikah v nadaljevanju. Algoritme preizkušamo trikrat, na podatkih z deleži znanih vrednosti 0.35, 0.45 in 0.6.

Smiselna  
postavitev  
teskst-slik  
in velikosti



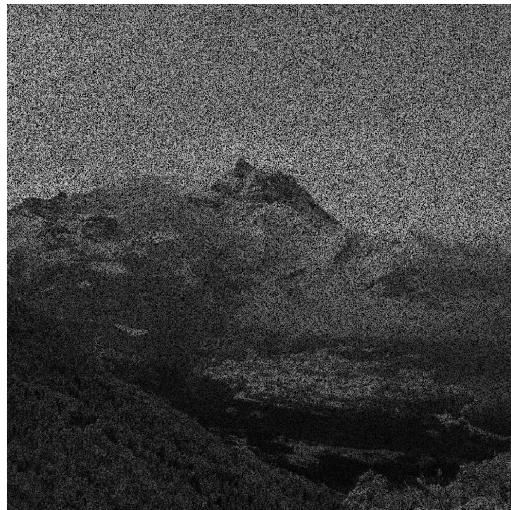
(a) Originalna slika



(b) Slika z 0.35 znanimi podatki



(c) Slika z 0.45 znanimi podatki

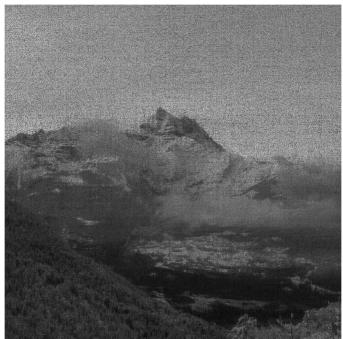


(d) Slika z 0.60 znanimi podatki

Slika 4.1: Vir slike: Unsplash



(a) SVT 0.35



(b) SVT 0.45



(c) SVT 0.6



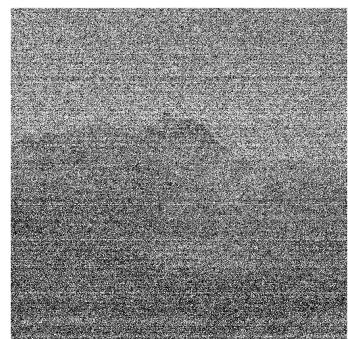
(d) TNNM 0.35



(e) TNNM 0.45



(f) TNNM 0.6



(g) ASD 0.35



(h) ASD 0.45



(i) LMaFit 0.35



(j) LMaFit 0.45

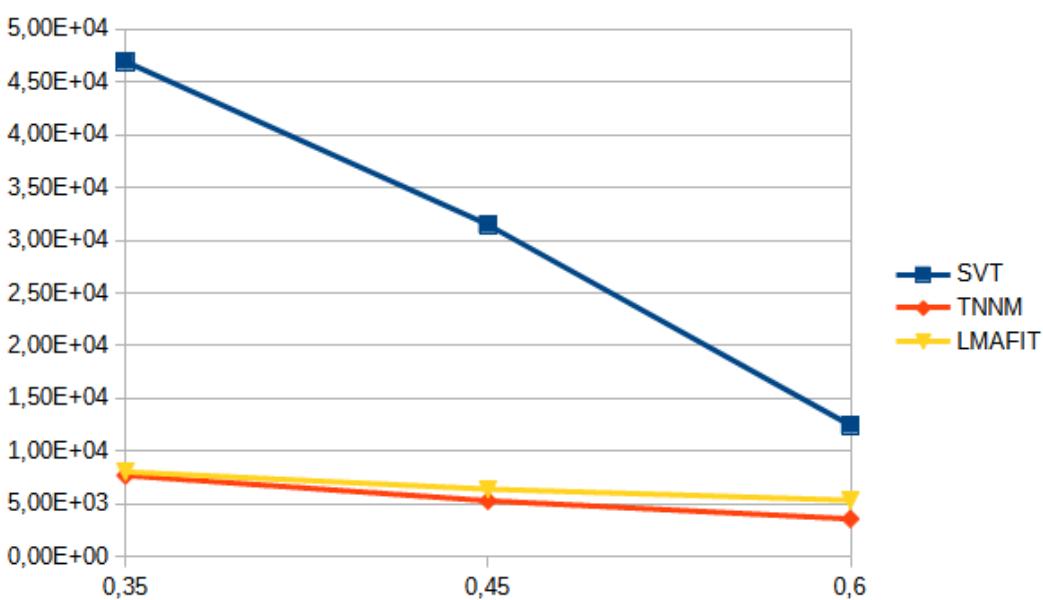


(k) LMaFit 0.60

Kot vidimo, je med rezultati velika razlika. Očitno je, da algoritmi TNNM, SVT in LMaFit delujejo najbolje, medtem ko ima algoritom ADDM vprašljive rezultate. Te si lahko interpretiramo kot posledico lastnosti, da lahko algoritom končata v lokalnem minimumu. Prav tako algoritom ADMM ni našel rešitve, ko je imel poznanih 0.60 podatkov. Zato je ta algoritma smiselnouporabljati, kadar imamo dober začeten približek matrik  $X$  in  $Y$  ter manj poznanih vrednosti. Algoritom NNM smo med rezultati izpustili, saj je zaradi velikega števila matrik, potrebnih za definicijo omejitev, algoritom preveč prostorsko kompleksen. Ta algoritom bomo zato obravnavali posebej. Zaradi teh opazk se v naslednjih podpoglavljih v večini osredotočamo na algoritme SVT, TNNM in LMaFit.

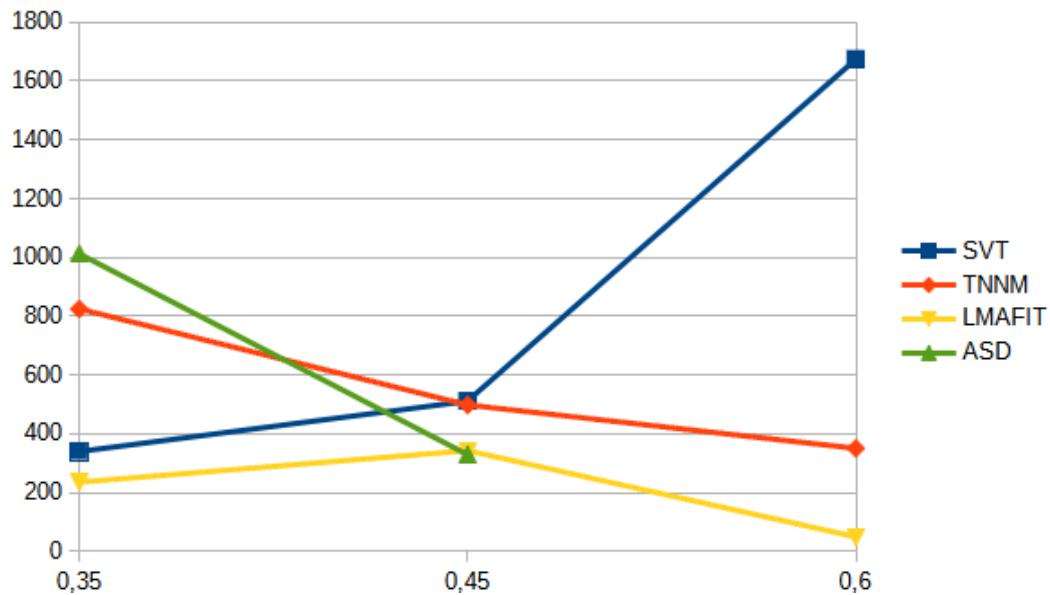
je potrebno in graf in tabelo?

	SVT	TNNM	LMAFIT	ASD
0.35	$4.69 \times 10^4$	$7.70 \times 10^3$	$8.03 \times 10^3$	$3.9743 \times 10^7$
0.45	$3.15 \times 10^4$	$5.30 \times 10^3$	$6.40 \times 10^3$	$6.0910 \times 10^7$
0.6	$1.25 \times 10^4$	$3.58 \times 10^3$	$5.35 \times 10^3$	-



Slika 4.3: Napake algoritmov glede na delež znanih vrednosti

	SVT	TNNM	LMAFIT	ASD
0.35	338s	824s	235s	1012s
0.45	510s	498s	342s	328s
0.6	1674s	350s	48s	-



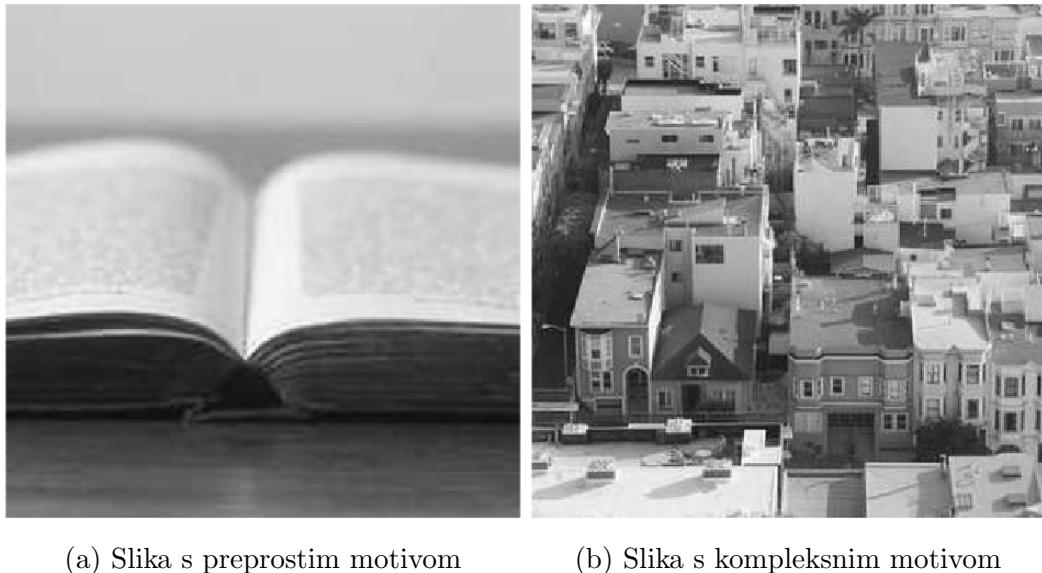
Slika 4.4: Časi izvajanja algoritmov glede na delež znanih vrednosti

## 4.2 Vpliv kompleksnosti slik na napolnjevanje

Eno izmed glavnih vprašanj, ki se nam lahko porodi pri implementaciji algoritmov za napolnitev matrik je, kako sama kompleksnost slik vpliva na točnost rezultatov. Ker slike naključnih vrednosti ni mogoče rekonstruirati, lahko sklepamo, da bodo slike s preprostimi motivi napolnjene bolje. Za namene testiranja je torej smiselno izbrati tako preprosto kot tudi vizualno nasičeno sliko. V naših testiranjih uporabljamo slike knjige in 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 močnejši

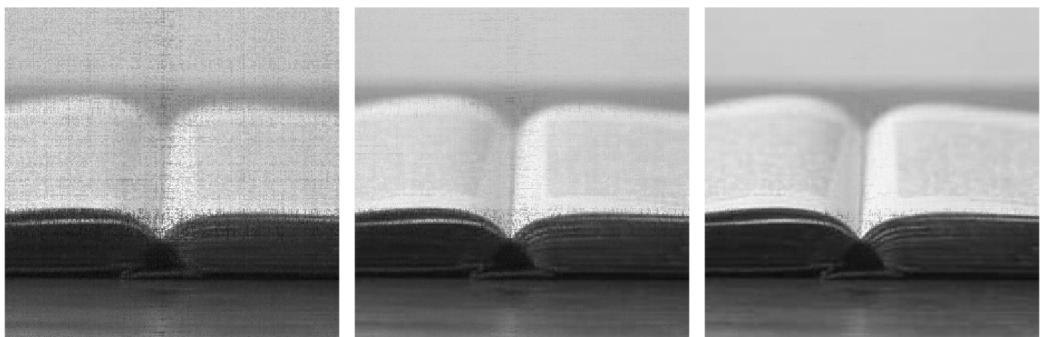
Lmafite  
vcasih  
potrebno  
zagnati  
veckat



(a) Slika s preprostim motivom

(b) Slika s kompleksnim motivom

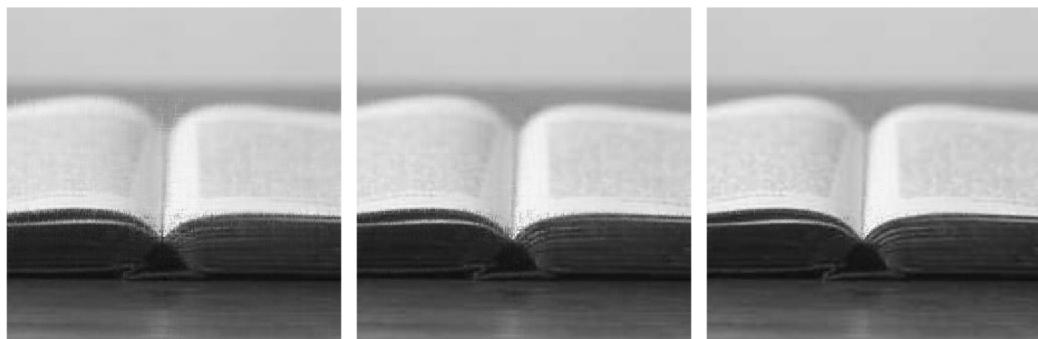
Slika 4.5: Vir slik: Unsplash



Slika 4.6: Rekonstrukcija preprostega motiva z algoritmom SVT



Slika 4.7: Rekonstrukcija kompleksnega motiva z algoritmom SVT



Slika 4.8: Rekonstrukcija preprostega motiva z algoritmom TNNM



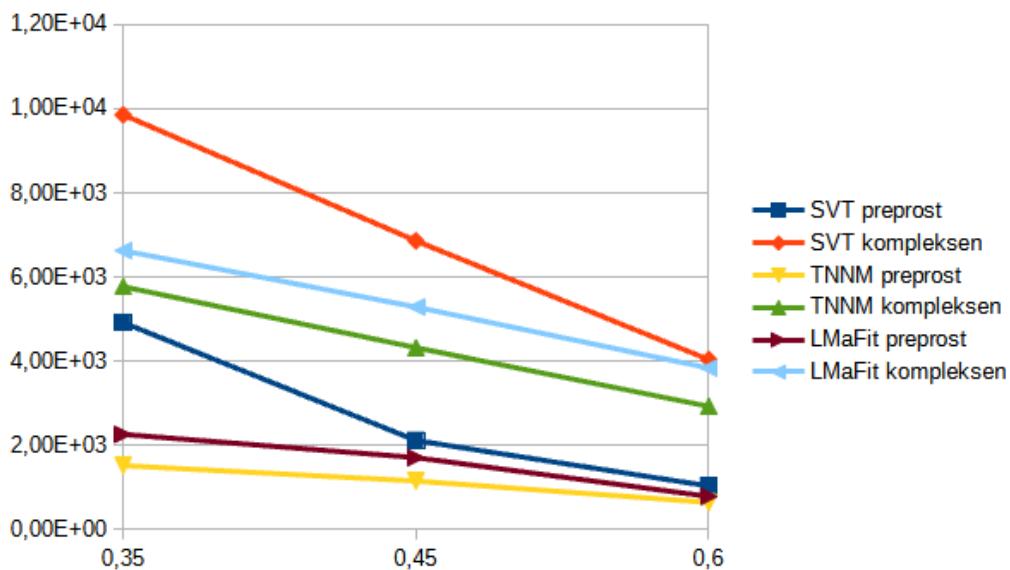
Slika 4.9: Rekonstrukcija kompleksnega motiva z algoritmom TNNM



Slika 4.10: Rekonstrukcija preprostega motiva z algoritmom LMaFit



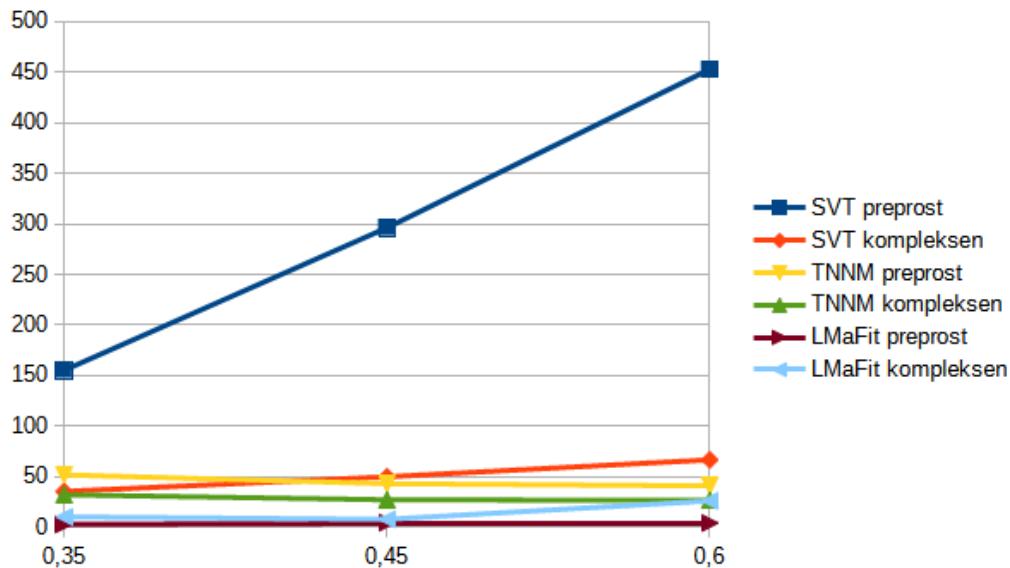
Slika 4.11: Rekonstrukcija preprostega motiva z algoritmom LMaFit



Slika 4.12: Graf napak algoritmov

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 sliki podati večjo vrednost ranga, da sta lahko algoritma prišla do dobrih rezultatov.

Sama 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

Sami časi izvajanja pa tu niso tako intuitivni. Prva glavna opazka je, da algoritem SVT potrebuje veliko več časa pri preprostem motivu kot pri kompleksem. To si lahko razlagamo kot posledico praga. Za dobre rezultate smo pri tako majhni matriki prag nastavili visoko. V našem primeru je imela vrednost 3600. V primeru preprostega motiva, lahko pričakujemo, da bomo imeli malo zelo velikih singularnih vrednosti. Zaradi tega se algoritem težko premika in išče rešitev. Iz tega sledi opazka, da je algoritem SVT bolj smiselnou uporabljati za kompleksne motive.

razmisli to  
interpretacijo

Naslednja pomembna opazka pa je, da delež znanih vrednosti različno vpliva na sam čas izvajanja. Tudi ta faktor je torej lahko pomemben pri izbiri algoritma za reševanje problema. Algoritem SVT potrebuje za rekonstrukcijo več časa, kadar ima znanih več vrednosti, medtem ko se algoritmu TNNM z deležem znanih vrednosti čas izvajanja manjša. Algoritmu LMaFit težko določimo pravilo, saj najpočasnejše izvede rekonstrukcijo pri 0.45 znanih podatkov. Iz tega sklepamo, da je pri nekem deležu med 0.35 in 0.6 rekonstrukcija najpočasnejša. Tu pa je vredno tudi omeniti, da zaradi naključnega generiranja začetne matrike algoritem lahko različno dolgo

Preveri  
znova

rekonstruira isti primer. Ker pa smo teste pognali večkrat ter v povprečju vedno najdlje čakali pri vrednosti 0.45 lahko sklepamo, da je v takih primerih rekonstrukcija res bolj zahtevna.

### 4.3 Rekonstrukcija barvnih slik

Naslednje vprašanje, ki se nam lahko porodi pri testiranju algoritmov je, kako vplivajo barvne slike na samo rekonstrukcijo. Barvne slike so podatkovno podane kot kombinacija barvnih kanalov rdeče, zelene in modre barve. Glavno vprašanje, na katerega bomo poskušali odgovoriti v tem poglavju je, ali je bolje napolnjevati vsak barvni kanal posebej, ali sliko kot celotno. V prvem primeru torej algoritom zaženemo trikrat, medtem ko v drugem sestavimo veliko matriko sestavljenko kot

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

kjer  $R$  predstavlja matriko z vrednostmi rdečega kanala,  $G$  vrednosti zelenega kanala ter  $B$  vrednosti modrega kanala. Vsi testi v tej fazi so bili izvedeni na podatkih, kjer imamo poznanih 0.35 informacij.

kako ome-  
niti da je  
frobenius

	SVT	TNNM	LMAFIT
Enojna rekonstrukcija	$1,50 \times 10^4$	$9,60 \times 10^3$	$1,15 \times 10^4$
Trojna rekonstrukcija	$1,66 \times 10^4$	$9,79 \times 10^3$	$1,14 \times 10^4$

	SVT	TNNM	LMAFIT
Enojna rekonstrukcija	352s	124s	66s
Trojna rekonstrukcija	112s	100s	42s

Lahko je videti, da medtem ko obe metodi vrnete približno enako dobre rezultate, je rekonstrukcija pri vseh algoritmih hitrejša, če ločene barvne kanale rekonstruiramo posebej. Iz tu je lahko videti, da je smiselno med sabo

neodvisne podatke ločiti, ter jih reševati samostojno. Rezultat je smiselen, saj nam iskanje podobnosti med nepodobnimi podatki poveča količino dela.

## 4.4 Vpliv podatka o rangu na rezultate

Kot smo omenili že nekajkrat, algoritma TNNM in LMaFit za svoje izvajanje potrebujejo informacijo o rangu (v nadaljevanju parameter). V tem podpoplavju bomo poskušali odgovoriti na vprašanje, kako pri obeh algoritmih ta informacija vpliva na same rezultate in izvajanje. Za namene testiranja smo algoritom na isti sliki pognali večkrat, ter postopoma večali rang. Ponovno smo teste izvajali na sliki mesta, z znanim deležem podatkov nastavljenim na 0,35.

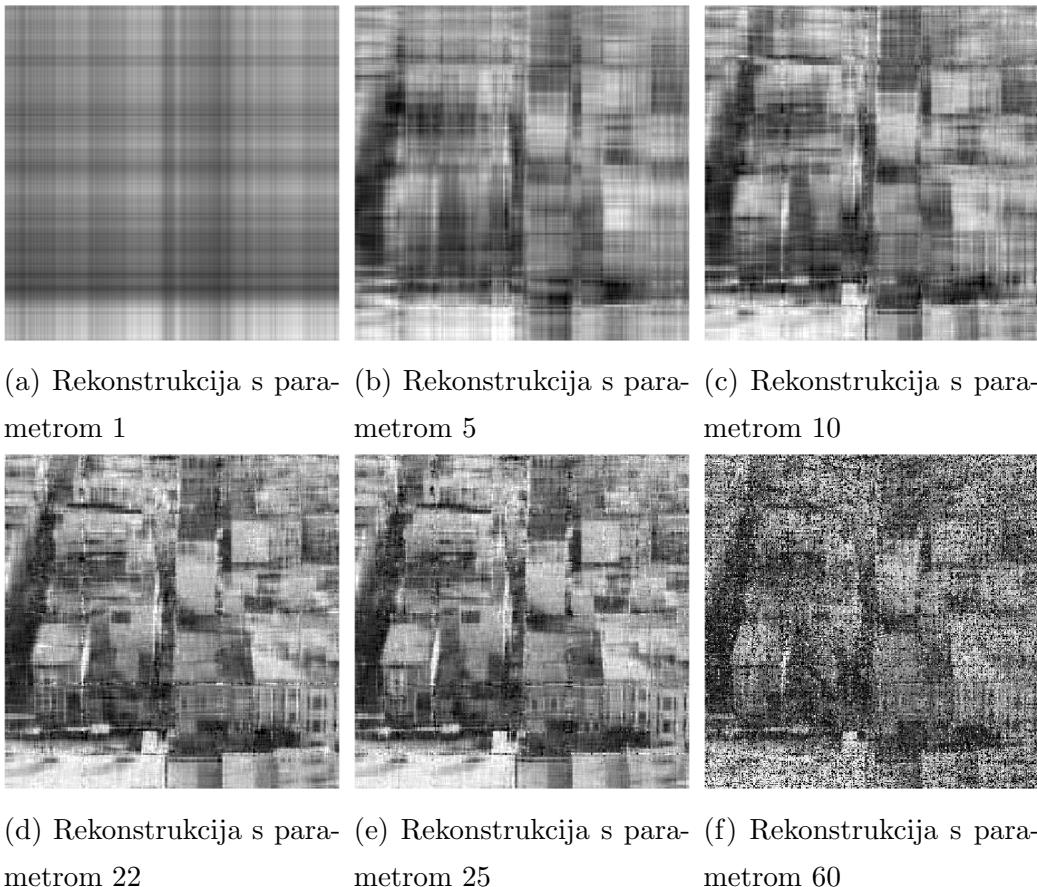
### 4.4.1 LMaFit

V tej fazi smo algoritom testirali štirikrat, z rangom rezultata določenega na 1, 5, 10, 22, 25 in 60. Vrednost 22 je bila izbrana, ker je ob večkratnem zagonu programa na različnih parametrih, ta dala najboljši rezultat. Posledično sta bili vrednosti 25 in 60 izbrani z namenom opazovanja, kakšne rezultate pridobimo s precenitvijo ranga rekonstruirane matrike.

Parameter	Napaka	Čas izvajanja
1	$1.36 \times 10^4$	0.08s
5	$9.60 \times 10^3$	0.65s
10	$7.65 \times 10^3$	0.80s
22	$6.75 \times 10^3$	15.5s
25	$6.87 \times 10^3$	212s
60	$2.39 \times 10^4$	306s

Tabela 4.1: Rezultati rekonstrukcije za različne parametre

Opazimo lahko izboljševanje podrobnosti slike, vse do parametra 22. Vidimo pa lahko tudi, da pride pri prevelikem parametru do preobrata. Re-



zultati se ponovno slabšajo, algoritem pa postane počasnejši. Iz tega lahko razberemo, da je pravilna izbira parametra ključna. Vredno je omeniti, da za vrednosti parametra na nekem intervalu algoritem ne konvergira. V primeru zgornje slike in parametra 30, algoritem LMaFit ni našel rešitve.

#### 4.4.2 TNNM

Zaradi večje časovne zahtevnosti, algoritem TNNM testiramo zgolj trikrat, na deležih informacij 1, 5 in 12. Za večje parametre je algoritem konvergiral zelo počasi, zaradi česar se testiranjem teh izognemo. Tukaj se je pomembno spomniti, da parameter pri algoritmu TNNM ne določa samega ranga rezultata, vendar je zgolj povezan z njim pri reševanju, saj uporablja matriki  $A_l$  in  $B_l$ . Zaradi tega je težje določiti pravilo, kateri parameter je najboljši.



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

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

Sami rezultati med seboj delujejo podobni. Izračun napak nam pokaže, da medtem ko se napaka zmanjšuje s povečevanjem ranga, se hitreje povečuje tudi sam čas reševanja. Torej je treba razmisliti, kako dober rezultat potrebujemo, saj je sama točnost rekonstrukcije časovno draga. Seveda pa velja povedati, da je že pri parametru 1 TNNM dosegel najboljše rezultate izmed algoritmov v tej diplomske nalogi.

## 4.5 Rekonstrukcija slike z besedilom

V tem podpoglavlju bomo preizkušali učinkovitost algoritmov na slikah, kjer se želimo znebiti besedila na sliki. Gre za drugačno vrsto šuma, kjer namesto da bi bili neznani podatki enakomerno razporejeni, so ti zgoščeni na določenem delu. V našem primeru je bil delež znanih podatkov enak 0.92, več kot v primerih, ki smo si jih ogledali do sedaj.

Ponovno lahko opazimo, da je najboljši rezultat vrnil algoritem TNNM. Pri algoritmu SVT lahko opazimo sledi besedila, zaradi česar je tudi sama napaka pri tem algoritmu večja. Algoritem LMaFit ni skonvergiral za para-



Slika 4.16: Slika z besedilom



(a) Rekonstrukcija z algo-  
ritmom SVT

(b) Rekonstrukcija z algo-  
ritmom TNNM

(c) Rekonstrukcija z algo-  
ritmom LMaFit

	Napaka	Čas izvajanja (s)
SVT	$4.64 \times 10^3$	674
TNNM	$2.64 \times 10^3$	83.6
LMaFit	$9.10 \times 10^3$	57.9

meter ranga večjega od 16, zaradi česar je tudi sam rezultat slab. Ta algoritem torej ni smiselno uporabljati pri problemih, kjer so neznane vrednosti zgoščene.

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 Poissonove enačbe

[Preveri](#)

Slike se v praksi pogosto rekonstruira z reševanjem Poissonove enačbe

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

kjer odvode zaradi diskretnosti aproksimiramo kot

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

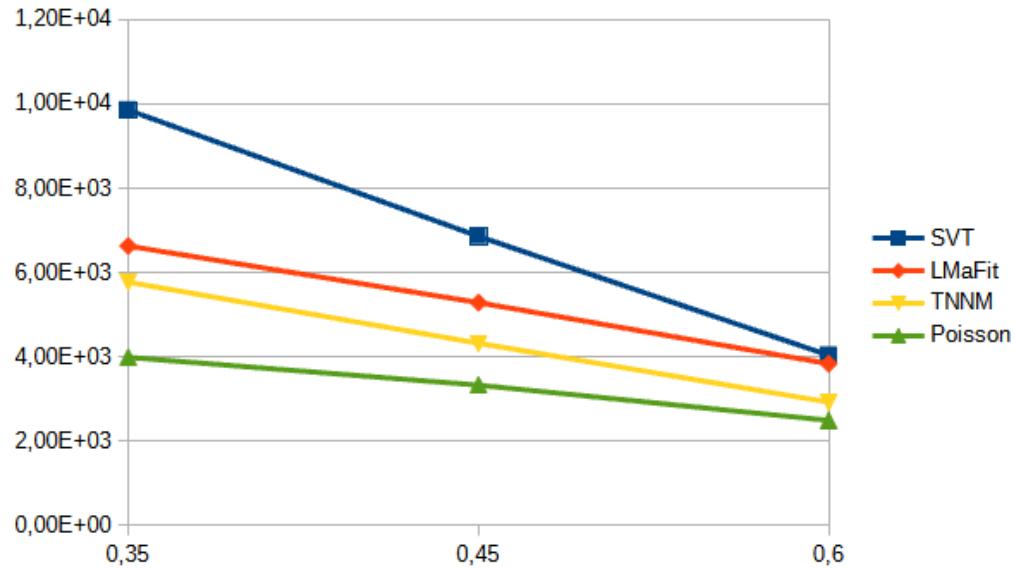
Z uporabo Jacobijeve iteracije, lahko problem rešujemo iterativno, tako da neznane vrednosti v vsaki iteraciji posodobimo.

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

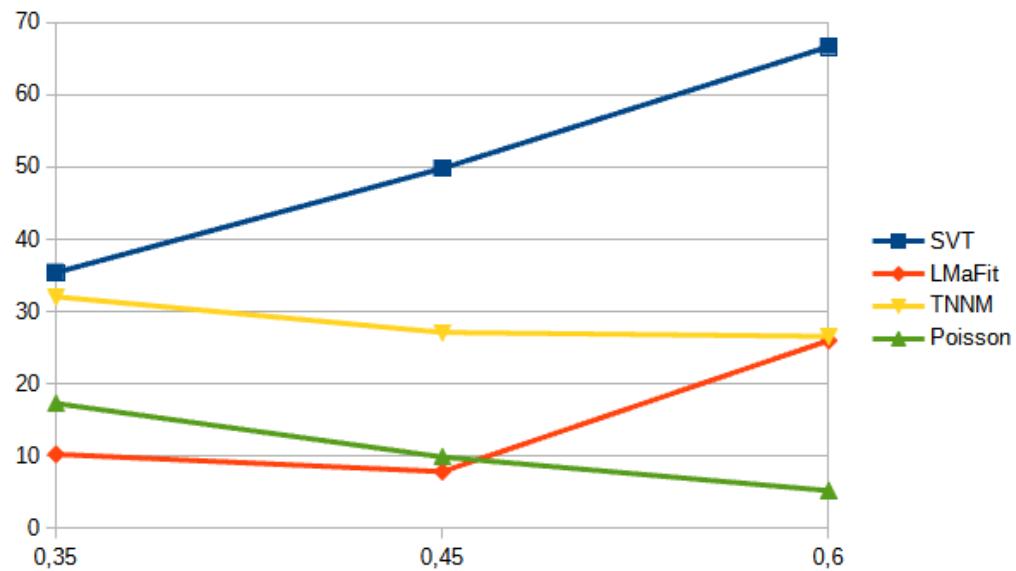
Spodaj si lahko ogledamo rezultate slik, ponovno preizkušene na sliki mesta.



(a) Rekonstrukcija na 0.35 znanih podatkih      (b) Rekonstrukcija na 0.45 znanih podatkih      (c) Rekonstrukcija na 0.60 znanih podatkih

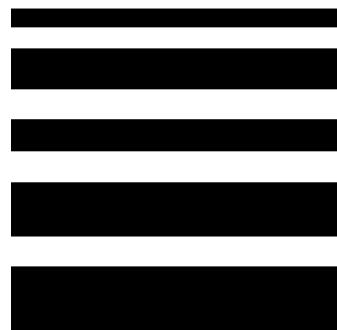


Slika 4.19: Napaka rekonstrukcij slike mesta glede na delež znanih podatkov



Slika 4.20: Čas izvajanja rekonstrukcije slike mesta glede na delež znanih podatkov

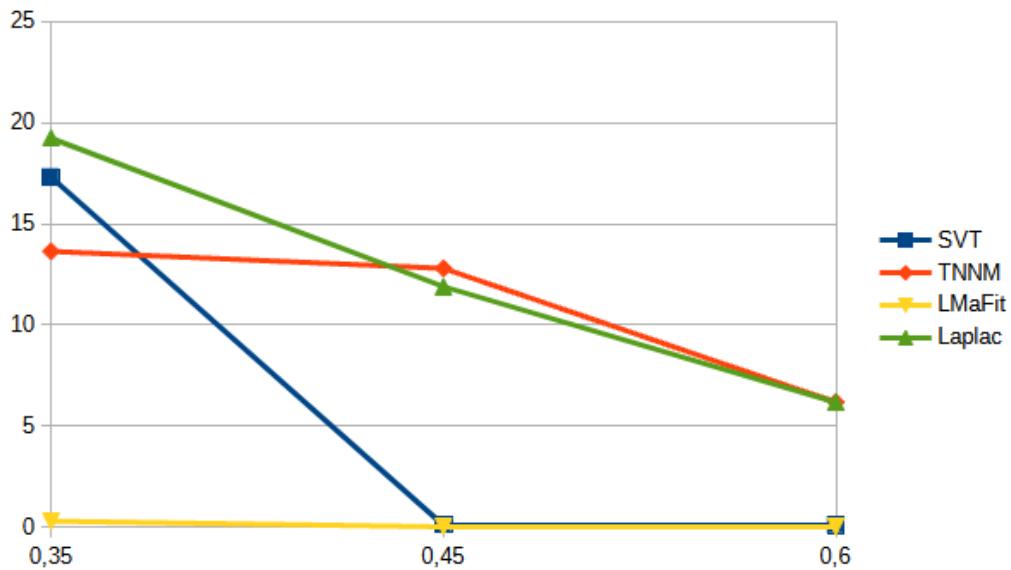
Vidimo, da je algoritem tako hitrejši, kot bolj točen. Vendar, je pri primerjavi potrebno upoštevati, da se tak algoritem zanaša na podobnost lokalnih podatkov. Pri problemu minimizacije ranga, pa se na take podobnosti ne moremo vedno zanašati. Omenili smo že, da lahko algoritem uporabljam v priporočilnih 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 4.21: Dvobarvna slika



(a) Rekonstrukcija na 0.35 znanih podatkih      (b) Rekonstrukcija na 0.45 znanih podatkih      (c) Rekonstrukcija na 0.60 znanih podatkih



Slika 4.23: Čas izvajanja rekonstrukcije dvobarvne slike glede na delež znanih podatkov

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

Ko bodo vsa prejšnja poglavja končana, se bom lahko lotil pisanja zaključka. V njem bom opisal kaj vse sem tekom pisanja diplomske naloge ugotovil ter opravil, ter povedal če sem z rezultati zadovoljen. Omenil bom tudi kaj bi lahko spremenil in kako bi algoritme nadgradil, ter omenil par idej, na katerih raziskovalci trenutno delajo, ter opisal kako se področje razvija.



# Literatura

- [1] Jian-Feng Cai, Emmanuel J. Candès in Zuowei Shen. *A Singular Value Thresholding Algorithm for Matrix Completion*. 2008. arXiv: 0810.3286 [math.OC].
- [2] Maryam Fazel. “Matrix rank minimization with applications”. PhD dežertacija. Stanford, CA: Stanford University, mar. 2002.
- [3] 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.
- [4] 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.
- [5] 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>.
- [6] 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>.

- [7] 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).