CHALMERS

GÖTEBORGS UNIVERSITET

# Lågrangkomplettering av matriser

En komparativ studie av tre algoritmer

## Low-Rank Matrix Completion

*Kandidatarbete inom civilingenjörsutbildningen vid Chalmers*

Erik Albinsson
Petter Ollinen
Allan Vahlenberg

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2025

# Lågrangkomplettering av matriser

En komparativ studie av tre algoritmer

*Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk fysik vid Chalmers*
Petter Ollinen

*Kandidatarbete i matematik inom civilingenjörsprogrammet Teknisk matematik vid Chalmers*
Erik Albinsson Allan Vahlenberg

Handledare: Andrii Dmytryshyn

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2025

# Förord

Till att börja med vill vi som jobbat på arbetet tacka vår alldeles ypperliga handledare Andrii Dmytryshyn, utan honom hade det inte blivit någon rapport.

Vi har alla gruppmedlemmar gjort ändringar i alla delar av arbetet, men det finns ändå några aspekter där vi vill lyfta fram individprestationer. Erik Albinsson läste in sig på teorin kring och skrev avsnittet om NIHT i stort sett helt själv. Petter Ollinen har varit extra delaktig i skapandet av tabeller och diagram och andra strukturella delar av rapporten och har även generellt skrivit mest. Allan Vahlenberg har varit huvudansvarig för implementationen av SVT och IRLS-GP, samt allting relaterat till verklig data. För en detaljerad uppdelning av vilket arbete som gjorts av vem hänvisas läsaren till vår dagbok. Där står varje timme som vi lagt ned på projektet.

På nästa sida finns tabell över vem eller vilka som kan anses ha huvudsakligt ansvar för varje delkapitel.

| | § | Rubrik | Huvudansvarig författare |
|---|---|---|---|
| | - | Populärvetenskaplig presentation | Petter |
| | - | Sammandrag/Abstract | Samtliga |
| **1** | | **Inledning** | - |
| | 1.1 | Syfte | Samtliga |
| | 1.2 | Problemformulering | Samtliga |
| | 1.3 | Avgränsningar | Samtliga |
| **2** | | **Teori** | - |
| | 2.1 | Singulärvärdesdekomposition (SVD) | Allan & Petter |
| | 2.2 | Matrisnormer | Petter |
| | 2.3 | Lågrang matrisapproximation med SVD | Petter |
| | 2.4 | Minimering av rang | Erik |
| | 2.5 | SVT algoritmen | Samtliga |
| | 2.6 | IRLS-GP algoritmen | Erik |
| | 2.7 | NIHT algoritmen | Erik |
| **3** | | **Metod** | - |
| | 3.1 | Parameterval | Petter |
| | 3.2 | Syntetisk data | Samtliga |
| | 3.3 | Verklig data | Allan & Petter |
| | 3.3.1 | Netflix Prize | Allan |
| | 3.3.2 | Jester Collaborative Filtering Dataset | Allan |
| | 3.3.3 | Etiskt relevanta aspekter | Allan |
| **4** | | **Resultat** | - |
| | 4.1 | Parameterval | - |
| | 4.1.1 | SVT:s $\tau$ och $\delta$ | Petter |
| | 4.1.2 | IRLS-GP:s $p$ och $\gamma^0$ | Erik & Petter |
| | 4.1.3 | NIHT:s $r$ | Allan & Erik |
| | 4.2 | Prestation på syntetisk data | - |
| | 4.2.1 | SVT | Erik & Petter |
| | 4.2.2 | IRLS-GP | Erik & Petter |
| | 4.2.3 | NIHT | Erik & Petter |
| | 4.3 | Prestation på verklig data | Allan |
| **5** | | **Diskussion** | - |
| | 5.1 | Allmän diskussion kring resultat på syntetisk data | - |
| | 5.1.1 | SVT | Petter |
| | 5.1.2 | IRLS-GP | Samtliga |
| | 5.1.3 | NIHT | Samtliga |
| | 5.2 | Diskussion kring resultat på verklig data | Allan & Erik |
| | 5.3 | Slutsatser | Erik |

## Populärvetenskaplig presentation

Om Alice och Bob nästan alltid brukar ha samma åsikt om en film de sett så borde vi kunna göra en bra gissning om vad Alice skulle tycka om en film som Bob älskar.

Vi börjar med ett litet exempel för att illustrera detta. Alice, Bob, Charlie och David har betygsatt några filmer som de har sett med betyg mellan 1 och 5. De filmer som de ännu inte hunnit se har de givetvis inte betygsatt vilket är markerat med ?. Kan vi utifrån något mönster gissa vad ? ska vara? David och Bob har satt samma betyg på både film 2 och på film 4 så deras

| | Film 1 | Film 2 | Film 3 | Film 4 |
|---|---|---|---|---|
| Alice | 3 | ? | 1 | 4 |
| Bob | 2 | 3 | ? | 1 |
| Charlie | 5 | 4 | 3 | 5 |
| David | ? | 3 | ? | 1 |

filmpreferenser verkar vara ganska lika och det verkar troligt att David hade gett film 1 en 2:a precis som Bob. Även för den som gillar sudoku blir detta pusslande svårt och till slut omöjligt när antalet filmer och användare ökar. Du har förmodligen aldrig träffat den personen vars filmsmak bäst matchar din egen.

Låt oss anta att det enda som spelar roll för betyget är vilken genre filmen är och hur mycket skärmtid en viss skådespelare får i filmen, det vill säga två oberoende faktorer. Om vi vet att det är just två olika faktorer som påverkar deras sammantagna omdöme om filmerna kan vi faktiskt ta reda på de okända värdena ?, oberoende om vi faktiskt vet vilka faktorerna är. Det finns nämligen bara en unik kombination av siffror som gör att alla redan kända recensioner kan förklaras med enbart två faktorer. Om det istället hade kunnat förklaras av tre olika faktorer hade möjliga lösningar varit fler och om alla fyra hade haft helt unika smaker hade siffrorna kunnat vara nästan vad som helst.

När det finns miljontals av användare och många tusentals filmer spelar det exakta antalet faktorer som påverkar smaken inte så stor roll för vilka siffror som är möjliga. Så länge faktorerna som påverkar är mycket färre än om alla hade haft en unik smak kan vi alltså göra en mycket kvalificerad gissning vad de saknade värdena ska vara. På samma sätt kan din favoritstreamingtjänst rekommendera dig en film som du kommer att tycka om baserat på vilka filmer du har gillat tidigare.

Vi undersöker hur väl olika algoritmer klarar av att fylla i de saknade värdena. Det är ganska självklart att du inte kan få några bra rekommendationer om du inte har betygsatt några filmer alls. Men hur många filmer måste folk egentligen ha sett och betygsatt för att ett rekommendationssystem som bygger på det här konceptet ska kunna ge dig bra rekommendationer? Sådana här faktorer undersöks i den här uppsatsen, först på syntetiserad data innan det mynnar ut i bland annat ett test på riktig data insamlad av Netflix.

Konceptet med att utgå från antalet oberoende faktorer har många fler tillämpningar än bara för rekommendationer. Samma princip kan användas för att återställa skadade bilder, sammanställa fullständig genstruktur från flera ofullständiga delar, få maskininlärningsmodeller som är tränade på ett fall att fungera på många andra liknande fall, samt för att självkörande bilar skall kunna hålla koll på skymda objekt i omgivningen. Lågrangkomplettering av matriser, som är namnet på konceptet och dessutom titeln på detta arbetet, är därför ett viktigare och viktigare ämne i en alltmer datadriven värld.

**Sammandrag**

Denna kandidatuppsats jämför tre algoritmer för lågrang matriskomplettering: Singular Value Thresholding (SVT), Iterative Reweighted Least Squares Gradient Projection (IRLS-GP) och Normalized Iterative Hard Thresholding (NIHT). Algoritmernas prestanda utvärderas på både syntetisk och verklig data för att identifiera deras styrkor och brister under olika förutsättningar. Resultaten visar att SVT lämpar sig bäst då hög kompletteringsnoggrannhet är viktigt för ett stort urval av matriser av olika typ. IRLS-GP fungerar mycket bra på specifika lågrangmatriser men är starkt beroende av ett parameterval där antingen stabilitet eller tidsåtgång prioriteras. Om den sanna matrisrangen är känd i förväg, visar sig NIHT ofta vara det mest effektiva valet.

**Abstract**

This bachelor thesis compares three algorithms designed for low-rank matrix completion: Singular Value Thresholding (SVT), Iterative Reweighted Least Squares Gradient Projection (IRLS-GP) and Normalized Iterative Hard Thresholding (NIHT). The performance of the algorithms is evaluated on both synthetic and real-world data to identify their strengths and weaknesses under varying conditions. The results show that SVT is most suitable when high reconstruction accuracy is desired for a wide variety of different matrices. IRLS-GP works very well on specific low-rank matrices but relies heavily on a parameter choice where either stability or time consumption is prioritized. When the true rank of the matrix is known in advance, NIHT is often the most effective choice.

# Innehåll

**1 Inledning** **1**
1.1 Syfte . . . 1
1.2 Problemformulering . . . 1
1.3 Avgränsningar . . . 1

**2 Teori** **1**
2.1 Singulärvärdesdekomposition (SVD) . . . 1
2.2 Matrisnormer . . . 2
2.3 Lågrang matrisapproximation med SVD . . . 2
2.4 Minimering av rang . . . 4
2.5 SVT algoritmen . . . 4
2.6 IRLS-GP algoritmen . . . 6
2.7 NIHT algoritmen . . . 8

**3 Metod** **9**
3.1 Parameterval . . . 10
3.2 Syntetisk data . . . 10
3.3 Verklig data . . . 10
3.3.1 Netflix Prize . . . 11
3.3.2 Jester Collaborative Filtering Dataset . . . 11
3.3.3 Etiskt relevanta aspekter . . . 11

**4 Resultat** **11**
4.1 Parameterval . . . 11
4.1.1 SVT:s $\tau$ och $\delta$ . . . 11
4.1.2 IRLS-GP:s $p$ och $\gamma_c$ . . . 12
4.1.3 NIHT:s $r$ . . . 13
4.2 Prestation på syntetisk data . . . 13
4.2.1 SVT . . . 14
4.2.2 IRLS-GP . . . 15
4.2.3 NIHT . . . 16
4.3 Prestation på verklig data . . . 16

**5 Diskussion** **17**
5.1 Allmän diskussion kring resultat på syntetisk data . . . 17
5.1.1 SVT . . . 17
5.1.2 IRLS-GP . . . 18
5.1.3 NIHT . . . 18
5.2 Diskussion kring resultat på verklig data . . . 19
5.3 Slutsatser . . . 19

**A Källkod** **i**
A.1 Generera matris . . . i
A.2 SVT kod . . . i
A.3 IRLS-GP kod . . . ii
A.4 NIHT kod . . . iv
A.5 Tester på syntetiska matriser . . . v

# 1 Inledning

Det är inom många områden vanligt förekommande att endast partiell data kan samlas in. Det kan bero på vitt skilda faktorer såsom att det är omotiverat dyrt eller på annat sätt kostsamt att utföra experiment eller för att den önskade datamängden helt enkelt inte är tillgänglig. Ett känt exempel är hur Netflix 2007 utlyste en tävling för att förbättra deras filmrekommendationssystem [10]. Med deras miljontals användare, ofantligt stora bibliotek och en affärsidé som bygger på att deras användare inte ska ha sett alla filmer, saknades givetvis data i termer av användarrecensioner. Att rekommendera nya filmer till sina användare baserat på vad de tyckt om tidigare är ekvivalent med att fylla i, eller åtminstone gissa, de saknade recensionerna, det vill säga datapunkterna. Många av de presenterade lösningarna i tävlingen utgick från det koncept som detta arbete är tänkt att djupdyka i, nämligen lågrangkomplettering av matriser.

Det huvudsakliga antagandet som en sådan komplettering bygger på är, som namnet antyder, att matrisen är av låg rang. I termer av Netflixexemplet innebär detta att det inte finns lika många filmsmaker som det finns användare. Mer matematiskt innebär det att den matris som data ordnas i endast har ett fåtal linjärt oberoende komponenter. Utöver det nämnda Netflix-problemet finns det många andra fall där detta antagande antingen är rimligt eller nödvändigt. Exempelvis är det då möjligt att reducera komplexiteten av ett annars omöjligt problem till en sådan nivå där det kan lösas.

## 1.1 Syfte

Syftet med projektet är att komplettera lågrangmatriser med hjälp av algoritmerna SVT, IRLS-GP och NIHT för att sedan utvärdera dem med avseende på tidsåtgång och noggrannhet på syntetiska och verkliga datamängder. Därutöver är syftet att undersöka vilka dataegenskaper som påverkar algoritmernas prestation.

## 1.2 Problemformulering

Specifikt ämnar vi svara på följande frågor:

- Hur påverkar valet av parametervärde respektive algoritms prestation och hur bör de väljas?
- Hur presterar algoritmerna på syntetisk respektive verklig data?
- Vilka faktorer spelar störst roll för algoritmval?

## 1.3 Avgränsningar

De valda algoritmparametrarna och datamängderna är begränsade av prestanda motsvarande en normal persondator. Vidare är urvalet av datamängder begränsat. Som syntetisk data används uteslutande matriser med Gaussiskt fördelade värden mellan 0 och 1. De två verkliga datamängderna är båda inom området recensioner vilket inte påvisar prestation inom andra tillämpningar.

Teorin som presenteras är nära anknuten till linjär algebra och i synnerhet singulärvärdesdekomposition. Teori kopplad till exempelvis optimering, däribland begreppet konvexitet, kommer behandlas mycket ytligt.

# 2 Teori

I följande kapitel presenteras först den matematiska bakgrunden och teorin som är nödvändig för kompletteringsproblemet. Detta mynnar ut i en presentation av de tre algoritmerna.

## 2.1 Singulärvärdesdekomposition (SVD)

Ett kraftfullt verktyg för att extrahera information från matriser är singulärvärdesdekomposition. Detta är en faktorisering som bryter ner matrisen i tre olika komponenter. Antag att $\mathbf{A}$ är en matris

av storlek $m \times n$ där att utan förlust av generalitet $m \geq n$. Då gäller det att $\mathbf{A}$ kan faktoriseras enligt

$$\begin{bmatrix} A_{1,1} & \dots & A_{1,n} \\ \vdots & & \vdots \\ A_{m,1} & \dots & A_{m,n} \end{bmatrix} = \begin{bmatrix} U_{1,1} & \dots & U_{1,m} \\ \vdots & & \vdots \\ U_{m,1} & \dots & U_{m,m} \end{bmatrix} \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_n \\ \vdots & & \vdots \\ 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} V_{1,1} & \dots & V_{1,n} \\ \vdots & & \vdots \\ V_{n,1} & \dots & V_{n,n} \end{bmatrix}^* . \tag{1}$$

Då $\mathbf{A} \in \mathbb{C}^{m \times n}$ är $\mathbf{U} := [U_{i,j}]$ och $\mathbf{V} := [V_{i,j}]$ unitära matriser, den komplexa motsvarigheten till ortogonala matriser, av storlek $m \times m$ respektive $n \times n$. $\mathbf{\Sigma}$ är en $m \times n$ rektangulär diagonal matris med de singulära värdena $\sigma_i$ där $i \in [1, n]$. $\mathbf{V}^\star$ är det komplexkonjugerade transponatet av $\mathbf{V}$. Det är vanligt förekommande att, utan förlust av generalitet, anta att elementen i $\mathbf{\Sigma}$ är ordnade så att $\sigma_1 \geq \dots \sigma_n \geq 0$ vilket fortsättningsvis kommer vara underförstått. Låt oss precisera detta i följande sats.

**Sats 2.1.** [12, s.367] Låt $\mathbf{A} \in \mathbb{C}^{m \times n}$ vara en godtycklig matris. Då kan $\mathbf{A}$ faktoriseras enligt

$$\mathbf{A} = \mathbf{U\Sigma V}^* \tag{2}$$

där kolumner i $\mathbf{U} \in \mathbb{C}^{m \times m}$ är egenvektorer till $\mathbf{AA}^*$, kolumner i $\mathbf{V} \in \mathbb{C}^{n \times n}$ är egenvektorer till $\mathbf{A}^*\mathbf{A}$, och $\mathbf{\Sigma}$ är en rektangulär diagonalmatris med diagonalelement $\sqrt{\lambda_i}$ för egenvärdena $\lambda_i$ tillhörande både $\mathbf{AA}^*$ och $\mathbf{A}^*\mathbf{A}$. Vidare kallas de nollskilda diagonalelementen i $\mathbf{\Sigma}$ singulära värden $\sigma_i$ till $\mathbf{A}$.

Notera att antalet singulära värden är detsamma som $\text{rang}(\mathbf{A})$.

## 2.2 Matrisnormer

För att jämföra olika matriser och på så sätt kunna utvärdera hur lik en approximation är en känd matris är det nödvändigt att införa två olika typer av matrisnormer.

**Definition 1** ([12, s.480])**.** Låt $\mathbf{A} \in \mathbb{C}^{m \times n}$. Då definieras Frobeniusnormen $\|\mathbf{A}\|_F$ av $\mathbf{A}$ enligt

$$\|\mathbf{A}\|_F := \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{i,j}|^2} = \sqrt{\text{tr}(\mathbf{AA}^*)}, \tag{3}$$

där $A_{i,j}$ är element i $\mathbf{A}$.

Vi utnyttjar Frobeniusnormen för att jämföra matriser och i synnerhet skillnaden mellan sådana.

**Definition 2** ([3, s.721])**.** Låt $\mathbf{A} \in \mathbb{C}^{m \times n}$. Då definieras den nukleära normen $\|\mathbf{A}\|_*$ av $\mathbf{A}$ enligt

$$\|\mathbf{A}\|_* := \sum_{i=1}^{n} \sigma_i = \text{tr}\left(\sqrt{\mathbf{A}^*\mathbf{A}}\right), \tag{4}$$

där $\sigma_i$ är det $i$:te största singulära värdet till $\mathbf{A}$.

Den nukleära normen är användbar då den kan användas som surrogat för rang, se kapitel 2.4.

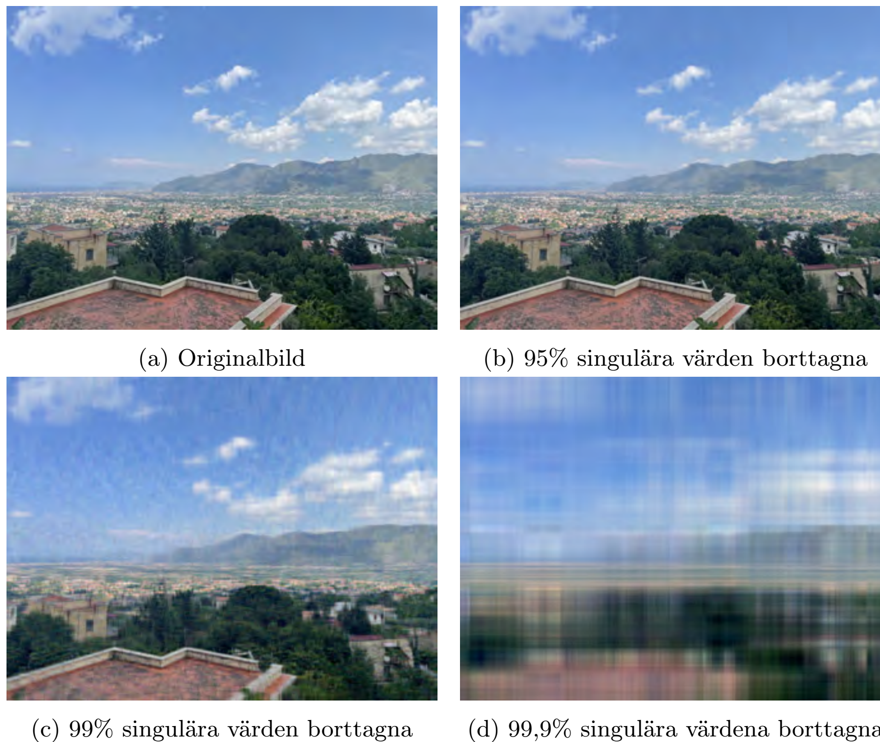
## 2.3 Lågrang matrisapproximation med SVD

En viktig tillämpning av SVD är att med de singulära värdena identifiera mönster i en matris. Låt $\sigma_1, \dots, \sigma_r$ vara singulära värden till en matris $\mathbf{A}$ med rang $r$. Det visas i [4, s.216] att den bästa approximationen av rang $s$ till $\mathbf{A}$, sett till Frobeniusnormen, erhålls genom singulärvärdesdekomposition där samtliga singulära värden mindre än de $s$ största singulära värdena sätts till 0. Det innebär att $\mathbf{A}_{\text{approx}}$ i ekvation (5) är den bästa approximationen av rang $s$ som är möjlig att göra

för $\mathbf{A}$.

$$\mathbf{A} \approx \mathbf{A}_{\text{approx}} = \begin{bmatrix} U_{1,1} & \dots & U_{1,m} \\ \vdots & & \vdots \\ U_{m,1} & \dots & U_{m,m} \end{bmatrix} \begin{bmatrix} \sigma_1 & \dots & 0 & 0 & \dots & 0 \\ \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & \dots & \sigma_s & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & \dots & 0 \\ \vdots & & \vdots & \vdots & & \vdots \\ 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} V_{1,1} & \dots & V_{1,n} \\ \vdots & & \vdots \\ V_{n,1} & \dots & V_{n,n} \end{bmatrix}^* . \tag{5}$$

**Exempel 2.1.** I figur 1 visas hur en bild kan approximeras med matriser av lägre rang genom att sätta de minst signifikanta singulära värdena till 0. Det vill säga att om bilden från början är av rang $r$ så approximeras den med matriser med rang $\lfloor 0{,}05r \rfloor$; $\lfloor 0{,}01r \rfloor$ respektive $\lfloor 0{,}001r \rfloor$.



(a) Originalbild

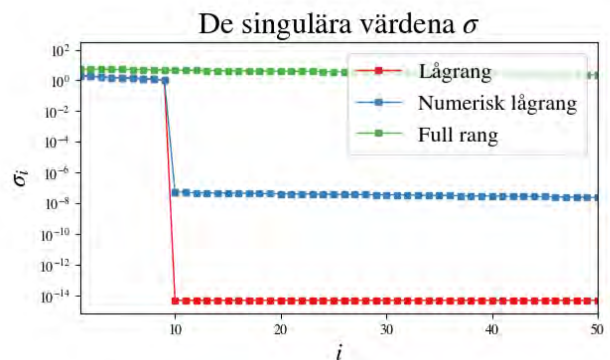
(b) 95% singulära värden borttagna

(c) 99% singulära värden borttagna

(d) 99,9% singulära värdena borttagna

Figur 1: Påverkan på bild då minst signifikanta singulära värden är borttagna. Originalbild fotograferad i Italien av Erik Albinsson.

Om någonting är numeriskt approximativt lågrang innebär detta inte att alla singulära värdena större än den rangen är 0 som ovan beskrivits. I figur 2 ses hur storleken på de singulära värden skiljer sig åt för en matris som är av full rang, en matris som är approximativt lågrang och en matris som de facto är lågrang. Att den riktiga lågrangmatisens singulära värden inte är identiskt lika med 0 beror på numeriska fel och maskinprecision i SVD-uppdelningen.




Figur 2: Storleken för de singulära värdena för tre olika matriser med olika rangegenskaper.

## 2.4 Minimering av rang

En essentiell aspekt av att lösa matriskomplettering och dataåterhämtning är att lösa minimeringsproblemet

$$\begin{aligned} &\text{minimera rang}(\mathbf{A}) \\ &\text{givet att } A_{ij} = M_{ij},\ (i,j) \in \Omega, \end{aligned} \tag{6}$$

där $\Omega$ är mängden som innehåller positioner $(i,j)$ där $\mathbf{M}$ har kända element definierad som $\Omega = \{(i,j)|M_{ij} \text{ är observerad}\}$. Artikel [3] beskriver att problemet i (6) är ett NP-svårt problem [5] och alltså inte av stor praktisk användning. Vi vill istället finna en surrogatfunktion för matrisrangen som är enklare att minimera. Ponera att vi har en matris $\mathbf{M}$ med dimensioner $m \times n$ och rang $r$ med $|\Omega|$ observerade element, då finns det konstanter $C, c$ så att om

$$|\Omega| \geq Cm^{1,2} r \log(m)$$

håller, då gäller det att lösningen till den konvexa avslappningen [1, s.88]

$$\begin{aligned} &\text{minimera } ||\mathbf{A}||_* \\ &\text{givet att } A_{ij} = M_{ij},\ (i,j) \in \Omega \end{aligned} \tag{7}$$

är unik och lika med $\mathbf{M}$ med sannolikhet $p \geq 1 - cm^{-3}\log(m)$ [3]. Alltså återhämtar lösningen till minimeringsproblemet (7) $\mathbf{M}$ med hög noggrannhet även för enbart ett fåtal kända element då $\mathbf{M}$ har låg rang.

## 2.5 SVT algoritmen

SVT (singular value thresholding) algoritmen [2] bygger på det tidigare nämnda konceptet att vi kan approximera det NP-svåra optimeringsproblemet (6) med den konvexa avslappningen (7) vilket i SVT ytterligare modifieras till (8). Operatorn $\mathcal{P}_\Omega$ projicerar elementen i $\mathbf{X}$ på de kända elementen och är definierad i definition 4. Termen med den kvadrerade Frobeniusnormen straffar avvikande värden på de kända elementen och $\tau$ är en positiv parameter.

$$\begin{aligned} &\text{minimera} \qquad \tau \|\mathbf{X}\|_* + \frac{1}{2}\|\mathbf{X}\|_F^2 \\ &\text{givet att } \mathcal{P}_\Omega(\mathbf{X}) = \mathcal{P}_\Omega(\mathbf{M}). \end{aligned} \tag{8}$$

Innan vi kan bygga vidare på algoritmen behöver vi introducera några centrala begrepp. Vi behöver en krympningsoperator som ser till att vi, i varje iteration, skapar en lågrangmatris.

**Definition 3** (Krympningsoperatorn (Shrinkage operator) $\mathcal{D}_\tau$)**.** För en matris $\mathbf{A}$ med $\mathbf{A} = \mathbf{U\Sigma V}^*$ definieras operatorn som

$$\mathcal{D}_\tau(\mathbf{A}) := \mathbf{U}\,\text{diag}\left(\max(\sigma_i - \tau, 0)\right)\,\mathbf{V}^*$$

där $\tau$ är en positiv parameter som avgör hur mycket de singulära värdena minskas i varje iteration.

För att hantera mängden av de observerade elementen $\Omega$ i en matris $\mathbf{M}$ är det användbart att införa projektionsoperatorn $\mathcal{P}_\Omega$ som projicerar en matris $\mathbf{X}$ på matrisrummet inducerat av $\Omega$.

**Definition 4** (Projektion på observerade element)**.** Låt $\Omega$ vara en godtycklig mängd av talpar $(i,j)$, då definieras projektionsoperatorn $\mathcal{P}_\Omega$ agerande på matrisen $\mathbf{X}$ enligt

$$(\mathcal{P}_\Omega(\mathbf{X}))_{ij} := \begin{cases} X_{ij} & \text{om } (i,j) \in \Omega, \\ 0 & \text{annars.} \end{cases}$$

I SVT algoritmen finns det två parametrar, nämligen $\delta$ och $\tau$. I varje iteration tas ett steg riktat så att vi närmar oss $M_{ij}$ för alla $(i,j) \in \Omega$, alltså de kända värdena. Steglängden betecknas $\delta$ och valet av $\delta$ påverkar därmed konvergenshastigheten. Optimal storlek på $\delta$ varierar mellan problem men empiriska studier [2] pekar på att ett bra val är $\delta = 1{,}2\frac{mn}{|\Omega|}$, där $m \times n$ är storleken

på matrisen $\mathbf{M}$ och $|\Omega|$ är antalet kända element i $\mathbf{M}$. I [2] skrivs det att algoritmen är garanterad att konvergera $\forall \delta \in (0, 2)$ från vilket faktumet att konvergens inte är garanterad för matriser med mer än 40% saknade värden, då $\delta$ väljs enligt ovan, blir uppenbart. Parametern $\tau$ återfinns i både (8) och i $\mathcal{D}_\tau$. I [2] framgår det att Frobeniusnormen och den nukleära normen för en kvadratisk matris med storlek $n \times n$ med Gaussiskt fördelade element mellan 0 och 1 har väntevärden på $n\sqrt{r}$ respektive $nr$. En lämplig viktning i (8) anses vara att termen med den nukleära normen skall vara 10 gånger så stor som den med den kvadrerade Frobeniusnormen[2]. Således är utgångspunkten $\tau = 5n$ vilket givetvis skalas beroende på de förväntade värdena.

Det har visat sig i både egna numeriska experiment och påpekats i [2] att med startmatris $Y^0 = \mathbf{0}$ minskar inte residualerna i de första iterationerna. Genom ett smart val av startmatris för den första iterationen $\mathbf{Y}^0 = k_0 \delta \mathcal{P}_\Omega(\mathbf{M})$, med $k_0$ definierat enligt (9), kan dessa effektivt hoppas över,

$$k_0 := \left\lceil \frac{\tau}{\delta \left\| \mathcal{P}_\Omega(\mathbf{M}) \right\|_2} \right\rceil \tag{9}$$

Med detta sagt kan vi nu beskriva hur algoritmen som beskrivs i [2] fungerar steg för steg.

- Initialisera en matris $\mathbf{Y}^0 = k_0 \delta \mathcal{P}_\Omega(\mathbf{M})$ som uppdateras under iterationerna. Sätt $k = 0$.
- Beräkna för varje iteration $k \geq 0$ singulärvärdesdekompositionen $\mathbf{Y}^k = \mathbf{U}\Sigma\mathbf{V}^*$ där sedan krympningsoperatorn används för att minska eller nollsätta de mindre signifikanta singulära värdena $\mathcal{D}_\tau(\mathbf{Y}^k) = \mathbf{U}\text{diag}(\max(\sigma_i - \tau), 0)\mathbf{V}^*$.

  Sätt sedan $\mathbf{X}^{k+1} = D_\tau(\mathbf{Y}^k)$ som den nya approximationen av $\mathbf{M}$ samt uppdatera $\mathbf{Y}^{k+1} = \mathbf{Y}^k + \delta \mathcal{P}_\Omega(\mathbf{M} - \mathbf{X}^{k+1})$. Sätt $k = k + 1$
- Fortsätt tills stoppvillkor eller eventuella maximala antal iterationer har uppnåtts. Residualen beräknas i [2] och denna rapport som

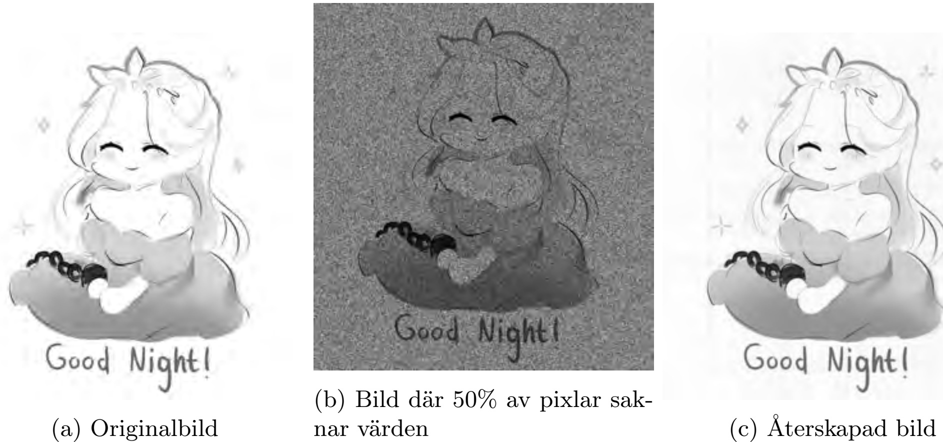
  $$\epsilon_{\text{SVT}} := \frac{||\mathcal{P}_\Omega(\mathbf{X}^k - \mathbf{M})||_F}{||\mathcal{P}_\Omega(\mathbf{M})||_F}, \tag{10}$$

  där stoppvillkoret är $\epsilon_{\text{SVT}} \leq \epsilon_{\text{tol}}$ för en förutbestämd toleransnivå $\epsilon_{\text{tol}}$.
- Returnera $\mathbf{X}^k$ som en lågrangkomplettering av $\mathbf{M}$

Fullständig pseudokod för SVT algoritmen så som den är presenterad i artikel [2] återfinns i algoritm 1.

Vi kan nu komplettera matriser som innehåller okända element, vilka kan vara skadad data eller data som inte insamlats som behöver approximeras. Vi visar detta i följande exempel.

**Exempel 2.2.** I figur 3 visas en bild där 50% av slumpmässigt valda pixlar saknar värden. Trots det kan bilden återhämtas med god noggrannhet.




(a) Originalbild (b) Bild där 50% av pixlar saknar värden (c) Återskapad bild

Figur 3: Ett exempel på hur SVT kan användas för att återhämta en bild fastän 50% av dess värden saknas. Originalbild ritad av Nisa-Nur Cann, används med hennes godkännande.

**Algoritm 1** Pseudokod för SVT algoritmen från [2]

**Input:** $\Omega$ and sampled entries $\mathcal{P}_\Omega(\mathbf{M})$, step size $\delta$, tolerance $\epsilon$, parameter $\tau$ and maximum iteration count $k_{\max}$
**Output:** $\mathbf{M}_{\text{comp}}$
1: **Description:** Recover a low-rank matrix $M$ from a subset of sampled entries
2: Set $\mathbf{Y}^0 = k_0\delta\mathcal{P}_\Omega(\mathbf{M})$
3: Set $r_0 = 0$
4: **for** $k = 1$ to $k_{\max}$ **do**
5: Set $s_k = r_{k-1} + 1$
6: **repeat**
7: Compute $[\mathbf{U}^{k-1}, \mathbf{\Sigma}^{k-1}, \mathbf{V}^{k-1}]_{s_k}$
8: Set $s_k = s_k + \ell$
9: **until** $\sigma^{k-1}_{s_k-\ell} \le \tau$
10: Set $r_k = \max\{j : \sigma^{k-1}_j > \tau\}$
11: Set $\mathbf{X}^k = \sum_{j=1}^{r_k}(\sigma^{k-1}_j - \tau)U^{k-1}_j V^{k-1}_j$
12: **if** $\frac{\|\mathcal{P}_\Omega(\mathbf{X}^k-\mathbf{M})\|_F}{\|\mathcal{P}_\Omega\mathbf{M}\|_F} \le \epsilon$ **then**
13: **break**
14: **end if**
15: **for** each $(i,j)$ **do**
16: **if** $(i,j) \notin \Omega$ **then**
17: Set $Y^k_{ij} = 0$
18: **else**
19: Set $Y^k_{ij} = Y^{k-1}_{ij} + \delta(M_{ij} - X^k_{ij})$
20: **end if**
21: **end for**
22: **end for**
23: **return** $\mathbf{M}_{\text{comp}} = \mathbf{X}^k$ as the low-rank completion of $\mathbf{M}$.

## 2.6 IRLS-GP algoritmen

Den andra algoritmen som används för matriskomplettering för matriser med låg rang som vi kommer undersöka är en IRLS (iterative reweighted least squares) algoritm, mer specifikt IRLS-GP (iterative reweighted least squares gradient projection) algoritmen. En central komponent i IRLS-GP algoritmen är den släta Schatten-p funktionen.

**Definition 5** (Släta Schatten-p funktionen [8])**.** Den släta Schatten-p funktionen definieras som

$$f_p(X) := \operatorname{tr}((\mathbf{X}^*\mathbf{X} + \gamma\mathbf{I})^{p/2}) = \sum_{i=1}^{n}(\sigma_i^2(\mathbf{X}) + \gamma)^{p/2}, \tag{11}$$

där $0 \le p$ och $\gamma > 0$ är positiva konstanter.

Några egenskaper för den släta Schatten-p funktionen är att då $\gamma = 0$ och $p = 1$ gäller $f_1(\mathbf{X}) = ||\mathbf{X}||_*$. Även, då $p > 0$ är $f_p(\mathbf{X})$ deriverbar samt då $p \ge 1$ är $f_p(\mathbf{X})$ konvex.

Utifrån definitionen av den släta Schatten-p funktionen (11) går det även att härleda följande lemma.

**Lemma 2.1** ([8])**.** Då $p = 1$ kan vi för släta Schatten-1 funktionen härleda olikheten

$$||\mathbf{X}||_* \le f_1(\mathbf{X}) \le ||\mathbf{X}||_* + m\sqrt{\gamma},$$

där $m \times n$ är storleken på matrisen $\mathbf{X}$.

Lemma 2.1 visar att $f_1(\mathbf{X})$ är en övre begränsning men även en nära approximation till den nukleära normen då $\gamma$ är litet, samt att då $\gamma \to 0$ går även $f_1(\mathbf{X})$ mot $||\mathbf{X}||_*$. Detta är egenskaper som algoritmen utnyttjar.

Istället för att minimera problemet (7) med nukleära normen minimerar IRLS-GP algoritmen den släta Schatten-p funktionen som fungerar som en deriverbar och nära surrogatfunktion till

problemet som vi vill lösa. Alltså approximerar vi nukleära normen som

$$\begin{aligned}||\mathbf{X}||_* = \mathrm{tr}((\mathbf{X}^*\mathbf{X})^{1/2}) \approx f_p(\mathbf{X}) = \mathrm{tr}((\mathbf{X}^*\mathbf{X} + \gamma\mathbf{I})^{p/2}) = \\ = \mathrm{tr}((\mathbf{X}^*\mathbf{X} + \gamma\mathbf{I})^{p/2-1}(\mathbf{X}^*\mathbf{X} + \gamma\mathbf{I})) = \\ = \mathrm{tr}(\mathbf{W}_p(\mathbf{X}^*\mathbf{X} + \gamma\mathbf{I})) \approx \mathrm{tr}(\mathbf{W}_p\mathbf{X}^*\mathbf{X}),\end{aligned}$$

där $\mathbf{W}_p$ är en viktmatris definierad som $\mathbf{W}_p := (\mathbf{X}^*\mathbf{X} + \gamma\mathbf{I})^{p/2-1}$ [8].

För att nu minimera Frobeniusnormen $\mathrm{tr}(\mathbf{W}_p\mathbf{X}^*\mathbf{X})$ använder IRLS-GP metoden Frobeniusnormens deriverbara egenskap och utför iterativt en gradientnedstigning för varje iteration $k \geq 0$ och sätter $\mathbf{X}_{\text{temp}} = \mathbf{X}_{\text{temp}} - s^k\nabla\mathrm{tr}(\mathbf{W}_p^k\mathbf{X}_{\text{temp}}^*\mathbf{X}_{\text{temp}})$, där $\nabla\mathrm{tr}(\mathbf{W}_p^k\mathbf{X}_{\text{temp}}^*\mathbf{X}_{\text{temp}}) = 2\mathbf{X}_{\text{temp}}\mathbf{W}_p^k$ och $s^k$ är en stegstorlek parameter som definieras som $s^k := \frac{\gamma_{k+1}^{1-p/2}}{2}$ [8].

De två för algoritmen relevanta parametrarna är $p$ och $\gamma_0$. $p$ kan väljas fritt i intervallet $[0, 1]$ med konsekvenser för både stabilitet och beräkningstid. $\gamma_0$ väljs enligt [8] till $\gamma_0 = \gamma_c \|\mathbf{M}\|_2$ där $\gamma_c$ är en skalningsfaktor som enligt [8] bör väljas till $\gamma_c = 1 \times 10^{-2}$.

Vi kan nu beskriva hur IRLS-GP algoritmen fungerar.

- Initialisera en matris $\mathbf{X}^0 = \mathbf{0}_{m\times n}$, $\gamma_0 > 0$ och $s^0 = \frac{\gamma_0^{1-p/2}}{2}$ som kommer uppdateras under iterationerna. Sätt $k = 0$.
- Beräkna för varje iteration $k \geq 0$ hjälpmatrisen $\mathbf{X}_{\text{temp}} = \mathbf{X}^k$ och viktmatrisen $\mathbf{W}_p^k = ((\mathbf{X}^k)^*\mathbf{X}^k + \gamma_k\mathbf{I})^{p/2-1}$. Utför därefter gradientnedstigning, tills ett konvergens villkor nås, iterativt som $\mathbf{X}_{\text{temp}} = \mathcal{P}_{\Omega^c}(\mathbf{X}_{\text{temp}} - s^k\mathbf{X}_{\text{temp}}\mathbf{W}^k) + \mathcal{P}_\Omega(\mathbf{M})$. Där $\Omega^c$ är de element vi inte känner till den ofullständiga matrisen $\mathbf{M}$. Alltså utförs gradientnedstigning på elementen vi inte känner till i $\mathbf{M}$ medan alla kända element är bevarade.

  Sätt sedan $\mathbf{X}^{k+1} = \mathbf{X}_{\text{temp}}$ och välj $\gamma_{k+1}$ så att $0 < \gamma_{k+1} \leq \gamma_k$ och låt $s^{k+1} = \frac{\gamma_{k+1}^{1-p/2}}{2}$. Sätt $k = k + 1$.
- Fortsätt tills stoppvillkor eller eventuella maximala antal iterationer har uppnåtts. Residualen beräknas i denna rapport som

  $$\epsilon_{\text{IRLS}} := \frac{||\mathbf{X}^k - \mathbf{X}^{k-1}||_F}{||\mathbf{M}||_F} \tag{12}$$

  där stoppvillkoret är $\epsilon_{\text{IRLS}} \leq \epsilon_{\text{tol}}$ för en förutbestämd toleransnivå $\epsilon_{\text{tol}}$.
- Returnera $\mathbf{X}^k$ som en lågrangkomplettering av $\mathbf{M}$

Nedan ses fullständig pseudokod för IRLS-GP algoritmen så som den är presenterad i artikel [8].

**Algoritm 2** Pseudokod för IRLS-GP algoritmen från [8]

```
Input: Ω and sampled entries P_Ω(M), parameter p and parameter γ_0
Output: M_comp
 1: Set k = 0. Initialize X^0 = 0, γ_0 > 0, s_0 = γ_0^(1−p/2);
 2: while IRLS iterates not converged do
 3:     W^k = ((X^k)^T X^k + γ_k I)^(p/2−1).
 4:     Set X_temp = X^k;
 5:     while Gradient projection iterates not converged do
 6:         X_temp = P_Ω^c(X_temp − s_k X_temp W^k) + P_Ω(X_0);
 7:     end while
 8:     Set X^(k+1) = X_temp;
 9:     Choose 0 < γ_(k+1) ≤ γ_k, s_(k+1) = γ_(k+1)^(1−p/2) / 2;
10:     k = k + 1;
11: end while
12: return M_comp = X^k as the low-rank completion of M.
```

## 2.7 NIHT algoritmen

Den tredje och sista algoritmen vi beskriver är en IHT (iterative hard thresholding) algoritm, där vi antar att rangen $r$ för matrisen vi vill komplettera är känd. Eftersom rangen är känd, löser vi inte längre minimeringsproblemet (6) som i de tidigare algoritmerna (SVT, IRLS-GP). Istället försöker IHT lösa följande minimeringsproblem:

$$\begin{aligned} &\text{minimera } ||\bar{b} - \mathcal{A}(\mathbf{X})||_2^2 \\ &\text{givet att } \operatorname{rang}(\mathbf{X}) = r. \end{aligned} \tag{13}$$

Här är $\mathcal{A}(\cdot)$ en operator som projicerar de kända värdena från en matris till en vektor [13] och $\bar{b} = \mathcal{A}(\mathbf{M})$ är de kända värdena från matrisen $\mathbf{M}$ vi försöker komplettera.

En vanlig metod för att lösa (13) är att iterativt ta steg mot målfunktionens negativa gradient, det vill säga i riktningen $2\mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}))$. Här fungerar $\mathcal{A}^*(\cdot)$ genom att omvandla en vektor till en matris med samma dimensioner som $\mathbf{M}$, där de kända värdena placeras på sina ursprungliga positioner och övriga element sätts till noll. Exempelvis är $\mathcal{A}^*(\bar{b}) = \mathbf{M}$.

Innan vi konstaterar hur detta används för att iterativt uppdatera $\mathbf{X}^{k+1}$ behöver vi följande definition.

**Definition 6** (Hård tröskeloperator (hard thresholding operator)[13])**.** Den hårda tröskeloperatorn definieras som $\mathcal{H}_r(\mathbf{X}) := \mathbf{U}\mathbf{\Sigma}_r\mathbf{V}^*$ där

$$\Sigma_r(i,i) := \begin{cases} \Sigma(i,i) & i \leq r, \\ 0 & i > r. \end{cases}$$

Den hårda tröskeloperatorn $\mathcal{H}_r(\mathbf{X})$ används för att begränsa matrisens rang till högst $r$. Detta görs genom att utföra SVD av $\mathbf{X}$ och sätta alla singulärvärden utöver de $r$ största till noll.

En enkel IHT metod för att lösa (13) är att iterativt uppdatera

$$\mathbf{X}^{k+1} = \mathcal{H}_r(\mathbf{X}^k + \mu\mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k))), \tag{14}$$

med konstant stegstorlek $\mu$. NIHT (normalized iterative hard threshold) är en utveckling av denna metod, där istället för ett konstant värde på $\mu$, uppdateras det iterativt för snabbare konvergens.

NIHT bygger på idén att då $\mathbf{X}^k$ närmar sig den sanna matrisen sker de största förändringarna främst i de singulära värdena snarare än i de singulära vektorerna. Därför sker uppdateringar i de riktningar som motsvarar de dominerande singulära vektorerna [13]. Därmed är följande definition användbar.

**Definition 7** (Projektion till singulära vektorer)**.** Projektion till det delrum som är spannet av de $r$ första vänstra respektive högra singulära vektorer definieras som $\mathbf{P}_{\mathbf{U}}^k := \mathbf{U}_k\mathbf{U}_k^*$ respektive $\mathbf{P}_{\mathbf{V}}^k := \mathbf{V}_k\mathbf{V}_k^*$, där $\mathbf{X}^k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^*$.

Eftersom vi vill minimera felet samtidigt som vi vill behålla matrisens låga rang, kan gradienten projiceras på delrum som definieras av de vänstra eller högra singulära vektorerna, eller båda samtidigt. De olika kombinationerna ger oss tre olika sökriktningar

$$\begin{aligned} \mathbf{W}_k^u &:= \mathbf{P}_{\mathbf{U}}^k\mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k)), \\ \mathbf{W}_k^v &:= \mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k))\mathbf{P}_{\mathbf{V}}^k, \\ \mathbf{W}_k^{uv} &:= \mathbf{P}_{\mathbf{U}}^k\mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k))\mathbf{P}_{\mathbf{V}}^k. \end{aligned}$$

Genom att använda någon av dessa sökriktningarna tvingar vi $\mathbf{X}^{k+1}$ att hålla sig inom de projicerade delrum vi får fram. Detta leder till att vi inte hittar den optimala lösningen om den inte ligger i de sökriktningarna som finns i det projicerade delrummet [13].

Även om dessa projicerade riktningar inte direkt används för att bestämma sökriktningen i NIHT algoritmen, ger de viktig information för att välja en lämplig stegstorlek. De tre tidigare

nämnda projicerade riktningar motiverar följande stegstorlekar,

$$\mu_k^u := \frac{||\mathbf{P}_\mathbf{U}^k \mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k))||_F^2}{||\mathcal{A}(\mathbf{P}_\mathbf{U}^k \mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k)))||_2^2},$$
$$\mu_k^v := \frac{||\mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k))\mathbf{P}_\mathbf{V}^k||_F^2}{||\mathcal{A}(\mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k))\mathbf{P}_\mathbf{V}^k)||_2^2},$$
$$\mu_k^{uv} := \frac{||\mathbf{P}_\mathbf{U}^k \mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k))\mathbf{P}_\mathbf{V}^k||_F^2}{||\mathcal{A}(\mathbf{P}_\mathbf{U}^k \mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k))\mathbf{P}_\mathbf{V}^k)||_2^2},$$

där $\mu_k^u$ har observerats ge bäst empiriska resultat [13]. Alltså använder NIHT samma iterativa uppdatering av $\mathbf{X}^{k+1}$ som IHT i ekvation (14) men med stegstorlek $\mu = \mu_k^u$ som uppdateras iterativt.

Nedan följer en beskrivning av NIHT algoritmen.

- Initialisera $\mathbf{X}^0 = \mathcal{H}_r(\mathcal{A}^*(\bar{b}))$ samt $\mathbf{U}_0$ som de första $r$ singulära vektorerna av $\mathbf{X}^0$ som uppdateras under iterationerna. Sätt $k = 0$.
- Beräkna för varje iteration $k \geq 0$ projektionen $\mathbf{P}_\mathbf{U}^k = \mathbf{U}_k \mathbf{U}_k^*$ samt stegstorleken $\mu_k^u$.
- Sätt $\mathbf{X}^{k+1} = \mathcal{H}_r(\mathbf{X}^k + \mu_k^u \mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k)))$ samt $\mathbf{U}_{k+1}$ som de första $r$ singulära vektorer av $\mathbf{X}^{k+1}$. Sätt $k = k + 1$.
- Fortsätt tills stoppvillkor eller eventuella maximala antal iterationer har uppnåtts. Residualen beräknas i [13] som $\left\|\bar{b} - \mathcal{A}(\mathbf{X}^k)\right\|$, där i denna rapport en normeringsterm har lagts till och beräknas som
$$\epsilon_{\text{NIHT}} := \frac{\left\|\bar{b} - \mathcal{A}(\mathbf{X}^k)\right\|_2}{\left\|\bar{b}\right\|_2} \tag{15}$$
där stoppvillkoret är $\epsilon_{\text{NIHT}} \leq \epsilon_{\text{tol}}$ för en förutbestämd toleransnivå $\epsilon_{\text{tol}}$.
- Returnera $\mathbf{X}^k$ som en lågrangkomplettering av $\mathbf{M}$

Nedan ses fullständig pseudokod för NIHT algoritmen så som den är presenterad i artikel [13].

**Algoritm 3** Pseudokod för NIHT algoritmen från [13]

**Input:** Rank $r$, operator $\mathcal{A}(\cdot)$ and observed data $\bar{b} = \mathcal{A}(\mathbf{M})$
**Output:** $\mathbf{M}_{\text{comp}}$.
1: Set $k = 0$. Initialize $\mathbf{X}^0 = \mathcal{H}_r(\mathcal{A}^*(\bar{b}))$.
2: Compute $\mathbf{U}_0$, the top $r$ left singular vectors of $\mathbf{X}^0$.
3: **while** $\frac{||\bar{b} - \mathcal{A}(\mathbf{X}^k)||_2}{||\bar{b}||_2} > \text{tol}$ **do**
4: Set the projection operator $\mathbf{P}_\mathbf{U}^k = \mathbf{U}_k \mathbf{U}_k^*$.
5: Compute step size:
$$\mu_k^u = \frac{||\mathbf{P}_\mathbf{U}^k \mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k))||_F^2}{||\mathcal{A}(\mathbf{P}_\mathbf{U}^k \mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k)))||_2^2}$$
6: Update:
$$\mathbf{X}^{k+1} = \mathcal{H}_r(\mathbf{X}^k + \mu_k^u \mathcal{A}^*(\bar{b} - \mathcal{A}(\mathbf{X}^k)))$$
7: Let $\mathbf{U}_{k+1}$ be the top $r$ left singular vectors of $\mathbf{X}^{k+1}$.
8: Set $k = k + 1$.
9: **end while**
10: **return** $\mathbf{M}_{\text{comp}} = \mathbf{X}^k$ as the low-rank completion of $\mathbf{M}$.

# 3 Metod

Samtliga algoritmer implementeras i programspråket Python (version 3.13.0) och koden återfinns i appendix A. Implementationerna följer i huvudsak pseudokoden presenterad i kapitel 2 med

undantag för några ad hoc-lösningar som specificeras där det är aktuellt. De paket som används är `NumPy` och `SciPy` då det är erkänt att dessa utför exempelvis singulärvärdesuppdelningar på ett effektivt sätt.

## 3.1 Parameterval

Undersökningen av vilka parametervärden som fungerar bra för vardera algoritm är explorativ till sin natur. Arbetet har i mångt och mycket skett parallellt med implementationen av algoritmerna och utformningen av de andra testerna. Utgångspunkten har varit de i artiklarna [2, 8, 13] rekommenderade värdena men med målet att bekräfta dessa med egna numeriska experiment. Ytterligare läggs stor vikt vid att förklara varför och under vilka förutsättningar dessa faktiskt fungerar samtidigt som alternativa värden utvärderas och i vissa fall förkastas.

## 3.2 Syntetisk data

De syntetiska datamängderna består av kända matriser som garanterat är lågrang. Det är möjligt att skapa sådana genom att slumpmässigt generera två matriser $\mathbf{A}_{m\times r}$ och $\mathbf{B}_{r\times n}$. Då gäller det att produkten $\mathbf{M}_{\text{sann}} = \mathbf{AB}$ är en $m \times n$ matris av högst rang $r$ och med väldigt stor sannolikhet exakt rang $r$. En slumpmässig mask kan sedan appliceras för att få en matris $\mathbf{M}$ med önskad gleshet, där det med gleshet menas andelen okända element att komplettera. För implementation av hur matriserna genereras se appendix A.1.

Fördelen med detta tillvägagångssätt är att en jämförelse med ett sant värde kan göras. Låt $\mathbf{M}_{\text{komp}}$ vara den kompletterade matrisen. De residualer $\epsilon_{\text{SVT}}$, $\epsilon_{\text{IRLS}}$ och $\epsilon_{\text{NIHT}}$ (se (10), (12)och (15)) som beräknas i varje iteration skiljer sig åt mellan algoritmerna vilket inte tillåter en meningsfull jämförelse. Exempelvis ändras inte de redan kända värdena i IRLS-GP, vilket med felmåttet för SVT hade blivit 0 vid varje iteration. För att kvantifiera felet i kompletteringen på ett sätt som tillåter en jämförelse används istället $\epsilon$ definierat i (16). Skillnaden mellan den kompletterade matrisen $\mathbf{M}_{\text{komp}}$ och den sanna matrisen $\mathbf{M}_{\text{sann}}$ beräknas med Frobeniusnormen och normeras mot $\mathbf{M}_{\text{sann}}$ för att svaret skall vara oberoende av storleken på de ingående elementen.

$$\epsilon := \frac{\|\mathbf{M}_{\text{komp}} - \mathbf{M}_{\text{sann}}\|_F}{\|\mathbf{M}_{\text{sann}}\|_F}. \tag{16}$$

Testerna är utformade enligt följande. Toleransnivån $\epsilon_{\text{alg}}$, vilken är unik för varje algoritm och definierad i (10), (12) respektive (15), väljs för samtliga tester till $\epsilon_{\text{alg}} = 1 \times 10^{-4}$. För varje given kombination av matrisstorlek, rang och andel okända värden genereras slumpmässigt fem olika matriser. Matriserna kompletteras med samtliga algoritmer, där samma matriser används för varje algoritm för att säkerställa en rättvis jämförelse. Felet för de kompletterade matriserna beräknas enligt ovan, och antalet iterationer innan konvergens samt körningstid noteras. Resultaten från de fem matriserna används därefter för att beräkna ett medelvärde för varje algoritm för varje given storlek, rang och gleshet. Se appendix A.5 för testerna implementerade i Python.

## 3.3 Verklig data

Förutom tester på känd syntetiserad data testas även algoritmerna på två verkliga datamängder. Denna data har inte känd rang och det är inte heller möjligt att jämföra kompletteringen med en sann matris, det vill säga att $\epsilon$ inte kan beräknas. Istället delas alla kända element upp i en kompletteringsmängd $\Omega_{\text{komp}}$ och en testmängd $\Omega_{\text{test}}$, där 90% tillhör kompletteringsmängden och 10% tillhör testmängden. Algoritmerna får elementen tillhörande $\Omega_{\text{komp}}$ som indata för att komplettera och där alltså elementen som tillhör testmängden är att betrakta som saknade. Efter kompletteringen jämförs elementen på positioner tillhörande testmängden med motsvarande värden i densamma för att utvärdera algoritmens prestation.

Vi inför därför $\epsilon_{\text{test}}$, vilket är mycket likt $\epsilon$ från (16) men med skillnaden att jämförelsen endast sker på testmängden.

$$\epsilon_{\text{test}} := \frac{\|\mathcal{P}_{\Omega_{\text{test}}}(\mathbf{M}_{\text{komp}} - \mathbf{M}_{\text{orginal}})\|_F}{\|\mathcal{P}_{\Omega_{\text{test}}}(\mathbf{M}_{\text{orginal}})\|_F}. \tag{17}$$

I nedanstående avsnitt diskuteras hur datamängderna är konstruerade, samt en etisk avvägning.

### 3.3.1 Netflix Prize

Datamängden som användes för Netflix Prize [10], finns att hitta på Kaggle [11] där den sammanställts i samarbete med Chris Crawford. Datamängden innehåller 17770 filmer och 480 189 användares recensioner med heltal mellan 1 och 5 med en gleshet på 0,85. Av praktiska skäl görs en avgränsning till 1000 filmer, och hänsyn tas bara till de 1000 mest aktiva användarna på dessa filmer. Detta resulterar i en kompletteringsmängd på $1000 \times 1000$ element varav 86,5 % är okända.

### 3.3.2 Jester Collaborative Filtering Dataset

Datamängden för Jester Collaborative Filtering Dataset härstammar från UC Berkeleys Jester-system [6] är sammanställt av Aakaash Jois på Kaggle [7] under en CC BY-NC-SA 4.0 licens. Datamängden består av kontinuerliga betyg mellan $-10$ och $+10$ för 100 skämt, satta av totalt 73 421 användare. Datamängden avgränsas till de 3000 mest aktiva användarna och ger då en kompletteringsmängd innehållande $100 \times 3000$ element där 10,3 % av dessa är okända.

### 3.3.3 Etiskt relevanta aspekter

Netflix Prize data var inte lika anonym som Netflix ursprungligen hävdade. Efter att Narayan och Shmatikov 2007 påvisat [9] att det var möjligt att, med hjälp av Netflix Prize data samt allmänt tillgänglig data från IMDb, ta reda på känslig privat information från användare, ställdes framtida Netflix tävlingar in. Vidare påpekar de att efter sådan identifiering gjorts kan den framtagna informationen användas för att bryta framtida anonymiserade datamängder. Således görs valet att inte publicera vidare den delmängd som använts och i största möjliga mån inte använda data som inte är nödvändigt för projektet.

All data är inhämtad med användares samtycke och används enligt gällande licenser. Vidare kommer en kopia av uppsatsen skickas till Netflix respektive Ken Goldberg [6] i enlighet med önskan på deras hemsida.

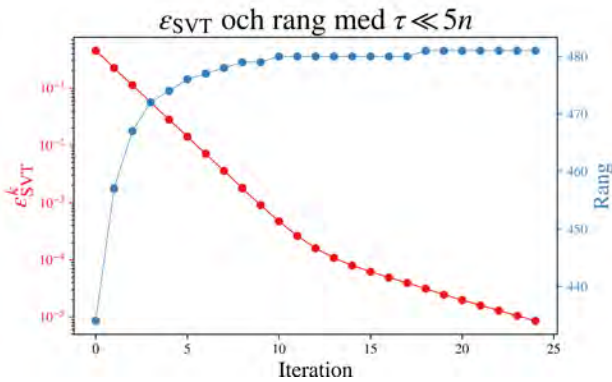
# 4 Resultat

Följande resultatdel är utformad i kronologisk ordning där delresultat, såsom parameterval, som är nödvändiga för senare tester presenteras först. Eftersom resultaten och i synnerhet insikterna som gjordes till följd av dessa är så pass viktiga för senare delar förs en diskussion om dessa löpande, medan en mer övergripande diskussion om metoderna sparas till diskussionskapitlet.

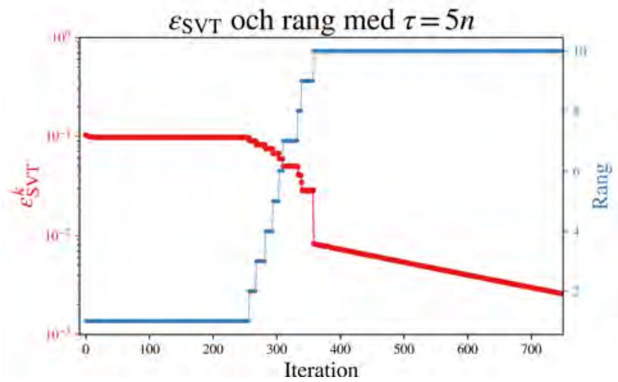
## 4.1 Parameterval

### 4.1.1 SVT:s $\tau$ och $\delta$

Valet av parametrar i SVT visar sig vara av yttersta vikt och dåliga val av $\tau$ och $\delta$ visar sig kunna ge otroligt dåliga resultat. Till att börja med ger ett för litet värde på $\tau$ en väldigt snabb konvergens då residualer snabbt blir små, vilket även kan ses och förklaras i (8). Eftersom SVT:s residual beräknas med Frobeniusnormen är detta tämligen självklart då ett litet $\tau$ låter termen med Frobeniusnormen dominera och på så sätt effektivt minimerar med avseende på dessa utan att minimera rangen. I figur 4 visas till vänster hur beteendet påverkas när $\tau$ väljs för litet. Till höger i figur 4 ses istället ett lämpligt val av $\tau = 5n$. Detta resulterar i en långsammare konvergens men förhindrar samtidigt att rangen når full rang.



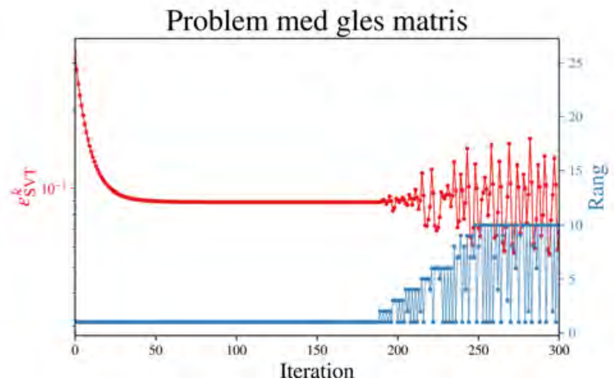

(a) Ett litet tau ger snabb konvergens och rangen når snabbt till full rang.




(b) Parametervalet $\tau = 5n$ ger långsam konvergens och korrekt rang.

Figur 4: En $500 \times 500$ matris med rang 10 och 20% saknade värden har kompletterats med SVT för olika värden på $\tau$. Notera skillnaden i antal iterationer.

Som diskuterades i 2.5 är konvergens endast garanterad för $\delta \in (0, 2)$ vilket snabbt kan tänkas bli problematiskt när andelen kända element minskar. Numeriska experiment visar att det i många fall konvergerar även för högre $\delta$ men även detta är starkt kopplat till andelen kända element. Standard-parametervalet $\delta = 1{,}2\frac{mn}{|\Omega|}$ är för de flesta matristyper ett bra val. Med låg andel kända element är det $\delta$ som standard-parametervalet ger för stort för algoritmen vilket ses i figur 5. Ett konsekvent val av $\delta < 2$ hade förvisso garanterat konvergens men potentiellt saktat ner kovergenshastigheten till en nivå där residualerna inte konvergerar innan eventuella iterationsbegränsningar uppnåtts.

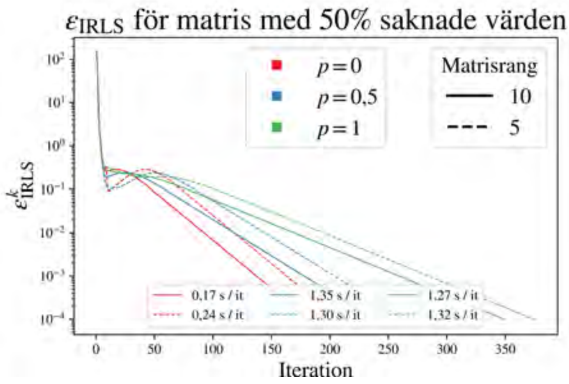



Figur 5: En $500 \times 500$ matris med rang 10 och 95% saknade värden, vilket ger $\delta = 24$, har kompletterats. Notera det oscillerande beteendet efter ungefär 200 iterationer.

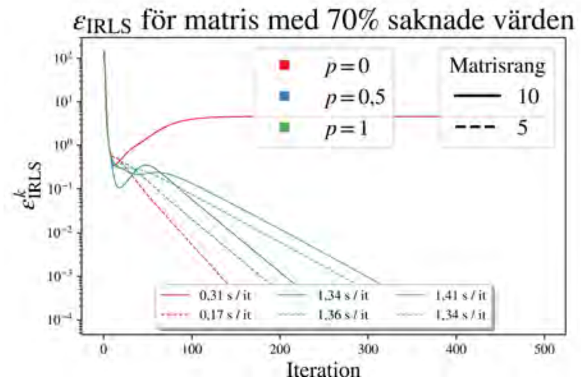
#### 4.1.2 IRLS-GP:s $p$ och $\gamma_c$

Relevanta parametrar för IRLS-GP algoritmen som måste väljas är $p$ och $\gamma_c$. $p$ används för att beräkna viktmatrisen $\mathbf{W}_p$ och stegstorleken $s$, och $\gamma_c$ används vid beräkningen av $\gamma^0 = \gamma_c||\mathbf{M}||_2$.

I enlighet med vad som nämns i artikel [8], ger $p = 0$ snabb men för vissa matriser dålig konvergens, medan $p > 0$ är långsammare men mer konsekvent ger konvergens. Detta visas tydligt i figur 6 där IRLS-GP inte konvergerar för matriser med en gleshet på 0,7 och rang $r = 10$ då $p = 0$.



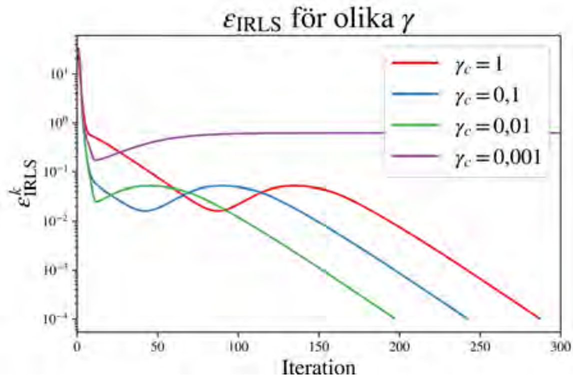

(a) En $500 \times 500$ matris med rang $\in \{5, 10\}$ och 50% saknade värden.




(b) En $500 \times 500$ matris med rang $\in \{5, 10\}$ och 70% saknade värden.

Figur 6: En $500 \times 500$ matris med rang $\in \{5, 10\}$ med 50% (vänster) respektive 70% (höger) saknade värden har kompletterats med IRLS-GP för olika värden på $p$, och total tid har noterats. I testerna har $\gamma = 0{,}01$ använts.

Ett för litet val av $\gamma_c$ visar sig kunna orsaka problem. I figur 7 ses det för en $100 \times 100$ matris hur valet $\gamma_c = 1 \times 10^{-3}$ kan leda till att algoritmen inte konvergerar. Det är värt att notera att detta problem inte uppkommer varje gång utan enbart då och då. Vidare ses det i figur 7 att ett mindre $\gamma_c$ kräver färre iterationer då konvergens faktiskt uppnås. Valet av $\gamma_c = 1 \times 10^{-2}$ har ensamt inte orsakat konvergensproblem i några av de numeriska testerna som har gjorts och anses därför i enlighet med [8] vara ett bra val.

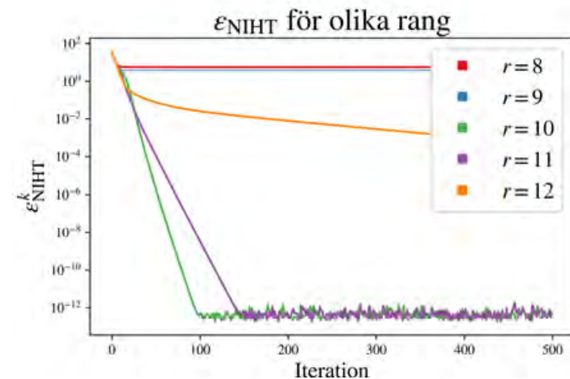



Figur 7: En $100 \times 100$ matris med rang 10 och 50% saknade värden har kompletterats med IRLS-GP

### 4.1.3 NIHT:s $r$

Den mest relevanta parametern som NIHT använder för att komplettera lågrangmatriser är dess rang $r$. I figur 8 ses att om $r$ överensstämmer med den sanna rangen fås snabb konvergens men redan vid små skillnader gentemot den sanna rangen fås betydligt långsammare och sämre konvergens. Det ses även att det ger ett sämre resultat att underskatta rangen än att överskatta den.

Testerna som gjordes i figur 8 kunde också skilja sig en aning till utseendet från gång till gång vilket ytterligare pekar på att det antingen finns olika lösningar eller att det finns andra egenskaper hos matriserna, som vi inte undersökt, som påverkar stabiliteten för algoritmen.




Figur 8: En $500 \times 500$ matris med rang 10 och 50% saknade värden har kompletterats med NIHT med olika $r$ som parameter.

## 4.2 Prestation på syntetisk data

Nedan presenteras resultaten för de tre undersökta algoritmerna i tre separata tabeller. Varje tabell innehåller uppmätta värden för total beräkningstid, antalet körda iterationer, algoritmspecifika residualer samt totalt fel i den utförda kompletteringen.

#### 4.2.1 SVT

I tabell 1 redovisas resultat för SVT som tidsåtgång, antal iterationer, den algoritmspecifika residualen $\epsilon_{\text{SVT}}$ (10) och det totala felet $\epsilon$ (16). SVT uppnår konsekvent ett lågt $\epsilon$ ($< 0{,}1$) i samtliga test och $\epsilon_{\text{SVT}}$ verkar korrelera starkt till $\epsilon$.

Det ses att i samtliga fall där algoritmen har konvergerat, det vill säga uppnått stoppvillkoret $\epsilon_{\text{SVT}} < 1 \times 10^{-4}$, att snittkvadratfelet ($\epsilon$) för varje element är av storleksordning $10^{-4}$. Detta sker i synnerhet för den lägsta glesheten 0,5 där sex av åtta test uppnår stoppvillkoret. För den högsta glesheten 0,95 har ingen av testerna uppnått stoppvillkoret. Vidare ses det att rangen har stor inverkan på prestationen och resultaten generellt är sämre för matriserna med högre rang, allt annat lika.

Tidsåtgången per iteration växer relativt både matrisstorlek och rang. Notera även att det finns en aningen mer subtil minskning av tid per iteration när glesheten ökar.

Tabell 1: Numeriska resultat för SVT testad på genererade lågrangmatriser med varierande storlek $n \times n$, rang $r$ och gleshet. Stoppvillkor är $\epsilon_{\text{SVT}} \leq 10^{-4}$ eller #iter = 3000. Relevanta parameterval som använts för testerna är $\tau = 5n$ och $\delta = 1{,}2n^2/|\Omega|$.

| Okänd matris **M** | | | Numeriska resultat | | | | |
|---|---|---|---|---|---|---|---|
| storlek ($n \times n$) | rang ($r$) | gleshet | tid (s) | #iter | tid/iter (ms) | $\epsilon_{\text{SVT}}$ | $\epsilon$ |
| 500 × 500 | 5 | 0,50 | 20,76 | 1640 | 12,7 | $9{,}98 \times 10^{-5}$ | $1{,}13 \times 10^{-4}$ |
| | 5 | 0,70 | 36,42 | 3000 | 12,1 | $2{,}05 \times 10^{-4}$ | $2{,}57 \times 10^{-4}$ |
| | 5 | 0,95 | 35,32 | 3000 | 11,8 | $3{,}96 \times 10^{-2}$ | $6{,}39 \times 10^{-2}$ |
| | 10 | 0,50 | 42,76 | 2960 | 14,4 | $1{,}11 \times 10^{-4}$ | $1{,}34 \times 10^{-4}$ |
| | 10 | 0,70 | 43,19 | 3000 | 14,4 | $1{,}72 \times 10^{-3}$ | $2{,}29 \times 10^{-3}$ |
| | 10 | 0,95 | 38,82 | 3000 | 12,9 | $5{,}16 \times 10^{-2}$ | $8{,}43 \times 10^{-2}$ |
| | 50 | 0,50 | 147,73 | 3000 | 49,2 | $5{,}78 \times 10^{-3}$ | $8{,}78 \times 10^{-3}$ |
| | 50 | 0,70 | 123,89 | 3000 | 41,3 | $1{,}25 \times 10^{-2}$ | $2{,}22 \times 10^{-2}$ |
| | 50 | 0,95 | 33,45 | 3000 | 11,2 | $4{,}46 \times 10^{-2}$ | $4{,}80 \times 10^{-2}$ |
| 1000 × 1000 | 5 | 0,50 | 66,52 | 1469 | 45,3 | $9{,}98 \times 10^{-5}$ | $1{,}07 \times 10^{-4}$ |
| | 5 | 0,70 | 125,58 | 2867 | 43,8 | $1{,}07 \times 10^{-4}$ | $1{,}23 \times 10^{-4}$ |
| | 5 | 0,95 | 118,81 | 3000 | 39,6 | $3{,}54 \times 10^{-2}$ | $4{,}55 \times 10^{-2}$ |
| | 10 | 0,50 | 140,67 | 2526 | 55,7 | $9{,}99 \times 10^{-5}$ | $1{,}11 \times 10^{-4}$ |
| | 10 | 0,70 | 173,43 | 3000 | 57,8 | $1{,}07 \times 10^{-3}$ | $1{,}27 \times 10^{-3}$ |
| | 10 | 0,95 | 145,84 | 3000 | 48,6 | $7{,}77 \times 10^{-2}$ | $8{,}72 \times 10^{-2}$ |
| | 50 | 0,50 | 380,48 | 3000 | 126,8 | $3{,}95 \times 10^{-3}$ | $4{,}97 \times 10^{-3}$ |
| | 50 | 0,70 | 336,08 | 3000 | 112,0 | $1{,}11 \times 10^{-2}$ | $1{,}54 \times 10^{-2}$ |
| | 50 | 0,95 | 144,42 | 3000 | 48,1 | $4{,}53 \times 10^{-2}$ | $4{,}71 \times 10^{-2}$ |
| 2000 × 2000 | 5 | 0,50 | 332,46 | 1364 | 243,7 | $9{,}98 \times 10^{-5}$ | $1{,}04 \times 10^{-4}$ |
| | 5 | 0,70 | 619,84 | 2622 | 236,4 | $9{,}99 \times 10^{-5}$ | $1{,}08 \times 10^{-4}$ |
| | 5 | 0,95 | 572,89 | 3000 | 191,0 | $2{,}97 \times 10^{-2}$ | $3{,}38 \times 10^{-2}$ |
| | 10 | 0,50 | 644,89 | 2321 | 277,9 | $9{,}99 \times 10^{-5}$ | $1{,}06 \times 10^{-4}$ |
| | 10 | 0,70 | 804,47 | 3000 | 268,2 | $5{,}54 \times 10^{-4}$ | $6{,}11 \times 10^{-4}$ |
| | 10 | 0,95 | 608,13 | 3000 | 202,7 | $9{,}50 \times 10^{-2}$ | $9{,}67 \times 10^{-2}$ |

### 4.2.2 IRLS-GP

I tabell 2 redovisas resultat för IRLS-GP som tidsåtgång, antal iterationer, den algoritmspecifika residualen $\epsilon_{\text{IRLS}}$ (12) och det totala felet $\epsilon$ (16) på samma sätt som för SVT. IRLS-GP presterar bäst på matriserna med rang 5 och uppnår i alla fall förutom $2000 \times 2000$ matrisen med gleshet 0,95 stoppvillkoret $\epsilon_{\text{IRLS}} < 1 \times 10^{-4}$. Förutom matriserna med rang 5 har även de med rang 10 kompletterats väl för 0,5 gleshet. Det ses även att $\epsilon_{\text{IRLS}}$ är svagt korrelerad till motsvarande $\epsilon$, där snittkvadratfelet är i storleksordningen $10^{-3}$. Det noteras dock att även för de test som inte konvergerar så är det totala felet $\epsilon$ tämligen lågt och indikerar i många fall ett snittkvadratfel på $\epsilon < 0{,}1$.

Till skillnad från SVT verkar tidsåtgången per iteration nästan enbart bero på matrisstorleken.

Tabell 2: Numeriska resultat för IRLS-GP testad på genererade lågrangmatriser med varierande storlek $n \times n$, rang $r$ och gleshet. Stoppvillkor är $\epsilon_{\text{IRLS}} \leq 10^{-4}$ eller #iter = 3000. Relevanta parameterval som använts för testerna är $p = 0$ och $\gamma_c = 0{,}01$.

| Okänd matris **M** | | | Numeriska resultat | | | | |
|---|---|---|---|---|---|---|---|
| storlek ($n \times n$) | rang ($r$) | gleshet | tid (s) | # iter | tid/iter (ms) | $\epsilon_{\text{IRLS}}$ | $\epsilon$ |
| $500 \times 500$ | 5 | 0,5 | 3,81 | 89 | 42,8 | $9{,}74 \times 10^{-5}$ | $1{,}38 \times 10^{-3}$ |
| | 5 | 0,7 | 3,95 | 90 | 43,9 | $9{,}63 \times 10^{-5}$ | $1{,}05 \times 10^{-3}$ |
| | 5 | 0,95 | 129,92 | 3000 | 43,3 | $4{,}12 \times 10^{-4}$ | $6{,}18 \times 10^{-2}$ |
| | 10 | 0,5 | 5,43 | 111 | 48,9 | $9{,}76 \times 10^{-5}$ | $1{,}38 \times 10^{-3}$ |
| | 10 | 0,7 | 121,84 | 3000 | 40,6 | $3{,}82 \times 10^{-2}$ | $5{,}19 \times 10^{-2}$ |
| | 10 | 0,95 | 119,62 | 3000 | 39,9 | $1{,}35 \times 10^{-2}$ | $2{,}17 \times 10^{-1}$ |
| | 50 | 0,5 | 115,28 | 3000 | 38,4 | $9{,}37 \times 10^{-3}$ | $3{,}29 \times 10^{-2}$ |
| | 50 | 0,7 | 115,13 | 3000 | 38,4 | $1{,}29 \times 10^{-2}$ | $3{,}85 \times 10^{-2}$ |
| | 50 | 0,95 | 109,00 | 3000 | 36,3 | $6{,}39 \times 10^{-3}$ | $1{,}32 \times 10^{-1}$ |
| $1000 \times 1000$ | 5 | 0,5 | 15,57 | 89 | 174,9 | $9{,}73 \times 10^{-5}$ | $1{,}38 \times 10^{-3}$ |
| | 5 | 0,7 | 16,05 | 88 | 182,4 | $9{,}67 \times 10^{-5}$ | $1{,}06 \times 10^{-3}$ |
| | 5 | 0,95 | 67,62 | 407 | 166,1 | $9{,}96 \times 10^{-5}$ | $9{,}57 \times 10^{-4}$ |
| | 10 | 0,5 | 20,58 | 110 | 187,1 | $9{,}86 \times 10^{-5}$ | $1{,}39 \times 10^{-3}$ |
| | 10 | 0,7 | 588,32 | 3000 | 196,1 | $4{,}03 \times 10^{-2}$ | $5{,}65 \times 10^{-2}$ |
| | 10 | 0,95 | 531,84 | 3000 | 177,3 | $2{,}81 \times 10^{-2}$ | $7{,}48 \times 10^{-2}$ |
| | 50 | 0,5 | 514,19 | 3000 | 171,4 | $8{,}49 \times 10^{-3}$ | $3{,}29 \times 10^{-2}$ |
| | 50 | 0,7 | 514,70 | 3000 | 171,6 | $1{,}15 \times 10^{-2}$ | $3{,}83 \times 10^{-2}$ |
| | 50 | 0,95 | 531,38 | 3000 | 177,1 | $1{,}10 \times 10^{-2}$ | $5{,}30 \times 10^{-2}$ |
| $2000 \times 2000$ | 5 | 0,5 | 89,65 | 88 | 1018,8 | $9{,}84 \times 10^{-5}$ | $1{,}39 \times 10^{-3}$ |
| | 5 | 0,7 | 88,14 | 87 | 1013,1 | $9{,}90 \times 10^{-5}$ | $1{,}08 \times 10^{-3}$ |
| | 5 | 0,95 | 2848,11 | 3000 | 949,4 | $5{,}54 \times 10^{-2}$ | $5{,}96 \times 10^{-2}$ |
| | 10 | 0,5 | 117,65 | 109 | 1079,4 | $9{,}84 \times 10^{-5}$ | $1{,}39 \times 10^{-3}$ |
| | 10 | 0,7 | 2936,09 | 3000 | 978,7 | $4{,}11 \times 10^{-2}$ | $6{,}45 \times 10^{-2}$ |
| | 10 | 0,95 | 2943,56 | 3000 | 981,2 | $3{,}51 \times 10^{-2}$ | $9{,}19 \times 10^{-2}$ |

### 4.2.3 NIHT

I tabell 3 redovisas resultat för NIHT som tidsåtgång, antal iterationer, den algoritmspecifika residualen $\epsilon_{\text{NIHT}}$ (15) och det totala felet $\epsilon$ (16) som tidigare. NIHT uppvisar konvergens för en stor variation av matriser, inklusive fall med den högsta rangen 50. Den uppvisar även bra resultat med ett snittkvadratfel av storleksordning $10^{-4}$ för tre av åtta matriser med gleshet 0,95. Ibland förekommer även konvergens till en felaktig lösning där $\epsilon_{\text{NIHT}} \leq 10^{-4}$ men $\epsilon$ är stort, exempelvis vid storlek $500 \times 500$, rang 50 och gleshet 0,95. Av de matriser som verkar ha konvergerat korrekt ses ett visst samband med att $\epsilon$ minskar när rangen och glesheten minskar.

Beräkningstiderna är generellt korta i förhållande till de andra algoritmerna. Förutom matrisstorleken är tidsåtgången per iteration starkt beroende av rangen där lägre rang ger en snabbare beräkningstid. Generellt ses ingen tydlig koppling mellan tidsåtgång per iteration och gleshet förutom i de fall där algoritmen når stoppvillkoret $\epsilon_{\text{NIHT}} < 1 \times 10^{-4}$ trots ett högt $\epsilon$.

Tabell 3: Numeriska resultat för NIHT testad på genererade lågrangmatriser med varierande storlek $n \times n$, rang $r$ och gleshet. Stoppvillkor är $\epsilon_{\text{NIHT}} \leq 10^{-4}$ eller #iter = 3000. Relevanta parameterval som använts för testerna är att $r$ antagits varit känd.

| Okänd matris $\mathbf{M}$ | | | Numeriska resultat | | | | |
|---|---|---|---|---|---|---|---|
| storlek ($n \times n$) | rang ($r$) | gleshet | tid (s) | # iter | tid/iter (ms) | $\epsilon_{\text{NIHT}}$ | $\epsilon$ |
| $500 \times 500$ | 5 | 0,5 | 0,37 | 18 | 20,6 | $8{,}58 \times 10^{-5}$ | $1{,}00 \times 10^{-4}$ |
| | 5 | 0,7 | 0,89 | 46 | 19,3 | $9{,}32 \times 10^{-5}$ | $1{,}19 \times 10^{-4}$ |
| | 5 | 0,95 | 33,47 | 3000 | 11,2 | $1{,}49 \times 10^{-2}$ | $7{,}38 \times 10^{-2}$ |
| | 10 | 0,5 | 0,62 | 25 | 24,8 | $7{,}54 \times 10^{-5}$ | $9{,}39 \times 10^{-5}$ |
| | 10 | 0,7 | 2,31 | 102 | 22,6 | $9{,}20 \times 10^{-5}$ | $1{,}30 \times 10^{-4}$ |
| | 10 | 0,95 | 38,14 | 3000 | 12,7 | $3{,}14 \times 10^{-2}$ | $3{,}90 \times 10^{-1}$ |
| | 50 | 0,5 | 184,29 | 2808 | 65,6 | $1{,}70 \times 10^{-3}$ | $1{,}51 \times 10^{-2}$ |
| | 50 | 0,7 | 195,60 | 3000 | 65,2 | $1{,}37 \times 10^{-2}$ | $1{,}57 \times 10^{-1}$ |
| | 50 | 0,95 | 7,66 | 120 | 63,8 | $9{,}79 \times 10^{-5}$ | $9{,}08 \times 10^{-1}$ |
| $1\,000 \times 1\,000$ | 5 | 0,5 | 1,13 | 15 | 75,3 | $7{,}68 \times 10^{-5}$ | $8{,}46 \times 10^{-5}$ |
| | 5 | 0,7 | 2,20 | 34 | 64,7 | $9{,}08 \times 10^{-5}$ | $1{,}06 \times 10^{-4}$ |
| | 5 | 0,95 | 43,51 | 1004 | 43,3 | $9{,}93 \times 10^{-5}$ | $1{,}97 \times 10^{-4}$ |
| | 10 | 0,5 | 1,41 | 17 | 82,9 | $8{,}35 \times 10^{-5}$ | $9{,}60 \times 10^{-5}$ |
| | 10 | 0,7 | 5,13 | 53 | 96,8 | $8{,}79 \times 10^{-5}$ | $1{,}07 \times 10^{-4}$ |
| | 10 | 0,95 | 169,72 | 3000 | 56,6 | $1{,}67 \times 10^{-2}$ | $8{,}35 \times 10^{-2}$ |
| | 50 | 0,5 | 56,45 | 274 | 206,0 | $9{,}10 \times 10^{-5}$ | $1{,}34 \times 10^{-4}$ |
| | 50 | 0,7 | 548,57 | 2621 | 209,3 | $1{,}24 \times 10^{-3}$ | $1{,}03 \times 10^{-2}$ |
| | 50 | 0,95 | 136,10 | 679 | 200,4 | $9{,}95 \times 10^{-5}$ | $8{,}03 \times 10^{-1}$ |
| $2\,000 \times 2\,000$ | 5 | 0,5 | 4,23 | 14 | 302,1 | $7{,}03 \times 10^{-5}$ | $7{,}44 \times 10^{-5}$ |
| | 5 | 0,7 | 8,21 | 29 | 283,1 | $8{,}75 \times 10^{-5}$ | $9{,}65 \times 10^{-5}$ |
| | 5 | 0,95 | 91,54 | 418 | 219,0 | $9{,}96 \times 10^{-5}$ | $1{,}44 \times 10^{-4}$ |
| | 10 | 0,5 | 5,53 | 15 | 368,7 | $7{,}38 \times 10^{-5}$ | $8{,}06 \times 10^{-5}$ |
| | 10 | 0,7 | 10,87 | 33 | 329,4 | $9{,}31 \times 10^{-5}$ | $1{,}07 \times 10^{-4}$ |
| | 10 | 0,95 | 484,82 | 1853 | 261,6 | $9{,}88 \times 10^{-5}$ | $1{,}62 \times 10^{-4}$ |

## 4.3 Prestation på verklig data

Nedan presenteras resultaten för de tre undersökta algoritmerna på datamängderna nämnda i kapitel 3.3. Samma parameterval används i följande tester som vid kapitel 4.2 med undantag för att tester med IRLS-GP görs för både $p = 0$ och $p = 0{,}5$. Dessutom har NIHT:s rang uppskattats till $r = 7$ för Netflix-datamängden och $r = 7$ för Jester-datamängden. Denna uppskattning gjordes genom att testa alla möjliga ranger på en delmatris med storlek $100 \times 100$ där det slutgiltiga parametervalet valdes till rangen som gav bäst resultat på testmängden för delmatrisen.

För varje kombination av algoritm och datamängd noteras den totala beräkningstiden, antal

iterationer som körts, den algoritmspecifika residualen $\epsilon_{alg} \in \{\epsilon_{SVT}, \epsilon_{IRLS}, \epsilon_{NIHT}\}$ från (10), (12) och (15) samt testfelet $\epsilon_{test}$ (17) på de kompletterade elementen i testmängden.

För Netflix-datamängden, som har hög gleshet (0,865), presterar NIHT bäst sett till $\epsilon_{test}$ med $\epsilon_{test} = 0{,}268$, vilket är lägre än SVT och IRLS-GP. Samtidigt är det tydligt att IRLS-GP presterar sämre då $p = 0$ jämfört med $p = 0{,}5$. SVT och IRLS-GP med $p = 0{,}5$ konvergerar men har något högre $\epsilon_{test}$.

På Jester-datamängden, som har mycket lägre gleshet (0,103), är resultaten mer varierande. SVT når inte ett lågt testfel $\epsilon_{test}$ men har kort beräkningstid. IRLS-GP med $p = 0$ uppnår konvergens och har lågt $\epsilon_{IRLS}$ men ett högt $\epsilon_{test}$ medan då $p = 0{,}5$ används fås det lägsta testfelet $\epsilon_{test} = 0{,}75$, men på bekostnad av mycket längre körningstid. NIHT har kort beräkningstid och relativt lågt $\epsilon_{NIHT}$ men kräver tidigare rangapproximation.

Tabell 4: Numeriska resultat för algoritmerna SVT, IRLS-GP för $p = 0$ och $p = 0{,}5$ samt NIHT testade på Netflix 3.3.1 och Jester datamängderna 3.3.2 där stoppvillkor som använts är $\epsilon_{alg} \leq 10^{-3}$ eller # iter = 2000.

| algoritm | datamängd | storlek | gleshet | tid (s) | # iter | $\epsilon_{alg}$ | $\epsilon_{test}$ |
|---|---|---|---|---|---|---|---|
| SVT | Netflix | $1000 \times 1000$ | 0,865 | 501,35 | 1800 | $9{,}99\times10^{-4}$ | $2{,}81\times10^{-1}$ |
| | Jester | $100 \times 3000$ | 0,103 | 50,54 | 2000 | $2{,}28\times10^{-1}$ | $7{,}95\times10^{-1}$ |
| IRLS $p = 0$ | Netflix | $1000 \times 1000$ | 0,865 | 478,14 | 2000 | $2{,}82\times10^{1}$ | $3{,}89\times10^{-1}$ |
| | Jester | $100 \times 3000$ | 0,103 | 848,92 | 1087 | $9{,}95\times10^{-4}$ | $1{,}77\times10^{0}$ |
| IRLS $p = 0{,}5$ | Netflix | $1000 \times 1000$ | 0,865 | 1097,34 | 409 | $9{,}66\times10^{-4}$ | $3{,}64\times10^{-1}$ |
| | Jester | $100 \times 3000$ | 0,103 | 11260,85 | 181 | $9{,}49\times10^{-4}$ | $7{,}50\times10^{-1}$ |
| NIHT | Netflix | $1000 \times 1000$ | 0,865 | 88,75 | 2000 | $2{,}36\times10^{-1}$ | $2{,}68\times10^{-1}$ |
| | Jester | $100 \times 3000$ | 0,103 | 35,93 | 2000 | $6{,}98\times10^{-1}$ | $7{,}65\times10^{-1}$ |

# 5 Diskussion

## 5.1 Allmän diskussion kring resultat på syntetisk data

Nedan utförs en sammanställd diskussion kring tester utförda på syntetisk data i kapitel 4.1 samt 4.2.

För samtliga algoritmer kan vi i tabellerna 1, 2 och 3 observera att tidsåtgången per iteration verkar vara starkt korrelerad till matrisstorleken. Resultaten indikerar empiriskt att tidsåtgången per iteration skalar linjärt mot antalet ingående element i matrisen, det vill säga $\mathcal{O}(n^2)$ vilket ses som rimligt.

### 5.1.1 SVT

Inledningsvis kan vi i tabell 1, där resultaten för SVT presenteras, se att när rangen ökar gör även tiden per iteration det. Detta är rimligt eftersom det mest tidskrävande steget i algoritmen är SVD-uppdelningen. Denna är implementerad på ett sådant sätt där antalet singulära värden som beräknas är kopplat till den för iterationen aktuella rangen. Som vi ser i figur 4b är rangen strikt växande, vilket dessutom bevisas i [2]. Detta leder till mer och mer kostsamma SVD-uppdelningar när rangen ökar snabbare, vilket är önskvärt för matriser med högre rang.

Vi uppmärksammar även en skillnad i tid per iteration för matriser med olika gleshet. En högre gleshet ger en kortare beräkningstid per iteration men är samtidigt mindre trolig att konvergera. Denna skillnad blir extra tydlig för matriserna med högre rang.

Som redan konstaterats i kapitel 4.1.1 konvergerar SVT-algoritmen för fler värden på $\delta$ än de som i [2] angavs som garanterade, det vill säga för en gleshet $< 0{,}4$ då $\delta = 1{,}2\frac{mn}{|\Omega|}$. Detta är helt i enlighet med vad författarna av [2] kommit fram till. Det problem som ses uppkomma i figur 5 är ett exempel på då ett för stort $\delta$ förhindrar konvergens. Den figuren är baserad på en matris med gleshet 0,95 och det är således inte förvånande att de glesa matriserna i tabell 1 aldrig presterar lika bra som de med lägre gleshet.

Faktum är dock att våra egna numeriska experiment visat att det i många fall räcker att välja $\delta$ något lägre än vad som är standard för de riktigt glesa matriserna. De mindre glesa matriserna är å andra sidan betydligt känsligare för ett $\delta \in (0, 2)$. Detta är rimligt då antalet giltiga lösningar intuitivt borde minska med ökande antal kända element.

Det presenterades i figur 4b att $\tau = 5n$ presterade bra enligt förväntningarna från [2]. Likväl påvisades att små $\tau$ leder till en överanpassning gentemot $\epsilon_{\text{SVT}}$ och att algoritmen med för stora värden inte hinner konvergera överhuvudtaget. Beroende på tillämpning är det troligt att andra viktningar mellan den nukleära normen och Frobeniusnormen i (8) är mer passande. Det blir en avvägning mellan att minimera rangen och att inte förändra de redan kända värdena.

Med detta sagt uppkommer frågan huruvida det är möjligt att välja dessa parametrar dynamiskt för att förbättra algoritmens prestation. Det vill säga att det exempelvis under de första iterationerna tas längre steg för att sedan minska steglängden $\delta$ i slutet av algoritmen och motsvarande ändring av $\tau$. Tester har gjorts på sådana modifieringar men ingen tydlig förbättring har konstaterats.

### 5.1.2 IRLS-GP

I tabell 2 visar IRLS-GP tydliga brister vid komplettering av matriser som är både glesa och har hög rang. Detta visades redan i figur 6 där parametervalet $p = 0$ orsakade problem om rangen var 10 eller större samtidigt som glesheten var över 0,70. Inga av dessa kombinationer konvergerade heller i testerna.

Denna brist av IRLS-GP då $p = 0$ skulle kunna vara en konsekvens av att teoretisk konvergens enbart är garanterat då $0 < p \leq 1$ [8]. Dessutom är Schatten-p funktionen (11), som algoritmen approximativt försöker minimera, icke-konvex och ej deriverbar då $p = 0$. Därmed är det möjligt att algoritmens gradientnedstigning fastnar vid ett lokalt minima och aldrig når den optimala lösningen. Vi ser i figur 6 att detta hade kunnat undvikas med ett annat val av $p$, men med en stor bekostnad på beräkningstider.

Den stora skillnaden i beräkningstid för IRLS-GP då $p \in (0, 1]$ jämfört med då $p = 0$ kan förklaras med hur viktmatrisen $\mathbf{W}_p = (\mathbf{X}^*\mathbf{X} + \gamma\mathbf{I})^{p/2-1}$ beräknas varje iteration. För $p = 0$ reduceras exponenten till $-1$ och enbart $\mathbf{W}_0 = (\mathbf{X}^*\mathbf{X} + \gamma\mathbf{I})^{-1}$, vilket är en matrisinvers, behöver beräknas. Detta kan med hjälp av SVD göras ytterligare snabbare och stabilare genom att istället beräkna

$$\mathbf{W}_0^k = \left(\mathbf{V}(\gamma_k(\mathbf{\Sigma}^2 + \gamma_k\mathbf{I}_r)^{-1} - \mathbf{I}_r)\mathbf{V}^* + \mathbf{I}_n\right)\frac{1}{\gamma_k}. \tag{18}$$

När däremot exponenten inte är ett heltal, det vill säga då $p \in (0, 1]$, beräknas istället $\mathbf{W}^k$ som en matrispotens. Denna beräkning gör varje iteration av IRLS-GP med $p \in (0, 1]$ avsevärt dyrare än motsvarande iteration med $p = 0$. Våra mätningar av iterationshastighet (s/it) i figur 6 bekräftar detta och visar på att för varje iteration blir den totala beräkningstiden flera gånger större på grund av den kostsamma beräkningen av $\mathbf{W}_p$.

Vi noterar även en märkbar skillnad i tabell 2 mellan residual $\epsilon_{\text{IRLS}}$ och totalt fel $\epsilon$ för IRLS-GP. Detta är delvis en konsekvens av hur $\epsilon_{\text{IRLS}}$ är vald. En residual likt $\epsilon_{\text{SVT}}$ eller $\epsilon_{\text{NIHT}}$, där de kända elementen i varje iteration jämförs med den kompletterade matrisens motsvarande positioner, är inte möjligt för IRLS-GP då dessa är bevarade. Därmed har $\epsilon_{\text{IRLS}}$ definierats baserat på förändringen mellan två iterationer snarare än på avvikelse från kända värden, vilket gör att den inte exakt speglar noggrannheten på den slutgiltiga kompletterade matrisen i förhållande till den sanna matrisen. Av samma anledning innebär detta att $\epsilon$ kan vara aningen missvisande. Eftersom hälften av elementen per definition är kända i en matris med 0,50 gleshet kommer skillnaden mellan den sanna och kompletterade matrisen vara 0 på hälften av positionerna. Således härrör sig allt fel från de okända elementen som således kan antas vara dubbelt så stort som vad en första anblick kan antyda.

### 5.1.3 NIHT

NIHT demonstrerar de objektivt bästa resultaten i kapitel 4.2 med avseende på antalet tester som konvergerar samt har kortast beräkningstid. Men, som tidigare nämnt, är en begränsning med NIHT att rangen i testerna har antagits varit känd och som observerats i kapitel 4.1.3 är algoritmen väldigt känslig för ett felaktigt val av rang.

Vidare förekommer även konvergens till en felaktig lösning där $\epsilon_{\text{NIHT}}$ är liten medan $\epsilon$ är stor. En intuitiv förklaring av detta kan vara att för hög rang och gleshet är lösningen som NIHT söker ej entydigt bestämd. För en $500 \times 500$ matris med rang 50 och gleshet 0,5 ser vi i tabell 3 att inget av stoppvillkoren är uppnådda. Detta kommer sig troligen av att testresultaten är medelvärdesbildade över 5 olika tester där det är troligt att testerna i något eller några fall konvergerat men inte i andra. En annan intressant aspekt är att tidsåtgången inte skiljer sig nämnvärt för olika gleshet förutom de fall där konvergensen är felaktig. Testerna som gjordes i figur 8 kunde också skilja sig en aning till utseendet från gång till gång vilket ytterligare pekar på att det antingen finns olika lösningar eller att det finns andra egenskaper hos matriserna, som vi inte undersökt, som påverkar stabiliteten för algoritmen.

Faktumet att NIHT är snabbare än både IRLS-GP och SVT för många av våra testade matrisvarianter gör att det finns ganska mycket spelrum att utföra en god rangapproximering innan beräkningarna startar. Det finns flera olika förslag på hur detta ska gå till. Flera av dessa bygger på det resultat som presenteras i figur 8. Eftersom residualerna $\epsilon_{\text{NIHT}}$ ofta blir avsevärt mindre om den rang som ges som input överensstämmer med den verkliga rangen innebär detta att algoritmen kan köras flera gånger och resultatet från den med lägst $\epsilon_{\text{NIHT}}$ väljs. Hur rangen testas kan skilja sig åt, både med avseende på hur testerna görs och vilka ranger som testas. Till att börja med ses det i figur 8 att det inte är nödvändigt att låta algoritmen iterera genom fullt så många iterationer som krävs för konvergens utan det för testerna är möjligt att avbryta tidigare. På samma sätt ses det att $\epsilon_{\text{NIHT}}$ avtar när rangen närmar sig den verkliga rangen. Detta öppnar upp möjligheten för att göra exempelvis binär sökning för att hitta den bästa rangen. Att på detta sätt söka genom olika ranger går utanför omfattningen av rapporten men är ett intressant område för vidare studier.

## 5.2 Diskussion kring resultat på verklig data

Resultaten på de verkliga datamängderna i tabell 4 bekräftar flera av de observationer som gjorts tidigare på syntetisk data. Vi ser i resultaten för Netflix-datamängden att precisionen för SVT är konkurrenskraftig med de övriga algoritmen. På Jester-datamängden visar SVT hög residual $\epsilon_{\text{SVT}}$ samt testfel $\epsilon_{\text{test}}$, men är fortfarande konkurrenskraftig med övriga algoritmer.

För IRLS-GP ses att valet $p = 0{,}5$ ger konsekvent bättre resultat än $p = 0$ både med avseende på residualen $\epsilon_{\text{alg}}$ och testfelet $\epsilon_{\text{test}}$, men med substantiellt längre beräkningstid. Detta stärker argumentet att valet $p = 0$ för IRLS-GP inte är särskilt lämpligt vid komplettering av matriser som endast är approximativt lågrang.

NIHT:s resultat är goda sett till $\epsilon_{\text{test}}$ och tidsanvändning, men det är ett resurskrävande moment att bestämma en lämplig rang. Att den algoritmspecifika residualen $\epsilon_{\text{NIHT}}$ är såpass hög kan ha att göra med att NIHT begränsas till den approximerade rangen, medan de andra algoritmerna iterativt kommer att välja en högre rang om residualerna har fastnat.

Det kan finnas flera olika förklaringar till varför resultaten skiljer sig så pass mycket mellan de olika datamängderna. Det är från resultatet för de syntetiska datamängderna förväntat att den stora skillnaden i gleshet torde leda till en märkbar skillnad i resultat då Netflix har en gleshet på 0,865 och Jester en gleshet på 0,103. Detta visar sig dock inte vara fallet. En möjlig förklaring till resultaten på verkliga datamängder skulle kunna vara att fastän en lågrangskomponent existerar är den inte mer signifikant än det brus som kommer från människors slumpmässiga handlande. Således är det inte särskilt troligt att varken SVT eller NIHT hade gett ett bättre resultat om de fick köras ännu längre.

## 5.3 Slutsatser

Baserat på våra tester på både syntetisk och verklig data i tabellerna 1, 2, 3 och 4 kan vi göra en sammanfattning av de tre algoritmerna samt försöka dra en slutsats angående vilken algoritm som är mest lämpad för olika matriser med specifika dataegenskaper.

SVT visar en mycket stabil konvergens och ger konsekvent lågt totalt fel $\epsilon$ även då det maximala antalet iterationer nås. Beräkningstiderna växer med matrisstorlek men är relativt korta i förhållande till precisionen av de kompletterade elementen som uppnås. Därmed lämpar sig SVT väl till situationer där en kompletteringsmetod med hyfsat förutsägbart och jämnt beteende krävs.

IRLS-GP når snabbt lågt $\epsilon$ och har korta beräkningstider då algoritmen konvergerar, vilket i våra tester framförallt sker för matriser med låg rang och låg gleshet. Däremot konvergerar inte algoritmen då rangen eller glesheten blir för hög och leder då till att både beräkningstid och totalt fel påverkas markant negativt. IRLS-GP kan därför anses vara bäst lämpad för problem där matrisen som ska kompletteras verkligen är en lågrangmatris med stor andel kända element.

NIHT är den snabbaste algoritmen i de fall där den konvergerar, vilket vi precis som för SVT och IRLS observerat sker främst för lågrangmatriser med låg gleshet. Men även då algoritmen inte når konvergens är den ofta fortfarande snabbare än SVT och IRLS-GP på motsvarande matris, dessvärre ibland med stort $\epsilon$. Detta gör NIHT till ett bra val som kompletteringsalgoritm då rangen på matrisen som ska kompletteras är känd på förhand.

Utöver dessa slutsatser kan vi sammanfatta utifrån våra undersökningar att generellt är matrisstorlek, rang och gleshet dataegenskaper som alla påverkar algoritmernas prestanda, där större värden på rang och gleshet korrelerar med en försämrad prestanda.

# Referenser

[1] N. Andréasson, A. Evgrafov, M. Patriksson, E. Gustavsson, Z. Nedelková, K. C. Sou och M. Önnheim. *An Introduction to Continuous Optimization.* Engelska. Tredje. Studentlitteratur, 2005, s. 508. ISBN: 9789144115290.

[2] J.-F. Cai, E. J. Candes och Z. Shen. *A Singular Value Thresholding Algorithm for Matrix Completion.* 2008. DOI: 10.48550/arXiv.0810.3286.

[3] E. J. Candès och B. Recht. "Exact matrix completion via convex optimization". I: *Found. Comput. Math.* 9.6 (2009), s. 717–772. DOI: 10.1007/s10208-009-9045-5.

[4] C. Eckart och G. Young. "The Approximation of One Matrix by Another of Lower Rank". I: *Psychometrika* 1.3 (sept. 1936), s. 211–218. DOI: 10.1007/BF02288367.

[5] Encyclopædia Britannica. *NP-complete problem.* [Online] Senast uppdaterad: 7:e februari 2025, Hämtad: 25:e februari 2025. URL: https://www.britannica.com/science/NP-complete-problem.

[6] K. Goldberg, T. Roeder, D. Gupta och C. Perkins. "Eigentaste: A constant time collaborative filtering algorithm". I: *Information Retrieval* 4.2 (2001), s. 133–151.

[7] A. Jois. *Jester Collaborative Filtering Dataset.* https://www.kaggle.com/datasets/aakaashjois/jester-collaborative-filtering-dataset. Hämtad: 20 April 2025. 2017.

[8] K. Mohan och M. Fazel. "Iterative reweighted algorithms for matrix rank minimization". I: *J. Mach. Learn. Res.* 13 (2012), s. 3441–3473. ISSN: 1532-4435,1533-7928.

[9] A. Narayanan och V. Shmatikov. *How To Break Anonymity of the Netflix Prize Dataset.* 2007. DOI: 10.48550/arXiv.cs/0610105.

[10] Netflix. *Netflix Prize Rules.* Hämtad: 2025-01-29. 2020. URL: https://web.archive.org/web/20090924184639/http://www.netflixprize.com/community/viewtopic.php?id=1537.

[11] Netflix och C. Crawford. *Netflix Prize Data.* https://www.kaggle.com/datasets/netflix-inc/netflix-prize-data. Hämtad: 20 April 2025. 2006.

[12] G. Strang. *Linear algebra and its applications.* Second. Academic Press [Harcourt Brace Jovanovich, Publishers], New York-London, 1980, s. xi+414. ISBN: 0-12-673660-X.

[13] J. Tanner och K. Wei. "Normalized iterative hard thresholding for matrix completion". I: *SIAM J. Sci. Comput.* 35.5 (2013), S104–S125. DOI: 10.1137/120876459.

## Användning av AI

I rapporten har AI, eller mer specifikt ChatGPT, använts som ett verktyg för att bland annat ge respons på text, vilket inkluderar grammatik och stavningskontroll samt feedback på stycken kring formuleringar som potentiellt behövs förtydligas. Ytterligare har AI använts vid kodningsprocessen och fungerat som ett felsökningsverktyg för att ge tips på eventuella orsaker till fel samt hur de kan lösas.

# A Källkod

## A.1 Generera matris

```
import numpy as np

def generateMatrix(m=100, n=100, r=10, sparsity=0.95, missingValue=0) -> np.ndarray:
    """
    Generates a sparse matrix with given dimensions, rank, sparsity, and missing values.

    Parameters:
    m (int): Number of rows in the matrix (default 100).
    n (int): Number of columns in the matrix (default 100).
    r (int): Rank of the matrix (default 10).
    sparsity (float): Sparsity of the matrix (default 0.95).
    missingValue (int or float): Value used for missing entries (default 0).

    Returns:
    np.ndarray: Generated sparse matrix.
    """
    A = np.random.rand(m, r)
    B = np.random.rand(r, n)

    low_rank_matrix = A @ B
    mask = np.zeros(m*n)
    mask[:int(m*n*(1-sparsity))] = 1
    np.random.shuffle(mask)
    mask = mask.reshape((m,n))
    sparse_low_rank_matrix = low_rank_matrix * mask
    return sparse_low_rank_matrix/np.max(low_rank_matrix), low_rank_matrix/np.max(low_rank_matrix)
```

## A.2 SVT kod

`custom_svds` är en funktion som baserat på rang väljer den snabbare implementationen av SVD från `NumPy` eller `SciPy`. While-loopen är en lösning för att undvika problem med att `SciPy`:s `svds` inte konvergerar.

```
import numpy as np
from tqdm import tqdm
from scipy.sparse.linalg import svds

def custom_svds(Y, k, solver, tol):
    if k < 100:
        while True:
            try:
                u, sigma, v = svds(Y, k, solver=solver, tol=tol, maxiter=k * 100)
                break
            except:
                continue
    else:
        u, sigma, v = np.linalg.svd(Y, full_matrices=False)
    return u, sigma, v

def svt(M, tau=False, max_iter=100, tol=1e-5, value_for_missing=-1, l=5, delta=False):
    '''Executes the singular value thresholding algorithm, as described in
    "A Singular Value Thresholding Algorithm for Matrix Completion".
```

```
20
21      Parameters:
22      M is the matrix to be completed.
23      tau is the value that the singular values are shrunk by each iteration.
24      max_iter is the maximum number of iterations before returning.
25      tol is the tolerance for an approximate solution to be considered acceptable.
26      value_for_missing is the value for elements of M that corresponds to elements that are missing.
27      l is the step size used when finding the appropriate amount of singular values.'''
28      # Setup
29      m, n = M.shape
30      M = M.astype(float)
31      Omega = (M != value_for_missing).astype(int)
32      known_values = Omega[Omega == 1].shape[0]
33      sigma0 = np.max(svds(M, k=1, return_singular_vectors=False, solver='propack', tol=1))
34      if not tau:
35          tau = 5 * min(m, n)
36      if not delta:
37          delta = 1.2 * m * n / known_values
38
39      Y = M * Omega * delta * np.ceil(tau / (delta * sigma0))
40      r = 0
41
42      for i in tqdm(range(max_iter), desc="Loading..."):
43          # SVD part
44          s = r + 1
45          while True:  # Get only the singular values that are larger than tau
46              u, sigma, v = custom_svds(Y, k=s, solver='propack', tol=tol)
47              s = s + l
48              if (min(sigma) < tau) or (s > min(m, n)):
49                  break
50          r = sum(s > tau for s in sigma)  # Number of singular values above the threshold
51
52          X = u @ np.diag(np.maximum(sigma - tau, 0)) @ v
53          residual = np.linalg.norm(Omega * (X - M), 'fro') / np.linalg.norm(Omega * M, 'fro')
54
55          # Check for convergence
56          if residual <= tol:
57              break
58
59          Y = (Y + delta * (M - X)) * Omega
60
61      return X, residual, i
62
```

## A.3 IRLS-GP kod

Notera att i fallet då $p = 0$ används singulärvärdesdekomposition för att göra beräkningen mer effektiv och stabil, vilket inte kan göras om $p \neq 0$.

```
1  from tqdm import tqdm
2  import scipy
3  import numpy as np
4
5  def new_gamma(gamma):
6      return max(gamma * 0.95, 1e-10)  # Prevent gamma from becoming too small
7
8  def custom_svds(Y, k, tol):
```

```python
    while True:
        try:
            u, sigma, v = scipy.sparse.linalg.svds(Y, k=k, tol=tol, solver='arpack')
            break
        except:
            continue
    return u, sigma, v

def irls_gp(M, p, max_iter=100, gamma=0.01, value_for_missing=-1, tol=1e-5):
    """
    Executes the Iteratively Reweighted Least Squares Gradient Projection (IRLS-GP) algorithm for matrix completion.

    Parameters:
    M: Input matrix with missing values indicated by `value_for_missing`
    p: Exponent in weight matrix W (p in [0,1])
    max_iter: Maximum number of iterations
    gamma: Initial regularization parameter
    value_for_missing: Value in M indicating missing entries
    tol: Tolerance for convergence
    """
    # Setup
    m, n = M.shape
    Omega = (M != value_for_missing).astype(int)
    Omega_complement = 1 - Omega

    gamma = gamma * np.linalg.norm(M, ord=2)**2  # Scale gamma based on spectral norm
    X = np.zeros_like(M)
    s = gamma**(1 - p / 2)
    r = 1
    residual = np.inf

    for i in tqdm(range(max_iter), desc="Loading..."):
        if p == 0:
            # Weight matrix when p = 0
            u, sigma, v = custom_svds(X, r, tol)
            v = np.transpose(v)
            sigma_sq = np.diag(sigma**2)
            regularizer = gamma * np.linalg.inv(sigma_sq + gamma * np.eye(r)) - np.eye(r)
            W = (v @ regularizer @ v.T + np.eye(n)) / gamma

            # Dynamically increase rank r
            if sigma[0] >= sigma[-1] / 100:
                r = min(r + 1, min(m, n))
            else:
                r = np.sum(sigma >= sigma[-1] / 100)
        else:
            # Weight matrix with fractional power for general p in (0,1]
            W = scipy.linalg.fractional_matrix_power(X.T @ X + gamma * np.eye(n), p / 2 - 1)

        # Gradient projection to update X
        X_temp = X
        for _ in range(5):
            X_temp = Omega_complement * (X_temp - s * X_temp @ W) + Omega * M

        residual = np.linalg.norm(X - X_temp, 'fro') / np.linalg.norm(Omega * M, 'fro')

        # Check for convergence
        if residual < tol:
```

```
67                  break
68
69              X = X_temp
70              gamma = new_gamma(gamma)
71              s = gamma**(1 - p / 2)
72
73          return X, residual, i
```

## A.4 NIHT kod

```
1   import numpy as np
2   from scipy.sparse.linalg import svds
3   import tqdm
4
5   def At_of(b, mask, shape): # Matrix with known values b and 0 for unknown
6       X = np.zeros(shape)
7       X[mask] = b
8       return X
9
10  def hard_threshhold(X, rank): # SVD and keep r largest singular values
11      U, S, Vt = svds(X, k=rank)
12      return U @ np.diag(S) @ Vt
13
14  def step_size_calc(A_residual, mask):
15      numerator = np.linalg.norm(A_residual, 'fro')**2
16      denominator = np.linalg.norm(A_residual[mask])**2
17      return numerator / denominator
18
19  def niht(M, r, max_iter=100, value_for_missing=-1, tol=1e-5):
20      """
21      Executes the Iterative Hard Thresholding (NIHT) algorithm for matrix completion.
22
23      Parameters:
24      M: Input matrix with missing values indicated by `value_for_missing`.
25      r: The rank for the factorization (assumed known or approximated).
26      max_iter: Maximum number of iterations.
27      tol : Tolerance for convergence.k
28      """
29      # Setup
30      mask = (M != value_for_missing)
31      b = M[mask] # Known values
32
33      X = hard_threshhold(At_of(b, mask, M.shape), r) # Hard threshhold
34      U, _, _ = svds(X, k=r)
35      for j in tqdm.tqdm(range(max_iter), desc="Loading..."):
36          P_U = U @ U.T
37          A_residual = At_of(b - X[mask], mask, M.shape)
38          step_size = step_size_calc(A_residual, mask)
39          X = hard_threshhold(X + step_size * A_residual, r)
40
41          U, _, _ = svds(X, k=r)
42          residual = np.linalg.norm(b - X[mask]) / np.linalg.norm(b)
43          if residual < tol:
44              break
45
```

```
46      return X,  residual, j
```

## A.5 Tester på syntetiska matriser

```
1   import sys
2   from pathlib import Path
3   import os
4   import numpy as np
5   import time
6
7   # Set working directory
8   os.chdir(r"C:\Path\To\KandidatResultat")
9
10  cwd = Path.cwd()
11  root = cwd.parent.parent
12  sys.path.append(str(root))
13  project_root = Path().resolve().parent
14  sys.path.append(str(project_root))
15  alg_path = "Algoritmer"
16  sys.path.append(str(alg_path))
17
18  # Import algorithm modules
19  from SVT import svt
20  from IRLS import irls_gp
21  from NIHT import niht
22  from numpy.linalg import norm
23  from generatematrix import generateMatrix
24
25  # Create output folder and timestamped log files
26  now = time.strftime('%d %b %Y %H.%M')
27  folder='Testresults from svtTest.py/'
28
29  f = open(f"{folder}SVT Test {now}.txt", "a")
30  g = open(f"{folder}IRLS Test p=0 {now}.txt", "a")
31  h = open(f"{folder}NIHT Test {now}.txt", "a")
32
33  # Write headers to log files
34  f.write(f"size;rank;sparsity;time;iter;rel_err;abs_err\n")
35  g.write(f"size;rank;sparsity;time;iter;rel_err;abs_err\n")
36  h.write(f"size;rank;sparsity;time;iter;rel_err;abs_err\n")
37
38  # Test parameters
39  sizes = [500, 1000, 2000]
40  ranks = [5, 10, 50]
41  sparsitys = [0.5, 0.7, 0.95]
42
43  # Test loops
44  for size in sizes:
45      for rank in ranks:
46          for sparsity in sparsitys:
47              if (rank != 50 or size != 2000): # Skip expensive test
48                  M, MT = generateMatrix(size, size, rank, sparsity = sparsity) # Generate test matrix
49
50                  # SVT
51                  t0 = time.time()
52                  tau = 5 * size
```

```
53                  C, res, it = svt(M, tau, max_iter=3000, tol=1e-5, value_for_missing=0, delta=2)
54                  t1 = time.time()
55                  t = t1 - t0
56                  abs_err = norm(C - MT, 'fro') / norm((MT), 'fro')
57                  f.write(f"{size};{rank};{sparsity};{t:.2f};{it+1};{res:.2e};{abs_err:.2e}\n")
58                  f.flush()
59                  os.fsync(f.fileno())
60
61                  # IRLS
62                  t0 = time.time()
63                  C, res, it = irls_gp(M, 0, max_iter=3000, value_for_missing=0, tol=1e-5)
64                  t1 = time.time()
65                  t = t1 - t0
66                  abs_err = norm(C - MT, 'fro') / norm((MT), 'fro')
67                  g.write(f"{size};{rank};{sparsity};{t:.2f};{it+1};{res:.2e};{abs_err:.2e}\n")
68                  g.flush()
69                  os.fsync(g.fileno())
70
71                  # NIHT
72                  t0 = time.time()
73                  C, res, it = niht(M, rank, max_iter=3000, tol=1e-5, value_for_missing=0)
74                  t1 = time.time()
75                  t = t1 - t0
76                  abs_err = norm(C - MT, 'fro') / norm((MT), 'fro')
77                  h.write(f"{size};{rank};{sparsity};{t:.2f};{it+1};{res:.2e};{abs_err:.2e}\n")
78                  h.flush()
79                  os.fsync(h.fileno())
```