

KU LEUVEN

MODELLERING EN SIMULATIE

Practicum 2: **Lagerangbenaderingen**

Armin Halilovic - r0679689

23 December 2016

Contents

1 Een aanbevelingssysteem voor films	2
1.1 Matrixvervollediging	2
1.1.1 Opdracht 2	2
1.1.2 Opdracht 3	3
1.1.3 Opdracht 4	3
1.1.4 Opdracht 5	4
1.1.5 Opdracht 6	4
1.1.6 Opdracht 8	4
1.1.7 Opdracht 10	4
1.1.8 Opdracht 11	5
1.1.9 Opdracht 12	5
1.1.10 Opdracht 14	6
1.1.11 Opdracht 15	6
1.2 Clustering	7
1.2.1 Opdracht 16	7
1.2.2 Opdracht 18	8
1.2.3 Opdracht 19	9
2 Evaluatie	9
2.1 Opdracht 1	9
2.2 Opdracht 2	9
2.3 Opdracht 3	10
Appendices	11
A Functies	11
A.1 r0679689_sparseModel.m	11
A.2 r0679689_optimalCoefficients.m	11
A.3 r0679689_userMeans	12
A.4 r0679689_RMSE	12
A.5 r0679689_rank1MatrixPursuit.m	12
A.6 r0679689_actualBestMovies.m	13
A.7 r0679689_predictedBestMovies.m	13
A.8 r0679689_correlationMatrix.m	14
A.9 r0679689_similarMovies.m	15
B Opdrachten	15
B.1 Opdracht 1	15
B.2 Opdracht 2	16
B.3 Opdracht 4	16
B.4 Opdracht 5	16
B.5 Opdracht 6	17
B.6 Opdracht 8	17
B.7 Opdracht 10	18
B.8 Opdracht 11	18
B.9 Opdracht 12	18
B.10 Opdracht 15	18
B.11 Opdracht 18	19

In dit verslag worden oplossingen gegeven voor de opdrachten in het practicum. De code voor alle opdrachten staat onder appendix A.

1 Een aanbevelingssysteem voor films

1.1 Matrixvervollediging

1.1.1 Opdracht 2

Hoeveel geheugenruimte is nodig om de volle beoordelingenmatrix $\text{full}(R)$ voor te stellen?
Hoeveel geheugenruimte is nodig om de ijle beoordelingenmatrix R voor te stellen in het coördinaatformaat?

Hoeveel geheugenruimte is vereist om de rang- r lagerangbenadering WF^T uit verg. (2) compact voor te stellen (als functie van r)?

Maak een duidelijke figuur waarin je het geheugengebruik van de lagerangbenadering uit verg. (2) en het geheugengebruik van de ijle matrix R uitzet als functie van r voor de waarden $r = 1, \dots, 500$

Voor welke waarde van r snijden deze twee lineaire functies?

Om de volle beoordelingenmatrix $\text{full}(R)$ ($R \in \mathbb{R}^{m \times n}$) voor te stellen is er $8 \times m \times n$ bytes geheugenruimte nodig.

Om een matrix in coördinaatformaat op te slaan zijn verzamelingen van rij-indices i , kolom-indices j , en waarden $r_{i,j}$ nodig. De verzamelingen van indices kunnen bestaan uit gehele getallen. Er is dus

$$4 \times r + 4 \times r + 8 \times r = 16 \times r$$

bytes geheugenruimte nodig voor het coördinaatformaat, waarbij r het aantal niet-nulwaarden is van R .

Om de rang- r lagerangbenadering $R \approx WF^T$ ($W \in \mathbb{R}^{m \times r}$, $F \in \mathbb{R}^{n \times r}$) voor te stellen is er

$$8 \times m \times r + 8 \times n \times r = 8r(m + n)$$

bytes geheugenruimte nodig.

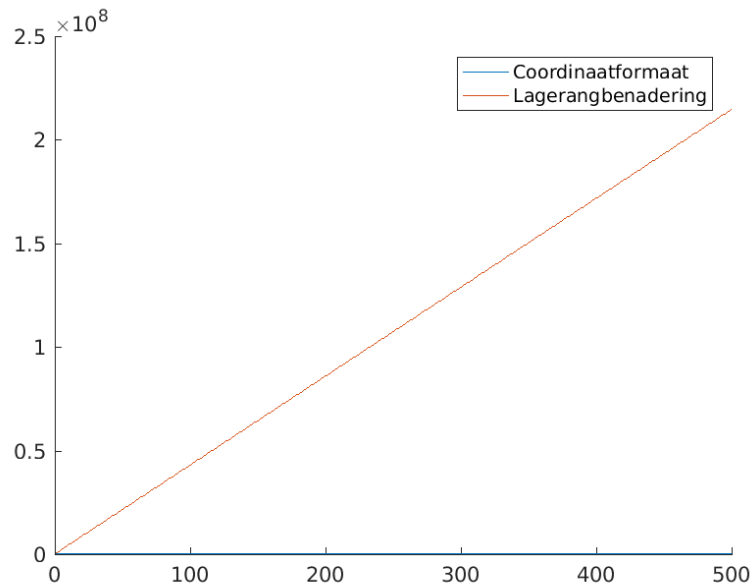


Figure 1: Het geheugengebruik van de lagerangbenadering en van de beoordelingenmatrix in coördinaat-formaat

TODO: De twee functies snijden nooit en ik moet de berekening eens checken.

1.1.2 Opdracht 3

Bewijs dat σ_j de j^{de} grootste singuliere waarde van A is en dat

$$A - E_k = \sum_{j=1}^k \sigma_j \mathbf{u}_j \mathbf{v}_j^T$$

de rang-k (met $k \leq r$) afgeknotte singulierewaardeontbinding van A is.

TODO: Bewijs eens uitschrijven.

1.1.3 Opdracht 4

Stel dat we de stappen 46 van algoritme 1 slechts eenmaal zouden uitvoeren. Kan men dit algoritme dan toepassen op een 240 000 x 33 000 matrix met 22 miljoen niet-nulwaarden (i.e., de volledige MovieLens databank) op een laptop met 8GB werkgeheugen en 32GB swap geheugen? Waarom wel of niet?

Dit is mogelijk als ijle matrices worden gebruikt en als het ijlheidspatroon kan worden behouden. Bij gebruik van volle matrices is er te veel geheugenruimte nodig.

Output van de code (zie B.3):

```
Totaal bruikbaar geheugen: 40GB
Nodig geheugen met volle matrices: 190.08GB
Nodig geheugen als het ijlheidspatroon behouden kan worden: 1.06GB
```

1.1.4 Opdracht 5

function [X] = SN_sparseModel(Uk, sk, Vk, A)

De geheugencomplexiteit van je algoritme in functie van m , n , r en ζ mag hoogstens $\mathcal{O}(\zeta)$ bedragen - je mag hierbij wel veronderstellen dat $\max\{m, n\} \leq \zeta$. Hoe heb je deze doelstelling bereikt?

Zie A.1.

De doelstelling werd bereikt door enkel plaats te alloceren voor vectoren met lengte ζ :

```
[i, j, v] = find(A);
```

i , j , en v zijn vectoren die de matrix A voorstellen in coördinaatformaat. De waarden in vector v worden dan overschreven door de waarden die horen te staan in $U_k * \text{diag}(s_k) * V_k^T$ op dezelfde posities (i, j) :

```
for val = 1:length(i)
    v(val) = Uk(i(val), :) .* sk' * Vk(j(val), :)';
end
```

Deze nieuwe waarden worden een per een berekend, aangezien een uitgewerkte kopie van $U_k * \text{diag}(s_k) * V_k^T$ bijhouden geheugencomplexiteit $\mathcal{O}(m * n)$ zou bedragens.

1.1.5 Opdracht 6

function [s] = SN_optimalCoefficients(Uk, Vk, A)

De geheugencomplexiteit van je algoritme in functie van m , n , k en ζ mag hoogstens $\mathcal{O}(k\zeta)$ bedragen - je mag hierbij wel veronderstellen dat $\max\{m, n\} \leq \zeta$. Hoe heb je deze doelstelling bereikt?

Zie A.2.

De doelstelling werd bereikt door plaats te alloceren voor $k\zeta$ elementen voor de matrix B :

```
B = sparse([], [], [], m*n, k, nnz(A) * k);
```

Tijdens het invullen van deze matrix werd enkel gebruik gemaakt van de `sparseModel` functie, die geheugencomplexiteit $\mathcal{O}(\zeta)$ bedraagt.

1.1.6 Opdracht 8

Hoeveel bedragen de 3 laagste gemiddelde beoordelingen? Hoeveel gebruikers gaven exact 5 als gemiddelde score?

Output van de code (zie B.6):

```
De drie laagste gemiddelde beoordelingen zijn:
    0.81 van gebruiker 40142
    0.98 van gebruiker 33685
    1.04 van gebruiker 10405
39 gebruikers hebben een gemiddelde score van exact 5.
```

1.1.7 Opdracht 10

Hoeveel bedraagt de RMSE tussen T en de ijle matrix $P_\Omega(\mu \mathbf{1}^T)$?

Output van de code (zie B.7):

RMSE tussen T en de ijle matrix: 0.905725

1.1.8 Opdracht 11

Pas `SN_rank1MatrixPursuit` toe op de gekende beoordelingenmatrix R , waarbij je $r = 30$ als gewenste rang kiest en de testdata T als derde argument kiest. Maak een duidelijke figuur van de evolutie van de RMSE.

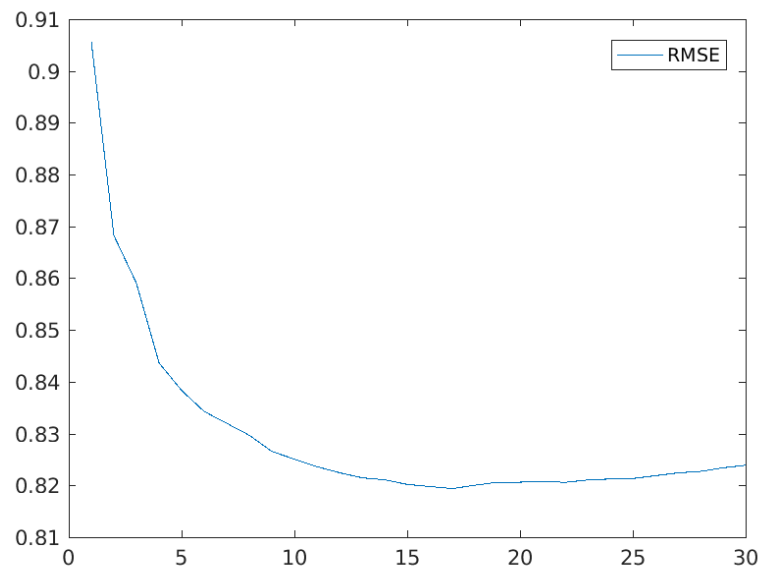


Figure 2: De evolutie van de RMSE bij toepassing van `rank1MatrixPursuit(R, 30, T)`

1.1.9 Opdracht 12

Bereken $[U_{20}, s_{20}, V_{20}, \sim] = \text{SN_rank1MatrixPursuit}(R, 20, T)$. Wat zijn de waarden van de vector s_{20} , afgerond tot 2 cijfers na de komma?

Output van de code (zie B.9):

```
0.97
1584.82
754.94
740.23
572.71
471.93
330.87
334.54
255.20
300.50
281.20
265.00
201.74
208.67
235.44
166.34
188.72
184.68
```

174.97
175.14

1.1.10 Opdracht 14

function [movieIDs, score] = SN_predictedBestMovies(Uk, sk, Vk)

De geheugencomplexiteit van je algoritme in functie van m, n en k mag hoogstens $\mathcal{O}(\max\{m, n\})$ bedragen. Hoe heb je deze doelstelling bereikt?

Zie A.7.

De doelstelling werd bereikt door per film elke beoordeling apart te berekenen en op te tellen in een variabele:

```
tmp = sk .* Vk(movie, :);  
sum = 0;  
for rating = 1:m  
    sum = sum + Uk(rating, :) * tmp;  
end
```

De gemiddelde score per film werd dan berekend en opgeslagen in de output vector **score**. Het berekenen van een beoordeling $U_k(\text{rating}, :) \cdot s_k \cdot V_k(\text{movie}, :)^T$ bedraagt geheugencomplexiteit $\mathcal{O}(k) < \mathcal{O}(\max\{m, n\})$.

1.1.11 Opdracht 15

Bereken met behulp van de functies uit de twee voorgaande opgaves de 25 films met de hoogste gemiddelde beoordelingen (op basis van de gekende beoordelingenmatrix R) en de 25 films met de hoogste gemiddelde voorspelde beoordelingen.

Welke lijst van films vind je het meest realistisch? Motiveer je keuze. Onderzoek in het bijzonder het aantal gekende beoordelingen in de trainingsdata R van de films in de twee lijsten en betrek deze informatie in je motivering.

Output van de code (zie B.10):

average rating	-	movie ID	-	movie name	-	number of reviews
Processing actualBestMovies...						
4.31	-	938	-	Band of Brothers (2001)	-	2988
4.29	-	4455	-	Death on the Staircase (Soupons) (2004)	-	17
4.24	-	510	-	City of God (Cidade de Deus) (2002)	-	9687
4.23	-	1750	-	Lives of Others, The (Das leben der Anderen) (2006)	-	4400
4.23	-	2486	-	Dark Knight, The (2008)	-	15052
4.20	-	390	-	Spirited Away (Sen to Chihiro no kamikakushi) (2001)	-	10553
4.20	-	183	-	Amelie (Fabuleux destin d'Amlie Poulain, Le) (2001)	-	18057
4.18	-	4418	-	Frozen Planet (2011)	-	30
4.18	-	3477	-	Inception (2010)	-	10541
4.16	-	822	-	Lord of the Rings: The Return of the King, The (2003)	-	23862
4.15	-	4066	-	Intouchables (2011)	-	2136
4.15	-	194	-	Lord of the Rings: The Fellowship of the Ring, The (2001)	-	27243
4.14	-	4158	-	Black Mirror (2011)	-	467
4.13	-	4227	-	Bleak House (2005)	-	30
4.12	-	483	-	Lord of the Rings: The Two Towers, The (2002)	-	25214
4.11	-	892	-	Eternal Sunshine of the Spotless Mind (2004)	-	17391
4.11	-	1932	-	Departed, The (2006)	-	11450
4.10	-	825	-	Fog of War: Eleven Lessons from the Life of Robert S. McNamara, The (2003)	-	2080
4.10	-	4781	-	Whiplash (2014)	-	505
4.09	-	1227	-	Old Boy (2003)	-	5161
4.08	-	4549	-	Louis C.K.: Oh My God (2013)	-	243
4.08	-	3786	-	Louis C.K.: Shameless (2007)	-	711

4.08 - 4755 - Normal Heart, The (2014)	- 33
4.08 - 942 - Best of Youth, The (La meglio gioventù) (2003)	- 340
4.07 - 3784 - Louis C.K.: Chewed Up (2008)	- 568
Processing rank1MatrixPursui	
Processing predictedBestMovies...	
4.09 - 183 - Amelie (Fabuleux destin d'Amélie Poulain, Le) (2001)	- 18057
4.07 - 822 - Lord of the Rings: The Return of the King, The (2003)	- 23862
4.07 - 2486 - Dark Knight, The (2008)	- 15052
4.06 - 194 - Lord of the Rings: The Fellowship of the Ring, The (2001)	- 27243
4.04 - 892 - Eternal Sunshine of the Spotless Mind (2004)	- 17391
4.03 - 483 - Lord of the Rings: The Two Towers, The (2002)	- 25214
4.00 - 510 - City of God (Cidade de Deus) (2002)	- 9687
3.99 - 390 - Spirited Away (Sen to Chihiro no kamikakushi) (2001)	- 10553
3.97 - 1932 - Departed, The (2006)	- 11450
3.95 - 3477 - Inception (2010)	- 10541
3.92 - 154 - Donnie Darko (2001)	- 14884
3.91 - 1456 - Batman Begins (2005)	- 15237
3.90 - 497 - Pianist, The (2002)	- 8414
3.90 - 1928 - Pan's Labyrinth (Laberinto del fauno, El) (2006)	- 9356
3.90 - 196 - Beautiful Mind, A (2001)	- 17031
3.90 - 2586 - WALLE (2008)	- 9987
3.90 - 1955 - Prestige, The (2006)	- 9379
3.88 - 761 - Kill Bill: Vol. 1 (2003)	- 17613
3.87 - 625 - Finding Nemo (2003)	- 18674
3.87 - 2336 - No Country for Old Men (2007)	- 8489
3.86 - 1109 - Incredibles, The (2004)	- 15972
3.86 - 310 - Bourne Identity, The (2002)	- 16295
3.86 - 1738 - V for Vendetta (2006)	- 11978
3.85 - 652 - Pirates of the Caribbean: The Curse of the Black Pearl (2003)	- 20025
3.85 - 2944 - Inglourious Basterds (2009)	- 7805

De tweede lijst van films vind ik het meest realistisch. Wat hier meteen opvalt is dat de films met weinig beoordelingen uit de top zijn verdwenen (in dit geval, alle films met minder dan 5162 beoordelingen). Deze films hadden weinig beoordelingen, maar alle beoordelingen waren hoog genoeg om de films in de top 25 te doen belanden. Een film met 1 beoordeling van 5 zal bijvoorbeeld als eerste in de lijst staan.

De lijst die uit predictedBestMovies komt werkt dit effect tegen. Het aantal beoordelingen per film speelt hier ook een belangrijke factor in het bepalen van de score. Om nu in de lijst te belanden, moet een film een relatief hoog aantal beoordelingen hebben met een hoge waarde.

Daarnaast valt ook op dat de Lord of the Rings films en The Dark Knight hoger in de lijst staan en dat Batman Begins en The Prestige nieuw in de lijst een vrij hoge plaatsen hebben gekregen, wat de tweede lijst duidelijk een betere keuze maakt.

1.2 Clustering

1.2.1 Opdracht 16

function [C] = SN_correlationMatrix(Uk, sk, Vk)

De geheugencomplexiteit van je algoritme in functie van m, n en k mag hoogstens $\mathcal{O}(k^2 + n^2)$ bedragen. Hoe heb je deze doelstelling bereikt?

Zie A.8.

De doelstelling werd bereikt door de formule voor de correlatiematrix C

$$C = \frac{1}{m-1}((R_k - \mathbf{1}\mu^T)\Sigma^{-1})^T((R_k - \mathbf{1}\mu^T)\Sigma^{-1})$$

te herschrijven zodat $R_k \in \mathbb{R}^{n \times n}$ ipv $R_k \in \mathbb{R}^{m \times n}$.
De berekening wordt dan

$$C = \left(\sum_{j=1}^{\lfloor m/n \rfloor} (R_j - \mathbf{1}\mu^T)^T * (R_j - \mathbf{1}\mu^T) \right) ./ (\sigma * \sigma^T) / (m - 1)$$

waarbij $R_j \in \mathbb{R}^{n \times n}$, en waarbij $\mu \in \mathbb{R}^n$ de gemiddeldes en $\sigma \in \mathbb{R}^n$ de standaardafwijkingen bevatten voor de beoordelingen van de films. In de code worden de blokken $R_j - \mathbf{1}\mu^T$ berekend met geheugencomplexiteit $\mathcal{O}(n^2)$:

```
R.j = Uk(row_from:min(row_from+n-1, m), :) * V - means';
C = C + (R.j' * R.j);
```

waarbij V gelijk is aan $\text{diag}(s_k) * V_k^T$. V bedraagt geheugencomplexiteit $\mathcal{O}(kn)$.

Aangezien $R_j \in \mathbb{R}^{n \times n}$ en $kn < k^2 + n^2$ altijd geldt wanneer $k > 0$, wordt de opgelegde beperking van $\mathcal{O}(k^2 + n^2)$ niet overschreden.

1.2.2 Opdracht 18

Welke n films zijn het sterkst positief gecorreleerd met de film met volgnummer k , voor volgende waarden van n en k ? Zijn deze resultaten realistisch? Motiveer. Zijn er gelijkenissen met de aanbevelingen van andere aanbevelingssystemen, zoals bijvoorbeeld IMDB.com, MovieLens, Amazon.com of Netflix?

Output van de code (zie B.11):

```
10 most similar movies for Harry Potter and the Chamber of Secrets (2002) (id 446):
    Harry Potter and the Sorcerer's Stone (a.k.a. Harry Potter and the Philosopher's Stone) (2001)
    Harry Potter and the Goblet of Fire (2005)
    Harry Potter and the Prisoner of Azkaban (2004)
    Harry Potter and the Order of the Phoenix (2007)
    Harry Potter and the Half-Blood Prince (2009)
    Chronicles of Narnia: The Lion, the Witch and the Wardrobe, The (2005)
    Harry Potter and the Deathly Hallows: Part 1 (2010)
    Pirates of the Caribbean: Dead Man's Chest (2006)
    Harry Potter and the Deathly Hallows: Part 2 (2011)
    Pirates of the Caribbean: At World's End (2007)

5 most similar movies for Captain America: The First Avenger (2011) (id 3854):
    Thor (2011)
    Amazing Spider-Man, The (2012)
    Iron Man 3 (2013)
    Quantum of Solace (2008)
    Hellboy II: The Golden Army (2008)

3 most similar movies for Zero Dark Thirty (2012) (id 4355):
    Captain Phillips (2013)
    Rush (2013)
    Lincoln (2012)

3 most similar movies for Futurama: Bender's Game (2008) (id 2747):
    Futurama: The Beast with a Billion Backs (2008)
    Fido (2006)
    World's End, The (2013)

9 most similar movies for Monsters, Inc. (2001) (id 160):
    Finding Nemo (2003)
    Incredibles, The (2004)
```

Shrek (2001)
Ratatouille (2007)
Shrek 2 (2004)
WALLE (2008)
Ice Age (2002)
Up (2009)
X2: X-Men United (2003)

Deze resultaten lijken mij heel realistisch. De films die sterk positief correleren met elkaar zijn telkens uit dezelfde flim franchises (bv Harry Potter en Futurama met andere Harry Potter en Futurama films), films in hetzelfde genre (Captain America met actie/avonturenfilms, Zero Dark Thirty met actie/drama films, Futurama met komedie films), en films in dezelfde stijl of met dezelfde animatiefilm studio's (Monsters, Inc.).

Dit gedrag doet zich ook voor in andere aanbevelingssystemen.

1.2.3 Opdracht 19

Bewijs dat de correlatiematrix C in de vergelijking

$$C = \frac{1}{m-1}((R_k - \mathbf{1}\mu^T)\Sigma^{-1})^T((R_k - \mathbf{1}\mu^T)\Sigma^{-1})$$

hoogstens rang $r + 1$ heeft wanneer de voorspelde beoordelingenmatrix A exact rang r heeft.

TODO: bewijs uitschrijven

2 Evaluatie

2.1 Opdracht 1

Hoeveel tijd heb je gespendeerd aan het oplossen van de opdrachten? Hoeveel tijd heb je gespendeerd aan het schrijven van het verslag?

Ik heb ongeveer 25 uur gespendeerd aan het oplossen van de opdrachten en ongeveer 8 uur aan het schrijven van het verslag.

2.2 Opdracht 2

In de loop van deze opdracht hebben we allerlei veronderstellingen gemaakt om ons nieuw aanbevelingssysteem op te stellen. Wat zijn je bedenkingen hierbij? Vind je de resultaten realistisch? Zou je het ontwikkelde aanbevelingssysteem durven toevoegen aan de lijst van aanbevelingssystemen van MovieLens?

Op basis van opdrachten 15 en 18 vind ik dat de resultaten van de veronderstellingen realistisch zijn. Aangezien een korte google search over aanbevelingssystemen mij een boek van meer dan 500 pagina's aanbeveelt zou ik voor een grote website zoals MovieLens niet meteen dit systeem aanbevelen zonder de andere opties eens te bekijken. Er zal waarschijnlijk wel iets efficiënter of nauwkeuriger zijn. Dit systeem zou misschien wel gebruikt kunnen worden bij mijn studentenjob voor een kleinere website, waar een aanbevelingssysteem op de planning staat voor later.

2.3 Opdracht 3

Welke bedenkingen heb je bij dit practicum? Was de opdracht (veel) te gemakkelijk, (veel) te moeilijk of van een gepaste moeilijkheidsgraad? Wat zou je zelf anders aangepakt hebben? Was de terminologie voldoende duidelijk?

Het theoretische aspect van de opdracht is helemaal in orde. Het was een heel interessante toepassing van wat we hebben geleerd. Alles in de opdracht was duidelijk. Alles snel genoeg krijgen binnen de opgelegde geheugencomplexiteiten zodat ik niet uren zou moeten wachten op resultaten was erg moeilijk. Daar is het meeste van de tijd naartoe gegaan.

Appendices

A Functions

A.1 r0679689_sparseModel.m

```
function [X] = r0679689_sparseModel(Uk, sk, Vk, A)
% Calculates Uk * diag(sk) * Vk' while keeping the sparsity pattern of sparse matrix A
% size(Uk) = m x k, size(sk) = k x 1, size(Vk) = n x k
% memory constraint: O(C), C = amount of nonzeros in A

% find at which positions A has values, then calculate output values only for those positions
[i, j, v] = find(A); % O(C)
for val = 1:length(i)
    v(val) = Uk(i(val), :) .* sk' * Vk(j(val), :)' ; % O(k) for intermediary calculation, k < m,n < C
end
X = sparse(i, j, v, size(A, 1), size(A, 2)); % O(C)
end

%{
% FULL MATRIX VERSION
Xx = zeros(size(A, 1), size(A, 2));
tmp = Uk * diag(sk) * Vk';
Xx(I, J) = tmp(I, J);
%}

%{
% FIRST SPARSE MATRIX VERSION, "X2(I, J) =" takes too much memory at once
X2 = sparse(size(A, 1), size(A, 2));
X2(I, J) = Uk(I, :) * S * Vk(J, :)' ;
X2(A == 0) = 0;
%}

%{
% SECOND SPARSE MATRIX VERSION
[~, S_rows] = size(Uk);
[~, S_cols] = size(Vk);
S = sparse(S_rows, S_cols);
for z = 1:length(sk)
    S(z, z) = sk(z);
end

[I, J] = find(A);
X = sparse([], [], [], size(A, 1), size(A, 2), length(I));

for z = 1:length(I)
    X(I(z), J(z)) = Uk(I(z), :) * S * Vk(J(z), :)' ;
end
%}
```

A.2 r0679689_optimalCoefficients.m

```
function [s] = r0679689_optimalCoefficients(Uk, Vk, A)
% Finds the optimal solution for least squares problem min(|| a - Bx ||)
% size(Uk) = m x k, size(Vk) = n x k
% memory constraint: O(kC), C = amount of nonzeros in A

[m, k] = size(Uk);
[n, ~] = size(Vk);

B = sparse([], [], [], m*n, k, nnz(A) * k); % allocate space for k*C elements
for i = 1:k
    tmp = r0679689_sparseModel(Uk(:, i), 1, Vk(:, i), A); % O(C)
    B(:, i) = tmp(:);
end
```

```

    [s, ~] = lsqr(B, A(:));
end

%{
% FULL MATRIX VERSION
BBB = zeros(mn, k);
for i = 1:k
    tmp = Uk(:, i) * Vk(:, i)';
    BBB(1:m*n, i) = tmp(:);
end
%}

%{
% SPARSE MATRIX VERSION 1
[I, J] = find(A);
BB = sparse(zeros(mn, k));
for i = 1:k
    tmp = sparse(m, n);
    tmp(I, J) = Uk(I, i) * Vk(J, i)';
    BB(:, i) = tmp(:);
end
%}

```

A.3 r0679689_userMeans

```

function [mu] = r0679689_userMeans(A)
% returns a vector of means of the rows of A

m = size(A, 1);
mu = zeros(m, 1);

for i = 1:m
    mu(i) = mean(nonzeros(A(i, :)));
end
end

```

A.4 r0679689_RMSE

```

function [err] = r0679689_RMSE(A, B)
% calculates the root mean square error between two matrices A and B
% A and B have the same sparsity pattern

C = length(find(A));
err = (1 / sqrt(C)) * norm(A - B, 'fro');
end

```

A.5 r0679689_rank1MatrixPursuit.m

```

function [U, s, V, rmse] = r0679689_rank1MatrixPursuit(R, k, T)
% Rank-1 matrix pursuit for completing an incomplete matrix R
%{
IN
    R = sparse matrix of known ratings, size(R) = m, n
    k = desired rank for approximation, 1 < k < min(m, n)
    T = sparse matrix of known ratings, size(T) = m, n
OUT
    U, s, V form the low-rank approximation R' of R, R' = U * diag(s) * V'
    rmse contains the root-mean-square error between incomplete matrix T
    and P(omega)(R) where omega is the sparsity pattern of T

    size(U) = m, k

```

```

size(s) = k
size(V) = n, k
size(rsme) = k

```

Also, approximation $R' = W * F' = U * \text{diag}(s) * V'$
`%}`

```

[m, n] = size(R);
rmse = zeros(k,1);
U = zeros(m,k);
V = zeros(n,k);
U(:,1) = r0679689_userMeans(R);
V(:,1) = ones(n,1);
fprintf('processing for j = 1\n');
S = r0679689_sparseModel(U(:,1),1,V(:,1),R);
P = r0679689_sparseModel(U(:,1),1,V(:,1),T);
E = R - S;
rmse(1) = r0679689_RMSE(T,P);

for j = 2 : k
    fprintf('processing for j = %i\n', j)
    [u,~,v] = svds(E,1);
    U(:,j) = u;
    V(:,j) = v;
    s = r0679689_optimalCoefficients(U(:,1:j),V(:,1:j),R);
    S = r0679689_sparseModel(U(:,1:j),s,V(:,1:j),R);
    P = r0679689_sparseModel(U(:,1:j),s,V(:,1:j),T);
    E = R - S;
    rmse(j) = r0679689_RMSE(T,P);
end
end

```

A.6 r0679689_actualBestMovies.m

```

function [movieIDs, score] = r0679689_actualBestMovies(R)
% Returns a list of movies and their scores, sorted on scores in descending order
% R is a sparse matrix of known reviews of users for movies, size(R) = m, n

means = r0679689_userMeans(R');
[score, movieIDs] = sort(means, 'descend');
end

```

A.7 r0679689_predictedBestMovies.m

```

function [movieIDs, score] = r0679689_predictedBestMovies(Uk, sk, Vk)
% returns a list of movieIDs with their scores, sorted on scores in descending order
% the scores are averages based on the full matrix of user reviews  $R_{:,k} = U_{:,k} * \text{diag}(sk) * V_{:,k}'$ 
% size(Uk) = m, k, size(sk) = k, size(Vk) = n, k
% memory complexity constraint =  $O(\max(m, n))$ 

m = size(Uk, 1);
n = size(Vk, 1);

score = zeros(n, 1); % O(n)
for movie = 1:n
    tmp = sk .* Vk(movie, :); % O(k), k < max(m, n)
    sum = 0;
    for rating = 1:m
        sum = sum + Uk(rating, :) * tmp; % O(k), k < max(m, n)
    end
    score(movie) = sum / m;
end
[score, movieIDs] = sort(score, 'descend');
end

```

```

%{

```

```
% FULL MATRIX VERSION
Rk = Uk * diag(sk) * Vk'; % O(m * n) > O(max(m, n))
[movieIDs, score] = r0679689_actualBestMovies(Rk);
%}
```

A.8 r0679689_correlationMatrix.m

```
function [C] = r0679689_correlationMatrix(Uk, sk, Vk)
% returns the correlation matrix C that belongs to the matrix formed by Uk * diag(sk) * Vk'
% size(Uk) = m x k, size(sk) = k x 1, size(Vk) = n x k
% size(C) = n x n
% memory complexity limit = O(k^2 + n^2)

[m, ~] = size(Uk);
[n, ~] = size(Vk);
V = diag(sk) * Vk'; % O(k * n), kn < k^2 + n^2

means = zeros(n, 1); % O(n)
stds = zeros(n, 1); % O(n)
for i = 1:n
    tmp = Uk * V(:, i); % O(m)
    means(i) = mean(tmp);
    stds(i) = std(tmp);
end

C = zeros(n, n); % O(n^2)
for i = 1:m/n
    row_from = 1+(i-1)*n;

    R_j = Uk(row_from:min(row_from+n-1, m), :) * V - means'; % O(n * n)

    C = C + (R_j' * R_j);
end

C = C ./ (stds .* stds') / (m - 1);
end

%{
for i = 1:n
    fprintf('correlation for movie %i\n', i)
    tic

    F_i = Uk(:, 1:length(sk)) * (sk .* Vk(i, :)); % O(m)
    meanF_i = mean(F_i);
    stdF_i = std(F_i);

    for j = i+1:n
        F_j = Uk(:, 1:length(sk)) * (sk .* Vk(j, :)); % O(m)
        meanF_j = mean(F_j);
        stdF_j = std(F_j);

        tmp = sum( (F_i - meanF_i).*(F_j - meanF_j) ) / (mm * stdF_i * stdF_j);
        C(i, j) = tmp;
        C(j, i) = tmp;
    end

    toc
end
%}

%{
for i = 1:n
    fprintf('correlation for movie %i\n', i)
    tic

    % calculate standard deviation using shifted data
    shift = Uk(1, :) * Vk(i, :);
    sum_F_i = 0;
    sum_squares_shifted = 0;
```

```

for row = 1:m
    x = Uk(row, :) * Vk(i, :)';
    sum_F_i = sum_F_i + x;
    sum_squares_shifted = sum_squares_shifted + (x - shift)^2;
end
stdF_i = sqrt( (sum_squares_shifted - ((sum_F_i-m*shift)^2/m)) / (m - 1));
meanF_i = sum_F_i / m;

for j = i+1:n

    shift = Uk(1, :) * Vk(j, :)';
    sum_F_j = 0;
    sum_squares_shifted = 0;
    for row = 1:m
        x = Uk(row, :) * Vk(j, :)';
        sum_F_j = sum_F_j + x;
        sum_squares_shifted = sum_squares_shifted + (x - shift) * (x - shift);
    end
    stdF_j = sqrt( (sum_squares_shifted - ((sum_F_j-m*shift)^2/m)) / (m - 1));
    meanF_j = sum_F_j / m;

    corrSum = 0;
    for k = 1:m
        x1 = Uk(k, :) .* sk' * Vk(i, :)' - meanF_i;
        x2 = Uk(k, :) .* sk' * Vk(j, :)' - meanF_j;
        corrSum = corrSum + x1 * x2;
    end

    tmp = corrSum / mm / stdF_i / stdF_j;
    C(i, j) = tmp;
    C(j, i) = tmp;
end

toc
end
%}

```

A.9 r0679689_similarMovies.m

```

function [moviedIDs] = r0679689_similarMovies(C, m, n)
% return a vector of n movies that are the most similar to movie m

    movies = C(m, :);
    [~, movies] = sort(movies, 'descend');
    moviedIDs = movies(2:n+1);

end

```

B Opdrachten

B.1 Opdracht 1

```

close all
clear
load('MovieLens20M_Subset.mat')

fig = figure;
spy(R(1:1000, 1:1000));
saveas(fig, 'ex1_R.png');

fig = figure;
spy(T(1:1000, 1:1000));
saveas(fig, 'ex1_T.png');

```

B.2 Opdracht 2

```
close all

m = 48769;
n = 4901;

r = linspace(1, 500, 500);

fullSize = m * n; % 239016869
coördinaatSize = 16 * r;
lagerangSize = 8 * r * (m + n); % 429360 .. 214680000

fig = figure;
hold on;
plot(r, coördinaatSize);
plot(r, lagerangSize);
legend('Coördinaatformaat', 'Lagerangbenadering');

saveas(fig, 'ex2.png')
```

B.3 Opdracht 4

```
int = 4; % 4 bytes
double = 8; % 8 bytes

m = 240000;
n = 33000;
amountOfValues = 22000000; % 22 000 000 niet-nul waarden in de matrix

% volle matrix: m * n = 7 920 000 000 cellen
% ijle matrix: 2 * 22 000 000 voor i en j vectoren + 22 000 000 voor r_{i,j} waarden = 66 000 000

memory = 8; % 8 GB
swapMemory = 32; % 32 GB
totalMemory = (memory+swapMemory) * 10^9; % convert to byte

% stap 4: beste rang-1 benadering zoeken: m+n+1 getallen voor u, v, en sigma
step4 = (m+n+1)*double;

% stap 5: (nieuwe matrix) = (oude matrix) - (rang-1 benadering)
step5 = 3*m*n*double;

% stap 6:
step6 = 1 * int;

totalMemoryUsage = step4 + step5 + step6;
fprintf('Totaal bruikbaar geheugen: %iGB \n', totalMemory / 10^9)
fprintf('Nodig geheugen met volle matrices: %.2fGB \n', totalMemoryUsage / 10^9)

step4 = (m+n+1)*double;
step5 = 3*(2*amountOfValues*int + amountOfValues*double);
step6 = 1 * int;
totalMemoryUsage = step4 + step5 + step6;
fprintf('Nodig geheugen als het ijlhidspatroom behouden kan worden: %.2fGB \n\n', totalMemoryUsage / 10^9)
```

B.4 Opdracht 5

```
TEST = [0 3 0 2 0 1 0 0 0 0 0 3 0 2 0 1 0 0 0 0;
        5 0 0 0 4 0 0 6 0 0 5 0 0 0 4 0 0 6 0 0;
        7 0 0 0 8 0 0 0 9 7 0 0 0 0 8 0 0 0 9;
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
        0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;]
```

```

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0;
size(TEST);

[U S V] = svd(TEST);
out = r0679689_sparseModel(U, diag(S), V, TEST);
num2str(full(out), '%2.0f')
```

B.5 Opdracht 6

```

clc
A = [11 22 33 12 58 78 45 0;
     44 0 66 5 0 2 0 0;
     77 88 0 0 0 0 0 0;
     0 7 0 0 0 4 5 0];

fprintf('norm(A, "fro") = %f \n', norm(A, 'fro'))
fprintf('norm(A(:, "fro") = %f \n', norm(A(:, 'fro'))
fprintf('norm(A(:)) = %f \n', norm(A(:)))

[U, S, V] = svd(A);
num2str(S, '%10.4f')
fprintf('norm(A - U*S*V', "fro") = %f \n', norm(A - U*S*V', 'fro'))

s_result = r0679689_optimalCoefficients(U, V, A);
S2 = zeros(size(S, 1), size(S, 2));
for i = 1:length(s_result)
    S2(i, i) = s_result(i);
end
num2str(S2, '%10.4f')
fprintf('norm(A - U*S_result*V', "fro") = %f \n \n', norm(A - U*S2*V', 'fro'))

s_result = diag(r0679689_optimalCoefficients(U(:, 1:3), V(:, 1:3), A));
fprintf('norm(A - U(:, 1:3)*s_result*V(:, 1:3)', "fro") = %f \n \n', norm(A - U(:, 1:3)*s_result*V(:, 1:3)', 'fro'))

s_result = diag(r0679689_optimalCoefficients(U(:, 1:2), V(:, 1:2), A));
fprintf('norm(A - U(:, 1:2)*s_result*V(:, 1:2)', "fro") = %f \n \n', norm(A - U(:, 1:2)*s_result*V(:, 1:2)', 'fro'))

s_result = diag(r0679689_optimalCoefficients(U(:, 1), V(:, 1), A));
fprintf('norm(A - U(:, 1)*s_result*V(:, 1)', "fro") = %f \n \n', norm(A - U(:, 1)*s_result*V(:, 1)', 'fro'))

```

B.6 Opdracht 8

```
mu = r0679689_userMeans(R);
[sorted, indices] = sort(mu);

fprintf('De drie laagste gemiddelde beoordelingen zijn: \n')
fprintf('\t %.2f van gebruiker %d\n', sorted(1), indices(1))
fprintf('\t %.2f van gebruiker %d\n', sorted(2), indices(2))
fprintf('\t %.2f van gebruiker %d\n', sorted(3), indices(3))
```

```
amount = length(mu(mu == 5));
fprintf('%d gebruikers hebben een gemiddelde score van exact 5.\n', amount)
```

B.7 Opdracht 10

```
load('MovieLens20M_Subset.mat')

mu = r0679689_userMeans(R);

[I, J] = find(T);
X = sparse(I, J, mu(I));
%{
X = sparse([], [], [], m, n, length(I));
for z = 1:length(I)
    X(I(z), J(z)) = mu(I(z)) * 1;
end
%}

error = r0679689_RMSE(T, X);
fprintf('RMSE tussen T en de ijle matrix: %f \n', error)
```

B.8 Opdracht 11

```
clear
close all
load('MovieLens20M_Subset.mat')

[U, s, V, rmse] = r0679689_rank1MatrixPursuit(R, 30, T);

fig = figure;
plot(rmse);
legend('RMSE');
saveas(fig, 'ex11.png');
```

B.9 Opdracht 12

```
clear
clc
load('MovieLens20M_Subset.mat')

[U20, s20, V20, ~] = r0679689_rank1MatrixPursuit(R, 20, T);

disp(num2str(s20, '%.2f'))
```

B.10 Opdracht 15

```
clear
clc
load('MovieLens20M_Subset.mat')

n = 25;

fprintf('Processing actualBestMovies...\n')
[movieIDs, scores] = r0679689_actualBestMovies(R);

disp('average rating - movie ID - movie name - number of reviews')
topNScoresAndMovies = [num2str(scores(1:n), '%.2f'), repmat('-', n, 1), ...
    num2str(movieIDs(1:n), '%i'), repmat('-', n, 1), ...
    char(movieLabel(movieIDs(1:n))), repmat('-', n, 1), ...]
```

```

        num2str(sum(R(:, movieIDs(1:25))~=0)));
disp(topNScoresAndMovies)

fprintf('Processing rank1MatrixPursuit...\n')
[U20, s20, V20, ~] = r0679689_rank1MatrixPursuit(R, 20, T);
fprintf('Processing predictedBestMovies...\n')
[movieIDs, scores] = r0679689_predictedBestMovies(U20, s20, V20);

topNScoresAndMovies = [num2str(scores(1:n), '%.2f'), repmat(' - ', n, 1), ...
    num2str(movieIDs(1:n), '%i'), repmat(' - ', n, 1), ...
    char(movieLabel(movieIDs(1:n))), repmat(' - ', n, 1), ...
    num2str(sum(R(:, movieIDs(1:25))~=0))];
disp(topNScoresAndMovies)

```

B.11 Opdracht 18

```

clear
clc
load('MovieLens20M.Subset.mat')

[U20, s20, V20] = r0679689_rank1MatrixPursuit(R, 20, T);
C = r0679689_correlationMatrix(U20, s20, V20);

n = [ 10  5  3  3  9];
k = [ 446 3854 4355 2747 160];

for i = 1:length(n)
    IDs = r0679689_similarMovies(C, k(i), n(i));

    fprintf('%i most similar movies for %s (id %i):\n', n(i), char(movieLabel(k(i))), k(i))
    disp([repmat(' ', n(i), 1), char(movieLabel(IDs))])
    disp(' ')
end

```
