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
1.1.2 Opdracht 3 . . . 3
1.1.3 Opdracht 4 . . . 3
1.1.4 Opdracht 5 . . . 3
1.1.5 Opdracht 6 . . . 4
1.1.6 Opdracht 8 . . . 4
1.1.7 Opdracht 10 . . . 4
1.1.8 Opdracht 11 . . . 4
1.1.9 Opdracht 12 . . . 4
1.1.10 Opdracht 14 . . . 4
1.1.11 Opdracht 15 . . . 4
1.2 Clustering . . . 4
1.2.1 Opdracht 16 . . . 4
1.2.2 Opdracht 18 . . . 4
1.2.3 Opdracht 19 . . . 5

**2 Evaluatie** **5**
2.1 Opdracht 1 . . . 5
2.2 Opdracht 2 . . . 5
2.3 Opdracht 3 . . . 5

**Appendices** **6**

**A Code** **6**
A.1 Opdracht 2 . . . 6
A.2 Opdracht 4 . . . 6
A.3 Opdracht 8 . . . 6
A.4 Opdracht 10 . . . 7
A.5 Opdracht 11 . . . 7
A.6 Opdracht 12 . . . 7
A.7 Opdracht 15 . . . 7
A.8 Opdracht 18 . . . 9
A.9 r0679689_sparseModel.m . . . 9
A.10 r0679689_optimalCoefficients.m . . . 10
A.11 r0679689_userMeans . . . 10
A.12 r0679689_RMSE . . . 11
A.13 r0679689_rank1MatrixPursuit.m . . . 11
A.14 r0679689_actualBestMovies.m . . . 12
A.15 r0679689_predictedBestMovies.m . . . 12
A.16 r0679689_correlationMatrix.m . . . 13
A.17 r0679689_similarMovies.m . . . 14

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

> Hoeveel geheugenruimte is nodig om de volle beoordelingenmatrix full(R) voor te stellen?
> Hoeveel geheugenruimte is nodig om de ijle beoordelingenmatrix R voor te stellen in het coordinaatformaat?
> 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 $r = 1, ..., 500$
> Voor welke waarde van r snijden deze twee lineaire functies?

Om de volle beoordelingenmatrix 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 coordinaatformaat 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 coordinaatformaat, 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.

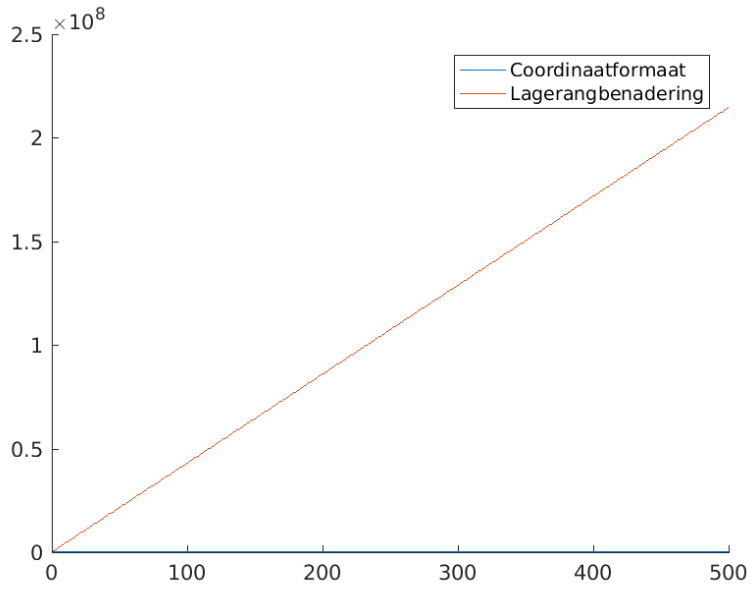


**Figure 1**

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

### 1.1.2 Opdracht 3

Bewijs dat $\sigma_j$ de $j^d e$ 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.

### 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 00033 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 enkel mogelijk als ijle matrices worden gebruikt. Bij gebruik van volle matrices is er te veel geheugenruimte nodig.
Output van de code (te vinden in A.2):

```
Totaal bruikbaar geheugen: 40GB
Nodig geheugen met volle matrices: 118.02GB
Nodig geheugen als het ijlhijdspatroon behouden kan worden: 0.66GB
```

### 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 $\zeta$ mag hoogstens $\mathcal{O}(\zeta)$ bedragen - je mag hierbij wel veronderstellen dat $max\{m, n\} \leq \zeta$. Hoe heb je deze doelstelling bereikt?

### 1.1.5 Opdracht 6

function [s] = SN_optimalCoefficients(Uk,Vk,A) De geheugencomplexiteit van je algoritme in functie van m, n, k en  mag hoogstens $\mathcal{O}(k\zeta)$ bedragen - je mag hierbij wel veronderstellen dat $max\{m,n\} \leq \zeta$. Hoe heb je deze doelstelling bereikt?

### 1.1.6 Opdracht 8

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

### 1.1.7 Opdracht 10

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

### 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 van algoritme 2. Neem de figuur op in het verslag.

### 1.1.9 Opdracht 12

Bereken [U20,s20,V20, ] = SN_rank1MatrixPursuit(R,20,T). Wat zijn de waarden van de vector s20, afgerond tot 2 cijfers na de komma?

### 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?

### 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.

## 1.2 Clustering

### 1.2.1 Opdracht 16

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?

### 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?

#### 1.2.3 Opdracht 19

Bewijs dat de correlatiematrix C in vergelijking (8) hoogstens rang r + 1 heeft wanneer de voorspelde beoordelingenmatrix A exact rang r heeft.

# 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 23 uur gespendeerd aan het oplossen van de opdrachten en ongeveer 4 uur aan het schrijven van het verslag.

## 2.2 Opdracht 2

In de loop van deze opdracht hebben we allerhande 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?

## 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?

# Appendices

## A Code

### A.1 Opdracht 2

```
close all

m = 48769;
n = 4901;

r = linspace(1, 500, 500);

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

fig = figure;
hold on;
plot(r, coordinaatSize);
plot(r, lagerangSize);
legend('Coordinaatformaat', 'Lagerangbenadering');

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

### A.2 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 matrx: 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) * 1024^3; % convert to byte

% stap 4: beste rang-1 benadering zoeken: m+n getallen voor berekening + m*n opslag
step4 = (m+n)*double + m*n*double;

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

% stap 6:
step6 = 1 * int;

totalMemoryUsage = step4 + step5 + step6;
fprintf('Totaal bruikbaar geheugen: %iGB \n', totalMemory / 1024^3)
fprintf('Nodig geheugen met volle matrices: %.2fGB \n', totalMemoryUsage / 1024^3)

step4 = (m+n)*double + 2*amountOfValues*int + amountOfValues*double;
step5 = 2*amountOfValues*int + amountOfValues*double;
step6 = 1 * int;
totalMemoryUsage = step4 + step5 + step6;
fprintf('Nodig geheugen als het ijlhijdspatroon behouden kan worden: %.2fGB \n\n', totalMemoryUsage / 1024^3)
```

### A.3 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 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];

size(TEST);

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

## A.4 Opdracht 6

```
clear
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'))
```

## A.5 Opdracht 8

```
load('MovieLens20M_Subset.mat')

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)
```

## A.6 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)
```

## A.7 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');
```

## A.8 Opdracht 12

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

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

num2str(s20, 2)
```

## A.9 Opdracht 15

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

n = 25;
```

```
profile on
fprintf('Processing actualBestMovies...\n')
[movieIDs, scores] = r0679689_actualBestMovies(R);
topNScoresAndMovies = [num2str(scores(1:n), '%.2f,'), ...
                       num2str(movieIDs(1:n), '%i, '), ...
                       char(movieLabel(movieIDs(1:n)))];
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,'), ...
                       num2str(movieIDs(1:n), '%i, '), ...
                       char(movieLabel(movieIDs(1:n)))];
disp(topNScoresAndMovies)
profile viewer

%{
OUTPUT

Processing actualBestMovies...
4.31, 938,Band of Brothers (2001)
4.29,4455,Death on the Staircase (Soupons) (2004)
4.24, 510,City of God (Cidade de Deus) (2002)
4.23,1750,Lives of Others, The (Das leben der Anderen) (2006)
4.23,2486,Dark Knight, The (2008)
4.20, 390,Spirited Away (Sen to Chihiro no kamikakushi) (2001)
4.20, 183,Amelie (Fabuleux destin d'Amlie Poulain, Le) (2001)
4.18,4418,Frozen Planet (2011)
4.18,3477,Inception (2010)
4.16, 822,Lord of the Rings: The Return of the King, The (2003)
4.15,4066,Intouchables (2011)
4.15, 194,Lord of the Rings: The Fellowship of the Ring, The (2001)
4.14,4158,Black Mirror (2011)
4.13,4227,Bleak House (2005)
4.12, 483,Lord of the Rings: The Two Towers, The (2002)
4.11, 892,Eternal Sunshine of the Spotless Mind (2004)
4.11,1932,Departed, The (2006)
4.10, 825,Fog of War: Eleven Lessons from the Life of Robert S. McNamara, The (2003)
4.10,4781,Whiplash (2014)
4.09,1227,Old Boy (2003)
4.08,4549,Louis C.K.: Oh My God (2013)
4.08,3786,Louis C.K.: Shameless (2007)
4.08,4755,Normal Heart, The (2014)
4.08, 942,Best of Youth, The (La meglio giovent) (2003)
4.07,3784,Louis C.K.: Chewed Up (2008)

Processing predictedBestMovies...
4.09, 183,Amelie (Fabuleux destin d'Amlie Poulain, Le) (2001)
4.07, 822,Lord of the Rings: The Return of the King, The (2003)
4.07,2486,Dark Knight, The (2008)
4.06, 194,Lord of the Rings: The Fellowship of the Ring, The (2001)
4.04, 892,Eternal Sunshine of the Spotless Mind (2004)
4.03, 483,Lord of the Rings: The Two Towers, The (2002)
4.00, 510,City of God (Cidade de Deus) (2002)
3.99, 390,Spirited Away (Sen to Chihiro no kamikakushi) (2001)
3.97,1932,Departed, The (2006)
3.95,3477,Inception (2010)
3.92, 154,Donnie Darko (2001)
3.91,1456,Batman Begins (2005)
3.90, 497,Pianist, The (2002)
3.90,1928,Pan's Labyrinth (Laberinto del fauno, El) (2006)
3.90, 196,Beautiful Mind, A (2001)
3.90,2586,WALLE (2008)
3.90,1955,Prestige, The (2006)
3.88, 761,Kill Bill: Vol. 1 (2003)
3.87, 625,Finding Nemo (2003)
3.87,2336,No Country for Old Men (2007)
3.86,1109,Incredibles, The (2004)
3.86, 310,Bourne Identity, The (2002)
```

```
3.86,1738,V for Vendetta (2006)
3.85, 652,Pirates of the Caribbean: The Curse of the Black Pearl (2003)
3.85,2944,Inglourious Basterds (2009)

%}
```

## A.10 Opdracht 18

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

profile on
[U20, s20, V20] = r0679689_rank1MatrixPursuit(R, 20, T);
C = r0679689_correlationMatrix(U20, s20, V20);
%[U20, s20, V20] = svds(R, 20);
%C = r0679689_correlationMatrix(U20, diag(s20), V20);
profile viewer

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
```

## A.11 r0679689_sparseModel.m

```
function [X] = r0679689_sparseModel(Uk, sk, Vk, A)
% Versie van stap 5 van het "successive rank−1 deflation" algoritme voor ijle matrices

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

%{
% SECOND SPARSE MATRIX VERSION

S = sparse(1:length(sk), 1:length(sk), sk, size(Uk, 2), size(Vk, 2));

[I, J] = find(A);
tmp = zeros(length(I), 1);
for z = 1:length(I)
    tmp(z) = Uk(I(z), :) * S * Vk(J(z), :)';
end
X = sparse(I, J, tmp);
%}
```

## A.12 r0679689_optimalCoefficients.m

```
function [s] = r0679689_optimalCoefficients(Uk, Vk, A)
% find the optimal solution for least squares problem min(|| a - Bx ||)

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

    B = sparse([], [], [], m*n, k, nnz(A) * k);
    for i = 1:k
        tmp = r0679689_sparseModel(Uk(:, i), 1, Vk(:, i), A);
        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.13 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.14 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

    %{
    % 2 seconds per call with the large matrices
    if any((A == 0) ~= (B == 0))
        full(A)
        full(B)
        error('Input matrices A and B should have the same sparsity patterns')
    end
    %}

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

## A.15 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'
    rsme 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 * 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.16 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.17 r0679689_predictedBestMovies.m

```
function [movieIDs, score] = r0679689_predictedBestMovies(Uk, sk, Vk)
% returns a list of movie IDs with their scores, list is sorted on scores in descending order
% approximate a matrix of user reviews R_k = Uk * diag(sk) * Vk'
% then calculate the average scores for the movies and sort the movies using their scores

%{
    size(Uk) = m, k
    size(sk) = k
    size(Vk) = n, k

    memory complexity constraint = O(max(m, n))

    example matrix 48769 x 4901 => SVD with k = 20: 48769x20 * 20x20 * 4901x20
    48769 * 4901 * 8 / 1024^3 = 1.78 GB
%}

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

    if (k > min(m, n))
        error('Rank k of approximation should be less than min(m, n)')
    end


    score = zeros(n, 1);
    for movie = 1:n
        approxRatingsForMovie = Uk * (sk .* Vk(movie, :)');
        % mem complexity of Uk                          = O(m * k)
        % mem complexity of Uk * diag(sk)                 = O(m * k)
        % mem complexity of Uk * diag(sk) * Vk(:, movie)' = O(m * 1)
        % => use a deeper for loop and calculate each rating per, add it
        % to an accumulator per movie and then calculate the averages => O(max(m, n))
        score(movie) = mean(approxRatingsForMovie);
    end
    [score, movieIDs] = sort(score, 'descend');

    % Rk = Uk * diag(sk) * Vk';
    % [movieIDs, score] = r0679689_actualBestMovies(Rk);
    % memory complexity: m * n > max(m, n), this will not fit in memory
end
```

## A.18 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(kn), kn < k^2 + n^2

    means = zeros(n, 1);       % O(n)
    stds = zeros(n, 1);        % O(n)
    % calculate means and standard deviations in one pass
    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_k = Uk(row_from:min(row_from+n-1, m), :) * V - means';

        C = C + (R_k' * R_k);
    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.19 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
```