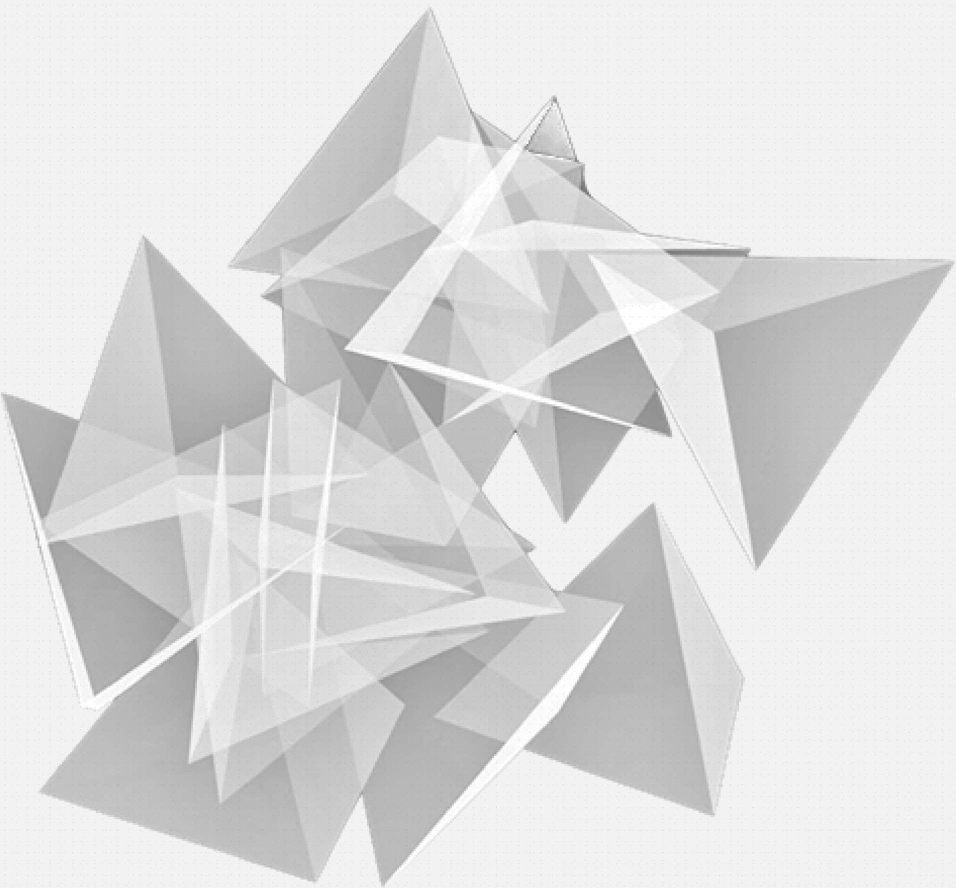
Practicum 1 - Lagerangbenaderingen

Bruno Vandekerkhove



Academisch Jaar 2019

G0Q57A: Modellering & Simulatie

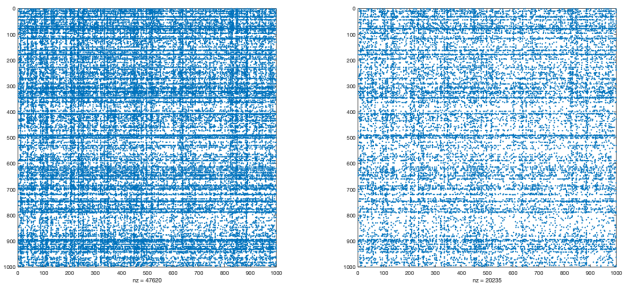
Inhoudsopgave

Een Aanbevelingssysteem voor Films

De broncode bevindt zich in de `src` folder. Het algemene script (`src/s0216676_script`) is opgedeeld in secties, één per opgave. Elke opgave wordt hieronder afzonderlijk beantwoord. Aan het einde van elk antwoord wordt (indien nodig) de broncode weergegeven.

Opdracht 1

We laden de dataset in met de `load` functie. De uitvoer staat weergegeven in figuur 1.



Figuur 1: Grafische voorstelling van ijle matrices R en T.

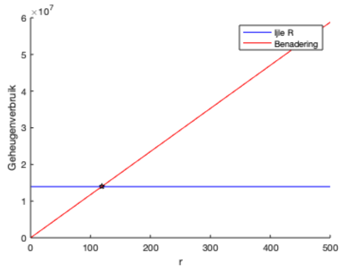
```
set(0, 'defaultFigurePosition', get(0, 'Screensize')); % Figuren vullen scherm
load('MovieLens_Subset.mat');
subplot(1,2,1)
spy(R(1:1000,1:1000))
subplot(1,2,2)
spy(T(1:1000,1:1000))
```

Opdracht 2

Stel dat gehele getallen 4 - en reële getallen 8 bytes innemen. De volle matrix `full(R)` zou dan 220388224 bytes ($\approx$ 210MB) in beslag nemen ; 1 reëel getal per element. Voor de ijle matrix `sparse(R)` lijkt het op het eerste gezicht 18525728 bytes ($<$ 18MB) te zijn ; 2 gehele getallen en 1 reëel getal per element dat niet nul is. MatLab verbruikt echter iets meer dan dat. Als men de juiste formule toepast voor het geheugenverbruik van ijle matrices [1] :

$$12 \times nnz + 4 \times n$$

dan komt men uit op bijna 14 miljoen bytes. Op mijn computer was het totaal meer dan 18,6 miljoen omdat gehele getallen daar met 8 bytes worden voorgesteld.



Figuur 2: Geheugenverbruik voor ijle matrix R en lagerangsbenadering. Het snijpunt bevindt zich in $r \approx 119$.

```
[m,n] = size(R);
ratings = nnz(R);
int_mem = 4;
double_mem = 8;
max_r = 500;
%
fprintf('Geheugenruimte full(R) : %i\n', m * n * double_mem)
%
size_sparse_naive = ratings * (int_mem * 2 + double_mem);
size_sparse = 12 * ratings + 4 * n;
fprintf('Geheugenruimte sparse(R) : %i\n', size_sparse)
%
fullR = full(R);
fprintf('Matlab zelf gebruikt respectievelijk %i en %i bytes.\n', whos('fullR').bytes, ...
    whos('R').bytes)
%
r = 1:max_r;
size_approx = (m + n) * double_mem * r;
snijpunt_r = size_sparse / ((m + n) * double_mem);
fprintf('Snijpunt in r = %i\n\n', snijpunt_r)
%
hold on
plot(r, repmat(size_sparse,1,max_r), 'b-')
plot(r, size_approx, 'r-')
plot(snijpunt_r, size_sparse, 'kp')
xlabel('r')
ylabel('Geheugenverbruik')
legend('Ijle R', 'Benadering', 'Location', 'northeast')
```

[1] `MATLAB` gebruikt het `CSC` formaat voor ijle matrices : *“Even though `MATLAB` is written in `C`, it follows its `LINPACK` and `Fortran` predecessors and stores full matrices by columns. This organization has been carried over to sparse matrices. A sparse matrix is stored as the concatenation of the sparse vectors representing its columns. Each sparse vector consists of a floating point array of nonzero entries (or two such arrays for complex matrices), together with an integer array of row indices. A second integer array gives the locations in the other arrays of the first element in each column. Consequently, the storage requirement for an $m \times n$ reals parse matrix with nnz nonzero entries is nnz reals and $nnz + n$ integers. On typical machines with 8-byte reals and 4-byte integers, this is $12nnz + 4n$ bytes.”* [1]

Opdracht 3

Men weet dat

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T \tag{1}$$

Het iteratief algoritme gaat bij elke stap σ_j, u_j en v_j bepalen zodat de Frobeniusnorm van $E_{j-1} - \sigma_j u_j v_j^T$ (de nieuwe E_j) minimaal is. Dit komt overeen met het bepalen van een afgeknotte singulierewaardenontbinding van graad 1. De σ_j dient daarbij de grootste singuliere waarde van E_{j-1} te zijn. Stel $j = 0$, dan :

$$E_1 = E_0 - \sigma_j u_j v_j^T = A - \sigma_j u_j v_j^T = \sum_{i=1}^{r} \sigma_i u_i v_i^T - \sigma_j u_j v_j^T = \sum_{i=2}^{r} \sigma_i u_i v_i^T$$

Na een aantal iteraties bekomt men uiteindelijk :

$$E_k = \sum_{i=k+1}^{r} \sigma_i u_i v_i^T$$

Want telkens wordt er een afgeknotte singulierewaardenontbinding bepaald van rang 1 en die komt overeen met de term horende bij de grootste singuliere waarde zodat deze term keer op keer verdwijnt na aftrekking. Totdat de laatste $r - k$ termen overblijven. Men kan opmerken dat dit niet verreist dat de u_j en v_j bekomen in stap 4 overeenstemmen met de u_i en v_i in (1). Het teken van u_j en v_j kan bijvoorbeeld verschillen.

Berekent men ten slotte $A - E_k$ dan bekomt men het gevraagde :

$$A - E_k = \sum_{i=1}^{r} \sigma_i u_i v_i^T - \sum_{i=k+1}^{r} \sigma_i u_i v_i^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

Het algoritme stopt na k stappen (mits men veronderstelt dat $k \leq r$, want anders stop the algoritme vroeger dan dat).

Opdracht 4

Gezien mijn studentennummer $s0216676$ is, is c gelijk aan 6. Met $sizeof(double) = 8$ zal stap 4 een totaal aan $8 * (m + n + 1)$ bytes nodig hebben. Voor stap 5 krijgt men te maken met de matrix E die maar één keer wordt gealloceerd en $8 * n * m$ bytes nodig heeft. Stap 6 verbruikt 4 bytes (voor het geheel getal). Het aantal keren dat men de stappen uitvoert maakt niet uit.

Men heeft dus te maken met $((280000 + 58000 + 1) * 8 + (280000 * 58000 * 8) + 4)/1024^3 \approx 120$ GB wat op zich al meer is dan het optimistische $8 + 2 + (c + 1)^2 = 8 + 2 + 49 = 59$ GB. Als men met ijle voorstellingen blijft werken kan hier iets aan gedaan worden.

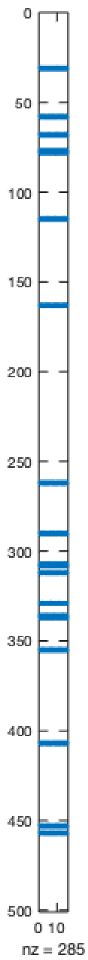
Opdracht 5

De drie vectoren `[i,j,x]` die teruggegeven worden door `find(A)` verbruiken elk $\mathcal{O}(\zeta)$ geheugen. In de implementatie van de functie `s0216676_sparseModel` wordt `[x]` zélf hergebruikt om de resultaten van de lineaire operator $P_\Omega(X)$ op te slaan. Elk element x_{ij} is gelijk aan de vermenigvuldiging van rij i van de matrix $Uk * diag(sk)$ en kolom j van Vk^T. Het volstaat dus elementsgewijs te vermenigvuldigen van rij i van Uk met sk^T en dit te vermenigvuldigen met de getransponeerde rij j van Vk.

```
function [P] = s0216676_sparseModel(Uk,sk,Vk,A)
    [i,j,x] = find(A);
    for idx = 1:nnz(A)
        x(idx) = Uk(i(idx),:) .* sk' * Vk(j(idx),:)';
    end
    P = sparse(i, j, x);
end
```

Opdracht 6

Het patroon in de bekomen figuur komt overeen met het ijlheidspatroon van R. Dit kan men gebruiken in de volgende opgave. De matrix B heeft dan al de gewenste structuur en neemt $\mathcal{O}(k\zeta)$ geheugen in.



Figuur 3: Resultaat voor opdracht 6. Men herkent het ijlheidspatroon van R.

```
B = repmat(spones(R(:)), 1, 15);
spy(B(1:500,:))
```

Opdracht 7

Door op voorhand $\mathcal{O}(k\zeta)$ aan geheugen vrij te maken en dan stap voor stap de kolommen van B aan te vullen met de resultaten van de `sparseModel` functie (die op zijn beurt $\mathcal{O}(\zeta)$ geheugen gebruikt) blijft het geheugengebruik binnen de opgelegde limiet.

```
function [s] = s0216676_optimalCoefficients(Uk,Vk,A)
    B = repmat(spones(A(:)), 1, size(Uk,2));
    for j = 1:size(Uk,2)
        B(:,j) = reshape(s0216676_sparseModel(Uk(:,j), 1, Vk(:,j), A), [], 1);
    end
    [s,¬] = lsqr(B, A(:));
end
```

Opdracht 8

Om de gemiddelde beoordeling per gebruiker te bekomen wordt de som van al diens beoordelingen genomen en gedeeld door het aantal beoordelingen dat niet nul is. Vervolgens wordt de bekomen ijle vector omgevormd tot een kolomvector.

```
function [mu] = s0216676_userMeans(A)
    mu = (sum(A, 1) ./ sum(A ≠ 0))'
end
```

Opdracht 9

Er zijn `length(mu(mu == 5))` (dwz. 13) gebruikers die een gemiddelde beoordeling hebben van 5. De laagste drie gemiddelden bedragen 0.5100, 0.5727 en 0.8983 (`sort(mu); ans(1:3)`).

Opdracht 10

Men kan de `RMSE` als volgt berekenen :

```
function [err] = s0216676_RMSE(A,B)
    err = norm(A-B, 'fro') / sqrt(nnz(A));
end
```

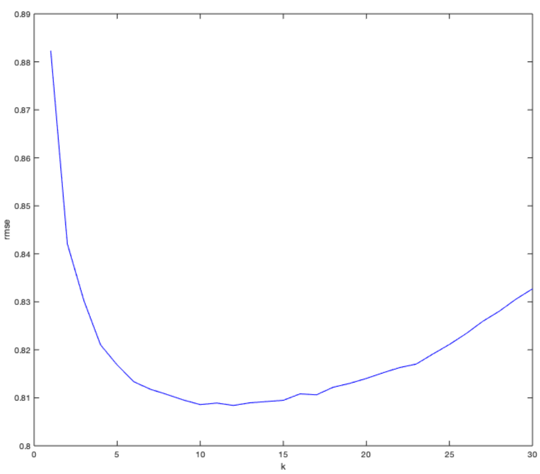
Opdracht 11

De `RMSE` bedraagt iets meer dan 26 :

```
[i,j] = find(T);
mu = s0216676_userMeans(R);
err = round(s0216676_RMSE(T, sparse(i,j,mu(j))), 4)
% Prints "err = 0.8823"
```

Opdracht 12

Figuur 4 geeft de fout weer in functie van het aantal iteraties. De volle berekening duurde ongeveer 250 seconden. De reden waarom er divergentie optreedt is dat men te maken krijgt met *overfitting* (het model veralgemeent niet goed meer wanneer het de oorspronkelijke data 'te goed' benadert).



Figuur 4: Resultaat van het algoritme voor $k = 30$, toegepast op de MovieLens dataset.

Opdracht 13

De vector s_{20} ziet er als volgt uit :

[1.0 806.2 386.4 383.2 284.9 248.9 192.7 199.2 179.9 171.3 133.1 152.1 136.0 143.8 122.0 120.5 128.8 116.9 120.2 124.7]

Opdracht 14

De functie kan in één lijn geschreven worden :

```
function [movieIDs,score] = s0216676_actualBestMovies(R)
    [score,movieIDs] = sort(sum(R,2) ./ sum(R≠0,2), 'descend');
end
```

Opdracht 15

Deze keer moet de berekening iteratief gebeuren om het gevraagde maximum aan geheugencomplexiteit te respecteren. De scores worden geaccumuleerd, dan wordt hun gemiddelde genomen en ten slotte worden ze gesorteerd.

```
function [movieIDs,score] = s0216676_predictedBestMovies(Uk,sk,Vk)
    [m,k] = size(Uk);
    score = zeros(m, 1);
    for i = 1:k
        score = score + Uk(:,i) * sk(i) * sum(Vk(:,i));
    end
    [score, movieIDs] = sort(score / size(Vk,1), 'descend');
end
```

Opdracht 16

De 'beste films' worden weergegeven in tabel 1 en 2. De mediaan voor het aantal beoordelingen bedraagt respectievelijk 213 en 4412. Dus niet alleen ziet de lijst van voorspelde beste films er heel wat realistischer uit, de films hebben ook heel wat meer beoordelingen. Dat maakt de lijst betrouwbaarder.

Opdracht 17

De functie werd als volgt geïmplementeerd :

```
function [movieIDs,score] = s0216676_predictedBestMoviesForUser(R,Uk,sk,Vk,j)
    [m,¬] = size(R);
    indices = setdiff(1:m,find(R))';
    n = length(indices);
    score = zeros(n,1);
    x = sk .* Vk(j,:)';
    for i = 1:n
       score(i) = Uk(indices(i),:) * x;
    end
    [score,permutation] = sort(score, 'descend');
    movieIDs = indices(permutation);
end
```

Feitelijke beste films	**# beoordelingen**
Planet Earth II (2016)	387
Black Mirror: White Christmas (2014)	594
Won't You Be My Neighbor? (2018)	57
Sherlock - A Study in Pink (2010)	131
Blue Planet II (2017)	131
Over the Garden Wall (2013)	218
Whiplash (2013)	357
Human Planet (2011)	177
Inception (2010)	7631
The Night Of (2016)	184
The Jinx: The Life and Deaths of Robert Durst (2015)	401
Making a Murderer (2015)	113
Piper (2016)	631
Frozen Planet (2011)	198
Whiplash (2014)	3535
Bo Burnham: what. (2013)	95
The Handmaiden (2016)	544
Your Name. (2016)	605
Story of Film: An Odyssey, The (2011)	59
Laurence Anyways (2012)	77
Intouchables (2011)	3331
Spotlight (2015)	2264
Winter on Fire: Ukraine's Fight for Freedom (2015)	54
John Mulaney: New In Town (2012)	155
O.J.: Made in America (2016)	213
Totaal	22142
Mediaan	213

Tabel 1: Resultaten van de oproep van functie `s0216676_actualBestMovies`.

Voorspelde beste films	**# beoordelingen**
Inception (2010)	7631
Whiplash (2014)	3535
Intouchables (2011)	3331
Interstellar (2014)	6181
Django Unchained (2012)	5851
The Martian (2015)	5268
Arrival (2016)	3508
Gone Girl (2014)	4337
Shutter Island (2010)	5505
Grand Budapest Hotel, The (2014)	4444
Spotlight (2015)	2264
Dark Knight Rises, The (2012)	6182
King's Speech, The (2010)	4225
Ex Machina (2015)	4412
The Imitation Game (2014)	4614
Big Short, The (2015)	2886
Edge of Tomorrow (2014)	4807
Inside Out (2015)	4549
Prisoners (2013)	2360
Guardians of the Galaxy (2014)	5700
Her (2013)	3856
Nightcrawler (2014)	3139
Room (2015)	1674
Wolf of Wall Street, The (2013)	5141
Dallas Buyers Club (2013)	2747
Totaal	108147
Mediaan	4412

Tabel 2: Resultaten van de oproep van functie `s0216676_predictedBestMovies`.

Opdracht 18

De resultaten worden weergegeven in tabel 3, 4 en 5.

Beste films	Score
Deadpool (2016)	5.7072
The Hunger Games (2012)	4.9250
Star Wars: Episode VII - The Force Awakens (2015)	4.8234
Her (2013)	4.6669
The Hunger Games: Mockingjay - Part 1 (2014)	4.5391
Black Swan (2010)	4.4652
Perks of Being a Wallflower, The (2012)	4.1907
Scott Pilgrim vs. the World (2010)	4.1496
Avengers: Age of Ultron (2015)	4.1399
Moonrise Kingdom (2012)	4.0860

Tabel 3: Best beoordeelde films voor gebruiker 98 (op basis van voorspellingen).

Beste films	Score
Captain America: The Winter Soldier (2014)	5.4375
Deadpool (2016)	5.4323
Intouchables (2011)	5.3363
Thor: Ragnarok (2017)	5.3028
Avengers: Infinity War - Part I (2018)	5.2873
Doctor Strange (2016)	5.2848
Big Hero 6 (2014)	5.2218
Logan (2017)	5.1846
John Wick (2014)	5.0464
Spotlight (2015)	4.9661

Tabel 4: Best beoordeelde films voor gebruiker 10100 (op basis van voorspellingen).

Beste films	Score
How to Train Your Dragon (2010)	5.0000
The Hunger Games (2012)	5.0000
Dark Knight Rises, The (2012)	5.0000
Pitch Perfect (2012)	5.0000
Hobbit: An Unexpected Journey, The (2012)	5.0000
Way, Way Back, The (2013)	5.0000
Short Term 12 (2013)	5.0000
Thor: The Dark World (2013)	5.0000
Wolf of Wall Street, The (2013)	5.0000
How to Train Your Dragon 2 (2014)	5.0000
The Hunger Games: Mockingjay - Part 1 (2014)	5.0000
Deadpool (2016)	5.0000
Guardians of the Galaxy 2 (2017)	5.0000
Sherlock: The Abominable Bride (2016)	5.0000
Kubo and the Two Strings (2016)	5.0000

Tabel 5: Best beoordeelde films voor gebruiker 98 (op basis van testmatrix T).

Voor gebruiker 10100 worden voornamelijk Marvel films aangeraden. Voor gebruiker 98 komen 3 aanbevelingen overeen met diens hoogst-beoordeelde films (in matrix T, niet in matrix R). De andere films hebben een gelijkaardig genre, dus is wat aangeraden wordt best oké.

Evaluatie

Ik spendeerde gemiddeld zo'n uur per opgave (de laatste gingen wat sneller dan de eerste) voor een totaal aan ongeveer 18 uur. Het schrijven van het verslag nam ongeveer 3 uur in beslag. Het practicum leek me van gepaste moeilijkheid en best leerrijk.

Het aanbevelingssysteem gaat uit van lineaire verbanden. Dit is een simplistische assumptie. Moderne aanpakken zoals neurale netwerken kunnen complexere verbanden aan, wat mogelijks tot betere resultaten zou kunnen leiden. Het lijkt op het eerste gezicht ook moeilijk om *on-line* te werken (naarmate de dataset evolueert zou men om de zo veel tijd het algoritme opnieuw moeten uitvoeren), hoewel er algoritmes bestaan die dit aankaarten.

Referenties

[1] John R. Gilbert, Cleve Moler, and Robert Schreiber. Sparse matrices in MATLAB: Design and implementation. *SIAM Journal on Matrix Analysis and Applications*, 13(1):333–356, January 1992.