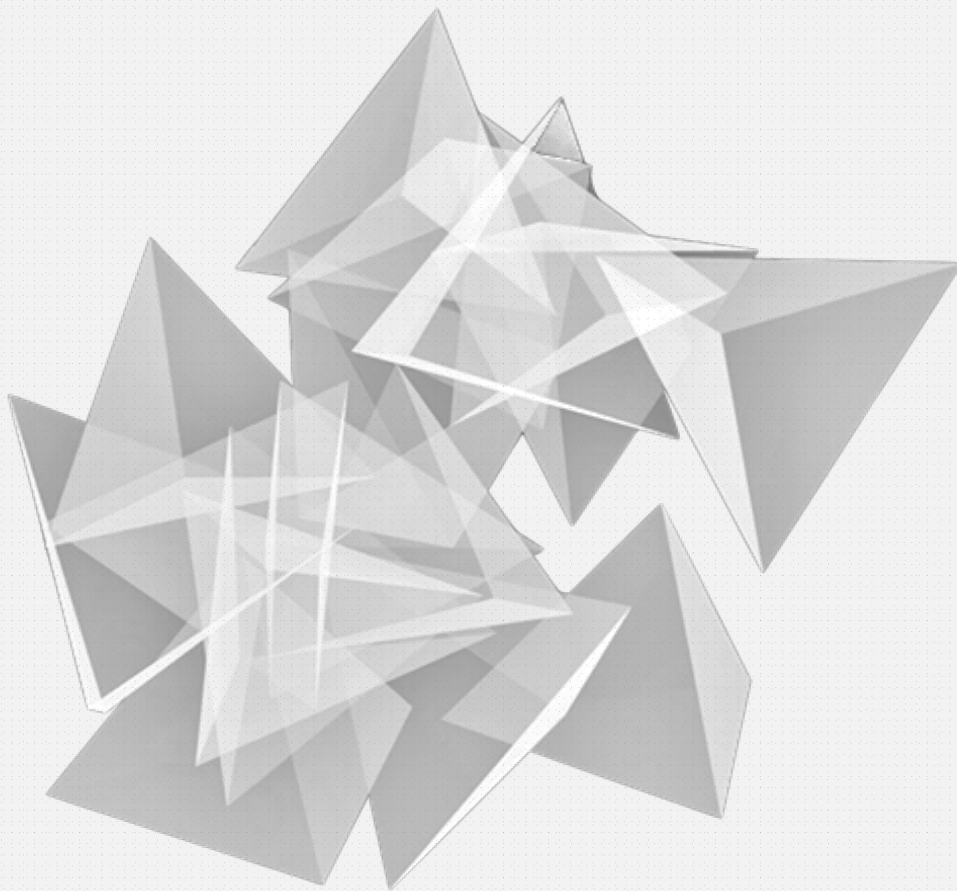


# Practicum 1 - Lagerangbenaderingen

Bruno Vandekerkhove



ACADEMISCH JAAR 2019

G0Q57A: MODELLERING & SIMULATIE

# Inhoudsopgave

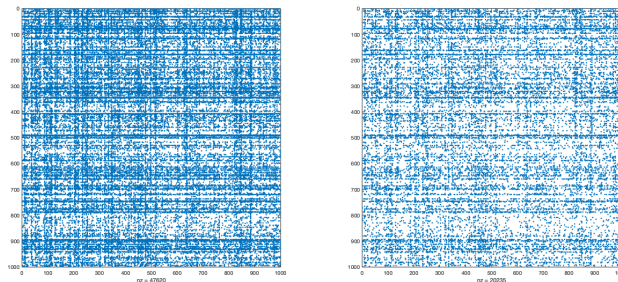
<b>Een Aanbevelingssysteem voor Films</b>	<b>1</b>
Opdracht 1 . . . . .	1
Opdracht 2 . . . . .	1
Opdracht 3 . . . . .	2
Opdracht 4 . . . . .	3
Opdracht 5 . . . . .	3
Opdracht 6 . . . . .	3
Opdracht 7 . . . . .	3
Opdracht 8 . . . . .	3
<b>Bronnen</b>	<b>3</b>
<b>Evaluatie</b>	<b>4</b>

## 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. De afzonderlijke opgaven worden hieronder 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  $T$ .

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

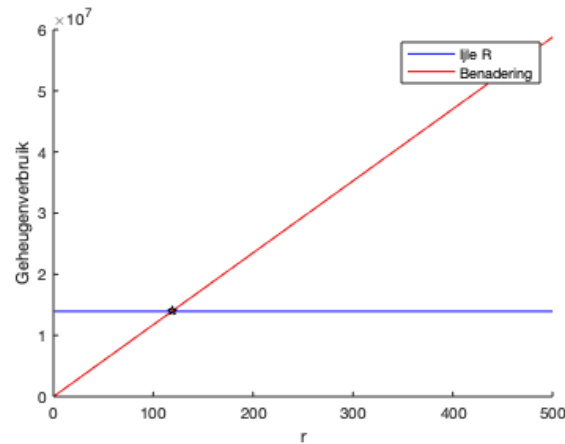
### Opdracht 2

Stel dat gehele getallen 4 - en reële getallen 8 bytes innemen. De volle matrix **full(R)** zou dan 220388224 bytes ( $\approx 210\text{MB}$ ) in beslag nemen ; 1 reël getal per element. Voor de ijle matrix **sparse(R)** lijkt het op het eerste gezicht 18525728 bytes ( $< 18\text{MB}$ ) te zijn ; 2 gehele getallen en 1 reël 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 <sup>1</sup> :

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

<sup>1</sup>**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

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  $r \approx 119$ .

```

1  [m,n] = size(R);
2  ratings = nnz(R);
3  int_mem = 4;
4  double_mem = 8;
5  max_r = 500;
6  %
7  fprintf('Geheugenruimte full(R) : %i\n', m * n * double_mem)
8  %
9  size_sparse_naive = ratings * (int_mem * 2 + double_mem);
10 size_sparse = 12 * ratings + 4 * n;
11 fprintf('Geheugenruimte sparse(R) : %i\n', size_sparse)
12 %
13 fullR = full(R);
14 fprintf('Matlab zelf gebruikt respectievelijk %i en %i bytes.\n', whos('fullR').bytes, ...
        whos('R').bytes)
15 %
16 r = 1:max_r;
17 size_approx = (m + n) * double_mem * r;
18 snijpunt_r = size_sparse / ((m + n) * double_mem);
19 fprintf('Snijpunt in r = %i\n\n', snijpunt_r)
20 %
21 hold on
22 plot(r, repmat(size_sparse,1,max_r), 'b-')
23 plot(r, size_approx, 'r-')
24 plot(snijpunt_r, size_sparse, 'kp')
25 xlabel('r')
26 ylabel('Geheugenverbruik')
27 legend('Ijle R', 'Benadering', 'Location', 'northeast')

```

### Opdracht 3

Men weet dat  $A = \sum_{i=1}^r \sigma_i u_i v_i^T$ . Het iteratief algoritme gaat bij elke stap  $\sigma_j$ ,  $u_j$  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 afgeknotten singulierewaardenontbinding van graad 1. De  $\sigma_j$  dient daarbij de grootste singuliere waarde van  $E_{j-1}$  te zijn. Stel  $j = 0$ , dan :

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

---

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]

Na een aantal iteraties bekomt men uiteindelijk :

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

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

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

#### Opdracht 4

Gezien mijn studentnummer s0216676 is, is  $c$  gelijk aan 6. Met  $\text{sizeof}(\text{double}) = 8$  zal stap 4 een totaal aan  $8 * (m + n + 1) + 8 * m * n + 4$  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 matrices blijft werken kan hier iets aan gedaan worden.

#### Opdracht 5

#### Opdracht 6

#### Opdracht 7

#### Opdracht 8

#### 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.

Evaluatie