Treball 1: Matrius disperses

Ismael El Habri, Marc Cané, Lluís Trilla

16 d'octubre de 2018

Índex

1 Què són les matrius disperses? 4

2 Formes d'emmagatzemar matrius disperses 5

2.1 Per Coordenades 5

2.1.1 Exemple 5

2.2 Per files 6

2.2.1 Exemple 6

2.2.2 Implementació del mètode CSR 6

2.2.2.1 Constructor 7

2.2.2.2 Mètodes per obtenir una fila, una columna i un element 8

2.3 Per perfil 9

2.3.1 Matrius banda 10

2.3.2 El mètode 10

2.3.3 Exemple 10

2.4 Altres mètodes 10

2.4.1 Diccionari de claus 10

2.4.2 Llista de llistes 11

2.4.3 Esquema DIA 11

3 Operacions amb matrius disperses emmagatzemades per files 12

3.1 Mètode per afegir una fila a una matriu dispersa 12

3.2 Mètode per sumar dues matrius disperses 13

3.3 Mètodes pel producte matriu-vector 13
3.3.1 Producte d'un vector per una matriu 13
3.3.2 Producte d'una matriu per un vector 14
3.4 Mètode per multiplicar dues matrius 15

4 Resolució de sistemes. Reordenació de les matrius. 17

4.1 Introducció 17
4.2 Resolució de grans sistemes d'equacions lineals 17

Capítol 1

Què són les matrius disperses?

Quan parlem de matrius disperses ens referim a matrius de gran tamany en la qual la majoria d'elements son zero. Direm que una matriu és disperrsa, quan hi hagi benefici en aplicar els mètodes propis d'aquestes.

Per identificar si una matriu és dispersa, podem usar el seguent:

> Una matriu $n \times n$ serà dispersa si el número de coeficients no nuls es $n^{\gamma+1}$, on $\gamma < 1$.

En funció del poblema, decidim el valor del paràmetre γ. Aquí hi ha els valors típics de γ:

- $\gamma = 0.2$ per problemes d'anàlisi de sistemes eléctics degeneració i de transpot d'enegía.
- $\gamma = 0.5$ per matrius en bandes associades a problemes d'anàlisi d'estructues.

Podem trobar dos tipus de matrius disperses:

- **Matrius estructurades:** matrius en les quals els elements diferents de zero formen un patró regular. Exemple: Les matrius banda.
- **Matrius no estructurades:** els elements diferents de zero es distribueixen de forma irregular.

Capítol 2

Formes d'emmagatzemar matrius disperses

Per Coordenades

És la primera aproximació que podríem pensar i és bastant intuïtiva. Per cada element no nul guardem una tupla amb el valor i les seves coordenades: (a_{ij}, i, j).

Exemple

$$\begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & -2 & 0 \end{pmatrix} =$$

índex	tupla(a_{ij}, i, j)
0	$(1, 1, 1)$
1	$(2, 1, 4)$
2	$(1, 2, 2)$
3	$(3, 4, 1)$
4	$(-2, 4, 3)$

Per emmagatzemar això podem usar tres vectors de la mateixa mida (n_z, el nombre d'elements diferents de zero): Un amb els valors, un amb les files i un amb les columnes:

Vector	Coeficients				
valors	1	2	1	3	-2
files	1	1	2	4	4
columnes	1	4	2	1	3

A la realitat però, aquest mètode d'emmagatzemar les dades és poc eficient quan hem de fer operacions amb les matrius.

Per files

També conegut com a *Compressed Sparse Rows (CSR)*, *Compressed Row Storage (CRS)*, o format *Yale*. És el mètode més estès.

Consisteix en guardar els elements ordenats per files, guardar la columna on es troben, i la posició del primer element de cada fila en el vector de valors. Així ens quedaran tres vectors:

- **valors:** de mida n_z, conté tots els valors diferents.
- **columnes:** també de mida n_z, conté la columna on es troba cada un dels elements anteriors.
- **iniFiles:** de mida $m+1$, conté la posició on comença cada fila en els vectors valors i columnes, sent m el nombre de files de la matriu.

Exemple

$$\begin{pmatrix} 1 & 0 & 0 & 2 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 3 & 0 & -2 & 0 \end{pmatrix} \quad = \quad$$

Vector	Coeficients				
índex	1	2	3	4	5
valors	1	2	1	3	-2
columnes	1	4	2	1	3
iniFiles	1	3	4	4	6

Si es canvien files per columnes, dona la implementació per columnes, o també anomenada *Compressed Sparse Columns (CSC)*.

Implementació del mètode CSR

Hem implementat un script Matlab amb una classe `CSRSparseMatrix` que guardi les dades necessàries. Aquestes les tenim en "l'atribut" `Matrix` dins del bloc `properties` (línia 10 del codi següent). Aquestes dades consisteixen en el següent:

- `Matrix.nColumns`: número de columnes de la matriu, necessari per recrear les files posteriorment.
- `Matrix.values`: vector valors comentat anteriorment, amb els valors no nuls de la matriu.
- `Matrix.columns`: vector de columnes, amb la columna corresponent a cada valor amb el mateix índex.
- `Matrix.beginningRow`: vector amb els índex comença cada fila en el vector de valors i de columnes.

Script Matlab

```
%===============CSRSparseMatrix=====================================================
% Classe que implementa el mètode d'emmagatzematge per files sobre matrius
%%% disperses
%
% El constructor rep una matriu i hi aplica el metode, tornant un objecte
%%% de tipus CSRSparseMatrix.
%
classdef CSRSparseMatrix
  properties
    Matrix
    %%% La matriu (estructura de quatre elements:
    %%%                 n: Matrix.nColumns,
    %%%                 valors: Matrix.values,
    %%%                 columnes: Matrix.columns,
    %%%                 inici files: Matrix.beginningRow)
  end

  methods
  %
  %%% MÈTODES
  %
  end
end
```

Constructor

Entenent com s'emmagatzema la matriu, fer el constructor de la classe és trivial: només és necessari recórrer la matriu per files i guardar al vector de valors els elements diferents de 0, guardar-nos la columna on es troba cada un d'aquests elements en el vector de columnes. A part, per cada fila, hem de actualitzar el vector d'índexs de files. Per fer-ho seguim el següent:

- `Matrix.beginningRow[1] = 0`.
- `Matrix.beginningRow[i] = Matrix.beginningRow[i-1]` + elements diferents de zero en la fila i.

Script Matlab

```
function obj = CSRSparseMatrix(A)
    [m,n] = size(A);
    obj.Matrix.nColumns = n;
    obj.Matrix.values = [];
```

```
5       obj.Matrix.columns = [];
6       obj.Matrix.beginningRow = [1];
7       for i = 1:m
8           nonZero=obj.Matrix.beginningRow(i);
9           nonZeroThisRow=0;
10          for j = 1:n
11              if(A(i,j) ~= 0)
12              obj.Matrix.values = [obj.Matrix.values, A(i,j)];
13              obj.Matrix.columns = [obj.Matrix.columns, j];
14              nonZeroThisRow=nonZeroThisRow+1;
15              end
16          end
17          obj.Matrix.beginningRow = [obj.Matrix.beginningRow, nonZero+nonZeroThisRow];
18      end
19  end
```

Mètodes per obtenir una fila, una columna i un element

El mètode per obtenir una fila es troba a partir de la línia 6 del codi posterior. Com podem veure consisteix en primer crear una fila amb `Matrix.nColumns` zeros. Llavors, sabent que `Matrix.beginningRow[i]` ens indica on comença la fila i en el vector de valors i en el de columnes, i deduint que acaben a `Matrix.beginningRow[i+1]-1`; podem delimitar els elements de la fila. Només quedaria afegir aquests elements a les seves posicions corresponents (usant el vector de columnes).

El mètode per obtenir un element es troba a partir de la línia 22 del codi posterior. Aquest mètode, usa el que hem vist anteriorment per crear la fila determinada, llavors només cal obtenir l'element amb la columna.

El mètode per obtenir una columna es troba a partir de la línia 32 del codi posterior. Sabent quantes files tenim (recordem que `Matrix.beginningRow` té $m+1$ elements, on m és el nombre de files de la matriu), inicialitzem cada element a 0 i busquem els elements de cada posició de la columna, usant el mateix mètode usat per obtenir les files. Cal tenir en compte, que per cada fila, les columnes estan ordenades, així que no cal buscar més quan ja hem passat la columna que estem mirant.

Script Matlab

```
1   %============getRow===============================================================
2   %
3   %%% Donats obj, sent el objecte actual, i x la fila que volem obtenir;
4   %%%% retorna la fila x de la matriu obj
5   %
6   function row = getRow(obj, x)
7       row = 0;
8       for i= 1:obj.Matrix.nColumns
9           row(i)=0;
```

```
    end
    for i = obj.Matrix.beginningRow(x):obj.Matrix.beginningRow(x+1)-1
        row(obj.Matrix.columns(i)) = obj.Matrix.values(i);
    end
end

%============getElem=================================================================
%
%%% Donats obj, sent el objecte actual i (x,y) les coordenades del element
%%%% que volem obtenir;
%%%% retorna l'element x,y de la matriu obj
%
function elem = getElem(obj, x, y)
    row = obj.getRow(x);
    elem = row(y);
end

%============getColumn================================================================
%
%%% Donats obj, sent el objecte actual, i y la columna que volem obtenir;
%%%% retorna la columna y de la matriu obj
%
function column = getColumn(obj, y)
    column = 0;
    m=size(obj.Matrix.beginningRow,2)-1;
    for j=1:m
        column(j)=0;
        for i=obj.Matrix.beginningRow(j):obj.Matrix.beginningRow(j+1)-1
            if obj.Matrix.columns(i)==y
                column(j) = obj.Matrix.values(i);
                break
            elseif obj.Matrix.columns(i)> y
                break
            end
        end
    end
end
```

Per perfil

Aquest mètode és una manera eficient de guardar un tipus concret de matrius, les matrius banda.

Matrius banda

Com vam veure a classe, una matriu $n \times n$ és banda si existeixen p i q naturals, tals que $1 < p, q < n$ i $a_{ij} = 0$ sempre que $p \leq j - i$ o $q \leq i - j$.

Anomenem ample de banda de la matriu a $p + q - 1$.

L'**envoltant** de una matriu banda consisteix en tots els elements de cada fila des de el primer no nul fins al ultim no nul, incloent els elements nuls que hi pugui haver entre mig. Aquí la definició formal:

L'envoltant de una matriu banda A, $env(A)$ es defineix com el conjunt $env(A) = \{i, j\} : f_i \leq j \leq l_i, 1 \leq i \leq n$ on f_i és on comencen els valors no nuls de la fila i, i l_i on acaben els valors no nuls de la fila i.

El mètode

Consisteix en guardar els elements de l'envoltant de la matriu, la columna on comença l'envoltant en cada fila, i la posició en el vector de valors on comença cada fila.

Doncs, ens quedarien tres vectors:

- **valors:** amb els valors de l'envoltant.
- **columnInici:** amb el valor f_i de cada fila (la columna on comença l'envoltant en cada fila.
- **iniFiles:** amb la posició on comença cada fila en el vector valors.

Exemple

$$\begin{pmatrix} 1 & 2 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 0 & -2 \end{pmatrix} \quad = \quad$$

Vector	Coeficients						
índex	1	2	3	4	5	6	7
valors	1	2	1	2	0	2	−2
columnInici	1	2	2	4			
iniFiles	1	3	4	7			

Altres mètodes

Diccionari de claus

Conegut també com a *Dictionary of Keys (DOK)*, consisteix en tenir un diccionari que mapeja parelles de (fila, columna) amb el valor de cada element. Els elements que no estan en el diccionari es poden considerar zero. Aquest mètode es bo per construir la matriu de manera incremental en un ordre aleatori, però es dolent al iterar pels elements diferents de zero en un ordre lexicogràfic. El seu ús més habitual es usar aquest format per construir la matriu, per després convertir-la en un format més eficient de processar.

Llista de llistes

Guarda una llista per fila, en la qual cada entrada és una parella de valors (valor, columna). S'acostumen a ordenar per número de columna per motius d'eficiència. Aquest mètode també es bo per construir la matriu de forma incremental.

Esquema DIA

Aquest esquema s'usa quan els valors no nuls estan restringits a un reduït nombre de diagonals. Consisteix en guardar una matriu de dades que conté els valors no nuls i un vector amb els *offsets*, que guarda el desplaçament de cada diagonal respecte la diagonal principal.
A la diagonal principal li correspon l'*offset* 0. A les diagonals superiors, els hi assignem valors positius; a les inferiors, valors negatius.

Exemple

Donada la matriu:

$$\begin{pmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{pmatrix}$$

Necessitaríem guardar el següent:

$$dat = \begin{pmatrix} * & 1 & 7 \\ * & 2 & 8 \\ 5 & 3 & 9 \\ 6 & 4 & * \end{pmatrix}, off = (-2, 0, 1)$$

Capítol 3

Operacions amb matrius disperses emmagatzemades per files

Mètode per afegir una fila a una matriu dispersa

Aquest mètode està pensat per construir les matrius de forma incremental, cosa que ens serà molt útil per fer les operacions posteriors.

El mètode en sí consisteix en fer exactament el mateix que fa el constructor per cada fila, per la donada per paràmetre en el mètode.

Script Matlab

```
%============addRow================================================================
%
%%% Donats obj, sent el objecte actual, i row la fila que volem afegir;
%%%% retorna obj amb la fila nova afegida
%
function obj = addRow(obj, row)
  i=size(obj.Matrix.beginningRow,2)
  nonZero=obj.Matrix.beginningRow(i)
  nonZeroThisRow=0;
  [_,n] = size(row);
  for j = 1:n
    if(row(j) ~= 0)
      obj.Matrix.values = [obj.Matrix.values, row(j)]
      obj.Matrix.columns = [obj.Matrix.columns, j]
      nonZeroThisRow=nonZeroThisRow+1
    end
  end
  obj.Matrix.beginningRow = [obj.Matrix.beginningRow, nonZero+nonZeroThisRow];
```

```
19 end
```

Mètode per sumar dues matrius disperses

Per sumar dues matrius disperses sense carregar tota la matriu en memòria sumem fila per fila i ho afegim a una nova matriu dispersa resultat.

Script Matlab

```
1  %============sum=============================================================
2  %
3  %%% Donats obj i una altre matriu dispersa de la mateixa mida;
4  %%%% retorna la suma de les dues matrius
5  %
6  function res = sum(obj, B)
7    nRowsA = length(obj.Matrix.beginningRow)-1;
8    nRowsB = length(B.Matrix.beginningRow)-1;
9
10   assert(obj.Matrix.nColumns == B.Matrix.nColumns && nRowsA == nRowsB)
11
12   res = CSRSparseMatrix([]);
13   res.Matrix.nColumns = obj.Matrix.nColumns;
14   for i=1:nRowsB
15     rowA = obj.getRow(i);
16     rowB = B.getRow(i);
17     res = res.addRow(rowA+rowB);
18   end
19 end
```

Mètodes pel producte matriu-vector

Aquest apartat es dividirà en dos mètodes:

- El producte d'un vector per una matriu.
- El producte d'una matriu per un vector.

Producte d'un vector per una matriu

Consisteix en la multiplicació $c = b \times A$, on A és una matriu $m \times n$, b és un vector de mida m i c és un vector de mida n.

El resultat d'aquesta operació, es pot representar així, pel cas que $m = 2$ i $n = 3$:

$$\begin{bmatrix} c_1 & c_2 & c_3 \end{bmatrix} = \begin{bmatrix} b_1 & b_2 \end{bmatrix} \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \quad \longrightarrow \quad \begin{array}{l} c_1 = b_1 a_{11} + b_2 a_{21} \\ c_2 = b_1 a_{12} + b_2 a_{22} \\ c_3 = b_1 a_{13} + b_2 a_{23} \end{array}$$

Això es podria generalitzar de la següent manera:

$$c_j = \sum_{i=1}^{m} b_i a_{ij}$$

Amb aquesta formula, la implementació es simplifica bastant: nomes cal aplicar-la per a cada columna j, quedant un doble bucle, com es pot veure en el codi a continuació:

Script Matlab

```
%============multRow=============================================================
%
%%% Donats obj i un vector de la mateixa mida que les columnes de obj
%%%% retorna el resultat de la multiplicació b*obj
%
function res = multRow(obj,b)
  res = zeros(1, length(obj.Matrix.beginningRow)-1);
  assert(obj.Matrix.nColumns == length(b));
  for i=1:obj.Matrix.nColumns
    bi = b(i);
    for ii = obj.Matrix.beginningRow(i):obj.Matrix.beginningRow(i+1)-1
      j = obj.Matrix.columns(ii);
      res(j) = res(j) + obj.Matrix.values(ii)*bi;
    end
  end
end
```

Producte d'una matriu per un vector

Consisteix en la multiplicació $c = A \times b$, on A és una matriu $m \times n$, b és un vector de mida n i c és un vector de mida m.

En aquest cas el que fem és cada fila i de la matriu, i la multipliquem per b. El resultat el guardem a c_i.

Script Matlab

```
 %============multColumn=============================================================
%
%%% Donats obj i un vector de la mateixa mida que les files de obj
%%%% retorna el resultat de la multiplicació obj*b
%
function res = multColumn(obj, b)
  res=zeros(1, length(obj.Matrix.nColumns));
  m = length(obj.Matrix.beginningRow)-1;
  assert ( m == length(b));
  for i = 1:m
    row = obj.getRow(i);
    res(i)= row * b;
  end
  res = transpose(res);
end
```

Mètode per multiplicar dues matrius

Consisteix en la multiplicació de $C = A \times B$, on A és una matriu $m \times n$ i B és una matriu $n \times o$, resultant en la matriu C que és de mida $m \times o$.

Aquesta operació la podem representar simbolicament de la següent manera per a $m = 2, n = 3, o = 2$:

$$\begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{pmatrix} \times \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} \quad \longrightarrow \quad \begin{array}{l} c_{11} = a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31} \\ c_{12} = a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32} \\ c_{21} = a_{21}b_{11} + a_{22}b_{21} + a_{23}b_{31} \\ c_{22} = a_{21}b_{12} + a_{22}b_{22} + a_{23}b_{32} \end{array}$$

Amb això podem veure com hi ha un patró semblant al vist en la multiplicació vector per matriu. Cada fila i de C és igual a la multiplicació de la fila i de A per la matriu B. Per tant podem fer que en la implementació, per cada fila i de la matriu C, obtenir la fila i de A i fer $A_i \times B$ usant el mètode implementat anteriorment.

Script Matlab

```
%============mtimes=================================================================
%
%%% Donats obj de mida m*n i una matriu dispersa B de mida n*o,
%%%% retorna el resultat de la multiplicació obj*B de mida m*o
%
function res = mtimes(obj, B)
```

```
  nRowsA = length(obj.Matrix.beginningRow)-1;
  nRowsB = length(B.Matrix.beginningRow)-1;
  assert(obj.Matrix.nColumns == nRowsB)
  res = CSRSparseMatrix([]);
  res.Matrix.nColumns = B.Matrix.nColumns;

  for i=1:nRowsB
    row = obj.getRow(i);
    res = res.addRow(B.multRow(row));
  end
end
```

Capítol 4

Resolució de sistemes. Reordenació de les matrius.

Introducció

Per resoldre un sistema d'equacions representat amb una matriu dispersa podriem usar qualsevol dels mètodes que coneixem per resoldre matrius, però ens trobariem fent un nombre molt gran d'operacions innecessàries i gastant molta més memòria de la necessària.

Suposem que apliquem el mètode de Gauss en una matriu dispersa, podem veure marcats en vermell tots els elements no nuls que hem introduit al aplicar Gauss (elements *fill-in*):

$$\begin{pmatrix} x & 0 & x & 0 & 0 \\ 0 & x & 0 & 0 & x \\ x & 0 & x & 0 & 0 \\ 0 & 0 & 0 & x & 0 \\ 0 & x & 0 & 0 & x \end{pmatrix} \xrightarrow{\text{Gauss}} \begin{pmatrix} x & 0 & x & 0 & 0 \\ 0 & x & \color{red}{x} & 0 & x \\ x & \color{red}{x} & x & 0 & \color{red}{x} \\ 0 & 0 & 0 & x & \color{red}{x} \\ 0 & x & \color{red}{x} & \color{red}{x} & x \end{pmatrix}$$

Això ens suposaria un increment del ús de memòria i ens obligaria a incrementar molt el número d'operacions, augmentant la complexitat numèrica. Així doncs, ens interessa reordenar files i columnes per tal de reduir o fins i tot desfer-nos dels elements de *fill-in*.

Resolució de grans sistemes d'equacions lineals

Per resoldre qualsevol sistema d'equacions representat amb una matriu dispersa, seguirem els següents passos:

1. Determinar l'estructura símbolica de A.

2. Obtenir les permutacions P i Q, tals que PAQ tingui una estructura de densitat òptima(no genera elements de *fill-in*).
3. Fer factorització LU: $LU = PAQ$ i $c = Pb$.
4. Resoldre $Lz = c, Uy = z$ i $x = Qy$.

Determinació de l'estructura símbolica de la matriu

Per poder fer el següent apartat, la reordenació, cal primer saber com és la matriu que tenim, posat que en funció d'això requerirem d'un algoritme diferent. Per exemple, amb matrius simètriques, hi ha un gran nombre de algoritmes per fer la reordenació més eficientment. En canvi per matrius no simètriques, els algoritmes necessàris son uns altres diferents.