

CIMAT

Matrices dispersas

Miguel Vargas-Félix

miguelvargas@ciamat.mx
<http://www.cimat.mx/~miguelvargas>

Matrices dispersas

Son matrices en las cuales la gran mayoría de las entradas son cero.

En inglés se les conoce como “sparse matrices”, en algunos países de habla hispana también se les conoce como “matrices ralas”.

Sea \mathbf{A} una matriz dispersa de tamaño $n \times n$. Definamos la notación $\eta(\mathbf{A})$, que indica el número de entradas no cero de \mathbf{A} .

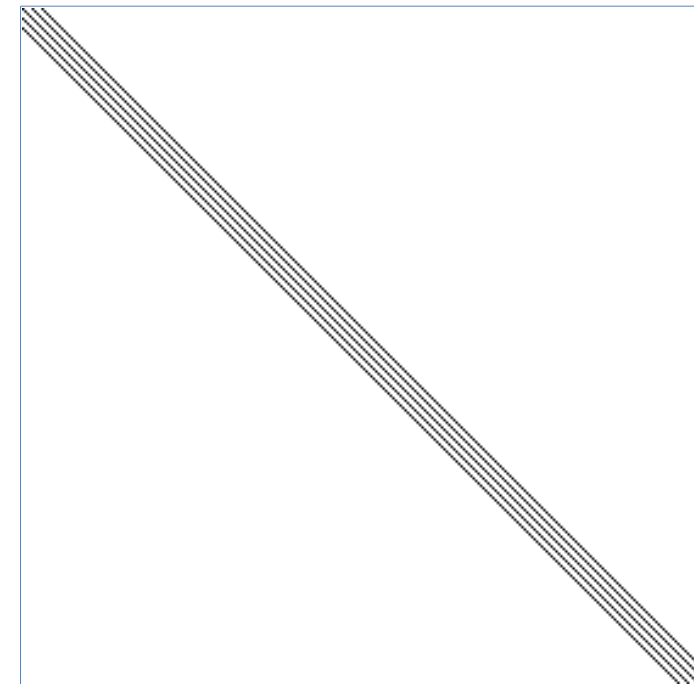
Por ejemplo

$$\mathbf{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & 0 & a_{24} & a_{25} & 0 & 0 \\ a_{31} & 0 & a_{33} & a_{34} & 0 & 0 & 0 \\ 0 & a_{42} & a_{43} & a_{44} & 0 & a_{46} & 0 \\ 0 & a_{52} & 0 & 0 & a_{55} & 0 & a_{57} \\ 0 & 0 & 0 & a_{64} & 0 & a_{66} & a_{67} \\ 0 & 0 & 0 & 0 & a_{75} & a_{76} & 0 \end{pmatrix},$$

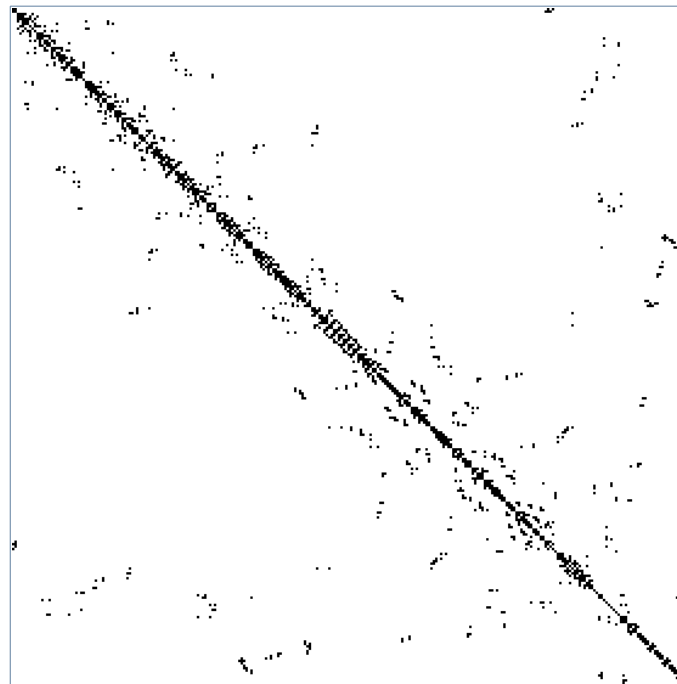
el número de entradas de \mathbf{A} es $7 \times 7 = 49$, el número de entradas distintas de cero es $\eta(\mathbf{A}) = 22$.

Con las matrices dispersas se pueden trabajar fácilmente matrices de millones de renglones por millones de columnas.

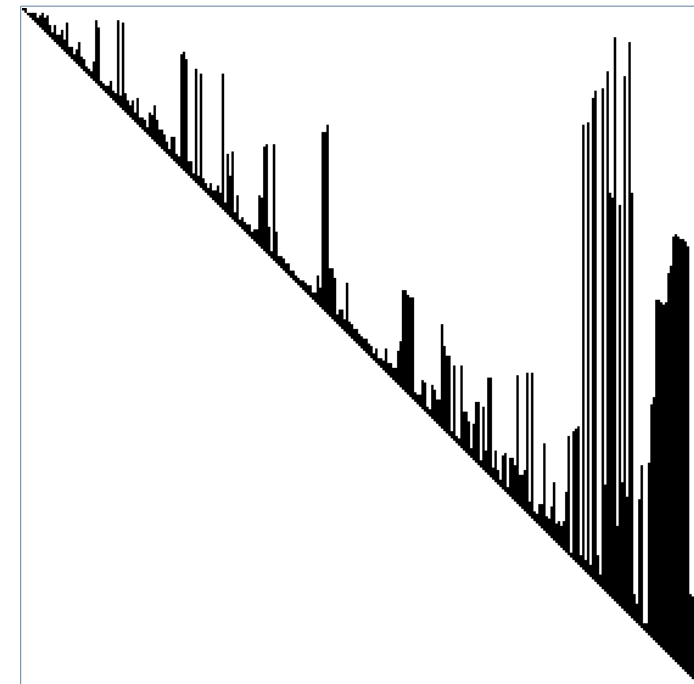
A veces se requiere solo visualizar las entradas distintas de cero, para ello se usan imágenes como las siguientes:



Bandada



No estructurada



Skyline

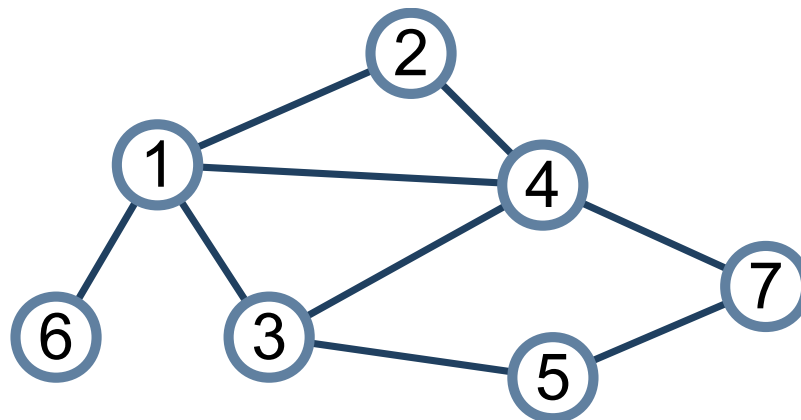
Los puntos negros representan las entradas de la matriz que son distintas de cero.

Matrices dispersas como grafos

Un grafo está formado por un conjunto de vértices y un conjunto de aristas $G=(\mathbf{V}, \mathbf{E})$.

- Los vértices están numerados, \mathbf{V} es el conjunto de todos los vértices.
- Cada arista conecta dos vértices. El conjunto de aristas \mathbf{E} está formado por pares no ordenados de vértices. Los grafos en los cuales no importa el orden se llaman grafos no dirigidos.

Por ejemplo:



En este caso, el conjunto de vértices es

$$\mathbf{V}=\{1, 2, 3, 4, 5, 6, 7\},$$

el conjunto de aristas es

$$\mathbf{E}=\{(1, 2), (1, 3), (1, 4), (1, 6), (2, 4), (3, 4), (3, 5), (4, 7), (5, 7)\}.$$

Una matriz dispersa con estructura simétrica se puede representar como un grafo no dirigido.

En el grafo no están representados los valores de la matriz, sólo la estructura de las entradas distintas de cero.

Sea \mathbf{A} una matriz dispersa de tamaño $n \times n$.

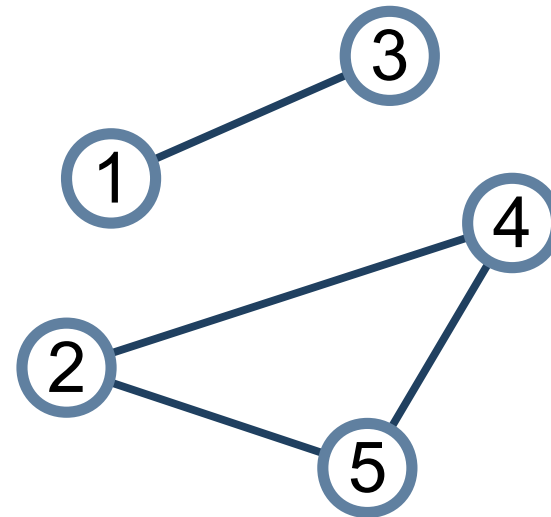
Cada vértice del grafo de \mathbf{A} representa un renglón (columna) de la matriz, $\mathbf{V} = \{v_1, v_2, \dots, v_n\}$.

El número de aristas es igual al número de entradas no cero de la matriz $\eta(\mathbf{A})$, así $\mathbf{E} = \{e_1, e_2, \dots, e_{\eta(\mathbf{A})}\}$.

Cada entrada no cero de la matriz $a_{ij} \neq 0$ es representada con una arista $e_k = (v_i, v_j)$, con $k = 1, 2, \dots, \eta(\mathbf{A})$.

Ejemplo:

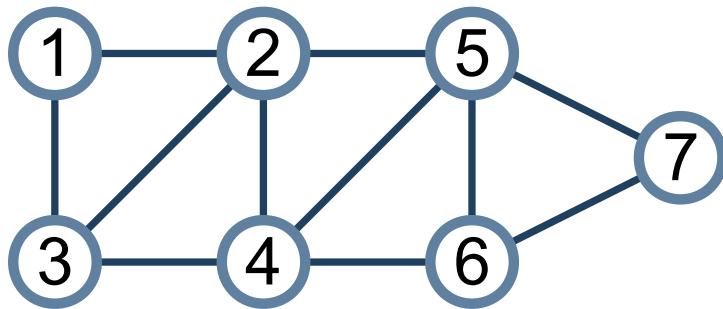
$$\begin{pmatrix} 3 & 0 & 9 & 0 & 0 \\ 0 & 4 & 0 & 6 & 1 \\ 9 & 0 & 2 & 0 & 0 \\ 0 & 6 & 0 & 8 & 5 \\ 0 & 1 & 0 & 5 & 3 \end{pmatrix}$$



Matrices dispersas de mallas de elemento finito

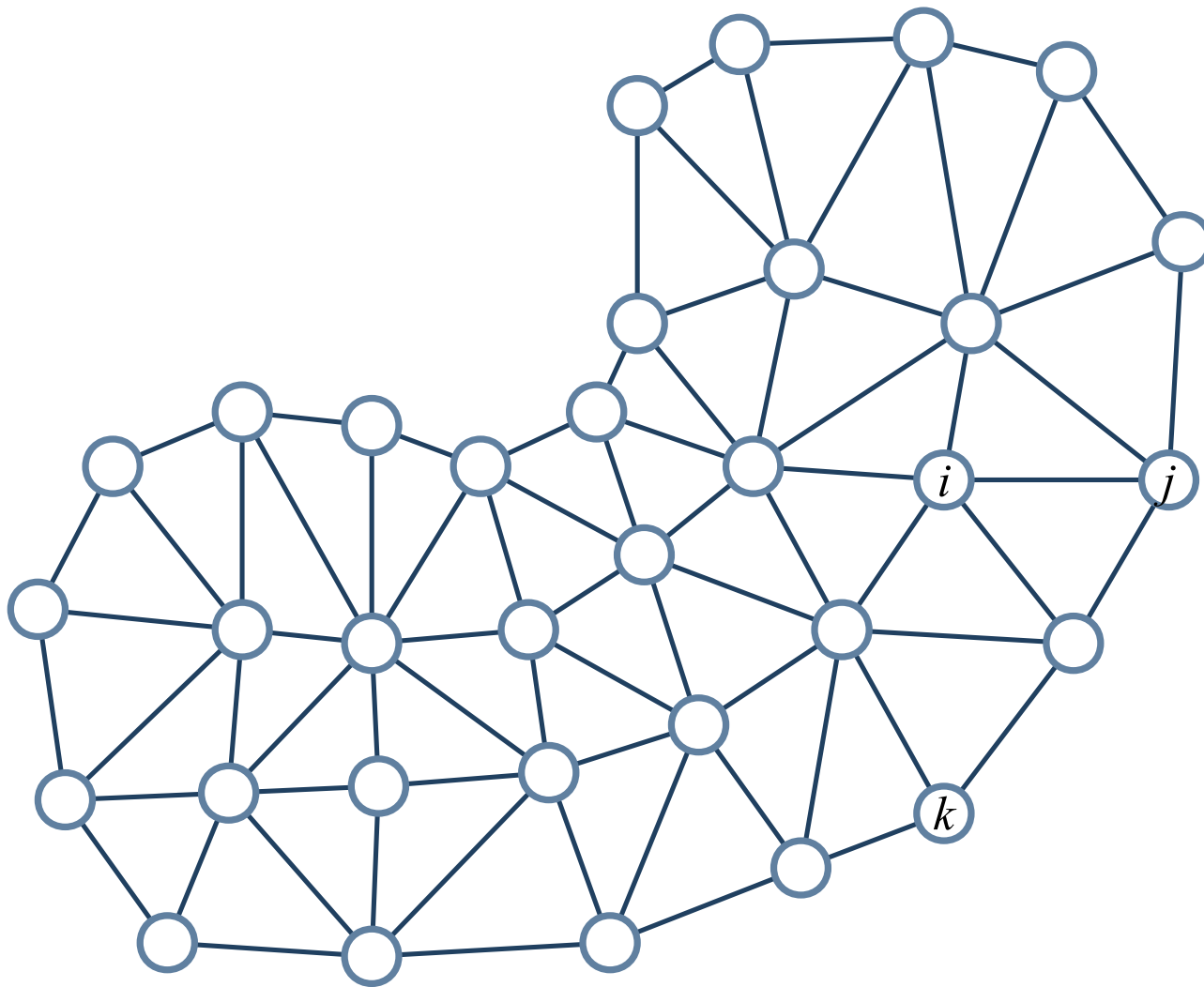
En el método de elemento finito, se trabaja de forma contraria, se parte de la malla (grafo) y con la matriz de conectividades se genera la matriz dispersa.

En las mallas de elemento finito tenemos un dominio discretizado en elementos, triángulos o cuadriláteros en 2D, tetraedros o hexaedros en 3D.



$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & 0 & 0 \\ a_{31} & a_{32} & a_{33} & a_{34} & 0 & 0 & 0 \\ 0 & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & 0 \\ 0 & a_{52} & 0 & a_{54} & a_{55} & a_{56} & a_{57} \\ 0 & 0 & 0 & a_{64} & a_{65} & a_{66} & a_{67} \\ 0 & 0 & 0 & 0 & a_{75} & a_{76} & a_{77} \end{pmatrix}$$

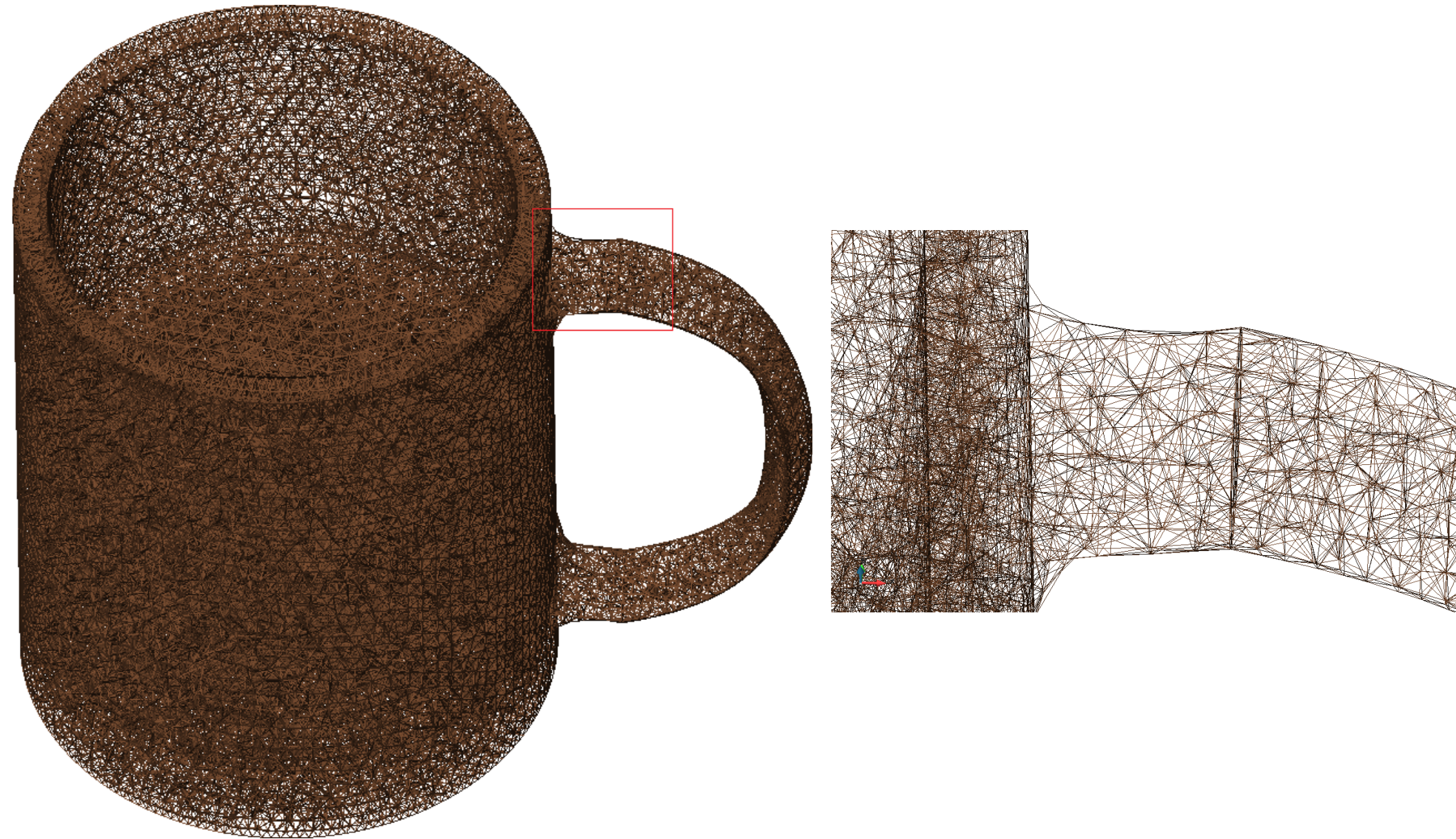
Dado que un nodo de la malla se conecta sólo con pocos nodos, tendremos una matriz muy dispersa. En general, el tamaño del número de entradas distintas de cero será $O(n)$.



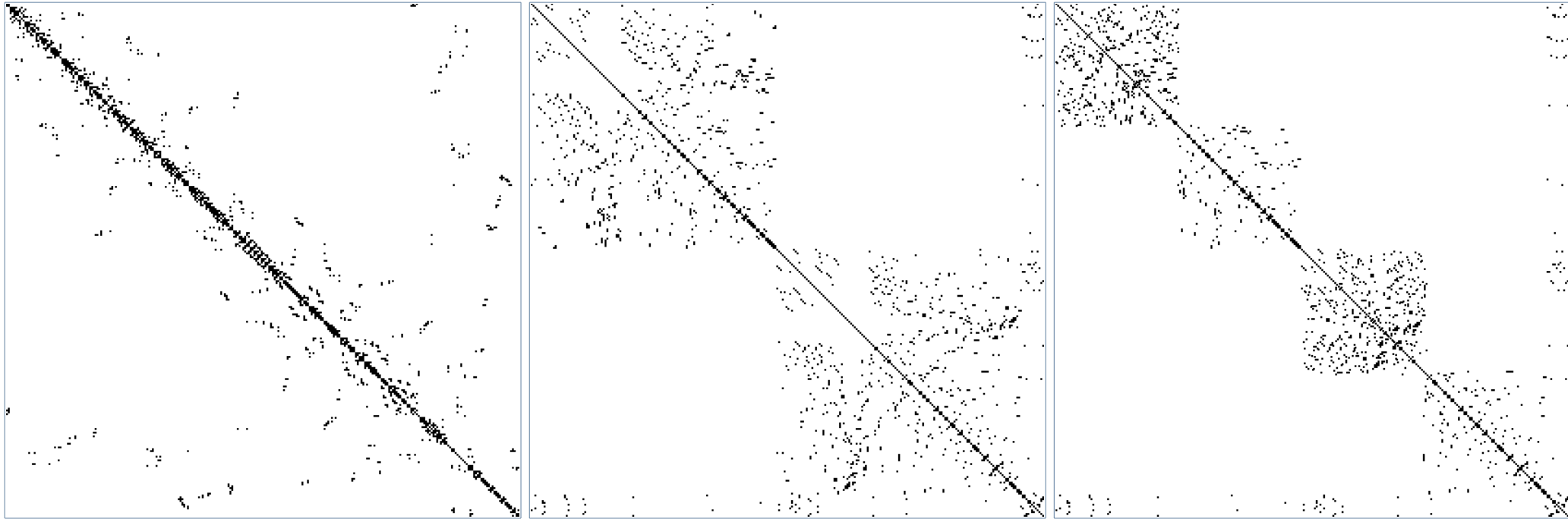
$$\begin{pmatrix} \circ & \circ & \circ & \circ & \circ & \circ & \circ & \dots \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \dots \\ \circ & a_{ii} & \circ & a_{ij} & \circ & \mathbf{0} & \circ & \dots \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \dots \\ \circ & a_{ji} & \circ & a_{jj} & \circ & \mathbf{0} & \circ & \dots \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \dots \\ \circ & \mathbf{0} & \circ & \mathbf{0} & \circ & a_{kk} & \circ & \dots \\ \circ & \circ & \circ & \circ & \circ & \circ & \circ & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

En el ejemplo, $a_{ij} \neq 0$ dado que están conectados con la arista (i, j) . En cambio $a_{ik} = 0$ puesto que no existe una arista (i, k) . De igual forma $a_{kj} = 0$ dado que no existe (k, j) .

Las mallas pueden llegar a ser muy complejas. El siguiente es el ejemplo de una malla de elemento finito en 3D formada por tetraedros.



Poder representar la estructura de matrices dispersas como grafos nos permitirá aplicar varios conceptos de teoría de grafos que resultarán en transformaciones a la matriz.



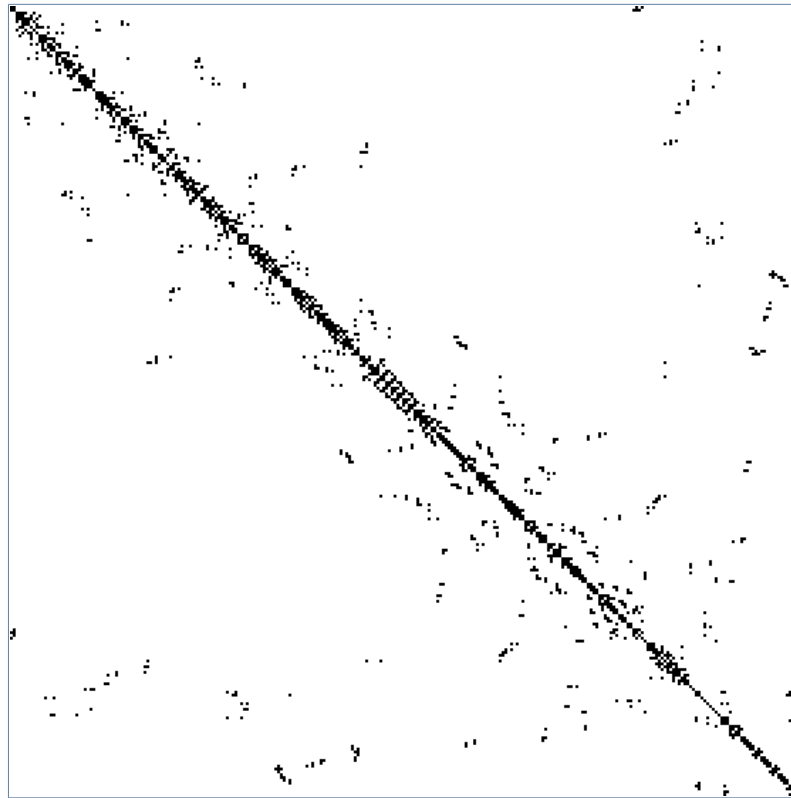
Por ejemplo:

- Reordenar la matriz para reducir el número de cálculos.
- Partir la matriz para enviarla a distintas computadoras y paralelizar los cálculos.

Almacenamiento de matrices dispersas

Las matrices llenas de tamaño $n \times n$ tienen un costo de almacenamiento de $O(n^2)$, las matrices dispersas suelen tener un costo de almacenamiento de $O(n)$.

Como ejemplo, la siguiente matriz $\mathbf{A} \in \mathbb{R}^{556 \times 556}$ contiene 309,136 entradas, con $\eta(\mathbf{A}) = 1810$, es decir sólo el 0.58% de las entradas son no cero.



Para ahorrar tanto memoria como tiempo de procesamiento sólo almacenaremos los elementos de la matriz \mathbf{A} que sean distintos de cero.

Hay varias estrategias de almacenamiento en memoria de matrices dispersas, dependiendo de la forma en que se accederán las entradas.

Por coordenadas

Para una matriz dispersa \mathbf{A} de tamaño $m \times n$, las entradas diferentes de cero serán enumeradas con $k = 1, 2, \dots, \eta(\mathbf{A})$.

Las entradas se almacenan por medio de tres vectores \mathbf{V} , \mathbf{I} y \mathbf{J} . Estos contendrán el valor V^k , el número de renglón I^k y el número de columna J^k de la k -ésima entrada no cero.

$$\begin{pmatrix} 8 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 & 0 & 0 \\ 2 & 0 & 1 & 0 & 7 & 0 \\ 0 & 9 & 3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{pmatrix} \begin{array}{cccccccccc} 3 & 3 & 8 & 7 & 4 & 2 & 1 & 1 & 1 & 5 & 9 \\ 4 & 2 & 1 & 3 & 1 & 3 & 4 & 3 & 2 & 5 & 4 \\ 3 & 4 & 1 & 5 & 2 & 1 & 6 & 3 & 3 & 6 & 2 \end{array} \begin{array}{l} \leftarrow \mathbf{V} \\ \leftarrow \mathbf{I} \\ \leftarrow \mathbf{J} \end{array}$$

Entre más grande sea el tamaño de la matriz más notorio será el ahorro en memoria.

Además de utilizar menos memoria, la otra gran diferencia con respecto a las matrices completas es que en las matrices dispersas las **entradas de la matriz deben ser buscadas**.

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

3	3	8	7	4	2	1	1	1	5	9	← V
4	2	1	3	1	3	4	3	2	5	4	← I
3	4	1	5	2	1	6	3	3	6	2	← J

Por ejemplo ¿cuál es el valor de la entrada a_{46} ?

¿Cuál es el valor de la entrada a_{55} ?

¿Cuál es la primer entrada distinta de cero de la columna 6?

¿Cuántas entradas distintas de cero tiene el renglón 3?

El costo de búsqueda es muy elevado, ya que en el peor caso hay que buscar en todas las entradas.

Este método no es adecuado si se necesitan acceder todos los elementos de un renglón o columna de forma secuencial.

Se pueden ordenar los vectores (por renglón o por columna). Al ordenar se puede utilizar una búsqueda binaria para encontrar una entrada a_{ij} , ésto reduce el costo de la búsqueda a $\log_2 \eta(\mathbf{A})$.

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

8	2	4	9	1	1	3	3	7	1	5	← V
1	3	1	4	2	3	4	2	3	4	5	← I
1	1	2	2	3	3	3	4	5	6	6	← J

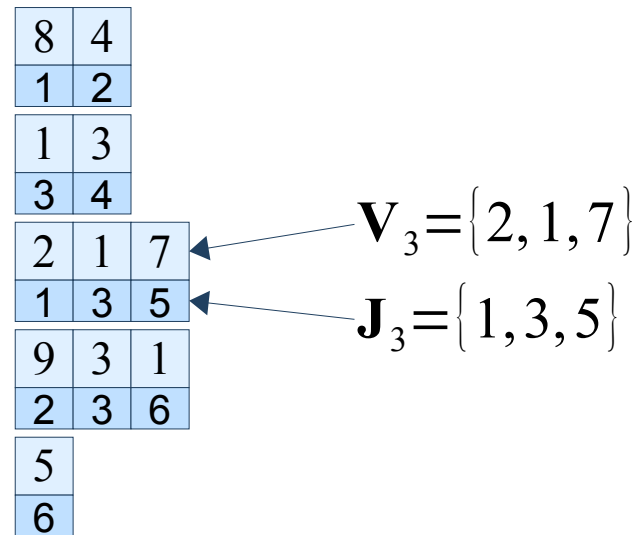
Acceso a:	Costo máximo
Entrada a_{ij} (sin ordenar)	$\eta(\mathbf{A})$
Entrada a_{ij} (ordenando)	$\log_2 \eta(\mathbf{A})$

Este tipo de almacenamiento es adecuado cuando se lee de forma incremental las entradas la matriz. Este formato es muy utilizado para almacenar matrices dispersas en archivos.

Compressed Row Storage

El método *Compressed Row Storage* (compresión por renglones) [Saad03 p362], se guardan las entradas no cero de cada renglón de \mathbf{A} por separado.

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



Con este método, por cada renglón de la matriz se guardan dos arreglos para las entradas distintas de cero de cada renglón. Un vector \mathbf{J}_i conteniendo los índices y otro \mathbf{V}_i los valores, con $i=1, \dots, m$.

Este esquema es adecuado cuando todos (o casi todos) los renglones tienen al menos una entrada distinta de cero.

Este tipo de esquema se puede crear utilizando un vector de vectores.

¿Como cambia el costo de búsqueda en relación al almacenamiento por coordenadas?

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

8	4	
1	2	
1	3	
3	4	
2	1	7
1	3	5
9	3	1
2	3	6
5		
6		

$\mathbf{V}_3 = \{2, 1, 7\}$

$\mathbf{J}_3 = \{1, 3, 5\}$

¿Cuál es el valor de la entrada a_{46} ?

¿Cuál es el valor de la entrada a_{55} ?

¿Cuál es la primer entrada distinta de cero de la columna 6?

¿Cuántas entradas distintas de cero tiene el renglón 3?

Sea $\eta(\mathbf{J}_i)$ el número de entradas no cero del renglón i de \mathbf{A} .

Los costos de búsqueda de las entradas ahora son:

Acceso a:	Costo máximo
Entrada a_{ij}	$\eta(\mathbf{J}_i)$
Primer valor no cero de un renglón	1
j -ésimo valor no cero de un renglón	1
Primer valor no cero de una columna	$\sum_{i=1}^m \eta(\mathbf{J}_i) = \eta(\mathbf{A})$
i -ésimo valor no cero de una columna	$\sum_{i=1}^m \eta(\mathbf{J}_i) = \eta(\mathbf{A})$

Podemos reducir el costo de acceso ordenando por índice las entradas de cada renglón.

Así, al momento de acceder podremos utilizar un patrón búsqueda binaria.

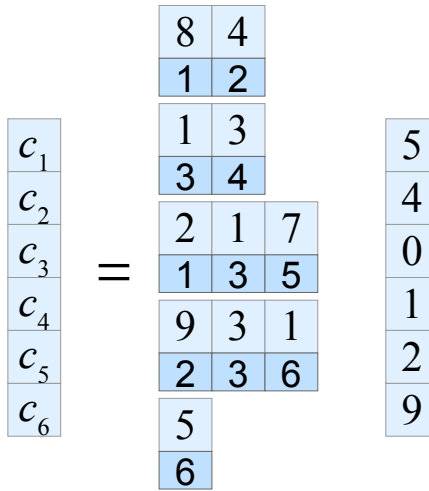
Acceso a:	Costo máximo
Entrada a_{ij}	$\log_2(\eta(\mathbf{J}_i))$
Primer valor no cero de un renglón	1
j -ésimo valor no cero de un renglón	1
Primer valor no cero de una columna	$\sum_{i=1}^m \log_2(\eta(\mathbf{J}_i))$
i -ésimo valor no cero de una columna	$\sum_{i=1}^m \log_2(\eta(\mathbf{J}_i))$

Multiplicación matriz vector

Realizar la multiplicación matriz-vector utilizando compresión por renglones resulta muy fácil.

En la multiplicación matriz-vector $\mathbf{c} = \mathbf{A} \mathbf{b}$ el orden de búsqueda es $O(1)$, esto es porque no se hace una búsqueda de las entradas del rengón, se toman las entradas una tras otra.

Sea \mathbf{J}_i el conjunto de índices de las entradas no cero del renglón i de \mathbf{A} . Sea $|\mathbf{J}_i|$ el número de entradas no cero del renglón i de \mathbf{A} . Nótese que $|\mathbf{J}_i| = |\mathbf{V}_i|$.

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \\ c_5 \\ c_6 \end{pmatrix} = \begin{pmatrix} 8 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 3 & 0 & 0 \\ 2 & 0 & 1 & 0 & 7 & 0 \\ 0 & 9 & 3 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 5 \end{pmatrix} \begin{pmatrix} 5 \\ 4 \\ 0 \\ 1 \\ 2 \\ 9 \end{pmatrix}$$


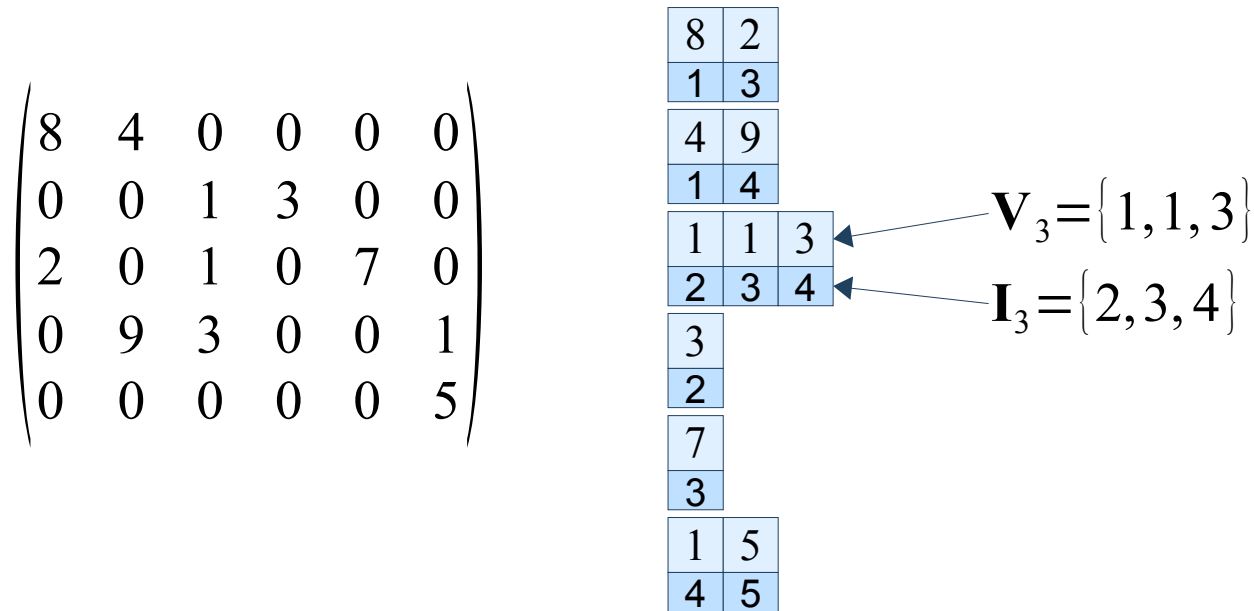
$$c_i = \sum_{j=1}^n a_{ij} b_j$$

$$c_i = \sum_{k=1}^{|\mathbf{J}_i|} \mathbf{V}_i^k b_{\mathbf{J}_i^k}$$

La ventaja de utilizar compresión por renglones es que los datos de cada renglón de la matriz de rigidez son accedidos en secuencia uno tras otro, esto producirá una ventaja de acceso al entrar el bloque de memoria de cada renglón en el *cache* del CPU.

Compressed Column Storage

Análogamente a la compresión por renglones, existe el método *Compressed Column Storage* en el cual se almacena por columna en vez de por renglón.



Por cada columna de la matriz se guardan dos arreglos para las entradas distintas de cero. Uno \mathbf{I}_j conteniendo los índices y otro \mathbf{V}_j los valores, con $j=1, \dots, n$.

Este esquema es adecuado cuando se quiere buscar entradas por columna y no por renglón.

Sea $\eta(\mathbf{I}_j)$ es el número de entradas no cero de la columna j de \mathbf{A} . Los costos de búsqueda de las entradas ahora son:

Acceso a:	Costo máximo
Entrada a_{ij}	$\eta(\mathbf{I}_j)$
Primer valor no cero de un renglón	$\sum_{j=1}^n \eta(\mathbf{I}_j) = \eta(\mathbf{A})$
j -ésimo valor no cero de un renglón	$\sum_{j=1}^n \eta(\mathbf{I}_j) = \eta(\mathbf{A})$
Primer valor no cero de una columna	1
i -ésimo valor no cero de una columna	1

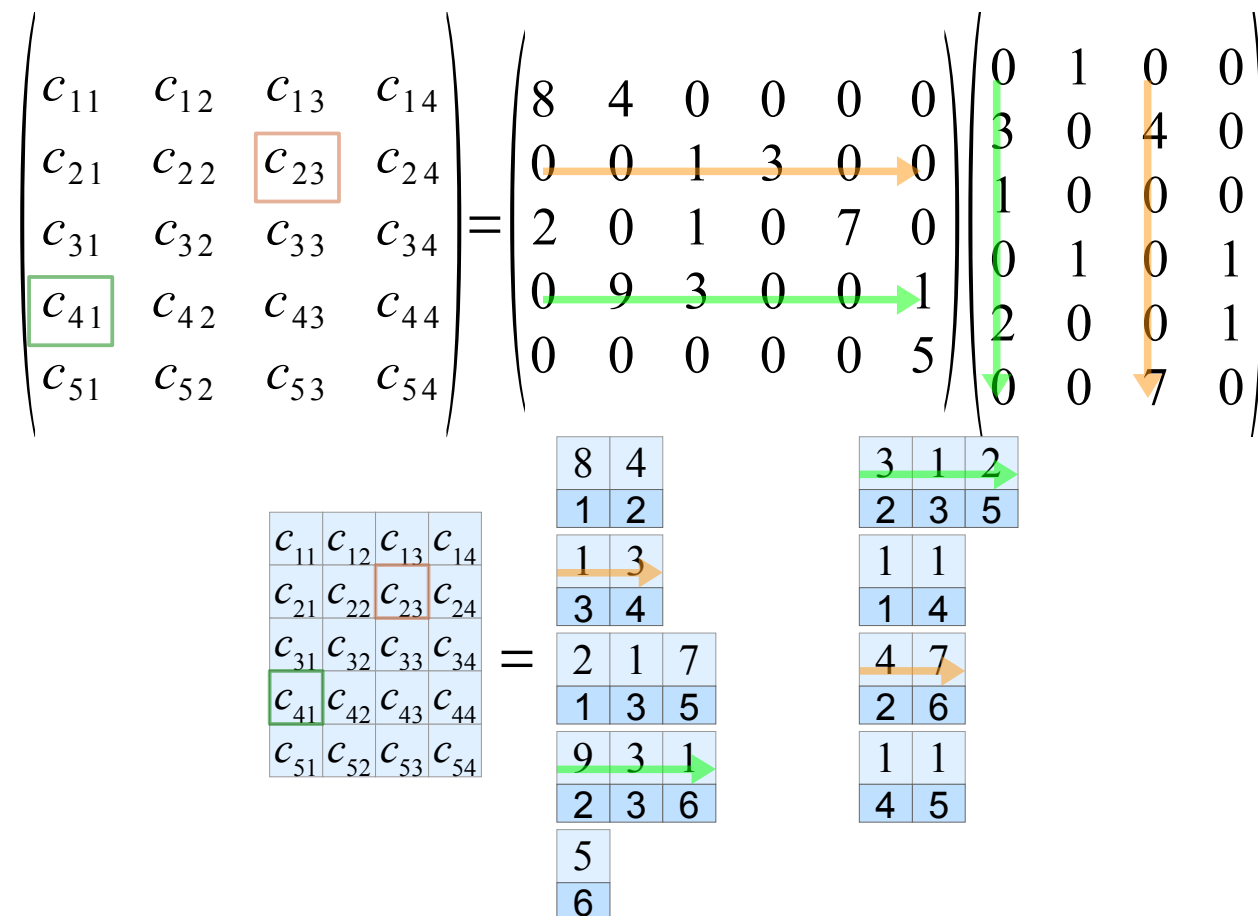
Este esquema es útil cuando se van a buscar entradas por columna y no por renglón. Por ejemplo, al multiplicar dos matrices dispersas.

Sean $\mathbf{A} \in \mathbb{R}^{m \times p}$ y $\mathbf{B} \in \mathbb{R}^{p \times n}$, la multiplicación $\mathbf{C} = \mathbf{A} \mathbf{B}$,

$$c_{ij} = \sum_{k=0}^p a_{ik} b_{kj}$$

es conveniente almacenar \mathbf{A} utilizando compresión por renglones y \mathbf{B} con compresión por columnas. El acceso de las entradas de ambas matrices dispersas será conveniente para el cache.

Por ejemplo:



Matrices dispersas en MatLab/Octave

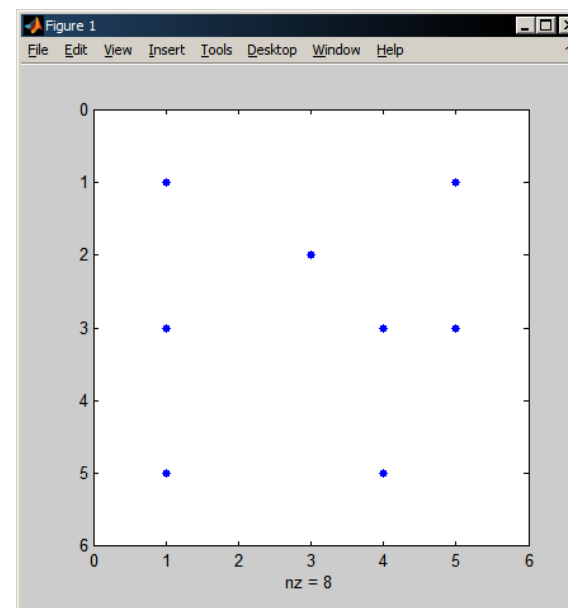
Tanto MatLab como Octave tienen soporte para manejar matrices dispersas.

El formato para definir matrices dispersas es utilizando almacenamiento por coordenadas.

Un ejemplo sencillo:

```
I = [1, 1, 2, 3, 3, 3, 5, 5];  
J = [1, 5, 3, 1, 4, 5, 1, 4];  
V = [9, 2, 5, 6, 2, 6, 1, 8];  
A = sparse(I, J, V);
```

```
A  
spy(A);  
nnz(A)  
full(A)
```



- **sparse(I, J, V)** crea una matriz dispersa, el **I** contiene los índices de renglón, **J** los índices de columna y **V** los valores.
- **spy(A)** despliega una figura mostrando las entradas no cero de la matriz **A**.
- **nnz(A)** indica el número de entradas distintas de cero en la matriz **A**.
- **full(A)** convierte la matriz dispersa en una matriz llena.

Almacenamiento con formato MAT-File version 4

Vamos a utilizar el formato MAT-File versión 4 de MATLAB para almacenar las matrices dispersas. Los archivos con formato MAT-File versión 4 sirven para almacenar escalares, vectores, matrices completas y matrices dispersas en formato binario.

Por ejemplo:

```
I = [1, 1, 2, 3, 3, 3, 5, 5];  
J = [1, 5, 3, 1, 4, 5, 1, 4];  
V = [9, 2, 5, 6, 2, 6, 1, 8];  
A = sparse(I, J, V);  
x = [0, 1.1, 4, 20, 11, 0, 0, 2];
```

Guardar en formato MAT-File 4:

```
save('-v4', 'test_file.mat', 'A', 'x');
```

Leer archivo:

```
load('test_file.mat');
```

Este formato funciona tanto con MATLAB como con Octave.

En el formato MAT-File versión 4 todos los elementos son almacenados como matrices. Por ejemplo:

- Un escalar es una matriz de tamaño 1x1.
- Un vector columna es una matriz con N renglones y una columna.

Cada archivo MAT-File versión 4 puede contener varias matrices. Se guardan una detrás de otra.

Hay 3 formatos de matrices: completas, dispersas y texto (estas últimas no las vamos a utilizar).

Las matrices se guardan en formato binario.

La forma más simple de abrir un archivo MAT-File versión 4 (con formato binario) es con:

```
FILE* file;  
file = fopen("file_name.mat", "rb");
```

"rb" significa **read binary**.

El encabezado

Cada matriz comienza con un *header* de **5 enteros** de 32bits que contiene información sobre los atributos de la matriz. Podemos leer estos enteros así:

```
int header[5];  
fread(header, sizeof(int), 5, file);
```

header[0] (**int**), indica el formato de la matriz y el tipo de datos para la matriz:

0	Matrix completa
1	Texto
2	Matriz rala
+0	double
+10	float
+20	int
+30	short
+40	unsigned short
+50	unsigned char

Podemos leer estos parámetros así:

```
int format = header[0] % 10;  
int type = header[0] - format;
```

El significado de los siguientes cuatro enteros depende del tipo de matriz.

Matrices completas

(**int**) header[1], número de renglones

(**int**) header[2], número de columnas

(**int**) header[3], tipo de valores

0	Reales
1	Complejos

(**int**) header[4], longitud del nombre de la matriz + 1. Normalmente la longitud máxima es 63 + 1.

(**char**[]), arreglo de caracteres, nombre de la matriz + '\0'

(**double**[]) Arreglo de valores, corresponde a las entradas reales de la matriz. El número de valores es renglones*columnas. **Los valores están almacenados por columnas.** El valor 1 corresponde a la entrada (1,1), el valor 2 a la entrada (2,1) y así sucesivamente.

(**double**[]) Sólo si es el tipo de valores (header[3]) indica complejos. Valores imaginarios de la matriz. El número de valores es renglones*columnas. Los valores están almacenados por columnas, de igual forma que los reales.

Ejemplo de matriz completa

```
A=[0, 1.1, 2; 3, 4, 5; 6, 7, 8]
```

```
A =
```

```
0.00000 1.10000 2.00000
3.00000 4.00000 5.00000
6.00000 7.00000 8.00000
```

```
save('-v4', 'full.mat', 'A');
```

El archivo contendrá:

0	0	0	0	3	0	0	0	3	0	0	0	0	0	0	0
2	0	0	0	65	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	8	64	0	0	0	0	0	0	24	64	154	153
153	153	153	153	241	63	0	0	0	0	0	0	16	64	0	0
0	0	0	0	28	64	0	0	0	0	0	0	0	64	0	0
0	0	0	0	20	64	0	0	0	0	0	0	32	64		

(int) header[0] Formato y el tipo de datos para la matriz (0 = full matrix, +0 = double)

(int) header[1] Número de renglones (3)

(int) header[2] Número de columnas (3)

(int) header[3] Tipo de valores (0 = real)

(int) header[4] Longitud del nombre de la matriz + 1 (2)

(char[2]) Arreglo de caracteres, nombre de la matriz + '\0' ("A\0")

(double[9]) Corresponde a las entradas reales de la matriz (doubles, 8 bytes cada valor)

Los valores **int32** y **double** están guardados con formato Little-Endian.

Ejemplo de cómo leer una matriz llena:

```
FILE* file = fopen(argv[1], "rb");
int header[5];
fread(header, sizeof(int), 5, file);
int format = header[0] % 10;
int type = header[0] - format;
if (format == 0) // Full matrix
{
    int rows = header[1];
    int cols = header[2];
    int field = header[3];
    int name_size = header[4];
    char name[128];
    fread(name, name_size, 1, file);
    double* real = new double[rows*cols];
    fread(real, sizeof(double), rows*cols, file);
    double* imag = NULL;
    if (field == 1) // Complex
    {
        imag = new double[rows*cols];
        fread(imag, sizeof(double), rows*cols, file);
    }
    // ...
    delete [] imag; delete [] real;
}
fclose(file);
```

Matrices dispersas

(**int**) header[1], número de entradas no cero + 1, $\eta(\mathbf{A})+1$

(**int**) header[2], tipo de valores

3	Reales
4	Complejos

(**int**) header[3], valor fijo en 0

(**int**) header[4], longitud del nombre de la matriz + 1

(**char**[]) Arreglo de caracteres, nombre de la matriz + '\0'

(**double**[]) Arreglo de índices de renglón. El número de índices es $\eta(\mathbf{A})$

(**double**) Número de renglones

(**double**[]) Arreglo de índices de columna. El número de índices es $\eta(\mathbf{A})$

(**double**) Número de columnas

(**double**[]) Valores. El número de valores es $\eta(\mathbf{A})$

(**double**) Valor fijo en 0

Nota importante: El número de renglones y de columnas están guardadas con **double**, no con **int**.

```
spa = sparse([1, 2, 1, 3], [1, 1, 2, 2], [-1, 2, 1.1, 3])
```

```
spa =
```

```
Compressed Column Sparse (rows = 3, cols = 2, nnz = 4 [67%])
```

```
(1, 1) -> -1
```

```
(2, 1) -> 2
```

```
(1, 2) -> 1.1000
```

```
(3, 2) -> 3
```

```
save('-v4', 'sparse.mat', 'spa');
```

2	0	0	0	5	0	0	0	3	0	0	0	0	0	0	0
4	0	0	0	115	112	97	0	0	0	0	0	0	0	240	63
0	0	0	0	0	0	0	64	0	0	0	0	0	0	240	63
0	0	0	0	0	0	8	64	0	0	0	0	0	0	8	64
0	0	0	0	0	0	240	63	0	0	0	0	0	0	240	63
0	0	0	0	0	0	0	64	0	0	0	0	0	0	0	64
0	0	0	0	0	0	0	64	0	0	0	0	0	0	240	191
0	0	0	0	0	0	0	64	154	153	153	153	153	153	241	63
0	0	0	0	0	0	8	64	0	0	0	0	0	0	0	0

(int) header[0] Formato y tipo de datos para la matriz (2 = sparse matrix, +0 = double)

(int) header[1] Número de entradas no cero + 1 (5 = 4 + 1)

(int) header[2] Tipo de valores (3 = real)

(int) header[3] Valor fijo en 0 (0)

(int) header[4] Longitud del nombre de la matriz + 1 (4 = 3 + 1)

(char[4]) Arreglo de caracteres, nombre de la matriz + '\0' ("spa\0")

(double[4]) Arreglo de índices de renglón (4 doubles)

(double) Número de renglones (3)

(double[4]) Arreglo de índices de columna (4 doubles)

(double) Número de columnas

(double[4]) Entradas de la matriz (4 doubles)

(double) Valor fijo en 0

Ejemplo de cómo escribir una matriz dispersa:

```
const char* name = "spa";  
int name_size = strlen(name) + 1;  
double rows = 3;  
double cols = 2;  
double I[4] = {1, 2, 1, 3};  
double J[4] = {1, 1, 2, 2};  
double V[4] = {-1, 2, 1.1, 3};  
  
FILE* file = fopen("sparse.mat", "wb");  
  
int header[5];  
header[0] = 2 + 0; // 2 = sparse matrix, +0 = double  
header[1] = 4 + 1; // Number of non-zero + 1  
header[2] = 3; // 3 = real  
header[3] = 0; // fixed value 0  
header[4] = name_size; // name length + 1  
fwrite(header, sizeof(int), 5, file);  
fwrite(name, sizeof(char), name_size, file);  
  
fwrite(I, sizeof(double), 4, file);  
fwrite(&rows, sizeof(double), 1, file);  
  
fwrite(J, sizeof(double), 4, file);  
fwrite(&cols, sizeof(double), 1, file);  
  
fwrite(V, sizeof(double), 4, file);  
  
double zero = 0;  
fwrite(&zero, sizeof(double), 1, file);  
  
fclose(file);
```

Nota importante: En el formato de matrices dispersas, las entradas deben estar ordenadas, primero por el índice de columna y segundo por el índice de renglón.

Por ejemplo, la siguiente matriz rala:

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

3	3	8	7	4	2	1	1	1	5	9	← V
4	2	1	3	1	3	4	3	2	5	4	← I
3	4	1	5	2	1	6	3	3	6	2	← J

Para poder ser leída correctamente por MatLab hay que reordenar las entradas en base a los índices de columna y renglón antes de guardarla. De la siguiente forma:

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

8	2	4	9	1	1	3	3	7	1	5	← V
1	3	1	4	2	3	4	2	3	4	5	← I
1	1	2	2	3	3	3	4	5	6	6	← J

Octave es menos riguroso con el orden de los índices, pero por compatibilidad es mejor ordenarlos.

¿Preguntas?

migueltvargas@cimat.mx

Referencias

[Piss84] S. Pissanetzky. *Sparse Matrix Technology*. Academic Press, 1984.

[Saad03] Y. Saad. *Iterative Methods for Sparse Linear Systems*. SIAM, 2003.