Computación 1
- 2021-

Matrices dispersas

Motivación

- La resolución de una gran cantidad de problemas ingenieriles involucra la utilización de matrices.

- Mejorar la precisión de la resolución de diversos problemas implica aumentar el tamaño de las matrices involucradas.

Matrices dispersas

- Situación:
  - Matrices muy grandes
  - Previsible “gran porcentaje” de valores = 0

Se busca una forma de representar esas matrices que “cueste” menos memoria y permita acelerar los cálculos.

Las matrices son *naturalmente* muy eficientes en el uso de memoria -para almacenar datos- y de procesador -para accederlos-.

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

```
0  0  0  0  0  0  0  0  0  0  0
0  1  0  0  0  0  0  0  0  0  0
0  0  0  2  0  0  0  0  0  0  0
0  0  0  3  0  0  0  0  0  0  0
0  0  0  0  4  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  6  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0
```

Matrices dispersas

Representación en memoria de una matriz convencional

La memoria es una secuencia de bits organizada en bloques de a 8

Si tenemos matriz (m, n) de reales de 4 Bytes

Trabajando en lenguajes como “C” que almacena por “fila” (“Fortran” por “columna”).

La posición en bits de la celda (*i, j*) en la memoria es:

inicio_matriz + **(((i – 1) * n) + (j - 1)) * (4 * 8)**

Matrices dispersas

Representación en memoria de una matriz convencional

El caso inverso es igual de sencillo:

Dado un entero que representa una posición dentro de una secuencia de bytes determinar fila y columna (con sintaxis Octave):

Fila i = floor(posición_de_celda-1 / n) + 1

Columna j = mod(posición_de_celda-1, n)+1

Matrices dispersas

Conclusiones sobre el uso de matrices convencionales

- Son pocos cálculos
- Esos cálculos son “elementales”: **+**, **-**, ***** y **/**
- Son cálculos con enteros
  - parte entera y módulo se pueden hacer con aritmética de punto fijo
- Si la matriz (tensor) es n-dimensional (n > 2) se agrega complejidad pero se mantienen estas características generales

Matrices dispersas

Definición de MATRIZ DISPERSA

Es aquella que está compuesta por muchos elementos de valor = 0 de tal forma que los que son distintos de 0 se encuentran muy dispersos en la matriz y sin relación entre sí.

El qué tan dispersos están depende de las circunstancias. En nuestro caso nos preocupa el uso de memoria.

Eventualmente se podría definir como dispersa con respecto a otro valor distinto de 0.

Matrices dispersas

Forma de representación

- Debemos tratar de preservar los principios de economía:
  - Mínimo consumo de memoria
  - Mínimo uso de procesador
  - “tiempo” para acceder a los datos (los usos de esos datos de por sí pueden requerir su cuota de procesador)
  - “tiempo” de cálculos
  - Localidad de datos

Matrices dispersas

- Dos grandes familias de estrategias para almacenamiento para matrices dispersas:
  - formatos estáticos: Útiles cuando se conoce la estructura de la matriz, o cuando ésta se genera en una única ocasión.
  - formatos dinámicos: Cuando la estructura de la matriz dispersa cambia en forma continua.

Formatos estáticos

- Simple o elemental (coordinate)
- Comprimido por fila (CRS)
- Comprimido por columna (CCS)
- Comprimido por fila a bloque (BCRS)
- Comprimido por diagonales (CDS)
- Jagged diagonal scheme and skyline

Matrices dispersas

Forma de representación

- Formato simple o elemental
  - 3 vectores
    - vector_f (fila), enteros
    - vector_c (columna), enteros
    - vector_d (dato), pto. flotante

Matrices dispersas

Forma de representación

Ventajas: (formato elemental)

- Seguimos usando vectores (matrices unidimensionales) que tienen las características descriptas y deseadas
- Funcionamiento sencillo, por ejemplo:

para mat(3, 1) = 100

- `vector_f( n ) = 3`
- `vector_c( n ) = 1`
- `vector_d( n ) = 100`

¿quién es n?

n = ordinal del elemento entre los distintos de 0

Formatos estáticos

- Formato simple o elemental
  - 3 vectores
    - vector_f (fila)
    - vector_c (columna)
    - vector_d (dato)

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

$$d = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \end{pmatrix}$$
$$f = \begin{pmatrix} 1 & 1 & 1 & 1 & 2 & 2 & 3 & 3 & 3 & 4 & 4 & 5 & 5 & 5 & 6 & 6 & 6 & 7 \end{pmatrix}$$
$$c = \begin{pmatrix} 1 & 2 & 5 & 6 & 2 & 4 & 2 & 3 & 4 & 1 & 4 & 2 & 5 & 6 & 3 & 5 & 6 & 7 \end{pmatrix}$$

Matrices dispersas

Una forma de representación: costos

Si quisiéramos representar una matriz que contiene enteros de 4 bytes:

- En una matriz ***completa***: (m x n) * 4
- En la matiz ***dispersa*** máx.: (nz * 4) * 3

Empezamos a “economizar” cuando la cantidad de elementos distintos de 0 es < 1/3 del total

Esto cambia según el tamaño de los elementos a guardar y el de los usados para los vectores fila y columna

Matrices dispersas

Una forma de representación: costos

Si quisiéramos representar una matriz que contiene reales de 4 bytes y utilizamos enteros de 2 bytes para los índices de filas y columnas:

- En una matriz ***completa***: (m x n) * 4
- En la matiz ***dispersa*** máx.: (nz * 4) + 2 * nz * 2

Empezamos a “economizar” cuando la cantidad de elementos distintos de 0 es < 1/2 del total

Formatos estáticos

- Ventajas
  - intuitivo
  - se utilizan estructuras del mismo tamaño
  - es igual si queremos acceder por fila que por columna
- Desventajas
  - utiliza más memoria que otras implementaciones
  - acceder por fila o columna es más difícil que en otros formatos

Matrices dispersas

forma de representación

- Formato CSR- (Compressed Storage Row) o CRS
  - vector_f (índice de cada nueva fila)
  - vector_c (columna)
  - vector_d (dato)

Matrices dispersas

forma de representación

- Formato CRS

  - Un vector de punto flotante de tamaño *nz* en el que se almacenan los valores de los coeficientes.
  - Otro vector de tamaño *nz* en el que se almacenan los números de columna de los elementos distintos de cero.
  - Un vector de tamaño *n+1* siendo *n* la cantidad de filas de la matriz, en el cual se almacena la posición de la primera ocurrencia de cada fila

Matrices dispersas

forma de representación

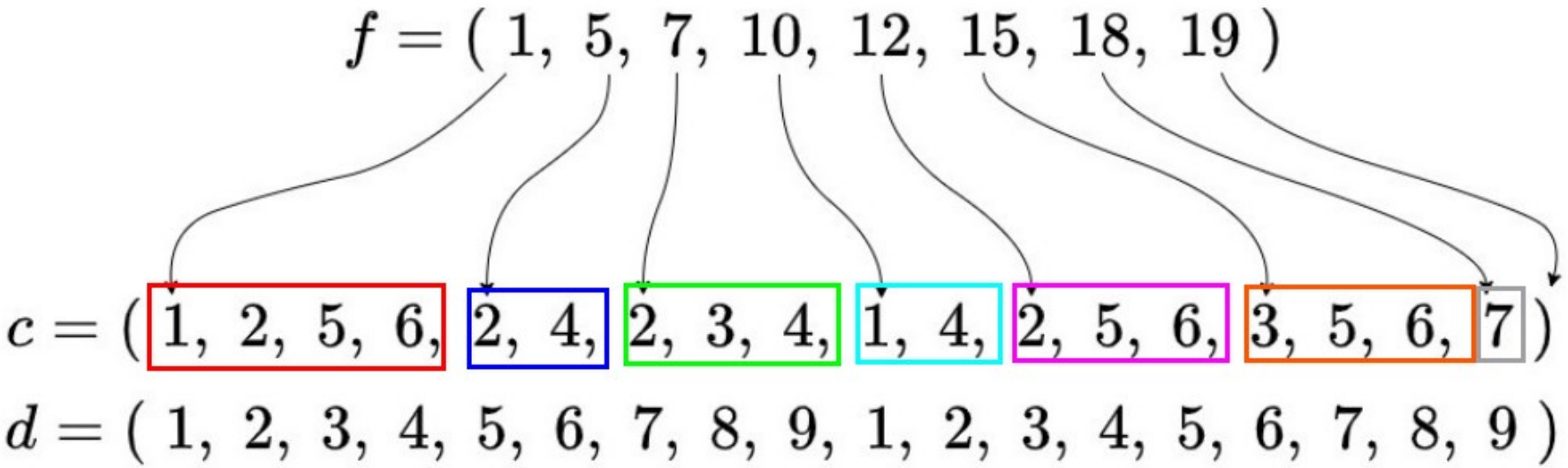
- Formato CRS

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

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

$$f = \begin{pmatrix} 1 & 5 & 7 & 10 & 12 & 15 & 18 & 19 \end{pmatrix}$$

$$c = \begin{pmatrix} 1 & 2 & 5 & 6 & 2 & 4 & 2 & 3 & 4 & 1 & 4 & 2 & 5 & 6 & 3 & 5 & 6 & 7 \end{pmatrix}$$



Para acceder al valor A(i, j) de la matriz A, se obtienen primero los índices $p_1 = f(i)$ y $p_2 = f(i + 1)$, posteriormente se busca el indice p tal que $p_1 \leq p < p_2$ y c(p) = j, y luego se puede obtener el valor buscado accediendo a d(p)

f(1) = 1 ,

f(i + 1) − f(i) = numero de elementos no nulos en la fila i .

Formatos estáticos

CRS

- Ventajas
  - utiliza menos memoria que otras estrategias
  - es fácil acceder a una fila “completa”
- Desventajas
  - no se utilizan estructuras del mismo tamaño
  - es difícil acceder a una columna “completa”

Matrices dispersas

forma de representación

- Formato CCS (Compressed Column Storage)
  - Igual al CRS pero almacena por columna

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

$$f = \begin{pmatrix} 1 & 4 & 1 & 2 & 3 & 5 & 3 & 6 & 2 & 3 & 4 & 1 & 5 & 6 & 1 & 5 & 6 & 7 \end{pmatrix}$$

$$c = \begin{pmatrix} 1 & 3 & 7 & 9 & 12 & 15 & 18 & 19 \end{pmatrix}$$

Matrices dispersas

forma de representación

- Otros formatos
  - DIA (Diagonal Storage Format)
  - Ellpack-itpack
  - Estrategias dinámicas

Matriz B almacenada en formato DIA.

$$B = \begin{pmatrix} \overset{0}{4.6} & \overset{1}{9.5} & 0 & \overset{3}{7.6} & 0 & 0 & 0 & 0 \\ 0 & 1.1 & 4.9 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2.5 & 7.1 & 0 & 6.6 & 0 & 0 \\ \overset{-3}{4.2} & 0 & 0 & 1.5 & 3.3 & 0 & 9.7 & 0 \\ 0 & 1.8 & 0 & 0 & 8.8 & 9.4 & 0 & 5.1 \\ 0 & 0 & 4.8 & 0 & 0 & 3.0 & 4.6 & 0 \\ 0 & 0 & 0 & 0.1 & 0 & 0 & 2.9 & 3.4 \\ 0 & 0 & 0 & 0 & 2.8 & 0 & 0 & 1.2 \end{pmatrix}$$

diag:	-3	0	1	3
val:		4.6	9.5	7.6
		1.1	4.9	0.0
		2.5	7.1	6.6
	4.2	1.5	3.3	9.7
	1.8	8.8	9.4	5.1
	4.8	3.0	4.6	
	0.1	2.9	3.4	
	2.8	1.2		

Matrices Dispersas

Solución de Octave: comando **sparse**

```
a =
   1   0   0   0   0   0   0   0   0   0
   0   1   0   0   0   0   0   0   0   0
   0   0   1   0   0   0   0   0   0   0
   0   0   0   1   0   0   0   0   0   0
   0   0   0   0   1   0   0   0   0   0
   0   0   0   0   0   1   0   0   0   0
   0   0   0   0   0   0   1   0   0   0
   0   0   0   0   0   0   0   1   0   0
   0   0   0   0   0   0   0   0   1   0
   0   0   0   0   0   0   0   0   0   1
```

La matriz ***b*** ocupa tan sólo 80 bytes frente a los 800 de la matriz ***a*** y la cantidad de elementos del usuario son 110.

Ahí no figuran los que Octave debe usar para mantener la estructura de la m. d., pero en los 164 bytes sí, ya que para los 10 datos son suficientes 80 bytes y se están usando 164.

```
» b = sparse( a )
b =
   (1,1)        1
   (2,2)        1
   (3,3)        1
   (4,4)        1
   (5,5)        1
   (6,6)        1
   (7,7)        1
   (8,8)        1
   (9,9)        1
  (10,10)       1
» whos
  Name      Size            Bytes  Class
  a         10x10             800  double array
  b         10x10             164  sparse array
Grand total is 110 elements using 964
    bytes
```

Matrices Dispersas

Ejemplos de uso de espacio de Octave

```
a =
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
» whos
  Name      Size         Bytes  Class

  a        10x10          800  double array

Grand total is 100 elements using 800 bytes
```

```
» a = [ ones( 10, 5 ) zeros( 10, 5) ]
a =
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
    1    1    1    1    1    0    0    0    0    0
» b = sparse( a );
» whos
  Name      Size         Bytes  Class

  a        10x10          800  double array
  b        10x10          644  sparse array

Grand total is 150 elements using 1444 bytes
```

Matrices Dispersas

Solución de Octave: comando **sparse**

- Octave utiliza CRS
  - Ejemplo 1, matriz de 10x10 con 10 entradas.
    - 10 entradas * 8 byte = 80 bytes
    - 10 filas * 4 byte = 40 bytes
    - 11 entradas en * 4 byte = 44 bytes

      Total 164 bytes
  - Ejemplo 2, matriz 10x10 con 50 entradas.
    - 50 entradas * 8 byte = 400 bytes
    - 50 filas * 4 byte = 200 bytes
    - 11 entradas en * 4 byte = 44 bytes

      Total 644 bytes

Matrices Dispersas

Solución de Octave: generación de matriz dispersa

Si tenemos una matriz *a* en *forma completa*

s = sparse(a)

Octave se encarga de dimensionar las estructuras auxiliares

Matrices Dispersas

Solución de Octave: generación de matriz dispersa

Si la matriz ya está en forma dispersa elemental:

- **s = sparse(f, c, d, m, n, nzmax)**

- f, c y d son los vectores que representan filas, columnas y datos respectivamente
- m y n indican cantidad de filas y columnas que tendrá -si no se especifican se tomará m = max(f) y n = max(c) -
- nzmax es la máxima cantidad de elementos distintos de ceros (NonZeroMax)

Matrices Dispersas

Convertir de dispersa a completa

- ¿cómo saber si una matriz es dispersa?:

issparse(mat)

- devuelve TRUE o FALSE (1 o 0)

- Convertir de dispersa a completa:

a = full(s)

Matrices Dispersas

```
» a(6, 2) = 1; a(2, 4) = 1;
» a

a =

     0     0     0     0     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

```
» vf( 1 ) = 2; vc( 1 ) = 4; vd( 1 ) = 1;
» vf( 2 ) = 6; vc( 2 ) = 2; vd( 2 ) = 1;
» a2 = sparse( vf, vc, vd)
a2 =
   (6,2)        1
   (2,4)        1
» whos
  Name      Size              Bytes  Class
   a2       6x4                  44  sparse array
   vc       1x2                  16  double array
   vd       1x2                  16  double array
   vf       1x2                  16  double array
Grand total is 8 elements using 92 bytes
» b2 = full( a2 )
b2 =
     0     0     0     0
     0     0     0     1
     0     0     0     0
     0     0     0     0
     0     0     0     0
     0     1     0     0
```

Matrices Dispersas

Solución de Octave: operaciones

- Todas las operaciones se hacen con matrices dispersas de igual modo que con las “completas”.

Se debe tener en cuenta lo siguiente:

- A_disp oper B_disp = R_disp
- A_comp oper B_comp = R_comp
- A_disp oper B_comp = R_comp (excepto que de antemano se sepa que será dispersa)

Matrices Dispersas

Funciones

- speye - Matriz identidad dispersa.
- find - Devuelve los indices de los coeficientes distintos de cero.
- nnz - Numero de elementos no cero de la matriz.
- nonzeros - Elementos no cero de la matriz.

- spy - Visualiza el patrón de la matriz.

Matrices Dispersas

Funciones

- Algebra lineal:
  - eigs - Algunos valores propios.
  - svds - Algunos valores singulares.
  - ilu - Factorización LU incompleta.
  - ichol - Factorización de Cholesky incompleta.
  - normest - Norma 2 estimada.
  - pcg - Gradiente conjugado precondicionado.
  - etree - Árbol de eliminación (treeplot).

Matrices Dispersas

para el curso

Igual que en polinomios, una estructura de datos “sencilla” (e intuitiva) para realizar diferentes algoritmos !!!!

Matrices Dispersas

Ejercicio

Obtener la transpuesta de una matriz dispersa en formato elemental...

Matrices Dispersas

Ejercicio

```
function [c,f,v] = transp(f,c,v)
```

Matrices Dispersas

Ejercicio

Escribir una función recursiva que reciba una matriz dispersa en formato elemental y un número x, y devuelva la matriz eliminando todas las entradas iguales a x.

Matrices Dispersas

Ejercicio

```
function [fs, cs, vs] = sacarx( fe, ce, ve, x )
       l = length(fe);
       if(l ==0)
           fs = [];
           cs = [];
           vs = [];
       else
           [fs,cs,vs] = sacarx( fe(2:l) , ce(2:l), ve(2:l), x )
           if ( ve(1) ~= x )
                fs = [fe(1), fs];
                cs = [ce(1), cs];
                vs = [ve(1), vs];
           end
       end
```