



# **Computación 1**

**- 2023-**

**Estructuras de datos I:**

**Polinomios y Matrices dispersas**

# Estructuras de datos

Organizar los datos en la memoria para un objetivo específico:

- Representar una estructura abstracta
- Realizar operaciones con los datos
- Almacenar los datos o transmitirlos
- Accederlos de forma eficiente

Ya vimos algunos ejemplos: **matrices y vectores**

Veamos dos ejemplos un poco más complejos:

- **Polinomios**
- **Matrices dispersas**

# Polinomios

## Elementos básicos

$$f(x) = a_0 x^N + a_1 x^{N-1} + a_3 x^{N-3} + a_2 x^{N-2} + \dots \\ + a_{N-2} x^2 + a_{N-1} x + a_N$$

Variable:  $x$

Coeficientes:  $a_i$ ,  $i = 0 .. N$

Grado:  $N$

# Polinomios

## Reglas de representación en Octave

- Los coeficientes ordenados en forma decreciente por su grado
- Completitud: deben estar TODOS los coeficientes, aún si su valor = 0  
(estructura *posicional*)

» a = [1 1 1]    % representa:  $x^2 + x + 1$

» a = [2 0 1 3]    %  $2x^3 + 0x^2 + x + 3$

# Polinomios

Operaciones: **Suma y resta**

$$\gg [1\ 1\ 1\ 1] + [3\ 2\ 1\ 0]$$

ans =

$$4\ 3\ 2\ 1 \quad \% = 4x^3 + 3x^2 + 2x + 1$$

¡Ambas representaciones deben ser de igual largo  
(cantidad de elementos)!

$$\gg [0\ 0\ 1\ 1] + [3\ 2\ 1\ 0] \quad \% (x+1) + (3x^3 + 2x^2 + x)$$

ans =

$$3\ 2\ 2\ 1 \quad \% \text{ la resta es análoga}$$

# Polinomios

## Operaciones: **Producto**

### Polinomio x escalar

»  $[3 \ 2 \ 1 \ 0] * 3$       %  $(3x^3 + 2x^2 + x) 3$

ans =

9    6    3    0    %  $9x^3 + 6x^2 + 3x$

### Polinomio x polinomio Ej: $(x + 1)(3x^3 + 2x^2 + x)$

»  $[0 \ 0 \ 1 \ 1] * [3 \ 2 \ 1 \ 0]'$

ans =

1

**% ¡Este resultado no es correcto!**

# Polinomios

## Operaciones: **Producto**

» conv( [ 0 0 1 1 ], [ 3 2 1 0 ] )

ans =

0 0 3 5 3 1 0

No es necesario que sean de igual largo:

» conv( [ 1 1 ], [ 3 2 1 0 ] )

ans =

3 5 3 1 0

**Objetivo:**

$$(x + 1)(3x^3 + 2x^2 + x)$$

$$3x^4 + 2x^3 + x^2$$

$$3x^3 + 2x^2 + x$$

$$-----$$
$$3x^4 + 5x^3 + 3x^2 + x + 0$$

# Polinomios

## Operaciones: **Cociente**

»  $[c, r] = \text{deconv}([3 \ 5 \ 3 \ 1 \ 0], [3 \ 2 \ 1 \ 0])$

c =

1 1

r =

0 0 0 0 0

»  $[c, r] = \text{deconv}([3 \ 5 \ 3 \ 1 \ 0], [1 \ 1])$

c =

3 2 1 0

r =

0 0 0 0 0

% El resultado se devuelve en 2 vectores (cociente y resto)

% vectores completos

% Si se recibe el resultado en un vector sólo obtenemos el cociente

# Polinomios

## Operaciones: Raíces

Octave provee una función que halla las raíces de polinomios con una precisión determinada; puede no ser la que le sirve al usuario.

Éste deberá verificar si precisión y tiempo de cálculo se adecúan a su problema

(en Métodos Numéricos se verán algunas alternativas)

```
» roots( [ 4 2 1 ] )  
ans =
```

```
-0.2500 + 0.4330i  
-0.2500 - 0.4330i
```

```
» roots( [ 8 4 2 1 ] )  
ans =
```

```
-0.5000 + 0.0000i  
0.0000 + 0.5000i  
0.0000 - 0.5000i
```

# Polinomios

## Operaciones: `poly()` Construir un polinomio

La función inversa a hallar las raíces es construir un polinomio que tenga raíces dadas:

```
» r1 = roots( [ 4 2 1 ] )
```

```
r1 =
```

```
-0.2500 + 0.4330i
```

```
-0.2500 - 0.4330i
```

```
» roots( [ 1 0.5 0.25 ] )
```

```
ans =
```

```
-0.2500 + 0.4330i
```

```
-0.2500 - 0.4330i
```

```
» poly( r1 )
```

```
ans =
```

```
1.0000 0.5000 0.2500
```

# Polinomios

Operaciones: **polyval()** **Evaluar un polinomio**

Argumentos: polinomio y un escalar

```
» polyval( [ 4 2 1 ], 2 )
```

```
ans =
```

```
21
```

Argumentos: polinomio y una matriz

```
» polyvalm( [3, 2, 1], [1, 0; 0, 1] )
```

```
ans =
```

```
6 0
0 6
```

# Polinomios

Operaciones: `polyder()` Deriva un polinomio

» `polyder( [ 4 2 1 ] )`

`ans =`

`8 2`

Operaciones: `polyint()` Integra un polinomio

» `polyint( [4, 2] )`

`ans =`

`2 2 0`

# Polinomios

**Ejercicio: Evaluar un polinomio (polyval)**

**Ej:  $p(x) = x^4 + 3x^2 + x + 1$**

**$P = [1, 0, 3, 1, 1]$**

**$\text{polyval}(P, 2)$ :**

**$1 + 2(1 + 2(3 + 2(0 + 2(1)))) = 1 + 2 * 1 + 2^2 * 3 + 2^3 * 0 + 2^4 * 1$**

# Polinomios

## Ejercicio: Evaluar un polinomio (polyval)

```
function f = polyval(p,x)
    lp=length(p);
    f=0;
    for i=1:lp
        f=x*f+p(i);
    end
    %for i=1:lp
    % f=f+x^(lp-i)*p(i);
    %end
```

```
function f = polyval(p,x)
    lp=length(p);
    if lp==0
        f=0;
    else
        f=p(lp)+
            x*polyval(p(1:lp-1),x);
    end
```

# Polinomios

## Operaciones: **Resumen**

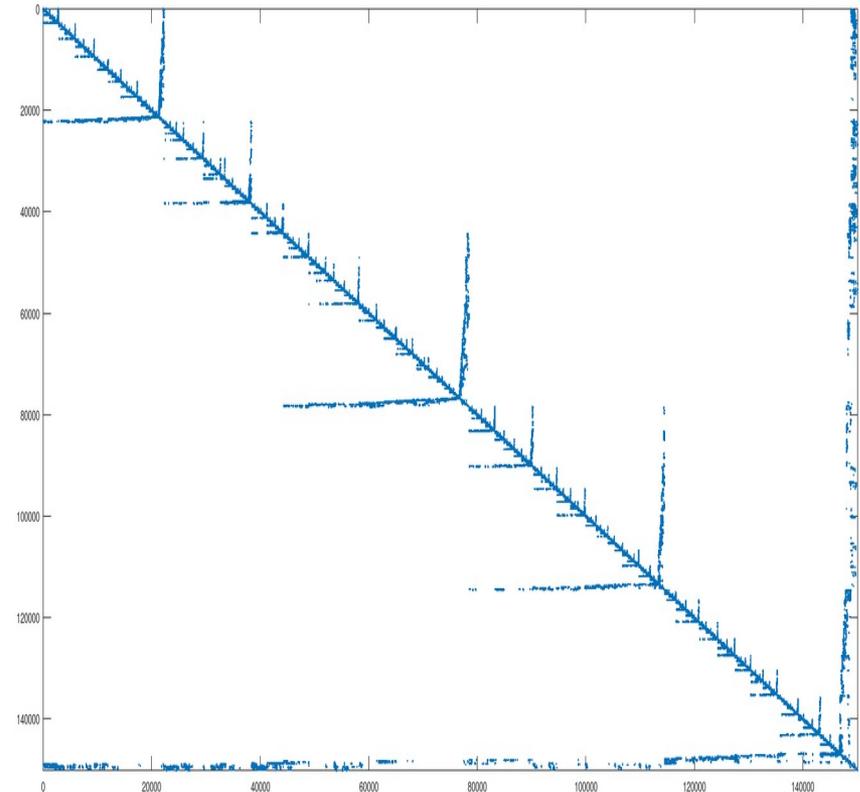
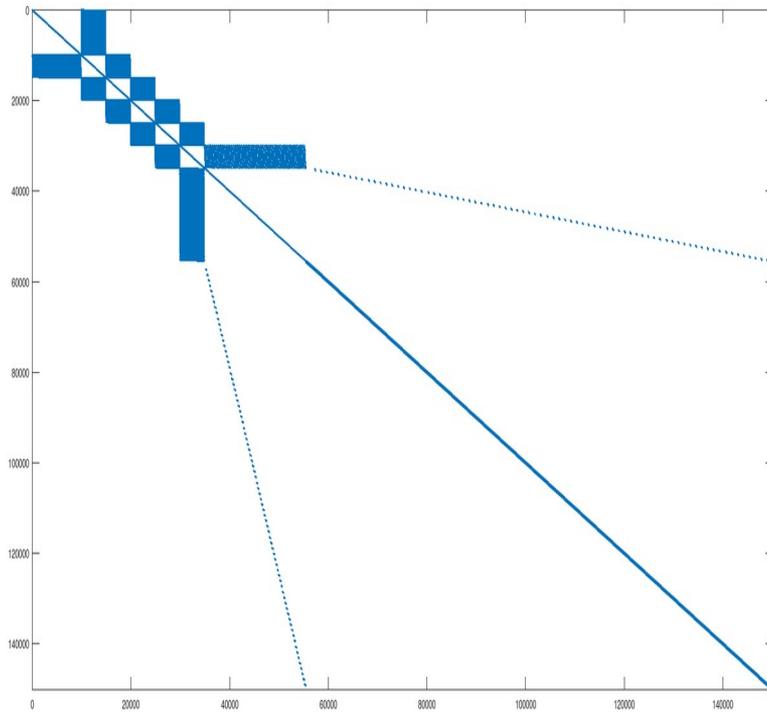
- Se representan usando vectores
- En algunos casos las operaciones de vectores resuelven correctamente las operaciones con polinomios
  - Suma (y resta)
  - Producto de un polinomio por un escalar

# Polinomios

## Operaciones: **Resumen**

- En otros casos hay funciones específicas:
  - Producto (y cociente) entre polinomios
  - Raíces (construcción de polinomio)
  - Evaluar polinomios
  - Derivar
  - Integrar

# Matrices dispersas



# 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-.

# 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

Si se almacena por filas la posición en bits de la celda  $(i, j)$  en la memoria es:

$$\text{inicio\_matriz} + (((i - 1) * n) + (j - 1)) * (4 * 8)$$

# 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*       $i = \text{floor}(\text{posición\_de\_celda}-1 / n) + 1$

*Columna*       $j = \text{mod}(\text{posición\_de\_celda}-1, n)+1$

# 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

## 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  $\text{mat}(3, 1) = 100$

□  $\text{vector\_f}(n) = 3$

□  $\text{vector\_c}(n) = 1$

□  $\text{vector\_d}(n) = 100$

**¿quién es n?**

**n = ordinal del elemento entre los distintos de 0**

# Matrices dispersas

Una forma de representación: costos

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

- En una matriz **completa**:  $(m \times n) * 4$
- En la matriz **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**:  $(m \times n) * 4$
- En la matriz **dispersa** máx.:  $(nz * 4) + 2 * nz * 2$

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

# 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  $k$  en el que se almacenan los valores de los coeficientes.
- Otro vector de tamaño  $k$  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

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

$$d = ( 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9 )$$

$$f = ( 1 \ 5 \ 7 \ 10 \ 12 \ 15 \ 18 \ 19 )$$

$$c = ( 1 \ 2 \ 5 \ 6 \ 2 \ 4 \ 2 \ 3 \ 4 \ 1 \ 4 \ 2 \ 5 \ 6 \ 3 \ 5 \ 6 \ 7 )$$

# Matrices dispersas

forma de representación

- Formato CCS (Compressed Column Storage)
  - Igual al CRS pero almacena por columna
- Muchos otros formatos...
  - DIA (Diagonal Storage Format)
  - Ellpack-itpack
  - BCSR
  - Estrategias dinámicas

# 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` no se usa pero está por compatibilidad con MATLAB (es el espacio pre-reservado para no-ceros de la matriz)

# 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

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:

- $A_{\text{disp}} \text{ oper } B_{\text{disp}} \rightarrow R_{\text{disp}}$
- $A_{\text{comp}} \text{ oper } B_{\text{comp}} \rightarrow R_{\text{comp}}$
- $A_{\text{disp}} \text{ oper } B_{\text{comp}} \rightarrow R_{\text{comp}}$  (excepto que de antemano se sepa que será dispersa)

# Matrices Dispersas

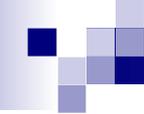
## Funciones

- `speye` - Matriz identidad dispersa.
- `find` - Devuelve los índices 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

## 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
end
```