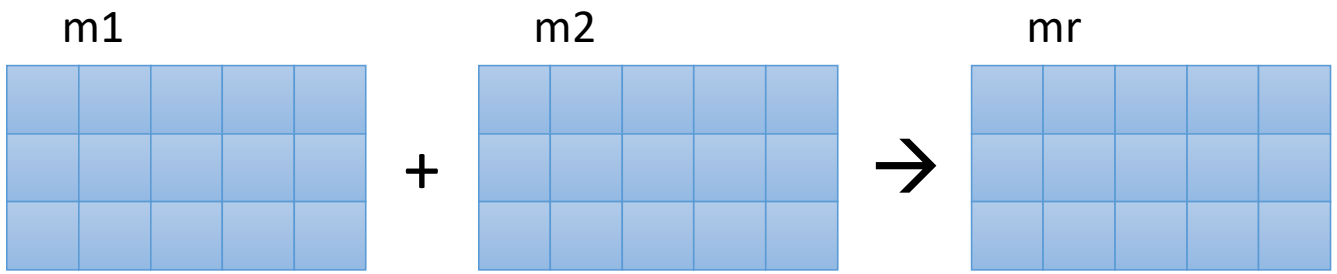
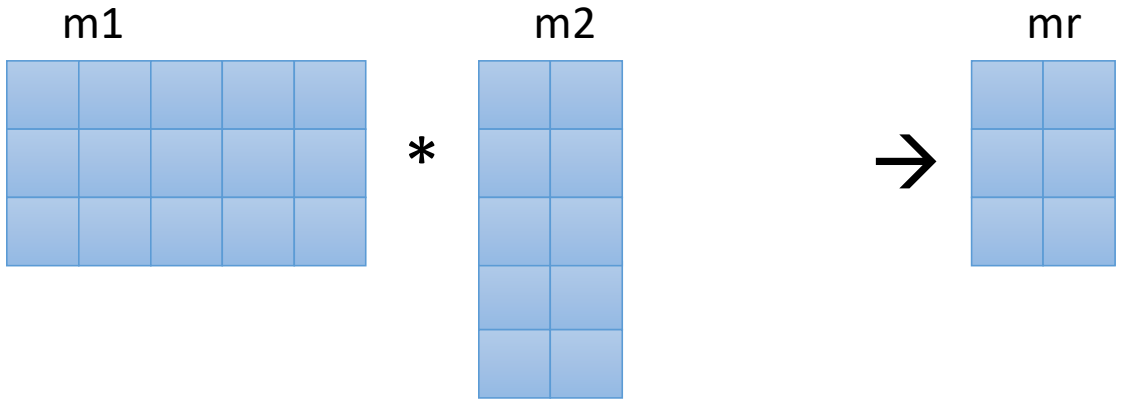
OPERACIONES MATRICIALES



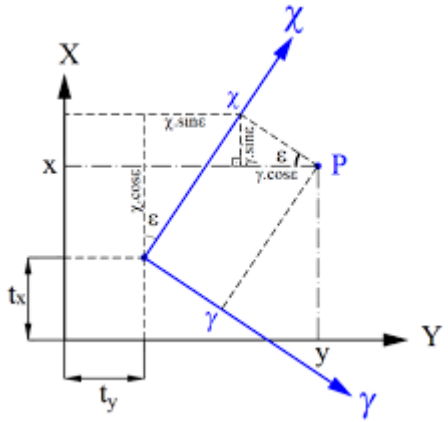


- Las matrices tienen tamaños muy variados
- Hay muchas operaciones que se pueden realizar sobre matrices (sumar, producto, inversa, transpuesta, …)
- Realizar programas que manipulen matrices es de interés pero requiere mucho trabajo de base

OPERACIONES MATRICIALES. ¿Para qué?

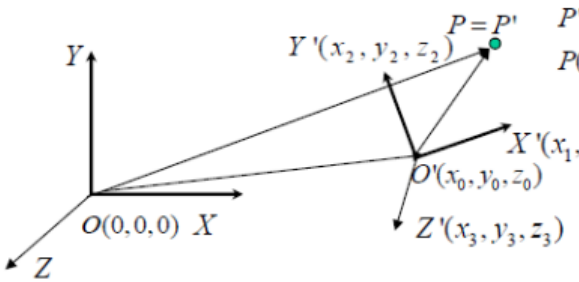
https://motion.cs.illinois.edu/RoboticSystems/CoordinateTransformations.html

Conversión de sistemas de coordenadas 2D



$$\begin{bmatrix} \mathcal{X} \\ \gamma \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} . \begin{bmatrix} \mathrm{X} \\ \mathrm{Y} \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

Conversión de sistemas de coordenadas 3D

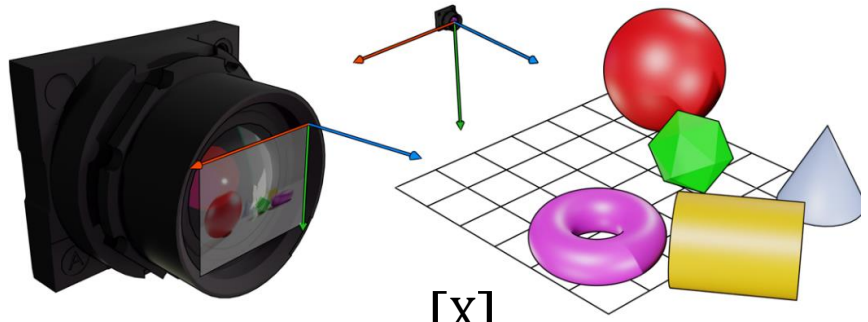


$P'(x', y', z')$, coordinate of the point P' in MC: O'X'Y'Z'

$P(x, y, z)$, coordinate of the point P in WC: OXYZ

$$\begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} . \begin{bmatrix} \mathrm{X} \\ \mathrm{Y} \\ Z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

Transformación proyectiva en una cámara: 3D → 2D



$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = [Mt] . \begin{bmatrix} \mathrm{X} \\ \mathrm{Y} \\ Z \\ 1 \end{bmatrix}$$

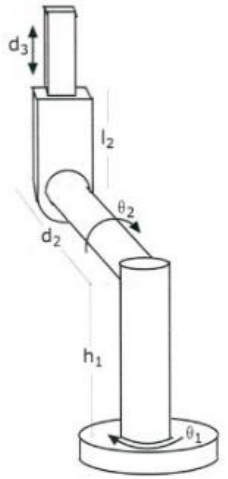
OPERACIONES MATRICIALES. ¿Para qué?

Modelos y control en el espacio de estados

$$\frac{d}{dt}\begin{bmatrix}\theta\\ \dot{\theta}\\ i\end{bmatrix}=\begin{bmatrix}0 & 1 & 0\\ 0 & -\frac{b}{J} & \frac{K}{J}\\ 0 & -\frac{K}{L} & -\frac{R}{L}\end{bmatrix}\begin{bmatrix}\theta\\ \dot{\theta}\\ i\end{bmatrix}+\begin{bmatrix}0\\ 0\\ \frac{1}{L}\end{bmatrix}V$$

$$y=\begin{bmatrix}1 & 0 & 0\end{bmatrix}\begin{bmatrix}\theta\\ \dot{\theta}\\ i\end{bmatrix}$$

Cinemática directa e inversa



$${}^{0}_{1}T=\begin{bmatrix}c\theta_1 & -s\theta_1 & 0 & 0\\ s\theta_1 & c\theta_1 & 0 & 0\\ 0 & 0 & 1 & h_1\\ 0 & 0 & 0 & 1\end{bmatrix}$$

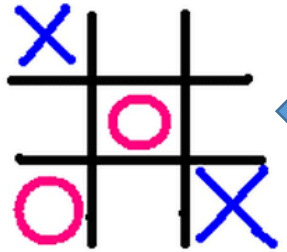
$${}^{1}_{2}T=\begin{bmatrix}c\theta_2 & -s\theta_2 & 0 & 0\\ 0 & 0 & -1 & -d_2\\ s\theta_2 & c\theta_2 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$${}^{2}_{3}T=\begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & (l_2+d_3)\\ 0 & -1 & 0 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

$${}^{0}_{3}T=\begin{bmatrix}c\theta_1c\theta_2 & s\theta_1c\theta_2 & s\theta_2 & -s\theta_2h_1\\ -c\theta_1s\theta_2 & -s\theta_1s\theta_2 & c\theta_2 & -c\theta_2h_1\\ s\theta_1 & -c\theta_1 & 0 & -d_2\\ 0 & 0 & 0 & 1\end{bmatrix}$$

OPERACIONES MATRICIALES. ¿Para qué?

Tablero de juego 3 en raya



$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 2 & 0 & 1 \end{bmatrix}$$

Resolución de ecuaciones

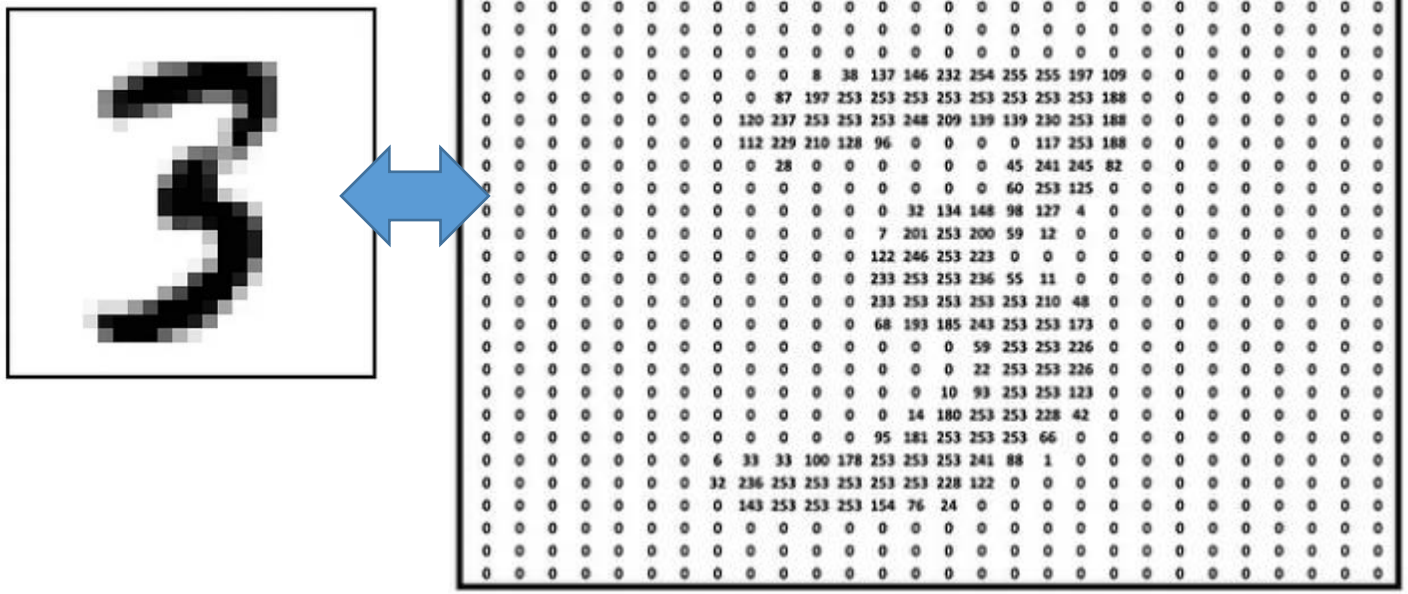
$$\begin{aligned} 2x_1 - x_2 &= 1 \\ -x_1 + 2x_2 - x_3 &= 1 \\ -x_2 + 2x_3 - x_4 &= 1 \\ -x_3 + 2x_4 &= 1 \end{aligned}$$

A x b

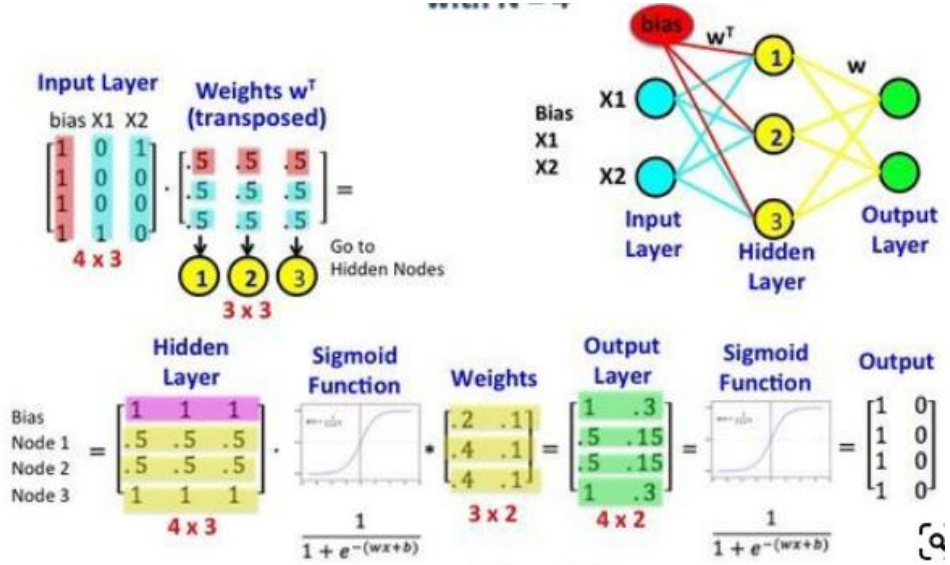
$$\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

x=A\B

Tratamiento de imagen



Machine learning y Neural Networks



OPERACIONES MATRICIALES. Usando Matlab

Resolución de ecuaciones

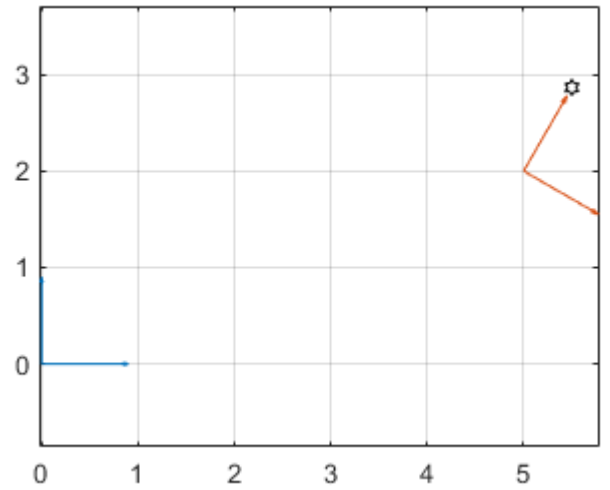
$$\begin{cases} 2x_1 - x_2 = 1 \\ -x_1 + 2x_2 - x_3 = 1 \\ -x_2 + 2x_3 - x_4 = 1 \\ -x_3 + 2x_4 = 1 \end{cases} \quad \underset{A}{\begin{bmatrix} 2 & -1 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & -1 & 2 \end{bmatrix}} \underset{x}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}} = \underset{b}{\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}}$$

x=A\B

```
>> A=[2,-1,0,0 ; -1, 2, -1, 0; 0,-1,2,-1;0,0,-1,2];
>> B=ones(4,1);
>> x=A\B;
>> disp(x)
    2.0000
    3.0000
    3.0000
    2.0000
>> disp(A*x)
    1.0000
    1.0000
    1.0000
    1.0000
```

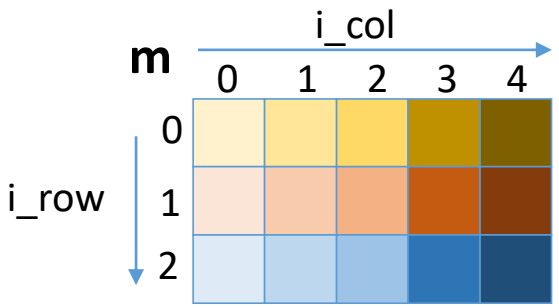
Cambio de coordenadas

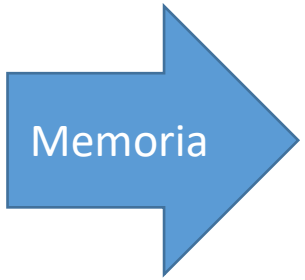
```
>> ang=30*pi/180;
>> mR=[cos(ang),sin(ang)];[-sin(ang),cos(ang)];
>> mT=[5;2];
>> figure; quiver([0,0],[0,0],[0,1],[1,0]);
>> hold on; axis equal; grid on;
>> quiver(mT(1)+[0,0],mT(2)+[0,0],mR(1,:),mR(2,:));
>> xy2=[0;1]; // Punto x=0,y=1 en sistema 2
>> xy1=mR*xy2+mT;  // Punto convertido a sistema 1
>> plot(xy1(1),xy1(2),'hk');
```



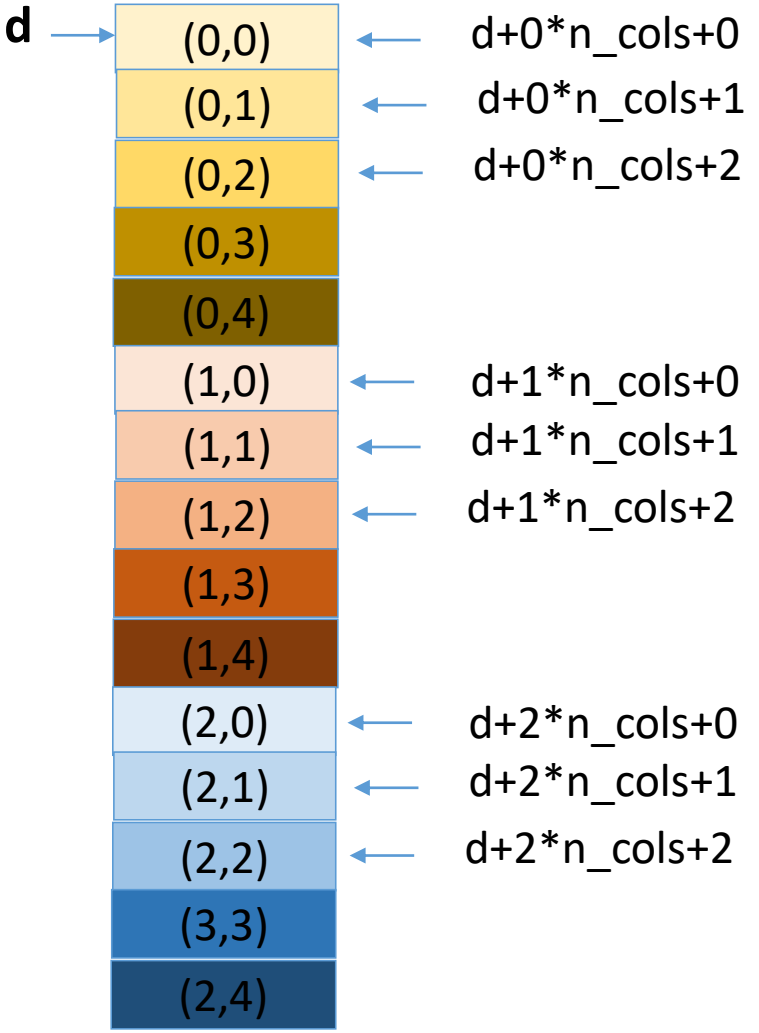
VARIABLES NECESARIAS PARA MANIPULAR UNA MATRIZ:

n_rows → int
n_cols → int
data → array (n_rows x n_cols) de float : data[i,j] es un float
El tamaño del array es muy variable
El acceso al elemento de la posición i,j debe ser 1D,
ya que la memoria se organiza en direcciones en 1D





m(i,j) → d[i*n_cols+j]



AL MANIPULAR UNA MATRIZ:

- Todas las funciones que manejen matrices necesitan todas las variables para cada matriz

```
int m1_n_rows, m1_n_cols;
float *m1_data;
int m2_n_rows, m2_n_cols;
float *m2_data;
int mr_n_rows, mr_n_cols;
float *mr_data;
```

- Todas las funciones que manejen matrices deben asignar memoria dinámica para los elementos de esa matriz, y liberarla cuando ya no se use:

```
m2_data=(float*) malloc(m2_n_rows*m2_n_cols*sizeof(float));
...
free(m2_data);
```

- Todas las funciones que manejen matrices necesitan 3 variables por matriz:

```
float CalcMax(int nr,int nc,const float d[] );
...
float vmax=CalcMax(m2_n_rows,m2_n_cols,m2_data);
```

- Todas las funciones que manejen matrices necesitan bucles que deben acceder correctamente a los elementos de cada matriz:

`m2_data[i*m1_n_cols+j]` es incorrecto

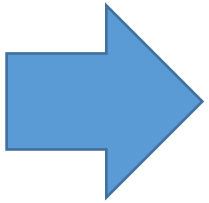
- Todas las funciones que den como resultado matrices deben generar y devolver 3 valores:

```
void Sum(int nr1,int nc1,const float d1[], ... , int* nr_result,int* nc_result,float** d_result);
...
Sum(m1_n_rows,m2_n_rows,m1_data, ... , &mr_n_rows,&mr_n_cols, &mr_data);
```

PASO 1: Usar estructuras simplifica el trabajo

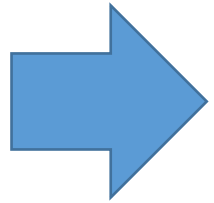
- Todas las funciones que manejen matrices necesitan todas las variables para cada matriz

```
int m1_n_rows, m1_n_cols;
float *m1_data;
int m2_n_rows, m2_n_cols;
float *m2_data;
int mr_n_rows, mr_n_cols;
float *mr_data;
```



```
struct matriz {
  int n_rows,n_cols;
  float* data;
} ;

struct matriz m1,m2,mr;
```



La vble m1 tiene los campos:
- m1.n_rows (int)
- m1.n_cols (int)
- m1.data (float*)

La vble m2 tiene los campos:
- m2.n_rows(int)
- …

- Todas las funciones que manejen matrices deben asignar memoria dinámica para los elementos de esa matriz, y liberarla cuando ya no se use:

```
m2.data=(float*) malloc(m2.n_rows*m2.n_cols*sizeof(float));
...
free(m2.data);
```

- Todas las funciones que manejen matrices necesitan 1 sola variable que ya contiene a todas:

```
float CalcMax(struct matriz m);
...
float vmax=CalcMax(m2);
```

- Todas las funciones que manejen matrices necesitan bucles que deben acceder correctamente a los elementos de cada matriz:

`m2.data[i*m1.n_cols+j]` es incorrecto

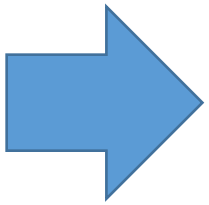
- Todas las funciones que den como resultado matrices pueden generar y devolver una struct matriz:

```
struct matriz Sum(struct matriz a,struct matriz b);
...
mr=Sum(m1,m2);
```

PASO 2: Usar clases simplifica el trabajo

- Todas las funciones que manejen matrices necesitan todas las variables para cada matriz

```
Matrix m1,m2,mr;
```



La vble m1 tiene los campos:
- m1.n_rows (int)
- m1.n_cols (int)
- m1.data (float*)

La vble m2 tiene los campos:
- m2.n_rows(int)
- …

Pero además se pueden aplicar funciones y operadores directamente a m1, m2 mr

- Las funciones de la clase asignan memoria dinámica para los elementos de cada matriz, y liberan cuando ya no se use:

```
Matrix m1(3,5); → Crea una matriz de 3x5, y asigna memoria automáticamente para ella
                → Cuando la vble m1 deje de ser necesaria, se elimina automáticamente la memoria asignada
```

- Todas las funciones que manejen matrices se declaran dentro de la clase:

```
...
float vmax=CalcMax(m2);
```

- Sólo las funciones de la clase tienen libre acceso a los elementos:

`m2.data[i*m1.n_cols+j]` es imposible → Ahora sólo la función data() tiene acceso a los elementos: `m2.data(i,j)`

- La clase puede incluso redefinir operadores, como la suma:

```
mr=m1+m2;
```

TODO EL CÓDIGO SE SIMPLIFICA Y ESTÁ MENOS SUJETO A ERRORES
LAS CLASES SON REUTILIZABLES EN MÚLTIPLES PROYECTOS
ES MÁS SENCILLO REALIZAR PROGRAMAS EN QUE INTERVENGAN VARIOS DESARROLLADORES

Un ejemplo de uso:

- ❑ Descargar MyMatrix.cpp y MyMatrix.h en el directorio del proyecto
- ❑ Añadir archivos al proyecto
- ❑ Un main() sencillo:

```
#include <iostream>
#include "MyMatrix.h"

int main()
{
    MyMatrix m1(2,3),mr;

    std::cout << "Introduzca matriz m1 de 2x3:\n";
    std::cin >> m1;  // Pide los datos de m1 por consola

    std::cout << "m1:\n" << m1 << std::endl; // Escribe el contenido de m1 por consola

    mr=m1+m1-2*m1;   // Calcula m1+m1-2*m1 (=  matriz de 2x3 ceros) y asigna a mr
    std::cout << "m1+m1+2*m1:\n" << mr << std::endl; // Escribe el contenido de mr por consola

    std::cout << "m1*m1':\n" << m1*m1.transpose() << std::endl; // Calcula m1*m1' y escribe el resultado por consola

    return 0;
}
```

- ❑ Explicación paso a paso (casos similares):

https://nms.kcl.ac.uk/john.armstrong/cppbook/cpp-website.html (chap 16)
https://cw.fel.cvut.cz/b212/_media/courses/b3b36prg/lectures/b3b36prg-lec11-handout-2x2.pdf