Universidad de Alcalá

PRÁCTICA 0

SISTEMAS DE CONTROL INTELIGENTE

Introducción a MATLAB

Grado en Ingeniería Informática

David Barreiro Zambrano

Natalia Montoya Gómez

ÍNDICE

PARTE I

EJERCICIO 1. Matrices y vectores

Realice un script de Matlab que permita desarrollar una serie de operaciones con una matriz:

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \qquad v = \begin{bmatrix} 14 \\ 16 \\ 18 \\ 20 \end{bmatrix}$$

En primera estancia, creamos un script que nos permite escribir todas las operaciones. Lo ubicamos en la carpeta en la que vamos a trabajar con los scripts de Matlab.

1. *Cree la siguiente matriz A y el vector v.*

Para crear la matriz y el vector, simplemente inicializamos las dos variables como corresponde. Al final de cada instrucción, podemos o no, poner un punto y coma. En caso de que lo pongamos, el resultado de la instrucción no saldrá por pantalla. Pero como en este caso no especifica que tengamos que visualizar las variables creadas, ponemos el punto y coma para que no lo muestre por pantalla.

```
A = [1 2 ; 3 4 ; 5 6 ; 7 8];
v = [14 ; 16 ; 18 ; 20];
```

De esta forma, los signos de puntuación punto y coma (;) separan las distintas filas de la matriz.

2. *Obtenga y visualice una matriz B concatenando la matriz A y el vector v.*

Para concatenar creamos una nueva variable llamada *B* para que nos guarde el resultado de la concatenación en algún sitio, y concatenamos horizontalmente, ya que no podemos concatenar ambas variables de forma vertical porque sus dimensiones no nos lo permiten.

```
>> B = [A v]

B =

     1     2    14
     3     4    16
     5     6    18
     7     8    20
```

3. *Obtenga y visualice un vector fila resultado de concatenar las filas de la matriz B.*

Cuando queremos que todo salga en un vector único, no ponemos punto y comas para separar filas, ya que tendremos una única fila.
Cogemos la matriz B e indicamos las filas, cogiendo todos los elementos de cada fila, con la instrucción: *(fila,:)*

```
>> vfila = [B(1,:) B(2,:) B(3,:) B(4,:)]

vfila =

     1     2    14     3     4    16     5     6    18     7     8    20
```

4. *Obtenga y visualice un vector columna resultado de concatenar las columnas de la matriz B.*

De la misma forma que en el ejercicio anterior, con la instrucción *(:,columna)* podemos coger cada una de las columnas de la matriz, con todos sus elementos, y concatenarlas de forma vertical. Esta vez sí que tenemos que hacer uso del punto y coma.

```
>> vColumna = [B(:,1) ; B(:,2) ; B(:,3)]

vColumna =

     1
     3
     5
     7
     2
     4
     6
     8
    14
    16
    18
    20
```

EJERCICIO 2. Matrices y vectores

Realice un script de Matlab que permita desarrollar una serie de operaciones con una matriz:

1. *El script ha de generar una matriz, cuadrada y aleatoria de tamaño indicado por el usuario. En la línea de comandos se ha de visualizar el mensaje: "Indique el tamaño de la matriz".*
   Funciones recomendadas: input, rand.

De los comandos recomendados, utilizamos ambos:
La función *input* muestra el texto en *prompt* y espera a que el usuario introduzca un valor y pulse la tecla Return.
La función *rand(n)* devuelve una matriz cuadrada de nxn de números aleatorios distribuidos de manera uniforme. Esta función a su vez posee muchas otras funcionalidades, pero la que nos interesa para este ejercicio es la indicada.

```
tamano = input("Indique el tamaño de la matriz: ");
matriz = rand(tamano);
```

2. *A partir de la matriz construida, el script deberá calcular y presentar por pantalla los siguientes datos:*
   Funciones recomendadas: disp, for, diag, det, max, min, mean, var, figure, plot, hold on, hold off, title, xlabel, ylabel, size.

   a. *Matriz generada.*

```
 matriz
matriz =

    0.7317    0.7447    0.6256    0.4868    0.5108
    0.6477    0.1890    0.7802    0.4359    0.8176
    0.4509    0.6868    0.0811    0.4468    0.7948
    0.5470    0.1835    0.9294    0.3063    0.6443
    0.2963    0.3685    0.7757    0.5085    0.3786
```

La matriz generada tiene números aleatorios entre (0,1).
En caso de que quisiéramos otro rango de valores de la matriz, tendríamos que utilizar la función *randi*.

   b. *Una segunda matriz formada por las columnas impares de la matriz inicial.*

Para poder realizar la selección de las columnas impares, tendremos que recorrer la matriz desde la columna inicial, hasta la columna final, con salto de 2. Así recorrerá la columna 1, 3, 5, … , n.
Para ello, declaramos la matriz vacía, llamada *matrizImpares*.

```
matrizImpares = [];
for i = 1:2:tamano
    matrizImpares(:,end+1) = matriz(:,i);
end
matrizImpares
```

```
matrizImpares =

    0.7317    0.6256    0.5108
    0.6477    0.7802    0.8176
    0.4509    0.0811    0.7948
    0.5470    0.9294    0.6443
    0.2963    0.7757    0.3786
```

A la hora de desarrollar el *for* y de asignar las columnas a la matriz objetivo, en un primer momento, pensé que lo que tendría que haber dentro del bucle, sería la siguiente línea:
matrizImpares(:,i) = matriz(:,i)
Pero el resultado de la matriz era el siguiente:

```
0.7317         0    0.6256         0    0.5108
0.6477         0    0.7802         0    0.8176
0.4509         0    0.0811         0    0.7948
0.5470         0    0.9294         0    0.6443
0.2963         0    0.7757         0    0.3786
```

Es decir, una matriz con 5 columnas, las pares, rellenadas por 0s. Este no era el resultado que quería obtener, por tanto, después de darle un par de vueltas, la instrucción era la siguiente:
matrizImpares(:,end) = matriz(:,i)
El problema ahí es el end, ya que, al haber declarado la matriz, estaba vacía, por tanto, end tendría un valor nulo, por ello no podía meter nada dentro de la matriz, porque no tenía ningún sitio al que asignarle la columna.
Por último, entendí que *end* es como *length*, así fue más fácil comprender el problema. Puesto que, en la primera iteración del bucle *for*, *end* tomaría el valor 0, por tanto, hay que sumarle 1 para que en la primera columna de la matriz de impares podamos meter la columna número 1 de la matriz original. En la segunda iteración, *end* toma el valor de 1, por tanto, le sumamos uno más para que en la segunda columna de la matriz de impares, se añada la columna número 3 de la matriz original, y así sucesivamente, hasta terminar con todas las columnas impares de la matriz original.

c. *El valor de los elementos de la diagonal de la matriz generada.*

Usamos dos funciones útiles que muestra el enunciado (*diag*), que calcula la diagonal de una matriz. Y *disp* nos muestra los valores de una variable.

```
disp(diag(matriz))
```

```
0.7317
0.1890
0.0811
0.3063
0.3786
```

d. *Valor máximo, mínimo, medio y varianza de cada fila. Estos valores se han de representar gráficamente, indicando en el eje de abscisas el número de fila.*

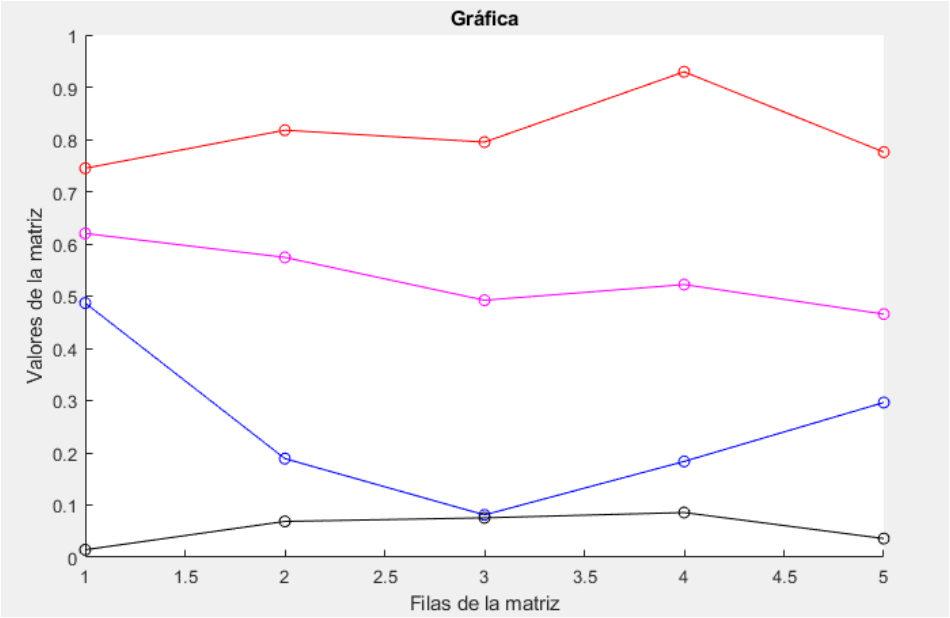
Teniendo en cuenta que estos valores los tenemos que representar gráficamente, pensamos en que la forma más fácil de hacerlo es creando vectores, inicializados a 0 en una primera instancia. Después se van rellenando a medida que se va recorriendo cada fila de la matriz.

```
vectorMax = zeros();
vectorMin = zeros();
vectorMean = zeros();
vectorVar = zeros();

for i = 1:tamano
    vectorMax(i) = max(matriz(i,:));
    vectorMin(i) = min(matriz(i,:));
    vectorMean(i) = mean(matriz(i,:));
    vectorVar(i) = var(matriz(i,:));
end
```

Una vez tenemos los vectores rellenos de cada una de las operaciones estadísticas, podemos hacer las gráficas. Para ello, utilizamos la función *plot*, encarada de crear gráficas bidimensionales.

Pero, antes de nada, tenemos que crear una ventana para mostrar el gráfico, con la función *figure*.

```
f = figure;
plot(1:tamano, vectorMax, 'o-r');
hold on
plot(1:tamano, vectorMin, 'o-b');
plot(1:tamano, vectorMean, 'm-o');
plot(1:tamano, vectorVar, 'o-k');
hold off
title('Gráfica')
xlabel('Filas de la matriz')
ylabel('Valores de la matriz')
```



Como podemos observar, el *hold on* es una función que trabaja sobre una función *plot*, por lo que, en el código, ponemos primero un plot sobre el que va a trabajar. Después del *hold on*, ponemos a continuación los siguientes *plot*, cerramos el *hold on* con el *hold off*. En cada uno de los *plot*, indicamos:

- En qué rango de valores tiene que pintar la gráfica
- El nombre del vector que queremos representar
- Forma de representación de los puntos y color de la función

Utilizamos también la función *title*, *xlabel* e *ylabel* para poner comentarios en el eje de abscisas y coordenadas, y seleccionar un título. Todo esto lo ponemos después del *hold off*.

Los problemas que pudimos encontrar fueron el entendimiento de las funciones *hold on* y *hold off*, hasta que nos dimos cuenta de que *hold on* trabaja sobre un *plot* y a raíz de ese escribe los siguientes que están comprendidos entre el *on* y el *off*.

EJERCICIO 3. Matrices y vectores

Programe un script en Matlab que permita realizar una serie de operaciones con dos matrices (**A** y **B**) que se introducirán por teclado. Para ello:
Funciones útiles: input, for, if..else, det, inv, pinv, rand, randn, rank, size.
En caso de que no sea posible realizar alguno de los cálculos solicitados, indíquelo por pantalla.

1. Solicite al usuario las dimensiones de las matrices en formato [filas cols], (si se introduce un único número, la matriz será cuadrada).
2. Genere dos matrices (A y B) de las dimensiones elegidas. Para rellenar las matrices, escriba una función en Matlab (en un archivo diferente) que reciba como parámetro las dimensiones deseadas [filas cols], y devuelva la matriz rellena:

   function Matriz = IntroducirMatriz(Dimensiones)
3. La función debe pedir datos al usuario para cada posición de la matriz. En caso de que el usuario escriba 'r', la matriz se rellenará de valores aleatorios.

Hemos tenido que realizar primero en conjunto los apartados 1 y 2, debido a la estrecha relación que hay entre ellos.
El primer paso del ejercicio es pedir las dimensiones de las matrices al usuario, para ello utilizamos la función *input* ya utilizada en apartados anteriores. Una particularidad del *input* en Matlab es que, si el usuario introduce varios números, se le obliga a indicar los corchetes, para que luego Matlab pueda interpretarlo. De esta manera, nos ahorramos trabajo, ya que no hay que comprobar si el formato es correcto.
Este procedimiento de pedir al usuario se realizará dos veces, una por cada una de las matrices pedidas: (**A y B**).
Como especifica el enunciado, hay que comprobar si se introduce un solo número, por lo que la matriz sería cuadrada, lo cual comprobamos con la función *size*. En caso de que se introdujeran varios datos, despreciamos los demás datos que no sean el primero y el segundo.
Por lo que si al introducir las dimensiones, ponemos: [2 3 4 6 7], el *input* solo cogerá los dos primeros valores (el 2 y el 3) en este caso. Esto lo podemos ver en el código reflejado con el condicional que comprueba si el tamaño de la dimensión de la matriz está formado por uno o varios números, en caso de que esté formado por varios números, llama a la función *IntroducirMatriz* con los parámetros siguientes:

- Valor de *dimension* en la posición 1
- Valor de *dimension* en la posición 2

Hacemos las llamadas a la función correspondientes.
A función *IntroducirMatriz* le entran las filas y las columnas que el usuario quiere para la matriz. Con la función *rand(x, y)*, devuelve un arreglo de *x* por *y*, de números aleatorios.
Después, con los dos bucles *for* se rellena la matriz con cada uno de los valores.
La función *strcat* la usamos para concatenar texto horizontalmente, la función *num2str* para convertir números en un arreglo de caracteres.
Como dice el enunciado, si la entrada del usuario es el caracter *r*, entonces la variable *valor* se quedará con el valor random definido anteriormente. Por ello, comprueba con el operador ~= si el valor que ha introducido el usuario no es *r*, entonces convierte el *str* en que lo había convertido anteriormente, a doble. Ya que el texto *str* contiene valores numéricos reales o complejos.

```
dimension = input('Introduzca la dimensión de la matriz A [filas cols]: ');

if size(dimension) == 1
    matrizA = IntroducirMatriz(dimension(1), dimension(1));
else
    matrizA = IntroducirMatriz(dimension(1), dimension(2));
end

dimension = input('Introduzca la dimensión de la matriz B [filas cols]: ');

if size(dimension) == 1
    matrizB = IntroducirMatriz(dimension(1), dimension(1));
else
    matrizB = IntroducirMatriz(dimension(1), dimension(2));
end
```

Función:

```
function [Matriz] = IntroducirMatriz(numfil, numcol)
    Matriz = rand(numfil, numcol);
    for i = 1:numfil
        for j = 1:numcol
            valor = input(strcat('Valor (', num2str(i), ',', ...
                num2str(j), '): '), 's');
            if num2str(valor) ~= 'r'
                Matriz(i, j) = str2double(valor);
            end
        end
    end
end
```

```
Introduzca la dimensión de la matriz A [filas cols]: [2 3]
Valor (1,1):2
Valor (1,2):4
Valor (1,3):3
Valor (2,1):4
Valor (2,2):r
Valor (2,3):6.7
Introduzca la dimensión de la matriz B [filas cols]: 3
Valor (1,1):5
Valor (1,2):3
Valor (1,3):4.5
Valor (2,1):3
Valor (2,2):r
Valor (2,3):4
Valor (3,1):2.2
Valor (3,2):1.9
Valor (3,3):9
```

La función me resultó complicada ya que al principio intentaba concatenar los caracteres con los valores *i* y *j* de la matriz, pero no era posible. Al igual que en la función *input*, hasta que no puse *'s'*, como parámetro de la función, no funcionaba, ya que evaluaba la entrada como expresión.

4. Calcule y muestre por pantalla:

- **a.** Las matrices generadas A y B

El comando *disp* nos permite visualizar las variables, de forma más estética, e imprimir por pantalla lo que el usuario desee.

```
disp('Matriz A')
disp(matrizA)
disp('Matriz B')
disp(matrizB)
```

```
Matriz A
    2.0000    4.0000    3.0000
    4.0000    0.6790    6.7000

Matriz B
    5.0000    3.0000    4.5000
    3.0000    0.6073    4.0000
    2.2000    1.9000    9.0000
```

b. La traspuesta e inversa de cada una de las matrices

Para hacer la traspuesta de una matriz se puede utilizar la operación de Matlab *(.')* o si no la función *transpose*, en ese caso, utilizamos ambas posibilidades.

```
disp('Traspuesta de la matriz A')
disp(transpose(matrizA))
disp('Traspuesta de la matriz B')
disp(matrizB.')
```

```
Matriz A
    2.0000    4.0000    3.0000
    4.0000    0.6790    6.7000

Matriz B
    5.0000    3.0000    4.5000
    3.0000    0.6073    4.0000
    2.2000    1.9000    9.0000
```

Para calcular la inversa de una matriz, utilizamos la función *inv()*, pero solo es calculable cuando la matriz es cuadrada, por tanto, con un *if* comprobamos si es cuadrada o no, en caso de que no lo sea, muestra un mensaje de que no es posible realizar la operación inversa sobre dicha matriz.

```
if size(matrizA,1) == size(matrizA,2)
    disp('Inversa de la matriz A: ')
    disp(inv(matrizA))
else
    disp('No se puede realizar la inversa de A porque no es cuadrada')
end

if size(matrizB,1) == size(matrizB,2)
    disp('Inversa de la matriz B')
    disp(inv(matrizB))
else
    disp('No se puede realizar la inversa de B porque no es cuadrada')
end
```

```
No se puede realizar la inversa de A porque no es cuadrada
Inversa de la matriz B
    0.0468    0.4043   -0.2031
    0.3988   -0.7692    0.1424
   -0.0956    0.0635    0.1307
```

c. El producto de A y B (matricial y elemento a elemento)

Para poder realizar la operación de producto de A y B, de forma matricial se puede realizar si el número de columnas de A coincide con el número de filas de B. En caso de que se haga el producto de ambas matrices elemento a elemento, el tamaño debe ser el mismo entre las dos matrices.
Por tanto, comprobamos primero la sentencia anterior, y a raíz de ahí, mostraremos el resultado utilizando las operaciones simples (*) y (.*), o simplemente lanzaremos un error.

```
if size(matrizA,2) == size(matrizB,1)
    disp('Producto matricial A y B: ')
    disp(matrizA*matrizB)
else
    disp('No se puede realizar el producto matricial de A y B')
end

if size(matrizA) == size(matrizB)
    disp(matrizA.*matrizB)
else
    disp('No se puede realizar el producto elemento a elemento de A y B')
end
```

```
Producto matricial A y B:
   28.6000   14.1292   52.0000
   36.7771   25.1424   81.0161

No se puede realizar el producto elemento a elemento de A y B
```

El problema aquí encontrado fue la multiplicación elemento por elemento, ya que pensaba que el producto matricial al igual que el producto elemento por elemento, tenían las mismas

condiciones, es decir, el número de columnas de la matriz A debía ser igual al número de filas de la matriz B.
Pero según la documentación de Matlab, esta operación solo se puede realizar si la ambas matrices tienen tamaños igual o compatibles, lo cual significa que una de ellas o las dos tengan tamaño 1.

d. Un vector fila obtenido concatenando la primera fila de cada una de las matrices

Para concatenar la primera fila de cada una de las matrices, lo hacemos de la siguiente forma:
Código

```
disp('Vector primera fila de A y B: ')
disp([matrizA(1,:) matrizB(1,:)])
```

```
Vector primera fila de A y B:
    2.0000    4.0000    3.0000    5.0000    3.0000    4.5000
```

e. Un vector columna obtenido concatenando la primera columna de cada de las matrices

Para concatenar la primera columna de ambas matrices, lo hacemos de la misma forma que en el apartado anterior, pero aquí haciendo uso del signo de puntuación *(;)*:

```
disp('Vector primera columna de A y B: ')
disp([matrizA(:,1) ; matrizB(:,1)])
```

```
Vector primera columna de A y B:
    2.0000
    4.0000
    5.0000
    3.0000
    2.2000
```

EJERCICIO 4. Tiempo de cómputo y representación gráfica

Realice un script en Matlab que permita obtener y representar el tiempo consumido para el cálculo del rango y el determinante de una matriz en función de su tamaño (entre 1x1 y 25x25). Tenga en cuenta que:

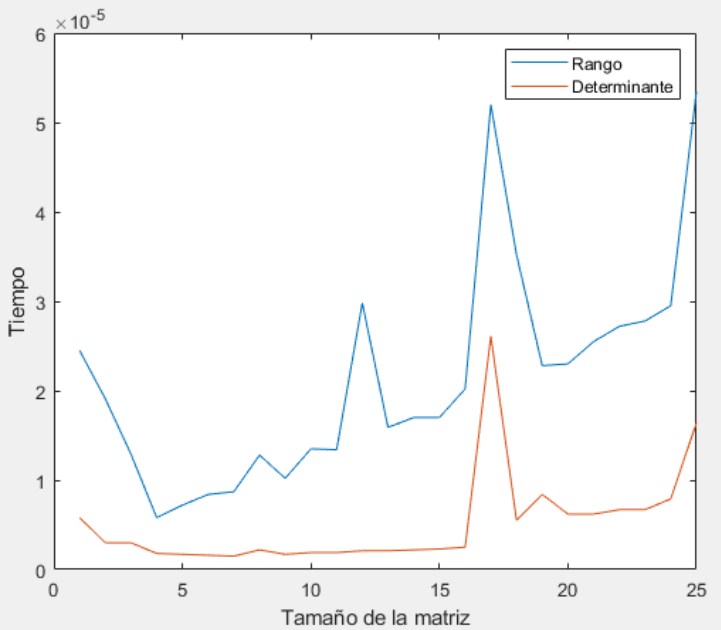
- La matriz se rellenará con valores aleatorios.
- El tiempo necesario para cada operación debe obtenerse por separado.
- Los tiempos de procesamiento para el cálculo del rango y del determinante se representarán en la misma gráfica, utilizando para ello diferentes colores.
- Deben añadirse etiquetas a los ejes, y una leyenda indicando que representa cada línea.

Funciones útiles: det, help, rand, randn, rank, tic, toc, title, xlabel, ylabel, plot.

Mediante un *for* iremos creando una matriz de valores aleatorios con la función *rand*, desde 1x1 hasta el tamaño máximo 25x25.
Creamos dos vectores para almacenar el tiempo del rango y del determinante. Y mediante los comando *tic* y *toc*, calculamos los tiempos en realizar ambas operaciones.
A la hora de las operaciones, las funciones que utilizamos son *rank()* y *det()*. La primera calcula el rango de la matriz, y la segunda calcula el determinante de la matriz.

Por ello, graficamos y obtenemos las diferencias entre los tiempos de hacer el rango y el determinante con distintos tamaños de matriz.

```
tiempoRango = [];
tiempoDeterminante = [];

for i = 1:25
    matriz = rand(i);
    tic;
    rank(matriz);
    tiempoRango(i) = toc;
    tic;
    det(matriz);
    tiempoDeterminante(i) = toc;
end

f = figure;
plot(1:25, tiempoRango);
hold on;
plot(1:25, tiempoDeterminante);
xlabel('Tamaño de la matriz');
ylabel('Tiempo');
legend('Rango', 'Determinante');
```



El único problema aquí fue el mal entendimiento del enunciado, ya que pensaba en una primera instancia que tendría que elegir el tamaño de la matriz. Pero después entendí que la finalidad era calcular la diferencia entre realizar operaciones con una matriz de tamaño x o de tamaño y.

EJERCICIO 5. Representación gráfica en 3D

Realice un script en Matlab que dibuje sobre el área $-5 \leq x, y \leq 5$ la superficie, la superficie en forma de malla y el contorno de la función:

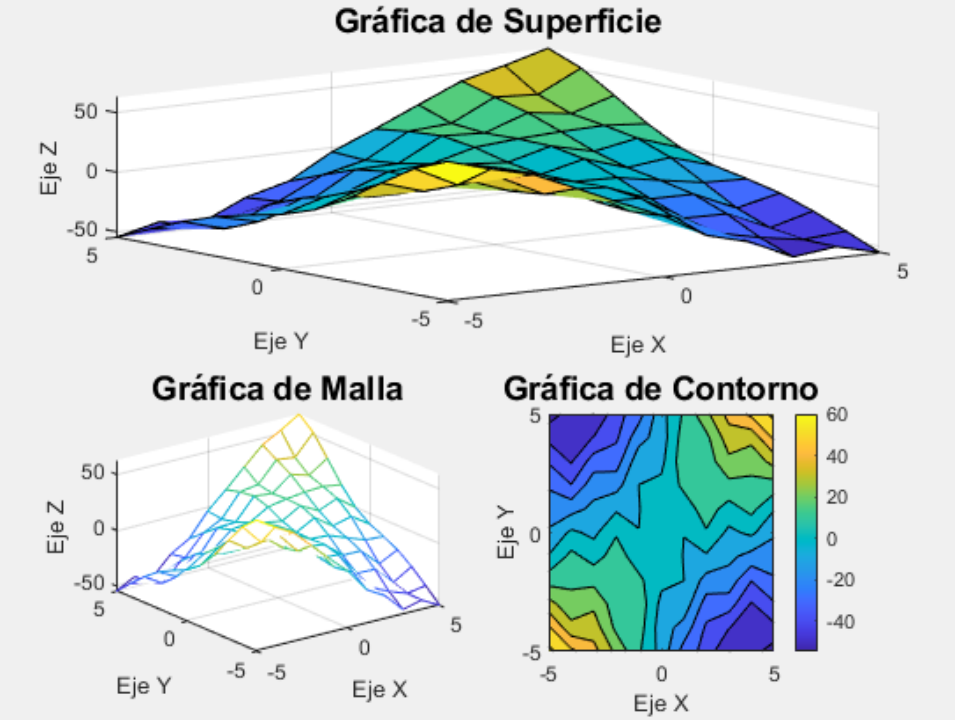
$$z = y * sin\left(pi * \frac{x}{10}\right) + 5 * \cos\left((x^2 + y^2)/8\right) + cos(x + y)cos(3x - y)$$

- En la misma figura dibuje en la parte superior y centrada la gráfica de la superficie, en la parte inferior izquierda la gráfica de la superficie en forma de malla y en la parte inferior derecha la gráfica del contorno. Además, añada la barra de color al contorno.
- Deben añadirse etiquetas a los ejes, y un título a cada gráfica.

Funciones útiles: meshgrid, mesh, surf, contour xlabel, ylabel, subplot.

Creamos el área en el que la figura tiene que ser representada, la cual viene dada por el enunciado, mediante la función *meshgrid*, la cual nos devuelve las coordenadas 2D en dos matrices según el rango de área que le hayamos especificado.
Guardamos la función en la variable Z.
Para crear las 3 gráficas definimos los cuadrantes de las subgráficas, como una matriz 2x2. Y poniendo la primera gráfica en la parte superior de la matriz, con la función *surf* obtenemos una gráfica de superficie, la función *mesh* crea una gráfica de malla, y por último la función *contourf* crea una gráfica de contorno (en 2D).

```
[X, Y] = meshgrid(-5:5);
Z = Y*sin(pi*(X/10)) + 5*cos((X.^2 + Y.^2)/8) + cos(X + Y)*cos(3*X - Y);

f = figure;
% Primera gráfica
subplot(2, 2, 1:2);
surf(X, Y, Z);
xlabel('Eje X');
ylabel('Eje Y');
zlabel('Eje Z');
title('Gráfica de Superficie', 'FontSize', 15);
% Segunda gráfica
subplot(2, 2, 3);
mesh(X, Y, Z);
xlabel('Eje X');
ylabel('Eje Y');
zlabel('Eje Z');
title('Gráfica de Malla', 'FontSize', 15);
% Tercera gráfica
subplot(2, 2, 4);
contourf(X, Y, Z);
colorbar;
xlabel('Eje X');
ylabel('Eje Y');
title('Gráfica de Contorno', 'FontSize', 15);
```


Gráfica de Superficie
Eje Z
50
0
-50
5
0
-5
Eje Y
-5
0
5
Eje X
Gráfica de Malla
Eje Z
50
0
-50
5
0
-5
Eje Y
-5
0
5
Eje X
Gráfica de Contorno
Eje Y
5
0
-5
-5
0
5
Eje X
60
40
20
0
-20
-40

EJERCICIO 6. Sistemas lineales

Dados los siguientes sistemas lineales de 10 ecuaciones con 4 incógnitas (x1, x2, x3, x4):

$$\begin{aligned}
2\cdot x_2 + 10\cdot x_3 + 7\cdot x_4 &= 90\\
2\cdot x_1 + 7\cdot x_2 + 7\cdot x_3 + x_4 &= 59\\
x_1 + 9\cdot x_2 + 5\cdot x_4 &= 15\\
4\cdot x_1 + 6\cdot x_4 &= 10\\
2\cdot x_1 + 8\cdot x_2 + 4\cdot x_3 + x_4 &= 80\\
10\cdot x_1 + 5\cdot x_2 + 3\cdot x_4 &= 17\\
2\cdot x_1 + 6\cdot x_2 + 4\cdot x_3 &= 93\\
x_1 + x_2 + 9\cdot x_3 + 3\cdot x_4 &= 51\\
6\cdot x_1 + 4\cdot x_2 + 8\cdot x_3 + 2\cdot x_4 &= 41\\
3\cdot x_2 + 9\cdot x_4 &= 76
\end{aligned}$$

$$\begin{aligned}
0.110\cdot x_1 + x_3 &= 317\\
3.260\cdot x_2 + x_4 &= 237\\
0.425\cdot x_1 + x_3 &= 319\\
3.574\cdot x_2 + x_4 &= 239\\
0.739\cdot x_1 + x_3 &= 321\\
3.888\cdot x_2 + x_4 &= 241\\
1.054\cdot x_1 + x_3 &= 323\\
4.202\cdot x_2 + x_4 &= 243\\
1.368\cdot x_1 + x_3 &= 325\\
4.516\cdot x_2 + x_4 &= 245
\end{aligned}$$

1. Exprese el sistema de forma matricial en Matlab. Para ello, cree las matrices A y B.

Separamos el sistema en dos matrices, por lo que, por cada uno de los sistemas, tenemos dos matrices.

```
A = [0 2 10 7 ; 2 7 7 1 ; 1 9 0 5 ; 4 0 0 6 ; 2 8 4 1 ; 10 5 0 3 ;
     2 6 4 0 ; 1 1 9 3 ; 6 4 8 2 ; 0 3 0 9];
Asol = [90 ; 59 ; 15 ; 10 ; 80 ; 17 ; 93 ; 51 ; 41 ; 76];
B = [0.110 0 1 0 ; 0 3.260 0 1 ; 0.425 0 1 0 ; 0 3.574 0 1 ; 0.739 0 1 0 ;
     0 3.888 0 1 ; 1.054 0 1 0 ; 0 4.202 0 1 ; 1.368 0 1 0 ; 0 4.516 0 1];
Bsol = [317 ; 237 ; 319 ; 239 ; 312 ; 241 ; 323 ; 243 ; 325 ; 245];

sistA = [A Asol];
```

```
A =                              Asol =

     0     2    10     7          90
     2     7     7     1          59
     1     9     0     5          15
     4     0     0     6          10
     2     8     4     1          80
    10     5     0     3          17
     2     6     4     0          93
     1     1     9     3          51
     6     4     8     2          41
     0     3     0     9          76
```

```
B =                                             Bsol =

    0.1100         0    1.0000         0       317
         0    3.2600         0    1.0000       237
    0.4250         0    1.0000         0       319
         0    3.5740         0    1.0000       239
    0.7390         0    1.0000         0       312
         0    3.8880         0    1.0000       241
    1.0540         0    1.0000         0       323
         0    4.2020         0    1.0000       243
    1.3680         0    1.0000         0       325
         0    4.5160         0    1.0000       245
```

2. Escriba un script en que permita:
Compare los resultados obtenidos en cada caso.
Funciones útiles: pinv, linsolve.

a. Obtener el número de condición de la matriz A respecto de la inversión.

Para obtener el número de condición usamos la función *cond()*.

```
disp('Número de condición matriz A: ');
disp(cond(A));
disp('Número de condición matriz B: ');
disp(cond(B));
```

```
Número de condición matriz A:
    2.7257

Número de condición matriz B:
   36.7102
```

No entiendo la diferencia entre sacar el número de condición de la matriz, y sacar el mismo número respecto a la inversión.

b. Resolver el sistema de ecuaciones para obtener la matriz x = [x1, x2, x3, x4]'.

Para resolver la ecuación utilizamos la función *linsolve()*.

```
disp('Resultado del sistema de ecuaciones A: ');
solA = linsolve(A, Asol);
disp(solA);
```

```
Resultado del sistema de ecuaciones A:
   -2.2571
    4.9336
    5.2986
    3.5649
```

c. Añadir ruido a la matriz b, sumándole un vector aleatorio de media 0 y desviación 1, y resuelva el sistema de ecuaciones resultante.

Según la documentación, la función *rand(n)* trabaja con una distribución normal de media 0, desviación 1. Por tanto, lo único que tenemos que hacer es poner en el *rand* el tamaño de la matriz.
Para añadirlo a las soluciones de los sistemas de ecuaciones, lo sumamos a los términos independientes de la matriz.

```
ruido = rand(10,1);
disp('Resultado del sistema de ecuaciones b: ');
solB = linsolve(B, Bsol);
disp(solB);
```

```
Resultado del sistema de ecuaciones b:
    6.3611
    6.3694
  314.4979
  216.2357
```

El único problema fue el desconocimiento de que la función *rand()* ya trabajaba con una distribución normal de media 0 y desviación 1. Por lo que lo único que tuvimos que hacer, fue definir el número de filas que tendría el vector.

d. Mostrar el resultado (con y sin ruido añadido) por pantalla.

Para mostrar los resultados por pantalla simplemente utilizamos el comando *disp* que veníamos utilizando de antes.

```
disp('Resultado del sistema de ecuaciones b sin ruido: ');
solB = linsolve(B, Bsol);
disp(solB);
disp('Resultado del sistema de ecuaciones b con ruido: ');
solBruido = linsolve(B, Bsol + ruido);
disp(solBruido);
```

```
Resultado del sistema de ecuaciones b sin ruido:
    6.3611
    6.3694
  314.4979
  216.2357

Resultado del sistema de ecuaciones b con ruido:
    6.6413
    6.4609
  314.9693
  216.5023
```

EJERCICIO 7. Polinomios

Realice una función de Matlab que permita obtener las raíces de un producto de polinomios y las clasifique en reales y complejas. Para ello ha de realizar los siguientes pasos:

1. Las entradas y salidas de la función son las que se especifican, según la siguiente sintaxis:

[solución, reales, complejas]=raices(poli_1, poli_2)

Ejemplo: [[-1+i, -1-i, -3], 1, 2]=raíces([1 2 2], [1 3])

   a. Recoge los arrays con los que se crean los polinomios.

Para recoger los arrays por teclado haremos uso de la función input, la cual nos servirá para almacenar en la variable asignada la entrada del usuario por teclado. Para introducir arrays, como explicamos en el propio mensaje, la entrada debe tener el formato [X Y Z].

```
p1 = "Introduce el polinomio con [ ] entre los números. Ej: [X Y Z]\n ";
pol1 = input(p1);

p2 = "Introduce el polinomio con [ ] entre los números. Ej: [X Y Z]\n ";
pol2 = input(p2);
```

```
Introduce el polinomio con [ ] entre los números. Ej: [X Y Z]
 [1 2 2]
Introduce el polinomio con [ ] entre los números. Ej: [X Y Z]
 [1 3]
```

   b. Solicita si la solución se aplica a uno de los polinomios o al producto: poli_1, poli_2,prod_poli

Utilizaremos la misma función para solicitar el polinomio sobre el que se ha de aplicar la función. En este caso, 1 representará el primer polinomio, 2 el segundo polinomio y 3 el producto de ambos.

```
polinomio = "¿Sobre que polonomio desea obtener las raíces? 1= polinomio 1; 2 = polinomio 2; 3 = producto de los polinomios: ";
polinomioE = input(polinomio);
```

```
¿Sobre que polonomio desea obtener las raíces? 1= polinomio 1; 2 = polinomio 2; 3 = producto de los polinomios: 1
```

   c. Devuelve las raíces del polinomio indicado y su clasificación (nº raíces reales y nº raíces complejas).

Para ello crearemos la función raíces, la cual tendrá como parámetros de entrada tanto el polinomio 1 como el polinomio 2 y como salida, tanto las raíces del polinomio deseado, como una clasificación de las mismas, donde mostraremos si son reales o imaginarias.
Una vez dentro de la función, crearemos los contadores para las raíces reales e imaginarias, solicitaremos al usuario sobre que polinomio quiere aplicar la función, haciendo uso de lo visto en el apartado b y dependiendo de la respuesta, entraremos a una de las 3 opciones del if.
En las 3 opciones tendremos el mismo proceso, únicamente cambiaremos el polinomio sobre el cual aplicaremos el proceso.
Al entrar a la opción seleccionada, se generarán las raíces del polinomio deseado gracias a la función roots(), a la cual daremos como argumento el polinomio seleccionado. Esta nos devolverá un listado de las raíces del polinomio, como podemos ver a continuación. Por

ejemplo, si damos como primer polinomio [1 2 2] y aplicamos la función, obtendremos lo siguiente:

```
sols1 = roots(pol1);

ans =

   -1.0000 + 1.0000i
   -1.0000 - 1.0000i
```

Una vez obtenido el conjunto de raíces, lo recorremos comprobando si cada una de ellas es real o imaginaria e incrementando el contador correspondiente. Para ello haremos uso del siguiente código:

```
for i = 1:(length(sols1))
    if isreal(sols1(i))
        real = real + 1;
    else
        nreal = nreal + 1;
    end
end
```

Una vez recorrido el conjunto de raíces, las devolveremos para obtener la solución final a nuestra función:

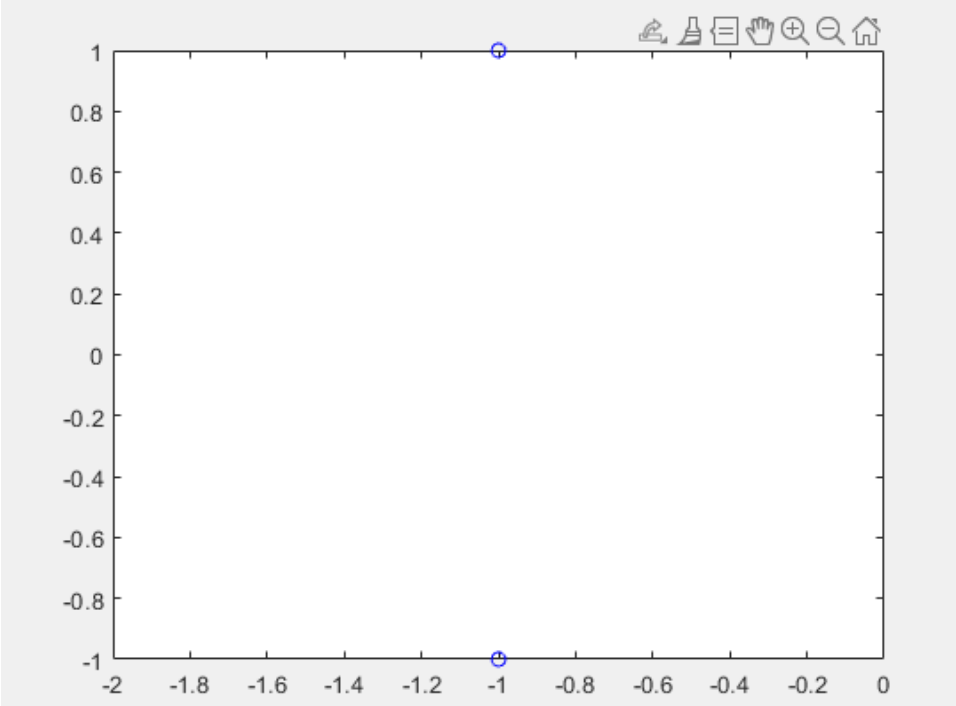
```
function [solucion, real, nreal] = raices (pol1, pol2)
    nreal =0;
    real =0;
    polinomio = "¿Sobre que polonomio desea obtener las raíces? 1= polinomio 1; 2 = polinomio 2; 3 = producto de los polinomios: ";
    polinomioE = input(polinomio);
    if polinomioE == 1
        sols1 = roots(pol1);
        for i = 1:(length(sols1))
            if isreal(sols1(i))
                real = real + 1;
            else
                nreal = nreal + 1;
            end
        end
        plot(sols1,'bo')
        roots(pol1)
    elseif polinomioE == 2
        sols2 = roots(pol2);
        for i = 1:(length(sols2))
            if isreal(sols2(i))
                real = real + 1;
            else
                nreal = nreal + 1;
            end
        end
        plot(sols2,'bo')
        roots(pol2)
    else
        pol = conv(pol1,pol2);
        sols = roots(pol);
        for i = 1:(length(sols))
            if isreal(sols(i))
                real = real + 1;
            else
                nreal = nreal + 1;
            end
        end
        plot(sols,'bo')
        roots(pol)
    end
    solucion = [real nreal];
end
```

```
Introduce el polinomio con [ ] entre los números. Ej: [X Y Z]
 [1 2 2]
Introduce el polinomio con [ ] entre los números. Ej: [X Y Z]
 [1 3]
¿Sobre que polonomio desea obtener las raíces? 1= polinomio 1; 2 = polinomio 2; 3 = producto de los polinomios: 1

ans =

  -1.0000 + 1.0000i
  -1.0000 - 1.0000i


ans =

     0     2
```

d. Representa en el plano complejo la ubicación de las raíces obtenidas.

Por último, para representar las raíces obtenidas en el plano complejo, haremos uso de la función plot(), a la cual pasaremos como parámetro el conjunto de raíces que nos ha dado la función roots().

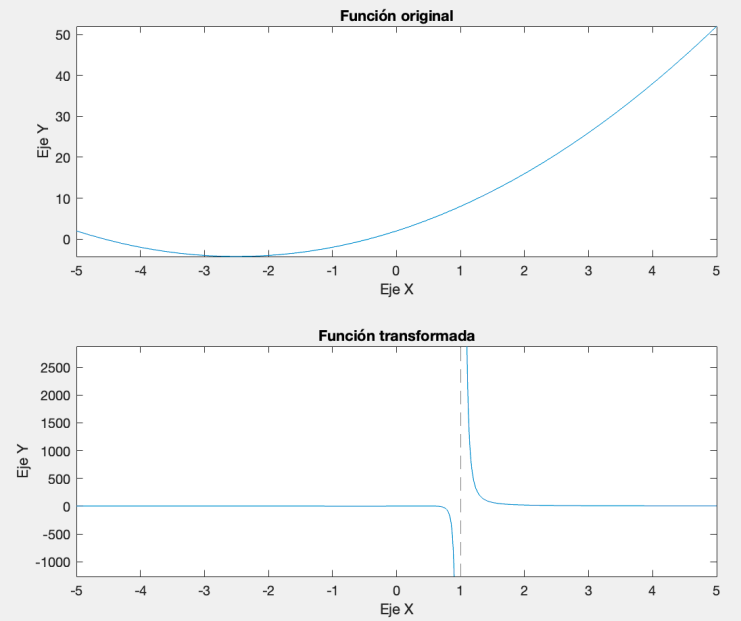
```
plot(sols1,'bo')
```



PARTE II

EJERCICIO 1. Transformadas de señales

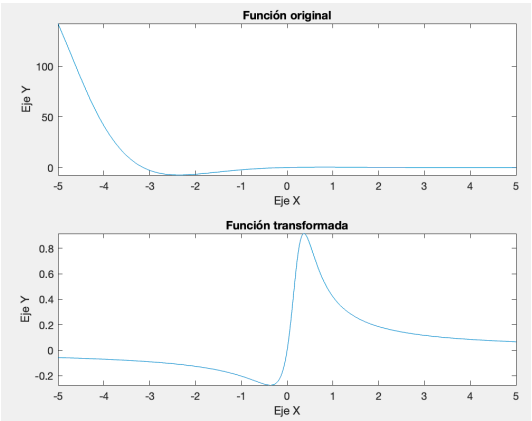
1. Obtenga la transformada z de la siguiente función: $f(k) = 2 + 5k + k^2$. Represente gráficamente las señales original y transformada.



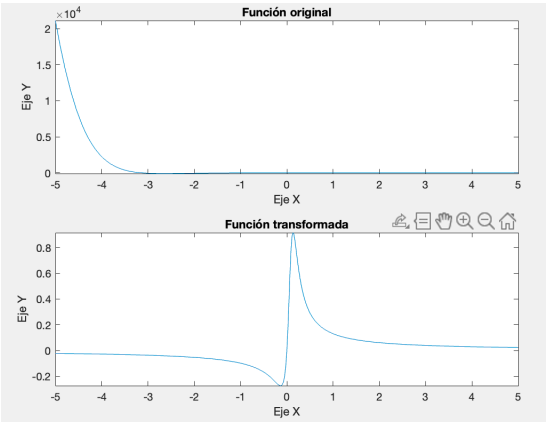
2. Obtenga la transformada z de la siguiente función: $f(k) = sen(k) \cdot e^{-ak}$. Represente gráficamente, de nuevo, las señales original y transformada.

Damos el valor de 1 a la variable a de la función, puesto que, si la inicializamos como una variable simbólica como lo hacemos con la variable 'k', no está permitido, pues no aparece ninguna salida.

a = 1



a = 2



Se puede notar una ligera diferencia en la forma de la curva.

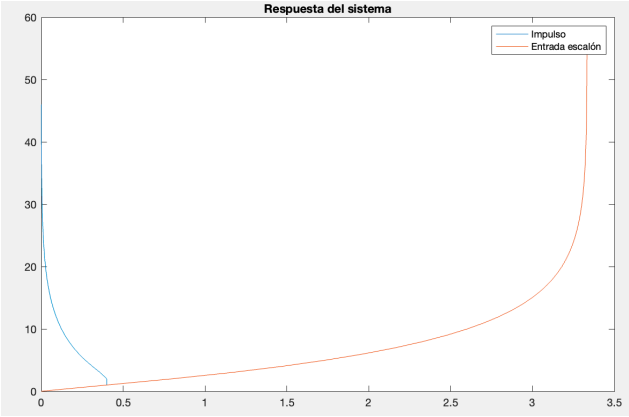
3. Dada la siguiente función de transferencia discreta:

$$T(z) = \frac{0.4 \cdot z^2}{z^3 - z^2 + 0.1 \cdot z + 0.02}$$

Como se indica en la documentación de Matlab, la función de transferencia viene dada por un modelo de función de transferencia llamado **tf**, al que le entran dos parámetros, el numerador y el denominador.
En los parámetros de numerador y denominador, debemos especificar los coeficientes de las variables del polinomio.
En este caso, como no queremos especificar el tiempo de muestreo, ponemos como argumento de entrada **ts** a -1. En de especificar un valor para este argumento, debería estar entre los valores $[0, +\infty] \in \mathbb{N}$.

- Obtenga y represente la respuesta al impulso del sistema.
- Obtenga y represente la respuesta del sistema ante una entrada escalón.

A continuación, se muestran las dos respuestas del sistema ante un impulso y una entrada escalón.

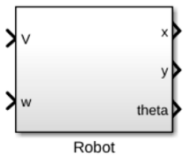


EJERCICIO 2. Modelado del comportamiento de un robot móvil en Simulink

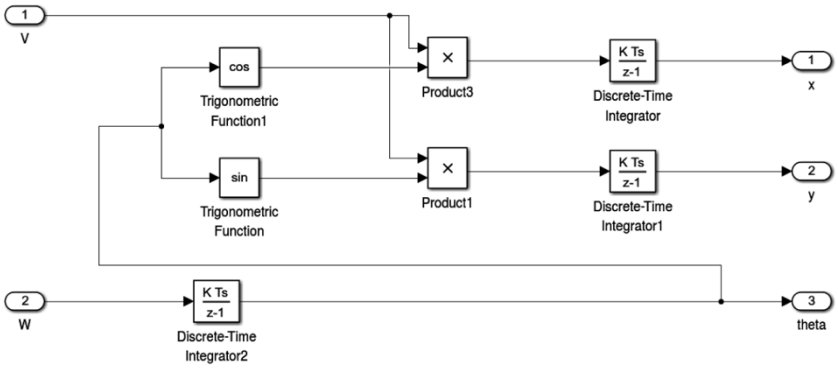
El estado del robot móvil se define mediante la posición en el entorno (x_k, y_k) y su orientación θ_k , donde el subíndice $k > 0$ indica la variable de tiempo discreto. El movimiento del robot se rige por el siguiente modelo dinámico:

$$x_k = x_{k-1} + V_{k-1}\mathrm{T_s}\cos(\theta_{k-1})$$
$$y_k = y_{k-1} + V_{k-1}T_s\sin(\theta_{k-1})$$
$$\theta_k = \theta_{k-1} + \omega_{k-1}T_s$$

Donde Ts es el tiempo de muestreo utilizado por el sistema, ω_k es la velocidad angular del robot y V_k es la velocidad lineal.
En Simulink, el robot se representará mediante un bloque cuyas entradas son las velocidades (angular y lineal) y las salidas son el valor de posición y orientación del robot:



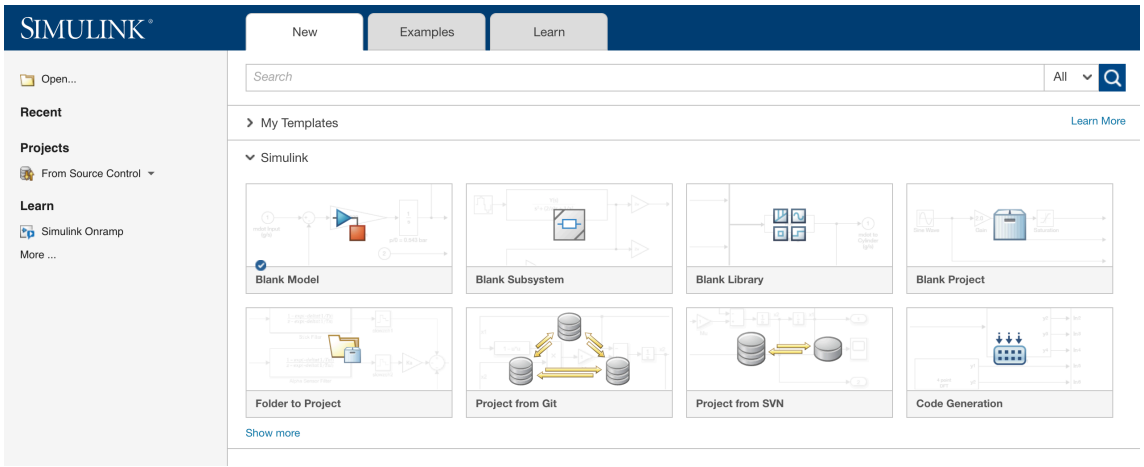
El bloque Robot deberá implementar las ecuaciones de movimiento mediante el conjunto de bloques básicos de Simulink mostrados en la siguiente figura:



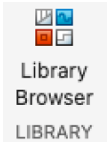
Se pide:

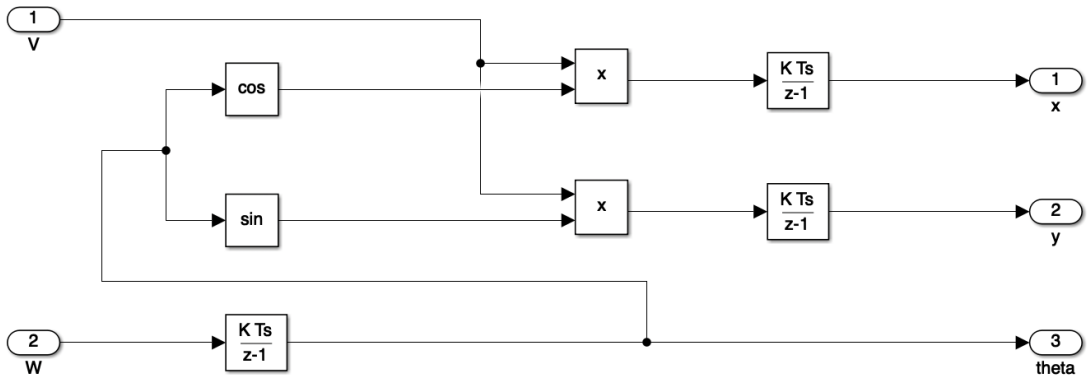
1. Implemente el modelo de la Figura 2 (todos los bloques utilizados pueden encontrarse en la librería estándar de Simulink).

Como dicta en la figura anterior, abrimos Simulink desde la ventana de comandos, de donde emergerá la siguiente ventana de inicio:



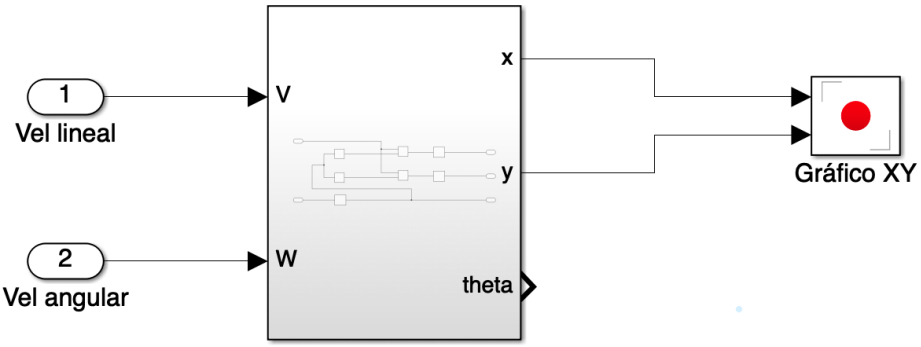
Desde aquí creamos un nuevo modelo en blanco, clicamos en Library Browser, donde podremos buscar las entradas, salidas y funciones, y por último, creamos el modelo especificado anteriormente.



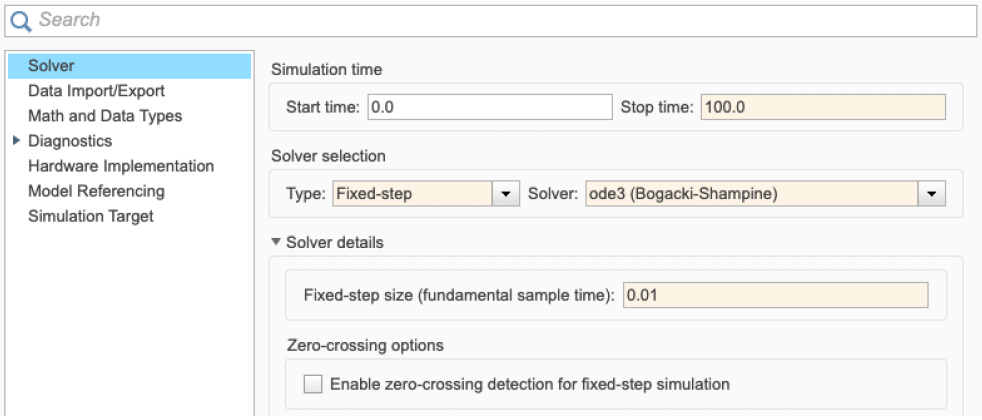


2. Una vez completado el apartado anterior, cree el subsistema mostrado en la Figura 1 y simule su funcionamiento con velocidad lineal y angular constante creando el sistema mostrado en la Figura 3. Configure los parámetros de la simulación (menú "Simulation/Model Simulation Parameters" y menú Simulation/Pacing options) de acuerdo con la Figura 4.

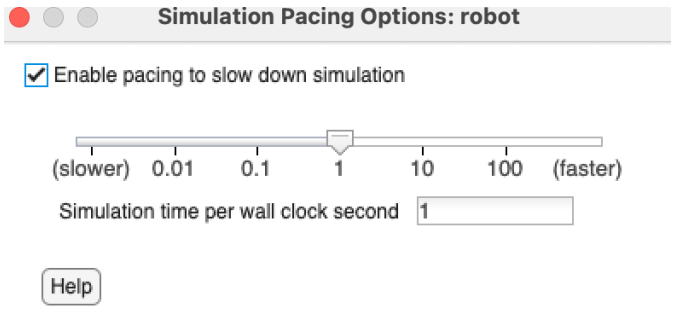
Para crear el subsistema siguiente, es necesario seleccionar el sistema creado en el apartado anterior. En el menú, seleccionamos 'Modeling', 'Component' y 'Crear subsistema'.



Para seleccionar los parámetros de la simulación, hacemos clic derecho y elegimos la opción 'Model Configuration Parameters'. Cambiamos los parámetros necesarios:

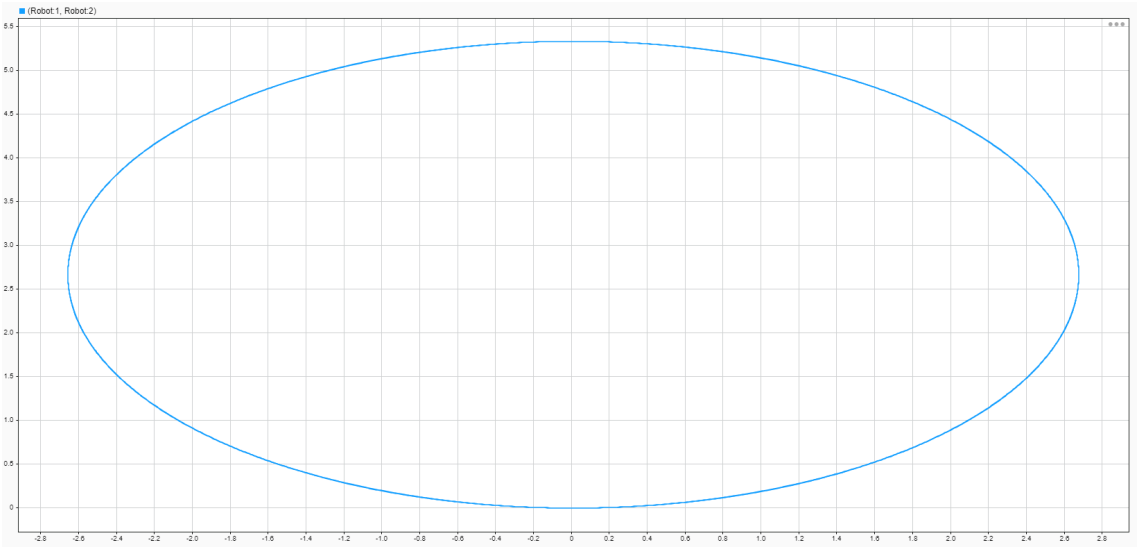


A la hora de seleccionar el menú que dicta el enunciado, 'Simulation Pacing Options', se debe clicar en 'Simulation', 'Simulate', y en las opciones de 'Run', elegir la opción de 'Simulation Pacing'.



3. Visualizando la gráfica generada por el módulo XY Graph, compruebe que el funcionamiento del robot se ajusta a lo esperado.

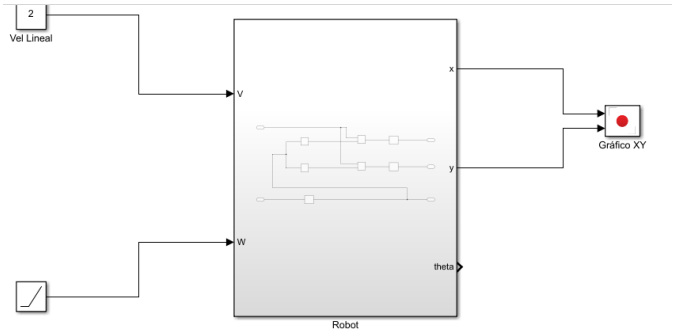
Ejecutamos el modelo, con lo que se obtiene la siguiente gráfica:

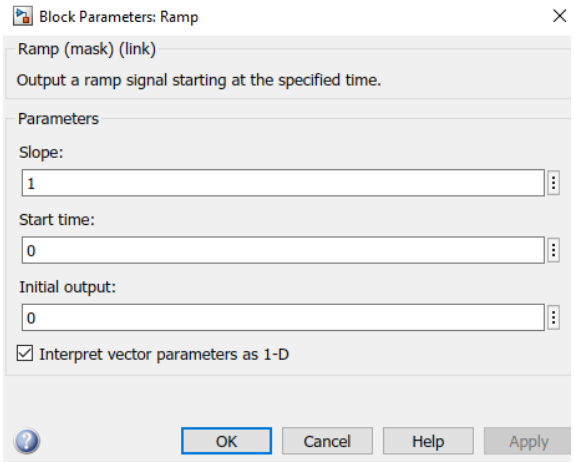


4. Realice de nuevo la simulación con velocidades angulares no constantes (estas fuentes están disponibles en la librería de Simulink/sources):

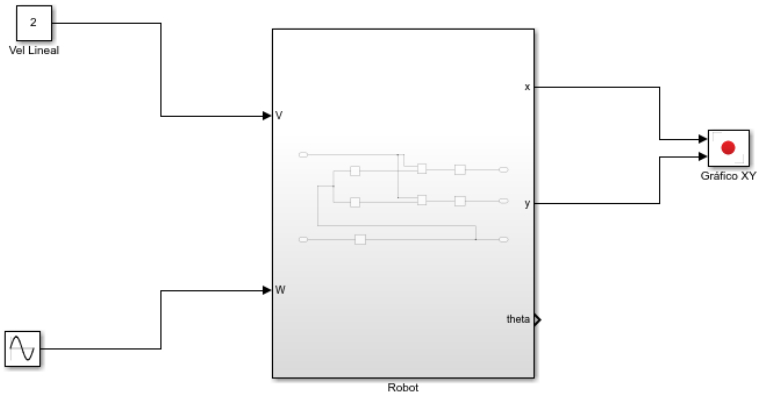
    a. Rampa

Realizamos los cambios correspondientes a la velocidad angular, obteniendo los siguientes modelos:

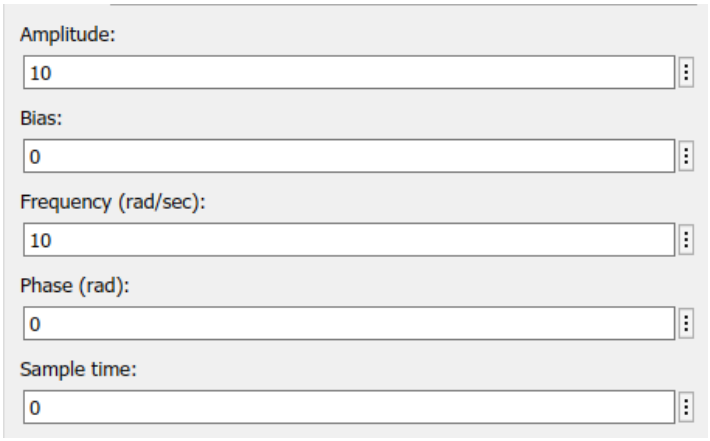




b. Variación sinusoidal

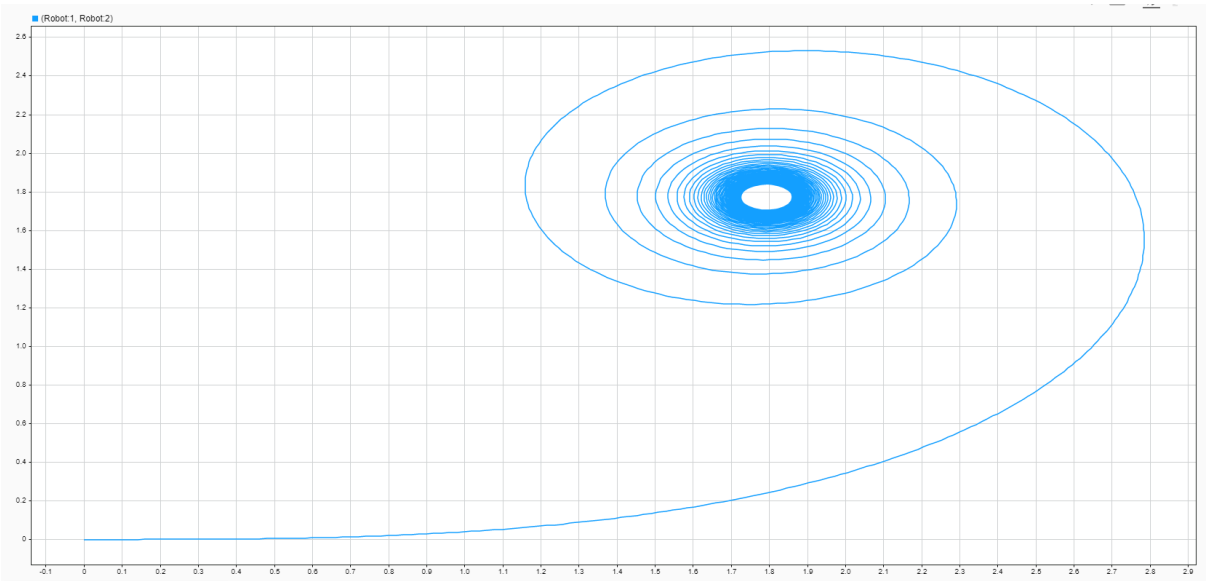


Con la siguiente configuración:

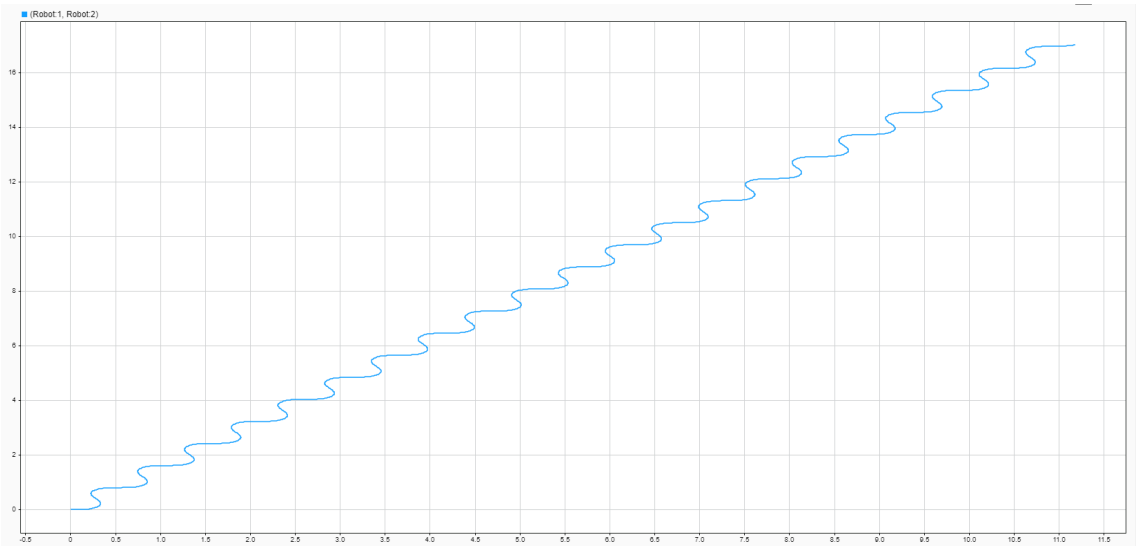


5. Estudie de nuevo las trayectorias realizadas por el robot y compruebe que se ajustan al comportamiento esperado.

Con la primera configuración (rampa) obtendremos la siguiente gráfica:

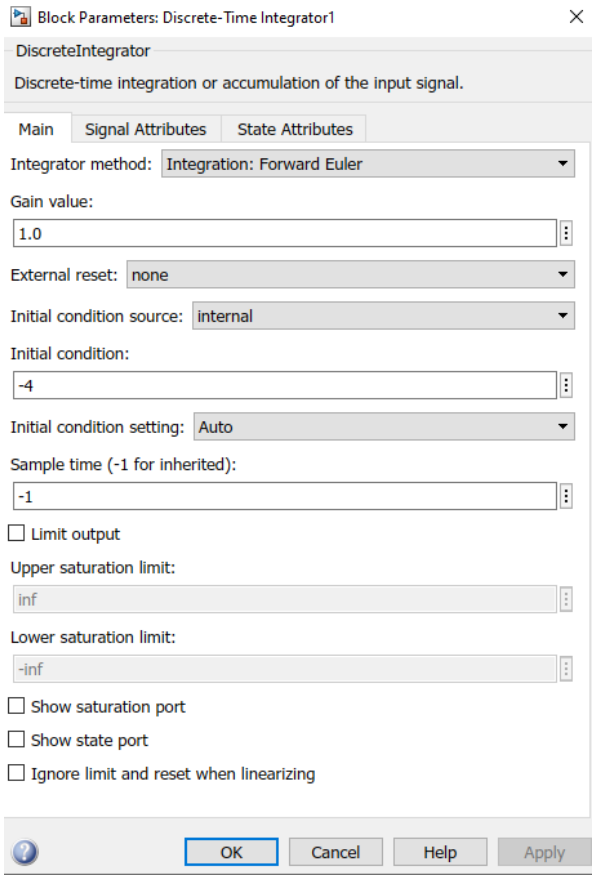


En cambio, con la variación sinusoidal obtenemos:



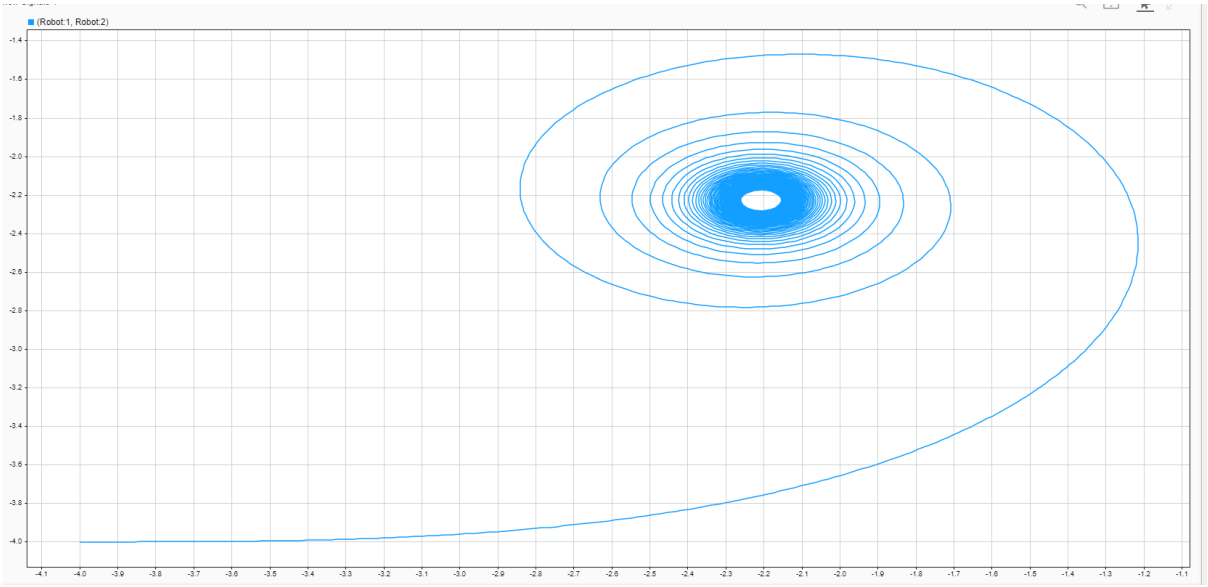
6. Modifique la posición inicial del robot para que comience a moverse desde el punto (-4,-4) y realice estas simulaciones de nuevo.

Para realizar este apartado, volvemos a la Figura 2 en Simulink, y entraremos a la configuración de los 'Discrete-Time Integrator' de los output X e Y. En ambos, modificaremos el valor de 'Initial condition' a -4, obteniendo así la modificación buscada.



Una vez realizados los cambios, realizamos las simulaciones de nuevo y obtenemos las siguientes gráficas:

- Para la rampa:



- Para la sinusoidal: