Universidad de Alcalá

PRÁCTICA 0 (1 y 2/2): INTRODUCCIÓN A MATLAB

SISTEMAS DE CONTROL INTELIGENTE GII
CURSO 2023/24

JAVIER JOSÉ GUZMÁN RUBIO 09246617L
LUCAS ÁLVAREZ RODRÍGUEZ 09109677K

Índice

PARTE 1 2
Ejercicio 1. Matrices y vectores. 2
Ejercicio 2. Matrices y vectores. 4
Ejercicio 3. Matrices y vectores. 7
Ejercicio 4. Tiempo de cómputo y representación gráfica. 13
Ejercicio 5. Representación gráfica en 3D. 15
Ejercicio 6. Sistemas lineales. 17
Ejercicio 7. Polinomios. 19
PARTE 2 22
Ejercicio 1. Transformadas de señales 22
Ejercicio 2. Modelado del comportamiento de un robot móvil en Simulink 25

PARTE 1

Ejercicio 1. Matrices y vectores.

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

   Los valores de la matriz y el vector son introducidos entre corchetes, utilizando el punto y coma para establecer las separaciones entre las filas:

```
A = [1 2; 3 4; 5 6; 7 8];
v = [14;16;18;20];
disp('Matriz A:');
disp(A);
disp('Vector v:');
disp(v);
```

   Se muestran por pantalla a través del comando ¨disp¨:

```
Matriz A:
     1     2
     3     4
     5     6
     7     8

Vector v:
    14
    16
    18
    20
```

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

   Para concatenar la matriz A y el vector v, se hace a través de la instrucción B=[A v], que funciona de forma similar a cómo se introducían los valores de la matriz en el apartado anterior:

```
B = [A v];
disp('Matriz B:');
disp(B);
```

   Resultado:

```
Matriz 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.**

A través de un bucle for i=1:nfilas, es decir, que se itera por cada fila de la matriz, se concatena al vector fila con la fila correspondiente de la matriz:

```
nFilas = size(B,1); %1 es el numero de filas y el 2 el de columnas
vFilas= [];
for i=1:nFilas
    fila=B(i,:);
    vFilas=[vFilas,fila];
end
disp('Filas de B concatenadas:');
disp(vFilas);
```

Resultado:

```
Filas de B concatenadas:
     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.**

Al igual que en el ejercicio anterior, a través de un bucle for i=1:ncolumnas, se recorre cada una de las columnas de la matriz. La columna recorrida queda almacenada en un vector columna=B(:,i), donde ¨:¨ indica toda la columna, para después concatenarlo con el vector vColumnas. Este contendrá las columnas de la matriz:

```
ncolumnas = size(B,2); %2 en lugar de 1
vColumnas= [];
for i=1:ncolumnas
    columna=B(:,i);
    vColumnas=[vColumnas;columna];
end
disp('Columnas de B concatenadas=')
disp(vColumnas);
```

Resultado:

```
Columnas de B concatenadas=
     1
     3
     5
     7
     2
     4
     6
     8
    14
    16
    18
    20
```

Ejercicio 2. Matrices y vectores.

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".**

   En primer lugar, se debe pedir por pantalla el tamaño de la matriz, a través de ¨input¨. Una vez indicado, utilizando rand, se genera una matriz con el tamaño especificado.

```
n=input('Indique el tamaño de la matriz: ');
A=rand(n);
```

   Resultado:

```
Indique el tamaño de la matriz: 4
```

2. **A partir de la matriz construida, el script deberá calcular y presentar por pantalla los siguientes datos:**

   a. **Matriz generada.**

      Al igual que anteriormente, se muestra la matriz mediante ¨disp(A)¨:

```
disp('Matriz aleatoria A:')
disp(A);
```

      Resultado para n=4:

```
Matriz aleatoria A:
    0.6324    0.9575    0.9572    0.4218
    0.0975    0.9649    0.4854    0.9157
    0.2785    0.1576    0.8003    0.7922
    0.5469    0.9706    0.1419    0.9595
```

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

      Se crea una nueva matriz B, que, mediante B=A(:,1:2:n), almacena todas las columnas desde la 1ª de 2 en 2 (para que sean las impares) de A:

```
B=A(:,1:2:n);
disp('Matriz de columnas impares B: ')
disp(B);
```

      Resultado:

```
Matriz de columnas impares B:
    0.6324    0.9572
    0.0975    0.4854
    0.2785    0.8003
    0.5469    0.1419
```

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

Para el valor de los elementos de la diagonal, se puede utilizar ¨diag(A)¨:

```
C=diag(A);
disp('C: Valores de la diagonal de la matriz A: ')
disp(C);
```

Resultado:

```
C: Valores de la diagonal de la matriz A:
    0.6324
    0.9649
    0.8003
    0.9595
```

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.

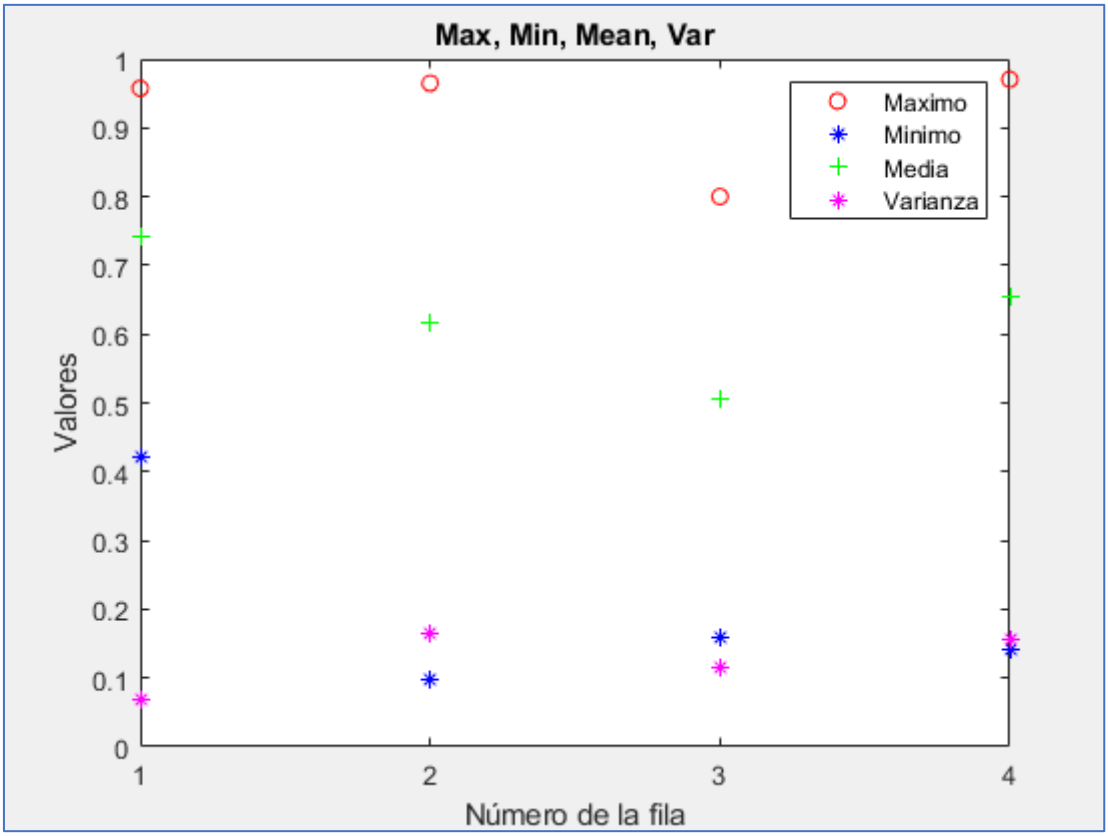
Se declara un vector distinto para el máximo, mínimo, media y varianza. A través de un bucle for i=1:n, se recorre una a una las filas de la matriz para almacenarlas en el vector ¨fila¨. Mediante las funciones max, min, mean y var, se obtienen los valores deseados de la fila, y, a continuación, se concatenan con el vector correspondiente:

```
vector_max=[];
vector_min=[];
vector_mean=[];
vector_var=[];
for i=1:n
    fila=A(i,:);
    maximo=max(fila);
    minimo=min(fila);
    media=mean(fila);
    varianza=var(fila);
    vector_max=[vector_max,maximo];
    vector_min=[vector_min,minimo];
    vector_mean=[vector_mean,media];
    vector_var=[vector_var,varianza];
end
```

Para la representación gráfica, se utiliza la función subplot. Se renombra la ventana con los parámetros de ¨figure¨, y se utiliza curtick para mostrar solamente valores enteros en el eje X:

```
figure('Name','Todos los valores a la vez','NumberTitle','off')
plot(vector_max,'ro')
hold on;
plot(vector_min,'b*')
plot(vector_mean,'g+')
plot(vector_var,'m*')
hold off;
xlabel('Número de la fila')
ylabel('Valores')
title("Max, Min, Mean, Var")
legend({'Maximo','Minimo','Media','Varianza'})
curtick = get(gca, 'xTick');
xticks(unique(round(curtick)));
```

Resultado:


Max, Min, Mean, Var
Maximo
Minimo
Media
Varianza
Valores
Número de la fila
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1
2
3
4

Ejercicio 3. Matrices y vectores.

1. **Solicite al usuario las dimensiones de las matrices en formato [filas cols], (si se introduce un único número, la matriz será cuadrada).**

   Se solicitan de nuevo los valores:

```
n = input ('Indique el tamaño de la matriz A: ');
```

   Resultado:

```
Indique el tamaño de la matriz A: [2 3]
```

```
m = input ('Indique el tamaño de la matriz B: ');
```

   Resultado:

```
Indique el tamaño de la matriz B: [2 3]
```

2. **Genere dos matrices (A y B) de las dimensiones elegidas. Para rellenar las matrices, escriba una función en Matlab (en un fichero diferente) que reciba como parámetro las dimensiones deseadas [filas cols], y devuelva la matriz rellena.**

   Para esto, en un fichero diferente, se crea la función IntroducirMatriz, que recibe como parámetro el tamaño de la matriz y devuelve una nueva matriz.

```
[A] = IntroducirMatriz(n);
```

```
[B] = IntroducirMatriz(m);
```

   Resultado, para los tamaños indicados en el apartado anterior:

```
Matriz A:
    0.6948    0.9502    0.4387
    0.3171    0.0344    0.3816

Matriz B:
    0.7655    0.1869    0.4456
    0.7952    0.4898    0.6463
```

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.**

   Para indicar si el usuario desea que los valores sean aleatorios o no, se muestra el siguiente mensaje, pidiendo que se seleccione una opción:

```
n = input('Introduce "r" para rellenar aleatoriamente o "n" para hacerlo a mano', 's');
```

   Con un switch n

```
switch n
```

   Se desarrollan los posibles casos:

   Si el usuario escribe una r, se rellenará la matriz de valores aleatorios, mediante la función rand:

```
case 'r'
    if size(dimensiones)==1  %Si solo hay un digito, la matriz es cuadrada
        A = rand(dimensiones);
    else
        A = rand (dimensiones(1), dimensiones(2));
    end
```

Resultado para cuando el usuario pulsa r:

```
Matriz A:
    0.6948    0.9502    0.4387
    0.3171    0.0344    0.3816

Matriz B:
    0.7655    0.1869    0.4456
    0.7952    0.4898    0.6463
```

En caso de que el usuario pulse n, deberá indicar los valores de la matriz:

```
case 'n'
    disp('Por favor, introduce los siguiente valores de la matriz:')
    for i=1:dimensiones(1)
        for j=1:dimensiones(2)
            fprintf('Posición [%i, %i]: ', i, j);
            A(i,j) = input('');
        end
    end
```

El primer bucle for representa las filas mientras que el segundo las columnas.

El resultado para cuando el usuario pulsa n:

```
Introduce "r" para rellenar aleatoriamente o "n" para hacerlo a mano: n
```

Se continúa pidiendo el valor para cada posición de la matriz. Una vez solicitados todos los valores, el resultado es:

```
Posición [1, 1]: 2
Posición [1, 2]: 2
Posición [1, 3]: 2
Posición [2, 1]: 3
Posición [2, 2]: 3
Posición [2, 3]: 3
Matriz A:
     2     2     2
     3     3     3

Matriz B:
     2     2     2
     3     3     3
```

4. Calcule y muestre por pantalla:

a. Las matrices generadas A y B.

Como ya hemos hecho anteriormente, los mostraremos mediante la función disp():

```
disp('Matriz A:')
disp(A);
disp('Matriz B:')
disp(B);
```

Resultado:

```
Matriz A:
     2     2     2
     3     3     3

Matriz B:
     2     2     2
     3     3     3
```

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

Las transpuestas A´ y B´ se consiguen utilizando ¨'¨:

```
disp('TRANSPUESTAS DE LAS MATRICES:')
disp('Transpuesta de A: ');
disp(A');
disp('Transpuesta de B: ');
disp(B');
```

Resultado:

```
TRANSPUESTAS DE LAS MATRICES:
Transpuesta de A:
     2     3
     2     3
     2     3

Transpuesta de B:
     2     3
     2     3
     2     3
```

Para la inversa, se ha de comprobar si es una matriz cuadrada, ya que si no lo es, no tiene inversa. Para obtener la inversa, se utiliza la función inv():

```
disp('INVERSAS DE LAS MATRICES:')
dimensionesA = size(A);
if dimensionesA(1)==dimensionesA(2)
    disp('Inversa de A: ');
    disp(inv(A));
else
    disp('La matriz A no es cuadrada, por lo que no tiene inversa')
end

dimensionesB = size(B);
if dimensionesB(1)==dimensionesB(2)
    disp('Inversa de B: ');
    disp(inv(B));
else
    disp('La matriz B no es cuadrada, por lo que no tiene inversa')
end
```

Para la matriz anterior, no hay inversa ya que no es cuadrada, por ello el resultado para este caso es el siguiente:

```
INVERSAS DE LAS MATRICES:
La matriz A no es cuadrada, por lo que no tiene inversa
La matriz B no es cuadrada, por lo que no tiene inversa
```

En caso de si ser una matriz cuadrada, el resultado para las dos siguientes matrices seria:

```
Matriz A:
    0.1190    0.3404    0.7513
    0.4984    0.5853    0.2551
    0.9597    0.2238    0.5060

Matriz B:
    0.6991    0.5472    0.2575
    0.8909    0.1386    0.8407
    0.9593    0.1493    0.2543
```

```
INVERSAS DE LAS MATRICES:
Inversa de A:
   -0.7655    0.0131    1.1301
    0.0235    2.1163   -1.1019
    1.4417   -0.9609    0.3202

Inversa de B:
   -0.3550   -0.3961    1.6690
    2.2809   -0.2724   -1.4092
    0.0001    1.6541   -1.5362
```

c. El valor del determinante y el rango de cada una de las matrices.

Para calcular el rango, se utiliza la función rank().Para el determinante, se comprueba si la matriz es cuadrada, y, en caso de que lo sea, se obtiene con la función det():

```
disp('RANGOS DE LAS MATRICES:')
disp('Rango de A: ');
disp(rank(A));
disp('Rango de B: ');
disp(rank(B));

disp('DETERMINANTES DE LAS MATRICES: ')

if dimensionesA(1)==dimensionesA(2)
    disp('Determinante de A: ');
    disp(det(A));
else
    disp('La matriz A no es cuadrada, por lo que no tiene determinante');
end

if dimensionesB(1)==dimensionesB(2)
    disp('Determinante de B: ');
    disp(det(B));
else
    disp('La matriz A no es cuadrada, por lo que no tiene determinante');
end
```

Resultado para la anterior matriz cuadrada:

```
RANGOS DE LAS MATRICES:
Rango de A:
     3

Rango de B:
     3

DETERMINANTES DE LAS MATRICES:
Determinante de A:
   -0.3122

Determinante de B:
    0.2543
```

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

Para el producto matricial y aquel obtenido elemento a elemento, se debe comprobar la multiplicidad de las matrices. Dos matrices son multiplicables si el número de columnas de A es igual al número de filas de B. En caso de que así sea, el productor matricial se calcula haciendo A*B y el producto elemento a elemento, A.*B:

```
disp('PRODUCTO MATRICIAL:');
if dimensionesA(2)==dimensionesB(1)
    disp(A*B);
else
    disp('¡Las dimensiones de las matrices impiden el producto matricial!');
end

disp('PRODUCTO ELEMENTO A ELEMENTO:');
if dimensionesA(2)==dimensionesB(1)
    disp(A.*B);
else
    disp('¡Las dimensiones de las matrices impiden el producto matricial!');
end
```

El resultado para las matrices anteriores es:

```
PRODUCTO MATRICIAL:
    1.1071    0.2245    0.5078
    1.1145    0.3919    0.6852
    1.3557    0.6317    0.5640

PRODUCTO ELEMENTO A ELEMENTO:
    0.0832    0.1863    0.1935
    0.4440    0.0811    0.2145
    0.9207    0.0334    0.1287
```

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

Para ello, se concatenan A(1,:) y B(1,:).

```
vFila = [A(1,:) B(1,:)];
disp('PRIMERAS FILAS CONCATENADAS:');
disp(vFila);
```

Resultado:

```
PRIMERAS FILAS CONCATENADAS:
    0.1190    0.3404    0.7513    0.6991    0.5472    0.2575
```

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

Similar al apartado anterior, se concatenan las columnas de las matrices, esta vez utilizando A(:,1) y B(:,1):

```
vColumna = [A(:,1);B(:,1)];
disp('PRIMERAS COLUMNAS CONCATENADAS:');
disp(vColumna);
```

Resultado:

```
PRIMERAS COLUMNAS CONCATENADAS:
    0.1190
    0.4984
    0.9597
    0.6991
    0.8909
    0.9593
```

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.**

Dado que con el número de iteraciones presentado en el enunciado no se llega a apreciar una diferencia notable en el gráfico, hemos decidido aumentar la muestra a 250 matrices, para así poder apreciar mejor en la grafica la diferencia de tiempo en el cálculo de cada una de las matrices.

En un bucle for, se generan cada una de las matrices, desde la matriz de 1x1 hasta la de 250x250.

Para cada matriz, se comienza a registrar el tiempo con ¨tic¨ y se calcula el determinante mediante la función det(). Una vez calculado el determinante, se almacena, en tocDeterminante, el tiempo transcurrido para después concatenarlo con el vector en el que se almacenan lo tiempos de cálculo del determinante decada una de las matrices.

Para el tiempo que tarda en calcular el rango de cada una de las matrices, se sigue el mismo proceso que con el determinante. Se almacena, en tiempoRango, el tiempo de cada una de las matrices:

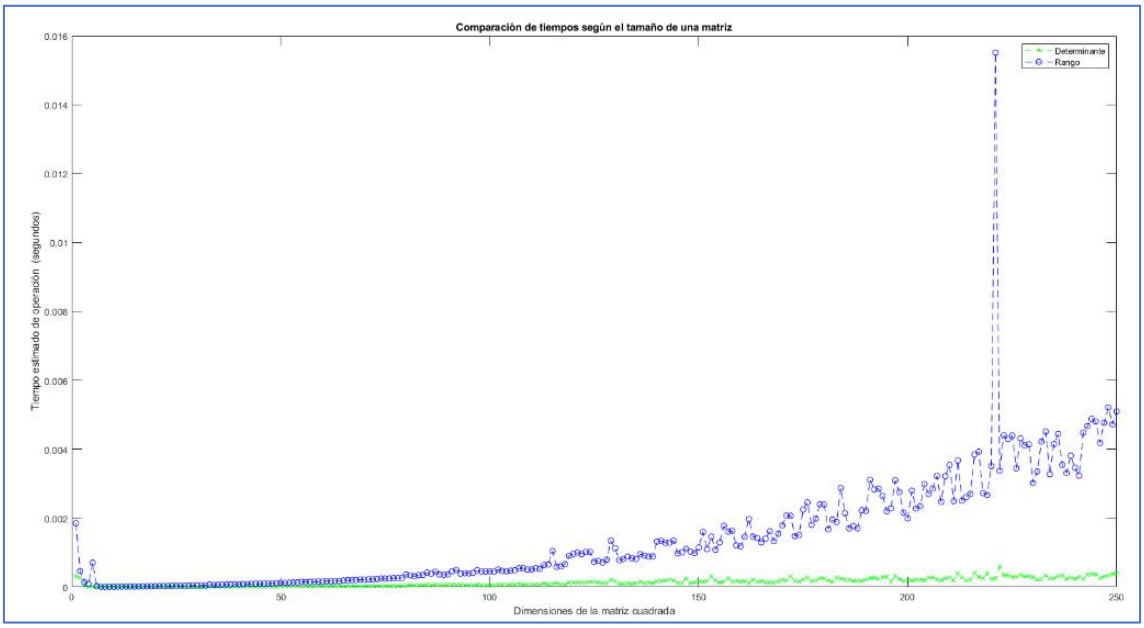
```
for i=1:n
    A = rand(i);
    tic;
    det(A);
    tocDeterminante = toc;
    tiempoDeterminante = [tiempoDeterminante tocDeterminante];

    tic;
    rank(A);
    tocRango = toc;
    tiempoRango = [tiempoRango tocRango];
end
```

A continuación, se muestra en una grafica el tiempo que tarda en calcular tanto el determinante como el rango de cada una de las 250 matrices generadas.

Para ello, se genera un plot de los vectores, tiempoDeterminante y tiempoRango:

```
figure('Name','Comparando tiempos','NumberTitle','off') %Titulo de la ventana
plot(1:n, tiempoDeterminante,'g--x', 1:n, tiempoRango, 'b--o');
title('Comparación de tiempos según el tamaño de una matriz');
xlabel('Dimensiones de la matriz cuadrada');
ylabel('Tiempo estimado de operación  (segundos)');
legend('Determinante', 'Rango');
```

Resultado:



Como se puede apreciar, cuanto más grande es la matriz, más tiempo tarda en calcular tanto el rango como el determinante. La mayor diferencia es el hecho de que, a diferencia del crecimiento lineal del tiempo de cálculo del determinante, calcular el rango de una matriz implica un tiempo de cálculo que aumenta de forma exponencial según el tamaño de dicha matriz.

Se pueden observar algunos *outlyers*, aunque estos valores se deben a picos de tiempo en el manejo de la memoria o capacidad de procesamiento del ordenador.

Ejercicio 5. Representación gráfica en 3D.

Realice un script en Matlab que dibuje sobre el área −5 ≤ x, y ≤ 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.**

En primer lugar, se crean un vector x dividido en 20 valores desde -5 a 5, y un vector y similar.

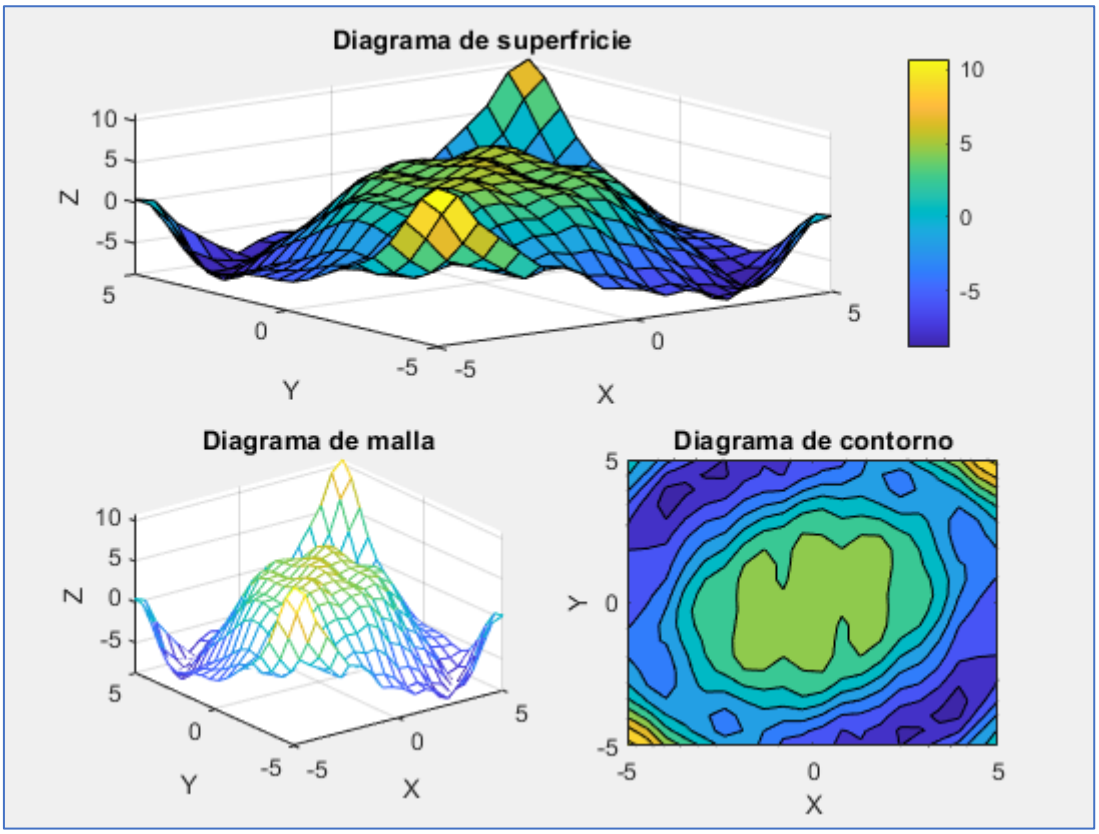
A partir de estos dos vectores, se obtiene una matriz X y una matriz Y a través de la función meshgrid(), que necesita como parámetros el vector_x y el vector_y para poder generar las matrices X e Y (20x20). El tamaño de esta matriz vendrá dado por el tamaño de ambos vectores. A partir de estas dos matrices, se generará una matriz Z con los valores de las matrices X e Y procesados por la función del enunciado.

```
% Crear rejilla uniforme de 20 valores x,y entre 5 y -5
vector_x=linspace(-5,5,20);
vector_y=linspace(-5,5,20);

[X,Y]=meshgrid(vector_x,vector_y);

% .* significa cada elemento de la matriz
Z = Y.*sin(pi*X/10)+5*cos((X.^2+Y.^2)/8)+cos(X+Y).*cos(3*X-Y);
```

Una vez se tienen las tres matrices, se ha de representar gráficamente la superficie, la malla y el contorno. Para la superficie, se utiliza la función surf(X,Y,Z), para la malla mesh(X,Y,Z) y para el contorno contourf(X,Y,Z). Todos estos se muestran en una misma ventana, utilizando subplot().

```
% Superficie
subplot(2,2,[1,2])
surf(X,Y,Z);
colorbar          % Aporta visibilidad a los plots
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Diagrama de superfricie');

% Malla
subplot(2,2,3)
mesh(X,Y,Z);
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Diagrama de malla');

% Contorno
subplot(2,2,4)
contourf(X,Y,Z);
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Diagrama de contorno');
```

Resultado:



Ejercicio 6. Sistemas lineales.

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

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

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

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

   Los sistemas se expresan de la siguiente forma, utilizando 0 para los valores inexistentes de X:

```
% -------------------------PRIMER SISTEMA:----------------
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];

% Terminos independientes
ATi = [90; 59; 15; 10; 80; 17; 93; 51; 41; 76];

% -------------------------SEGUNDO SISTEMA:----------------
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];

% Terminos independientes
BTi = [317; 237; 319; 239; 321; 241; 323; 243; 325; 245];
```

2. **Escriba un script en que permita:**

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

   Se utiliza la función cond(A):

```
nCondicion = cond(A);
disp('Número de condición de la matriz A:');
disp(nCondicion);
```

   Resultado:

```
Número de condición de la matriz A:
    2.7257
```

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

   Para resolver el sistema, se utiliza la función linsolve():

```
SolucionesA = linsolve(A,ATi);
SolucionesB = linsolve(B,BTi);
```

Resultado:

```
SOLUCIONES A: [X1:-2.257095e+00, X2:4.933635e+00, X3:5.298567e+00, X4:3.564928e+00]
SOLUCIONES B: [X1:6.359299e+00, X2:6.369427e+00, X3:3.162992e+02, X4:2.162357e+02]
```

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

Para aplicar ruido, hemos se crea un vector aleatorio de desviación 0 y media 1, como indica el enunciado:

```
media = 0; desv = 1;
vRuido = desv.*randn(1,10) + media;
```

Para crear una matriz con ruidos, se suman a cada uno de los valores de la matriz de términos independientes el vector ruido. Estas sumas se almacenan en un vector BTiR. Para obtener la solución de la matriz B con ruidos, se utiliza de nuevo la función linsolve(B, BTiR´), con el vector de términos independientes siendo el nuevo vector con ruido:

```
media = 0; desv = 1;
vRuido = desv.*randn(1,10) + media;

BTiR = [];

for i=1:10
    BTiR(i) = BTi(i)+vRuido(i);
end

SolucionesBRuido = linsolve(B,BTiR');
```

Resultado:

```
SOLUCIONES B CON RUIDO: [X1:5.632169e+00, X2:7.583099e+00, X3:3.169420e+02, X4:2.112894e+02]
```

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

Se muestran los resultados.

```
fprintf('SOLUCIONES A: [X1:%i, X2:%i, X3:%i, X4:%i]\n', SolucionesA(1), ...
    SolucionesA(2), SolucionesA(3), SolucionesA(4));

fprintf('SOLUCIONES B: [X1:%i, X2:%i, X3:%i, X4:%i]\n', SolucionesB(1), ...
    SolucionesB(2), SolucionesB(3), SolucionesB(4));

fprintf('SOLUCIONES B CON RUIDO: [X1:%i, X2:%i, X3:%i, X4:%i]\n', ...
    SolucionesBRuido(1), SolucionesBRuido(2), SolucionesBRuido(3), ...
    SolucionesBRuido(4));

fprintf('COMPARACION DE VALORES: [ΔX1:%i, ΔX2:%i, ΔX3:%i, ΔX4:%i]\n', ...
    SolucionesB(1)-SolucionesBRuido(1), SolucionesB(2)-SolucionesBRuido(2), ...
    SolucionesB(3)-SolucionesBRuido(3), SolucionesB(4)-SolucionesBRuido(4));
```

Resultado:

```
SOLUCIONES A: [X1:-2.257095e+00, X2:4.933635e+00, X3:5.298567e+00, X4:3.564928e+00]
SOLUCIONES B: [X1:6.359299e+00, X2:6.369427e+00, X3:3.162992e+02, X4:2.162357e+02]
SOLUCIONES B CON RUIDO: [X1:5.632169e+00, X2:7.583099e+00, X3:3.169420e+02, X4:2.112894e+02]
COMPARACION DE VALORES: [ΔX1:7.271295e-01, ΔX2:-1.213672e+00, ΔX3:-6.428059e-01, ΔX4:4.946300e+00]
```

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:

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, se pide al usuario por pantalla que introduzca el polinomio 1, que se guardará en poli1, y el polinomio 2, que será almacenado en poli2.

```
poli1 = input ('Introduzca el Polinomio 1: ');
poli2 = input ('Introduzca el Polinomio 2: ');
```

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

Para esto, se pide al usuario por pantalla para qué polinomio desea la solución. La respuesta del usuario quedará almacenada en opción.

```
disp('¿Para qué polinomio desea encontrar solución? ');
opcion   = input ('¿p1, p2, o prod? ', 's');
```

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

A partir de un switch con la variable ¨opcion¨ del apartado anterior, se accede al caso que haya sido seleccionado por el usuario:

```
switch opcion
    case "p1"
        [solucion, reales, complejas] = raices(poli1);
    case "p2"
        [solucion, reales, complejas] = raices(poli2);
    case "prod"
        [solucion, reales, complejas] = raices(conv(poli1,poli2));
end
```

Para conocer tanto la solución como el número de raíces reales y complejas, se llama a la nueva función ¨raices¨:

```
function [solucion, reales, complejas] = raices(poli)
    reales=[]; complejas=[];
    solucion = roots(poli);
    for i=1:size(solucion)
        if isreal(solucion(i))
            reales=[reales;solucion(i)];
        else
            complejas=[complejas;solucion(i)];
        end
    end
end
```

En esta función, en primer lugar, se inicializa un vector ¨reales¨ y otro ¨complejas¨. Con un bucle for, se recorre el vector soluciones en el que están todas las raíces del polinomio y se comprueba si la raíz es real o compleja. En caso de que la raíz sea real, se concatena al vector reales. En el caso de que sea una raíz compleja, se concatena la raíz con el vector de complejas.

Por último, en el script del ejercicio 7, se muestran las raíces del polinomio, el número de raíces reales y el número de raíces complejas:

```
disp('Raices del polinomio seleccionado: ')
disp(solucion);

disp('Número de raices reales del polinomio seleccionado: ')
disp(size(reales,1));
disp('Raices reales del polinomio seleccionado: ')
disp(reales);

disp('Número de raices complejas del polinomio seleccionado: ')
disp(size(complejas,1));
disp('Raices complejas del polinomio seleccionado: ')
disp(complejas);
```

Resultado para el producto del polinomio 1 y 2 del enunciado:

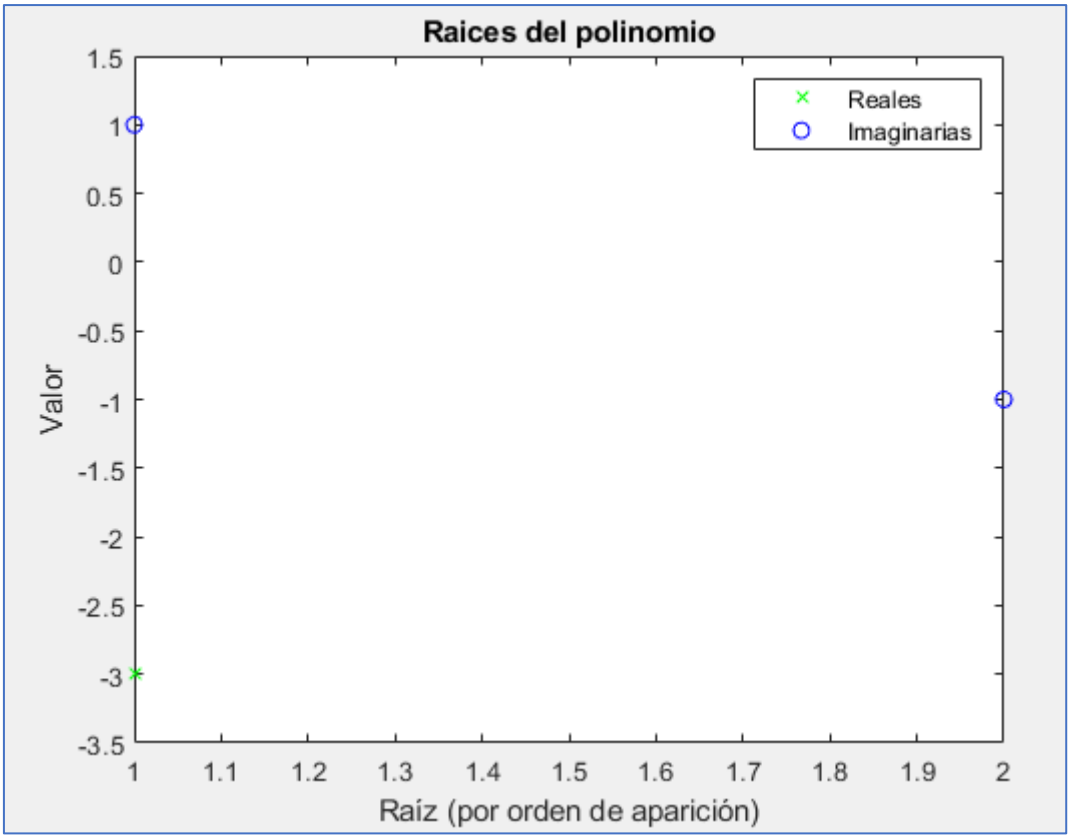
```
Introduzca el Polinomio 1: [1 2 2]
Introduzca el Polinomio 2: [1 3]
¿Para qué polinomio desea encontrar solución?
¿p1, p2, o prod? prod
Raices del polinomio seleccionado:
  -3.0000 + 0.0000i
  -1.0000 + 1.0000i
  -1.0000 - 1.0000i

Número de raices reales del polinomio seleccionado:
     1

Raices reales del polinomio seleccionado:
   -3.0000

Número de raices complejas del polinomio seleccionado:
     2

Raices complejas del polinomio seleccionado:
  -1.0000 + 1.0000i
  -1.0000 - 1.0000i
```

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

Con la función plot(), se representan gráficamente las raíces reales y complejas. En el eje de las x estará el número de aparición de la raíz real o compleja, mientras que, en el eje de las y, estará el valor de las raíces:

```
figure('Name','Raices del polinomio seleccionado','NumberTitle','off') %Titulo de la ventana
plot(1:size(reales),real(reales),'gx', 1:size(complejas), imag(complejas), 'bo');
title('Raices del polinomio');
xlabel('Raíz (por orden de aparición)');
ylabel('Valor');
legend('Reales', 'Imaginarias');
```

Resultado para el producto del polinomio 1 y del polinomio 2 del enunciado:



PARTE 2

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.

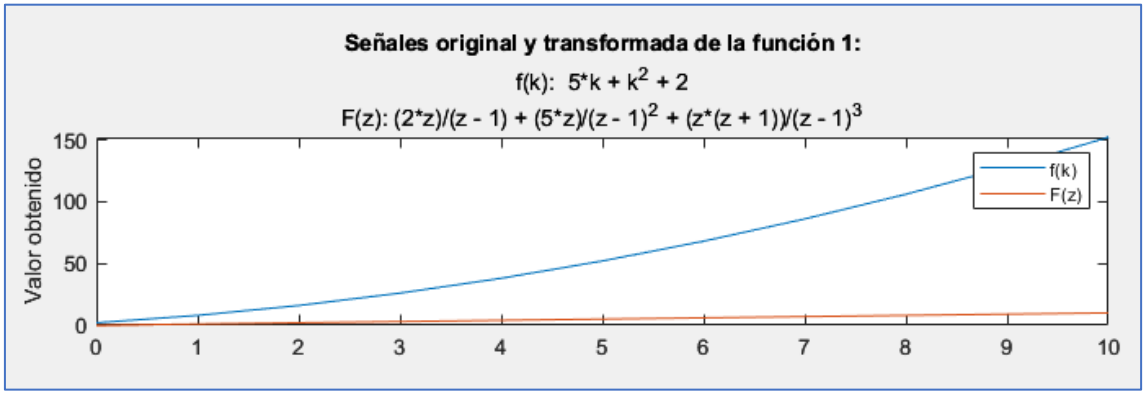
Para obtener la transformada de la función, se hace uso del método ztrans (habiendo instalado las librerías *Symbolic Math Toolbox* y *Control System Toolboz*). Es necesario también declarar ¨k¨ como una variable operable con *syms*:

```
syms k; % Necesario  para operar con k y otras variables
fk = 2 + 5*k + k^2;

disp('Transformada Z de la función 1:');
Fz = ztrans(fk)
```

La representación de la función y su transformada se realiza en la misma imagen, mostrando cada una en una línea distinta:

```
figure('Name','Transformadas Z','NumberTitle','off','Position',[100 -100 800 1000]);
subplot(4,1,1);
plot(0:10, subs(fk, k, 0:10), 0:10, subs(Fz, Fz, 0:10));
legend('f(k)','F(z)');
title('Señales original y transformada de la función 1:');
subtitle(txt);
ylabel('Valor obtenido');
```

Resultado:



2. **Obtenga la transformada z de la siguiente función:**

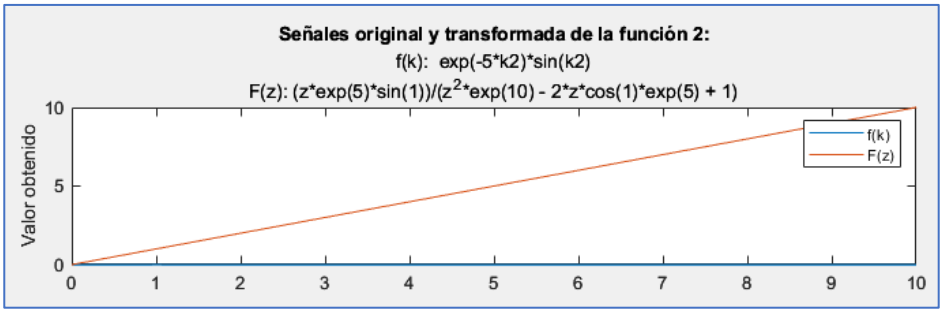
$$f(k) = \text{sen}(k) \cdot e^{-ak}.$$

Represente gráficamente las señales original y transformada.

Este apartado se ha llevado a cabo de forma similar al anterior:

```
syms k2;
fk2 = sin(k2) * exp(-5*k2);

disp('Transformada Z de la función 2:');
Fz2 = ztrans(fk2)

txt2 = {'f(k):  '+string(fk2),'F(z): '+string(Fz2)}

subplot(4,1,2);
plot(0:10, subs(fk2, k2, 0:10), 0:10, subs(Fz2, Fz2, 0:10));
legend('f(k)','F(z)');
title('Señales original y transformada de la función 2:');
subtitle(txt2);
ylabel('Valor obtenido');
```

Resultado:



3. Dada la siguiente función de transferencia discreta:

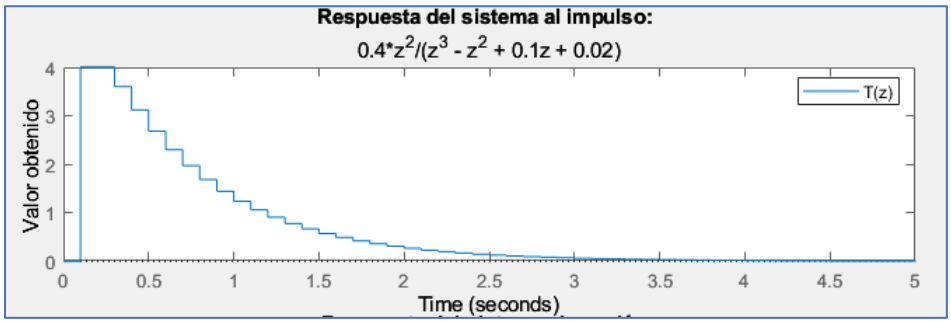
$$T(z) = \frac{0.4 \cdot z^2}{z^3 - z^2 + 0.1z + 0.02}$$

- **Obtenga y represente la respuesta al impulso del sistema.**

La función se expresa haciendo uso del método tf(), que permite también proporcionar una frecuencia de muestreo para la representación gráfica:

```
num = [0.4,0,0];
den = [1,-1,0.1,0.02];

% • Obtenga y represente la respuesta al impulso del sistema.
sys = tf(num,den,0.1); % 10 milisegundos de tiempo de muestreo
subplot(4,1,3);
impulse(sys);
title('Respuesta del sistema al impulso: ');
subtitle('0.4*z^2/(z^3 - z^2 + 0.1z + 0.02)');
legend('T(z)');
ylabel('Valor obtenido');
```

Resultado:

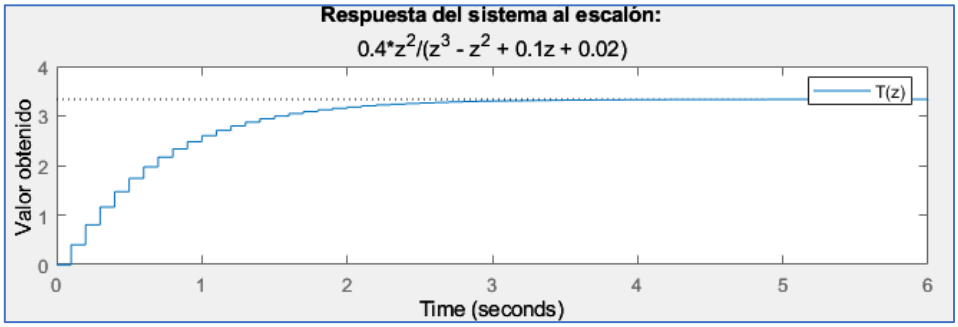


- **Obtenga y represente la respuesta del sistema ante una entrada escalón**

Este apartado es similar al anterior, haciendo uso esta vez de step() en vez de impulse():

```
subplot(4,1,4);
step(sys);
title('Respuesta del sistema al escalón:');
subtitle('0.4*z^2/(z^3 - z^2 + 0.1z + 0.02)');
legend('T(z)');
ylabel('Valor obtenido');
```

Resultado:



Se puede comprobar claramente la diferencia en el efecto que tienen las dos entradas en el sistema. El impulso aumenta el valor obtenido en la función de forma instantánea, y, tras este salto inicial, este empieza a decrecer.

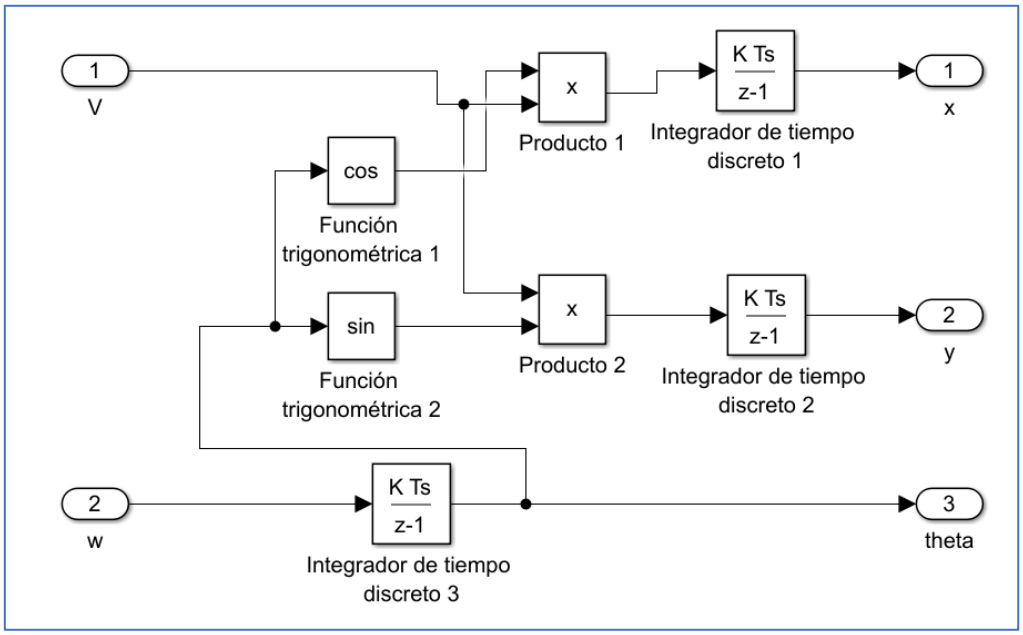
Por otro lado, el escalón supone una subida más paulatina, aunque luego mantiene este valor más elevado.

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

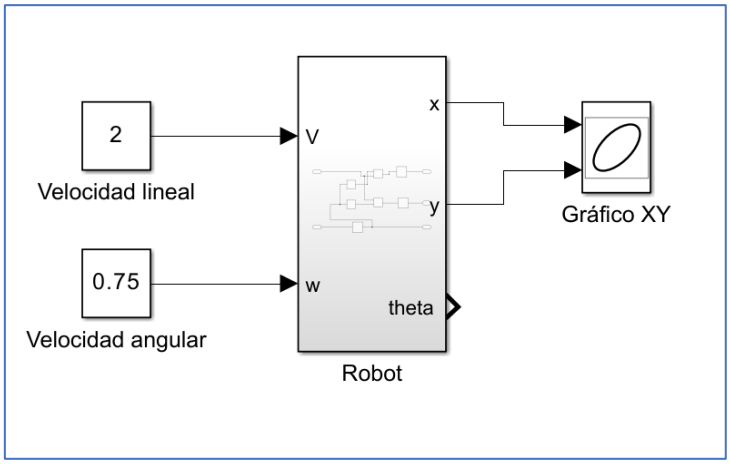
El estado del robot móvil se define mediante la posición en el entorno y su orientación, donde el subíndice k > 0 indica la variable de tiempo discreto. El movimiento del robot se rige por el siguiente modelo dinámico:

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

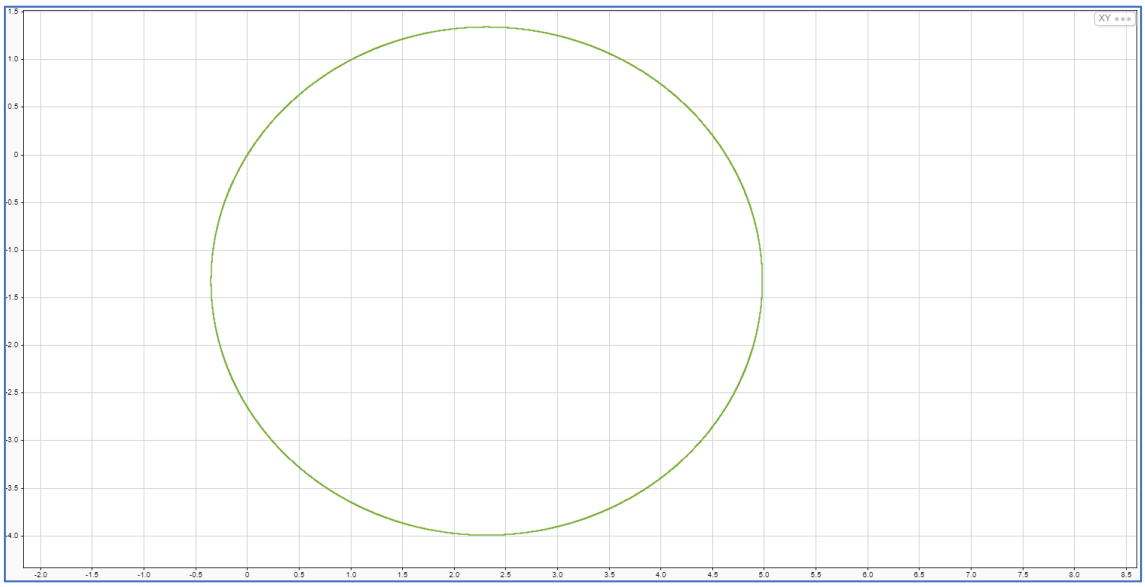
1. **Implemente el modelo de la Figura 2**



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.**

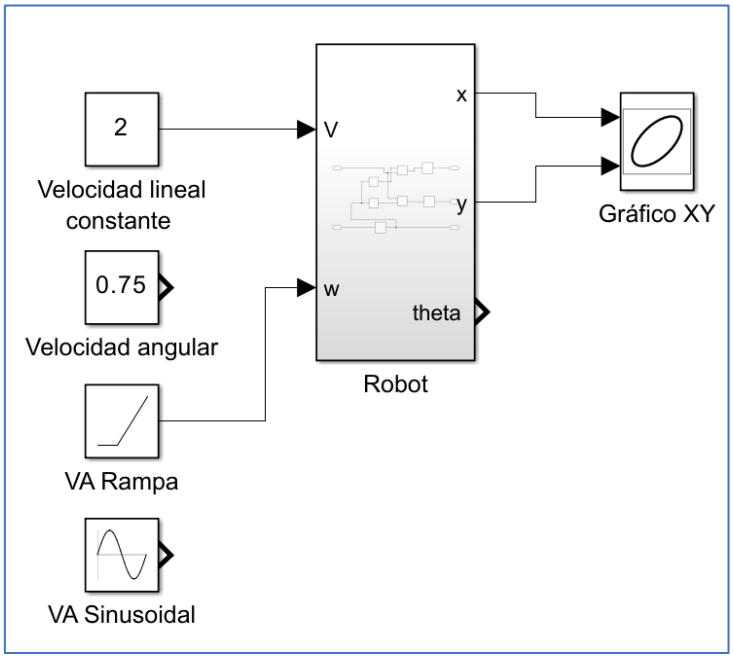


3. **Visualizando la gráfica generada por el módulo XY Graph, compruebe que el funcionamiento del robot se ajusta a lo esperado.**

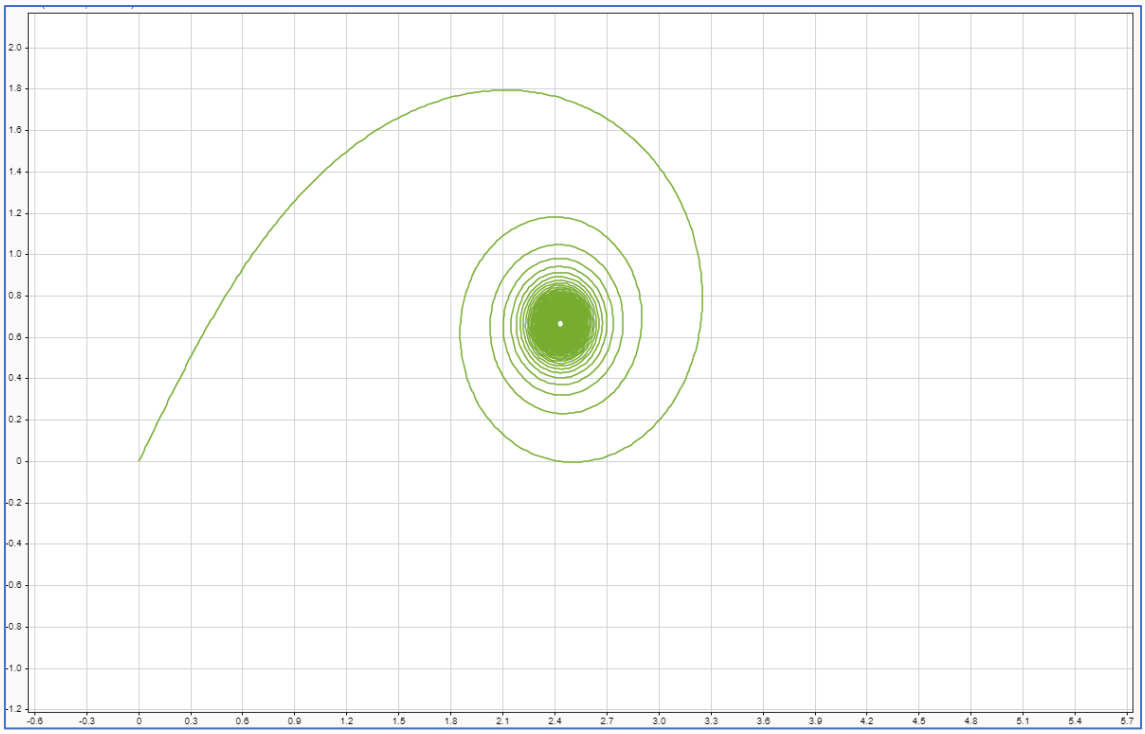


Partiendo del punto 0,0, el robot mantiene una velocidad lineal y angular constantes, por lo que su comportamiento, un movimiento circular, es el esperado.

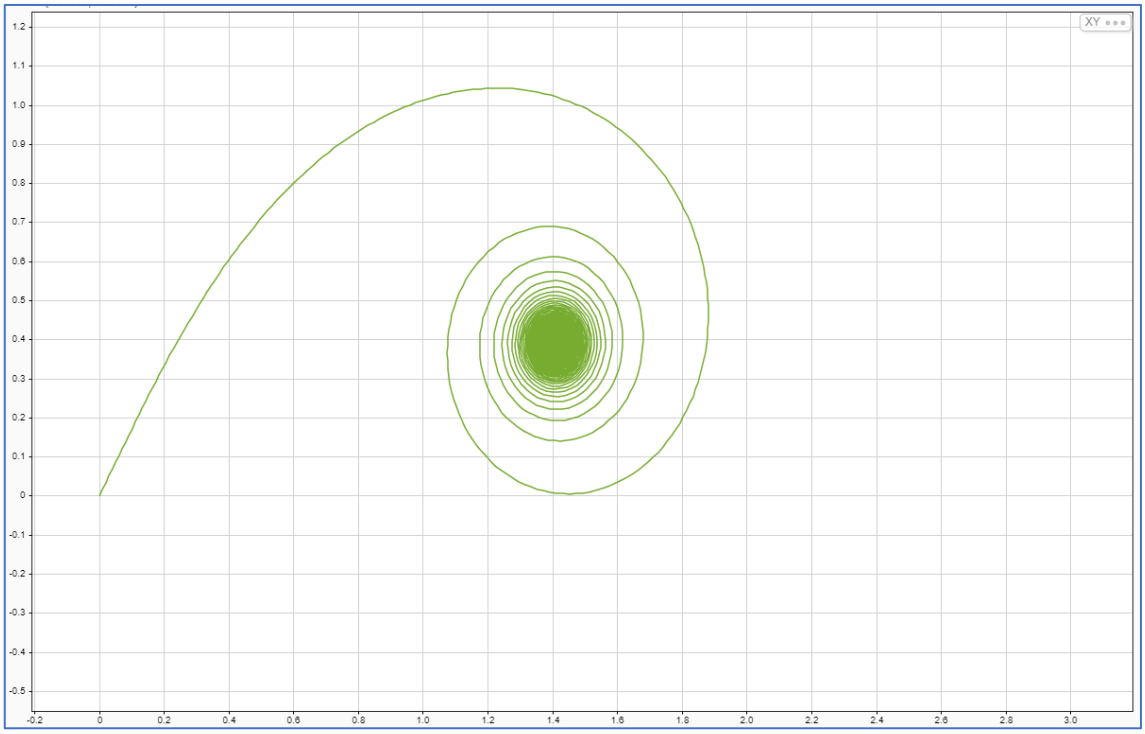
4. **Realice de nuevo la simulación con velocidades angulares no constantes (estas fuentes están disponibles en la librería de Simulink/sources):**



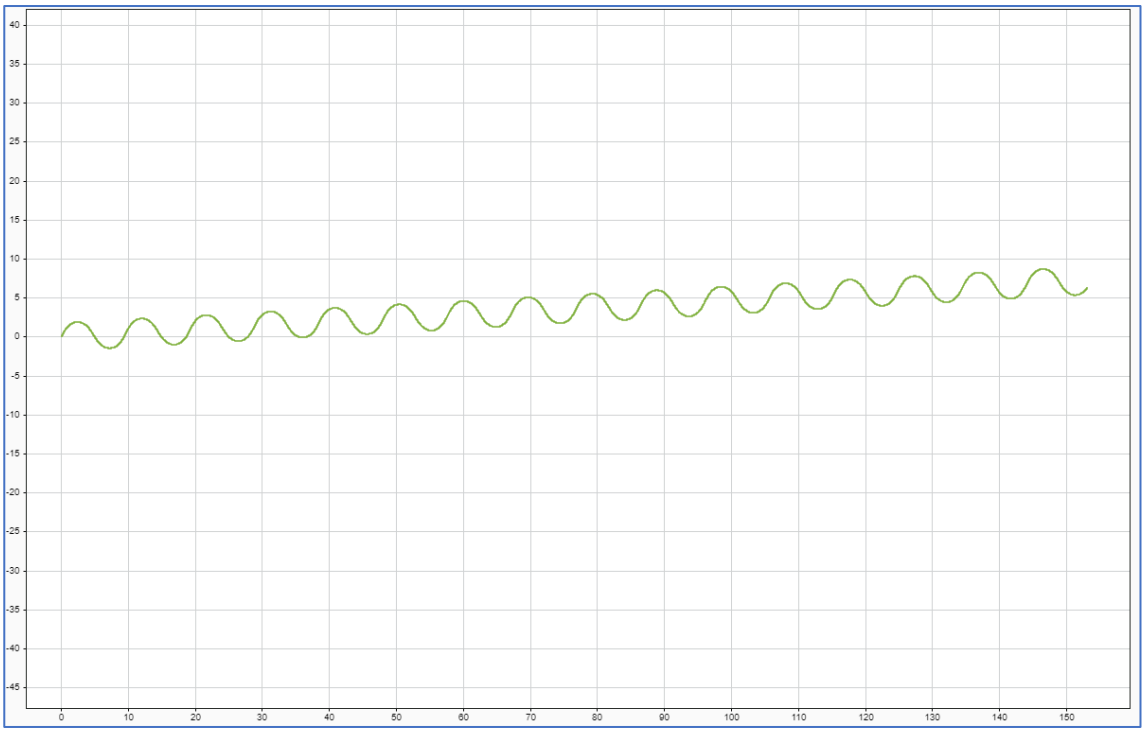
5. **Estudie de nuevo las trayectorias realizadas por el robot y compruebe que se ajustan al comportamiento esperado.**



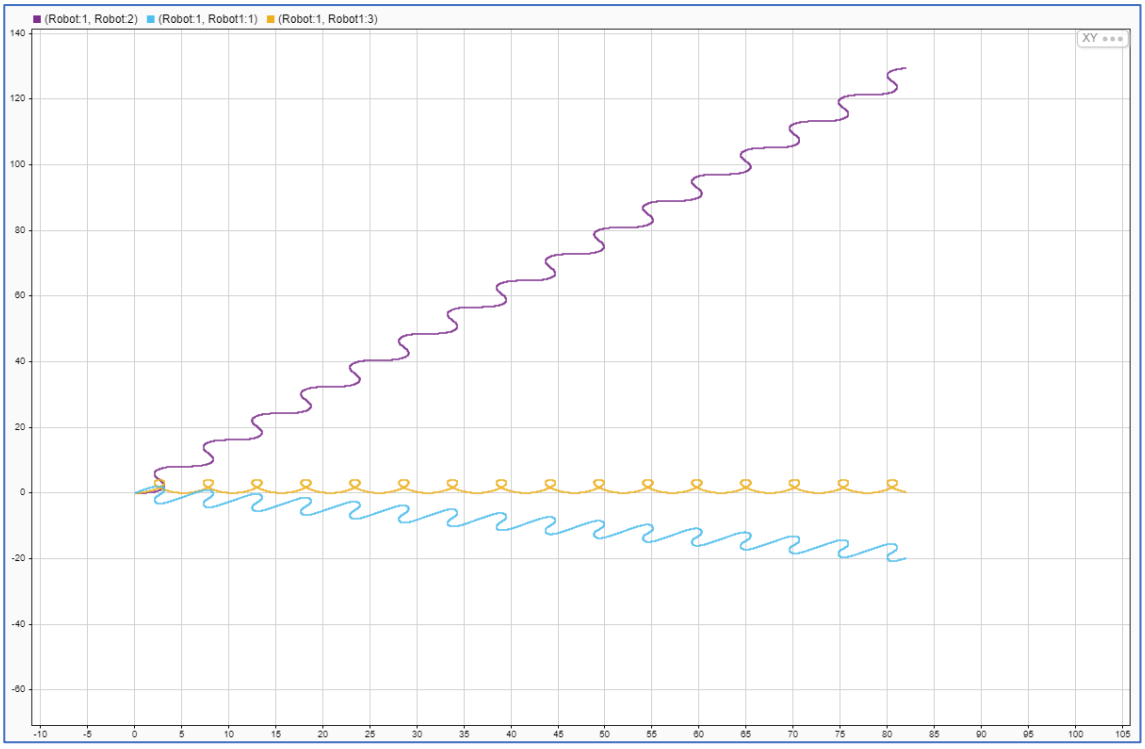
En el caso de la señal de rampa, un incremento constante de la velocidad angular resulta en un giro cada vez más cerrado, creando un recorrido en espiral. Cambiando el valor de la pendiente de la función rampa, se obtienen espirales más cerradas a medida que aumenta este número:



En el caso de la función sinusoidal, el movimiento del robot es, también, sinusoidal. Esto se debe a las alteraciones en positivo y negativo que sufre su velocidad angular, lo que hace que gire a la derecha y a la izquierda de forma periódica.

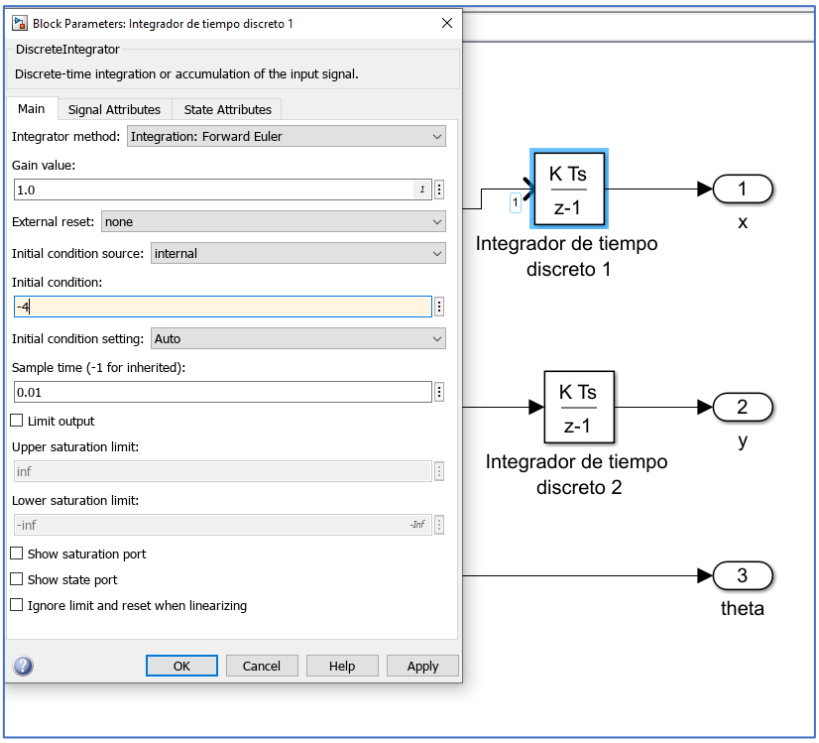


Alterar los valores de amplitud y frecuencia de la onda sinusoidal resulta en distintos tipos de movimiento, aunque todo ellos siguen una trayectoria equivalente a una línea recta:

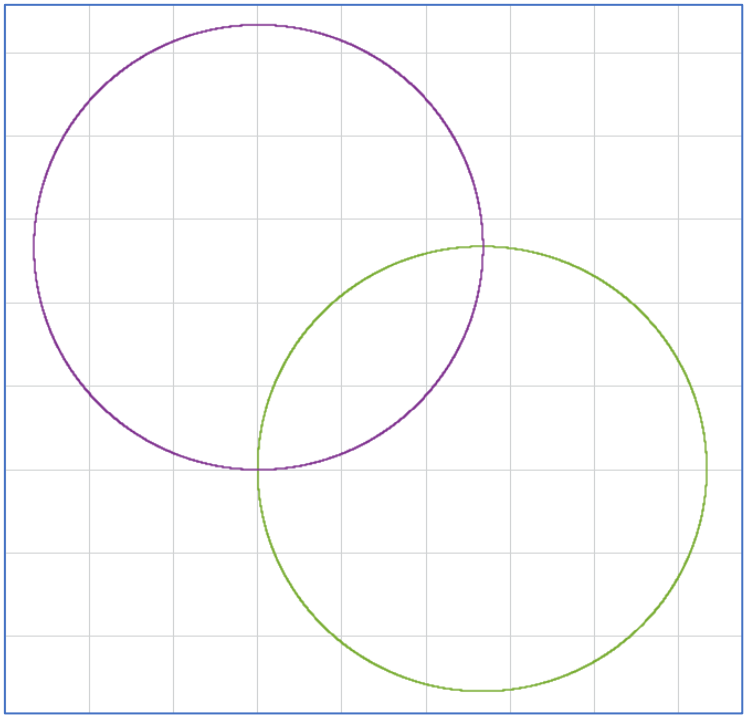


6. Modifique la posición inicial del robot para que comience a moverse desde el punto (-4,-4) y realice estas simulaciones de nuevo.

Para cambiar el punto de origen del movimiento del robot, se alteran los valores de condición inicial en los integradores de tiempo discreto 1 y 2:



Comparado con el movimiento circular anterior, se puede apreciar el efecto del nuevo punto de partida:



En el ejemplo de las múltiples funciones sinusoidales, se puede apreciar también un ángulo inicial distinto, cambiado en el integrador de tiempo discreto 3.