Universidad de Alcalá

Práctica 0. Introducción a Matlab (II)

Jaime Díez Buendía
Eduardo García Huerta

Sistemas de Control Inteligente

Grado en Ingeniería Computadores Grado en Ingeniería Informática Grado en Sistemas de Información

Universidad de Alcalá

Ejercicio 1. Matrices y vectores.

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

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

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

Para este apartado utilizamos la notación de Matlab para construir vectores y matrices. Los elementos de la misma fila separados por "," y, de filas distintas, separamos por ";"
Así, un vector fila, iría separado solo por comas y un vector columna sólo por puntos con coma, como en el ejemplo v.

```
%1.1
A = [1,2;3,4;5,6;7,8]
v= [14;16;18;20]
```

```
A =

     1     2
     3     4
     5     6
     7     8


v =

    14
    16
    18
    20
```

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

Obviamente, al tratarse de un vector columna, hay que concatenar la columna.
Esto lo haremos con la siguiente notación:

```
%1.2
B = [A,v]
```

Concatenará el elemento de la fila x de v a la fila x de A. Aquí podemos ver el resultado.

```
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.

Esto, según las pruebas que hemos hecho, se puede conseguir de varias formas.
Encontramos la forma manual digamos, la primera que podemos ver. Donde concatenamos todas las filas de B, pero para ello tenemos que conocer el número de estas y escribirlas una por una.
La otra forma es convertir B en un vector columna y trasponerlo.

```
%1.3
vector_fila = [B(1,:),B(2,:),B(3,:),B(4,:)]
otra_forma = B(:)'
vector_fila =

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


otra_forma =

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

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

Para este apartado nos sirve el método empleado en ele anterior de convertir una matriz a vector columna.

```
%1.4
vector_columna = B(:)

vector_columna =

     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”.

El número que nos introduce el usuario lo podemos coger gracias a un input, posterior a mostrarle el mensaje que queramos.

```
%matriz aleatoria
n= input("Introduce un número para el prden de la matriz: \n");
```

Una matriz de números aleatorios se genera utilizando funciones del estilo rand(). Hemos utilizado dos. La primera te genera una distribución uniforme de números del 0 al 1 y la segunda números en el rango que tu le introduzcas, del 1 al 50 por ejemplo.

```
%distribuida del 0 al 1
matrix= rand(n)
%distribuida en el rango que definas
matrix_2= randi([1,50],n)
```

Algunas funciones útiles: input, rand.

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

   a) Matriz generada.

   La matriz generada sale por pantalla automáticamente con las instrucciones anteriores.

```
%distribuida del 0 al 1
matrix= rand(n)
%distribuida en el rango que definas
matrix_2= randi([1,50],n)

Introduce un número para el prden de la matriz:
4

matrix =

    0.8147    0.6324    0.9575    0.9572
    0.9058    0.0975    0.9649    0.4854
    0.1270    0.2785    0.1576    0.8003
    0.9134    0.5469    0.9706    0.1419


matrix_2 =

    22    33    34    33
    46     2    38     9
    40    43    38    36
    48    47    20     2
```

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

   Decimos que queremos coger de todas las filas, los elementos desde el primero hasta el final, en intervalos de 2.

```
% columnas impares
impares = matrix_2(:, 1:2:end)
```

```
impares =

    22    34
    46    38
    40    38
    48    20
```

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

Cogemos los elementos de la matriz con intervalos de n+1 elementos.

```
%sacar diagonal
diagonal = matrix_2(1:n+1:end)
```

```
diagonal =

    22     2    38     2
```

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.

Las funciones que nos recomiendan, establecen su funcionalidad para columnas, por lo que como nos lo piden para filas, hemos de aplicar estas funciones a sus traspuestas.

```
%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
maximos = max(matrix_2')
minimos = min(matrix_2')
medias = mean(matrix_2')
varianzas = var(matrix_2')
```

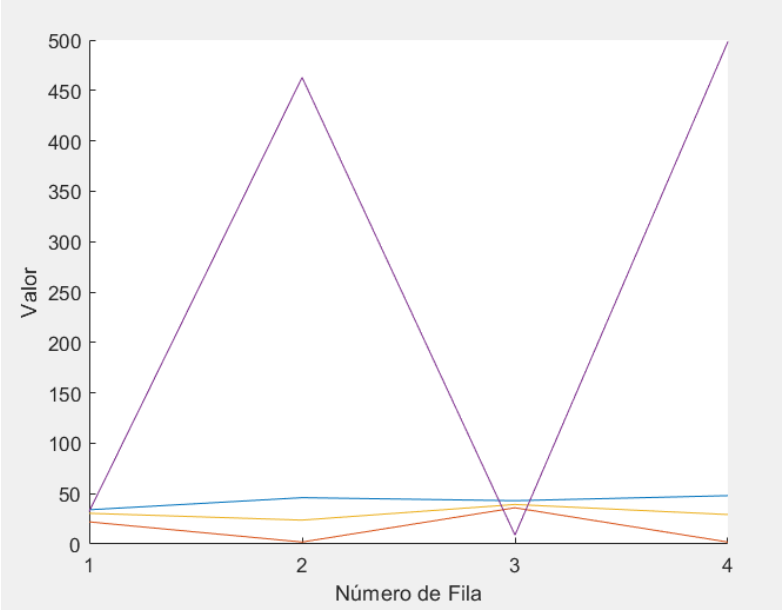
```
maximos =

    34    46    43    48


minimos =

    22     2    36     2


medias =

   30.5000   23.7500   39.2500   29.2500


varianzas =

   32.3333  462.9167    8.9167  498.2500
```

Para la gráfica, como queremos que las marcas sean los números de las filas, creamos un vector de estos números llamado filas en el que cogemos los números del 1 al número del tamaño de la matriz. Después creamos una figura con **figure** y, con el comando **hold on**, permitimos que se puedan superponer distintos valores de una misma gráfica (pintar varias líneas).

Luego, con el comando plot, que es el que se emplea para generar una gráfica, vamos representando los valores deseados con respecto al vector filas. Damos nombre a los ejes x e y con **xlabel** e **ylabel**

y, con el siguiente **set** y los valores de dentro de dicha función, establecemos que las marcas de la gráfica sean efectivamente las del vector filas, porque sino no nos funcionaba correctamente.

```
%representar con plot

filas = 1:size(matrix_2,1);

figure;
hold on;
plot(filas, maximos);
plot(filas, minimos);
plot(filas, medias);
plot(filas, varianzas);
xlabel('Número de Fila');
ylabel('Valor');
set(gca,'XTick',filas); % Establecer los valores del eje x manualmente
```



Algunas funciones útiles: disp, for, diag, det, max, min, mean, var, figure, plot, hold on, hold off, title, xlabel, ylabel, size.

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:

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

Esto se puede hacer fácilmente con un input y sentencias if else.

```
%pedir al usuario las dimensiones de la matriz

dimensiones = input("Introduce las dimensiones de la matriz: \n" + ...
    "Debes introducirlo como un vector ( entre [] )")


if (size(dimensiones:1)==1)
    x= dimensiones(1)
    y= dimensiones(1)
else
    x= dimensiones(1)
    y= dimensiones(2)
end

A = IntroducirMatriz(x,y)
B = IntroducirMatriz(x,y)
```

```
Introduce las dimensiones de la matriz:
Debes introducirlo como un vector ( entre [] )[2,2]

dimensiones =

     2     2


x =

     2


y =

     2
```

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. **function Matriz = IntroducirMatriz(Dimensiones);**

```
A = IntroducirMatriz(x,y)
B = IntroducirMatriz(x,y)
```

_ejer3.m | IntroducirMatriz.m | pl0_ejer4.m

```
function matriz = IntroducirMatriz(X, Y)
```

La función debe llamarse igual que el archivo para que funcione

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 conseguir esto, hay que tener en cuenta dos aspectos clave.
El primero es el de crear una matriz de todo ceros con las dimensiones requeridas. Se consigue con **zeros(X,Y).**
La funcioalidad general de la función auxiliar que hemos creado, se basa en los bucles **for** típicos para recorrer una matriz y, las sentencias **if** para comprobar si se trata de un número o de la letra “r” lo introducido por el usuario.
Aquí es donde viene el otro punto importante: para poder comprobar si se trata de la letra “r” el carácter introducido por el usuario, convertimos la cadena en un string. (se consigue con esa ‘s’ en el segundo parámetro del imput)

```
n = input("Introduce un número:\n",'s');
```

Es importante también, después hecha la comprobación, si no se trata de este carácter, convertir el string a número para meterlo correctamente en la matriz. Esto se haría con la función **str2num()**.

```
function matriz = IntroducirMatriz(X, Y)

    % Inicializa una matriz de X por Y con ceros
    matriz = zeros(X,Y);

    for i = 1:X
        for j = 1:Y
            n = input("Introduce un número:\n",'s');
            if strcmp(n, 'r')
                % Si el usuario ingresó 'r', llena el resto de la matriz con valores aleatorios
                matriz(i:end, j:end) = rand(X - i + 1, Y - j + 1);
                % Rompe el bucle, ya que el usuario eligió llenar el resto aleatoriamente
                return;
            else
                matriz(i, j) = str2num(n);
            end
        end
    end
end
```

4. Calcule y muestre por pantalla:
   - Las matrices generadas A y B

   Aquí podemos ver tanto cómo nos pide números la función como las matrices A y B.

```
Introduce un número:
1
Introduce un número:
2
Introduce un número:
3
Introduce un número:
4

A =

     1     2
     3     4

Introduce un número:
4
Introduce un número:
3
Introduce un número:
6
Introduce un número:
7

B =

     4     3
     6     7
```

   - La transpuesta e inversa de cada una de las matrices

   La traspuesta se obtiene utilizando la comilla a la derecha de la matriz y la inversa con la función **inv().**

```
%traspuestas
A_tras= A'
B_tras=B'

%inversas

if x==y
    A_inv= inv(A)
    B_inv= inv(B)
else
    disp("No te muestro la inversa xq no es cuadrada la matriz")
end
```

```
A_tras =

     1     3
     2     4


B_tras =

     4     6
     3     7


A_inv =

   -2.0000    1.0000
    1.5000   -0.5000

B_inv =

    0.7000   -0.3000
   -0.6000    0.4000
```

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

Utilizamos las funciones **det()** y **rank()** en este caso.

```
%determinante y rango
if x==y
    A_det = det(A)
    B_det = det(B)
else
    disp("No te muestro el determinante xq no es cuadrada la matriz")
end
A_rango = rank(A)
B_rango= rank(B)
```

```
A_det =

    -2


B_det =

   10.0000


A_rango =

     2


B_rango =

     2
```

- El producto de A y B (matricial y elemento a elemento)

El producto matricial es la operación convencional de multiplicación de matrices y se hace con el operador "*".
El producto elemento a elemento, como su nombre indica, es una operación en la que cada elemento de una matriz se multiplica por el elemento correspondiente de otra matriz del mismo tamaño. Se consigue con el operador ".*" (el punto en Matlab significa que la operqción que va justo detrás suya se haga elemento a elemento)

```
%producto matricial
if x==y
    prod_matricial= A*B
else
    disp("No te muestro la multipl xq no es cuadrada la matriz")
end


%producto elemento a elemento
prod_elementos = A.*B
```

```
prod_matricial =

    16    17
    36    37


prod_elementos =

     4     6
    18    28
```

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

Hacemos un vector cuyos elementos son los elementos de las filas nº 1 de las matrices A y B. Con A(1,:) cogemos todos los elementos (:) de la fila 1 (1) de A.

```
%vector fila de las primeras filas de A y B
primeras_filas=[A(1,:) B(1,:)]
```

```
primeras_filas =

     1     2     4     3
```

- Un vector columna obtenido concatenando la primera columna de cada una de las matrices En caso de que no sea posible realizar alguno de los cálculos solicitados, indíquelo por pantalla.

De manera análoga conseguimos lo mismo con las columnas.

```
%vector columna de las primeras columnas de A y B
primeras_columnas=[A(:,1) ; B(:,1)]
```

```
primeras_columnas =

     1
     3
     4
     6
```

Algunas funciones útiles: input, for, if..else, det, inv, pinv, rand, randn, rank, size.

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:

Primero, creamos dos vectores vacíos donde guardaremos los resultados.

```
tamanos=1:25;
tiempos_rango = zeros(size(tamanos));
tiempos_determinante = zeros(size(tamanos));
```

- La matriz se rellenará con valores aleatorios.
- El tiempo necesario para cada operación debe obtenerse por separado.

Matrix es la matriz aleatoria del tamaño i.
Con la función **tic**, empezamos a contar el tiempo, después ejecutamos la operación que queramos (rank o det) y posteriormente, con **toc** paramos el contador y guardamos ese valor en su posición correspondiente del vector creado anteriormente.

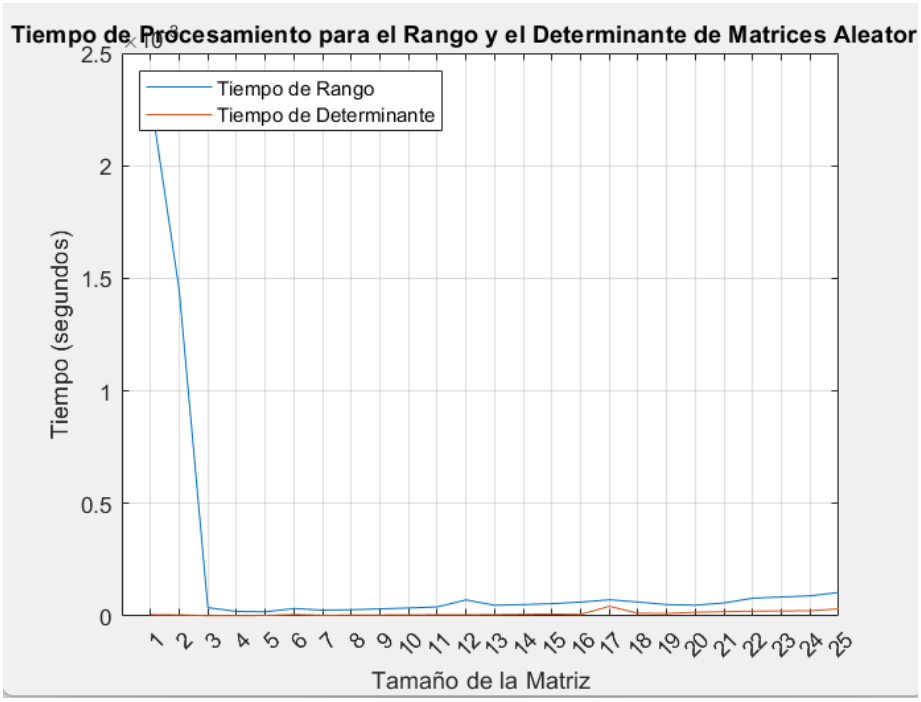
```
for i = 1:length(tamanos);
    matrix = rand(tamanos(i));


    tic;
    rank(matrix);
    tiempos_rango(i) =toc;

    tic;
    det(matrix);
    tiempos_determinante(i)= toc;

end
```

- 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.

Para todo esto, utilizamos las funciones ya explicadas anteriormente.

Adicionalmente usamos **title()** para el titulo y activamos la cuadrícula con **grid on .**

```
% Crear una gráfica para mostrar los tiempos de procesamiento
figure;
plot(tamanos, tiempos_rango, 'DisplayName', 'Tiempo de Rango');
hold on;
plot(tamanos, tiempos_determinante, 'DisplayName', 'Tiempo de Determinante');
xlabel('Tamaño de la Matriz');
ylabel('Tiempo (segundos)');
legend('Location', 'Northwest');
title('Tiempo de Procesamiento para el Rango y el Determinante de Matrices Aleatorias');
grid on;
set(gca,'XTick',tamanos); % Establecer los valores del eje x manualmente
% Mostrar la leyenda y la cuadrícula en la gráfica
legend('show');
grid on;
```



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

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:

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

Creamos una malla de 100x100 con meshgrid a partir de dos vectores con 100 números cada uno, equidistantes en una proporción.

```
% Definir un rango para x e y
x = linspace(-5, 20, 100);
y = linspace(5, 20, 100);

% Crear una cuadrícula de coordenadas (x, y)
[X, Y] = meshgrid(x, y);
```

Calculamos los valores de la coordenada Z en función de la X e Y que acabamos de crear.

```
% Calcular los valores de z en función de x e y
Z = Y .* sin(pi * X / 10) + 5 * cos((X.^2 + Y.^2) / 8) + cos(X + Y) .* cos(3 * X - 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

Para que ocupe la parte central entera, utilizamos **subplot(2, 2, [1, 2]);**

Los dos primeros números quiere decir que crearemos 2 por dos huecos para figuras en la hoja y, el vector [1,2] quiere decir que la siguiente figura ocupará la primera y segunda posición de las 4 disponibles.

Para la gráfica de superficie usamos **surf().**

```
% Crear una figura
figure;

% Subtrama superior central: superficie
subplot(2, 2, [1, 2]);
surf(X, Y, Z);
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Superficie');
```

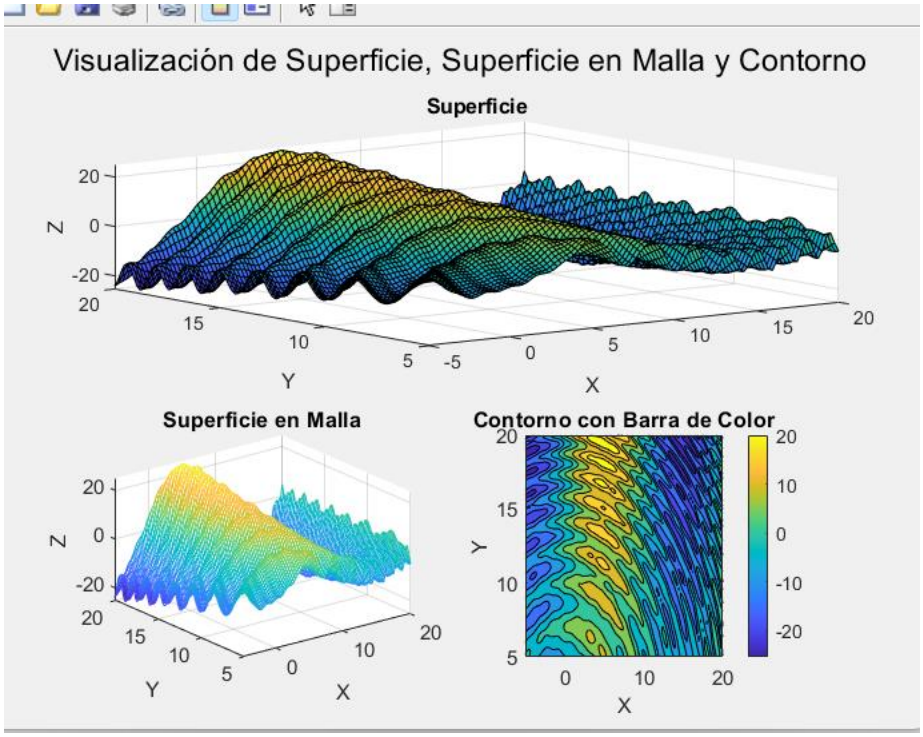
Para representar la superficie en forma de malla utilizamos **mesh**().

```
% Subtrama inferior izquierda: superficie en forma de malla
subplot(2, 2, 3);
mesh(X, Y, Z); %%surf es casi lo mismo pero para malla
xlabel('X');
ylabel('Y');
zlabel('Z');
title('Superficie en Malla');
```

Para representar el contorno utilizamos **contfourf**() y para la barra de color **colorbar().**

```
% Subtrama inferior derecha: contorno con barra de color
subplot(2, 2, 4);
contourf(X, Y, Z); %%lo mismo q las anteriores pero para contorno
colorbar;
xlabel('X');
ylabel('Y');
title('Contorno con Barra de Color');

% Ajustar el espacio entre las subtramas
sgtitle('Visualización de Superficie, Superficie en Malla y Contorno');
```

Así queda el resultado:



Algunas funciones útiles: meshgrid, mesh, surf, contourf xlabel, ylabel, subplot.

Ejercicio 6. Sistemas lineales

Dados los siguientes sistemas lineales de 10 ecuaciones con 4 incógnitas (x_1, x_2, x_3, x_4)

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

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

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

```
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]

a= [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]

b= [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

   Primero invertimos las matrices. Para lo que usamos **pinv()**, una función que nos calcula la pseudoinversa, ya que si la función no es cuadrada no se podría hallar la inversa.

```
A_inv= pinv(A)
B_inv= pinv(B)
condicionA= cond(A_inv)
condicionB= cond(B_inv)
```

   Utilizamos la función **cond()** para averiguar las condiciones.

```
condicionA= cond(A)
condicionB= cond(B)
```

```
A_inv =

   -0.0182   -0.0085   -0.0275    0.0302   -0.0111    0.0727   -0.0030   -0.0003    0.0376   -0.0210
   -0.0178    0.0262    0.0504   -0.0240    0.0390   -0.0048    0.0269   -0.0193   -0.0120    0.0053
    0.0329    0.0171   -0.0252   -0.0098    0.0011   -0.0171    0.0064    0.0357    0.0251   -0.0189
    0.0307   -0.0176    0.0180    0.0368   -0.0150    0.0002   -0.0185    0.0049   -0.0117    0.0572


B_inv =

   -0.6361         0   -0.3177         0   -0.0002         0    0.3183         0    0.6357         0
    0.0000   -0.6369    0.0000   -0.3185    0.0000   -0.0000   -0.0000    0.3185   -0.0000    0.6369
    0.6702         0    0.4348         0    0.2001         0   -0.0353         0   -0.2699         0
   -0.0000    2.6764   -0.0000    1.4382   -0.0000    0.2000    0.0000   -1.0382    0.0000   -2.2764


condicionA =

    2.7257


condicionB =

   36.7102
```

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

Utilizamos **linsolve()** para resolver ecuaciones.

```
sol_A= linsolve(A,a)
sol_B= linsolve(B,b)
```

```
sol_A =

   -2.2571
    4.9336
    5.2986
    3.5649


sol_B =

    6.3593
    6.3694
  316.2992
  216.2357
```

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

Creamos un vector de ruido y se lo sumamos a los vectores a y b.

```
ruido = randn(size(a)); % Vector de ruido

a_ruido= a+ruido
b_ruido= b+ruido
```

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

```
sol_a_ruido =

   -2.2914
    4.8328
    5.2285
    3.5900


sol_b_ruido =

    6.6872
    6.1511
  315.4942
  216.2184
```

```
sol_A =

   -2.2571
    4.9336
    5.2986
    3.5649


sol_B =

    6.3593
    6.3694
  316.2992
  216.2357
```

Compare los resultados obtenidos en cada caso.

El ruido altera sensiblemente los resultados, pero no de manera muy drástica.

Algunas funciones útiles: pinv y linsolve.

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.

Creamos una función con la forma que nos indican y, desde otro script la llamamos introduciendo los vectores que queramos.

```
[raices, reales, complejas] = pl0_ejer7([1, 2, 2], [1, 3]);
```

```
function [solucion, reales, complejas] = pl0_ejer7(poli_1, poli_2)
```

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

Con varios prints y un input, solucionamos el problema.

```
% Solicita si la solución se aplica a uno de los polinomios o al producto
fprintf('Seleccione el polinomio para encontrar sus raíces:\n');
fprintf('1. Polinomio 1\n');
fprintf('2. Polinomio 2\n');
fprintf('3. Producto de Polinomios\n');
opcion = input('Ingrese el número de la opción deseada: ');
```

```
>> llamada_ejer7
Seleccione el polinomio para encontrar sus raíces:
1. Polinomio 1
2. Polinomio 2
3. Producto de Polinomios
Ingrese el número de la opción deseada:
```

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

Según la opción que eliges calcula unas raíces u otras.

```
% Calcula las raíces y su clasificación
if opcion == 1
    solucion = roots(poli_1);
elseif opcion == 2
    solucion = roots(poli_2);
elseif opcion == 3
    solucion = roots(conv(poli_1, poli_2));
else
    error('Opción no válida');
end
```

Posteriormente, una vez obtenidas las raíces, las clasificamos.

```
% Clasifica las raíces en reales y complejas
reales = sum(imag(solucion) == 0)
complejas = length(solucion) - reales
```

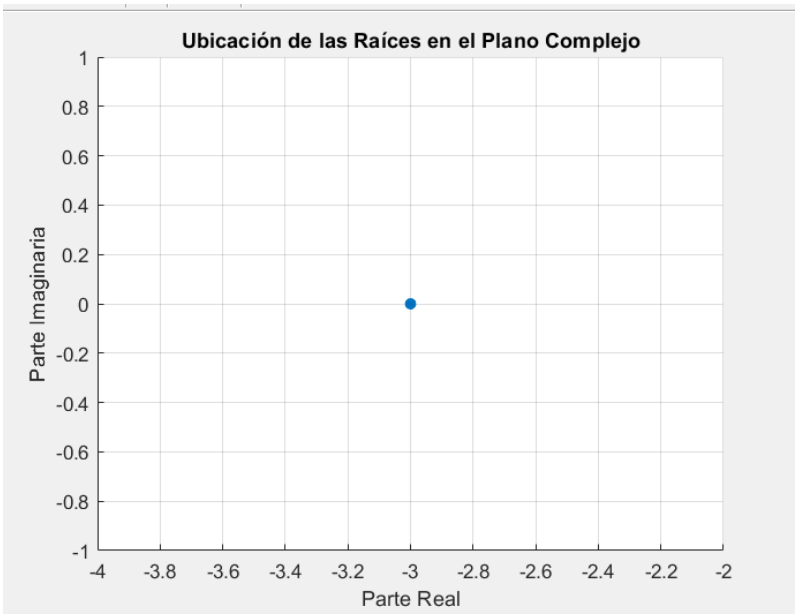
```
reales =

     1


complejas =

     0
```

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

Finalmente representamos las raíces con la función **scatter()**.

Se utiliza para crear un diagrama de dispersión (scatter plot) que muestra las partes real e imaginaria de las soluciones complejas contenidas en el vector solucion. El argumento 'filled' se utiliza para rellenar los puntos del diagrama de dispersión con colores sólidos.

```
    % Representa en el plano complejo la ubicación de las raíces obtenidas
    figure;
    scatter(real(solucion), imag(solucion), 'filled');
    xlabel('Parte Real');
    ylabel('Parte Imaginaria');
    title('Ubicación de las Raíces en el Plano Complejo');
    grid on;
end
```



Algunas funciones útiles: cla, if…else, plot, xlabel, error, input, roots, ylabel, figure, isreal, size, hold off, length, switch…case, hold on, nargin, title

Universidad de Alcalá

Práctica 0. Introducción a Matlab (II)

Jaime Díez Buendía
Eduardo García Huerta

Sistemas de Control Inteligente

Grado en Ingeniería Computadores Grado en Ingeniería Informática Grado en Sistemas de Información

Universidad de Alcalá

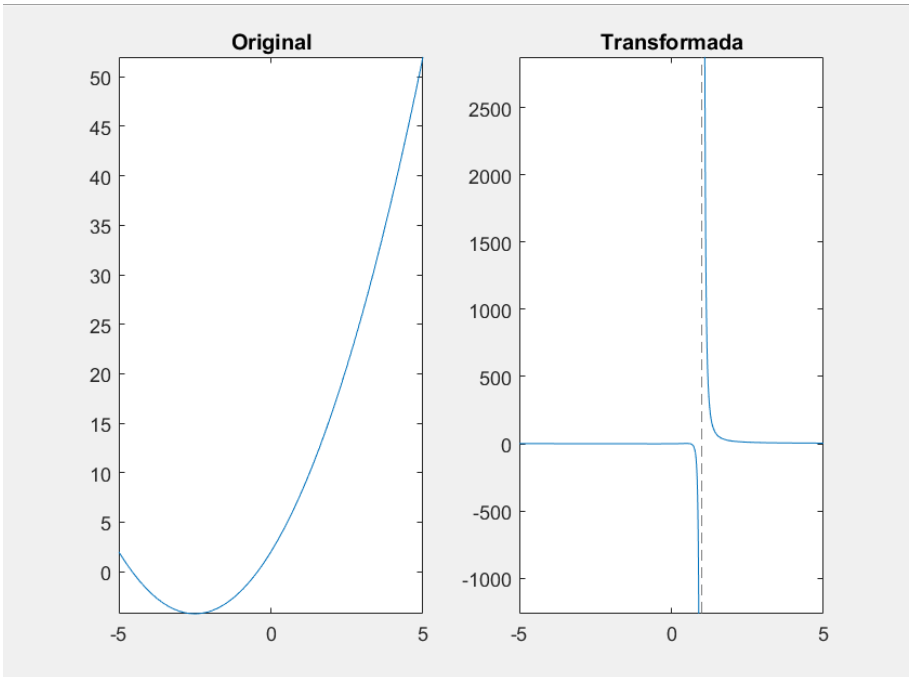
Ejercicio 1. Transformadas de señales.

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

Para realizar este apartado primero debemos definir una variable de programa que sea equivalente a la función dada por el enunciado, para ello, debemos crear un símbolo o una variable que represente la variable k. Una vez hayamos definido la función podremos obtener la transformada mediante la función **ztrans()**. Una vez obtenida ya podremos comparar la función original y la transformada mediante gráficas gracias a la función **fplot()**.

```
%Declaración de la función del primer apartado
syms k;
fk = 2+5*k+k^2;
transformada = ztrans(fk);

%Representación grafica
figure;
hold on;

subplot(1,2,1);
fplot(fk);
title('Original');

subplot(1,2,2);
fplot(transformada);
title('Transformada');

hold off
```

El resultado de la ejecución de este código quedaría de la siguiente manera:

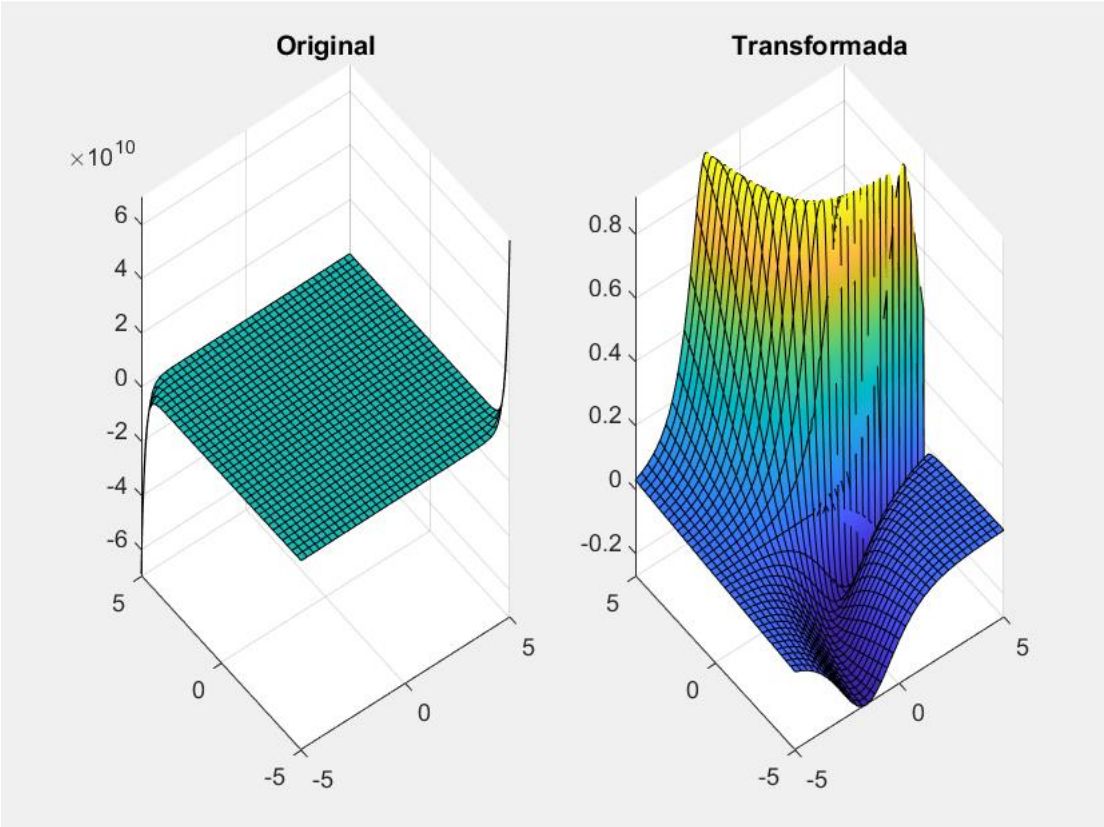


2. Obtenga la transformada z de la siguiente función: f(z) = sen(k) * e^{-ak}. Represente gráficamente, de nuevo, la señales original y transformada.

Este apartado, al igual que el anterior, hay que definir la función para después transformarla. La diferencia entre este y el anterior es que para representar gráficamente esta función y su transformada se debe de hacer mediante la función **fsurf()**.

```
%Declaración de la función del segundo apartado
syms a;
fk2 = sin(k)*exp(-a*k);
transformada2 = ztrans(fk2);

%Representación grafica
figure;
hold on;

subplot(1,2,1);
fsurf(fk2);
title('Original');

subplot(1,2,2);
fsurf(transformada2);
title('Transformada');

hold off
```

El resultado de este código quedaría de la siguiente manera:



3. Dada la siguiente función de transferencia discreta:

$$T(z) = \frac{0{,}4 * z}{z^3 - z^2 + 0{,}1z + 0{,}02}$$

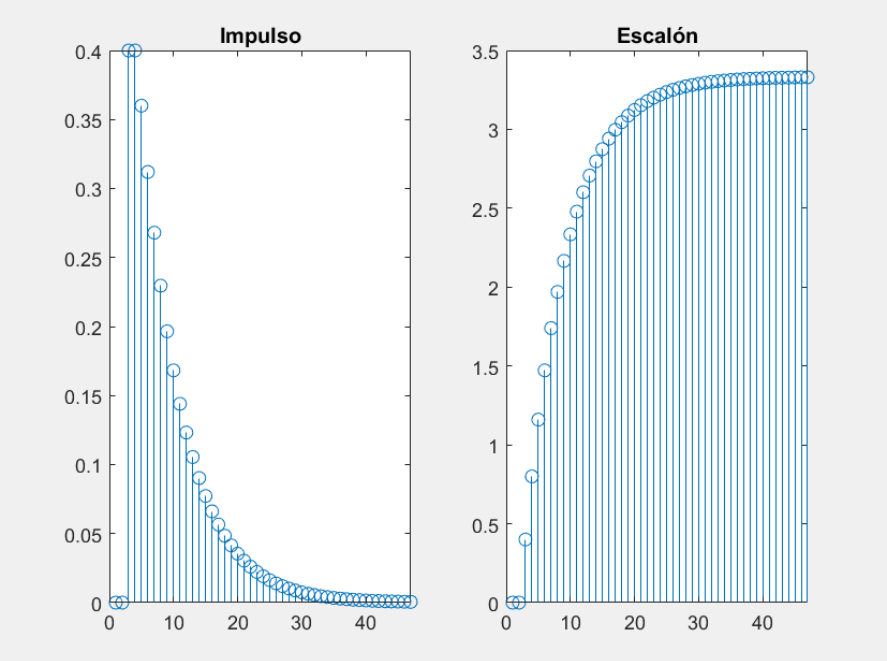
- Obtenga y represente la respuesta al impulso del sistema.
- Obtenga y represente la respuesta del sistema ante una entrada escalón.

Al igual que los dos apartados anteriores, primero hay que definir la función para después obtener la respuesta de impulso mediante la función "impulse" y la respuesta del sistema ante la entrada del escalón mediante la función **step()**. Una vez obtenido las respuestas para representarlas gráficamente se usará la función **stem()**.

.

```
%Declaración de la función del tercer apartado
T1 = tf([0.4, 0],[1, -1, 0.1, 0.02],1);
impulso = impulse(T1);
escalon = step(T1);

%Representación grafica al impulso del sistema
figure;
hold on;

subplot(1,2,1);
stem(impulso);
title('Impulso');

%Representación grafica al impulso del sistema
subplot(1,2,2);
stem(escalon);
title('Escalón');

hold off
```

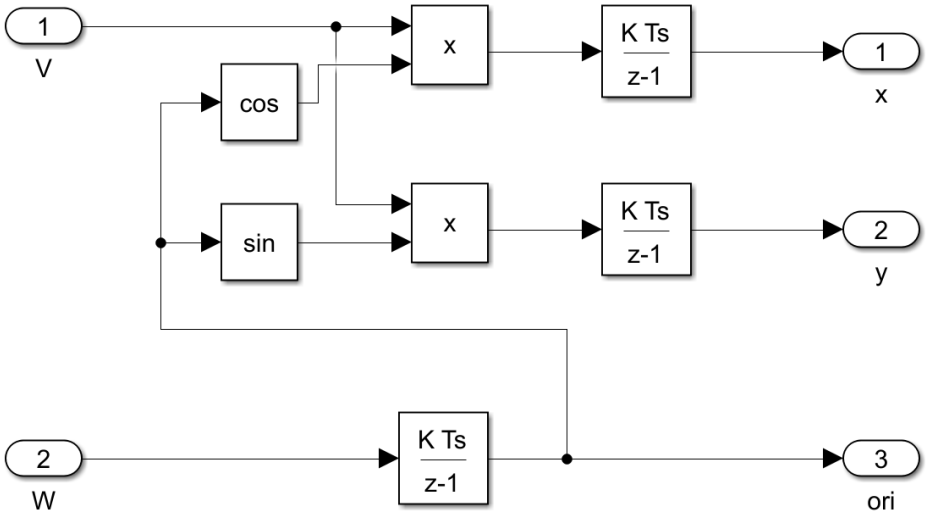
El resultado de este código quedaría de la siguiente manera:



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

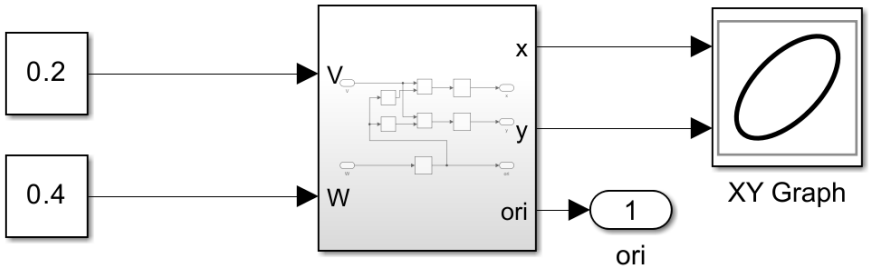
1. Implemente el modelo de la Figura 2 (todos los bloques utilizados pueden encontrarse en la librería estándar de Simulink).

Tras realizar los pasos realizados previamente en el documento de la práctica, queda un subsistema con la siguiente estructura:



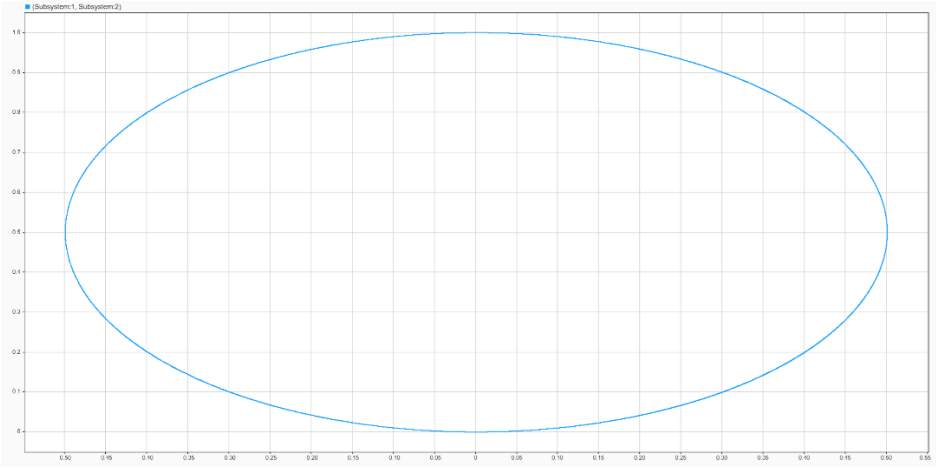
2. Una vez completado el apartado anterior, cree el subsistema mostrado en la Figura 1 y simule su funcionamiento con velocidad lineal y angular constante creando el sistema mostrado en la Figura 3. Configure los parámetros de la simulación (menú “Simulation/Model Simulation Parameters” y menú Simulation/Pacing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

Tras ponerle al robot velocidad lineal y la velocidad constante el robot quedaría de la siguiente manera:



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

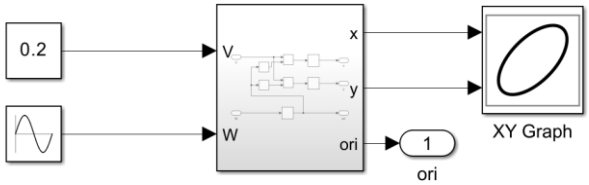
Una vez entramos en la gráfica y ejecutamos el programa, el robot se moverá de una manera constante formando un círculo.



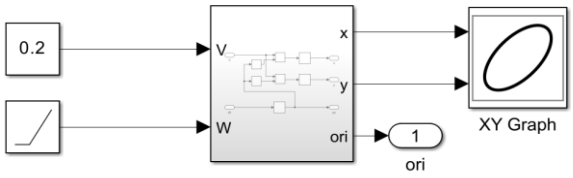
En la gráfica se ve como una elipse, esto se debe a que el eje X y el eje Y no tienen la misma escala.

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

   Para realizar esto cambiamos la velocidad angular por un módulo llamado rampa y por otro llamado w sinusoidal.



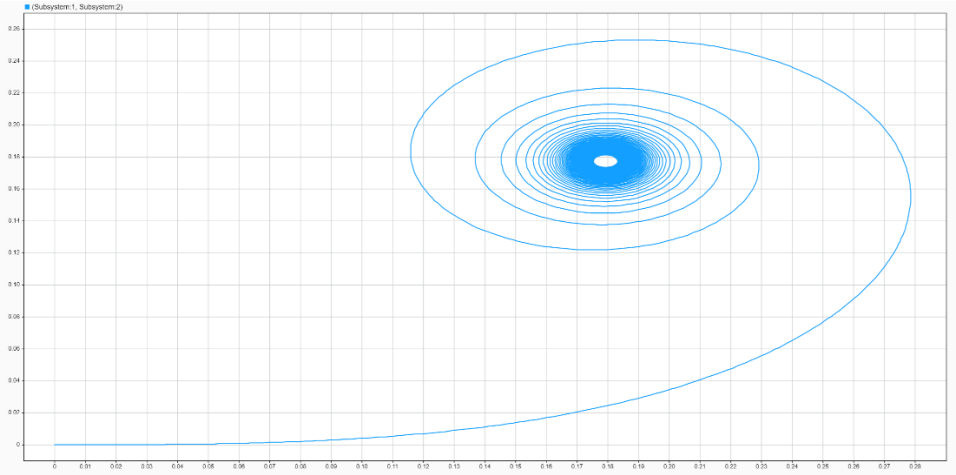
Sinusoidal



Rampa

- Rampa

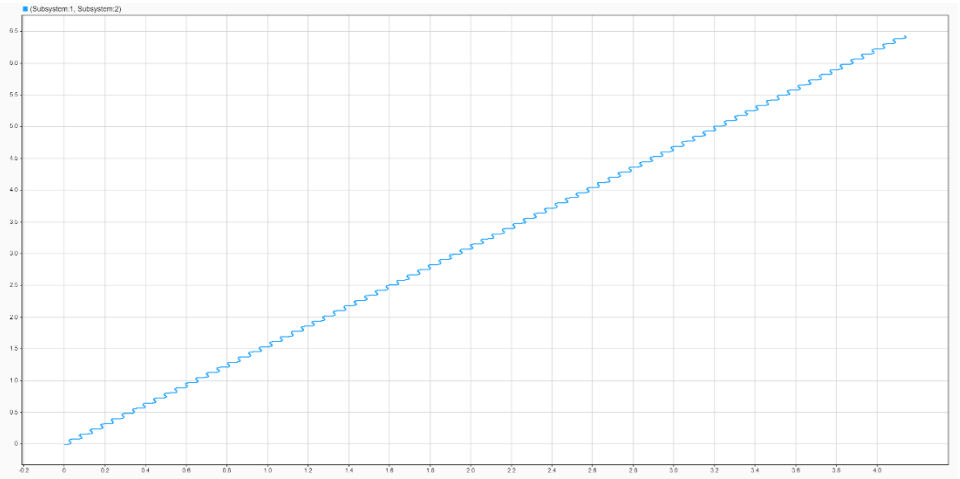
En este apartado se mostrará el resultado de la simulación con una velocidad angular que aumenta como una rampa.



Podemos observar que el resultado de la simulación es un recorrido en forma de espiral.

- W sinusoidal

En este apartado se mostrará el resultado de la simulación con una velocidad angular que disminuye y aumenta sucesivamente.



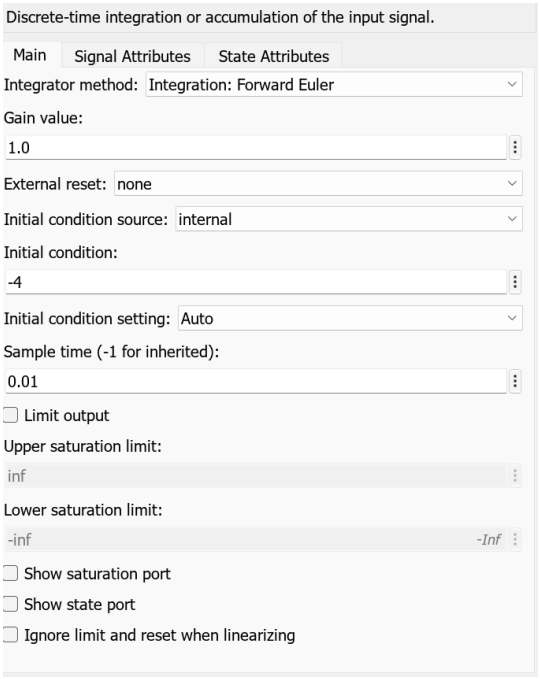
Se puede observar que el resultado de la simulación de este caso es una diagonal creciente que zigzaguea.

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

   El resultado de una nueva simulación en los dos casos es igual, esto se debe a que presentan los mismos valores iniciales, por lo que el resultado de las simulaciones será el mismo.

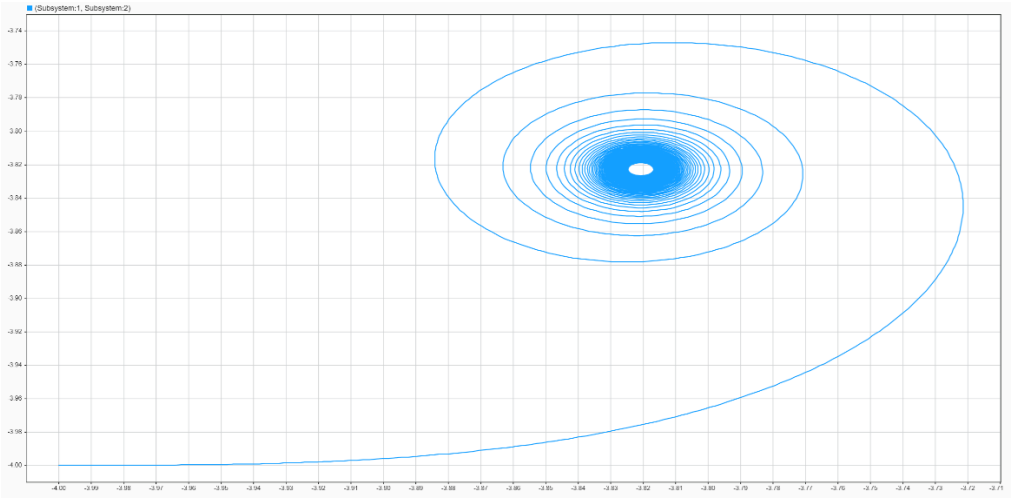
6. Modifique la posición inicial del robot para que comience a moverse desde el punto (-4,-4) y realice estas simulaciones de nuevo.

   Para cambiar la posición inicial del robot debemos cambiar las coordenadas X e Y el subsistema, para esto debemos cambiar la condición inicial de los módulos discrte-time integrator que forman a X y a Y, poniendo la coordenada deseada.



Los nuevos resultados de la ejecución son los siguientes:

- Rampa



- W sinusoidal