Universidad de Alcalá

Escuela Politécnica Superior

Universidad de Alcalá

Sistemas de Control Inteligente

Práctica 0

Laboratorio Lunes 12:00 – 14:00

Grado en Ingeniería Informática – Curso 2019/2020

Marcos Barranquero Fernández – 51129104N

Eduardo Graván Serrano – 03212337L

ÍNDICE

INTRODUCCIÓN

Mediante esta práctica se pretende introducir a los alumnos al entorno de desarrollo y lenguaje de programación matemática de MatLab. Se han realizado los ejercicios propuestos en clase.

PRÁCTICA 0 – PARTE 1

EJERCICIO 1

El ejercicio 1 consiste en realizar una matriz y un vector, para familiarizarse con la sintaxis de Matlab.

```
% clear all permite limpiar la memoria de datos de programas anteriores.
clear all;

% Ejercicio 1.1

% Creo matriz A
A = [1 2;
    3 4;
    5 6;
    7 8; ];

% Creo vector V
V = [14; 16; 18; 20]

% Ejercicio 1.2

B = [A V]; % concateno vectores

% Visualizo B
disp(B)

% Ejercicio 1.3

% Permite copiar partes de matrices. Podemos visualizar
% por consola si no ponemos el ";" al final.
vector_fila = [B(1, :); B(2, :); B(3, :)]

% Ejercicio 1.4

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

EJERCICIO 2

En este ejercicio se realizan varias operaciones con matrices. En primer lugar, leemos el tamaño de la matriz y generamos una matriz con números aleatorios de ese tamaño.

```
clear all;

tamano = input("Dame tamaño de matriz: ");

% Ejercicio 2.1
% Randi hace matrices con numeros aleatorios entre 0 y 100 y tamaño dado.
matriz = randi([0, 100], tamano);
```

Tras esto, creamos otra matriz con las columnas impares de la matriz inicial, y cogemos su diagonal:

```
% Ejercicio 2.2
% a)
disp(matriz)

% b) Para construir matrices se puede poner matriz(filas, columnas)
% adem◆s, podemos usar inicio:salto:end pa coger filas/columnas de salto en salto
matriz_columnas_impares = matriz(:, 1:3:end);

% c)
diagonal = diag(matriz);
```

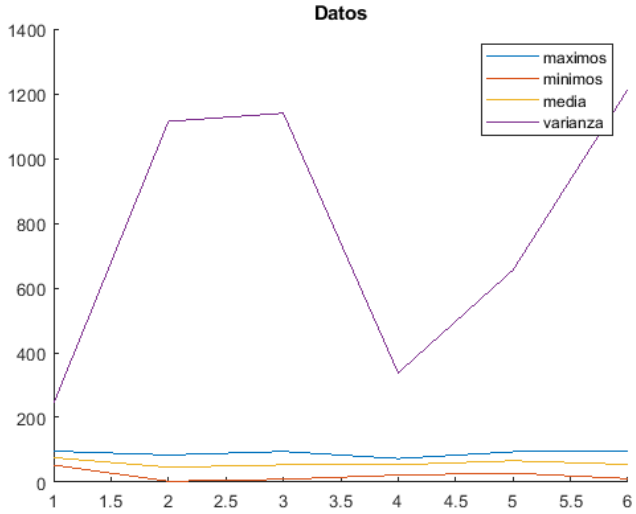
Finalmente, sacamos ciertas estadísticas de la matriz pedida:

```
% d)
maximos_fila = max(matriz);
minimos_fila = min(matriz);
medio_fila = mean(matriz);
varianza_fila = var(matriz);
```

Y los representamos gráficamente:

```
figure(1)
title("Datos")
hold on
plot(maximos_fila)
plot(minimos_fila)
plot(medio_fila)
plot(varianza_fila)
legend('maximos', 'minimos', 'media', 'varianza')

hold off

maximo = max(max(matriz)); % max(vector de maximos de cada columna)
minimo = min(min(matriz));
medio = mean(mean(matriz));
varianza = var(matriz);
```

Ejecutando el código para una matriz de 6x6, genera la siguiente gráfica:



EJERCICIO 3

El ejercicio 3 propone programar un script para realizar operaciones con dos matrices. En primer lugar, solicitamos dimensiones de matrices y crea matrices de 0s para esas dimensiones.

Tras esto, se llama a la función introducirMatriz, que solicita si se quiere rellenar aleatoriamente o a mano. Si se desea rellenar a mano, se ejecuta el bucle que va rellenando posición por posición con el input del usuario. Si es aleatorio, se rellena como en el ejercicio anterior:

```
clear all;
% Pido matriz A

dim_A = input("Dame dimensiones para matriz A: ")
entrada_A = input("¿Quieres que la matriz A sea aleatoria? (S/N): ", 's')

if entrada_A == "S"
    aleatorio = true;
else
    aleatorio = false;
end

if length(dim_A) == 1
    A = zeros(dim_A);
elseif length(dim_A) == 2
    disp("Matriz normal")
    A = zeros(dim_A(1), dim_A(2));
else
    return
end
A = introducirMatriz(A, aleatorio);
% Para la matriz B es idéntico.
```

La función introducirMatriz:

```
function [matriz] = introducirMatriz(matriz, aleatorio)
    [filas, columnas] = size(matriz)
    disp("Filas: "+filas)
    disp("Columnas: "+columnas)

    for fila = 1:filas
        for columna = 1:columnas
            if aleatorio
                matriz(fila, columna) = randi([1, 100])
            else
                cadena = "Dame posici?n ["+fila + "]["+columna + "]: "
                disp(cadena)
                matriz(fila, columna) = input(' ');
            end
        end
    end
end
```

Después de la lectura, muestro los datos (para B) es igual:

```
% Muestro datos pedidos para A:

disp("Matriz A: ")
disp(A)
disp("Matriz A transpuesta: ")
disp(A')
disp("Matriz A inversa: ")
disp(inv(A))
disp("Determinante A: ")
disp(det(A))
disp("Rango A: ")
disp(rank(A))
```

Y realizo productos y suma de columnas:

```
% Producto de A * B

C = A * B
disp("Matriz C = A * B")
disp(C)

disp("Matriz D resultado de multiplicar elemento a elemento. ")
D = times(A, B)

suma_columnas = A(:, 1) + B(:, 1)
disp("Primera columna concatenada: ")
disp(suma_columnas)
```

EJERCICIO 4

Este ejercicio propone estudiar el tiempo de procesamiento de operaciones sobre matrices de su tamaño. Se realizarán las operaciones de cálculo del rango y del determinante.

Creamos dos arrays que almacenarán el tiempo de cada operación y matriz:

```
array_tiempo_rango = zeros(1, 25)
array_tiempo_det = zeros(1, 25)
```

Creamos cada matriz, la rellenamos con valores aleatorios de 0 a 100. Utilizamos "tic" para establecer un punto de inicio, y "toc" para un punto de finalización. La diferencia en tiempo entre estos dos puntos queda almacenada en el array.

```
for tamano = 1:25
    matriz = randi([0, 100], tamano)

    disp("Tiempo que tarda en calcular el rango:")
    tic
    rango = rank(matriz);
    array_tiempo_rango(tamano) = toc;

    disp("Tiempo que tarda en calcular el determinante:")
    tic
    determinante = det(matriz);
    array_tiempo_det(tamano) = toc;

end
```

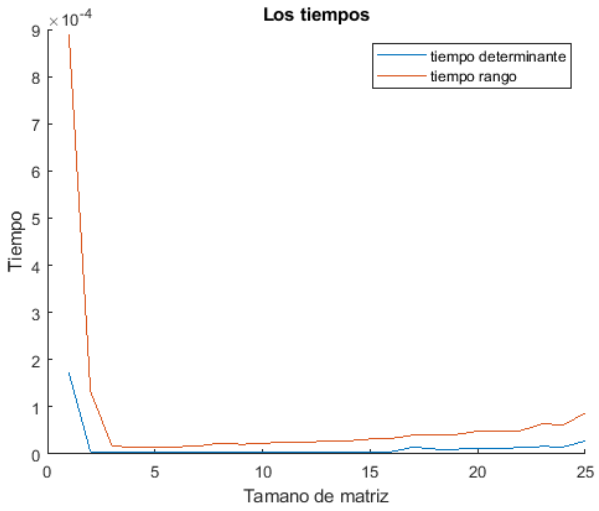
Tras esto, imprimimos por consola los arrays:

```
disp("Tiempo rango")
disp(array_tiempo_rango)

disp("Tiempo det")
disp(array_tiempo_det)
```

Y creamos la gráfica:

```
figure(1)
title("Los tiempos")
hold on

xlabel("Tamano de matriz")
ylabel("Tiempo")
plot(array_tiempo_det)
plot(array_tiempo_rango)
legend('tiempo determinante', 'tiempo rango')

hold off
```

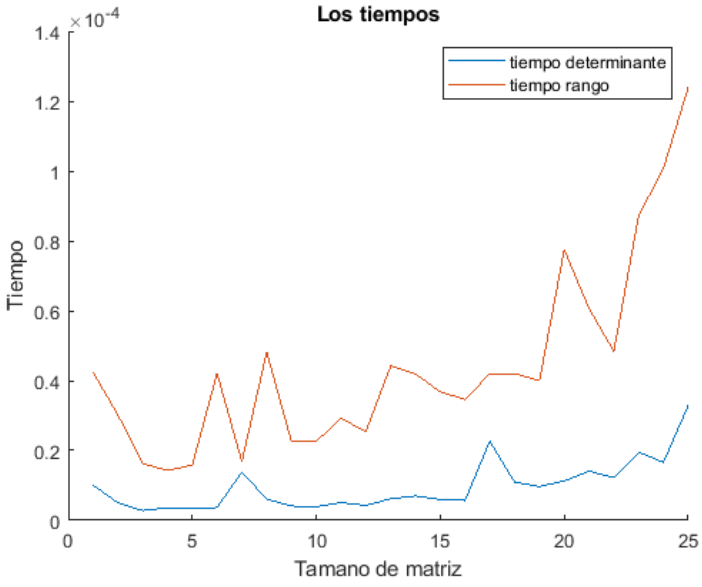
La gráfica generada es la siguiente:



Observamos que el tiempo empleado asciende conforme la matriz se hace más grande, lo cual tiene sentido, pues las operaciones se realizan con mayor cantidad de elementos y es más costosa.

Podríamos pensar que no tiene sentido que al principio le cueste más tiempo, cuando precisamente las matrices son más pequeñas. Sospechamos que esto es debido a que, como las operaciones coinciden con el inicio de ejecución del programa, puede ser que debido a ejecuciones y operaciones en segundo plano el cálculo del determinante y del rango pasen a un segundo plano y por tanto tomen más tiempo que el necesario para las matrices de mayor tamaño.

Esta salida no es uniforme. Ejecutando el mismo programa en momentos diferentes obtenemos gráficas distintas. Entendemos que, una vez más, es debido al kernel, operaciones en segundo plano tanto de Matlab como de Windows,etc.



EJERCICIO 5

En este ejercicio se pide dibujar una función de varias variables sobre una superficie.

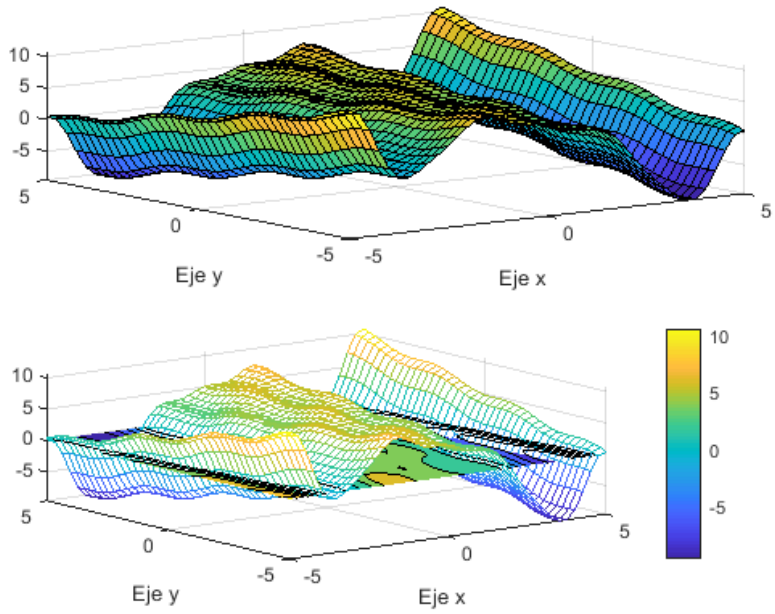
Para ello, primero cargo espacio y función:

```
% Coordenadas (x, y) que van desde -5 a 5 haciendo steps de 0.25
x = -5:0.25:5;
y = -5:0.25:5;

% Cargo grid
[X, Y] = meshgrid(x);

% Función
Z = Y.*sin(pi.*(X/10))+5.*cos((X.^2+y.^2)/8)+cos(X + Y).*cos(3.*X-Y);
```

Después, creo la gráfica de superficie, la superficie en forma de malla, y la gráfica de contorno.

```
% Dibujo la gráfica
figure(1);

% Parte superior y centrada, gráfica de la superficie
subplot(2,1,1);

% Título
title('Grafica de la superfice');
surf(X,Y,Z);

% Etiquetas
ylabel('Eje y');
xlabel('Eje x');

% Parte inferior, superficie de malla y gráfica de contorno
subplot(2,1,2);

% Título
title("Graficas forma de malla y contorno")
mesh(X,Y,Z);
hold on;
contourf(X,Y,Z);

% Etiquetas
ylabel('Eje y');
xlabel('Eje x');
colorbar
hold off;
```

La figura generada es la siguiente:



EJERCICIO 6

En este ejercicio se tienen dos sistemas de ecuaciones lineales a resolver. También se pide sacar la condición de la inversión y comparar resultados de la solución del sistema de ecuaciones con y sin ruido en los términos independientes.

Primero, cargamos las matrices en Matlab:

```
%Creacion de las matrices

% Primer sistema de ecuaciones
A1 = zeros(10,4);
A1(1,:)  = [0 2 10 7];
A1(2,:)  = [2 7 7 1];
A1(3,:)  = [1 9 0 5];
A1(4,:)  = [4 0 0 6];
A1(5,:)  = [2 8 4 1];
A1(6,:)  = [10 6 0 3];
A1(7,:)  = [2 6 4 0];
A1(8,:)  = [1 1 9 3];
A1(9,:)  = [6 4 8 2];
A1(10,:) = [0 3 0 9];

% Soluciones
b1 = [90 ; 59 ; 15 ; 10 ; 80 ; 17 ; 93 ; 51 ; 41 ; 76];

% Segundo sistema de ecuaciones
A2 = zeros(10,4);
A2(1,:)  = [0.110 0 1 0];
A2(2,:)  = [0 3.260 0 1];
A2(3,:)  = [0.425 0 1 0];
A2(4,:)  = [0 3.574 0 1];
A2(5,:)  = [0.739 0 1 0];
A2(6,:)  = [0 3.888 0 1];
A2(7,:)  = [1.054 0 1 0];
A2(8,:)  = [0 4.202 0 1];
A2(9,:)  = [1.368 0 1 0];
A2(10,:) = [0 4.516 0 1];

% Soluciones
b2 = [317 ; 237 ; 319 ; 239 ; 321 ; 241 ; 323 ; 243 ; 325 ; 245];
```

Calculo y muestro condiciones:

```
% a) condiciones de A respecto a la inversión
condicionesA1 = cond(A1);
cadenaA1 = sprintf("a) Condiciones sobre la matriz A1: %i", condicionesA1);
disp(cadenaA1);

% a2) condiciones de A2 respecto a la inversión
condicionesA2 = cond(A2);
cadenaA2 = sprintf("a) Condiciones sobre la matriz A2: %i", condicionesA2);
disp(cadenaA2);
```

Resuelvo matrices sin ruido:

```
% b) Resolver para obtener matriz x = [x1, x2, x3, x4]
% Resuelvo A1
linSolveA1 = linsolve(A1,b1);
disp("b) Solución para A1: ");
disp(linSolveA1);

% Resuelvo A2
linSolveA2 = linsolve(A2,b2);
disp("b) Solución para A2: ");
disp(linSolveA2);

% Guardo valores para apartado c)
pinvA1 = pinv(A1)*b1;
pinvA2 = pinv(A2)*b2;
```

Añado ruido y comparo:

```
% c) Añadir ruido sumando vector aleatorio y resolución de ecuaciones
% resultante.

% Extraigo filas y columnas
[nFilasA1, nColumnasA1] = size(b1);
[nFilasA2, nColumnasA2] = size(b2);

% Calculo ruido
ruidoB1     = rand(nFilasA1, nColumnasA1)*1+0;
ruidoB2     = rand(nFilasA2, nColumnasA2)*1+0;

% Lo añado a los vectores de términos independientes:
b1           = b1 + ruidoB1;
b2           = b2 + ruidoB2;

% Resuelvo A1 con linsolve
linsolveA1R   = linsolve(A1,b1);

% Resuelvo A2 con linsolve
linsolveA2R = linsolve(A2,b2);

% Resuelvo A1 y A2 con pinv
pinvA1R  = pinv(A1)*b1;
pinvA2R  = pinv(A2)*b2;
```

Muestro todo por pantalla:

```
disp("c) C�lculos con Pinv: ");
disp("A1: ");
disp(pinvA1);
disp("A1 (Ruido): ");
disp(pinvA1R);
disp("A2: ");
disp(pinvA2);
disp("A2 (Ruido): ");
disp(pinvA2R);
disp("--------------------------------")
disp("c) Cálculos con Linsolve: ");
disp("A1: ");
disp(linSolveA1);
disp("A1 (Ruido): ");
disp(linsolveA1R);
disp("A2: ");
disp(linSolveA2);
disp("A2 (Ruido): ");
disp(linsolveA2R);
```

EJERCICIO 7

Se pide un script que obtenga raíces de un producto de polinomios, pasando como argumento dichos polinomios. Debe obtener las raíces reales y complejas.

Declaramos la función y calculamos producto de polinomios:

```
% Funci□n para obtener raices de un producto de polinomios y las clasifique
% en reales y complejas.
function [solucion, nReales, nComplejas] = Ejercicio7(poli_1, poli_2)

% Inicializo variables
nReales = 0;
nComplejas = 0;

% Calculo el producto de polinomios
productoPolinomios = conv(poli_1, poli_2);
```

Doy a elegir al usuario sobre qué polinomio quiere calcular las raíces:

```
% Doy a elegir polinomio:
disp("p1 = Polinomio 1: ")
disp(poli_1);
disp("p2 = Polinomio 2: ");
disp(poli_2);
disp("pp = Polinomio producto: ");
disp(productoPolinomios);
eleccion = input('□Sobre qu□ polinomio se aplicar□ soluci□n?: ', 's');
```

Según la elección, se elige un polinomio u otro con el que trabajar:

```
% Establezco polinomio a solucionar
switch eleccion
    case 'p1'
        polinomio = poli_1;
    case 'p2'
        polinomio = poli_2;
    case 'pp'
        polinomio = productoPolinomios;
end
```

Calculamos raíces:

```
% Obtengo raices polinomio
solucion = roots(polinomio);

% Cuento soluciones reales y complejas
for i = 1:length(solucion)
    % Si es real a□ado a reales
    if isreal(solucion(i))
        nReales = nReales + 1;

    % Si es compleja, a□ado a complejas
    else
        nComplejas = nComplejas + 1;
    end
end
```

Y mostramos por pantalla, y finalizamos declaración de la función:

```
% Imprimo por pantalla
disp("N□ de  raices reales: " + nReales);
disp("N□ de  raices compleas: " + nComplejas);
end
```

PRÁCTICA 0 – PARTE 2

EJERCICIO 1

El ejercicio uno se compone de tres partes, para las cuales definiremos las siguientes variables simbólicas:

```
syms k a;
```

En la primera de ellas se nos pide obtener la transformada z de la función $f(k) = 2 + 5k + k^2$, y representar gráficamente las señales original y transformada.

Para ello, el primer paso es almacenar la función en una variable para poder trabajar con ella fácilmente y a partir de ahí obtener su transformada z.

```
func1 = 2 + 5*k + k^2;
ztrans_func1 = ztrans(func1, k);
```

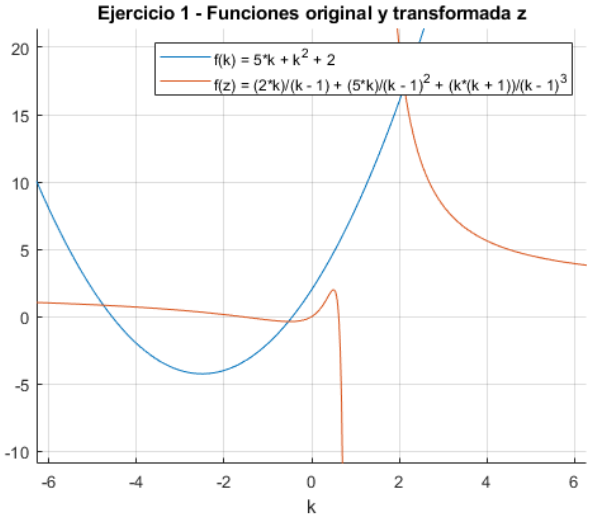
Una vez tenemos esto, creamos dos nuevas variables en las que almacenar la función y su transformada en arrays de caracteres para poder utilizarlos en la leyenda de la gráfica.

```
char_func1 = sprintf('f(k) = %s', func1);
char_ztrans_func1 = sprintf('f(z) = %s', ztrans_func1);
```

Abrimos una figura, pintamos la función y su transformada, añadimos un grid para poder visualizar la función con mayor claridad y añadimos el título y la leyenda.

```
figure(1);
hold on;
plot1 = ezplot(func1);
ztrans_plot1 = ezplot(ztrans_func1);
hold off;

grid on;
title('Ejercicio 1 - Funciones original y transformada z');
legend([plot1 ztrans_plot1], {char_func1, char_ztrans_func1});
```

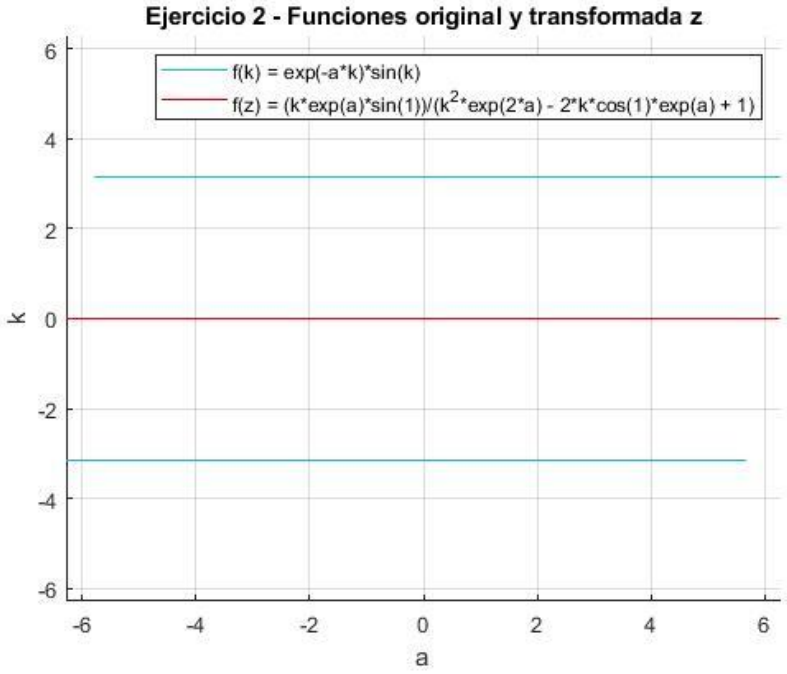
El resultado de este código es la siguiente gráfica:



La segunda parte de este ejercicio nos pide hacer exactamente lo mismo con la función $f(k) = \text{sen}(k) \cdot e^{(-ak)}$. Se presenta sólo el código con los comentarios, ya que la explicación es idéntica a la de la primera parte de este ejercicio.

```
% Almacenamos la función en una variable y calculamos su transformada Z
func2 = sin(k) * exp(-a*k);
ztrans_func2 = ztrans(func2, k);

% Metemos las variables en arrays de caracteres para poder usarlas en la
% leyenda de la gráfica
char_func2 = sprintf('f(k) = %s', func2);
char_ztrans_func2 = sprintf('f(z) = %s', ztrans_func2);

figure(2);
hold on;
plot2 = ezplot(func2);
ztrans_plot2 = ezplot(ztrans_func2);
set(ztrans_plot2, 'Color','r');
hold off;

grid on;
title('Ejercicio 2 - Funciones original y transformada z');
legend([plot2 ztrans_plot2], {char_func2, char_ztrans_func2});
```

La gráfica resultante para este código es la siguiente:



En la tercera parte del ejercicio se pide obtener y representar la respuesta de un sistema respecto al impulso y a una entrada escalón. El sistema viene definido por la siguiente función de transferencia discreta:

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

Para representar esta función de transferencia discreta, usamos la función tf del toolbox "Symbolic Math Toolbox" de MatLab. La función es llamada de la siguiente manera:

```
sys = tf([0.4 0 0], [1 -1 0.1 0.02], 0.1);
```

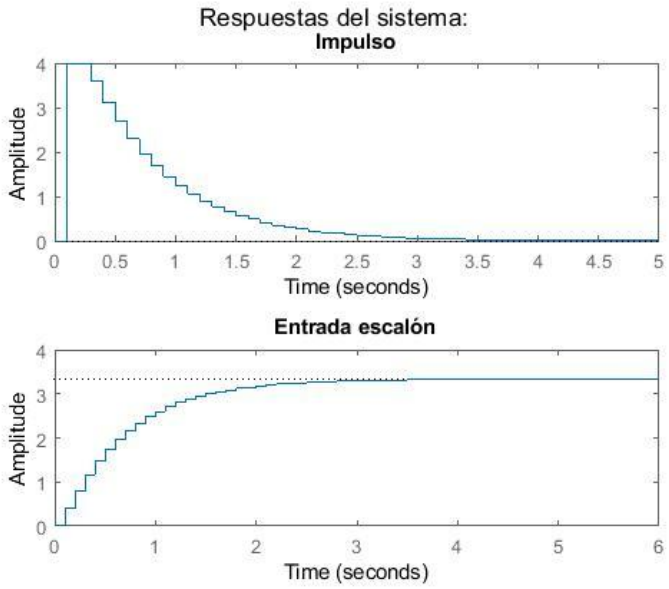
Si consultamos el contenido de la variable, podemos ver que la función se ha almacenado correctamente:

```
sys =

          0.4 s^2
  -----------------------
  s^3 - s^2 + 0.1 s + 0.02
```

Lo único que queda es pintar la gráfica. Para ello, usaremos la función subplot, aprovechando una sola figura para pintar ambas respuestas del sistema. El código es el siguiente:

```
figure(3);
sgtitle('Respuestas del sistema:');

hold on;
% Pintamos la respuesta al impulso en un subplot
subplot(2, 1, 1);
impulse(sys);
title('Impulso');

% Pintamos la respuesta a una entrada escalón en un subplot
subplot(2, 1, 2);
step(sys);
title('Entrada escalón');
hold off;
```

La gráfica generada por este código es la siguiente:



EJERCICIO 2

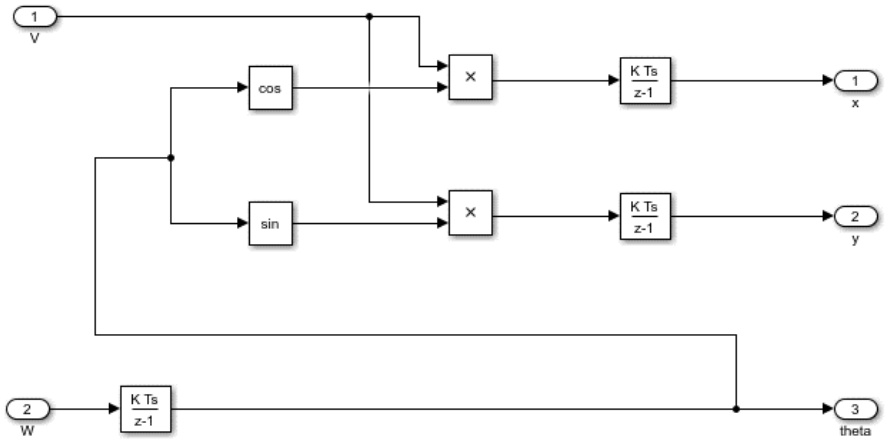
El ejercicio dos de esta segunda parte nos pide implementar un robot móvil cuyo movimiento se rige por las ecuaciones:

$$x_k = x_{k-1} + V_{k-1}T_s\cos(\theta_{k-1})$$

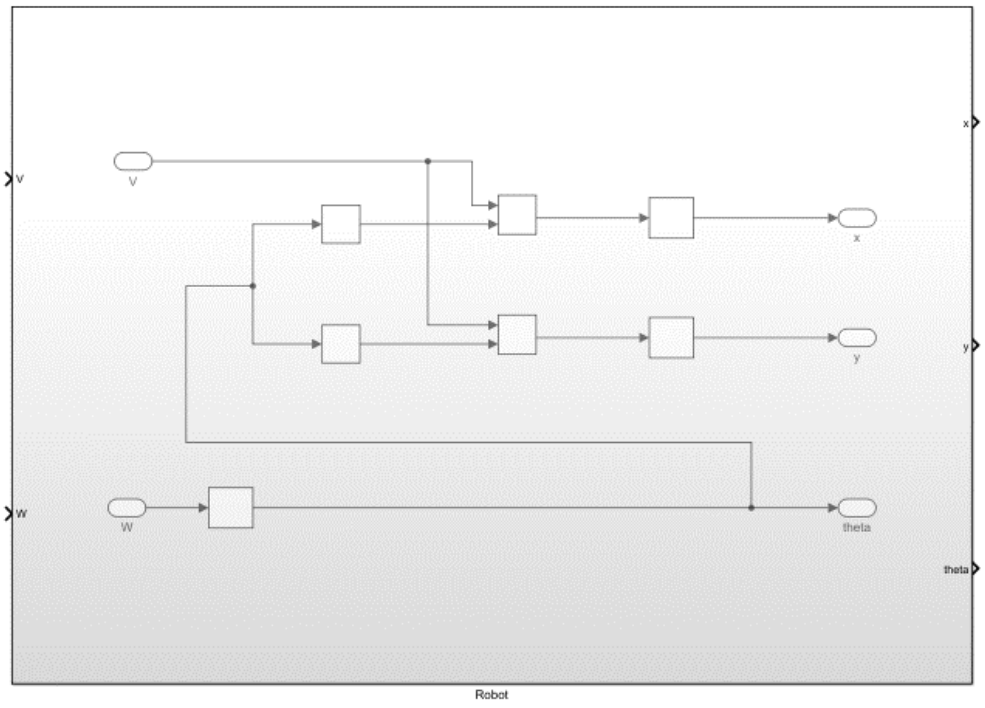
$$y_k = y_{k-1} + V_{k-1}T_s\sin(\theta_{k-1})$$

$$\theta_k = \theta_{k-1} + \omega_{k-1}T_s$$

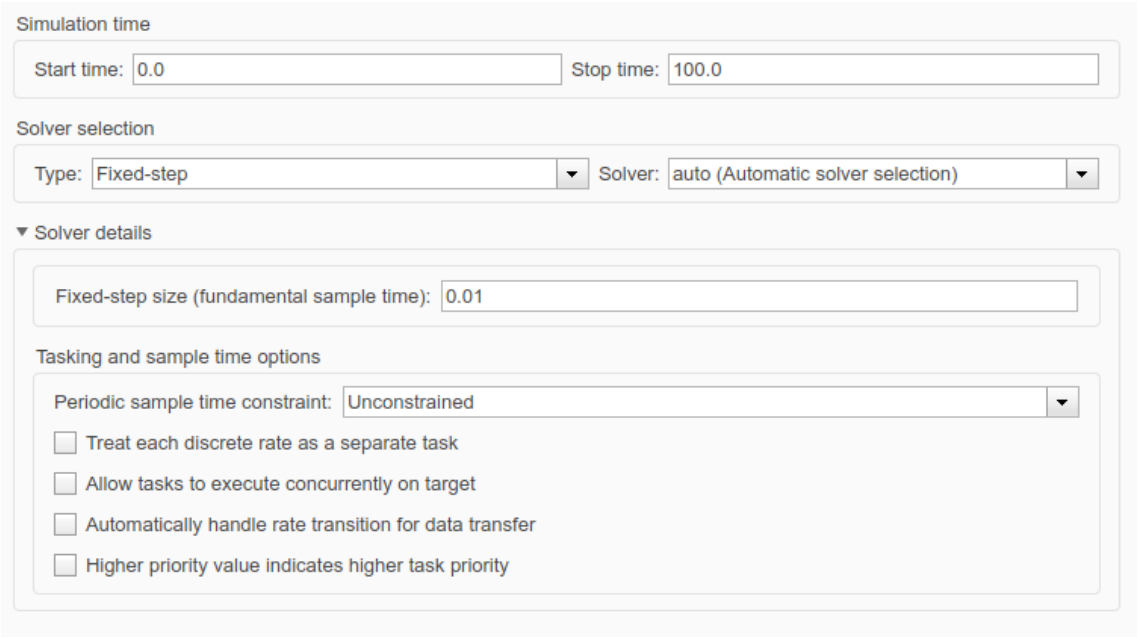
Vamos a añadir un bloque de subsistema en el que construiremos el robot. Este se representa con el siguiente esquema:

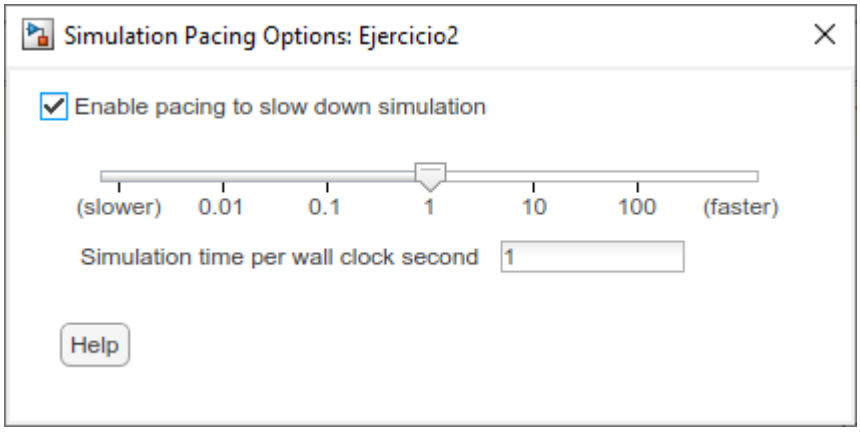


El robot ha sido construido siguiendo el enunciado y usando bloques de la librería estándar de simulink. El resultado del subsistema es el siguiente:

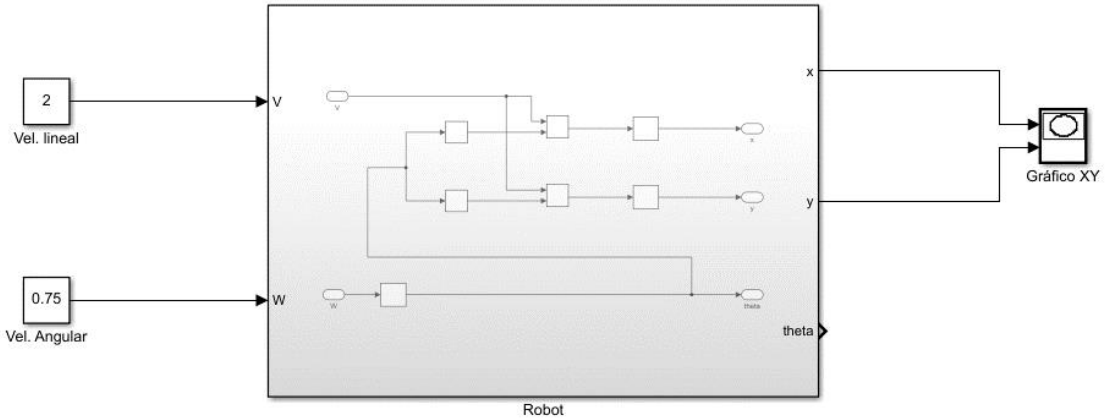


Se configura la simulación como indica el enunciado:

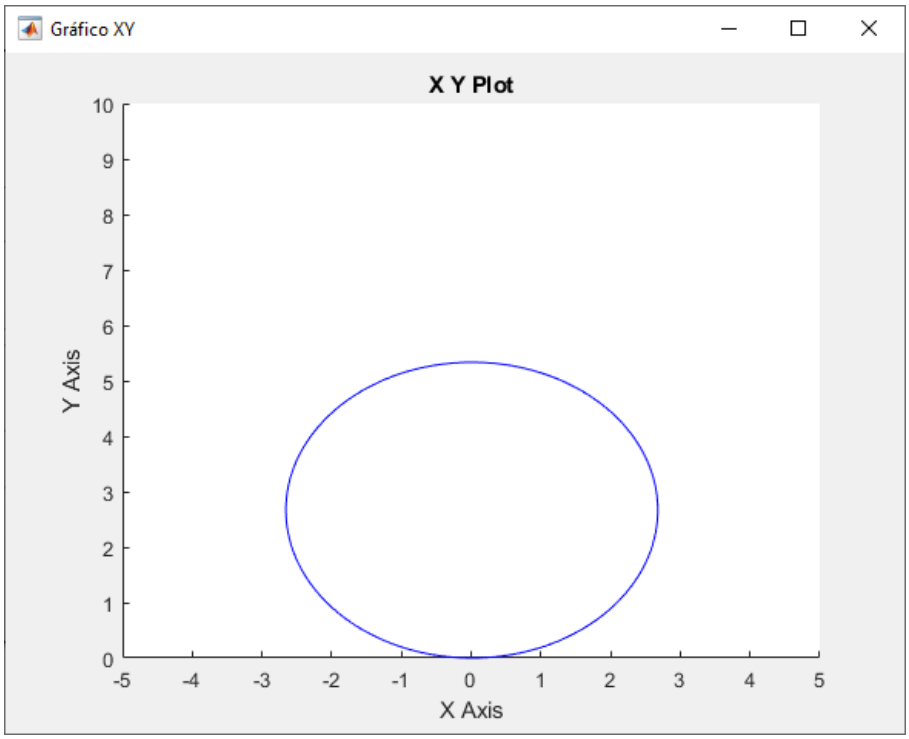




Y se añaden las constantes de velocidad angular y lineal, así como la salida a un gráfico XY. El circuito resultante es el siguiente:

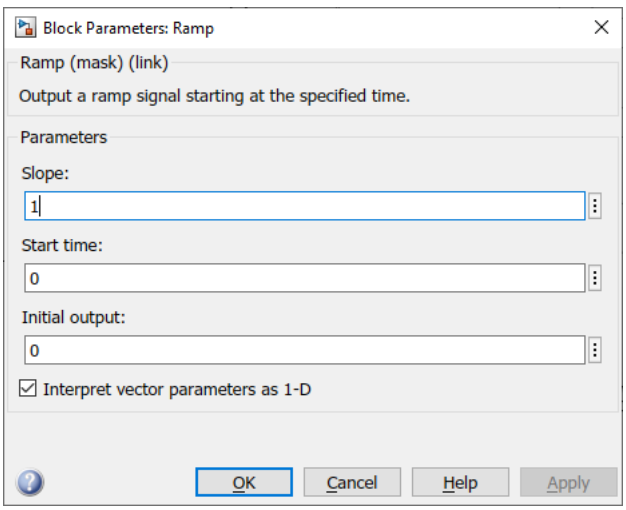


Para estos valores de velocidad, el movimiento del robot viene determinado por la siguiente gráfica:

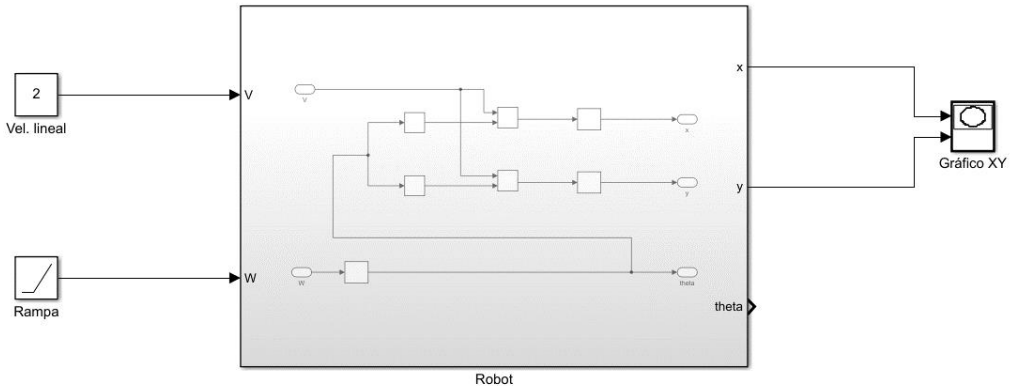


El gráfico tiene sentido, el movimiento no es totalmente circular ya que hay más velocidad lineal que angular. Si se dejan los 100 segundos de simulación, se ve que el movimiento del robot siempre sigue esa elipse y no se mueve de ella.

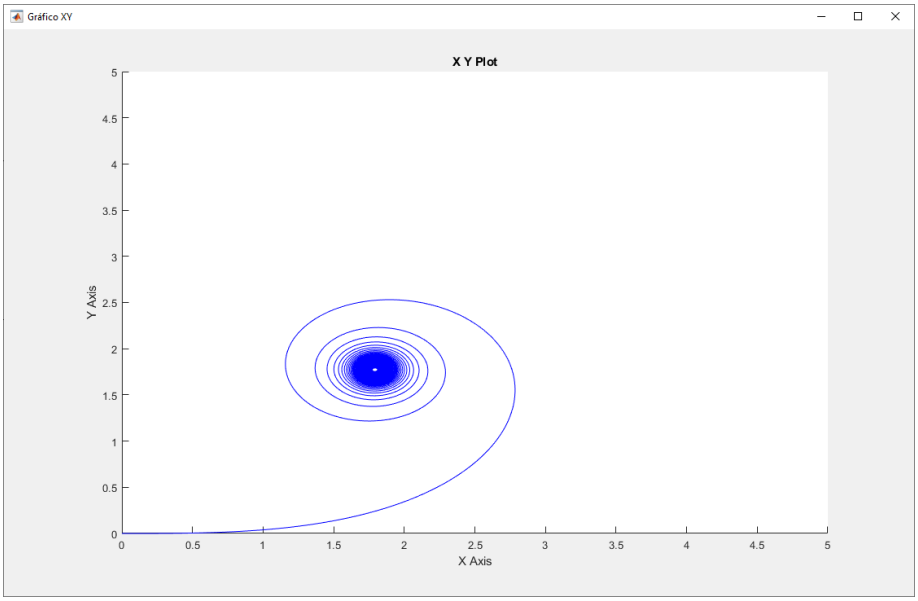
Si ahora probamos a cambiar los valores de la velocidad angular de entrada a velocidades Rampa no constante con la siguiente configuración:



Nos queda el siguiente diagrama:

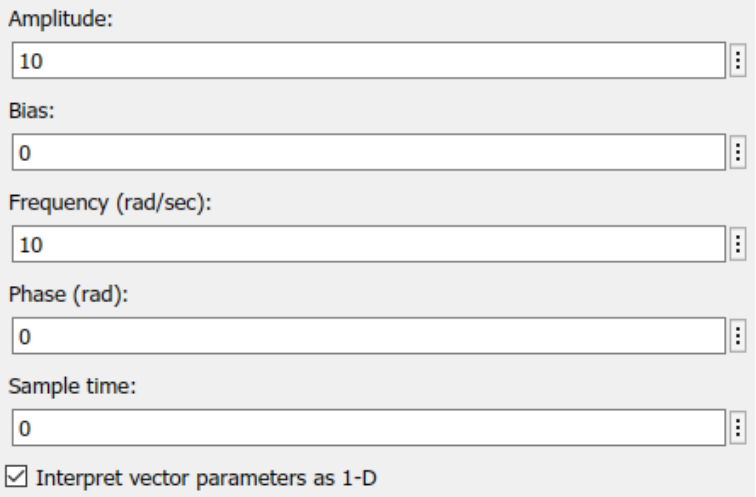


Al ejecutar la simulación, el gráfico XY resultante después de los 100 segundos es el siguiente:

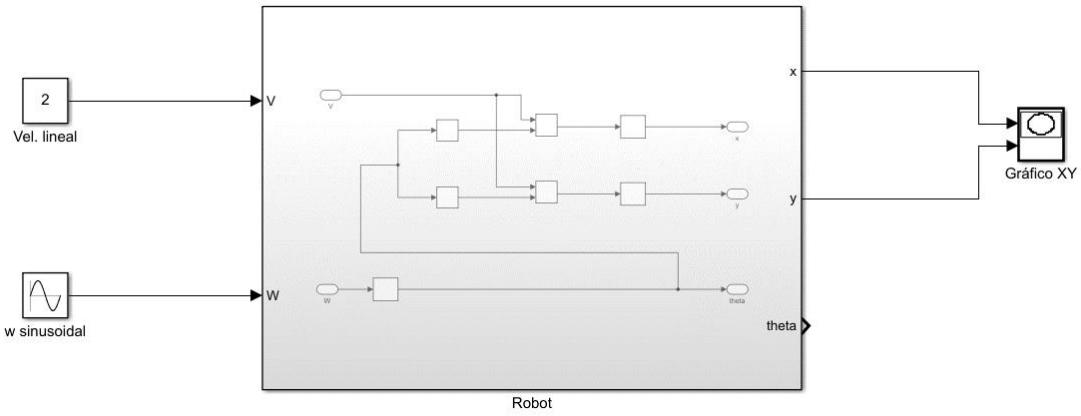


Lo cual se ajusta a lo esperado, ya que la velocidad angular no para de subir en ningún momento, por lo que podemos ver que cada vez el robot gira con más fuerza, haciendo que el radio de las elipses vaya disminuyendo según avanza la simulación.

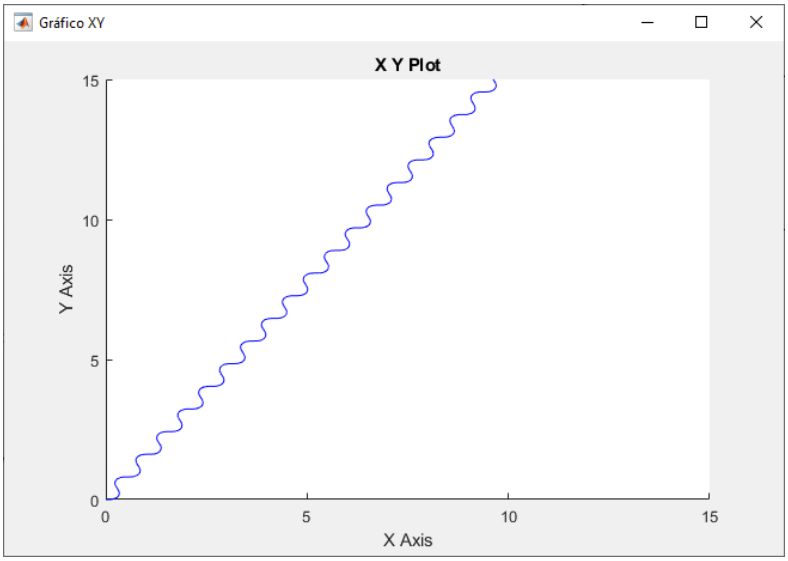
Si ahora utilizamos una señal sinusoidal con la siguiente configuración:



El esquema queda así:



Y el movimiento del robot se representa con la siguiente gráfica:



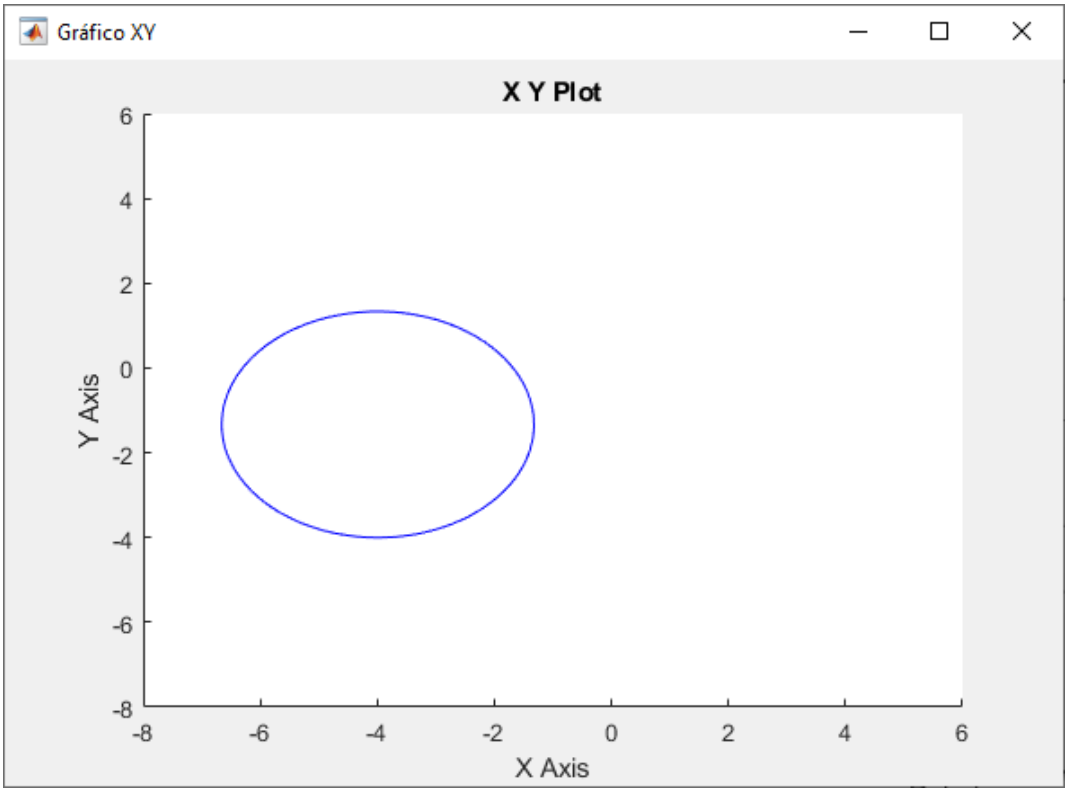
Lo cual también tiene sentido, ya que la velocidad angular va cambiando de sentido, haciendo que el robot avance en zigzagueando, pero moviéndose en una dirección constante, ya que la velocidad lineal es constante.

El último apartado nos pide modificar la posición inicial del robot a -4,-4. Para ello, abrimos los integradores en tiempo discreto que están conectados a las variables "x" e "y" y modificamos sus condiciones iniciales para que sean -4.

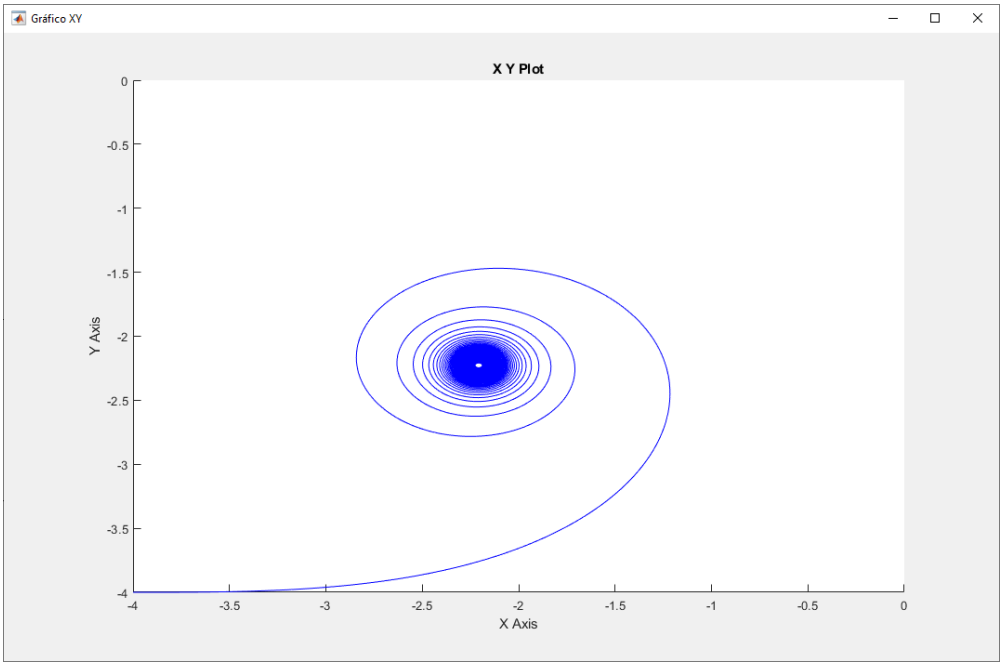
El que esta unido a theta lo dejamos tal y como está, es decir, en 0.

Las gráficas resultantes son estas:

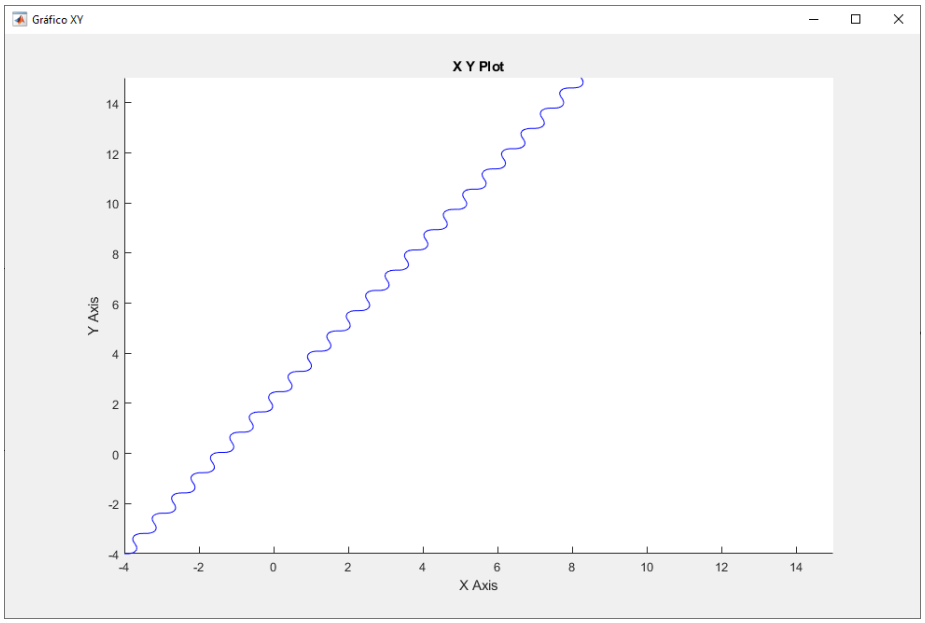
Valor constante de 0.75 de velocidad angular:



Rampa en velocidad angular:



Función sinusoidal en velocidad angular:



Las gráficas son exactamente iguales, solo varía la posición inicial.