PRACTICA O

Sistemas de control inteligente

Luis Alejandro Cabanillas Prudencio

Daniel López Moreno

Ingeniería informática - UAH

PARTE 1: Introducción a Matlab (I)

EJERCICIO 1: MATRICES Y VECTORES.

A) Cree la siguiente matriz A y el vector v:

En este apartado creamos una matriz de 4x2 llamada A y un vector columna de 4 filas llamado v.

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

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

Uniendo la matriz A con el vector v como si fuera una tercera columna conseguimos crear una matriz 4x3 llamada B.

```
B = [A v];
disp(B);
```

C) Obtenga y visualice un vector fila resultado de concatenar las filas de la matriz B.

Utilizando un bucle for desde 1 hasta 4 logramos concatenar en el vector fil cada una de las filas de la matriz B.

```
fil = [];
for i=1:4
    fil = [fil B(i,:)];
end
disp(fil);
```

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

Utilizando la simple instrucción ":" logramos crear un vector columna que concatene cada una de las columnas de la matriz B.

```
col = B(:);
disp(col);
```

EJERCICIO 2: MATRICES Y VECTORES.

A) 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".

Con la función "input" somos capaces de guardar en la variable size el tamaño de la matriz cuadrada que desea el usuario. A través de la función randi generamos una matriz con números entre [0-10] de tamaño sizeXsize y la guardamos en la variable Y.

```
%Apartado 1
    %Pedimos la entrada al usuario
    size=input("Indique el tamaño de la matriz: ");

    %Generamos la matriz
    Y=randi(10,size);
```

B) Matriz generada.

En este apartado mostramos nuestra matriz generada, para realizar una prueba generaremos una matriz de tamaño 3x3.

```
%a) Matriz generada
disp("Matriz generada");
disp(Y);
```

Salida:

```
Matriz generada
     9    10     3
    10     7     6
     2     1    10
```

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

Con un bucle for comprobando si el número de la columna es impar vamos añadiendo aquellas columnas que cumplan la condición a una nueva matriz llamada B.

```
%b) Matriz con columnas impares
B=[];
for i=1:size
    if rem(i,2) ~= 0
        B = [B Y(:,i)];
    end
end
disp("Matriz con columnas impares");
disp(B);
```

Como resultado podemos comprobar que de la matriz Y, generada en el apartado anterior, solo han quedado la columna 1 y 3.

```
Matriz con columnas impares
     9     3
    10     6
     2    10
```

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

A través de la función diag(Y) somos capaces de crear un vector c con los datos de la diagonal de nuestra matriz Y.

```
%c) Valor de la diagonal
c= diag(Y);
disp("Diagonal de la matriz");
disp(c);
```

Salida:

```
Diagonal de la matriz
     9
     7
    10
```

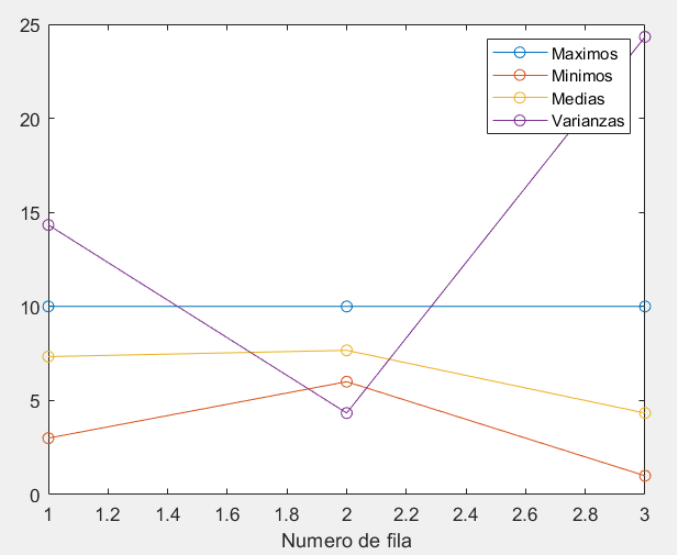
E) Valor máximo, mínimo, medio y varianza de cada fila. Estos valores se han de representar gráficamente, indicando en el eje de abscisas el número de fila

Utilizando las funciones max, min, mean y var logramos obtener los máximos, mínimos, medias y varianza de cada una de las filas de nuestra matriz Y.

```
%d) Valor maximo, minimo, medio y varianza
M = max(Y,[],2);
disp("Valor maximo de cada fila")
disp(M);
Z = min(Y,[],2);
disp("Valor minimo de cada fila")
disp(Z);
R = mean(Y,2);
disp("Valor medio de cada fila");
disp(R);
V = var(Y,0,2);
disp("Varianza de cada fila");
disp(V);
G = [M Z R V];
plot(G,'-o','MarkerIndices',1:1:size);
xlabel("Numero de fila");
legend('Maximos','Minimos','Medias','Varianzas');
```

Uniendo todos los vectores en una matriz G y utilizando la función plot(G) logramos mostrar nuestros valores en una gráfica que podremos personalizar utilizando leyendas o atributos de modificación.

```
Valor maximo de cada fila
    10
    10
    10

Valor minimo de cada fila
     3
     6
     1

Valor medio de cada fila
    7.3333
    7.6667
    4.3333

Varianza de cada fila
   14.3333
    4.3333
   24.3333
```



EJERCICIO 3: MATRICES Y VECTORES.

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

Lo primero que haremos en nuestro programa principal será pedir al usuario el tamaño de las dos matrices A y B y llamaremos a la función IntroducirMatriz ubicada en un archivo distinto para rellenar nuestras matrices.

```
%Pedimos la entrada al usuario
sz=input("Indique el tamaño de la matriz A en formato [fil col]: ");
sz2=input("Indique el tamaño de la matriz B en formato [fil col]: ");
%Creamos la matriz y la mostramos por pantalla
A = IntroducirMatriz(sz);
disp("La matriz A generada");
disp(A);
B= IntroducirMatriz(sz2);
disp("La matriz B generada");
disp(B);
[m,n] = size(A);
[o,p]= size(B);
```

B) 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.

Aquí podemos ver nuestra función IntroducirMatriz que recibe como paraemtros las dimensiones de nuestra matriz y devuelve una matriz llena.

Podemos ver una condición “if” que comprueba si el usuario desea completar la matriz de manera aleatoria o manualmente, y procedemos a llenar la matriz de salida.

```
%Funcion para rellenar una matriz dadas sus dimensiones
function Matriz = IntroducirMatriz(Dimensiones)
    fil = Dimensiones(1,1);
    if length(Dimensiones)==2
        col = Dimensiones(1,2);
    else
        col = fil;
    end

    inst= input('Introduce r para datos aleatorios o cualquier otra tecla para manual: ', 's');
    if inst=='r'
        Matriz= randi(10,fil, col);
    else
        Matriz=[];
        for i=1:col
            Col= [];
            for j=1:fil
                texto= strcat('Posicion fila[', num2str(j),'] col[',num2str(i),']: ');
                num= input(texto);
                Col = [Col ; num];
            end
            Matriz= [Matriz Col];
        end
    end
end
```

C) Las matrices generadas A y B

De vuelta a nuestro programa principal podemos ver las dos matrices A y B que hemos generado con nuestra función. Para realizar la prueba, hemos generado dos matrices A y B de tamaño 3x3 de manera aleatoria.

```
Introduce r para datos aleatorios o cualquier otra tecla para manual: r
La matriz A generada
    10    10     2
     2     5     5
    10     9    10

Introduce r para datos aleatorios o cualquier otra tecla para manual: r
La matriz B generada
     8     1     7
    10     9     8
     7    10     8
```

D) *La transpuesta e inversa de cada una de las matrices*

Para realizar la matriz transpuesta de A y B simplemente nos vale con utilizar la función transpose() sobre nuestras matrices.

Sin embargo, cuando queremos realizar la inversa, primero debemos comprobar si es posible realizarla. Si nuestra matriz es cuadrada y el determinante de nuestra matriz no es 0, podremos encontrar la inversa utilizando la función inv() sobre nuestra matriz.

En caso de que no se pueda realizar la inversa, mostraremos un mensaje al usuario.

```
%Calculamos la transpuesta y la inversa de cada matriz
Ta = transpose(A);
disp("La transpuesta de la matriz A es");
disp(Ta);
Tb = transpose(B);
disp("La transpuesta de la matriz B es");
disp(Tb);

%Calculamos la inversa si se puede
if (length(sz)==1||m==n) && (det(A)~=0)
    Ia=inv(A);
    disp("La inversa de la matriz A es");
    disp(Ia);
%Si no tiene inversa avisamos al usuario
else
    disp("La matriz A no tiene inversa")
end
%Calculamos la inversa si se puede
if (length(sz2)==1||o==p) && (det(B)~=0)
    Ib=inv(B);
    disp("La inversa de la matriz B es");
    disp(Ib);
%Si no tiene inversa avisamos al usuario
else
    disp("La matriz B no tiene inversa")
end
```

```
La transpuesta de la matriz A es
    10     2    10
    10     5     9
     2     5    10

La transpuesta de la matriz B es
     8    10     7
     1     9    10
     7     8     8

La inversa de la matriz A es
    0.0175   -0.2867    0.1399
    0.1049    0.2797   -0.1608
   -0.1119    0.0350    0.1049

La inversa de la matriz B es
   -0.0468    0.3626   -0.3216
   -0.1404    0.0877    0.0351
    0.2164   -0.4269    0.3626
```

E) *El valor del determinante y el rango de cada una de las matrices*

El determinante de una matriz se puede calcular utilizando la función det() sobre la matriz comprobando primero si la matriz es cuadrada. En caso de no serlo, avisaríamos al usuario de que no es posible realizar el determinante.

En cuanto al rango es una función que se puede realizar sobre cualquier matriz, por lo que lo único que debemos hacer es utilizar la función rank() sobre nuestras matrices.

```
%Determinante y rango de cada matriz
if m==n
    Da = det(A);
    disp("El determinante de A es");
    disp(Da);
%Si no tiene determinante avisamos al usuario
else
    disp("La matriz A no tiene determinante");
end

if o==p
    Db = det(B);
    disp("El determinante de B es");
    disp(Db);
%Si no tiene determinante avisamos al usuario
else
    disp("La matriz B no tiene determinante");
end

%Calculamos el rango
Ra = rank(A);
disp("El rango de A es");
disp(Ra);
Rb = rank(B);
disp("El rango de B es");
disp(Rb);
```

```
El determinante de A es
   286

El determinante de B es
  171.0000

El rango de A es
     3

El rango de B es
     3
```

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

Para realizar el producto elemento a elemento de 2 matrices en necesario comprobar si el tamaño de ambas es igual, para ello utilizaremos la función size().

En caso de ser de igual tamaño, podremos utilizar la función times() sobre nuestras 2 matrices para guardar en la variable Mule la multiplicación elemento a elemento de A y B.

En cuanto al producto matricial de AxB la única condición que se debe cumplir es que el número de columnas de A sea igual al número de filas de B. En tal caso simplemente deberíamos utilizar la función mtimes() sobre nuestras matrices.

```
%El producto de A y B (matricial y elemento a elemento)
%Producto elemento a elemento
if size(A)==size(B)
    Mule=times(A,B);
    disp("El producto elemento por elemento de A y B es");
    disp(Mule);
%Si no se puede multiplicar avisamos al usuario
else
    disp("Las matrices no pueden multiplicarse elemento por elemento");
end

%Producto matricial
if n==o
    Mul=mtimes(A,B);
    disp("El producto matricial de A y B es");
    disp(Mul);
%Si no se puede multiplicar avisamos al usuario
else
    disp("Las matrices no puden multiplicarse")
end
```

```
El producto elemento por elemento de A y B es
    80    10    14
    20    45    40
    70    90    80

El producto matricial de A y B es
   194   120   166
   101    97    94
   240   191   222
```

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

El primer paso que debemos realizar es guardar en 2 variables la primera fila de cada matriz. A continuación, simplemente concatenaremos ambas filas en la variable Vf.

```
%Vector fila concatenando la primera fila de cada matriz
FilA = A(1,:);
FilB = B(1,:);
Vf=[FilA FilB];
disp("El vector fila obtenido es");
disp(Vf);
```

Salida:

```
El vector fila obtenido es
    10    10     2     8     1     7
```

H) Un vector columna obtenido concatenando la primera columna de cada una de las matrices

Para generar un vector columna debemos hacer lo mismo que para generar un vector fila, pero en este caso guardaremos en nuestras variables la primera columna de cada matriz.

```
%Vector columna concatenando la primera columna de cada matriz
ColA = A(:,1);
ColB = B(:,1);
Vc=[ColA;ColB];
disp("El vector columna obtenido es");
disp(Vc);
```

Salida:

```
El vector columna obtenido es
    10
     2
    10
     8
    10
     7
```

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).

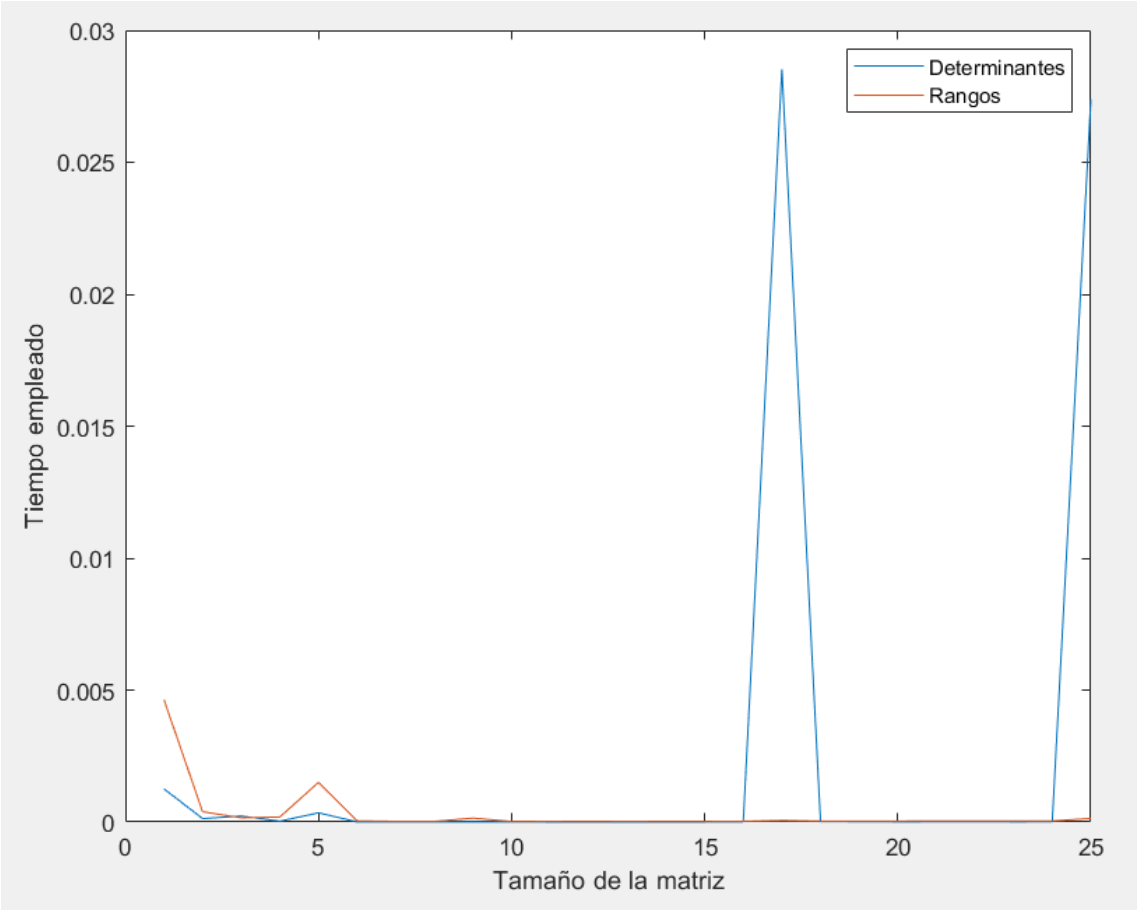
Con un bucle for desde 1 hasta 25 generaremos matrices cuadradas con valores entre [0-100]. A continuación, para cada matriz calcularemos el determinante y el rango utilizando las funciones det(M) y rank(M). Sin embargo, para calcular el tiempo de ejecución de las funciones

las situaremos entre dos funciones tic y toc. Con estas funciones podemos medir el tiempo que pasa entre un tic y un toc, por lo que podemos saber el tiempo de ejecución de nuestras funciones.

Dichos tiempos los guardaremos en dos vectores (tiemposDet y tiemposRank), para posteriormente representarlos de manera gráfica.

```
%Ejercicio 4
clear all;
tiemposDet=[];
tiemposRank=[];
for i=1:25
    M=randi([0,100],i);

    %Calculamos el tiempo en hacer determinante
    tic;
    det(M);
    tiemposDet(i)= toc;

    %Calculamos el tiempo en hacer el rango
    tic;
    rank(M);
    tiemposRank(i)=toc;
end

G=[tiemposDet ; tiemposRank];
plot(transpose(G));
xlabel("Tamaño de la matriz");
ylabel("Tiempo empleado");
legend("Determinantes","Rangos");
```

Salida:



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:

$$z = y * sin(pi * x/10) + 5 * cos((x2 + y2)/8) + cos(x + y)cos(3x - y).$$

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.

Para calcular los valores de z en función de x,y debemos asignar valores a x,y entre [-5,5]. A continuación, generaremos una malla utilizando meshgrid(x) y crearemos la matriz z utilizando la función con los valores de las matrices X e Y.

```
%Ejercicio 5
x = -5:0.25:5;
y = x;
[X,Y] = meshgrid(x);
Z=Y.*sin(pi.*X/10)+5.*cos((X.*X+Y.*Y)/8)+cos(X+Y).*cos(3.*X-Y);
```

Para mostrar una gráfica de la superficie utilizaremos la función surf sobre nuestras 3 variables X, Y y Z. Mediante la utilización de la función subplot(2,2,[1 2]) logramos situar la gráfica en la parte superior centrada, ya que las posiciones [1 2] hacen referencia a la parte superior.

```
%Arriba centrada (grafica superficie)
subplot(2,2,[1 2]);
surf(X,Y,Z);
title("Superficie");
xlabel("Coordenada x");
ylabel("Coordenada y");
zlabel("Coordenada z");
```

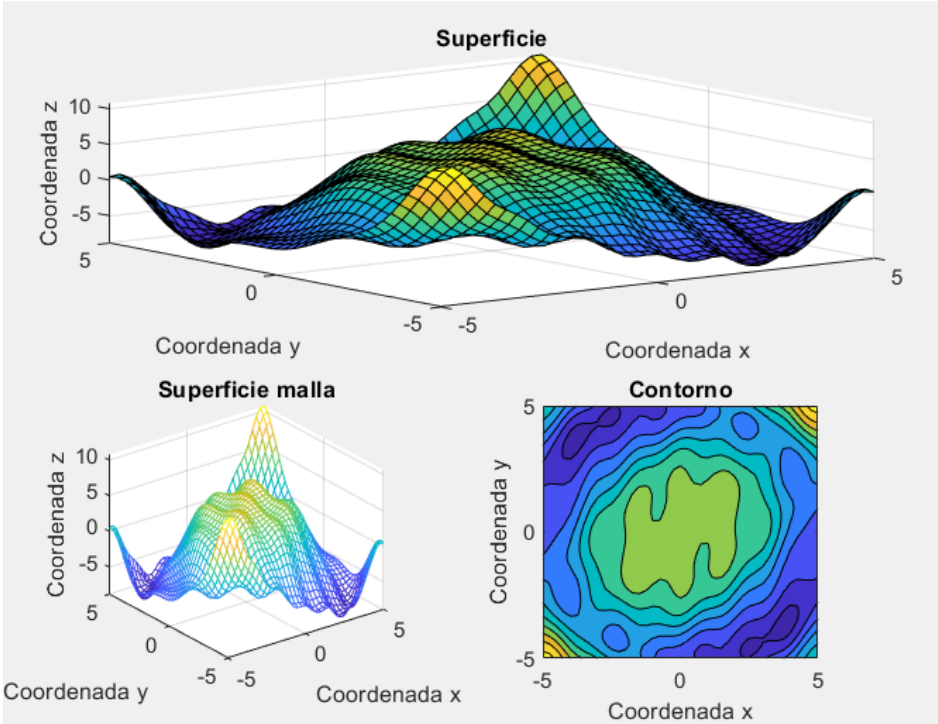
Para representar la gráfica en malla debemos utilizar la función mesh() sobre nuestras 3 variables X, Y y Z. En este caso queremos situar la gráfica en la posición inferior izquierda, por lo que utilizaremos subplot(2,2,3), ya que el 3 hace referencia a la parte inferior izquierda.

```
%Abajo izquierda (grafica en malla)
subplot(2,2,3);
mesh(X,Y,Z);
title("Superficie malla");
xlabel("Coordenada x");
ylabel("Coordenada y");
zlabel("Coordenada z");
```

Por último, para representar la gráfica del contorno debemos utilizar la función contourf() sobre nuestras 3 variables X, Y y Z. En este caso queremos situar la gráfica en la posición inferior derecha, por lo que utilizaremos subplot(2,2,4), ya que el 4 hace referencia a la parte inferior derecha.

```
%Abajo derecha (grafica contorno)
subplot(2,2,4);
contourf(X,Y,Z);
title("Contorno");
xlabel("Coordenada x");
ylabel("Coordenada y");
zlabel("Coordenada z");
```

Salida:



EJERCICIO 6: SISTEMAS LINEALES.

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

Introducimos tanto las matrices A1 y A2 como los vectores de resultado b1 y b2.

```
%Creacion de las matrices
A1= [0 2 10 7;
    2 7 7 1;
    1 9 0 5;
    4 0 0 6;
    2 8 4 1;
    10 6 0 3;
    2 6 4 0;
    1 1 9 3;
    6 4 8 2;
    0 3 0 9];
b1 = [90 ; 59 ; 15 ; 10 ; 80 ; 17 ; 93 ; 51 ; 41 ; 76];

%Segundo sistema
A2= [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];
b2 = [317 ; 237 ; 319 ; 239 ; 321 ; 241 ; 323 ; 243 ; 325 ; 245];
```

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

Para obtener el número de condición de la matriz A solo debemos emplear la función cond() sobre las matrices A1 y A2

```
%Apartado a
condA1 = cond(A1);
disp("El numero de condiciones sobre la matriz A1 es");
disp(condA1);
condA2 = cond(A2);
disp("El numero de condiciones sobre la matriz A2 es");
disp(condA2);
```

Salida:

```
El numero de condiciones sobre la matriz A1 es
    2.8413

El numero de condiciones sobre la matriz A2 es
   36.7102
```

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

Para resolver los sistemas de ecuaciones simplemente debemos utilizar la función linsolve() sobre las matrices A1 y A2 con sus respectivos vectores b1 y b2.

```
%Apartado b
SolA1 = linsolve(A1,b1);
SolA2 = linsolve(A2,b2);
```

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

Para añadir ruido a los vectores b1 y b2 les sumaremos unos vectores aleatorios b1_ruido y b2_ruido de media 0 y desviación 1. A continuación resolveremos los sistemas con los nuevos vectores b1 y b2.

```
%Apartado c
[r1, c1] = size(b1);
b1_ruido = rand(r1, c1)*1+0;
%Añadimos el ruido a b1
b1 = b1 + b1_ruido;
SolA1_r = linsolve(A1,b1);

[r2, c2] = size(b2);
b2_ruido = rand(r2, c2)*1+0;
%Añadimos el ruido a b2
b2 = b2 + b2_ruido;
SolA2_r = linsolve(A2,b2);
```

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

Por último, mostraremos por pantalla al usuario la solución de los distintos sistemas diferenciando de si contienen ruido o no.

```
%Apartado d
disp("Solución del primer sistema");
disp(SolA1);
disp("Solución del primer sistema con ruido");
disp(SolA1_r);
disp("Solución del segundo sistema");
disp(SolA2);
disp("Solución del segundo sistema con ruido");
disp(SolA2_r);
```

Salida:

```
Solución del primer sistema
   -2.6282
    4.9763
    5.3781
    3.5590

Solución del primer sistema con ruido
   -2.5906
    5.0063
    5.3921
    3.6150

Solución del segundo sistema
    6.3593
    6.3694
  316.2992
  216.2357

Solución del segundo sistema con ruido
    6.3950
    5.8950
  317.0217
  218.7445
```

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.

A) Recoge los arrays con los que se crean los polinomios.

Nuestro archivo en referencia al Ejercicio 7 simplemente contiene una petición de los polinomios a usar, una llamada a la función raíces() que se encuentra en otro archivo aparte raíces.m y muestra los resultados de dicha llamada.

```
%Ejercicio 7
poli_1= input("Introduce el vector para el polinomio 1: ");
poli_2= input("Introduce el vector para el polinomio 2: ");
x= raices(poli_1,poli_2);
disp(x);
```

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

La mayor parte de código se puede encontrar en el archivo raíces.m. En él declaramos la función raíces() con sus respectivos atributos de entrada y salida.

Para empezar, esta función pregunta al usuario sobre que polinomio quiere extraer las raíces, sobre p1, p2 o sobre el producto de ambos. En función de la elección, asigna a poli un polinomio.

```
function [solucion, reales, complejas]=raices(poli_1, poli_2)
    reales = 0;
    complejas = 0;

    %Seleccionamos el polinomio
    modo= input("Aplicar sobre p1, p2 o prod: ",'s');
    switch modo
        case 'p1'
            poli=poli_1;
        case 'p2'
            poli=poli_2;
        case 'prod'
            poli=conv(poli_1,poli_2);
    end
```

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

El siguiente paso que realizará nuestra función será extraer las raíces de nuestro polinomio utilizando la función roots(), la cual llena el vector solución con las raíces del polinomio.

Por último, clasificaremos cada raíz en 2 vectores distintos en función de si es real o compleja.

```
        %Extraemos las raices y las clasificamos
        solucion= roots(poli);
        for i = 1:length(solucion)
            if isreal(solucion(i))
                reales = reales + 1;
            else
                complejas = complejas + 1;
            end
        end

        fprintf('Cantidad de raices reales %i \n', reales);
        fprintf('Cantidad de raices complejas %i \n', complejas);

    end
```

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

Para comprobar el funcionamiento de nuestra función hemos ejecutado el programa con el producto de los 2 polinomios de prueba [1 2 2] y [1 3] y hemos encontrado como resultado 1 raíz real y 2 complejas.

```
Introduce el vector para el polinomio 1: [1 2 2]
Introduce el vector para el polinomio 2: [1 3]
Aplicar sobre p1, p2 o prod: prod
Cantidad de raices reales 1
Cantidad de raices complejas 2
  -3.0000 + 0.0000i
  -1.0000 + 1.0000i
  -1.0000 - 1.0000i
```

PARTE 1: Introducción a Matlab (II)

EJERCICIO 1: TRANSFORMADAS DE SEÑALES.

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

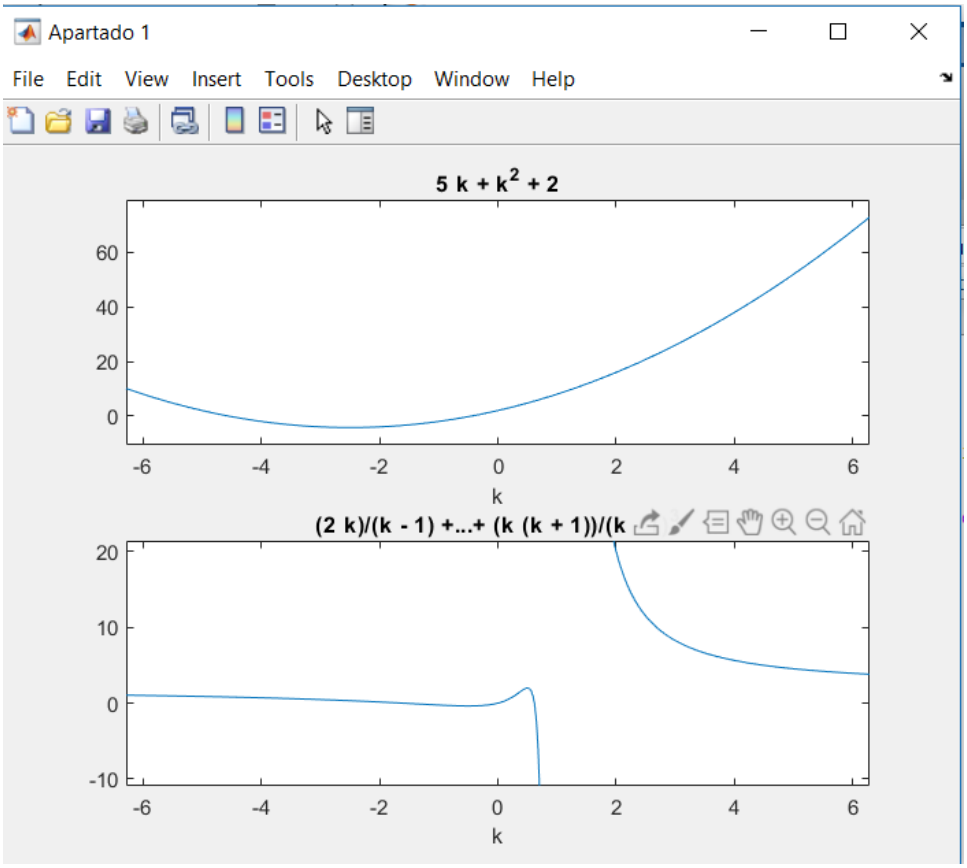
El ejercicio nos pide que representemos gráficamente las señales origen y transformada, así que lo primero que hemos hecho ha sido representar la función origen en Matlab mediante ezplot():

```
%Función origen y su representación
f1 = 2 + 5*k + k^2;
figure('Name','Apartado 1','NumberTitle','off');
subplot(2,2,[1 2]);
p1 = ezplot(f1);
```

A continuación, debemos obtener la transformada Z de la función origen y representarla. Para ello hemos hecho uso de ztrans(f1,k) que nos devuelve la transformada de la función f1. Con ezplot() representaremos la señal transformada gráficamente:

```
%Función transformada y su representación
f1_trans = ztrans(f1, k);
disp(f1_trans);
subplot(2,2,[3 4]);
p2 = ezplot(f1_trans);
```

El



resultado obtenido ha sido el siguiente:

B) Obtenga la transformada z de la siguiente función: f(k) = sen(k) * e^-ak. Represente gráficamente las señales original y transformada.

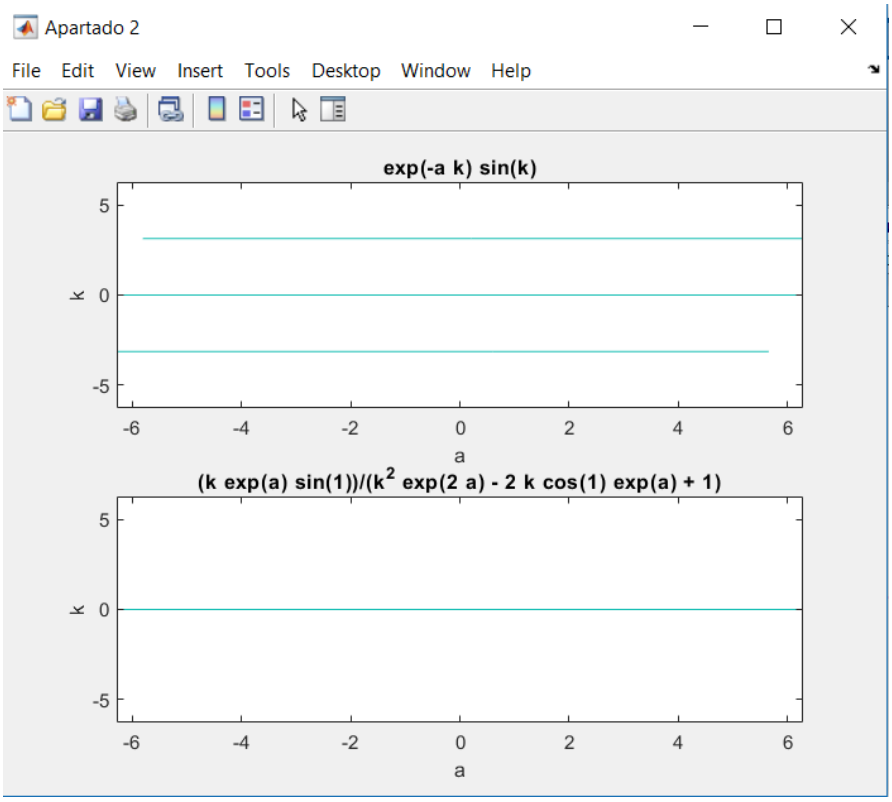
De la misma forma que en el apartado anterior, introduciremos la función origen en Matlab y la representaremos con ezplot():

```
% Apartado 2
%Función origen y su representación
f2 = sin(k)*exp(-a*k);
figure('Name','Apartado 2','NumberTitle','off');
subplot(2,2,[1 2]);
p3 = ezplot(f2);
```

Posteriormente, realizamos la transformada de la misma forma que en el apartado anterior y lo representamos gráficamente:

```
%Función transformada y su representación
f2_trans = ztrans(f2, k);
disp(f2_trans);
subplot(2,2,[3 4]);
p4 = ezplot(f2_trans);
```

El resultado obtenido ha sido el siguiente:



C) *Dada la función de transferencia discreta*

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

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

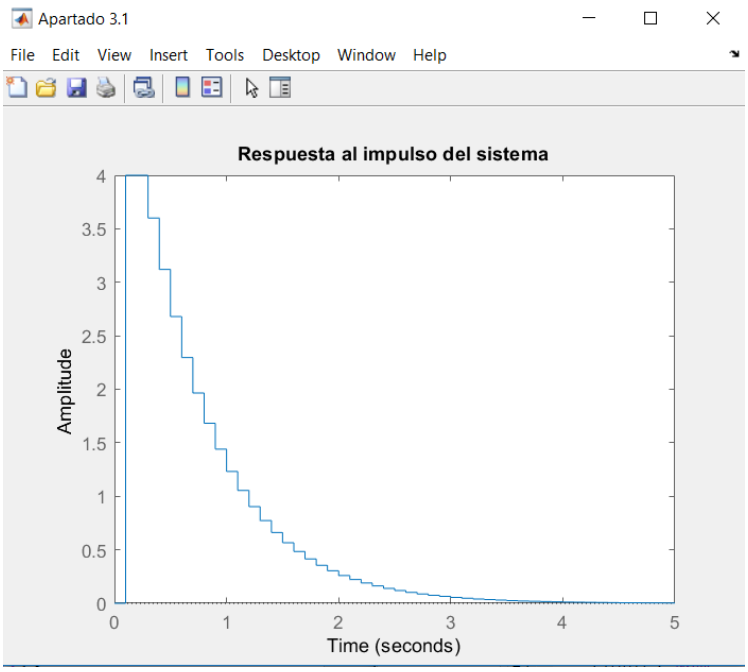
Lo primero que tenemos que hacer es crear dos variables, una para el numerador de la función y otra para el denominador. Las llamaremos num y den. En num y den introduciremos los coeficientes ordenados descendientemente. Una vez tenemos las variables, establecemos un tiempo de muestra de 0.1 segundos. Lo único que queda para tener la función de transferencia en nuestro script de Matlab es llamar a la función tf() con los parámetros anteriores. Esta función nos ayuda a construir funciones de transferencia en Matlab:

```
%Polinomio en forma vectorial
num = [0.4 0 0];
den = [1 -1 0.1 0.02];

%Tiempo de muestra
t=0.1;

%Transformada
fun = tf(num,den,t);
```

Para representar gráficamente la respuesta al impulso del sistema haremos uso de la función impulse(), y le pasaremos como parámetro la función de transferencia discreta:

```
%Representación gráfica del impulso
figure('Name','Apartado 3.1','NumberTitle','off');
impulse(fun);
title('Respuesta al impulso del sistema');
```

El resultado obtenido ha sido el siguiente:

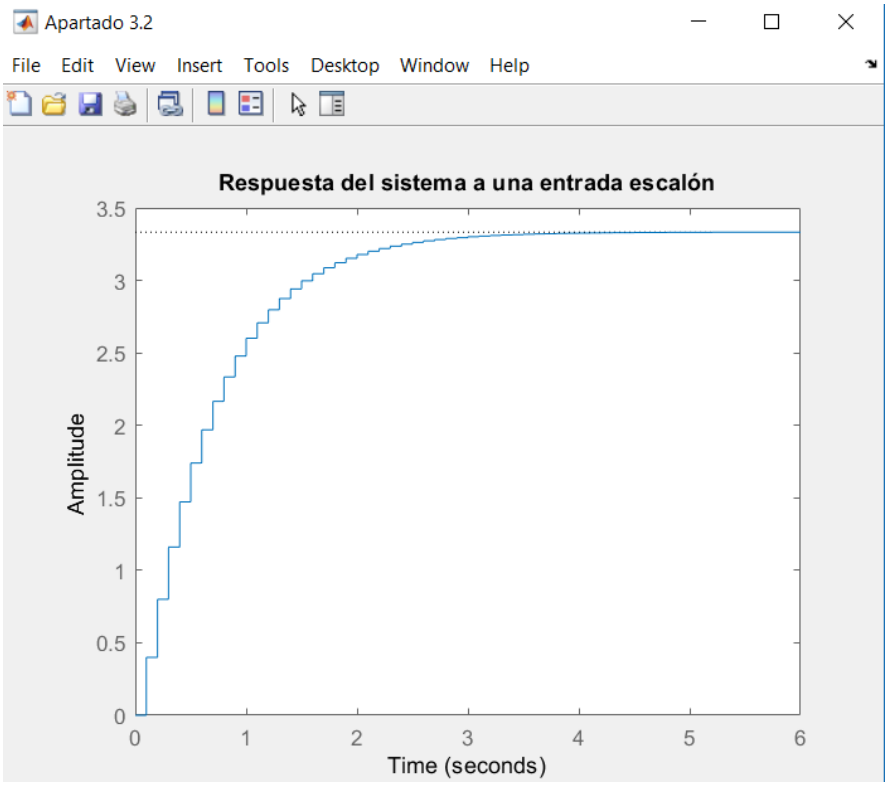


b. *Obtenga y represente la respuesta del sistema ante una entrada escalón.*

Como ya tenemos la función de transferencia hecha, lo único que tenemos que hacer para obtener la representación gráfica del sistema ante una entrada escalón es llamar a la función step() con nuestra función de transferencia discreta:

```
%Representación gráfica entrada escalón
figure('Name','Apartado 3.2','NumberTitle','off');
step(fun);
title('Respuesta del sistema a una entrada escalón');
```

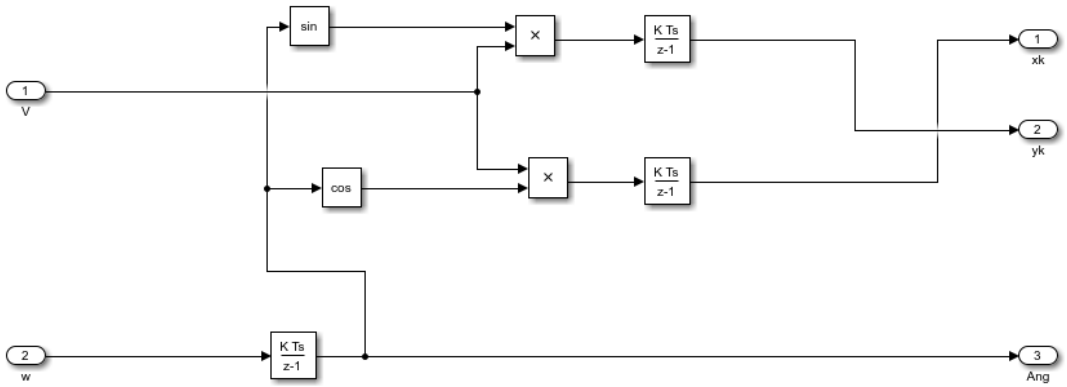
El resultado obtenido ha sido el siguiente:



EJERCICIO 2: MODELADO DEL COMPORTAMIENTO DE UN ROBOT MÓVIL EN SIMULINK.

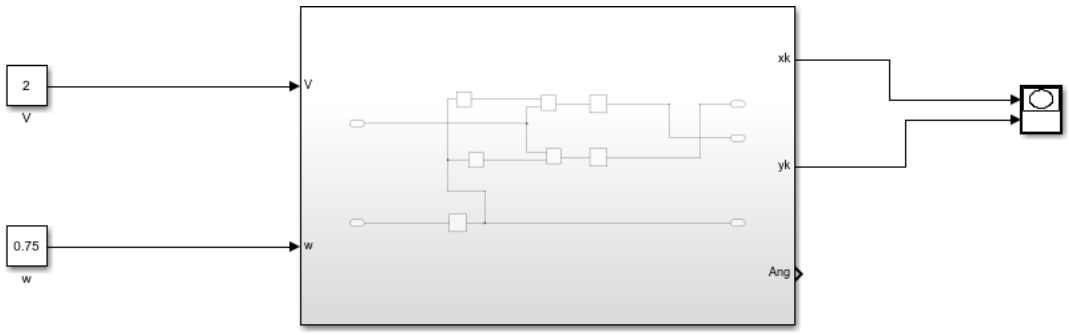
A) Implemente el modelo de la Figura 2 (todos los bloques utilizados pueden encontrarse en la librería estándar de Simulink).

Tras montar el modelo, el resultado ha sido el siguiente:

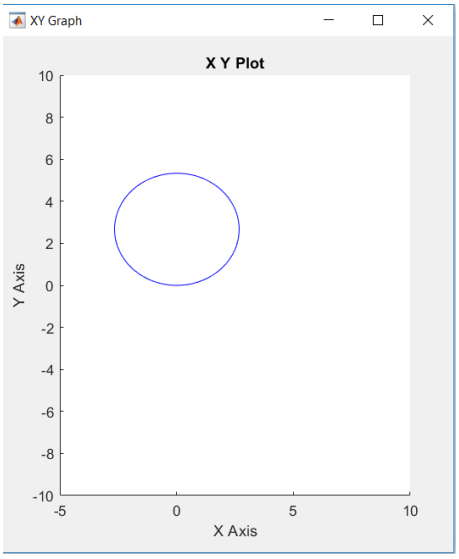


B) Una vez completado el apartado anterior, cree el subsistema mostrado en la Figura 1 y simule su funcionamiento con velocidad lineal y angular constante creando el sistema mostrado en la Figura 3. Configure los parámetros de la simulación (menú "Simulation/Model Simulation Parameters" y menú Simulation/Pacing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

Tras montar el subsistema, el resultado ha sido el siguiente:



Si lo ejecutamos, según las opciones especificadas, obtenemos la siguiente gráfica donde se detalla la posición del robot en cada instante:

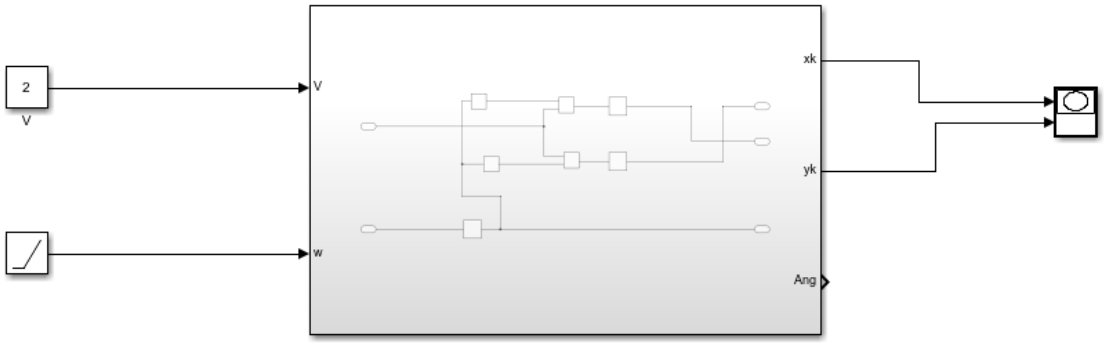


C) Visualizando la gráfica generada por el módulo XY Graph, compruebe que el funcionamiento del robot se ajusta a lo esperado.

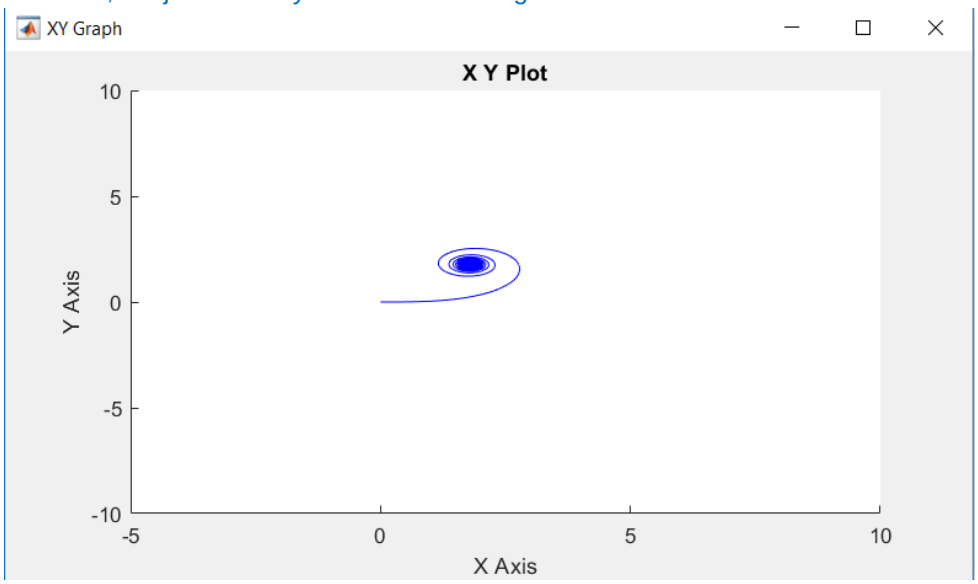
Como podemos ver en la gráfica del apartado, el robot sigue una trayectoria circular empezando en la posición (0,0) del plano. Esto es correcto, ya que en el subsistema habíamos introducido que el robot empezase en esa posición y, como la velocidad angular y lineal es constante, sigue la misma trayectoria circular durante todo el tiempo de muestra.

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

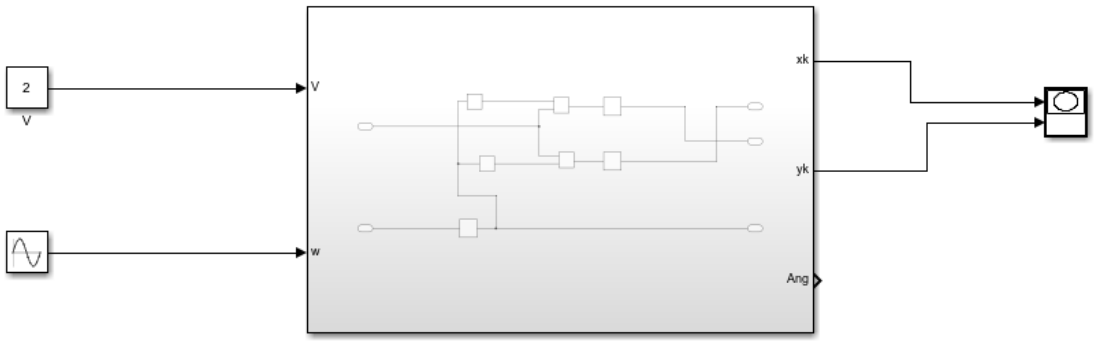
Lo primero que tenemos que hacer es introducir en el sistema la rampa:



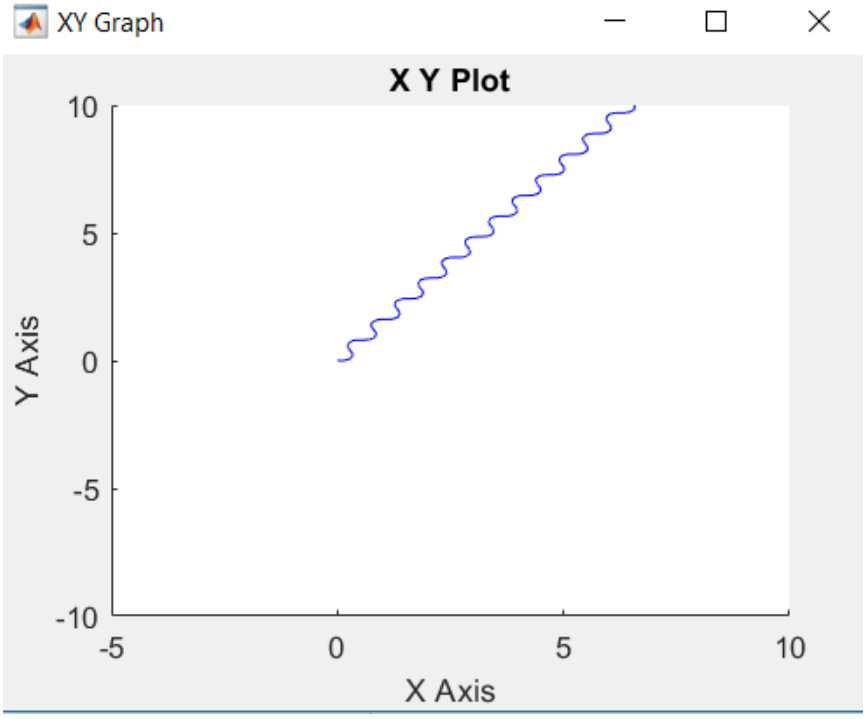
Tras ello, lo ejecutamos y observamos el siguiente resultado:



Para la w senoidal, deberemos proceder de igual manera y añadir al sistema la onda sinusoidal:



El resultado al simularlo ha sido el siguiente:



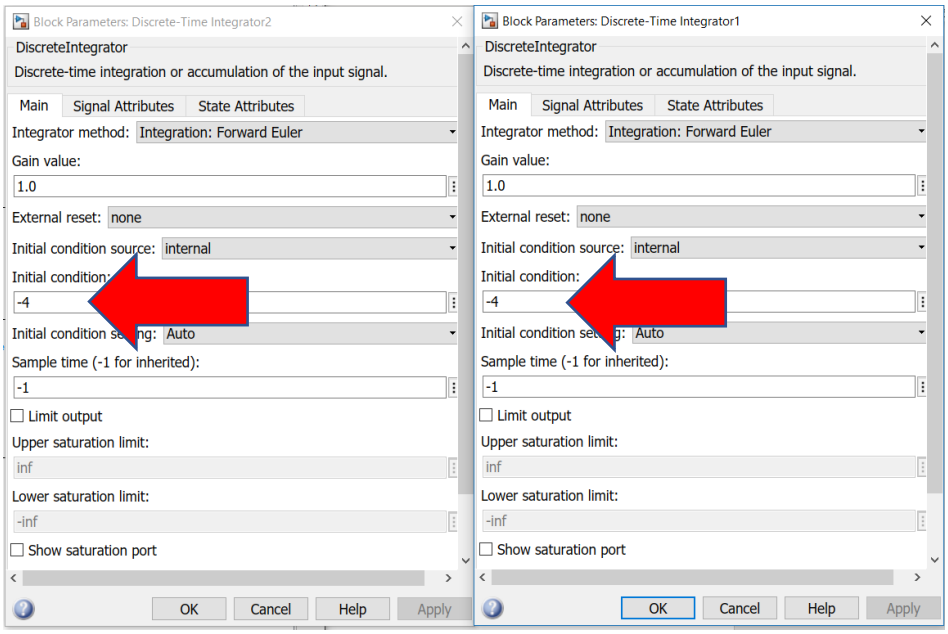
E) Estudie de nuevo las trayectorias realizadas por el robot y compruebe que se ajustan al comportamiento esperado.

En el caso de la rampa, los resultados se ajustan a lo esperado, ya que estamos modificando la velocidad angular y no la lineal, por lo que el robot sigue una trayectoria de una rampa curva con velocidad lineal constante hasta que el tiempo de muestra acaba.

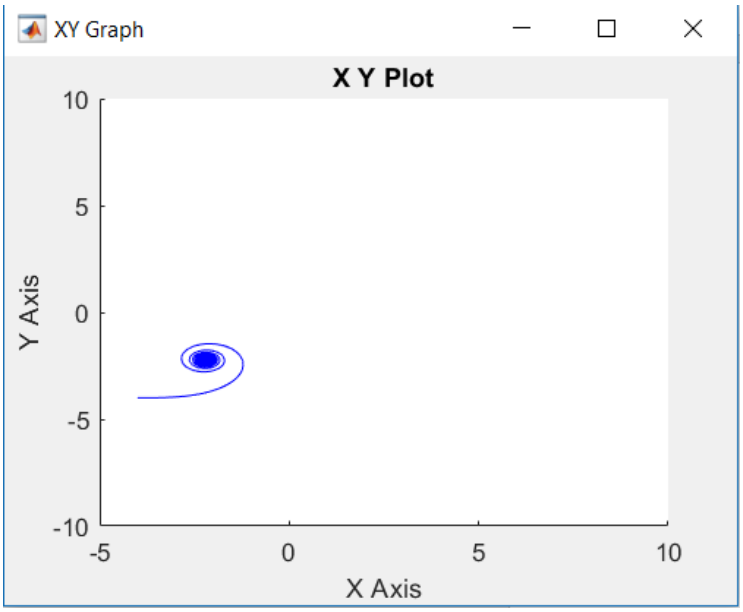
Por otro lado, con la onda sinusoidal los resultados también son los esperados, ya que el robot sigue esta trayectoria hasta que finaliza el tiempo de muestra.

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

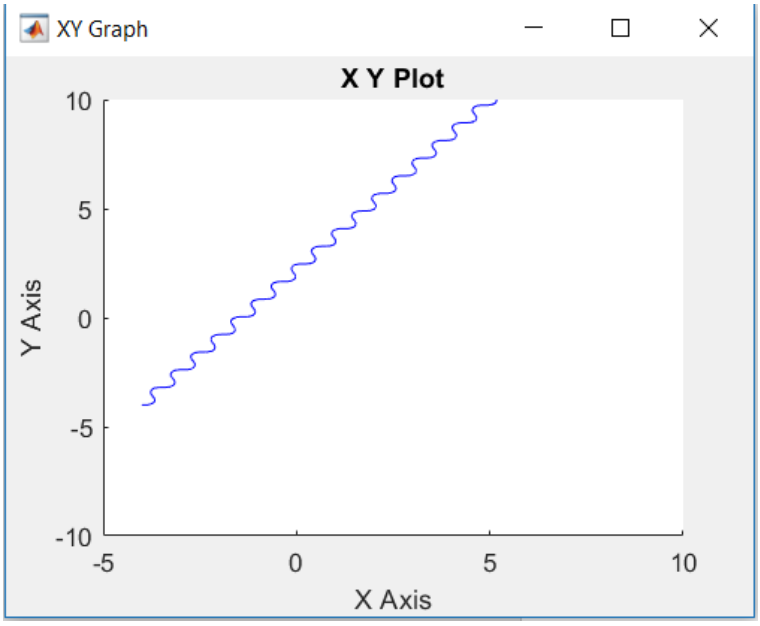
Para cambiar la posición de inicio del robot a (-4,-4) tenemos que acceder al subsistema y hacer doble clic sobre el *"Discrete Time Integrator"* de la x y de la y. En el cuadro de opciones que aparece tenemos que poner la *initial condition* a -4:



Para la función rampa hemos obtenido la siguiente gráfica:



Para la función sinusoidal hemos obtenido la siguiente gráfica:



Como podemos ver, el resultado es el mismo que en las simulaciones anteriores pero desplazado, ya que antes empezábamos en (0,0) y ahora empezamos en (-4,-4), por lo que nuestro robot empieza desde más atrás que antes.