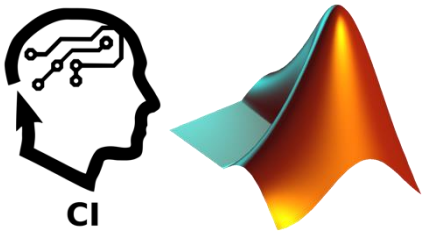
Sistemas de control Inteligente

Ejercicios de Introducción a Matlab

Juan José Córdoba Zamora
Juan Casado Ballesteros



Universidad de Alcalá

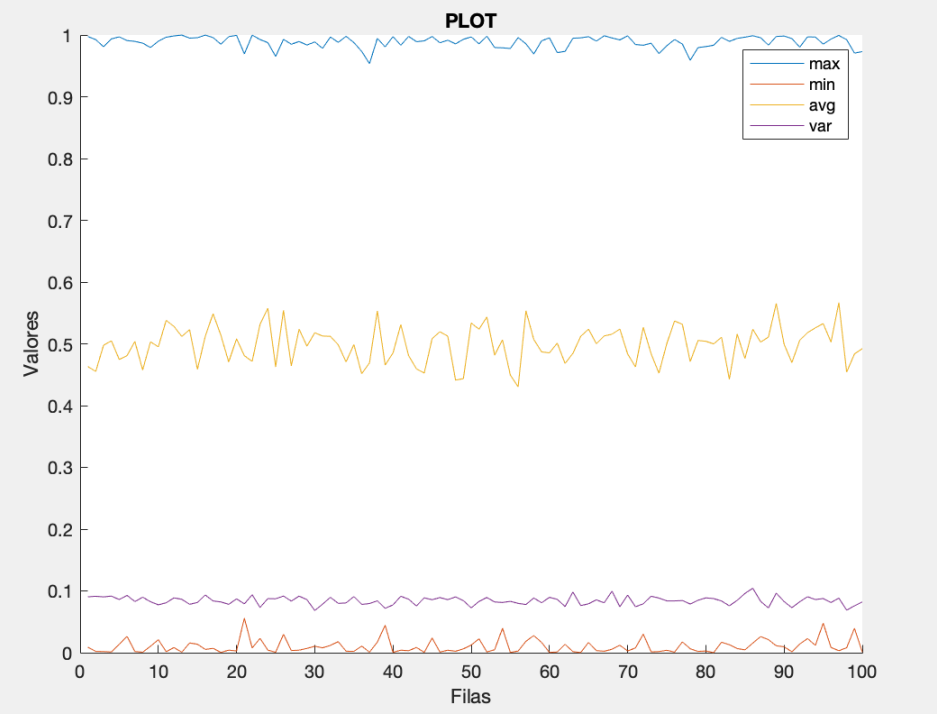
Ejercicio 1 Parte 1

Hemos podido ver lo fácil que es crear matrices nuevas con la sintaxis de Matlab. Escribiendo un espacio u una coma los números se ubicarán en una nueva fila, por el contrario, si escribimos un punto y coma estos se ubicarán en una nueva columna. También es muy sencillo concatenar matrices por filas o columnas utilizando la misma sintaxis siempre y cuando estas coincidan en tamaño en la dimensión por la que las estemos uniendo.

Para visualizarlas podremos hacerlo no poniendo punto y como al final de una asignación o utilizando la sentencia *display* lo que es más recomendable.

Ejercicio 2 Parte 1

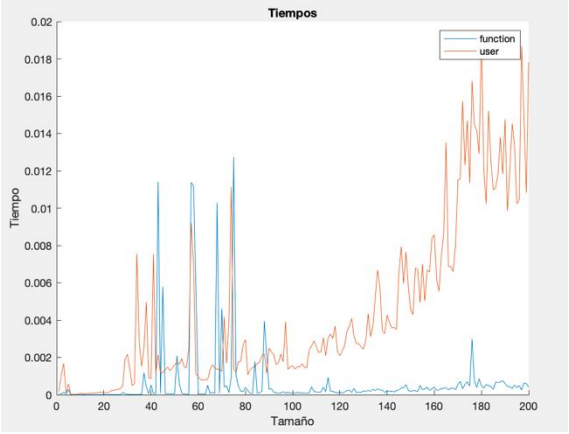
Hemos aprendido como se puede introducir datos desde el teclado para luego utilizarlos en el script mediante la sentencia *input*. También se ha podido ver lo sencillo que es crear una matriz cuadrada de forma aleatoria sin tener que ir dígito a dígito, simplemente utilizando la función rand. Para la segunda parte del ejercicio, se ha utilizado la estructura de los bucles for para poder ir recorriendo la matriz columna a columna, hemos observado que si ponemos entre medias del rango del bucle un número se puede ir iterando sumando el número indicado, en vez de 1 en 1. Existe una sentencia llamada *diag* que permite obtener directamente los valores que están en la diagonal de la matriz. Se ha observado también que existen muchas funciones predefinidas en Matlab que hace más fácil programar, por ejemplo, las funciones de máximo (max), mínino (min), media (mean) y varianza (var). Y por último realizar la gráfica de las funciones ha sido bastante sencillo, simplemente hemos tenido que utilizar la sentencia *plot* indicando lo que se quiera mostrar en la *x* y en la *y*, si realizamos varias plot y tienen el mismo rango se superponen las líneas en la misma ventana. También se puede cambiar el texto que sale en la ventana para "customizar" la gráfica y que podamos entenderla mejor, indicando el label de x e y, la leyenda y el título.



Ejercicio 3 Parte 1

En este ejercicio hemos creado una función para rellenar los datos de una matriz. Para crear funciones es recomendable hacerlo un archivo a parte cuyo nombre coincida con la función.

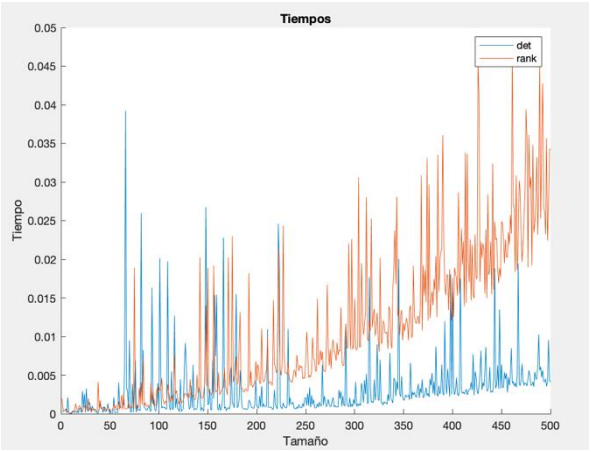
Una vez más hemos comprobado la gran variedad de funciones que proporciona Matlab para trabajar con matrices estando ya implementados la inversa, el rango y el determinante.



Hemos realizado una comparación del tiempo de cómputo de la multiplicación de matrices elemento a elemento *user* y utilizando la multiplicación proporcionada por Matlab *function* pudiendo observar que la creada por nosotros tarda cada vez más según crece el tamaño de las matrices a multiplicar mientras que la implementada en Matlab no crece de forma tan rápida de modo que se puede decir que está mucho más optimizada. Esto nos da a entender que es preferible utilizar las funciones proporcionadas en vez de implementarlas nosotros.

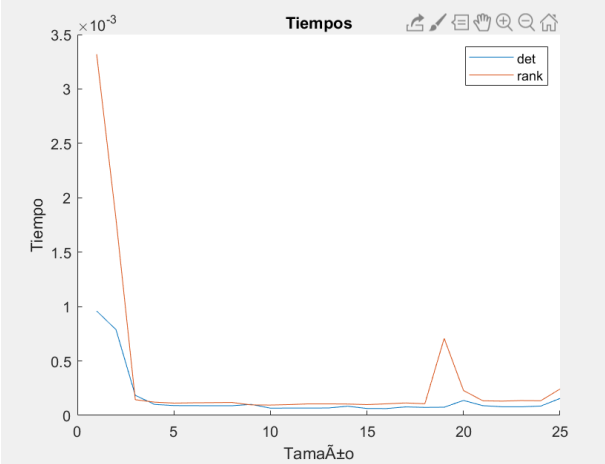
Ejercicio 4 Parte 1

Para obtener los tiempos de cómputo utilizamos las funciones *tic* para iniciar un contador y *toc* para obtener el valor de este. Solo con 25 valores no se obtenía un resultado claro de que al aumentar el tamaño de la matriz aumentara el tiempo de cómputo de modo que probamos con una mayor cantidad de valores obtenido la siguiente gráfica para matrices desde 1x1 a 500x500.



Se puede ver como el rango es más costoso en tiempo que el determinante y que el tiempo de cómputo de ambos crece al aumentar el tamaño de las matrices. Se aprecia también al ejecutar el código múltiples veces que para ciertos tamaños hay picos en los que aumenta el tiempo de cómputo en vez de crearse una curva suave, es decir sin picos.

Aunque no se pudiera obtener un resultado claro, hemos realizado la gráfica también hasta 25 valores:



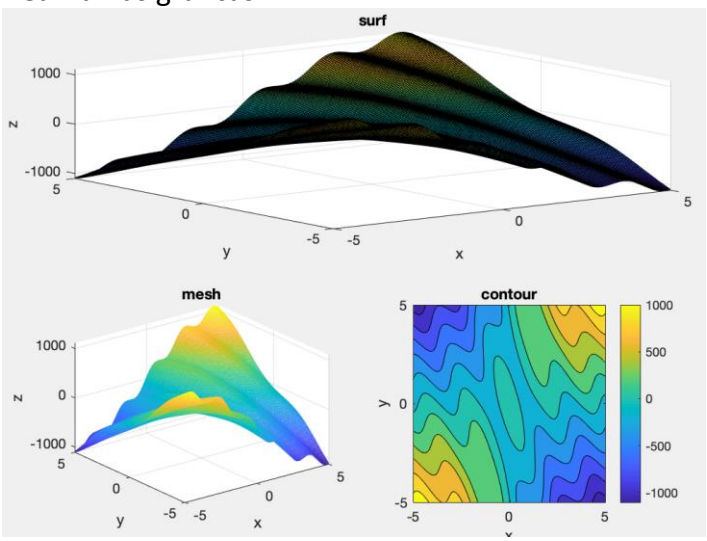
Ejercicio 5 Parte 1

Para realizar este ejercicio se ha dado valores en el rango indicado a la *X* e *Y*, desde -5 hasta 5 cada 0.5. Mediante la sentencia *meshgrid* se ha creado una matriz para poder intersectarla con la función, X es una matriz donde en cada fila tiene una copia de X e Y es una matriz donde tiene en cada columna una copia de Y.

Para representar la función simplemente la hemos ido copiando a Matlab.

A la hora de realizar las gráficas nos hemos encontrado que existen muchas funciones que permite realizar distintas formas de representar las gráficas, en este caso las funciones que hemos utilizado (surf, mesh, contourf), permitía respectivamente representar la superficie, la superficie en forma de malla y el contorno. También existe una sentencia llamada *subplot* que es capaz de dividir la ventana es varias cuadrículas y permitir poner una gráfica en cada cuadrícula o dividir la ventana en las subventanas y el tamaño que se necesite. A la hora de dibujar el área sobre la formula ha sido tan sencillo como poner la gráfica que queremos visualizar y en los parámetros de ésta poner la matriz X e Y y la función Z. Por último, se pone el título y las etiquetas para X Y Z.

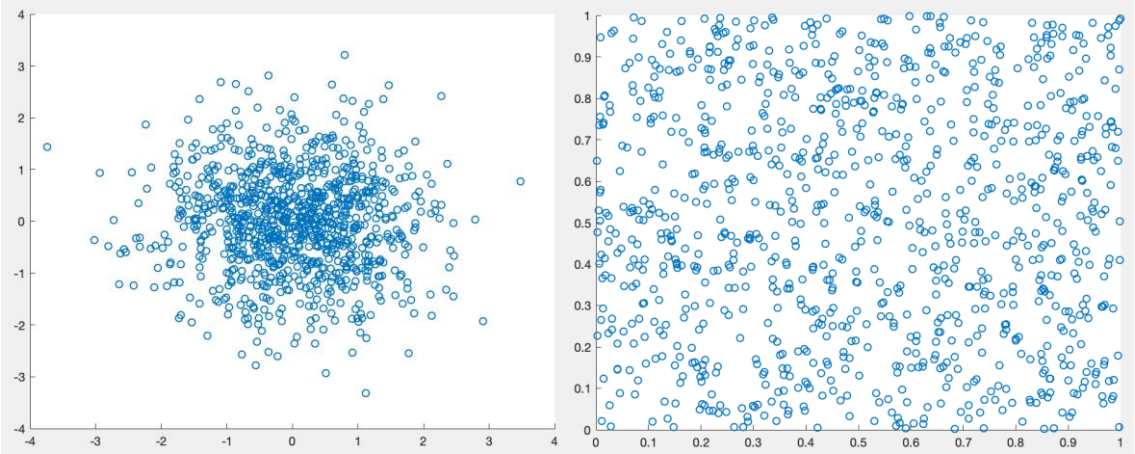
Importante, a la hora de realizar todas las gráficas se ha utilizado la sentencia *hold on* para indicar que se iba a comenzar a realizar las gráficas y *hold off* para indicar que se ha terminado de realizar las gráficas.



Ejercicio 6 Parte 1

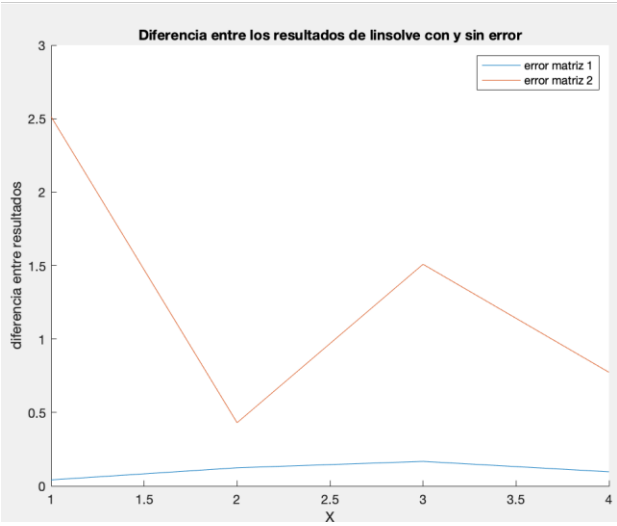
Hemos podido ver la potencia de Matlab para resolver sistemas de ecuaciones con la función *linsolve*. Para calcular el número de condición de las matrices hemos utilizado la función *cond*. Con respecto a la generación de ruido de media 0 y desviación 1 hemos utilizado la función *randn* que genera números aleatorios de forma similar a *rand,* pero con esas características.

Se puede ver una comparación entre los números aleatorios generados por *randn* a la izquierda y por *rand* a la derecha.



Tras calcular el número de condición para ambas matrices vemos que en una es mucho mayor que en la otra. Cuando resolvemos los sistemas de ecuaciones con y sin ruido comprobamos que en la que era el número de condición mayor se obtienen unos resultados muchos más distintos que en la otra. Es decir, cuando el número de condición es menor al introducir ruido la diferencia entre la solución con él y sin él es mayor.

Hemos representado esta diferencia entre las soluciones con y sin ruido para ambas matrices en una gráfica.

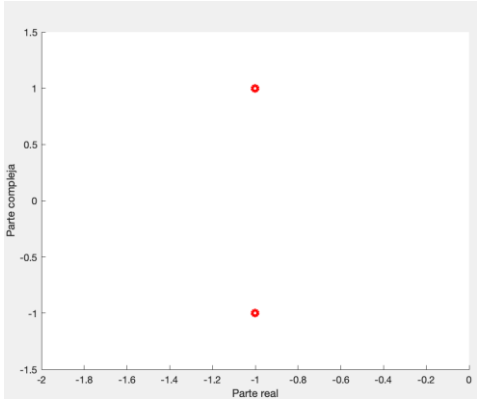


La matriz1 era la que tenía menor número de condición y por eso la diferencia entre los resultados con y sin ruido es más baja.

Ejercicio 7 Parte 1

Para este ejercicio hemos tenido que crear una función en un script aparte que permitía identificar el número de polinomios pasados por parámetro (1 o 2). Mediante la sentencia *nargin* se puede obtener el número de parámetros introducidos, que en este caso son vectores y dependiendo de si son 1 o 2 se hacen las raíces de correspondientes de ese, o de los del producto de ambos. Para hacer las raíces se utiliza la sentencia *roots*. Por último, se calculan el número de raíces que son reales e imaginarias mediante *isreal*. Esta función creada devuelve el número de soluciones reales, el número de soluciones imaginarias y un vector con las soluciones.

En el script principal se realiza un vector con todas las soluciones imaginarias. Esas soluciones se representan una gráfica, hemos indicado que los puntos se representen con un círculo en rojo y de tamaño 3. En el eje X se representa la parte real de la solución y en el eje Y se representa la parte imaginaria de la solución. Al igual que hemos indicado anteriormente se utilizan las sentencias *hold on* y *hold off* para indicar que se ha iniciado el proceso de realizar una gráfica y de terminarlo.

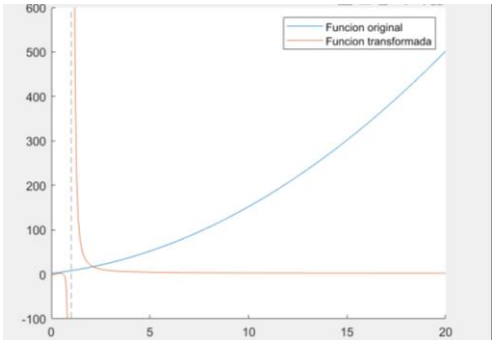


Ejercicio 1 Parte 2

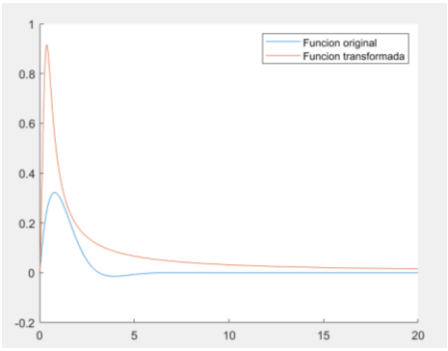
Para calcular la transformada Z de una función debemos utilizar la función *syms* para crear variables simbólicas que podremos utilizar dentro de la función que queramos transformar. La transformación la realizamos con la sentencia *ztrans*.

Para la primera y la segunda función se ha realizado la gráfica mediante *fplot*, realizando la gráfica de la función original y de la transformada respectivamente en la misma figura, también se ha creado un rango del valor *Y* para poder representarlo y visualizarlo de forma más cómoda. Para realizar la función 2, había una variable llamada a, que para poder representarlo en la gráfica le hemos dado un valor constante, en este caso 1.

Gráfica de la función 1:



Gráfica de la función 2:

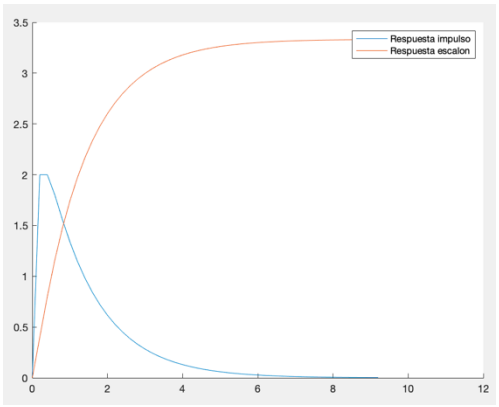


Para la parte 3 del ejercicio hemos creado una función de transferencia discreta utilizando la sentencia *tf*, indicando el *numerador* y el *denominador* en forma de vector y también un *tiempo de muestreo* para la que la función sea discreta, en este caso el tiempo de muestreo es 0.2.

Una vez que se ha generado la función de transferencia discreta se ha procedido a realizar un impulso al sistema mediante la sentencia *impulse*, devolviendo la *respuesta* y el *tiempo de simulación*. Para generar la respuesta a una entrada escalón se ha utilizado la sentencia *step*, devolviendo la *respuesta* y el *tiempo de simulación*.

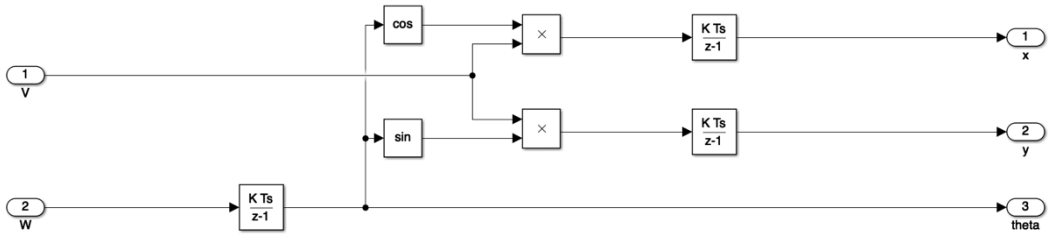
Por último, para realizar la gráfica de este apartado se han utilizado los dos parámetros, en la X el tiempo de simulación y en la Y la respuesta.

Gráfica 3:



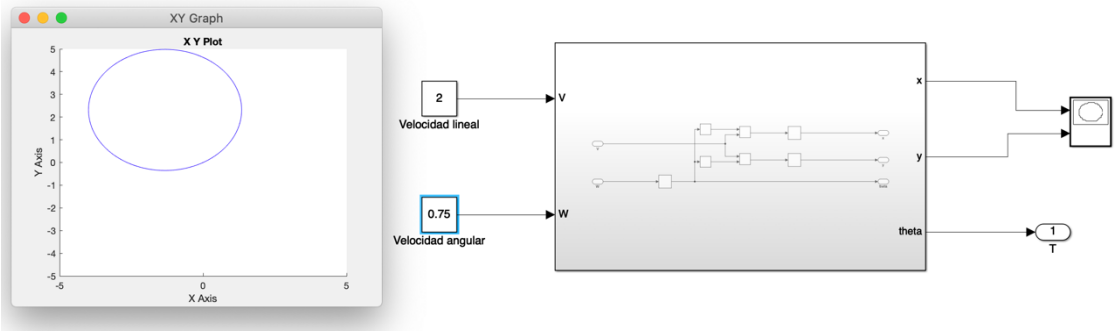
Ejercicio 2 Parte 2

Hemos creado el subsistema del robot simulando con él una plataforma diferencial que toma como entradas una velocidad lineal y una velocidad angular produciendo como salidas la posición x e y del robot, así como su orientación theta.

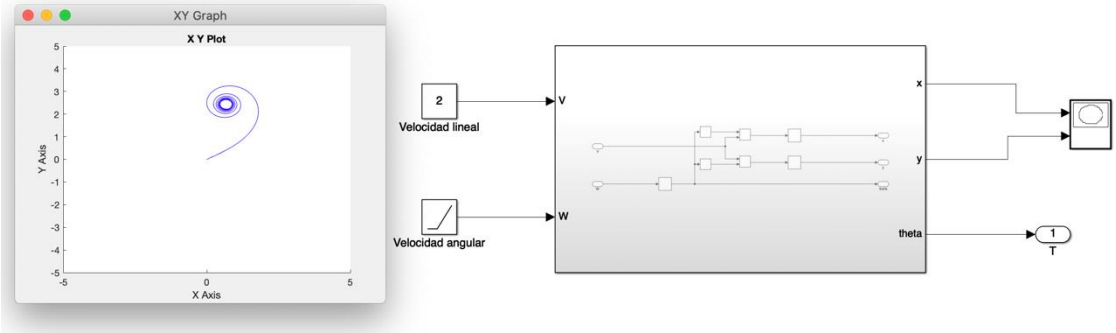


Con el subsistema hemos creado un bloque para poder utilizarlo proporcionándole entras mediante las funciones que proporciona Simulink y visualizando la posición x e y del robot en un gráfico.

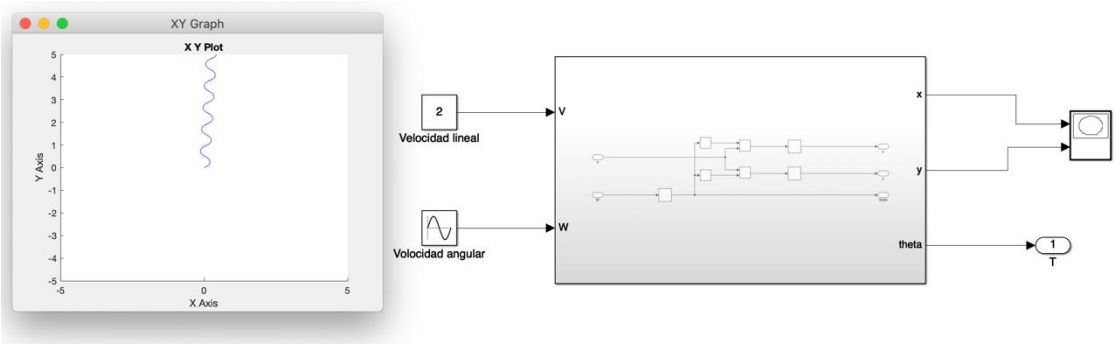
Al proporcionarle constante como entradas hemos visto que el robot traza una trayectoria cerrada elipsoidal tal y como esperábamos.



Cuando le proporcionamos una velocidad angular creciente vemos que el robot traza una trayectoria en espiral cerrada ya que la velocidad angular crece con el tiempo.



Cuando la entrada de la velocidad angular es una función sinusoidal vemos que el robot traza una trayectoria sinusoidal también.



Modificamos la condición inicial del parámetro X y de la Y dentro del subsistema para cambiar las posiciones iniciales en x y en y lo que en las gráficas se muestra de modo que el robot traza las mismas trayectorias, pero empezando en la posición -4, -4.