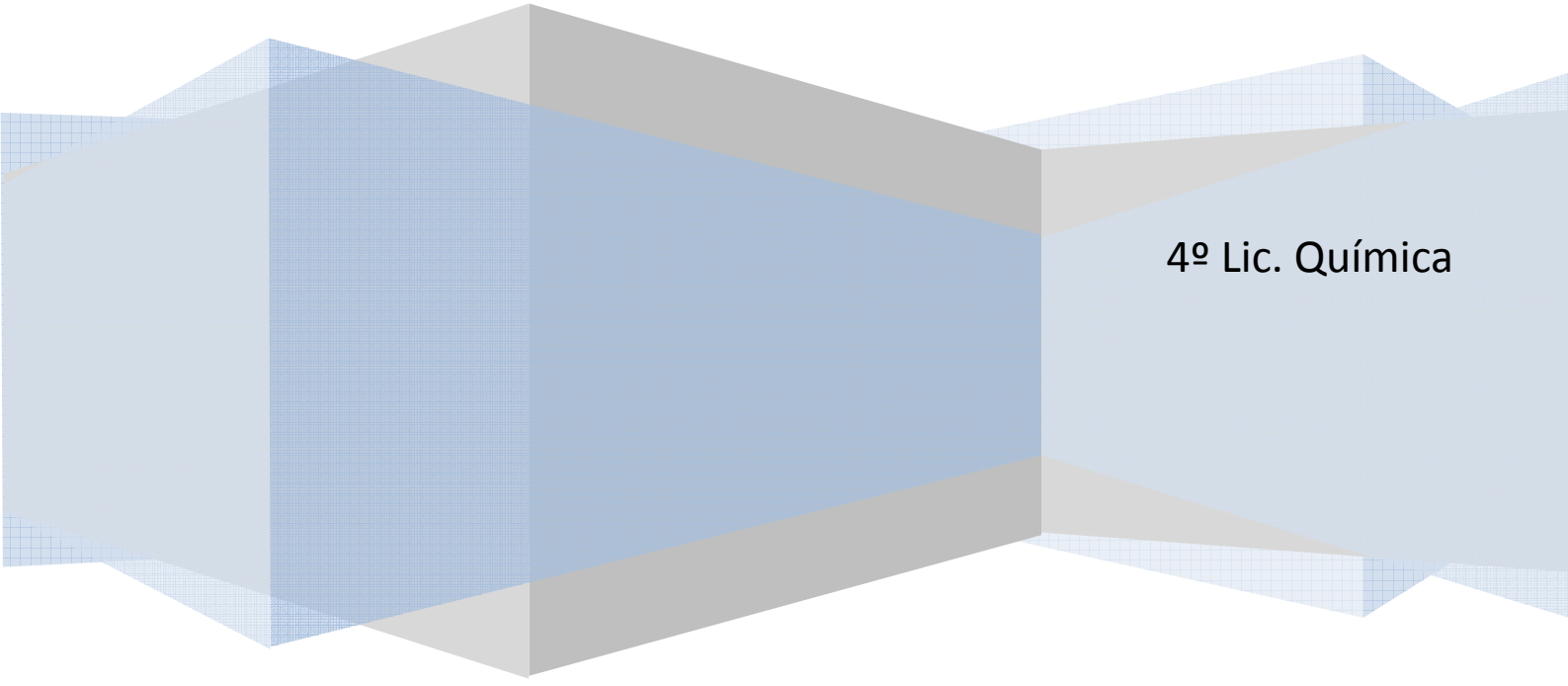
Curso 2007-2008

Sistemas Dinámicos

Rocío Villarán Márquez



4º Lic. Química

1. INICIACIÓN EN EL MATLAB.

Una vez que el programa se inicia, la primera ventana que aparece contiene tres pequeñas ventanas en su interior (es la visión por defecto que ofrece MALTAB), aunque posee ocho:

NOMBRE	SIGNIFICADO	PROPÓSITO
Command Window	Ventana de comandos	Es la ventana principal, se utiliza para introducir variables y ejecutar programas
Figure Window	Ventana de graficos	Se utiliza para visualizar gráficos MATLAB
Editor Window	Ventana de editor	Se usa para crear y depurar ficheros de *script* y funciones MATLAB
Help Window	Ventana de ayuda	Proporciona ayuda e información sobre MATLAB
Launch Pad Window	Ventana de plataforma	Da acceso a herramientas, demos y documentación

Command History Window	Ventana del histórico de comandos	Almacena y visualiza los comandos que se introducen en la ventana de comandos
Workspace Window	Ventana del espacio de trabajo	Proporciona información sobre las variables utilizadas
Current Directory Window	Ventana del directorio de trabajo actual	Muestra los ficheros que hay en el directorio de trabajo actual

1. Ventana de Comandos.

La ventana de comandos es la ventana principal de MATLAB, y se utiliza para la ejecución de comandos, abrir otras ventanas, ejecutar programas escritos por el usuario y gestionar el software de MATLAB.

Debemos de tener en cuanta:

- Para teclear un comando el cursor debe estar situado después del símbolo “>>”, también denominado prompt.
- Una vez qu el comando se ha tecleado y se pulsa la tecla “Intro”, el comando es ejecutado. Sin embargo, solo se ejecuta el último comando. Todo lo ejecutado anteriormente permanecerá inalterado.

- Se puede teclear más de un comando en una sola línea. Para ello solo hay que poner una coma entre comando y comando. Cuando se pulsa “Intro”, todos los comandos se ejecutaran en orden de izquierda a derecha.
- No es posible ir hacia arriba, a una línea supeior, realizar una corrección y reejecutar de nuevo el comando.
- Un comando anteriormente tecleado puede ser invocado de nuevo. Para ello sólo hay que utilizar las flechas arriba y debajo de los cursores para localizar el comando deseado, visualizarlo en el *prompt* y hacer cuantas modificaciones sean necesarias antes de ejecutarlo de nuevo pulsando “Intro”.
- Si un comando es demasiado grande y ocupa más de una línea, éste se puede distribuir en una segunda línea tecleando al final de la primera puntos suspensivos (…) y pulsando “Intro”.
- Cuando se teclea un comando y se pulsa “Intro”, el comando es ejecutado inmediatamente. Cualquier salida que genere el comando se visualizará en la ventana. Si se teclea un punto y coma (;) al final de un comando, la salida de dicho comando no será visualizada.
- El símbolo % al principio de una línea o después de un comando, MATLAB lo considera como un comentario. Esto significa que cuando se pulsa “Intro” no se ejecuta el comentario solo el comando.
- El comando *clc* borra la ventana de comandos. Este comando no cambia nada que haya sido creado antes.

2. FUNCIONES Y FICHEROS DE FUNCIONES.

Una función matemática, f(x), asocia un único número a cada uno de los valores de x. Las funciones se pueden expresar en la forma y=f(x), donde f(x) es habitualmente una expresión matemática en función de x. Cuando se introduce un valor x (entrada) en la expresión de la función, se obtiene un valor y (salida).

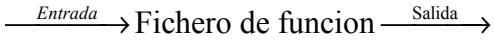
Existen muchas funciones que están ya programadas en MATLAB, y que pueden ser utilizadas en expresiones simplemente tecleando su nombre junto con el argumento de entrada:

FUNCIÓN	DESCRIPCIÓN
sqrt (x)	Raíz cuadrada
exp (x)	exponencial
abs (x)	Valor absoluto
log (x)	Logaritmo natural Logaritmo de base e (ln)
log10 (x)	Logaritmo en base 10
factorial (x)	Función factorial x! (x debe ser un entero

	positivo)
sin (x)	Seno del ángulo x (x en radianes)
cos (x)	Coseno del ángulo x (x en radianes)
tan (x)	Tangente del ángulo x (x en radianes)
cot (x)	Cotangente del ángulo x (x en radianes)

Frecuentemente, a la hora de programar, existe la necesidad de operar con funciones de distintas que no están predefinidas. Cuando la expresión de la función es sencilla y solo necesita ser ejecutada una vez, ésta se puede incluir como código del propio programa. Sin embargo, cuando la expresión se tiene que evaluar muchas veces para diferentes tipos de argumentos, es conveniente crear una función definida por el usuario.

Una función definida por el usuario es un programa MATLAB que el usuario crea y almacena en disco en forma de un fichero que contiene la propia función, de forma que esta función pueda ser usada al igual que el resto de las funciones del sistema. A continuación se muestra de forma esquemática cómo se comporta los ficheros de función:



Los ficheros de función se crean y editan como si se trataran de ficheros *scripts*. Para crear el fichero podemos utilizar el comando

>>edit *nombres del fichero*.m

Una vez que se ha abierto la Ventana del Editor, los comandos se van introduciendo línea por línea. MATLAB enumera automáticamente las líneas cada vez que se pulsa la tecla "Intro".

Antes de ejecutar un fichero *script* tiene que estar guardado en el disco. Para hacer esto sólo hay que ir a la opción Save AS del menú File, y a continuación seleccionar la localización del archivo y su nombre.

Un fichero *script* se puede ejecutar, bien tecleando su nombre en la ventana de comandos (y pulsando Intro) o bien directamente desde la ventana del editor a través del icono Run (Ejecutar).

Cuando se ejecuta un fichero *script*, las variables utilizadas en los cálculos dentro del fichero deben tener valores asignados previamente. La asignación de valores a estas variables se puede realizar de tres formas, dependiendo de donde y como se haya definido la variable:

- Variable definida y asignada en el fichero *script*. En este caso, la asignación del valor de la variable forma parte del fichero. El usuario quiere ejecutar el fichero con un valor diferente para dicha variable, el fichero debe ser editado para asignar un nuevo valor a la variable. Una vez se guardado en el disco, se puede ejecutar de nuevo.

- Variable definida y asignada en la ventana de comandos. En este caso, la asignación de un valor a la variable se realiza en la ventana de comandos. Si el usuario quiere ejecutar el fichero *script* con un valor diferente para la variable, se debe asignar el nuevo valor en la ventana de comandos, y después ejecutar el fichero de nuevo.
- Variable definida y asignada en el fichero *script*, pero además introduce un valor concreto para la variable cuando se ejecuta el fichero en la ventana de comandos. En este caso, la variable se define en el fichero *script* y cuando se ejecuta dicho fichero al usuario se le pide un valor concreto, a través de la ventana de comandos, para asignárselo a la variable del fichero.

2.2. Definición de la función.

La primera línea ejecutable en un fichero de función debe ser la definición de la propia función. La forma que tiene la línea de definición es:

function [argumentos de salida] = nombre de la funcion (argumentos de entrada)

La palabra *function*, tecleada en minúscula debe ser la primera palabra de la primera línea del fichero. En la pantalla esta palabra aparecerá en azul. El nombre de la función se introduce después del signo igual. Este nombre puede estar formado por letras, dígitos y el carácter de subrayado. Es una buena costumbre evitar usar para este propósito nombres de funciones ya predefinidas en o por MATLAB.

Los argumentos de entrada y salida se utilizan para transferir datos hacia dentro y hacia fuera de la propia función. Los argumentos de entrada se introducen entre paréntesis a continuación del nombre de la función. Por lo general, las funciones se declaran con al menos un argumento de entrada, aunque también es posible definir funciones sin ningún argumento de entrada. En cualquier caso, si la función se piensa para más de un parámetro, los argumentos de entrada deben ir separados por comas.

Los argumentos de salida que se encuentran entre corchetes en la parte izquierda del operador de asignación transfieren la salida desde el fichero de función. Estos ficheros pueden tener uno o varios argumentos de salida, e incluso pueden no tener ninguno. Si hay más de uno, entonces los argumentos se deben separar por comas. Si solo hay un argumento de salida, este se puede teclear sin corchete. Para que funcione correctamente un fichero de función, a los argumentos de salida se les deben asignar valores durante la ejecución del código correspondiente al cuerpo de la función.

2. REPRESENTACIÓN GARFICA DE FUNCIONES.

Los gráficos son herramientas muy utilizadas para representar todo tipo de información; información que puede proceder de cualquier campo del conocimiento, pero especialmente de las disciplinas de relacionadas con las ciencias y la ingeniería, donde MATLAB es ampliamente utilizado.

Con los comandos de MATLAB se pueden crear distintos tipos de graficos: estándares con ejes lineales, logarítmicos o semilogaritmicos, de barra y escaleras, polares, de malla y de superficie de contorno tridimensional, etc. Estos gráficos se pueden personalizar para que tenga la apariencia deseada. Así, se puede establecer el tipo, el color y el grosos de línea; se pueden añadir líneas de referencias y cuadriculas; y también títulos y comentarios. Además se pueden superponer varios gráficos sobre un mismo sistema de ejes de coordenadas, o poner varios gráficos en la misma página. Cuando un grafico tiene varios tipos de datos, también se pueden añadir leyendas.

Para poder explicar todo lo que hay que realizar para obtener una representación grafica, lo haremos siguiendo un ejemplo:

Queremos dibujar la función $y = x^3 \sin(x^2)$ $\quad x \in (-1,1)$. Para ello, abrimos un fichero

```
>> edit f.m
```

```
function y=f(x)

y=x.^3.*sin(x.^2);
```

Posteriormente debemos comprobar que el fichero que hemos definido con la función no tiene errores para ello:

```
>> n(0)

                n =0

>> n(1)
```

n = 0.8415

Ahora bien, para dibujar la función debemos antes de nada definir el comando plot.

EL COMANDO plot.

El comando plot se utiliza para crear gráficos bidimensionales. La forma mas sencilla de utilizar este comando es el siguiente

```
plot(x,y)
```

Los argumentos x e y son vectores. Ambos vectores deben tener el mismo número de elementos.

Cuando se ejecuta el comando plot, el grafico se crea en la ventana de graficos. Si no se ha abierto previamente, esta ventana se abrirá automáticamente al ejecutar el comando.la representación del grafico que se mostrará se corresponde con una curva donde los valores de x serán los de la abscisa y los de y la ordenada.

La curva se construye mediante segmentos de recta que unen los puntos cuyas coordenadas están definidas por los elementos de los vectores x e y. Estos vectores pueden tener cualquier nombre, ya que esto no afecta a la representación grafica. Sin embargo, el vector que se introduce como primer argumento de plot será el que define el eje horizontal, mientras que el segundo definirá el eje vertical.

En la pantalla o monitor la curva aparece en color azul, que es el color de línea por defecto. Para personalizar los graficos, si se desea, el comando

plot admite en su sintaxis otros argumentos que se pueden utilizar para definir el color y estilo de líneas y marcadores. Al utilizar estas opciones, el comando plot amplia sintaxis:

```
plot(x,y,'especificadores de linea','propiedades','valores')
```

Los especificadores de línea son opcionales y se pueden utilizar para definir el color y estilo de línea, así como el tipo de marcadores.

Los especificadores de estilo de línea son:

Estilo de línea	Especificador
Solida (por defecto)	-
Discontinua	--
Punteada	:
Rayas y puntos	-.

Para concretar el color, tenemos los siguientes especificadores:

Color de línea	Especificadores
Rojo	R
Verde	G
Azul	B
Cian	C
Magenta	M
Amarillo	Y
Negro	K
Blanco	W

Las posibles marcas que podemos incluir en un gráfico vienen definidas por los siguientes especificadores:

Tipos de marcadores	Especificador
Signo más	+
Circulo	O
Asterisco	*
Punto	.
Cuadrado	s

Diamante	d
Estrella de cinco puntas	p
Estrella de seis puntas	h

Las propiedades y los valores también forman parte de la sintaxis de plot vista anteriormente. Éstos son opcionales, y se utilizan para concretar el grueso de la línea, el tamaño de los marcadores, así como los colores de relleno y del borde del marcador. La propiedad se teclea como cadena, seguida de su valor, también separado por comas. De esta forma deben constituirse pares de propiedad y valor dentro de un mismo comando plot.

Propiedad	Descripción	Posible valor de la propiedad
Linewidth	Especifica el grosor de la línea.	Un nº representado en unidades de puntos (el valor por defectos es 0.5)
Markersize	Especifica el tamaño de las marcas	Un nº representado en unidades de puntos
markeredgecolor	Especifica el color del marcador, o el color del borde de la línea para marcadores con relleno.	Especificadores de color, como los vistos en las tablas anteriores, introducidos en forma de cadena.

markerfacecolor	Especifica el color de relleno de los marcadores.	Especificadores de color, como los vistos en tablas anteriores, introducidos en forma de cadena.

La función *linspace* que también utilizaremos para definir el fichero que nos permita dibujar la función, nos permite generar un vector con n valores igualmente espaciados entre x1 y x2

Linspace(x1,x2,n)

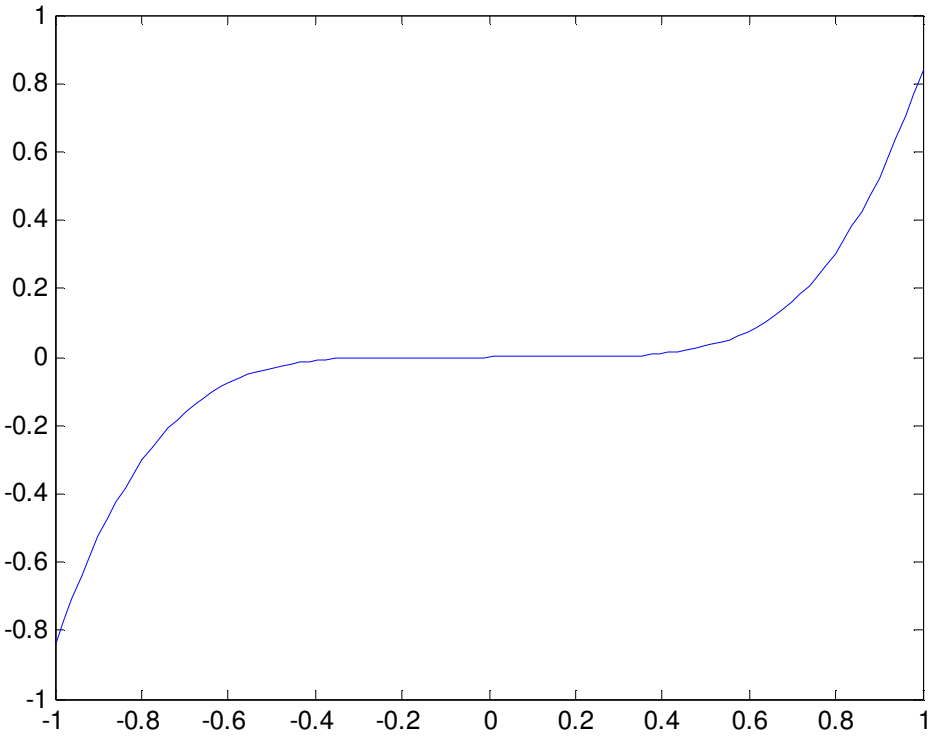
Siguiendo con el ejemplo anterior, creamos otro programa para que nos permita obtener el dibujo de la función

>> edit dibujo.m

```
x=linspace(-1,1,100);
y=f(x);
plot(x,y)
```

Como resultamos obtenemos una grafica como la que se presenta a continuación

>> dibujo

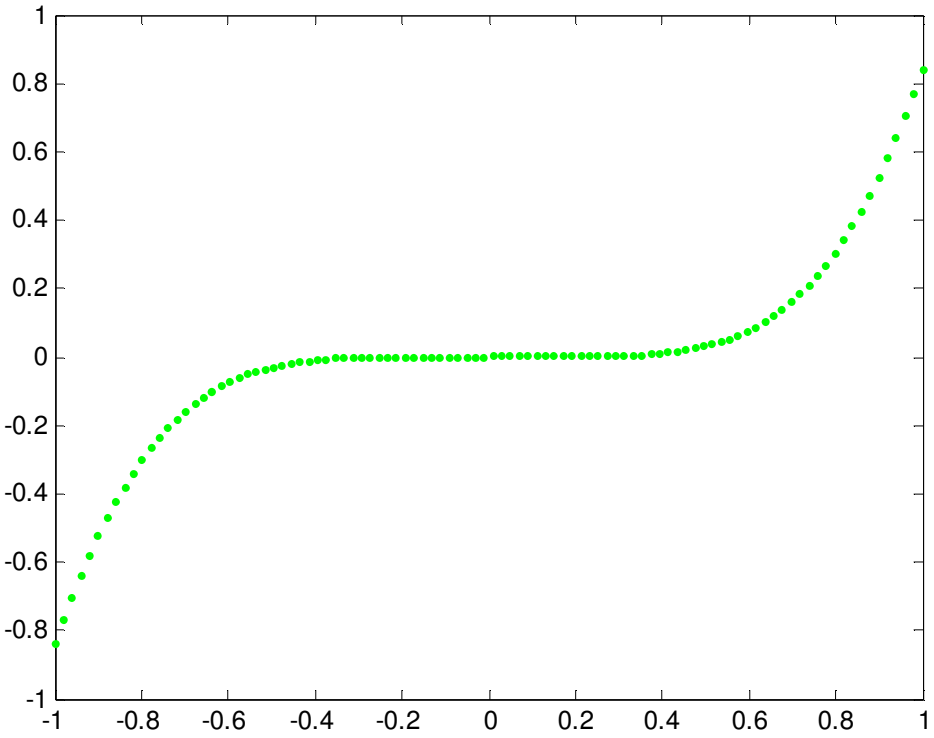


Podemos hacer como hemos dicho anteriormente que la grafica, aparezca con otros especificadores de línea, propiedades y valores, para ello por ejemplo haremos que la curva sea de puntos y en verde. Esto se hace cambiado el fichero creado

>> edit dibujo.m

```
x=linspace(-1,1,100);
y=f(x);
plot(x,y,'.g')
```

>> dibujo



A veces resulta interesante representar varia s funciones a la vez. La representacion de varias funciones se puede realizar de tres formas diferentes. Una de ellas consiste en utilizar el comando *plot*, otra en utilizar los comandos *hold on* y *hold off*, y la tercera consiste en utilizar el comando *line.*

UTILIZACION DEL COMANDO *plot.*

Para representar más de un grafico con este comando es necesario ampliar su sintaxis, tecleando las funciones que se van a representar como pares de vectores, de la forma

```
plot(x, y, v, u, t, h)
```

Este comando crearía tres funciones: y frente a x, v frente a u y h frente a t, todas ellas en la misma región gráfica de la ventana de gráfico. Para ello, los vectores deben ser de la misma longitud. MATLAB dibuja automáticamente las gráficas en distintos colores para que éstas se puedan identificar más fácilmente. En cualquier caso, es posible añadir además especificadores de línea para cada par de vectores o para cada función.

UTILIZACION DE LOS COMANDOS *hold on* Y *hold off.*

La forma de representar varias funciones en un mismo gráfico con estos comandos es utilizar primero el comando *plot* para representar la primera función, y luego introducir el comando *hold on.* Este comando mantiene la ventana de gráficos con el primer gráfico abierto, conservando los mismos ejes y el formato establecido. Una vez introducido este comando se procede a ejecutar tantos comandos *plot* como se quiera.

Finalmente se introduce o ejecuta el comando *hold off* para decirle al sistema que no se desean más representaciones sobre la misma región gráfica, eliminando las posibles propiedades de ejes y formato que se hubieran introducido. El comando *plot* vuelve al estado por defecto.

UTILIZACION DEL COMANDO *line*.

Con el comando *line* se puede añadir curvas (líneas) adicionales a un gráfico que ya existe. La sintaxis del comando es la siguiente:

Line (x, y, propiedades, valores)

La sintaxis del comando *line* es muy parecida a la del comando *plot*, aunque este comando no tiene especificadores de línea. El estilo, el color y los marcadores utilizados se deben especificar mediante las ya conocidas propiedades y sus correspondientes valores.

La diferencia fundamental entre *plot* y*line* es que el comando *plot* crea un nuevo gráfico cuando se ejecuta, mientras que el comando *line* simplemente añade una nueva curva a uno que ya existe previamente. Para crear una representación de varias funciones, la primera se crea con un comando *plot*, y seguidamente se introducen tantos comandos *line* como funciones adicionales se desee.

Para crear un ejemplo de este tipo y tomando como primera función la definida anteriormente y como segunda función $y=(x+2)\cos x$*, representaremos las dos funciones en una misma gráfica. Por esto, debemos definir la segunda función ya que la primera está definida y guardada en el programa.*

>> edit g.m

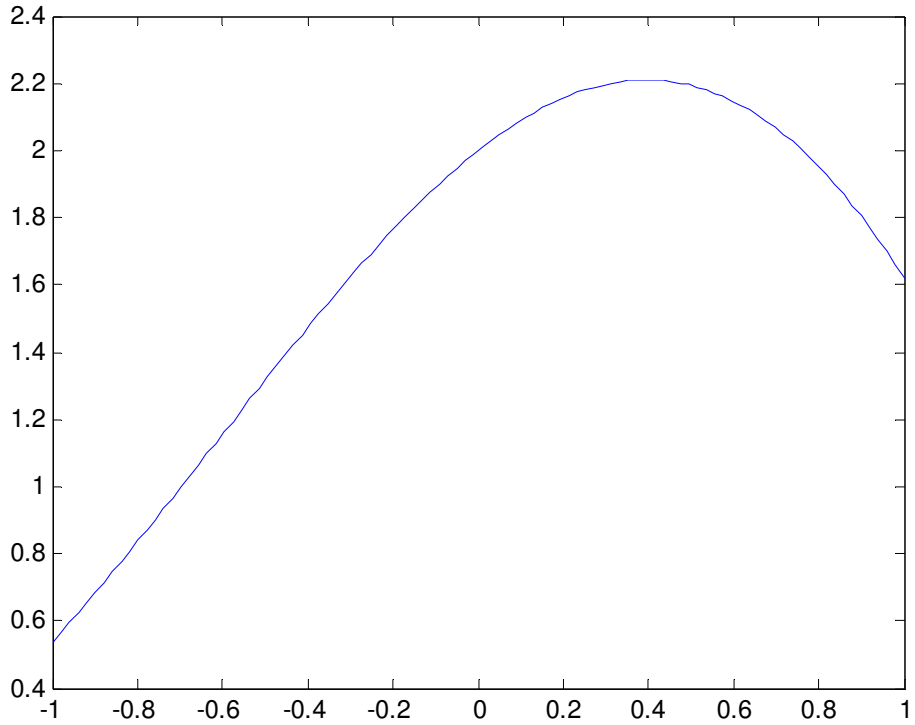
```
function y=g(x)

y=x.^3.*sin(x.^2);
```

Antes de hacer las dos las dos representaciones a la vez, representaremos la segunda función individualmente.

>> edit dibujo.m

```
x=linspace(-1,1,100);

y=g(x);

plot(x,y)
```

>> dibujo

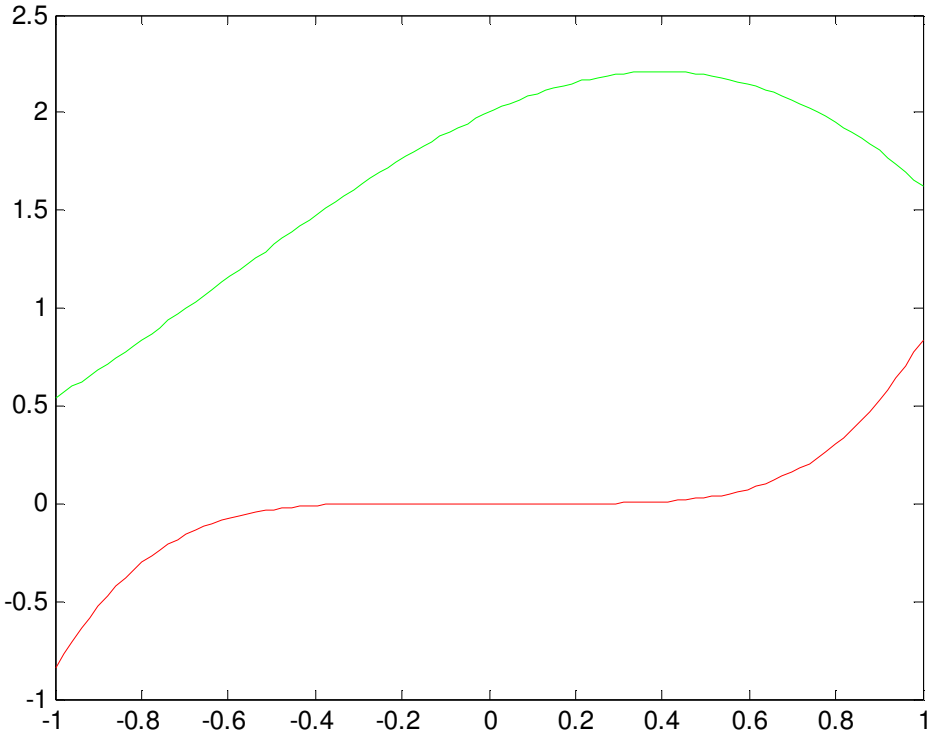


Una vez sabemos la forma de las dos funciones individualmente, podemos hacer la representación conjunta. Para ello modificamos el fichero dibujo de la siguiente forma

>> edit dibujo.m

```
x=linspace(-1,1,100);
y=g(x);z=f(x)
plot(x,y,'g',x,z,'r')
```

>> dibujo



Comandos para poder modificar los gráficos una vez creado:

- Los comandos *xlabel* e *ylabel*, sirven para poner un titulo, en forma de texto, a los ejes. En realidad definen etiquetas que se situarán cerca de cada eje. Su sintaxis es:

```
xlabel('texto');
ylabel('texto');
```

- El comando *title*, añade un titulo (principal) al gráfico, en la parte superior del mismo. Su sintaxis es:

```
title('texto')
```

- El comando *text* permite situar una etiqueta de texto dentro del gráfico. El comando admite dos variantes:

  - El comando *text* coloca el texto en el grafico de manera que el primer carácter se sitúe en el punto con coordenadas x e y.

```
text(x,y,'texto')
```

  - El comando *gtext* coloca el texto en la posición especifica por el usuario. Cuando se ejecuta este comando, se abre la ventana de grafico y el usuario especifica la posición pulsando con el ratón en el punto deseado.

```
gtext('texto')
```

- Con el comando *legend* nos permite colocar una leyenda en la representación gráfica. Las leyendas incluyen una muestra del tipo de línea de cada función que se representa y una eiqueta especificada por el usuario, que permite indicar a qué corresponde cada muestra. La sintaxis de este comando es la siguiente

```
legend('cadenal','cadena2',…,posición)
```

PROBLEMAS.

1. Representar gráficamente la función $y = 3,5^{-0,5x}\cos(6x)$ en el intervalo $-2 \leq x \leq 4$.

Para hacer la representación primero debemos definir la función mediante un fichero

>> edit p.m

```
function y=p(x)
y=(3.5.^(-0.5*x)).*cos(6*x);
```

>> p(0)

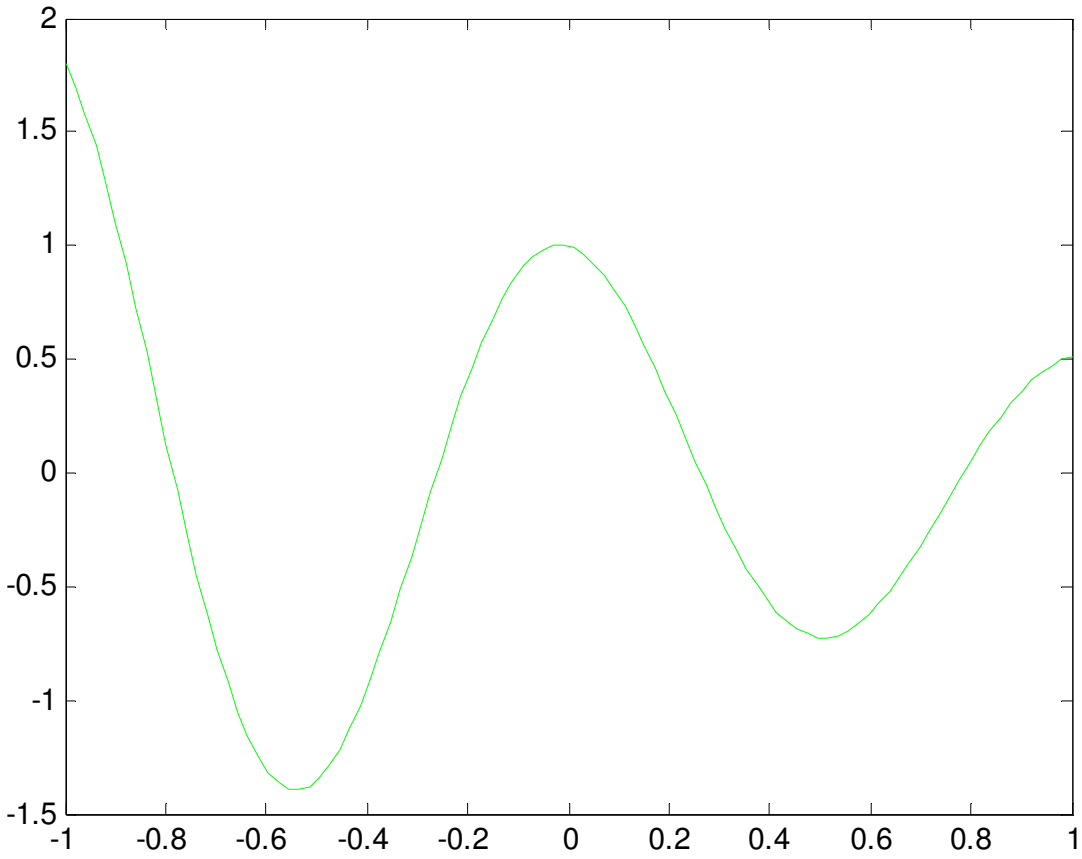
ans = 1

Una vez definida la función y podemos hacer la representación gráfica, haciendo actiar el fichero dibujo:

>> edit dibujo.m

```
x=linspace(-1,1,100);
y=p(x);
plot(x,y,'g')
```

>>dibujo



2. Representa las siguientes funciones en una misma gráfica.

$$y = x\sin(x)$$
$$y = x\cos(x)$$

Lo primero que vamos hacer, es crear los ficheros que nos definan las funciones

>> edit b.m

```
function y=b(x)

y=x.*sin(x);
```

>> edit p.m

```
function y=p(x)
y=x.*cos(x);
```

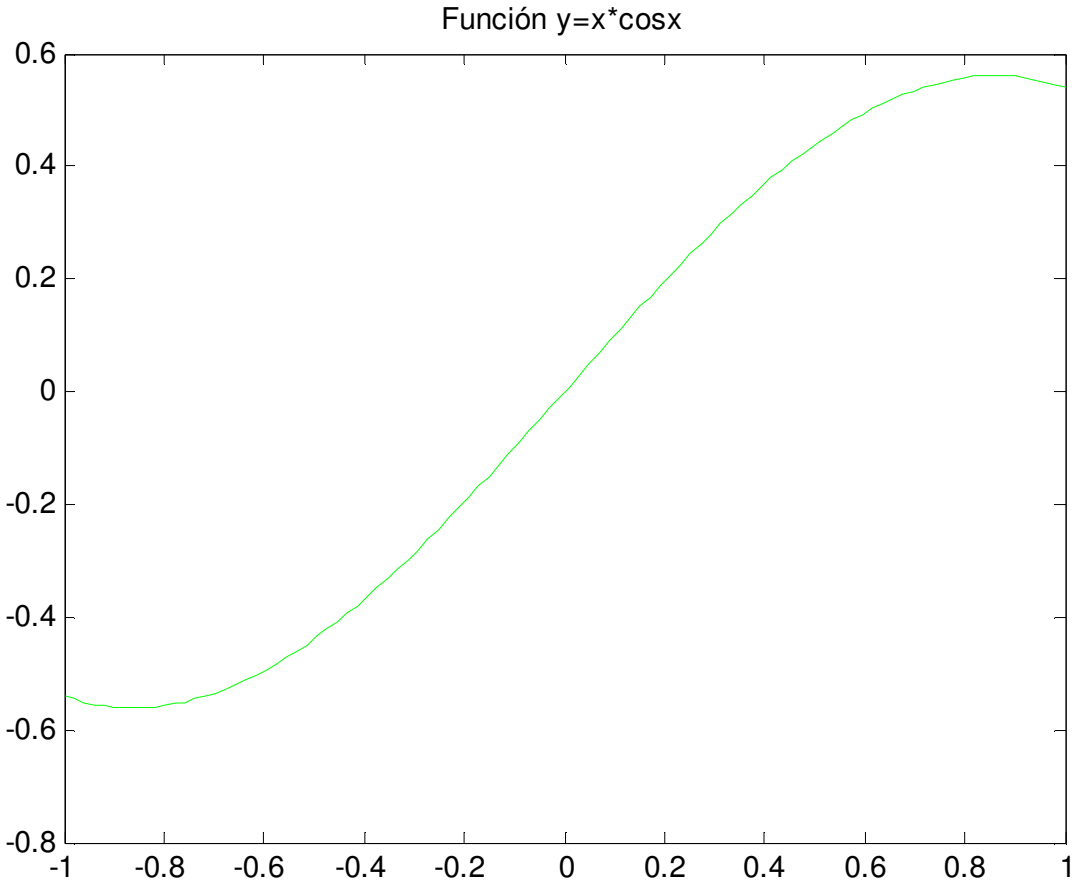
Posteriormente sería conveniente representar por separado las dos funciones

>> edit dibujo.m

```
x=linspace(-1,1,100);
y=p(x);
plot(x,y,'g')
```

>>dibujo

>> title('Función y=x*cosx')

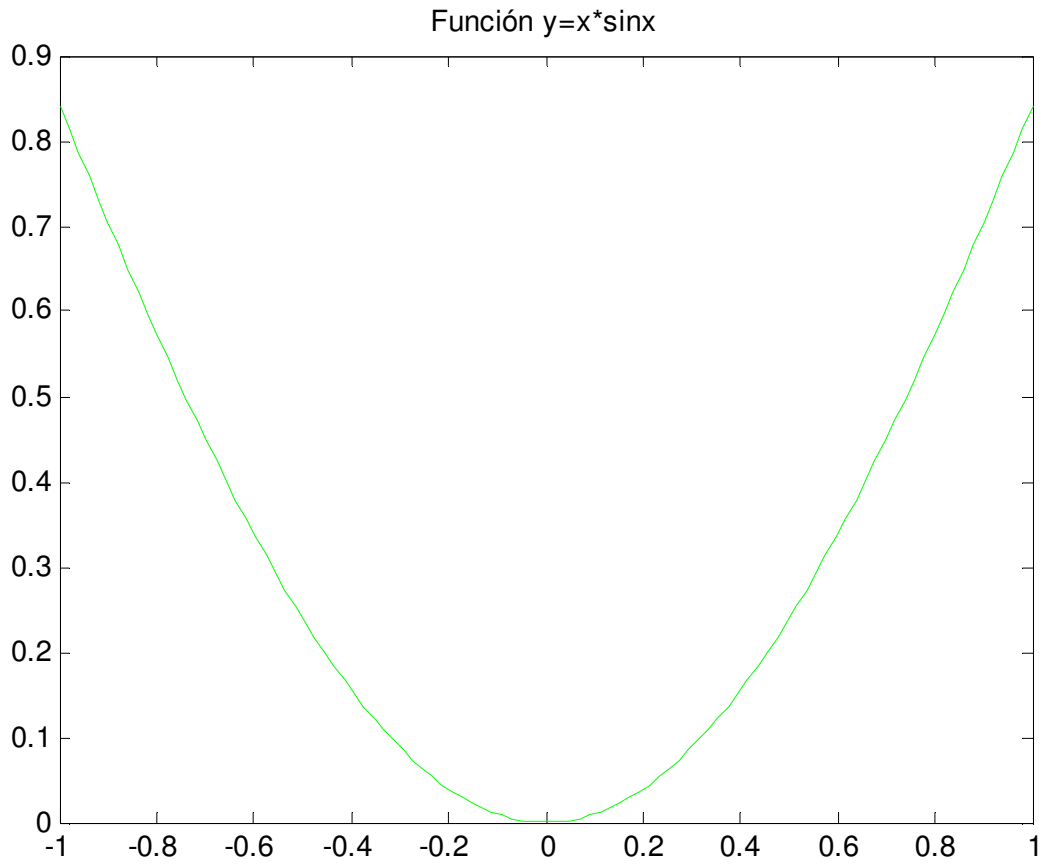


>> edit dibujo.m

```
x=linspace(-1,1,100);
y=b(x);
plot(x,y,'g')
```

>>dibujo

>> title('Función y=x*sinx')



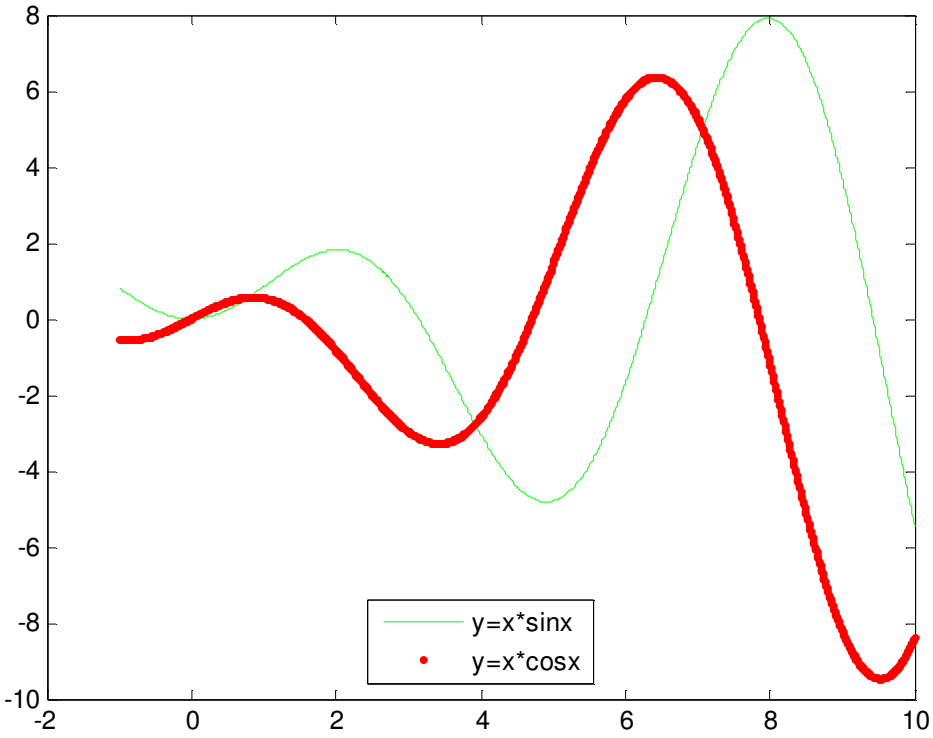
La representación de las dos funciones a la vez se realizara:

>> edit dibujo.m

```
x=linspace(-1,10,1000);
y=b(x);y=p(x)
plot(x,y,'g',x,z,'.r')
```

>>dibujo

>> legend('y=x*sinx','y=x*cosx',0)



3. Representa las tres funciones siguientes en una misma gráfica:

$$\left.\begin{array}{l} y = \sin x \\ y = \cos x \\ y = x^2 - 1 \end{array}\right\} \in [-1,1]$$

Para ello lo primero que hacemos es el definir el intervalo del eje x y el número de puntos en este intervalo.

>> x=-1:0.01:1;

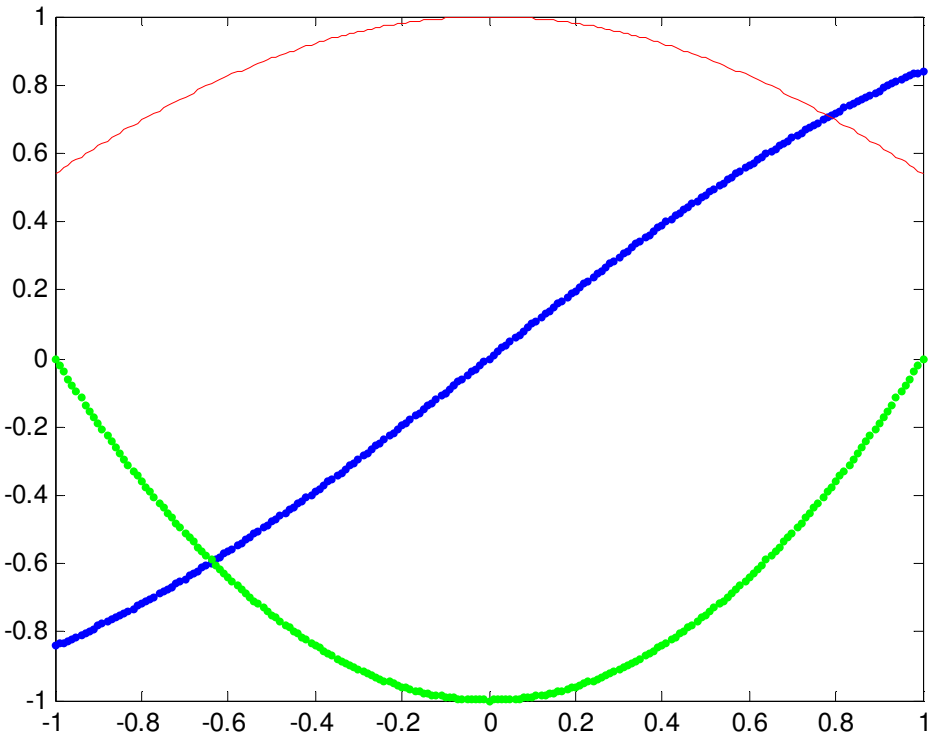
Posteriormente definimos las tres funciones a representar de la siguiente forma

>> y1=sin(x);

```
>> y2=cos(x);

>> y3=x.^2-1;
```

Por último, tenemos que hacer la representación de las 3 funciones descritas

```
>> plot(x,y1,'.b',x,y2,'-r',x,y3,'.g')
```



4. Dibuja la función $y = 3x^3 - 26x + 10$, así como su primera y segunda derivada, en el intervalo $-2 \le x \le 4$. Todas las funciones deben representarse juntas en el mismo gráfico.

Primero creamos un vector x con el dominio de la función y la función

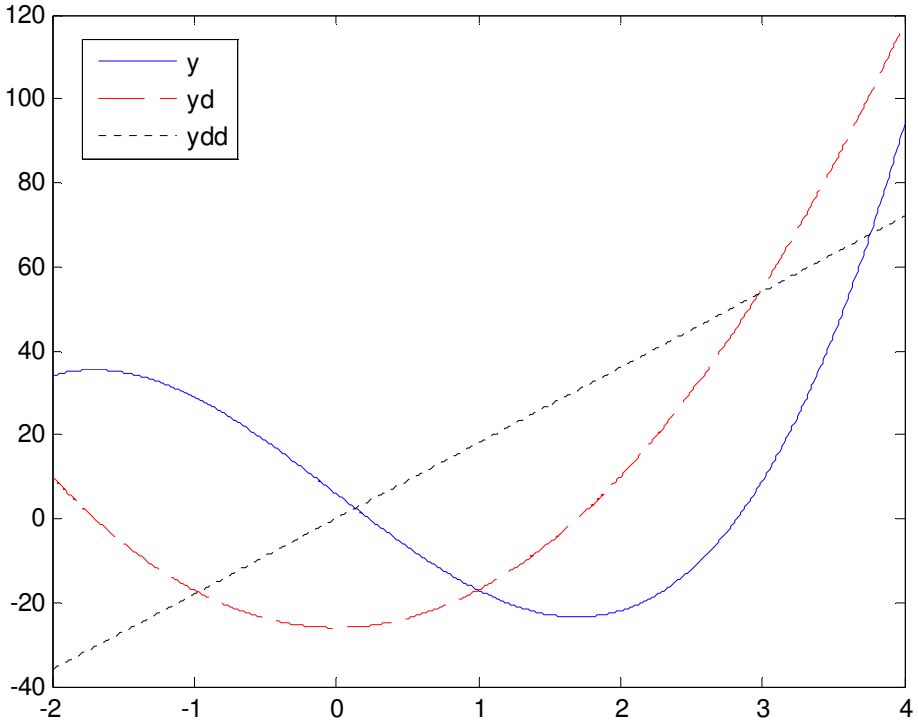
```
>> x=[-2:0.01:4];

>> y=3*x.^3-26*x+6;
```

Sabiendo que la primera deriva y la segunda son $y' = 9x^2 - 26$ e $y'' = 18x$, respectivamente. Creamos los vectores con los valores de la primera y la segunda derivada

```
>> yd=9*x.^2-26;

>> ydd=18*x;
```

Una vez definido todas las funciones a representar, podemos hacer una representación de las ellas en el mismo gráfico.

```
>> plot(x,y,'-b',x,yd,'--r',x,ydd,':k')

>> legend('y','yd','ydd',0)
```



3. INTERPOLACION.

La interpolación es un proceso para estimar valores que se encuentran entre los puntos (datos) conocidos. MALTAB posee funciones para la interpolación basada en polinomios y transformada de Fourier. Nosotros trataremos las primas.

Cuando sólo se tienen dos puntos conocidos, estos se pueden unir mediante una línea racta. Es decir, se puede utilizar la ecuación de la recta (polinomio de primer grado) para estimar diferentes puntos entre dos conocidos.

Cuanto mayor es el número de puntos conocidos, mayor será el grado del polinomio necesario para que pase por todos esos puntos. Sin embargo, el polinomio resultante no tiene por qué proporcionar necesariamente una buena aproximación de los valores que se encuentran entre los puntos conocidos.

Si en lugar de considerar todos los puntos conocidos (utilizando el polinomio que se pasa por todos esos puntos) se consideran sólo unos pocos puntos pertenecientes al entorno donde se va a realizar la interpolación, se puede obtener una interpolación mucho más precisa.

PROBLEMAS.

1. Dado los siguientes datos, hacer la interpolación para obtener el valor de la función para los puntos 1.2 y 1.37.

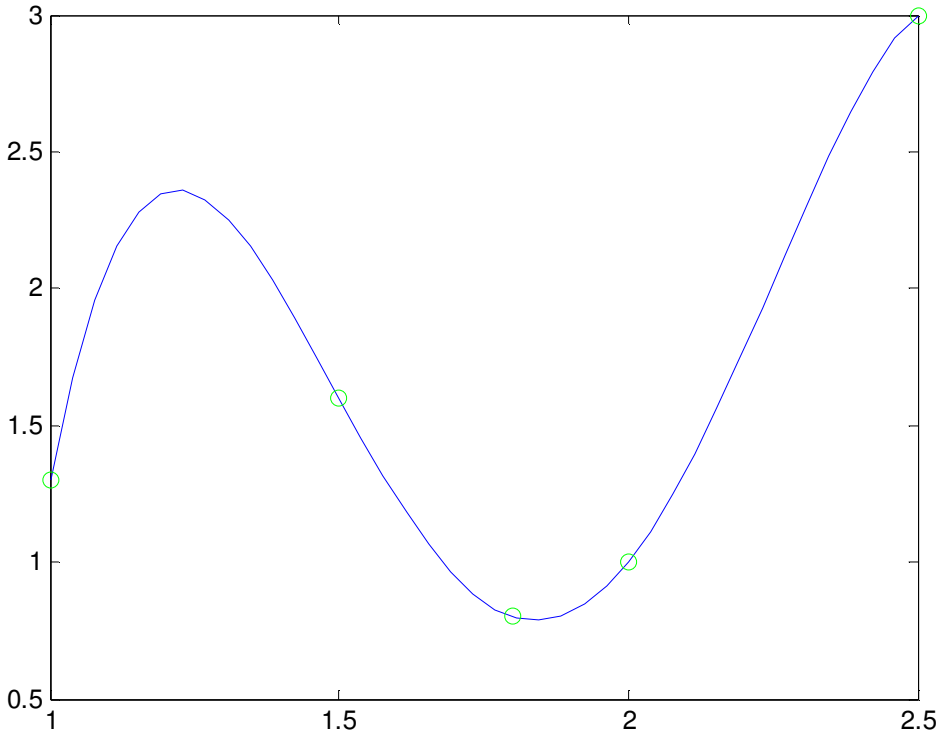
X	1	1.5	1.8	2	2.5
y	1.3	1.6	0.8	1	3

Para obtener estos datos debemos introducir los datos dados anteriormente

```
>> x=[1,1.5,1.8,2,2.5];

>> y=[1.3,1.6,0.8,1,3];

>> p=polyfit(x,y,4)

p =   -9.6429   72.1667 -193.9607  221.0583  -88.3214
```

Ahora debemos realizar un dibujo del polinomio obtenido para ello:

```
>> xx=linspace (1,2.5,40);

>> yy=polyval(p,xx);

>> plot(xx,yy,x,y,'og')
```



Por último, para obtener los valores de los dos puntos por interpolación, damso las siguientes ordenes en el MATLAB.

```
>> format long

>> polyval(p,1.2)

ans =   2.35371428571408

>> polyval(p,1.37)

ans =   2.08018847499986
```

2. Calcula el polinomio de interpolación para los siguientes datos

X	0	1	1.5	2
y	1	1.2	1.7	0.7

Haciendo lo mismo que para el ejercicio anterior obtenemos el polinomio deseado

```
>> x=[0,1,1.5,2];

>> y=[1,1.2,1.7,0.7];

>> p=polyfit(x,y,3)
```

p = $-1.777x^3 + 4.95x^2 - 2.983x + 1$

3. Dado la siguiente tabla de valores de una función, obtener el valor de esta en el punto 0.7.

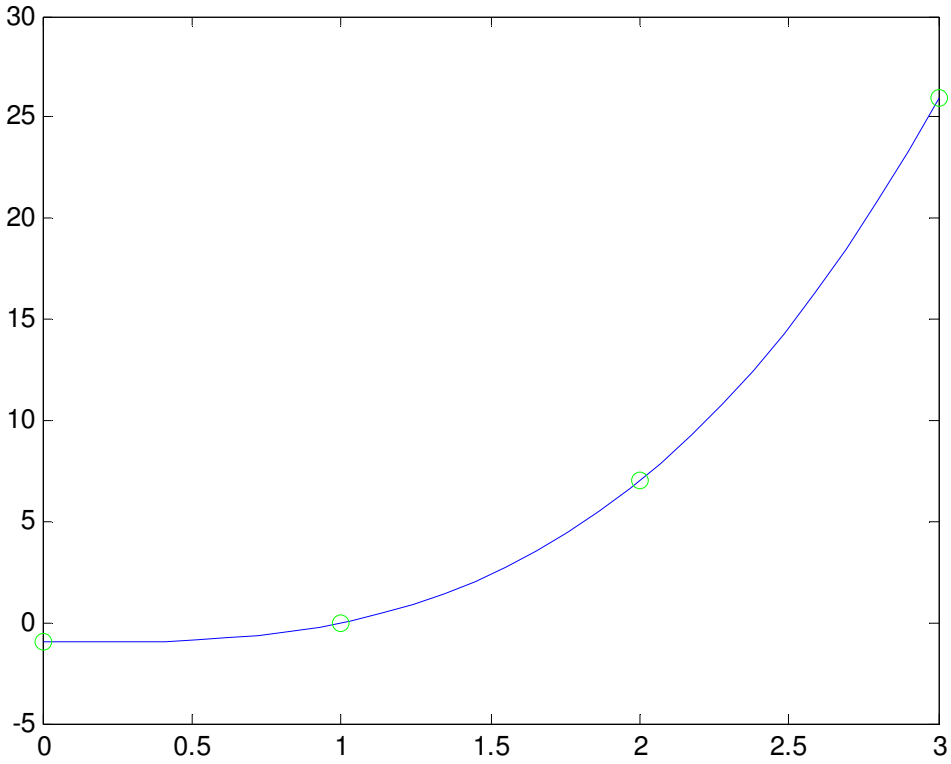
De igual forma que la anterior y siendo los pasos tenemos

```
>> x=[0,1,2,3];

>> y=[-1,0,7,26];

>> p=polyfit(x,y,3)
```

p = $x^3 - 1$

```
>> xx=linspace (0,3,30);
```

```
>> yy=polyval(p,xx);

>> plot(xx,yy,x,y,'og')
```

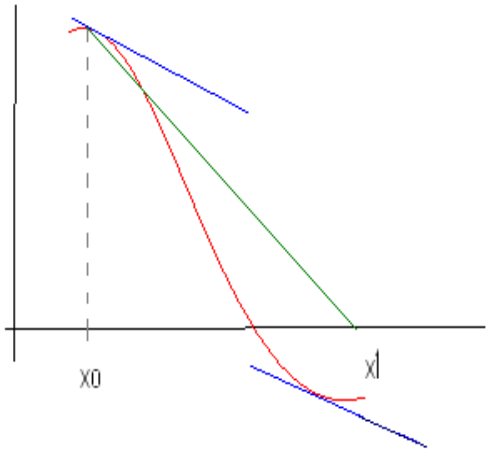


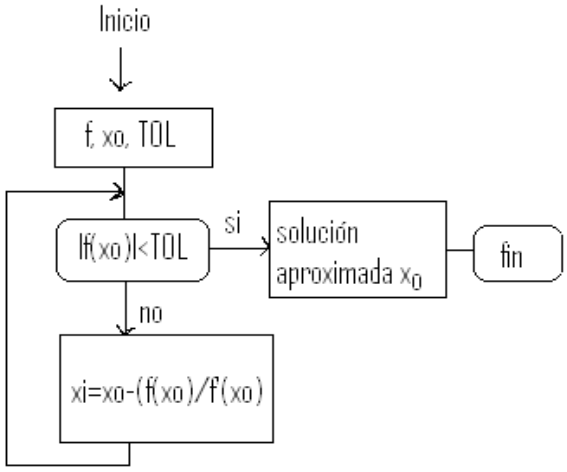
Para el valor de 0.7 tenemos que la función toma el valor de:

```
>> polyval(p,0.7)

ans = -0.657
```

4. METODO DE NEWTON.

El método de Newton resuelve ecuaciones f(x)=0, bajo denominadas condiciones exigidas a f, mediante la iteración $x_{r+1} = x_r - f(x_r)/f'(x_r)$ para un valor inicial dado lo suficientemente próximo a la solución.



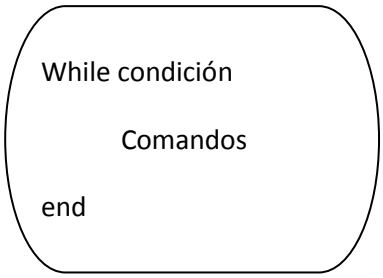


Una vez visto esto, podemos crear el fichero que nos ayudara a calcular las raíces de las funciones

>>edit newtonsol.m

```
function x0=newtonsol(f,fprima,x0,TOL)
while abs(feval(f,x0))>TOL
   x0=x0-feval(f,x0)/feval(fprima,x0);
end
x1=x0;
```

El bucle *while* permite ejecutar de forma repetitiva un comando o grupo de comandos un número determinado de veces mientras se cumple una condición lógica especificada. La sintaxis general de este bucle es:



El bucle empieza siempre con la clausula *while* seguida de una condición, y termina con la clausula *end*, e incluye en su interior yodo un conjunto de comandos que se separan por comas y que se ejecutan mientras se cumple la condición. Si algún comando define una variable, se finaliza con punto y coma para evitar repeticiones en la salida.

Una cosa importante en las matemáticas, es visualizar el caso que estamos estudiando para ello lo primero será ver gráficamente la función en cuestión. Como el dibujo de funciones es muy importante y se deberá de realizar frecuentemente, por este motivo creamos un fichero de dibujo general que no deba ser cambiado según la función que queramos dibujar.

>>edit dibujogeneral.m

```
function []=dibujogeneral(f,a,b,n)
x=linspace(a,b,n);
y=feval(f,x);
plot(x,y)
```

Lo único que debemos hacer antes de usar este fichero es definir la función en cuestión.

EL COMANDO *feval.*

El comando *feval*, del inglés función evaluable, evalúa el valor de una función para un valor dado del argumento de la función. Su sintaxis es:

Variable = feval ('nombre_ función', valor_argumento)

El valor que retorna *feval* se puede asignar directamente a una variable. En caso de que el comando se ejecute sin argumento, MATLAB visualiza *ans =* y el valor de la función.

- El nombre de la función debe introducirse en forma de cadena.
- La función puede ser una función MATLAB u otra cualquiera definida por el usuario.
- Se puede introducir más de un argumento de entrada, para ello los argumentos se separan por comas y se introducen en orden, justo después del nombre de la función.
- Se puede establecer más de un argumento de salida. Para ello sólo es necesario introducir las variables de la parte izquierda de la igualdad entre corchetes.

El comando *feval* es útil en situaciones en las que se necesita calcular el valor de una función dentro de otra función, y es posible que esa función interna sea diferente cada vez que se llama a la función externa.

PROBLEMAS

1. Calcular la raíz de la siguiente función $y = x^3 + x^2 - 1$

Para ello podemos hacer uso de del programa de Newton. De toda forma, el primer paso que debemos dar es representar gráficamente la función.

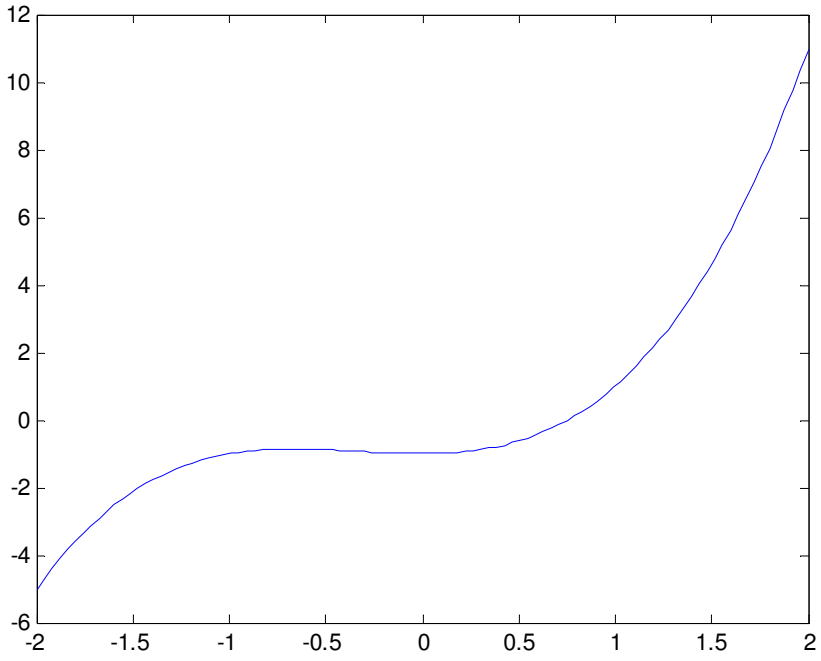
Primero definimos la función de la forma siguiente

>>edit m.m

```
function  y=m(x)

y=x.^3+x.^2-1;
```

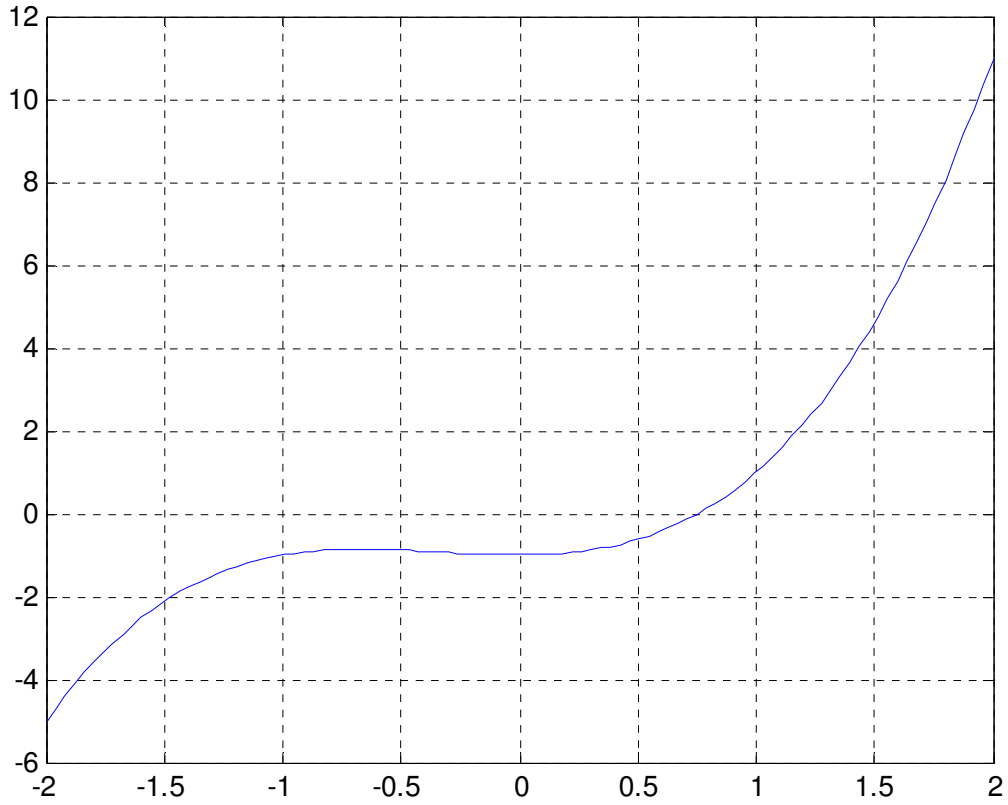
Posteriormente ejecutamos el programa creado para hacer el dibujo de la función definida con anterioridad.

>> dibujogeneral('m',-2,2,100)



Para poder con más certeza donde se encuentra la raíz de la función, la grafica se presenta de la forma siguiente

>> grid



Podemos ver que la función se anula cuando x toma el valor de 0.75 aproximadamente. Para saber el valor exacto creamos el programa que define el método de newton

>>edit newtonsol.m

```
function  x0=newtonsol(f,fprima,x0,TOL)
while abs(feval(f,x0))>TOL
   x0=x0-feval(f,x0)/feval(fprima,x0);
end
x1=x0;
```

Para ejecutar este programa tenemos que definir la primera derivada de la función que definimos como:

>>edit fprima.m

```
function  y=fprima(x)

y=3*x.^2+2*x;
```

Ejecutando el programa obtenemos el siguiente resultado

>> newtonsol ('m','fprima',0.5,10^(-6))

ans = 0.75487767371396

3. Dar el valor de la raíz perteneciente a la siguiente función

$$f(x) = x^3 + x - 1$$

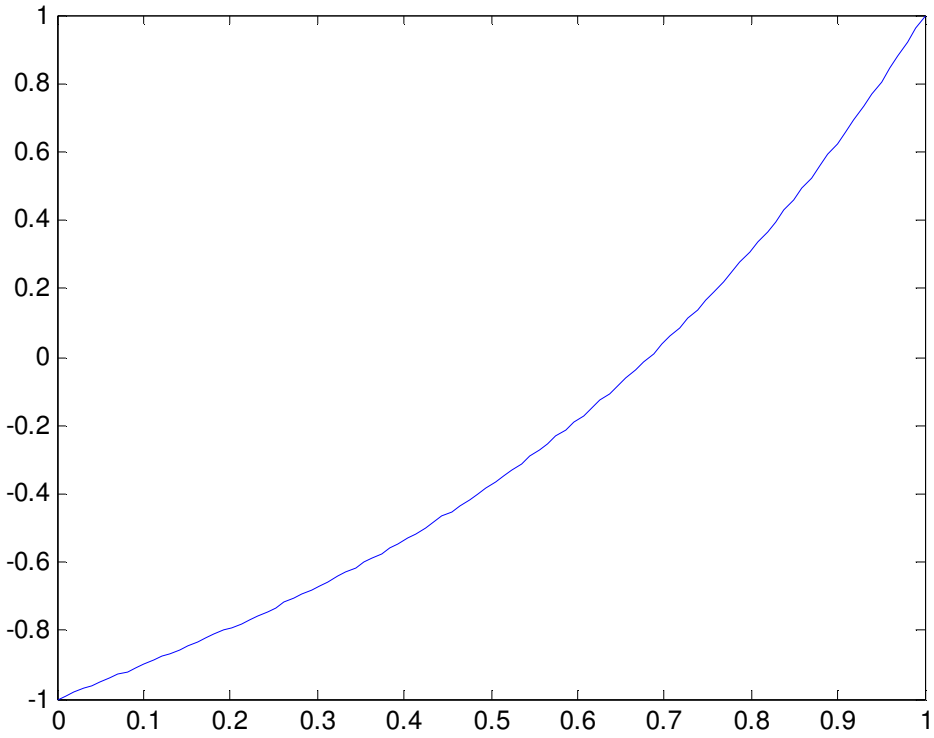
En primer lugar, como hicimos en el ejercicio anterior debemos dibujar la función

>>edit n.m

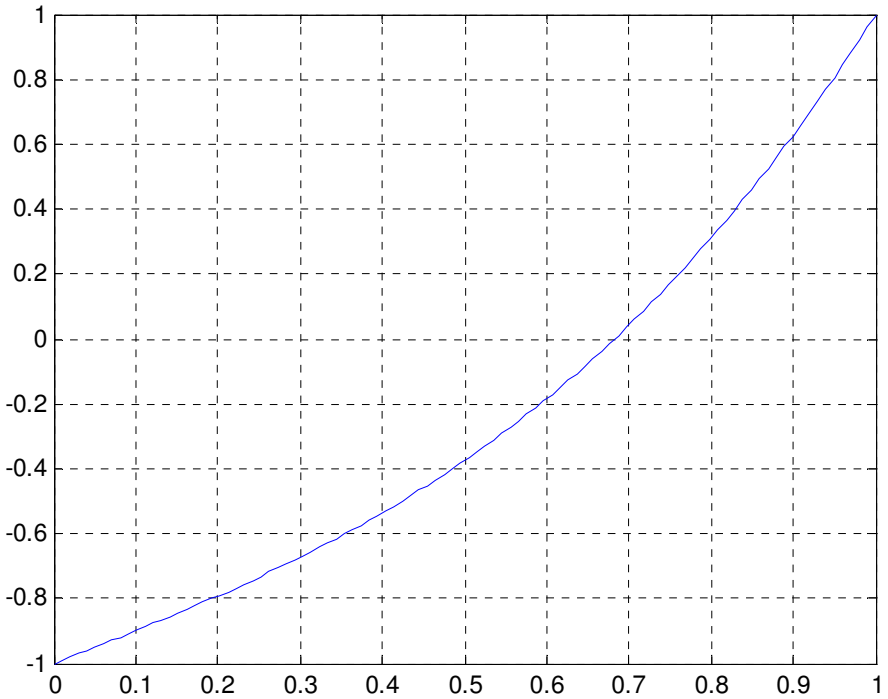
```
function  y=n(x)

y=x.^3.+x-1;
```

Ejecutando el programa de dibujo se obtiene la representación grafica

>> dibujogeneral('n',0,1,100)


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

>> grid


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Como se puede ver en la grafica, el valor de la raíz toma un valor entre 0.6 y 0.7. Para obtener el valor exacto de la raíz, ejecutamos el programa de newton. Sin embargo, antes debemos definir la primera derivada de la función.

>>edit fprima.m

```
function  y=fprima(x)

y=3*x.^2+1;
```

>> newtonsol ('n','fprima',0.6,10^(-6))

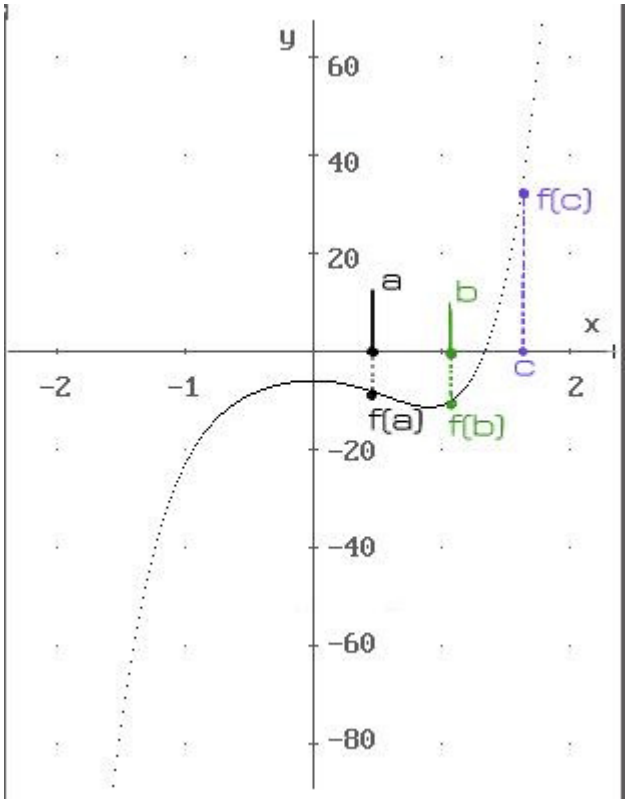
ans = 0.68232780470196

5. METODO DE BISECCIÓN

Con este método, se busca determinar la raíz de una ecuación, o sea, su intersección con el eje de las X o su solución, por lo que se debe tener en cuenta que no todas las ecuaciones tienen una sola solución, y que no todas tienen solución, así que se hay tener una idea de la forma de la curva de la ecuación antes de comenzar a aplicar el método.

Primero hay que saber que lo que hace el método de bisección es, como su nombre lo dice, ir partiendo en dos la distancia entre 2 puntos para obtener un punto central, se hace de la siguiente manera: Se tiran 2 puntos cualesquiera que sean sobre el eje de las X, y entre los cuales se piense que puede estar la raíz, y si no está, el mismo método lo señalara. Después de poner esos 2 puntos que llamaremos A y C se saca un tercero

llamado B, B es el promedio de la distancia entre A y C, por lo que B=(A+C)/2.



El método de bisección se basa en el Teorema del Valor Intermedio

Sea $f(x)$ continua en un intervalo $[a,b]$ y supongamos que $f(a)<f(b)$. Entonces para cada z tal que $f(a)<z<f(b)$, existe un $x_0 \in (a,b)$ tal que $f(x_0)=z$. La misma conclusión se obtiene para el caso que $f(a)>f(b)$.

Básicamente el Teorema del Valor Intermedio nos dice que toda función contínua en un intervalo cerrado, una vez que alcanzó ciertos valores en los extremos del intervalo, entonces debe alcanzar todos los valores intermedios.

En particular, si $f(a)$ y $f(b)$ tienen signos opuestos, entonces un valor intermedio es precisamente $z=0$, y por lo tanto, el Teorema del Valor Intermedio nos asegura que debe existir $x_0 \in (a,b)$ tal que $f(x_0)=0$, es decir, debe haber *por lo menos* una raíz de $f(x)$ en el intervalo (a,b).

El método de bisección sigue los siguientes pasos:

Sea $f(x)$ contínua,

i) Encontrar valores iniciales x_a y x_b tales que $f(x_a)$ y $f(x_b)$ tienen signos opuestos, es decir,

$$f(x_a)\cdot f(x_b) < 0$$

ii) La primera aproximación a la raíz se toma igual al punto medio entre x_a y x_b:

$$x_r = \frac{x_a + x_b}{2}$$

iii) Evaluar $f(x_r)$. Forzosamente debemos caer en uno de los siguientes casos:

- $$f(x_a)\cdot f(x_r) < 0$$

  En este caso, tenemos que $f(x_a)$ y $f(x_r)$ tienen signos opuestos, y por lo tanto la raíz se encuentra en el intervalo $[x_a, x_r]$.

-

$$f(x_a) \cdot f(x_r) > 0$$

En este caso, tenemos que $f(x_a)$ y $f(x_r)$ tienen el mismo signo, y de aquí que $f(x_b)$ y $f(x_r)$ tienen signos opuestos. Por lo tanto, la raíz se encuentra en el intervalo $[x_b, x_r]$.

-

$$f(x_a) \cdot f(x_r) = 0$$

En este caso se tiene que $f(x_r) = 0$ y por lo tanto ya localizamos la raíz.

El proceso se vuelve a repetir con el nuevo intervalo, hasta que:

$$|\epsilon_a| < \epsilon_s$$

es decir,

$$\left| \frac{x_{actual} - x_{previa}}{x_{actual}} \times 100\% \right| < \epsilon_s$$

PROBLEMAS

A. calcular la raíz de la siguiente función $y = x^3 + x^2 - 1$.

Para ello podemos hacer uso de del programa de Newton pero el que nos interesa en este momento es el método de Bisección. De toda forma, el primer paso que debemos dar es representar gráficamente la función.

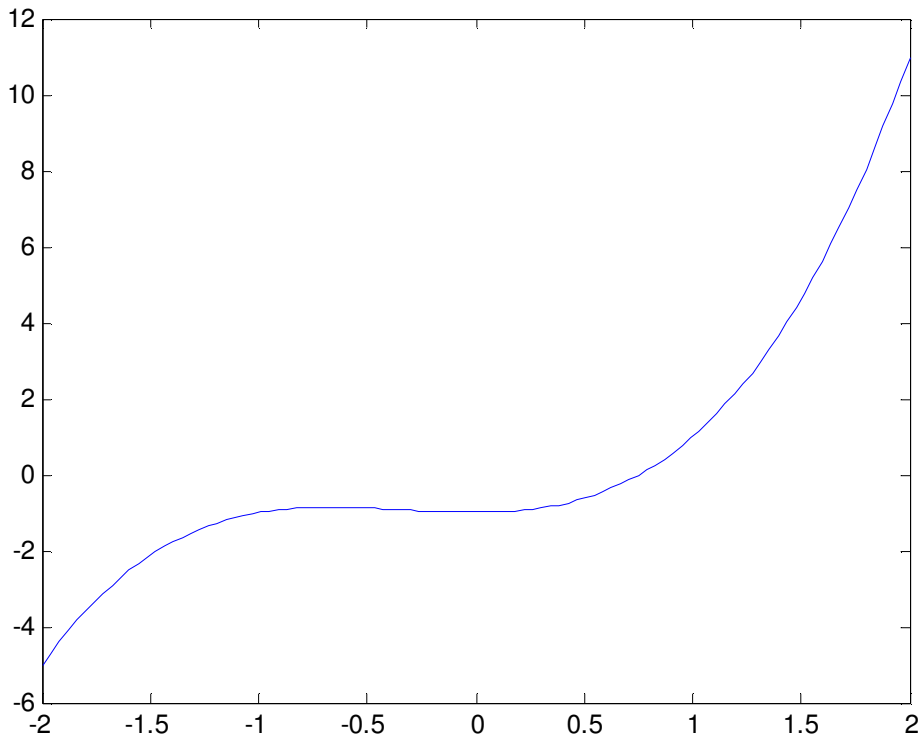
Primero definimos la función de la forma siguiente

>>edit m.m

```
function  y=m(x)

y=x.^3+x.^2-1;
```

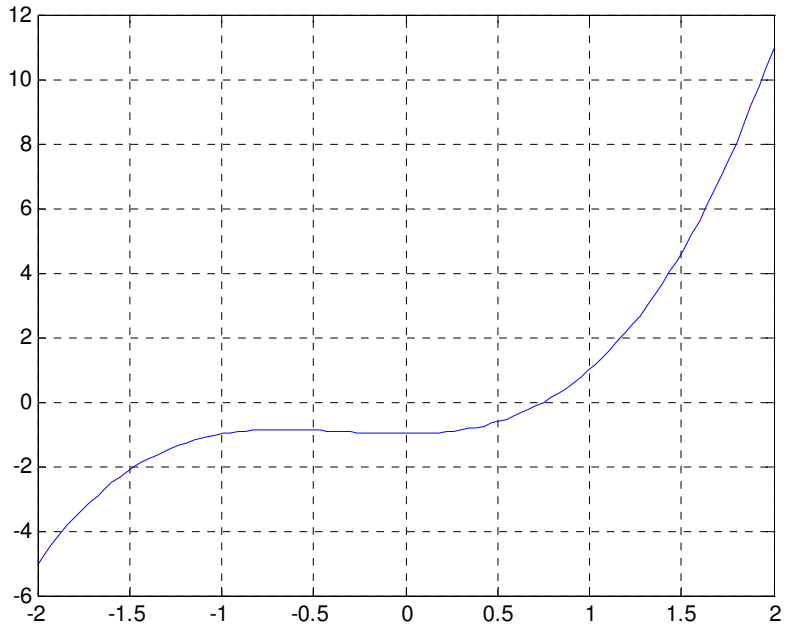
Posteriormente ejecutamos el programa creado para hacer el dibujo de la función definida con anterioridad.

>> dibujogeneral('m',-2,2,100)



Para poder con más certeza donde se encuentra la raíz de la función, la grafica se presenta de la forma siguiente

>> grid



Podemos ver que la función se anula cuando x toma el valor de 0.75 aproximadamente. Para saber el valor exacto creamos el programa que define el método de bisección

>>edit biseccion.m

```
function  p=biseccion(f,a,b,TOL)
while abs(a-b)>TOL
   p=(a+b)/2;
   if feval(f,p)==0
      returm
   end
   if feval(f,a)*feval(f,p)<0
      b=p;
   else
      a=p;
   end
end
p=(a+b)/2;
```

Ejecutando el programa obtenemos el resultado

>> biseccion ('m',0.5,1,10^(-6))

ans = 0.7549

Para tener un valor más exacto:

>> format long

>> biseccion ('m',0.5,1,10^(-6))

ans = 0.75487756729126

B. Dar el valor de la raíz perteneciente a la siguiente función

$$f(x) = x^3 + x - 1$$

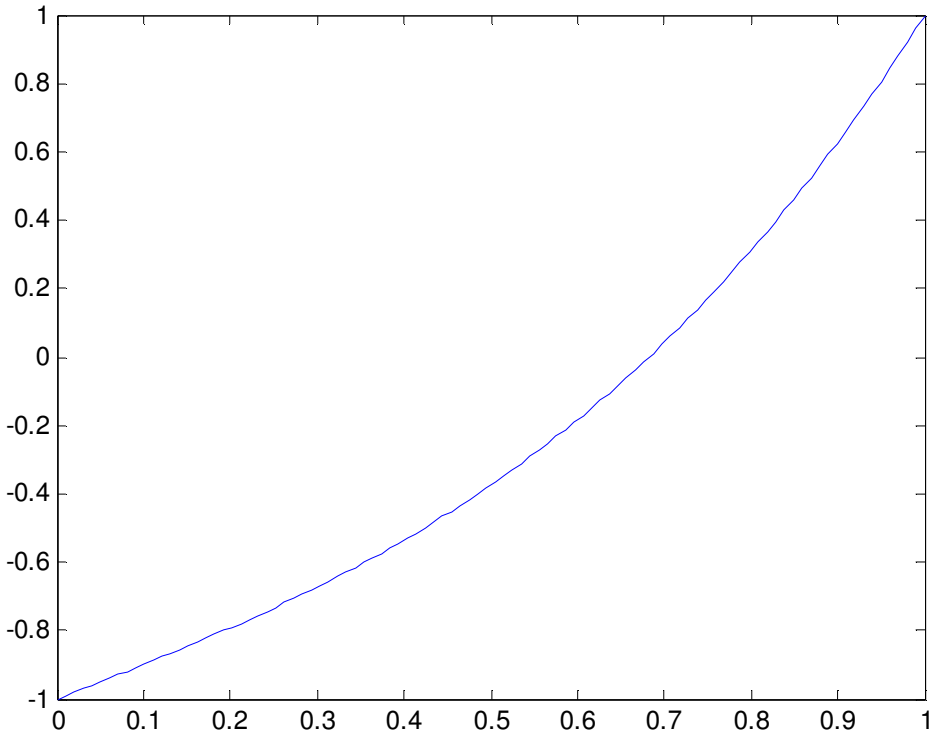
En primer lugar, como hicimos en el ejercicio anterior debemos dibujar la función

>>edit n.m

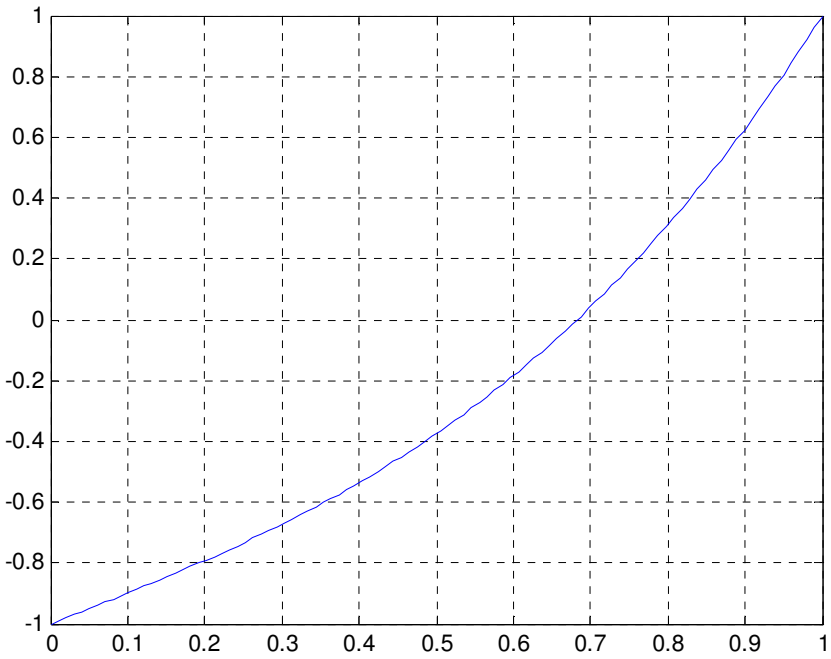
```
function  y=n(x)
y=x.^3.+x-1;
```

Ejecutando el programa de dibujo se obtiene la representación grafica

>> dibujogeneral('n',0,1,100)


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

>> grid


1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

Como se puede ver en la grafica, el valor de la raíz toma un valor entre 0.6 y 0.7. Para obtener el valor exacto de la raíz, ejecutamos el programa de bisección.

```
>> biseccion ('n',0,1,10^(-6))

ans =  0.6823

>> format long

>> biseccion ('n',0,1,10^(-6))

ans =   0.68232774734497
```

6. AJUSTE DE CURVAS

El cálculo de curvas de ajuste, también llamado análisis de regresión, es un proceso que consiste en ajustar mediante una función un conjunto de datos representados por puntos. Esta función puede ser utilizada posteriormente como modelo matemático para los datos. Como hay diferentes tipos de funciones (lineales, logarítmicas, exponenciales, etc) el cálculo de curvas de ajuste puede llegar a ser un proceso complejo. En determinadas ocasiones existe a priori una idea predeterminada sobre el tipo de función que se puede utilizar para ajustar los datos. En este caso el proceso se simplifica, ya que sólo habrá que calcular los coeficientes de dicha función. En otros problemas, cuando no se sabe nada sobre el tipo de datos con los que se trabaja, lo normal es realizar primero distintas representaciones gráficas de los datos para tener una idea sobre el tipo de

función que se puede utilizar para obtener el mejor ajuste posible de los datos.

Se puede emplear polinomios para realizar ajustes sobre datos (puntos) de dos formas distintas. En una de ellas el polinomio pasa por todos los puntos dados, y en la otra el polinomio no pasa necesariamente por todos los puntos, pero en general ofrece una buena aproximación de los datos que se tiene. A continuación se describen estas dos formas de trabajar.

- *Polinomios que pasan por todos los puntos:*

Cuando se tiene n puntos de tipo (x_i,y_i), es posible encontrar un polinomio de grado n-1 que pase por todos los puntos. Por ejemplo, si tenemos dos puntos es posible escribir una ecuación lineal de la forma y=mx+b que pase por estos dos puntos. Si lo que tenemos son tres puntos, la ecuación tendrá la forma $y=ax^2+bx+c$. Generalizando, para n puntos tendremos un polinomio del tipo:

$$a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \ldots + a_1 x + a_0$$

El coeficiente de los polinomios se obtendrá sustituyendo cada punto en la expresión general anterior, resolviendo posteriormente las n ecuaciones resultantes. Los polinomios de grado alto pueden producir errores considerables si se utilizan para estimar los valores que se encuentran entre datos (puntos).

- *Polinomios que no pasan necesariamente por todos los puntos:*

Cuando se tiene n puntos, es posible definir un polinomio de grado menor a n-1 que no tiene por qué pasar necesariamente por todos los puntos

dados, pero que puede darnos una buena aproximación de los datos. El método más común para encontrar el mejor ajuste es el método de los mínimos cuadrados. En este método los coeficientes del polinomio se calculan minimizando la suma de los cuadrados de los valores de todos los puntos (datos) estudiados. El valor residual de un punto se define como la diferencia entre el valor del polinomio (en ese punto) y el punto dado. Por ejemplo, considérese el caso en que se pretende encontrar la ecuación de la línea recta que ajuste a cuatro datos (puntos), tal como se muestra en la figura.

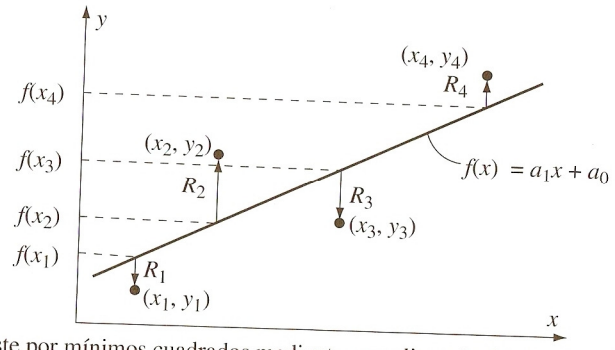


Figura 8.1: Ajuste por mínimos cuadrados mediante un polinomio de primer grado para cuatro puntos.

El polinomio de primer grado necesario para el ajuste es: $f(x) = a_1x + a_0$. El valor residual R_i en cada punto es la diferencia entre el valor de la función en x_i e y_i, es decir, $R_i = f(x_i) - y_i$. Como se pretende obtener el cuadrado de cada residuo, extendiendo esta expresión para todos los puntos se tendrá:

$$R = [f(x_1) - y_1]^2 + [f(x_2) - y_2]^2 + [f(x_3) - y_3]^2 + [f(x_4) - y_4]^2$$

Finalmente R_i queda en función de a_1 y a_0. Para obtener el mínimo de R se calcula la derivada parcial de R con respecto a a_1 y a_0 (dos ecuaciones), igualando las expresiones resultantes a cero.

$$\frac{\partial R}{\partial a_1} = 0 \qquad \frac{\partial R}{\partial a_0} = 0$$

El resultado es un sistema de ecuaciones con dos incógnitas, a_1 y a_0. La solución de este sistema de ecuaciones proporciona los valores de los coeficientes del polinomio que mejor se ajusta a los datos. Se puede seguir el mismo procedimiento en el caso de tener más puntos, así como polinomios de grado superior.

La forma de realizar ajustes con polinomios en MATLAB es mediante la función *polyfit,* cuya sintaxis más básica se muestra a continuación:

```
P=polyfit(x,y,n)
```

La función *polyfit* se utiliza para calcular polinomios de ajuste sobre m puntos, para cualquier grado hasta, como máximo, m-1. Si n=1, el polinomio resultante será una línea recta, si n=2 será una parábola, y así sucesivamente. El polinomio calculado pasará por todos los puntos si n=m-1 (el grado del polinomio es menor que el número de puntos). Es necesario señalar que los polinomios con grados elevados, o que pasen por todos los puntos, no dan siempre el mejor ajuste posible. Los polinomios de grado alto pueden a veces desviarse significativamente entre algunos puntos dados.

En ciencia e ingeniería, así como en muchas otras situaciones, es necesario utilizar funciones distintas de las polinómicas para ajustar datos. Desde el punto de vista teorico se puede utilizar cualquier función para modelar datos dentro de un rango establecido. Sin embargo, para un conjunto específico de datos algunas funciones proporcionan mejores ajustes que otras. Además, determinar los mejores coeficientes de ajuste de ajuste de algunas funciones puede resultar más complejo que en otra.

$y = bx^m$	Función potencia
$y = be^m x$ ó $y = b10^{mx}$	Función exponencial
$y = m\ln(x) + b$ ó $y = m\log(x) + b$	Función logarítmica
$y = \frac{1}{mx + b}$	Función recíproca

Es fácil ajustar datos mediante estas funciones utilizando el comando *polyfit.* Para ello sólo hay que escribir de nuevo las funciones en forma de polinomio lineal (n=1), respetando la forma:

$$y = mx + b$$

La función logarítmica ya tiene esta forma, para el resto de ecuaciones se escribiría:

$\ln(y) = m\ln(x) + \ln(b)$	Función potencia
$\ln(y) = mx + \ln(b)$ o bien $\log(y) = mx + \log(b)$	Función exponencial
$\frac{1}{y} = mx + b$	Función recíproca

Estas ecuaciones reflejan una relación lineal entre *ln(y)* y *ln(x)* para la función potencia, entre *ln(y)* y *x* para la función exponencial, entre *y* y *ln(x)* o *log(x)* para la función logarítmica y entre *1/y* y *x* para la función recíproca. Esto significa que se puede utilizar el comando *polyfit(x,y,1)* para calcular las constantes *m* y *b* que permitan el mejor ajuste posible si se utilizan los siguientes argumentos en lugar de sólo *x* e *y*:

Funcion		**Forma de uso del comando *polyfit***
Potencia	$y = bx^m$	p=plyfit(log(x),log(y),1)
Exponencial	$y = be^m x$ o bien $y = b10^{mx}$	P=polyfit(x,log(y),1) o bien p=ployfit(x,log10(y),1)
Logarítmica	$y = m\ln(x) + b$ o bien $y = m\log(x) + b$	P=polyfit(log(x),y,1) o bien p=ployfit(log10(x),y,1)
Recíproca	$y = \frac{1}{mx + b}$	P=polyfit(x,1./y,1)

El resultado de la función *polyfit* se asigna a *p*, que es un vector de dos elementos. El primer elemento, *p(1)*, es la constante *m*, y el segundo

elemento, *p(2)*, es *b* para la funciones logarítmica y recíproca, ln(b) o log(b) para la función exponencial y ln(b) para la función potencia ($b = e^{p(2)}$ o $b = 10^{p(2)}$ para la función exponencial y $b = e^{p(2)}$ para la función potencia).

Para un conjunto dado de datos, es posible predecir, hasta cierto punto, cuál de las funciones proporcionará un buen ajuste. Para ello hay que representar los datos utilizando diferentes combinaciones de ejes lineales y logarítmicos. Si en uno de los gráficos los datos parecen ajustarse a una línea recta, entonces la función correspondiente puede proporcionar un buen ajuste según la siguiente lista:

Eje x	Eje y	Función
Lineal	Lineal	Lineal $y = mx + b$
Logarítmica	Logarítmica	Potencia $y = bx^m$
Lineal	Logarítmica	Exponencial $y = be^m x$ ó $y = b10^{mx}$
Logarítmica	Lineal	Logarítmica $y = m\ln(x) + b$ ó $y = m\log(x) + b$
lineal	Lineal Plot(1/y)	Recíproca $y = \frac{1}{mx + b}$

Otras consideraciones que debemos tener en cuenta a la hora de elegir una función de ajuste son:

- Las funciones exponenciales no pasan por el origen

- Las funciones exponenciales sólo sirven para realizar ajustes de datos cuyos valores *y* sean todos positivos o todos negativos
- Las funciones logarítmicas no pueden modelar valores nulos (cero) o negativos de *x*
- En la función potencia, *y=0* cuando *x=0*
- La función recíproca no puede modelar *y=0*.

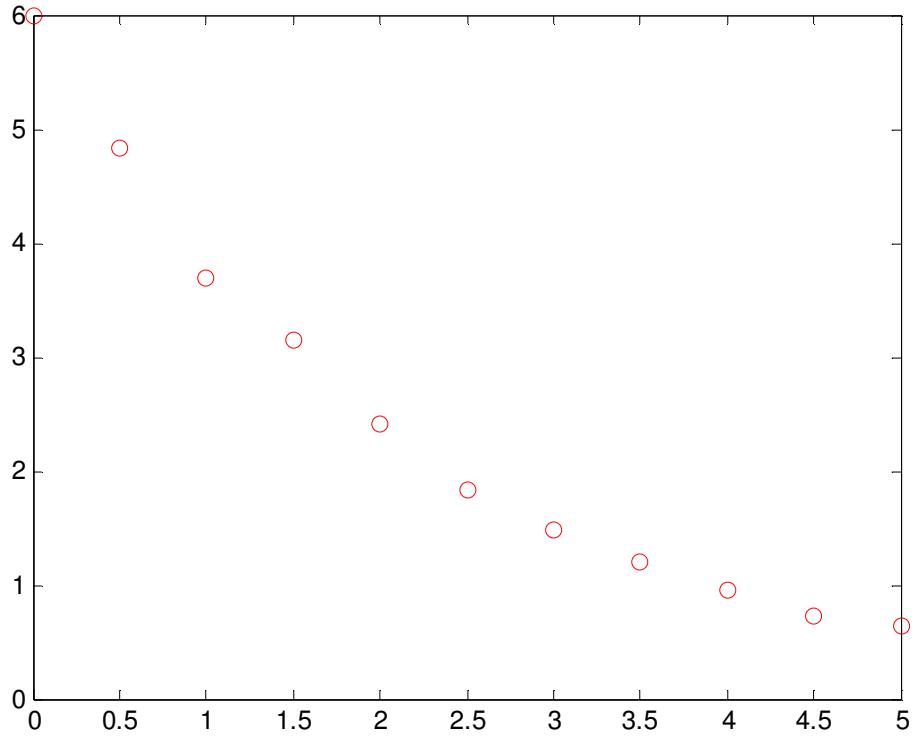
PROBLEMAS

1. Encontrar una función *w=f(t)* (t es la variable independiente y w la dependiente) que mejor ajuste los datos que se detallan a continuación.

t	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0
w	6.00	4.83	3.70	3.15	2.41	1.83	1.49	1.21	0.96	0.73	0.64

En primer lugar se representa los datos con escala lineales en ambos ejes.

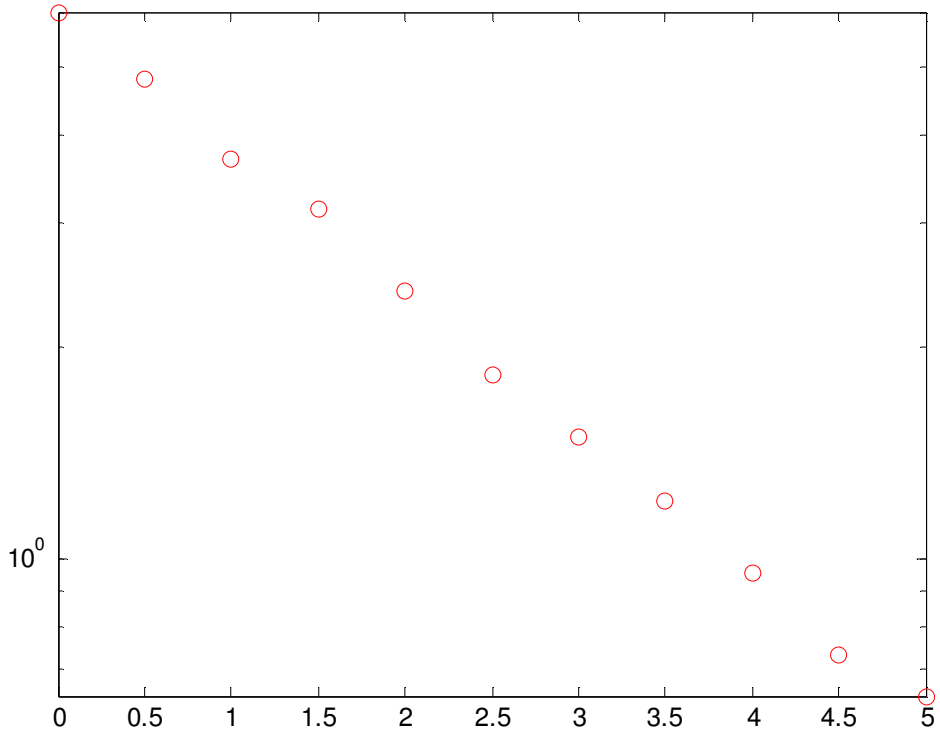
```
>> x=[0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5];

>> y=[6,4.83,3.7,3.15,2.41,1.83,1.49,1.21,0.96,0.73,0.64];

>> plot(x,y,'or')
```



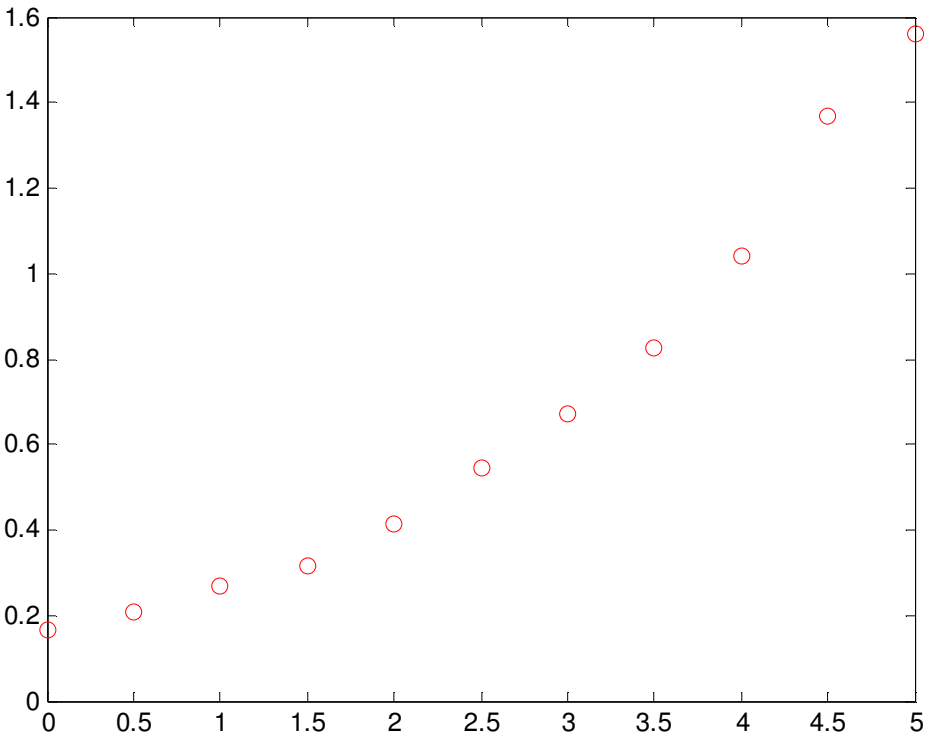
Como se observa en la figura, la función lineal no proporciona el mejor ajuste porque los puntos parecen no seguir una línea recta. De las restantes funciones, la logarítmica también se excluye, ya que el primer punto es t=0. Lo mismo le sucede con la función potencia, ya que para t=0, $w \neq 0$.

Para averiguar cuál de las otras dos funciones posibles (exponencial y recíproca) proporcionará el mejor ajuste, se representan nuevamente los datos.

```
>> x=[0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5];

>> y=[6,4.83,3.7,3.15,2.41,1.83,1.49,1.21,0.96,0.73,0.64];

>> semilogy(x,y,'or')
```




```
>> x=[0,0.5,1,1.5,2,2.5,3,3.5,4,4.5,5];

>> y=[6,4.83,3.7,3.15,2.41,1.83,1.49,1.21,0.96,0.73,0.64];

>>
h=[1/6,1/4.83,1/3.7,1/3.15,1/2.41,1/1.83,1/1.49,1/1.21,1/0.96,1/0.73,1/0.
64];

>> plot(x,h,'or')
```



El primero tiene escala logarítmica en el eje vertical y lineal en el horizontal. El de la debajo tiene escala lineal en ambos ejes, y *1/y* se representa en el eje vertical.

En el primer grafico los puntos parecen seguir una línea recta. Esto indica que una función exponencial, de la forma $y = be^{mx}$, puede dar un mejor ajuste para todos estos datos. A continuación se muestra el fichero que calcula las constantes b y m y que dibuja los puntos y el ajuste calculado.

>>edit exponencial.m

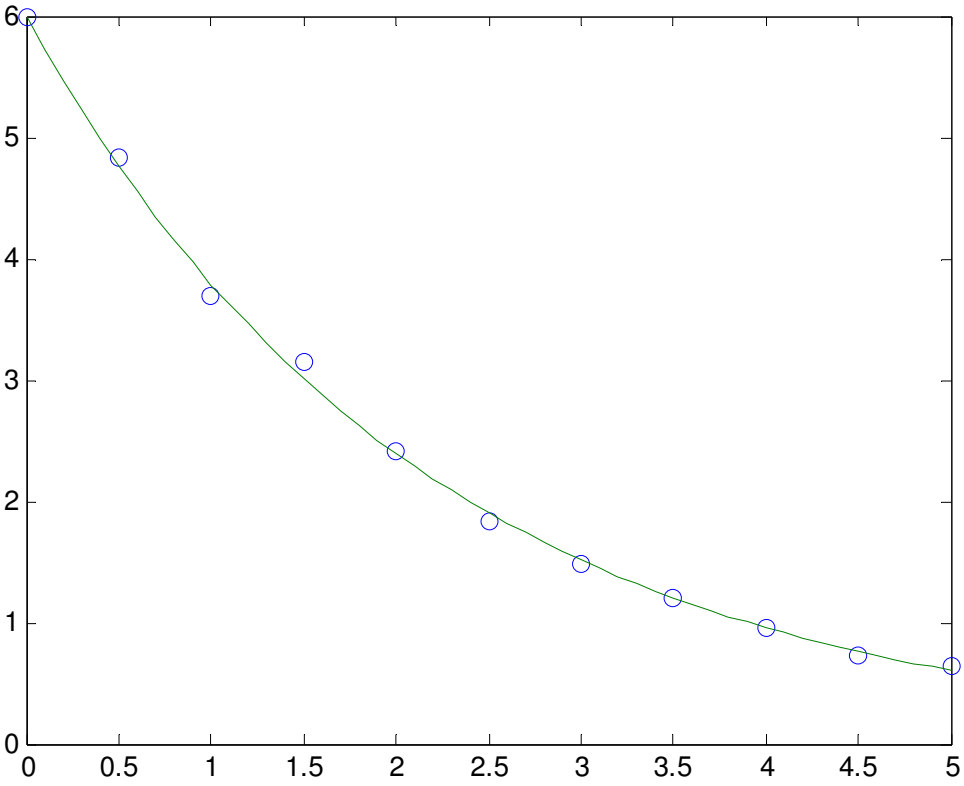
```
x=0:0.5:5;
y=[6,4.83,3.7,3.15,2.41,1.83,1.49,1.21,0.96,0.73,0.64];
p=polyfit(x,log(y),1);
m=p(1)
b=exp(p(2))
xm=0:0.1:5;%se crea un vector para dibujar el polinomio
ym=b*exp(m*tm);%se calcula el valor de la función para cada elemento xm
plot(x,y,'o',xm,ym)
```

Cuando se ejecuta, el programa calcula los valores de las constantes m y b

```
>> exponencial
```

```
m =  -0.4580
b =  5.9889
```

Además, el grafico generado por el programa muestra los puntos y la función de ajuste:

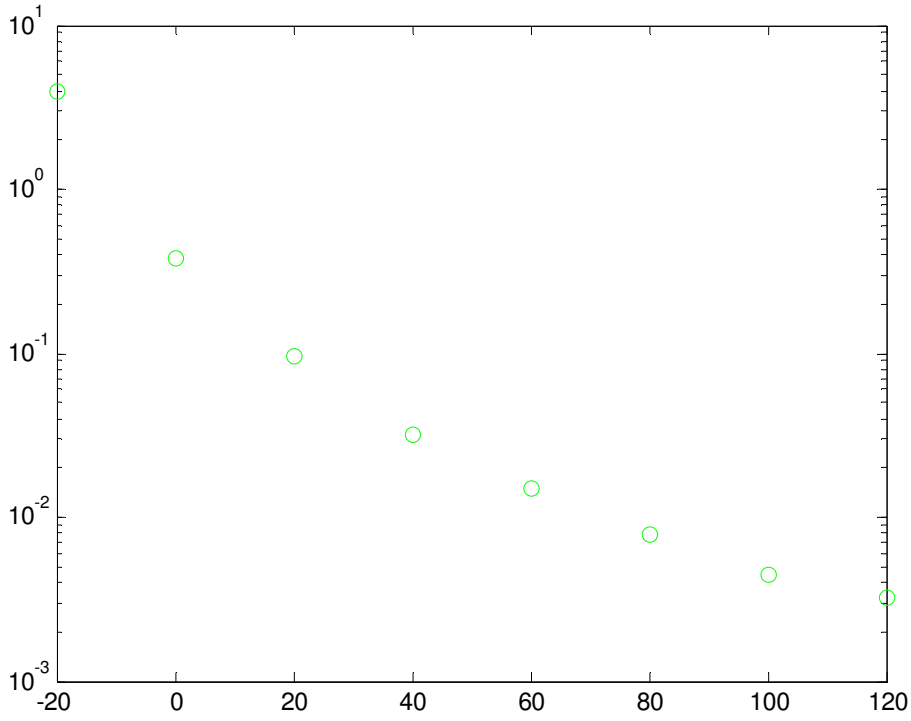


2. La viscosidad, μ, es una propiedad de los fluidos que caracteriza su resistencia a fluir. La viscosidad de la mayoría de los fluidos es muy sensible a la temperatura. A continuación se nuestra una tabla donde se representa la viscosidad de un aceite tipo SAE 10W a diferentes temperaturas. Calcular una ecuación para ajustar los datos de la tabla.

T (ºC)	-20	0	20	40	60	80	100	120
M (N s/m^2)(x 10^{-5})	4	0.38	0.095	0.032	0.015	0.0078	0.00445	0.0032

Para ver qué tipo de ecuación proporciona el mejor ajuste de los datos anteriores, se representa gráficamente μ en función de T, utilizando una escala lineal para T y una logarítmica para μ.

```
>> T=[-20,0,20,40,60,80,100,120];

>> u=[4,0.38,0.095,0.032,0.015,0.0078,0.0045,0.0032];

>> semilogy(T,u,'og')
```



El grafico nos indica que los puntos no aparecen estar dispuestos a lo largo de una línea recta. Esta significa que una función exponencial simple de la forma $y = be^{mx}$, que modela una línea recta en estos ejes, no proporcionará un buen ajuste en este caso.

Como los puntos de la figura parecen ajustarse mejor a una línea curva, una función que posiblemente dé mejor resultados para el ajuste será:

$$\ln(\mu) = a_2 T^2 + a_1 T + a_0$$

Esta función se puede ajustar a los datos utilizando la función *polyfit (x,y,2)* (polinomio de segundo grado), donde la variable independiente es T y la variable dependiente es ln(μ). La ecuación anterior se puede resolver en μ, para dar la viscosidad en función de la temperatura:

$$\mu = e^{(a_2 T^2 + a_1 T + a_0)}$$

El programa siguiente calcula la mejor función de ajuste posible, y crea un grafico para visualizar los puntos y la función calculada.

>>edit viscosidad.m

```
T=[-20:20:120];
mu=[4 0.38 0.095 0.032 0.015 0.0078 0.0045 0.0032];
TK=T+273
p=polyfit(TK,log(mu),2);
Tplot=273+[-20:120];
muplot=exp(p(1))*Tplot.^2+p(2)*Tplot+p(3));
semilogy(TK,mu,'o',Tplot,muplot)
```

Cuando este programa se ejecuta, se calcula los coeficientes mediante la función *polyfit*, y éstos se visualizan en la pantalla.

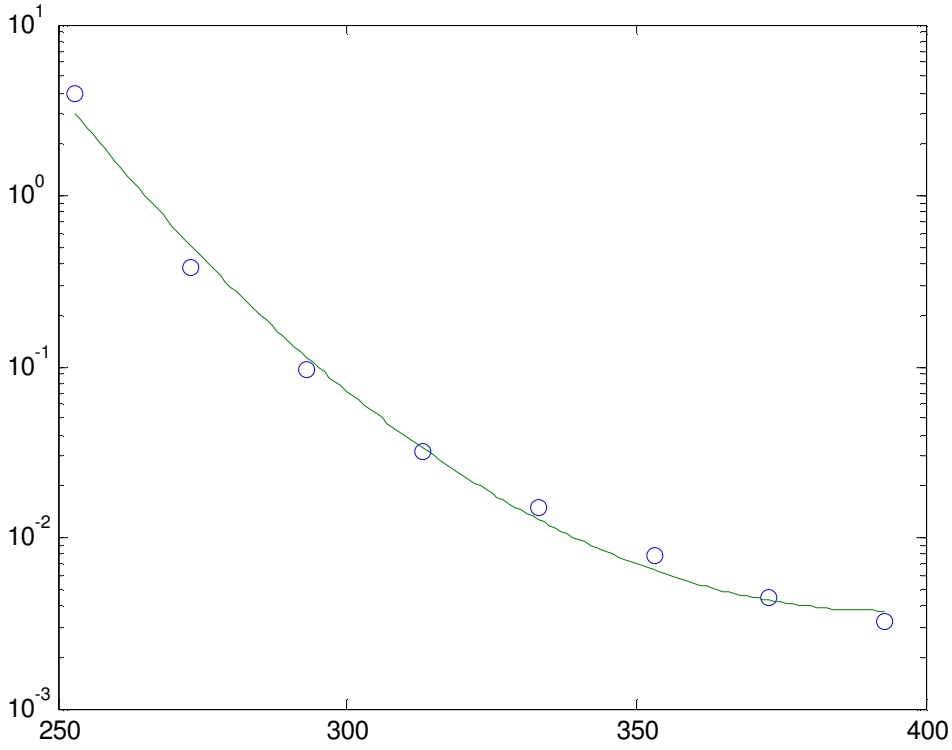
>> viscosidad

p = 0.0003 -0.2685 47.1673

Con estos coeficientes se puede calcular la viscosidad del aceite en función de la temperatura, de la forma:

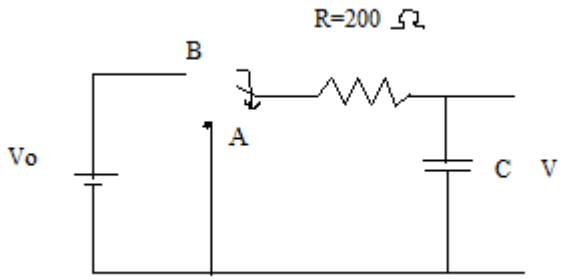
$$\mu = e^{(0.0003T^2 - 0.2685T + 47.1673)} = e^{47.1673} e^{(-0.2685)T} e^{0.0003T^2}$$

El gráfico generado muestra que la ecuación calculada se ajusta bien a los puntos dados en el enunciado



3. Un condensador eléctrico posee una capacidad desconocida. Para calcular su capacidad se conecta a un circuito como el que se muestra en la figura. En este circuito, el conmutador se conecta primera a B, de forma que el condensador se carga. Seguidamente, el conmutador se conecta a A, de forma que el condensador se

descarga a través de la resistencia. Cuando el condensador se está descargando, el valor del voltaje que circula a través de él se mide durante 10 segundos, en intervalos de un segundo. Los valores medidos se muestran en la siguiente tabla.



T(s)	1	2	3	4	5	6	7	8	9	10
V(v)	9.4	7.31	5.15	3.55	2.81	2.04	1.26	0.97	0.74	0.58

Representa gráficamente el voltaje en función del tiempo y calcula la capacidad del condensador ajustando una curva exponencial a los puntos anteriores.

Cuando el condensador se descarga a través de la resistencia, el voltaje del condenasdor en función del tiempo viene dado por:

$$V = V_0 e^{(-t)/(RC)}$$

Donde V_0 es el voltaje inicial, Rel valor de la resistencia y C la capacidad del condensador. Una función exponencial se puede escribir en forma lineal. Si representamos la ecuación exponencial anterior de forma lineal para ln(V) y t, ésta quedará:

$$\ln(V) = \frac{-1}{RC} t + \ln(V_0)$$

Esta ecuación tiene la forma y=mx+b y se puede ajustar a los puntos utilizando la función polyfit (x,y,1)con t como variable independiente x y ln(V) con variable independiente y. los coeficientes m y b, calculados mediante la función polyfit, se puede utilizar para obtener C y V_0 de la forma:

$$C = \frac{-1}{Rm} t$$

$$V_0 = e^b$$

En el siguiente programa, calcula la mejor función de ajuste exponencial para los puntos dados, calculando C y V_0 y dibujando los puntos y la función de ajuste.

>>edit viscosidad.m

```
R=2000;
t=1:10;
v=[9.4,7.31,5.15,3.55,2.81,2.04,1.26,0.97,0.74,0.58];
p=polyfit(t,log(v),1);
c=-1/(R*p(1))
v0=exp(p(2))
tplot=0:0.1:10;
vplot=v0*exp(-tplot./(R*C));
plot(t,v,'o',tplot,vplot)
```

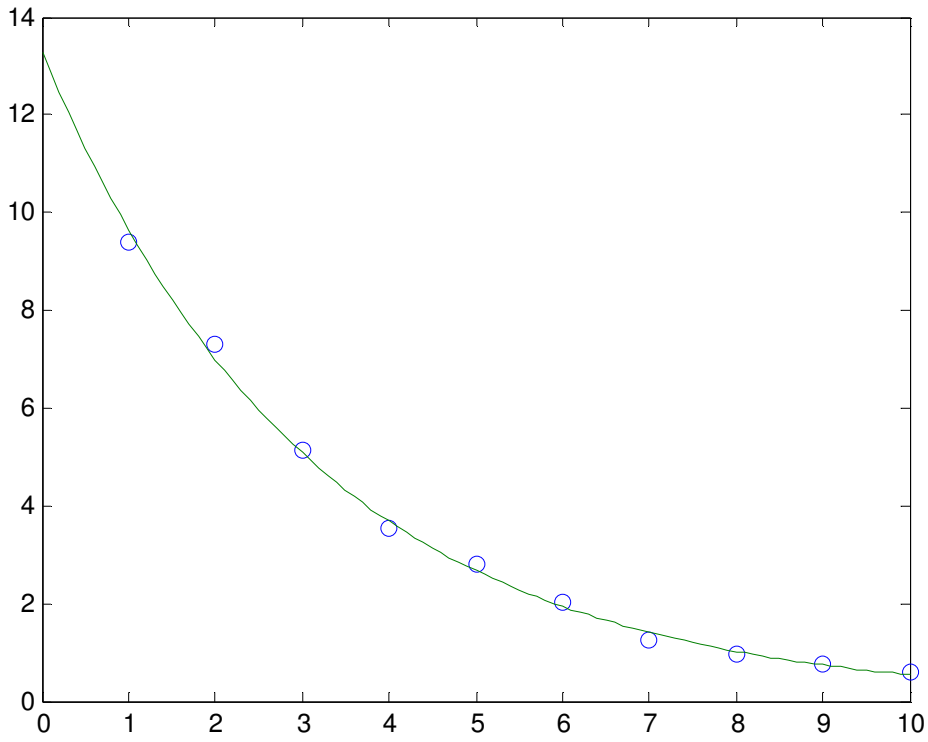
Cuando se ejecuta el fichero, los valores de C y V_0 son:

>> condensado

C=0.0016

V0=13.2796

El grafico obtenido al ejecutar el programa es:

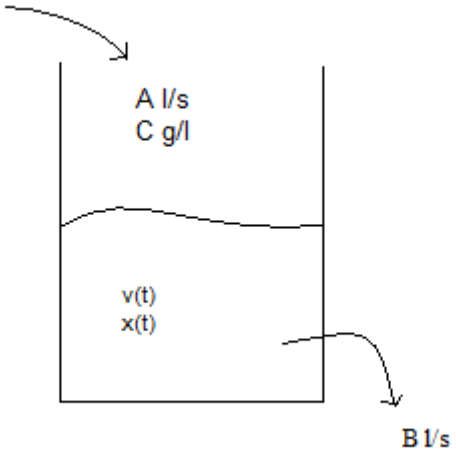


7. TANQUES

Sin duda el estudio de los tanques es de gran importancia en la ciencia e ingeniería ya que con ellos se pude hacer el estudio, del balance de materia y energía, de un proceso industrial o a escala de laboratorio.

Por la gran importancia que hay en la química haremos un estudio de estos mediante varios ejemplos o problemas.

A. Un lago tiene 475 Km^3 y actualmente tiene 0.05% del contamínate. Por encima de 0.02% es contaminado. Se mete al año 175 Km^3 con una contaminación de 0.01%. de ese lago salen todos los años otros 175 Km^3.¿qué tiempo tardara el lago en llegar al 0.02%?



$$\frac{dx}{dt} = AC - \frac{Bx}{V}$$

Sabiendo que x(t) son los Km^3 de contaminante en el tiempo t

Sustituyendo los datos tenemos que la ecuación que debemos resolver es:

$$\frac{dx}{dt} = \frac{175 \cdot 0.01}{100} - \frac{175x}{475}$$

$$x(0) = \frac{0.05 \cdot 475}{100}$$

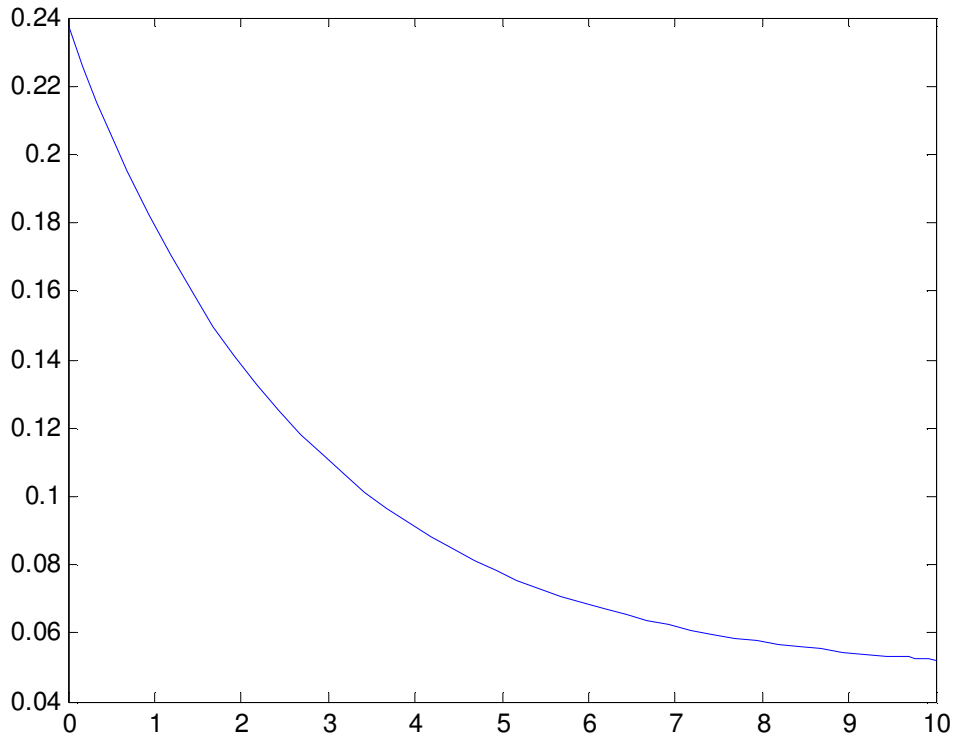
Una vez claro la función, habrá que definirla y representarla para ver cómo cambia la cantidad de contaminante con el tiempo. La función queda definida de la siguiente forma.

>>edit tanque.m

```
function  y=tanque(t,x)

y=(175*0.0001)-(175*x./475);
```

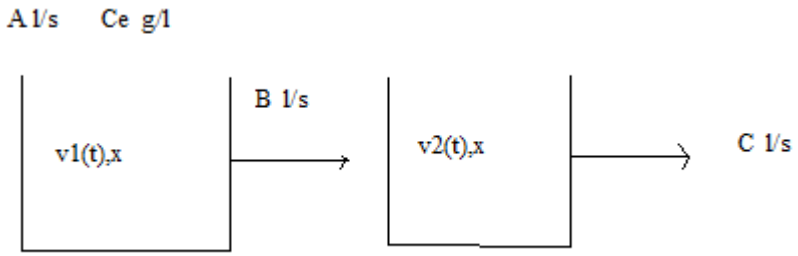
Usando la orden *ode 45* y haciendo una representación gráfica, podemos ver el tiempo (en años) con una determinada concentración.

```
>> [t,x]=ode45('tanque',[0,10],475*0.0005);

>> plot (t,x)
```



En 3 o 4 años la concentración de contamínate es de 0.950.

C. Teniendo el esquema siguiente. hacer la representación grafica del tanque.



Las condiciones iníciales son:

$$A = 7; B = 2; V_1(0) = 100; V_2(0) = 200; C = 5; Ce = 0.3$$

Las ecuaciones que definen este sistema son

$$V_1(t) = V_1(0) + \int_0^t (A(s) - B(s))ds$$
$$V_2(t) = V_2(0) + \int_0^t (B(s) - C(s))ds$$

$$\frac{dx}{dt} = ACe - \frac{Bx}{V_1(t)}$$
$$\frac{dy}{dt} = \frac{Bx}{V_1(t)} - \frac{Cy}{V_2(t)}$$

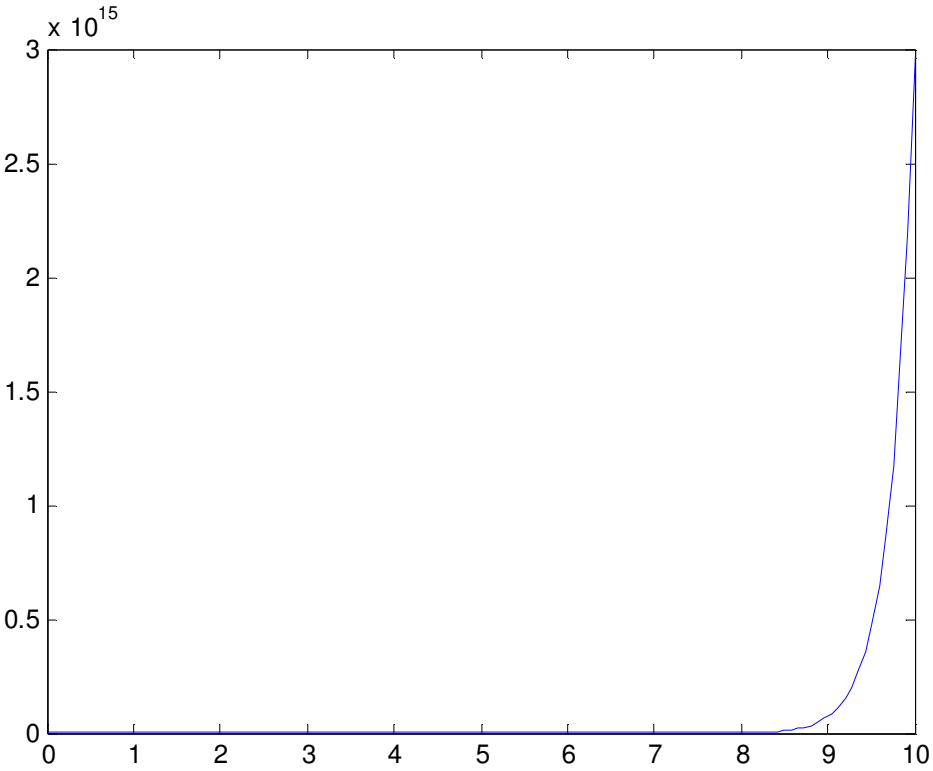
La función queda definida de la siguiente forma:

>>edit fg.m

```
function  z=fg(t,xx)
x=xx(1);y=xx(2);
z(1)=t+x-2*y;
z(2)=t-x+3*y;
z=z';
```

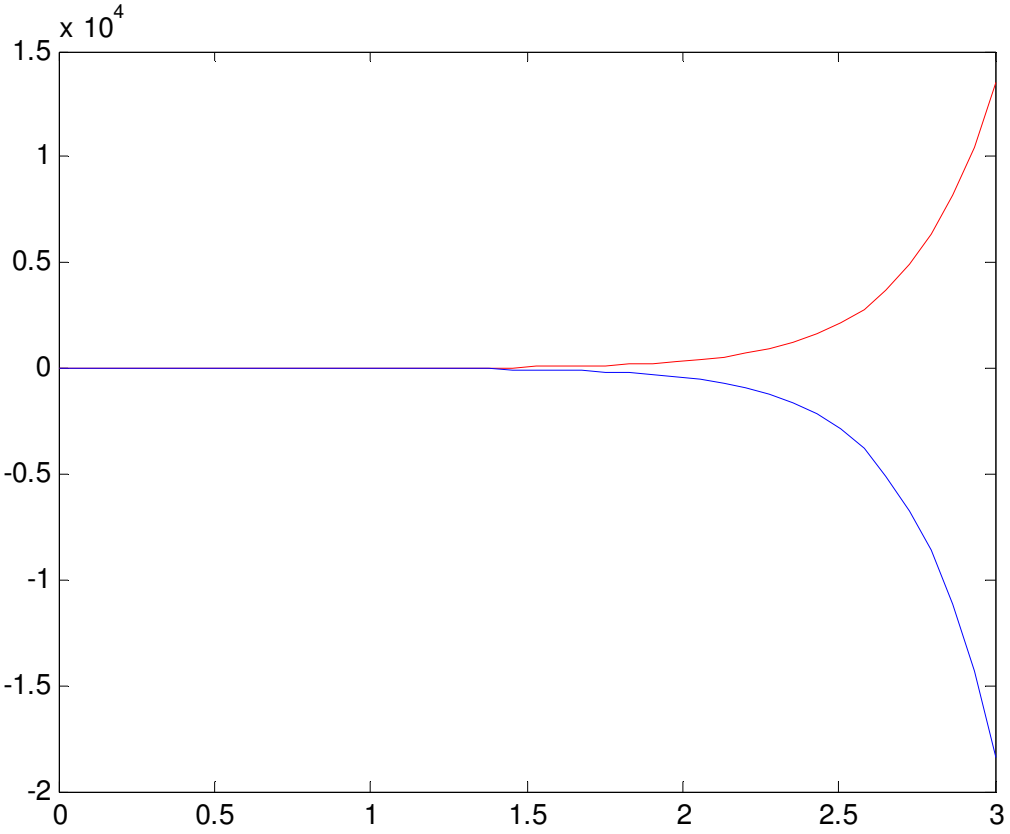
>> [t,xx]=ode45('fg',[0,10],[1,0]);

```
>> plot (t,xx(:,1))
```

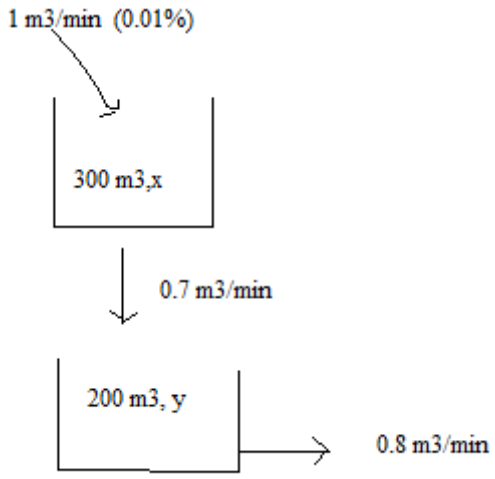


Hacemos una representación entre 0 y3

```
>> [t,xx]=ode45('fg',[0,3],[1,0]);

>> plot (t,xx(:,1),'r',t,xx(:,2),'b')
```



B.



Las condiciones iniciales son: $x_0 = 0.1\%$
$y_0 = 0.1\%$

Las ecuaciones son:

$$\left.\begin{aligned}\frac{dx}{dt} &= 1Ce - \frac{0.7x}{V_1(t)} = 0.0001 - \frac{0.7x}{300+0.3t} \\ \frac{dy}{dt} &= \frac{0.7x}{V_1(t)} - \frac{0.8y}{V_2(t)} = \frac{0.7x}{300+0.3t} - \frac{0.8y}{200-0.1t}\end{aligned}\right\}$$

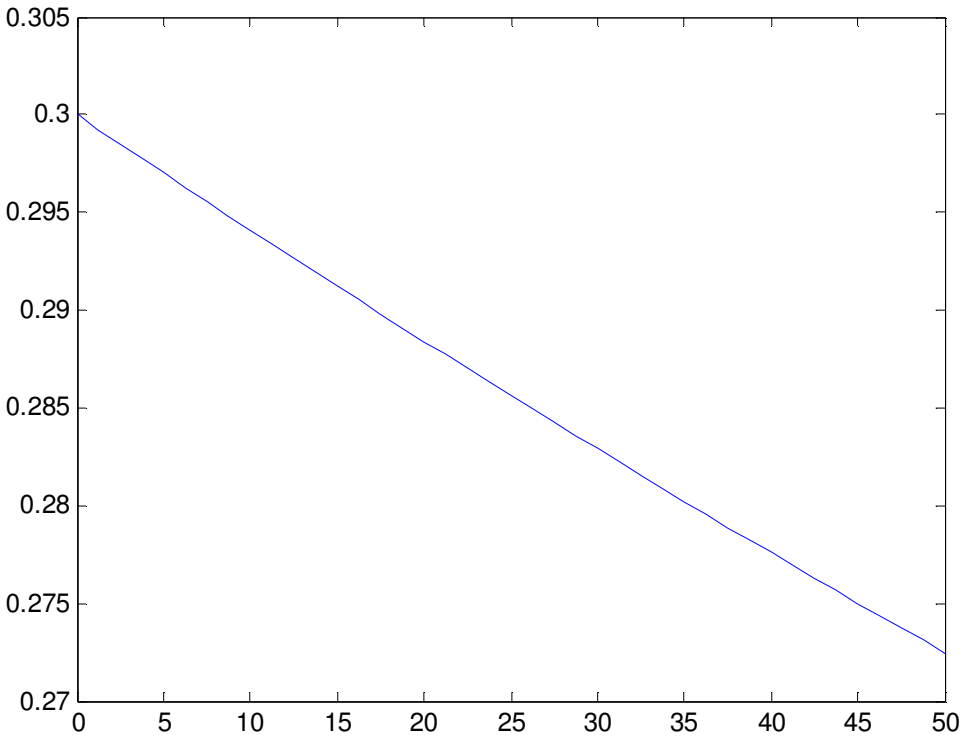
En MATLAB definimos estas funciones de la forma siguiente:

>>edit fg.m

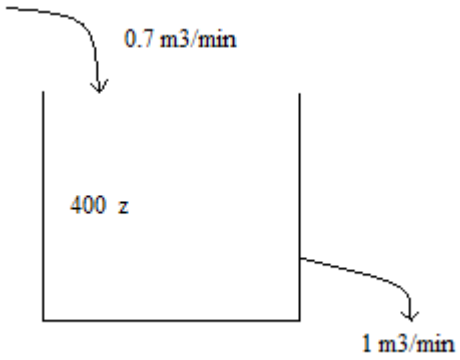
```
function  z=fg(t,xx)

x=xx(1);y=xx(2);

z(1)=0.0001-((0.7*x)/(300+0.3*t));

z(2)=((0.7*x)/(300+0.3*t))-((0.8*y)/(200-0.1*t));

z=z';
```

>> [t,xx]=ode45('fg',[0,50],[0.3,0.2]);

>> plot (t,xx(:,1))



Si ahora al ejerció anterior le añadimos otro tanque con las siguientes condiciones:



Las condiciones iniciales ahora son las siguientes:

$$x(0)=\frac{300\cdot 0.1}{100}=0.3$$
$$y(0)=\frac{200\cdot 0.1}{100}=0.2$$
$$z(0)=\frac{0.7\cdot 400}{100}=2.8$$

La ecuación del tanque 3 será ahora

$$V_3(t)=400-0.3t$$

Las funciones quedan definidas de la forma

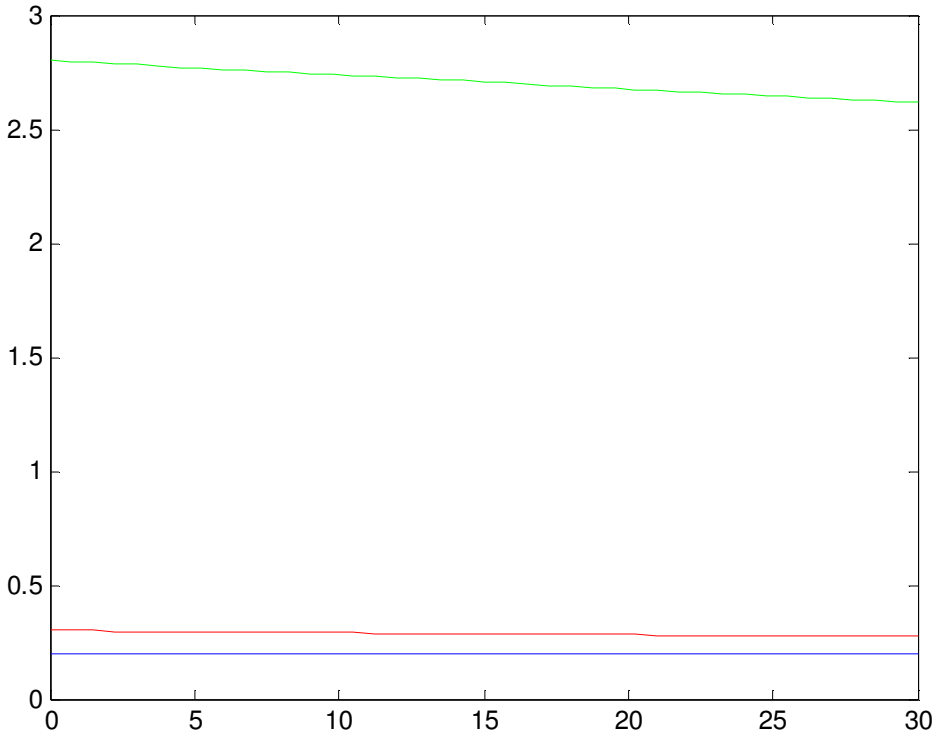
>>edit fg.m

```
function  w=fg(t,xx)
x=xx(1);y=xx(2);y=xx(3);
w(1)=0.0001-((0.7*x)/(300+0.3*t));
w(2)=((0.7*x)/(300+0.3*t))-((0.7*y)/(200-0.1*t));
w(3)=((0.7*y)/(200-0.1*t))-(z/(400-0.3*t));
w=w';
```

Siguiendo el mismo procedimiento que en los problemas anteriores tenemos

>> [t,w]=ode45('fg',[0,30],[0.3;0.2;2.8]);

>> plot (t,xx(:,1),'r',t,w(:,2),'b',t,w(:,3),'g')



8. REACIONES.

Las reacciones bioquímicas están continuamente producidas en cualquier organismo vivo. La mayoría de ellas comprenden la acción de determinadas proteínas, también llamadas enzimas sobre ciertos compuestos, o sustratos. Por ejemplo, la hemoglobina en la serie roja sanguínea es una enzima y el oxigeno, con el que se combina, sería el sustrato. Las enzimas son muy importantes en la regulación de los procesos biológicos, como activadores o inhibidores de una reacción. Para estudiar su comportamiento nos centramos en el estudio de la llamada cinética de reacciones que se centra fundamentalmente en el análisis de

las cantidades de cada uno de los reactivos, el tiempo en que se produce la reacción y las condiciones que la influencia.

1. UN MODELO BÁSICO.

Una de las reacciones enzimáticas más simples fue propuesta por Michaelis yMenten (1913). Estas reacciones constan de una enzima E actuando sobre un sustrato S con el que se combina para formar un compuesto "complejo" SE que, a su vez, acaba transformándose en un nuevo producto P y liberando de nuevo la enzima. Todo este proceso se representa de forma esquemática con las reacciones

$$E + S \underset{k_{-1}}{\overset{k_1}{\rightleftarrows}} ES \xrightarrow{k_2} E + P$$

Aquí k_1, k_{-1} y k_2 son constantes propias de cada reacción. La doble flecha indica que la reacción es reversible mientras que la flecha simple expresa que sólo actúa en el sentido señalado.

La ley de acción de masas afirma que la velocidad de una reacción es proporcional al producto de las concentraciones de las sustancias que reaccionan. Sean pues s, e, c y p las concentraciones de los reactantes S, E, SE y P respectivamente.

$$s = [S] \quad e=[E] \quad c=[SE] \quad p=[P]$$

Dicha ley nos proporciona una ecuación para cada una de las sustancias que están involucradas en la reacción.

$$\begin{cases} \dfrac{ds}{dt} = -k_1 es + k_{-1} c \\ \dfrac{de}{dt} = -k_1 es + (k_{-1} + k_2) c \\ \dfrac{dc}{dt} = k_1 es - (k_{-1} + k_2) c \\ \dfrac{dp}{dt} = k_2 c \end{cases}$$

Para terminar la formulación matemática de este problema, necesitamos ofrecer condiciones iníciales apropiadas para cada una de las variables en cuestión. En este caso, resulta lógico suponer conocidas las concentraciones iníciales de enzima y sustrato y nula la del producto final.

$$s(0) = s_0 \quad \text{e(0)=e}_0 \quad \text{c(0)=c}_0 \quad p(0) = 0$$

Naturalmente sólo tiene sentido considerar valores positivos para estas concentraciones iniciales.

Suponiendo un ejemplo en el las condiciones iniciales son:

$$k_1 = 2 \qquad \text{s(0)=1.5}$$
$$k_{-1} = 1 \qquad \text{e(0)=2.7}$$
$$k_3 = 3 \qquad \text{c(0)=p(0)=0}$$

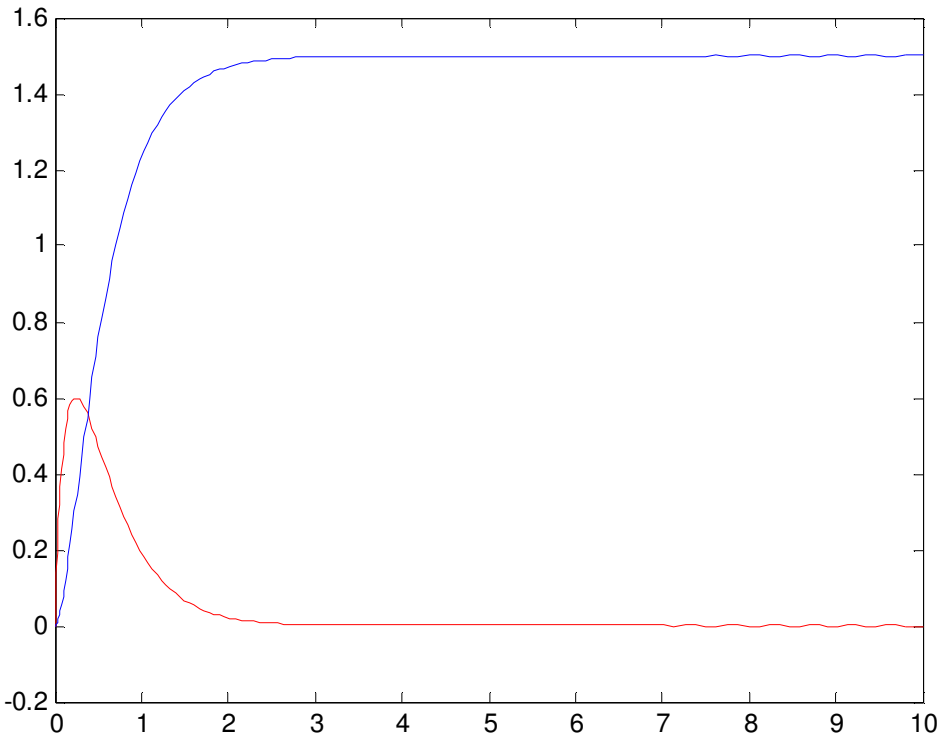
Y como las ecuaciones se encuentran definidas anteriormente, hacemos uso del MATLAB para ver gráficamente la evolución de la reacción con el tiempo. Para ello creamos un fichero que define a la ecuación que estamos estudiando.

>>edit cinetica.m

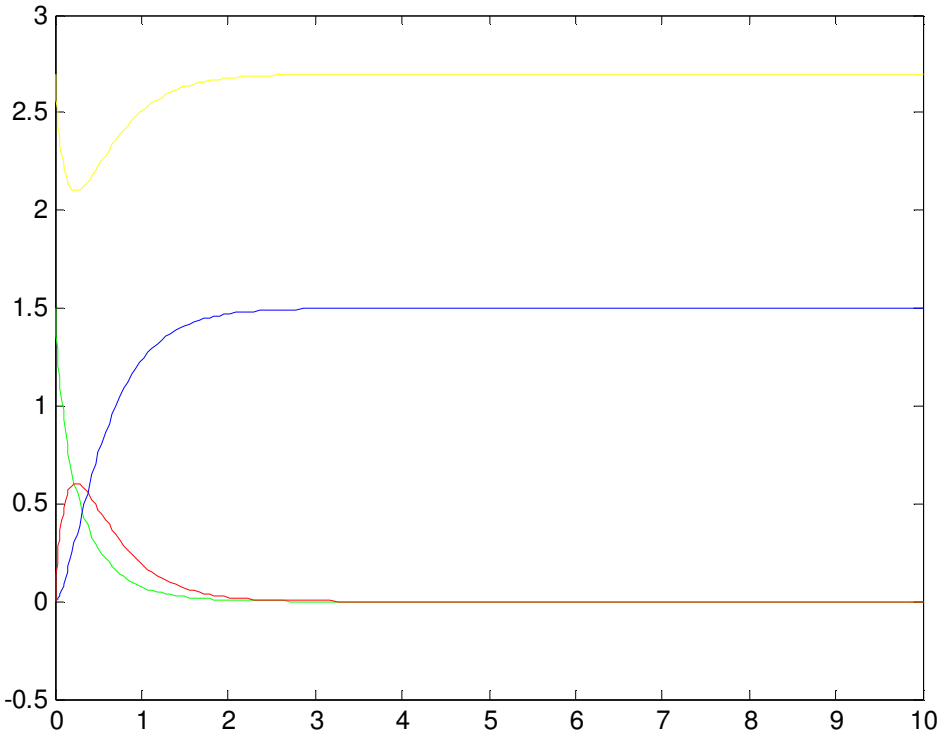
```
function  z=cinetica(z,yy)

s=yy(1); e=yy(2); c=yy(3); p=yy(4);

z(1)=(-2.*e.*s)+c;

z(2)=(-2.*e.*s)+(4.*c);

z(3)=(2.*e.*s)-(4.*c);

z(4)=3.*c;

z=z';
```

Aplicando la orden *ode 45* y posteriormente el comando *plot* obtenemos la representación grafica de las 4 ecuaciones descritas.

```
>> [t,y]=ode45('cinetica', [0,10],[1.5,2.7,0,0]);

>> plot(t,y(:,3),'r',t,y(:,4),'b')
```



```
>> plot(t,y(:,1),'g',t,y(:,2),'y',t,y(:,3),'r',t,y(:,4),'b')
```



2. AUTOCATÁLISIS: ACTIVACIÓN E INHIBICIÓN

El ejemplo anterior modela y resuelve un determinado sistema de ecuaciones diferenciales no lineal que describe la variación de las concentraciones de reactivos y productos para una reacción bastante sencilla. Veamos ahora como todos los fenómenos de interacción que aparecían al observar el comportamiento de especies que conviven en el mismo hábitat pueden reproducirse en determinadas reacciones químicas.

Muchos procesos biológicos o químicos conllevan ciertos mecanismos de retroalimentación, esto es, se trata de situaciones en las que el producto de una etapa de la secuencia de reacciones tiene efectos en etapas

posteriores. El efecto, normalmente, es de tipo no lineal y puede activar o inhibir este proceso.

Es fácil pensar en circunstancias que involucren un mecanismo de este tipo: la división celular sería un buen ejemplo de ello. Además, está muy bien documentado, en experiencias realizadas sobre ciertas bacterias, el hecho de que en una gran cantidad de cultivos celulares, algunas de las enzimas implicadas aumentan periódicamente su actividad durante la división y esto provoca cambios en la tasa de síntesis enzimática. Estas situaciones requieren algún tipo de control de retro (o auto) alimentación. Se han propuesto varios modelos que son capaces de describir dicho control. Uno de ellos sugiere que ciertos metabolitos reducen la producción de enzima que se necesitan para su propia fabricación. Para ello se inhibe la transcripción de la molécula de ADN a ARN mensajero, que es la "plantilla" que fabrica la enzima.

Pretendemos aquí estudiar los modelos relativos a sistemas que experimentan algún tipo de mecanismos de *feedback*. Quizás el modelo de autocatálisis más simple sea representado por la ecuación

$$A + X \rightleftharpoons 2X$$

Siendo x la concentración de la sustancia X. Obsérvese que la ecuación es el equivalente químico a la ecuación logística. El origen es un punto de equilibrio inestable y x(t) tiene cuando t→∞ hacia el valor de equilibrio estable $x_e = \frac{k_1 a}{k_{-1}}$. Esta reacción autocatalitica exhibe un fuerte proceso de retroalimentación con el producto inhibiendo la velocidad de la reacción ya que, claro está, es necesaria algún tipo de reversibilidad de la reacción,

$k_{-1} \neq 0$. En otro caso, $k_{-1} = 0$, la variación relativa de la concentración de X crecería indefinidamente.

Como en el caso anterior vamos a hacer una demostración de un problema autocatalitico simple, como el que hemos explicado anteriormente. Para ello debemos crear también un fichero que defina las ecuaciones expuestas en la explicación , además saber las condicione iniciales del problema que son:

$$k_1 = 1 \qquad A=3$$
$$k_{-1} = 2 \qquad x_0=0$$

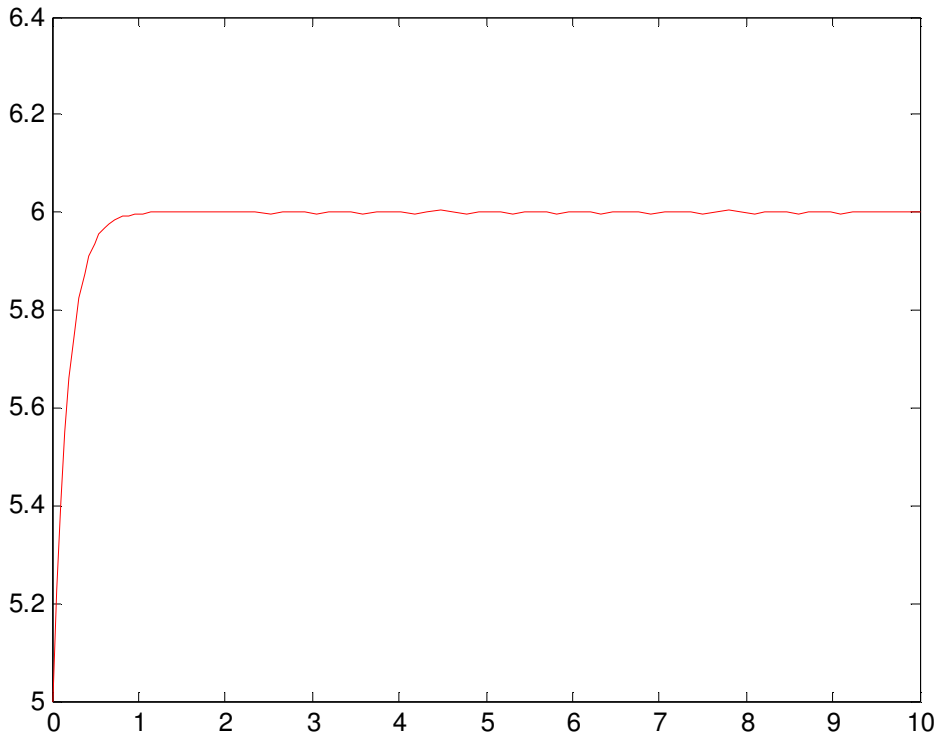
Ahora creamos el fichero:

>>edit autoc.m

```
function  z=autocat(t,x);

z=6.*x-x.^2;
```

Como podemos ver esta ecuación es mucho más simple que la del ejemplo visto con anterioridad. Pero, como es de esperar, para realizar la representación grafica debemos hacer usa de la orden *ode 45* y del comando *plot.*

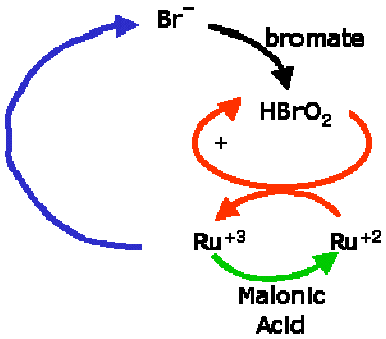
```
>> [t,x]=ode45('autocat',[0,10],[5])

>> plot(t,x,'r')
```



3. REACCIÓN DE BELOSOV-ZHABOTINSKY.

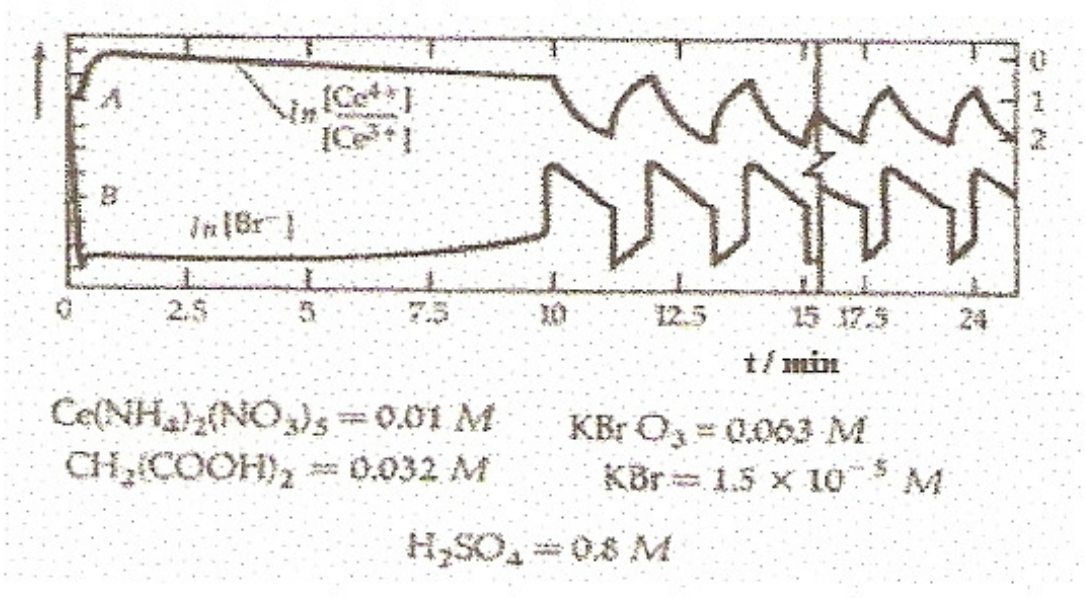
Un ejemplo paradigmático de una reacción química oscilante es la de Belousov-Zhabotinsky (BZ), la cual ha sido extensamente estudiada tanto experimental como teóricamente. La reacción, inicialmente investigada por B. Belousov (1950) como caricatura del ciclo de Krebs, consiste básicamente en la oxidación de ácido malónico por iones bromatos, en presencia de cerio como catalizador, en medio de acido sulfúrico.. Belousov descubrió que la mezcla se tornaba amarilla, luego perdía el color, para retornar pasado un minuto, a tomar la misma coloración amarilla que la comienza. Las oscilaciones periódicas espontaneas se mantenían aproximadamente durante el lapso de una hora, antes de alcanzar finalmente un estado de equilibrio. Cuando Boris P. Belousov, director del Instituto de Biofísica en la Unión Soviética, presentó un

documento a una revista científica que supuestamente han descubierto una reacción química oscilante en 1951, se rechazó rotundamente crítico con una nota del editor que es claramente imposible. La confianza en su imposibilidad era tal que a pesar de que el documento iba acompañado por el procedimiento relativamente sencillo para llevar a cabo la reacción, no podía ser posible. Hasta que el desarrollo de la termodinámica de los irreversibles hizo posible comprender los mecanismos por los cuales una reacción química no necesariamente evoluciona en forma monótona hacia el equilibrio, el hallazgo de Belousov fue tan sorprendente, que en su momento se adjudicó a algún error en sus experimentos. Nadie se animó a publicar su trabajo. La confirmación posterior por parte de Zhabotinsky, junto con el interés que comenzaba a evidenciarse hacia el comienzo de la década de los 70 por las oscilaciones bioquímicas, convirtieron al experimento de BZ en el prototipo de la reacción oscilante, a partir de la cual la analogía entre el comportamiento de sistemas químicos artificiales y el de reacciones comunes dentro de la bioquímica celular se hizo evidente.

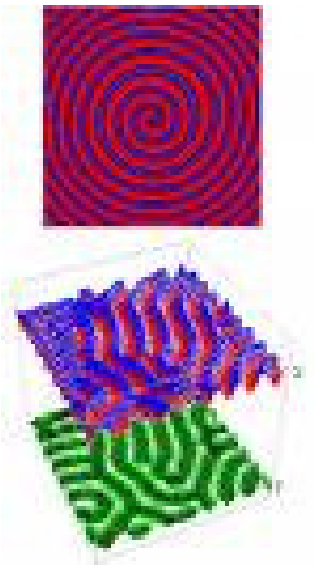


El curso de la reacción puede seguirse mediante medidas potenciométricas o mediante los cambios de color que se producen si se añade un indicador redox; por ejemplo ferroína (o-fenantrolina ferrosa) el

medio oscila entre color rojo (forma reducida) a color azul (forma oxidada). Para que en la disolución se puedan producir oscilaciones sostenidas es necesario mantener el sistema en agitación continua, con alimentación de reactivos y retirada de productos.



Cuando el proceso se lleva a cabo en una placa de petri, en la que se extiende una delgada capa de disolución, en ausencia de agitación, el acoplamiento de la reacción oscilante con los procesos de difusión de reactivos y productos da lugar a estructuras especiales como las que se muestran en la figura, donde la alternancia de zonas de oxidación (azul) y de reducción (rojo) se suceden en forma de bandas circulares, en forma de ondas. Estas ondas químicas difieren de alguna forma de las ondas hidrodinámicas que se producen en la superficie de los líquidos.



Si el proceso tiene lugar con todos los reactivos mezclados en un tubo de ensayo, excepto el ácido sulfúrico, el cual se añade en forma de unas gotas en la parte superior, el sistema se presenta inicialmente como una masa de precipitado que va evolucionando formando franjas de diferente color a medida que el acido difunde a través del precipitado y dispara el proceso.

En 1970, Richard M. Noyes, Richard J. Field, y Endre Koros en la Universidad de Oregon, propusieron un sólido mecanismo compuesto de dieciocho reacciones y veintiún especies químicas diferentes, lo que resulta en un impresionante e intimidatoria sistema de tipo de ecuaciones (ecuaciones diferenciales que describen el tipo de reacciones).

Veamos un ejemplo sencillo de esta situación. Trabajaremos con las reacciones:

$$A + X \xrightarrow{k_1} 2X$$
$$X + Y \xrightarrow{k_2} 2Y$$
$$Y \xrightarrow{k_3} B$$

Donde la concentración de A se mantiene constante. Las dos primeras reacciones son autocatalíticas así que, aplicando la ley de acción de masas, planteamos el sistema de ecuaciones diferenciales

$$\frac{dx}{dt} = k_1 ax - k_2 xn$$
$$\frac{dn}{dt} = k_2 xn - k_3 n$$
$$\frac{db}{dt} = k_3 n$$

Donde x, n, b representan las concentraciones de X, Y, B, respectivamente.

Si lo que pretendemos es resolver matemáticamente este sistema de ecuaciones sabiendo que las condiciones iníciales son:

$$k_1 = 1$$
$$k_2 = 2$$
$$k_3 = 3$$
$$x(0) = 1.5$$
$$n(0) = 2.7$$
$$b(0) = 0$$

La concentración de A toma el valor de 4.

Es posible resolver este sistema con un programa matemático que en nuestro caso usaremos el Matlab. Para ello es necesario crear un programa que recoja todas las ecuaciones descritas anteriormente:

>> edit oscilaciones.m

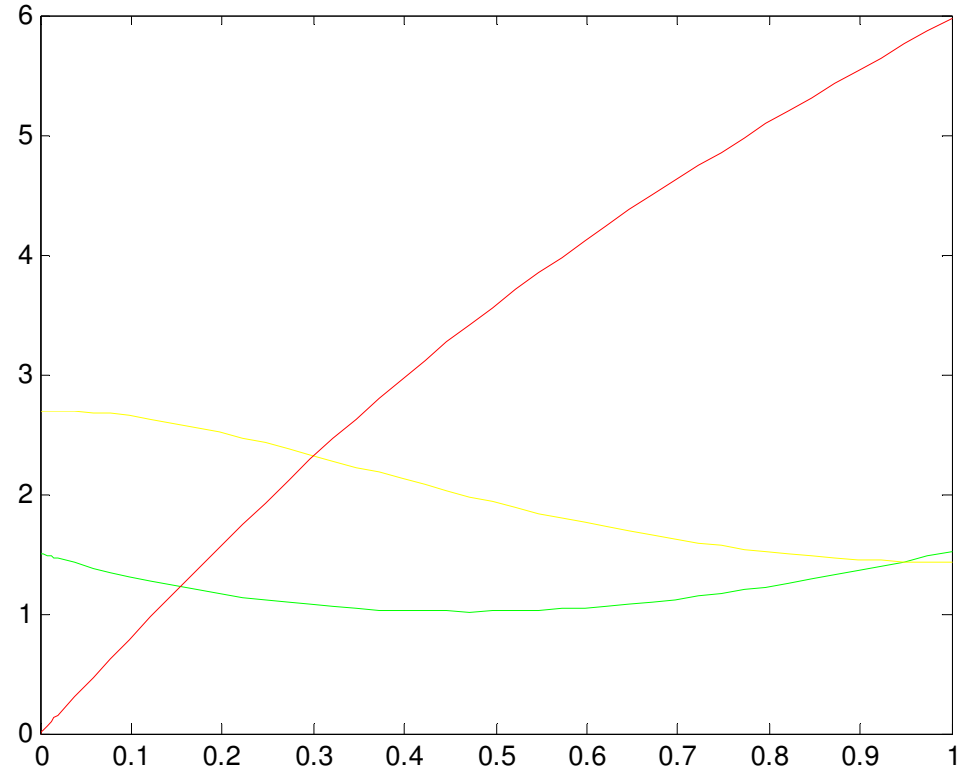
```
function z=oscilaciones(t,yy)

x=yy(1); n=yy(2); b=yy(3);

z(1)=4*x-2*x.*n;

z(2)=2*x.*n-3*n;

z(3)=3*n;

z=z';
```

Usamos ahora la orden *ode45* para resolver el sistema de ecuaciones:

>> [t,y]=ode45('oscilaciones',[0,1],[1.5,2.7,0]);

Si ahora se lleva a cabo una representación gracia obtenemos:

>> plot(t,y(:,1),'g',t,y(:,2),'y',t,y(:,3),'r')



4. REACCIÓN DE BRUSELATOR.

Este "*Bruselator*" es un famoso modelo, debido a los trabajos pioneros de *Prigogine* y colaboradores. Aunque ha sido muy criticado por incluir una etapa trimolecular (responsable del monomio □ X2Y), puede soslayarse tal dificultad introduciendo una tercera variable que se elimina adiabáticamente. En todo caso sus ecuaciones diferenciales son:

$$\dot{X} = k_1 - k_2 X - k_3 X + k_4 X^2 Y$$
$$\dot{Y} = k_5 X - k_6 X^2 Y$$

Tiene estado estacionario en

$$X_0 = \frac{k_1}{k_2 + k_3 - \frac{k_4 k_5}{k_6}}, Y_0 = \frac{k_5}{k_6 k_1}(k_2 + k_3 - \frac{k_4 k_5}{k_6})$$

Se pueden amoldar a estas ecuaciones polimerizaciones [0,1] y [0,2].

En los estudios realizados del *Brusselator* se suele suponer, por sencillez los valores, k1 = A, k2 = k5 =B y k3 = k4 = k6 =1 (en unidades adecuadas), con lo cual la ecuación anterior toma la forma:

$$\dot{X} = A - BX - X + X^2 Y$$
$$\dot{Y} = BX - X^2 Y$$

y así X0 = A, Y0 = B/A; la matriz secular es

$$M_0 = \begin{pmatrix} c = B-1 & g = A^2 \\ a = -B & b = -A^2 \end{pmatrix} = \begin{pmatrix} B-1 & A^2 \\ -B & -A^2 \end{pmatrix}$$

con T0 = B – 1 - A2, y Det0 = A2 > 0. Para B > 1 + A2 , se tiene T0 > 0, y hay un ciclo límite rodeando al estado estacionario inestable.

Teniendo en cuenta la difusión, el sistema anterior se escribe

$$\frac{\partial X}{\partial t} = A - BX - X + X^2 Y + D_x \frac{\partial^2 X}{\partial r^2}$$
$$\frac{\partial Y}{\partial t} = BX - X^2 Y + D_y \frac{\partial^2 Y}{\partial r^2}$$

y la matriz es ahora

$$M_n = \begin{pmatrix} B - 1 - D_x n^2 & A^2 \\ -B & -A^2 - D_y n^2 \end{pmatrix}$$

cuya Tn = B – 1 - A2 - (Dx + Dy)n2, y cuyo determinante es:

$$Det_n = A^2 - n^2[(B - 1)D_y - A^2 D_x] + D_x D_y n^4$$

Se puede tener inestabilidad del estado estacionario homogéneo, desde luego, si Tn >

0, o sea si B > B1(n) = 1 + A2 + (Dx + Dy)n2. El valor crítico B1 depende de n; su valor mínimo se alcanza (dB1/dn = 0) cuando n = 0 (fluctuación homogénea), y vale B1min = 1 + A2, como en el caso no difusivo, confirmando en efecto que la difusión no desestabiliza un estado estacionario homogéneo que sea estable por causa de T0 < 0. También puede haber estado estacionario homogéneo inestable si Detn < 0, lo que implica que

$$B > B_2(n) = 1 + A^2 \frac{D_x}{D_y} + D_x n^2 + \frac{A^2}{D_y n^2}$$

El mínimo valor del límite crítico B2 se obtiene (dB2/dn = 0) para

$$n^2 = \sqrt{\frac{A^2}{D_x D_y}} \text{ y vale } B_{2\min} = \left(1 + A\sqrt{\frac{D_x}{D_y}}\right)^2$$

Es obvio que si Dx = Dy, será B1min = 1 + A2 < B2min = (1 + A)2, lo que significa que al ir aumentando B se alcanza antes la bifurcación para B1min, en la cual la perturbación homogénea es la más rápida, y no se verán estructuras espaciales inhomogéneas (sí se verá el ciclo límite, porque B2 > B > B1 implica T0 > 0). Pero si Dy >> Dx , entonces puede ocurrir que B2min < B1min (en el caso extremo (Dx/Dy) → 0, B2min= 1 < B1min= 1 + A2), y al aumentar B acaece antes la bifurcación en B2min, y aparece (orden a través de fluctuaciones) una estructura disipativa correspondiente a n2 = (A2/DxDy)1/2.

Siguiendo con el mismo esquema que en las reacciones anteriores, debemos crear un fichero que defina las ecuaciones de la reacción. Estas son:

$$\frac{dx}{dt} = 1 - (B+1)x + Ax^2 y$$
$$\frac{dy}{dt} = Bx - Ax^2 y$$

Siendo las constantes A y B:

$$A = \frac{k_3}{k_4}$$
$$B = \frac{k_2 b}{k_1 a} = \frac{k_2 b}{k_4}$$

Una vez definidas las ecuaciones podemos crear el fichero

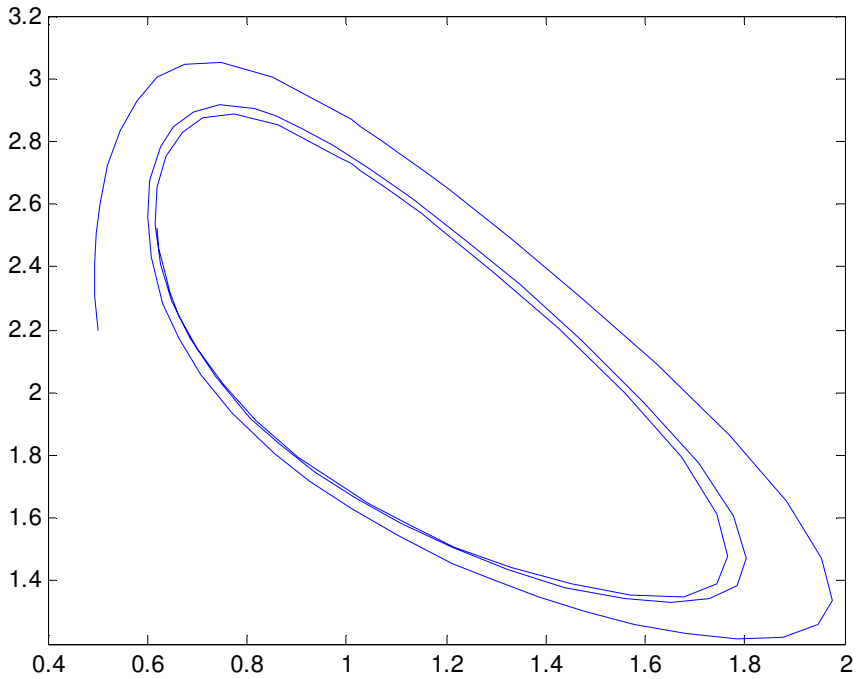
>>edit bruselator.m

```
function  z=bruselator(t,yy)

X=yy(1);Y=yy(2);

B=2.2;A=1;

z(1)=1-(B+1).*X+(A.*X.^2.*Y);

z(2)=B.*X-(A.*X.^2.*Y);

z=z';
```

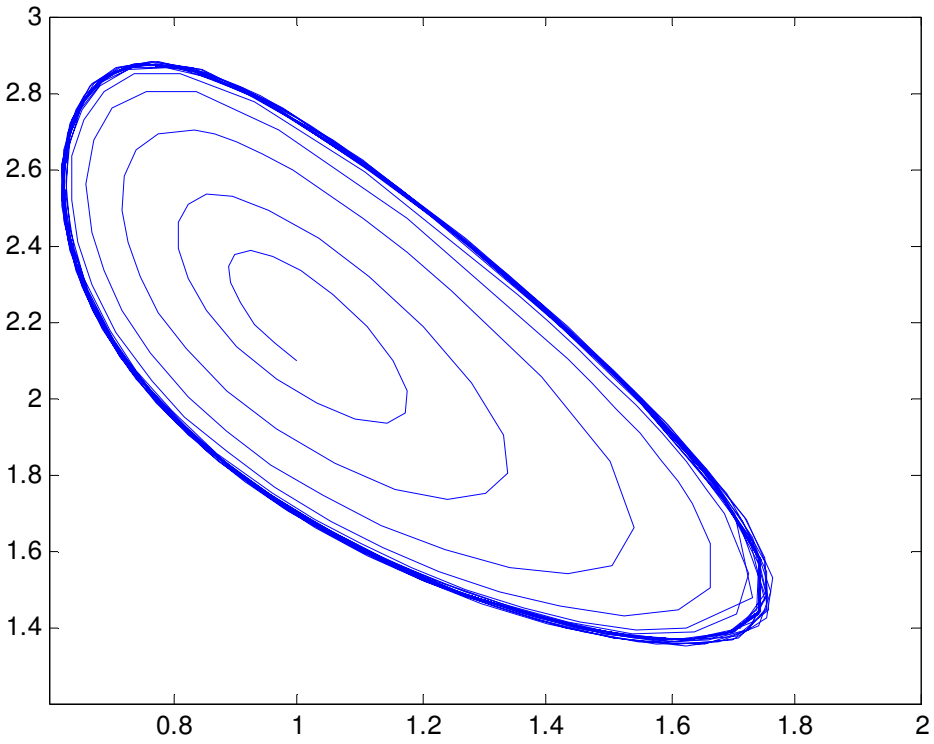
Ya podemos aplicar la ecuación para obtener la representación grafica del mismo

>> [t,y]=ode45('bruselator',[0,20],[0.5,2.2]);

>> plot(y(:,1),y(:,2))



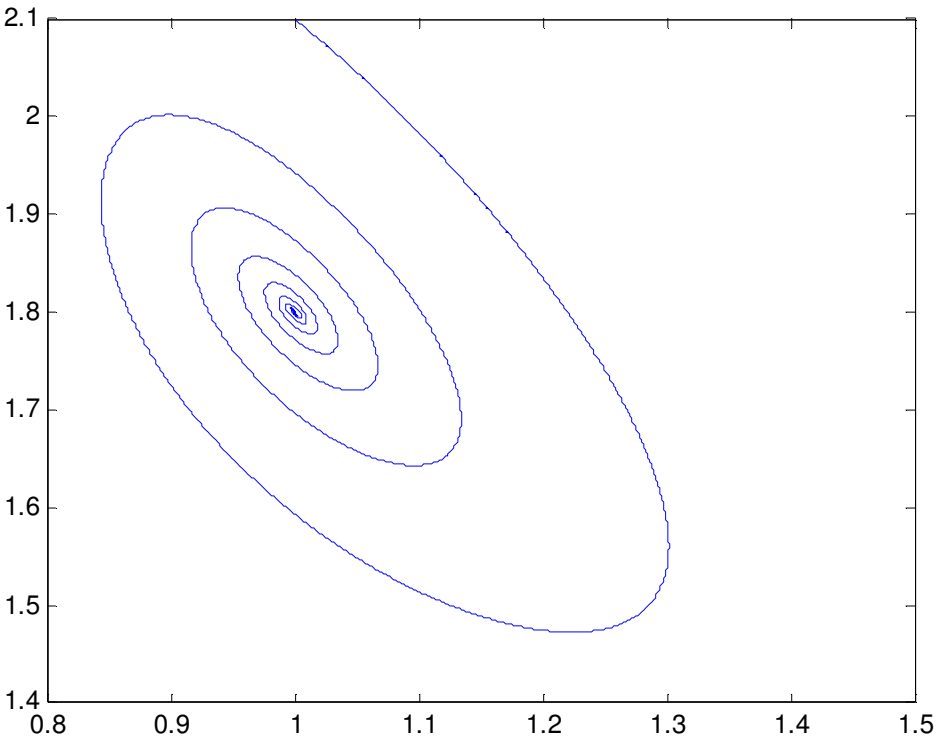
```
>> [t,yy]=ode45('bruselator',[0,1000],[1;2.1]);

>> plot(yy(:,1),yy(:,2))
```



Como se puede observar el ciclo límite es estable y el equilibrio es inestable, ya que lo que vemos es un movimiento oscilatorio.

El ciclo límite de la reacción se puede obtener también de forma grafica de la forma siguiente

```
>>options=odeset('absTol',10^(-14),'relTol',10^(-14));

>> [t,yy]=ode45('bruselator',[0,1000],[1;2.1],options);

>> plot(yy(:,1),yy(:,2))
```


2.1
2
1.9
1.8
1.7
1.6
1.5
1.4
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5