1.-El programa MATLAB

MATLAB es el nombre abreviado de "MATrix LABoratory". MATLAB es un programa para realizar cálculos numéricos con ***vectores*** y ***matrices***. Como caso particular puede también trabajar con números escalares –tanto reales como complejos–, con cadenas de caracteres y con otras estructuras de información más complejas. Una de las capacidades más atractivas es la de realizar una amplia variedad de ***gráficos*** en dos y tres dimensiones. MATLAB tiene también un lenguaje de programación propio.

MATLAB es un gran programa de cálculo técnico y científico. Para ciertas operaciones es muy rápido, cuando puede ejecutar sus funciones en código nativo con los tamaños más adecuados para aprovechar sus capacidades de vectorización. En otras aplicaciones resulta bastante más lento que el código equivalente desarrollado en C/C++ o Fortran. En la versión 6.5, MATLAB incorporó un ***acelerador JIT*** (Just In Time), que mejoraba significativamente la velocidad de ejecución de los ficheros ****.m*** en ciertas circunstancias, por ejemplo cuando no se hacen llamadas a otros ficheros ****.m***, no se utilizan estructuras y clases, etc. Aunque limitado en ese momento, cuando era aplicable mejoraba sensiblemente la velocidad, haciendo innecesarias ciertas técnicas utilizadas en versiones anteriores como la ***vectorización*** de los algoritmos. En cualquier caso, el lenguaje de programación de MATLAB siempre es una magnífica herramienta de alto nivel para desarrollar aplicaciones técnicas, fácil de utilizar y que, como ya se ha dicho, aumenta significativamente la productividad de los programadores respecto a otros entornos de desarrollo.

MATLAB dispone de un código básico y de varias librerías especializadas (***toolboxes***). En estos apuntes se hará referencia exclusiva al código básico.

MATLAB se puede arrancar como cualquier otra aplicación de ***Windows***, clicando dos veces en el icono correspondiente en el escritorio o por medio del menú ***Inicio***). Al arrancar MATLAB se abre una ventana. Ésta es la vista que se obtiene eligiendo la opción ***Desktop Layout/Default***, en el menú ***View***. Como esta configuración puede ser cambiada fácilmente por el usuario, es posible que en muchos casos concretos lo que aparezca sea muy diferente. En cualquier caso, una vista similar se puede conseguir con el citado comando ***View/Desktop Layout/ Default***. Esta ventana inicial requiere unas primeras explicaciones.

En la parte superior izquierda de la pantalla aparecen dos ventanas también muy útiles: en la parte superior aparece la ventana ***Current Directory***, que se puede alternar con

Workspace clicando en la pestaña correspondiente. La ventana ***Current Directory*** muestra los ficheros del directorio activo o actual. El directorio activo se puede cambiar desde la ***Command Window***, o desde la propia ventana (o desde la barra de herramientas, debajo de la barra de menús) con los métodos de navegación de directorios propios de ***Windows***. Clicando dos veces sobre alguno de los ficheros ****.m*** del directorio activo se abre el ***editor de ficheros*** de MATLAB, herramienta fundamental para la programación sobre la que se volverá en las próximas páginas. El ***Workspace*** contiene información sobre todas las variables que se hayan definido en esta sesión y permite ver y modificar las matrices con las que se esté trabajando.

En la parte inferior derecha aparece la ventana ***Command History*** que muestra los últimos comandos ejecutados en la ***Command Window***. Estos comandos se pueden volver a ejecutar haciendo doble clic sobre ellos. Clicando sobre un comando con el botón derecho del ratón se muestra un menú contextual con las posibilidades disponibles en ese momento. Para editar uno de estos comandos hay que copiarlo antes a la ***Command Window***.

En la parte inferior izquierda de la pantalla aparece el botón ***Start***, con una función análoga a la del botón ***Inicio*** de ***Windows***. ***Start*** da acceso inmediato a ciertas capacidades del programa. Las posibilidades son de ***Start/MATLAB*** y de ***Start/Desktop Tools***, que permiten el acceso a las principales componentes o módulos de MATLAB.

El menú ***Desktop*** realiza un papel análogo al botón ***Start***, dando acceso a los módulos o componentes de MATLAB que se tengan instalados. Puede hacerse que al arrancar MATLAB se ejecute automáticamente un fichero, de modo que aparezca por ejemplo un saludo inicial personalizado. Esto se hace mediante un ***fichero de comandos*** que se ejecuta de modo automático cada vez que se entra en el programa (el fichero ***startup.m***, que debe estar en un directorio determinado, por ejemplo ***C:\Matlab701\Work***.

Para apreciar desde el principio la potencia de MATLAB, se puede comenzar por escribir en la ***Command Window*** la siguiente línea, a continuación del *prompt*. Al final hay que pulsar ***intro***.

>> A=rand(6), B=inv(A), B*A

A =

0.9501 0.4565 0.9218 0.4103 0.1389 0.0153

0.2311 0.0185 0.7382 0.8936 0.2028 0.7468

0.6068 0.8214 0.1763 0.0579 0.1987 0.4451

0.4860 0.4447 0.4057 0.3529 0.6038 0.9318

0.8913 0.6154 0.9355 0.8132 0.2722 0.4660

0.7621 0.7919 0.9169 0.0099 0.1988 0.4186

B =

5.7430 2.7510 3.6505 0.1513 -6.2170 -2.4143

-4.4170 -2.5266 -1.4681 -0.5742 5.3399 1.5631

-1.3917 -0.6076 -2.1058 -0.0857 1.5345 1.8561

-1.6896 -0.7576 -0.6076 -0.3681 3.1251 -0.6001

-3.6417 -4.6087 -4.7057 2.5299 6.1284 0.9044

2.7183 3.3088 2.9929 -0.1943 -5.1286 -0.6537

ans =

1.0000 0.0000 0 0.0000 0.0000 -0.0000

0.0000 1.0000 0.0000 0.0000 -0.0000 0.0000

0 0 1.0000 -0.0000 -0.0000 0.0000

0.0000 0 -0.0000 1.0000 -0.0000 0.0000

-0.0000 0.0000 -0.0000 -0.0000 1.0000 0.0000

-0.0000 -0.0000 -0.0000 -0.0000 -0.0000 1.0000

En realidad, en la línea de comandos anterior se han escrito tres instrucciones diferentes, separadas por comas. Como consecuencia, la respuesta del programa tiene tres partes también, cada una de ellas correspondiente a una de las instrucciones. Con la primera instrucción se define una matriz cuadrada (6×6) llamada **A**, cuyos elementos son números aleatorios entre cero y uno (aunque aparezcan sólo 4 cifras, han sido calculados con 16 cifras de precisión). En la segunda instrucción se define una matriz **B** que es igual a la inversa de **A**. Finalmente se ha multiplicado **B** por **A**, y se comprueba que el resultado es la matriz unidad2.

Es con grandes matrices o grandes sistemas de ecuaciones como MATLAB obtiene toda la potencia del ordenador. Por ejemplo, las siguientes instrucciones permiten calcular la ***potencia de cálculo del ordenador en Megaflops*** (millones de operaciones aritméticas por segundo). En la primera línea se crean tres matrices de tamaño 1000×1000, las dos primeras con valores aleatorios y la tercera con valores cero. La segunda línea toma tiempos, realiza el producto de matrices, vuelve a tomar tiempos y calcula de modo aproximado el número de millones de operaciones realizadas. La tercera línea calcula los Megaflops por segundo, para lo cual utiliza la función ***etime()*** que calcula el tiempo transcurrido entre dos instantes definidos por dos llamadas a la función ***clock3***:

```
>> n=1000; A=rand(n); B=rand(n); C=zeros(n);

>> tini=clock; C=B*A; tend=clock; mflops=(2*n^3)/1000000;

>> mflops/etime(tend,tini)
```

Otro de los puntos fuertes de MATLAB son los gráficos, que se verán con más detalle en una sección posterior. A título de ejemplo, se puede teclear la siguiente línea y pulsar ***intro***:

```
>> x=-4:.01:4; y=sin(x); plot(x,y), grid, title('Función seno(x)')
```

En la Figura 1 se puede observar que se abre una nueva ventana en la que aparece representada la función ***sin(x)***. Esta figura tiene un título "Función seno(x)" y una cuadrícula o "grid". En realidad la línea anterior contiene también varias instrucciones separadas por comas o puntos y comas. En la primera se crea un vector **x** con 801 valores reales entre -4 y 4, separados por una centésima. A continuación se crea un vector **y**, cada uno de cuyos elementos es el seno del correspondiente elemento del vector **x**. Después se dibujan los valores de **y** en ordenadas frente a los de **x** en abscisas. Las dos últimas instrucciones establecen la cuadrícula y el título.

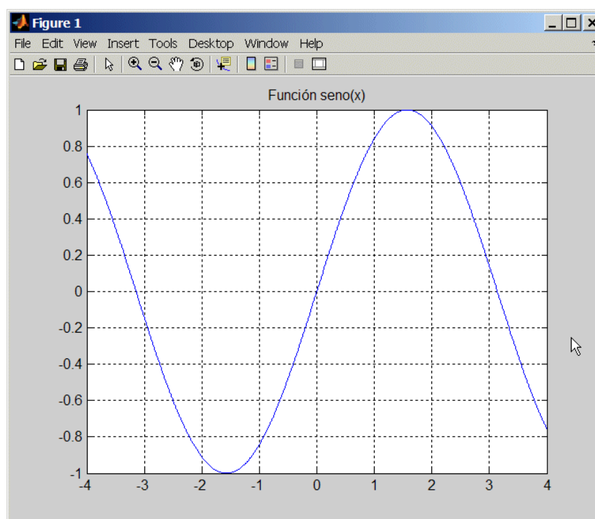


Figura 1. Gráfico de la función ***seno(x)***.

Un pequeño aviso antes de seguir adelante. Además de con la ***Command History***, es posible recuperar comandos anteriores de MATLAB y moverse por dichos comandos con el ratón y con las teclas- flechas ↑ y ↓. Al pulsar la primera de dichas flechas aparecerá el comando que se había introducido inmediatamente antes. De modo análogo es posible moverse sobre la línea de comandos con las teclas ← y →, ir al principio de la línea con la tecla ***Inicio***, al final de la línea con ***Fin***, y borrar toda la línea con ***Esc***. Recuérdese que sólo hay una línea activa (la última).

Para borrar todas las salidas anteriores de MATLAB y dejar limpia la ***Command Window*** se pueden utilizar las funciones ***clc*** y ***home***. La función ***clc*** (*clear console*) elimina todas las salidas anteriores, mientras que ***home*** las mantiene, pero lleva el ***prompt*** (**>>)** a la primera línea de la ventana. Si se desea salir de MATLAB basta teclear los comandos ***quit*** o ***exit***, elegir ***Exit*** MATLAB en el menú ***File*** o utilizar cualquiera de los medios de terminar una aplicación en ***Windows***.

2.-Operadores lógicos

Los operadores lógicos de MATLAB son los siguientes:

- & *and* (función equivalente: **and(A,B)**). Se evalúan siempre ambos operandos, y el resultado es ***true*** sólo si ambos son ***true***.
- && *and* breve: si el primer operando es ***false*** ya no se evalúa el segundo, pues el resultado final ya no puede ser más que ***false***.
- | *or* (función equivalente: **or(A,B)**). Se evalúan siempre ambos operandos, y el resultado es ***false*** sólo si ambos son ***false***.
- || *or* breve: si el primer operando es ***true*** ya no se evalúa el segundo, pues el resultado final no puede ser más que ***true***.
- ~ *negación lógica* (función equivalente: **not(A)**)
- xor(A,B) realiza un "or exclusivo", es decir, devuelve 0 en el caso en que ambos sean 1 ó ambos sean 0.

Los operadores lógicos se combinan con los relacionales para poder comprobar el cumplimiento de condiciones múltiples. Más adelante se verán otros ejemplos y ciertas funciones de las que dispone MATLAB para facilitar la aplicación de estos operadores a vectores y matrices.

Los ***operadores lógicos breves*** (&&) y (||) se utilizan para simplificar las operaciones de comparación evitando operaciones innecesarias, pero también para evitar ciertos errores que se producirían en caso de evaluar incondicionalmente el segundo argumento. Considérese por ejemplo la siguiente sentencia, que evita una división por cero:

r = (b~=0) && (a/b>0);

3.-Operaciones básicas y variables

Matlab distingue entre mayúsculas y minúsculas. Entonces A y a pueden ser nombres apropiados para dos variables distintas. El nombre de toda variable debe comenzar por una letra. X, x, X1ó media son nombres válidos de variables. +, -, *, /, ^ son los símbolos que denotan las operaciones suma, diferencia, producto, cociente y potencia. Las operaciones se van realizando por orden de prioridad, primero las potencias, después las multiplicaciones y divisiones y, finalmente, las sumas y restas. Las operaciones de igual prioridad se llevan a cabo de izquierda a derecha. Si se quiere asignar a una variable x el valor 3 + 25, se escribe a continuación del prompt >> x=3+2^5;

Se pueden utilizar las funciones matemáticas usuales. Por ejemplo, si se quiere asignar a la variable a el valor *√3*, se escribe >> a=sqrt(3). Los comentarios deben ir precedidos del signo %. Cuando una instrucción no cabe en una línea se puede cortar en dos trozos, en cuyo caso el primero de ellos debe acabar con tres puntos seguidos y el segundo trozo va en la línea siguiente.

Si se quiere conocer el valor de una variable, basta teclear su nombre. Para conocer las variables que se han usado hasta el momento, se utiliza el comando who y, si se quiere más información, whos. Para eliminar una variable denominada nombre >>clear nombre. A continuación damos una relación con los nombres de las funciones más habituales:

Raíz cuadrada: sqrt(x).

Función exponencial: exp(x).

Función seno: sin(x).

Función coseno: cos(x).

Función tangente: tan(x).

Función cotangente: cot(x).

Función arcoseno: asin(x).

Función arcotangente: atan(x).

Función logaritmo neperiano: log(x).

Función logaritmo en base 10: log10(x).

Resto de la división entera de *m* por *n*: rem(m,n).

Matlab tiene definidas variables con un valor predeterminado. Veamos algunos ejemplos más importantes:

pi. Su valor es el número π.

inf. Su valor es infinito. Aparece, por ejemplo, cuando hacemos 1/0.

eps. Es el número positivo más pequeño con que trabaja Matlab

>>eps

ans 2.2204e-016

Terminamos esta sección explicando cómo puede controlarse el formato numérico con que se muestran en pantalla los resultados numéricos de los cálculos. Sin embargo, hay que señalar que esto no tiene nada que ver con la precisión con la que se realizan dichos cálculos. Si queremos que se muestren en pantalla los resultados con sólo 5 dígitos, se teclea format short

>> format short a=1/3

ans 0.3333

Si deseamos ver en pantalla los resultados con 15 dígitos, se tecleará format long. Si tecleamos format rat, Matlab busca una aproximación racional.

Ejemplo. Matlab denota el número π por pi. Veamos cuál es el resultado de teclear format rat antes de pi:

>> format rat

pi

ans 355/113.

Finalmente, format short reproduce el resultado en notación punto flotante con cinco dígitos

Ejemplo.

>> format short e

a=1/3

ans 3.3333e-001

4.-Vectores

Si queremos introducir las componentes de un vector *v*, las escribiremos entre corchetes separándolos con comas o espacios. Ejemplo. El vector fila *v* = (1*;* 2*;*-1) se introduce en Matlab como sigue:

>> *v* = [1 2 - 1];

Nótese el punto y coma final. Si no se pone, al pulsar enter Matlab muestra en pantalla la fila 1 2 -1. Si se pone el punto y coma, Matlab guarda en memoria el vector *v* = (1*;* 2*;*-1) y no lo muestra en pantalla. Esto es un hecho general que se producirá cada vez que escribimos alguna instrucción.

Si colocamos el punto y coma final, Matlab ejecuta la instrucción en cuestión y no muestra en pantalla el resultado ni los cálculos involucrados. Por tanto, es importante tener en cuenta esta característica de Matlab. Por ejemplo, si los cálculos que debe realizar Matlab son numerosos, puede que no nos interese verlos en pantalla. Si el vector ó matriz fila tienen la particularidad de que sus componentes están igualmente espaciadas, hay una forma más simple de introducirlo. Así, el vector *v* = (1*;* 3*;* 5*;* 7) se puede introducir de la forma siguiente:

>> v=1:2:7;

Es decir, se indica, separados por dos puntos, la primera componente, el desfase de uno al siguiente y el último. Cuando el desfase es la unidad se puede omitir. Entonces $v = 4 : 8$, es la forma más simple de indicar el vector $v = (4; 5; 6; 7; 8)$.

Otra forma de introducir un vector fila con las componentes igualmente espaciadas consiste en indicar la primera componente, la última y el número total de componentes. Ejemplo:

El vector *v* que tiene 10 componentes igualmente espaciadas siendo 1 la primera y 18 la última se indica >> *v* = *linspace*(1; 18; 10);

Recuérdese que, si no colocamos al final el punto y coma, aparecerá en pantalla una fila de 10 números que no son otra cosa que las componentes de *v*. Cuando necesitemos conocer el número de componentes de un vector *v* bastará recurrir a la función length(v).

sort(v) es el vector que resulta al escribir las componentes de *v* de menor a mayor.

Para finalizar, veamos algunas operaciones habituales entre vectores:

- Suma: >> u+v.

- Producto por un escalar: >> a*v.

- Producto escalar: >> dot(u,v).

- Producto vectorial: >> cross(u,v).

4.1.-Funciones que actúan sobre vectores

Las siguientes funciones ***sólo actúan sobre vectores*** (no sobre matrices, ni sobre escalares):

- [xm,im]=max(x) máximo elemento de un vector. Devuelve el valor máximo **xm** y la posición que ocupa **im**
- min(x) mínimo elemento de un vector. Devuelve el valor mínimo y la posición que ocupa
- sum(x) suma de los elementos de un vector
- cumsum(x) devuelve el vector suma acumulativa de los elementos de un vector (cada elemento del resultado es una suma de elementos del original)
- mean(x) valor medio de los elementos de un vector

- std(x) desviación típica
- prod(x) producto de los elementos de un vector
- cumprod(x) devuelve el vector producto acumulativo de los elementos de un vector
- [y,i]=sort(x) ordenación de menor a mayor de los elementos de un vector **x**. Devuelve el vector ordenado **y**, y un vector **i** con las posiciones iniciales en **x** de los elementos en el vector ordenado **y**.

En realidad estas funciones ***se pueden aplicar también a matrices***, pero en ese caso ***se aplican por separado a cada columna de la matriz***, dando como valor de retorno un vector resultado de aplicar la función a cada columna de la matriz considerada como vector. Si estas funciones se quieren aplicar a las filas de la matriz basta aplicar dichas funciones a la matriz traspuesta.

5.- Funciones para cálculos con polinomios

Para MATLAB un polinomio se puede definir mediante un vector de coeficientes. Por ejemplo, el polinomio:

$$x^4 - 8x^2 + 6x - 10 = 0$$

se puede representar mediante el vector [1, 0, -8, 6, -10]. MATLAB puede realizar diversas operaciones sobre él, como por ejemplo evaluarlo para un determinado valor de x (función polyval()) y calcular las raíces (función roots()):

```
>> pol=[1  0  -8  6  -10]

pol =

1  0  -8  6  -10

>> roots(pol)

ans =

-3.2800

2.6748
```

0.3026 + 1.0238i

0.3026 - 1.0238i

>> polyval(pol,1)

ans =

-11

Para calcular producto de polinomios MATLAB utiliza una función llamada conv() (de producto de convolución). En el siguiente ejemplo se va a ver cómo se multiplica un polinomio de segundo grado por otro de tercer grado:

>> pol1=[1 -2 4]

pol1 =

1 -2 4

>> pol2=[1 0 3 -4]

pol2 =

1 0 3 -4

>> pol3=conv(pol1,pol2)

pol3 =

1 -2 7 -10 20 -16

Para dividir polinomios existe otra función llamada ***deconv()***. Las funciones orientadas al cálculo con polinomios son las siguientes:

- poly(A) polinomio característico de la matriz **A**
- roots(pol) raíces del polinomio **pol**
- polyval(pol,x) evaluación del polinomio **pol** para el valor de **x**. Si **x** es un vector, **pol** se evalúa para cada elemento de **x**
- polyvalm(pol,A) evaluación del polinomio **pol** de la matriz **A**
- conv(p1,p2) producto de convolución de dos polinomios **p1** y **p2**

- [c,r]=deconv(p,q) división del polinomio **p** por el polinomio **q**. En **c** se devuelve el cociente y en **r** el resto de la división
- residue(p1,p2) descompone el cociente entre **p1** y **p2** en suma de fracciones simples (ver ***>>help residue***)
- polyder(pol) calcula la derivada de un polinomio
- polyder(p1,p2) calcula la derivada de producto de polinomios
- polyfit(x,y,n) calcula los coeficientes de un polinomio **p(x)** de grado **n** que se ajusta a los datos **p(x(i)) ~= y(i)**, en el sentido de mínimo error cuadrático medio.
- interp1(xp,yp,x) calcula el valor interpolado para la abscisa **x** a partir de un conjunto de puntos dado por los vectores **xp** e **yp**.
- interp1(xp,yp,x,'m') como la anterior, pero permitiendo especificar también el método de interpolación. La cadena de caracteres **m** admite los valores 'nearest', 'linear', 'spline', 'pchip', 'cubic' y 'v5cubic'.

6.- Integración numérica de funciones

Lo primero que se va a hacer es calcular la integral definida de esta función entre dos valores de la abscisa x. En inglés, al cálculo numérico de integrales definidas se le llama quadrature. Sabiendo eso, no resulta extraño el comando con el cual se calcula el área comprendida bajo la función entre los puntos 0 y 1 (obsérvese que la referencia de la función a integrar se pasa por medio del operador @ precediendo al nombre de la función. También podría crearse una variable para ello):

```
>> area = quad(@prueba, 0, 1)
area =
29.8583
```

Si se teclea help quad se puede obtener más de información sobre esta función, incluyendo el método utilizado (Simpson) y la forma de controlar el error de la integración. La función quadl() utiliza un método de orden superior (Lobatto), mientras que la función dblquad() realiza integrales definidas dobles y la función triplequad() realiza integrales de volumen. Ver el Help o los manuales online para más información.

7.- Gráficos

7.1.- Gráficos bidimensionales

A estas alturas, después de ver cómo funciona este programa, a nadie le puede resultar extraño que los gráficos 2-D de MATLAB estén fundamentalmente orientados a la representación gráfica de vectores (y matrices). En el caso más sencillo los argumentos básicos de la función ***plot*** van a ser vectores. Cuando una matriz aparezca como argumento, se considerará como un conjunto de vectores columna (en algunos casos también de vectores fila).

MATLAB utiliza un tipo especial de ventanas para realizar las operaciones gráficas. Ciertos comandos abren una ventana nueva y otros dibujan sobre la ventana activa, bien sustituyendo lo que hubiera en ella, bien añadiendo nuevos elementos gráficos a un dibujo anterior. Todo esto se verá con más detalle en las siguientes secciones.

Funciones gráficas 2D elementales

MATLAB dispone de cinco funciones básicas para crear gráficos 2-D. Estas funciones se diferencian principalmente por el *tipo de escala* que utilizan en los ejes de abscisas y de ordenadas. Estas cuatro funciones son las siguientes:

- plot() crea un gráfico a partir de vectores y/o columnas de matrices, con escalas lineales sobre ambos ejes
- plotyy() dibuja dos funciones con dos escalas diferentes para las ordenadas, una a la derecha y otra a la izquierda de la figura.
- loglog() ídem con escala logarítmica en ambos ejes
- semilogx() ídem con escala lineal en el eje de ordenadas y logarítmica en el eje de abscisas
- semilogy() ídem con escala lineal en el eje de abscisas y logarítmica en el eje de ordenadas

En lo sucesivo se hará referencia casi exclusiva a la primera de estas funciones (***plot***). Las demás se pueden utilizar de un modo similar.

Existen además otras funciones orientadas a añadir títulos al gráfico, a cada uno de los ejes, a dibujar una cuadrícula auxiliar, a introducir texto, etc. Estas funciones son las siguientes:

- title('título') añade un título al dibujo
- xlabel('tal') añade una etiqueta al eje de abscisas. Con ***xlabel off*** desaparece
- ylabel('cual') añade una etiqueta al eje de ordenadas. Con ***ylabel off*** desaparece

- text(x,y,'texto') introduce 'texto' en el lugar especificado por las coordenadas **x** e **y**. Si **x** e **y** son vectores, el texto se repite por cada par de elementos. Si **texto** es también un vector de cadenas de texto de la misma dimensión, cada elemento se escribe en las coordenadas correspondientes
- gtext('texto') introduce **texto** con ayuda del ratón: el cursor cambia de forma y se espera un clic para introducir el texto en esa posición
- legend() define rótulos para las distintas líneas o ejes utilizados en la figura. Para más detalle, consultar el ***Help***
- grid activa la inclusión de una cuadrícula en el dibujo. Con ***grid off*** desaparece la cuadrícula

Borrar texto (u otros elementos gráficos) es un poco más complicado; de hecho, hay que preverlo de antemano. Para poder hacerlo hay que recuperar previamente el *valor de retorno* del comando con el cual se ha creado. Después hay que llamar a la función ***delete*** con ese valor como argumento.

Considérese el siguiente ejemplo:

>> v = text(1,.0,'seno')

v =

76.0001

>> delete(v)

Los dos grupos de funciones anteriores no actúan de la misma forma. Así, la función ***plot*** dibuja una nueva figura en la ventana activa (en todo momento MATLAB tiene una ventana activa de entre todos las ventanas gráficas abiertas), o abre una nueva figura si no hay ninguna abierta, sustituyendo cualquier cosa que hubiera dibujada anteriormente en esa ventana. Para verlo, se comenzará creando un par de vectores **x** e **y** con los que trabajar:

>> x=[-10:0.2:10]; y=sin(x);

Ahora se deben ejecutar los comandos siguientes (se comienza cerrando la ventana activa, para que al crear la nueva ventana aparezca en primer plano):

>> close % se cierra la ventana gráfica activa anterior

>> grid % se crea una ventana con una cuadrícula

>> plot(x,y) % se dibuja la función seno borrando la cuadrícula

Se puede observar la diferencia con la secuencia que sigue:

>> close

>> plot(x,y) % se crea una ventana y se dibuja la función seno

>> grid % se añade la cuadrícula sin borrar la función seno

En el primer caso MATLAB ha creado la cuadrícula en una ventana nueva y luego la ha borrado al ejecutar la función ***plot***. En el segundo caso, primero ha dibujado la función y luego ha añadido la cuadrícula. Esto es así porque hay funciones como ***plot*** que por defecto crean una nueva figura, y otras funciones como ***grid*** que se aplican a la ventana activa modificándola, y sólo crean una ventana nueva cuando no existe ninguna ya creada. Más adelante se verá que con la función ***hold*** pueden añadirse gráficos a una figura ya existente respetando su contenido.

Función plot

Esta es la función clave de todos los gráficos 2-D en MATLAB. Ya se ha dicho que el elemento básico de los gráficos bidimensionales es el **vector**. Se utilizan también cadenas de 1, 2 ó 3 caracteres para indicar *colores* y *tipos de línea*. La función ***plot()***, en sus diversas variantes, no hace otra cosa que dibujar vectores. Un ejemplo muy sencillo de esta función, en el que se le pasa un único vector como argumento, es el siguiente:

>> x=[1 3 2 4 5 3]

x =

1 3 2 4 5 3

>> plot(x)

El resultado de este comando es que se abre una ventana mostrando el gráfico de la Figura 2. Por defecto, los distintos puntos del gráfico se unen con una línea continua. También por defecto, el color que se utiliza para la primera línea es el azul.

Cuando a la función ***plot()*** se le pasa un único vector –real– como argumento, dicha función dibuja en ordenadas el valor de los **n** elementos del vector frente a los índices 1, 2, ... **n** del mismo en abscisas. Más adelante se verá que si el vector es complejo, el funcionamiento es bastante diferente.

En la pantalla de su ordenador se habrá visto que MATLAB utiliza por defecto color blanco para el fondo de la pantalla y otros colores más oscuros para los ejes y las gráficas.

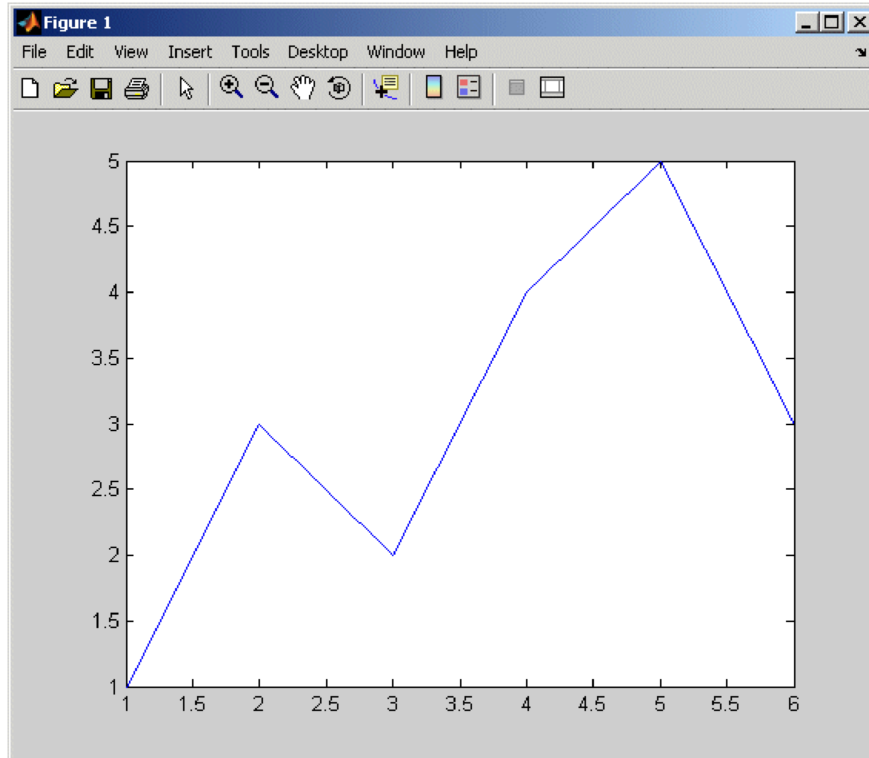


Figura 2.- Gráfico del vector x=[1 3 2 4 5 3].

Una segunda forma de utilizar la función ***plot()*** es con dos vectores como argumentos. En este caso los elementos del segundo vector se representan en ordenadas frente a los valores del primero, que se representan en abscisas. Véase por ejemplo cómo se puede dibujar un cuadrilátero de esta forma (obsérvese que para dibujar un polígono cerrado el último punto debe coincidir con el primero):

```
>> x=[1 6 5 2 1]; y=[1 0 4 3 1];
>> plot(x,y)
```

La función ***plot()*** permite también dibujar múltiples curvas introduciendo varias parejas de vectores como argumentos. En este caso, cada uno de los segundos vectores se dibujan en ordenadas como función de los valores del primer vector de la pareja, que se representan en abscisas. Si el usuario no decide otra cosa, para las sucesivas líneas se utilizan colores que son permutaciones cíclicas del ***azul***, ***verde***, ***rojo***, ***cyan***, ***magenta***, ***amarillo*** y ***negro***. Obsérvese bien cómo se dibujan el seno y el coseno en el siguiente ejemplo:

```
>> x=0:pi/25:6*pi;
>> y=sin(x); z=cos(x);
>> plot(x,y,x,z)
```

Ahora se va a ver lo que pasa con los **vectores complejos**. Si se pasan a ***plot()*** varios vectores complejos como argumentos, MATLAB simplemente representa las partes reales y

desprecia las partes imaginarias. Sin embargo, un único argumento complejo hace que se represente la parte real en abscisas, frente a la parte imaginaria en ordenadas. Véase el siguiente ejemplo. Para generar un vector complejo se utilizará el resultado del cálculo de valores propios de una matriz formada aleatoriamente:

>> plot(eig(rand(20,20)),'+')

donde se ha hecho uso de elementos que se verán en la siguiente sección, respecto a dibujar con distintos tipos de “markers” (en este caso con signos +), en vez de con línea continua, que es la opción por defecto. En el comando anterior, el segundo argumento es un carácter que indica el tipo de marker elegido. El comando anterior es equivalente a:

>> z=eig(rand(20,20));

>> plot(real(z),imag(z),'+')

Como ya se ha dicho, si se incluye más de un vector complejo como argumento, se ignoran las partes imaginarias. Si se quiere dibujar varios vectores complejos, hay que separar explícitamente las partes reales e imaginarias de cada vector, como se acaba de hacer en el último ejemplo.

El comando ***plot*** puede utilizarse también con matrices como argumentos. Véanse algunos ejemplos sencillos:

- plot(A) dibuja una línea por cada columna de **A** en ordenadas, frente al índice de los elementos en abscisas
- plot(x,A) dibuja las columnas (o filas) de **A** en ordenadas frente al vector **x** en abscisas. Las dimensiones de **A** y **x** deben ser coherentes: si la matriz **A** es cuadrada se dibujan las columnas, pero si no lo es y la dimensión de las filas coincide con la de **x**, se dibujan las filas
- plot(A,x) análogo al anterior, pero dibujando las columnas (o filas) de **A** en abscisas, frente al valor de **x** en ordenadas
- plot(A,B) dibuja las columnas de **B** en ordenadas frente a las columnas de **A** en abscisas, dos a dos. Las dimensiones deben coincidir
- plot(A,B,C,D) análogo al anterior para cada par de matrices. Las dimensiones de cada par deben coincidir, aunque pueden ser diferentes de las dimensiones de los demás pares

Se puede obtener una excelente y breve descripción de la función ***plot()*** con el comando ***help plot*** o ***helpwin plot***. La descripción que se acaba de presentar se completará en la siguiente sección, en donde se verá cómo elegir los colores y los tipos de línea.

Estilos de línea y marcadores en la función plot

En la sección anterior se ha visto cómo la tarea fundamental de la función ***plot()*** era dibujar los valores de un vector en ordenadas, frente a los valores de otro vector en abscisas. En el caso general esto exige que se pasen como argumentos un par de vectores. En realidad, el conjunto básico de argumentos de esta función es una *tripleta* formada por dos vectores y una cadena de 1, 2 ó 3 caracteres que indica el color y el tipo de línea o de marker. En la tabla siguiente se pueden observar las distintas posibilidades.

Símbolo	Color	Símbolo	Marcadores (markers)
y	yellow	.	puntos
m	magenta	o	círculos
c	cyan	x	marcas en x
r	red	+	marcas en +
g	green	*	marcas en *
b	blue	s	marcas cuadradas (square)
w	white	d	marcas en diamante (diamond)
k	black	^	triángulo apuntando arriba
		v	triángulo apuntando abajo
Símbolo	**Estilo de línea**	>	triángulo apuntando a la dcha
-	líneas continuas	<	triángulo apuntando a la izda
:	líneas a puntos	p	estrella de 5 puntas
-.	líneas a barra-punto	h	estrella se seis puntas
--	líneas a trazos		

Tabla 1. Colores, markers y estilos de línea,

Cuando hay que dibujar varias líneas, por defecto se van cogiendo sucesivamente los colores de la tabla comenzando por el azul, hacia arriba, y cuando se terminan se vuelve a empezar otra vez por el azul. Si el fondo es blanco, este color no se utiliza para las líneas. También es posible añadir en la función ***plot*** algunos especificadores de línea que controlan el espesor de la línea, el tamaño de los marcadores, etc. Considérese el siguiente ejemplo:

plot(x,y,'-.rs', **'LineWidth'**,4, **'MarkerEdgeColor'**,'k', **'MarkerFaceColor'**, 'g',... **'MarkerSize'**,10)

Ejes a medida

Para fijar los valores máximo y mínimo de los ejes:

>>axis([xmin xmax ymin ymax])

Para que la escala sea la misma en ambos ejes:

>>axis equaló axis('equal')

Para que la gráfica sea un cuadrado:

>>axis square

Si se quiere que aparezca una rejilla: grid on.

Por ejemplo: las Curvas planas

Si se desea obtener la gráfica de la función $y = y(x)$ en el intervalo [a,b], debemos tener presente que Matlab dibuja las curvas punto a punto; es decir, calcula los puntos $(x; y(x))$, para los valores de x que le indiquemos y representa dichos puntos unidos por un segmento. Por ello, se empieza estableciendo la matriz fila x cuyos elementos son los valores de x para los que se computará el valor correspondiente de (x). Lo usual será tomar puntos igualmente espaciados en el intervalo [a,b], incluyendo los extremos. Tomando la distancia entre dos valores consecutivos de x convenientemente pequeña, el aspecto final será el de una verdadera curva en lugar de una poligonal.

Ejemplo . *Dibujar la curva de ecuación y = xsenx en el intervalo* $[-2\pi; 2\pi]$.

>> x=linspace(-2*pi,2*pi,60); % Tomamos 60 valores de x igual % mente espaciados en [-2π; 2π]

y=x.^ 2.*sin(x);% matriz fila con los valores de $y(x)$ plot(x,y)

Pulsando enter, se abre una ventana gráfica con la curva. Se pueden dibujar varias curvas en la misma ventana gráfica. Si el intervalo de variación de x es el mismo, se puede proceder como se muestra en el ejemplo siguiente. Ejemplo. *Representar gráficamente las curvas y1 = x2 e y2 = xex en el intervalo [-3,3].*

>> x=linspace(-3,3,90);

*y1=x.^ 2;y2=x.*exp(x);*

plot(x,y1,x,y2)

De esta forma se consigue que se abra una ventana gráfica con las dos curvas. Otra forma de conseguir el mismo resultado consiste en usar la orden hold on. Si ya tenemos una ventana gráfica con una curva y queremos dibujar una segunda curva en la misma ventana, ponemos hold on y a continuación las órdenes necesarias para dibujar la segunda curva.

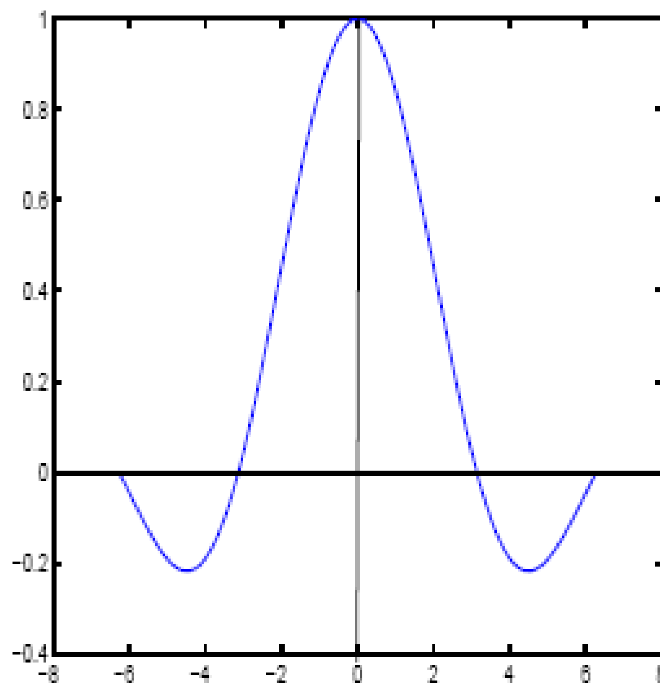
Si el intervalo donde se quiera dibujar cada curva no es el mismo, se puede conseguir el mismo resultado de la forma siguiente. Se dibuja primero una de las curvas, se teclea hold on y acto seguido se dibuja la otra Ejemplo. Dibujar y = x en [-1,1] e y = xex en [0,2].

```
>> x1=-1:.1:1;

y1=x1;

plot(x1,y1)

hold on

x2=0:.1:2; y2=x2.*exp(x2);

plot(x2,y2)
```

Por el contrario, cuando ya se tiene una ventana gráfica abierta y se quiere dibujar una nueva curva en otra ventana gráfica, pero sin perder la primera ventana, tecleamos figure y se abre una ventana gráfica nueva donde podremos hacer la nueva representación. EJEMPLO. Supongamos que se desea obtener la gráfica de f(x) = sen(x)=x, para x є [-2π; 2π]. Se presenta el problema de que para x = 0 no está definida la función, aunque $\lim_{x-0}$ f(x) existe y vale 1. Manejando adecuadamente la variable eps, podemos evitar este problema.

```
>>x=linspace(-2*pi,2*pi,100);

t=x+eps;

y=sin(t)./t;

plot(t,y)
```

y obtenemos:



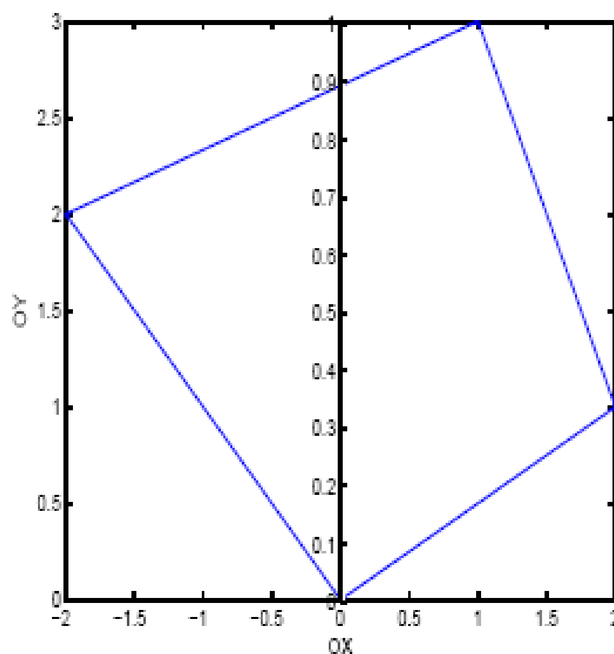
Por ejemplo: Dibujo de poligonales

Supongamos que se desea dibujar la poligonal de vértices ($x_i; y_i$) con $i = 1; ::; n$. Definiríamos las matrices fila x e y que contienen las coordenadas correspondientes y el comando plot(x,y) dibuja la poligonal (recordar cómo dibuja las curvas Matlab). Si la poligonal es cerrada, el último vértice ha de ser ($x_1; y_1$).

Ejemplo. *Dibujar la poligonal cerrada de vértices* (0*;* 0)*;* (2*;* 1)*;* (1*;* 3) y (-2*;* 2).

```
>>x=[0 2 1 -2 0];

y=[0 1 3 2 0];

plot(x,y)
```

y el resultado sería



Por ejemplo: Curvas en polares

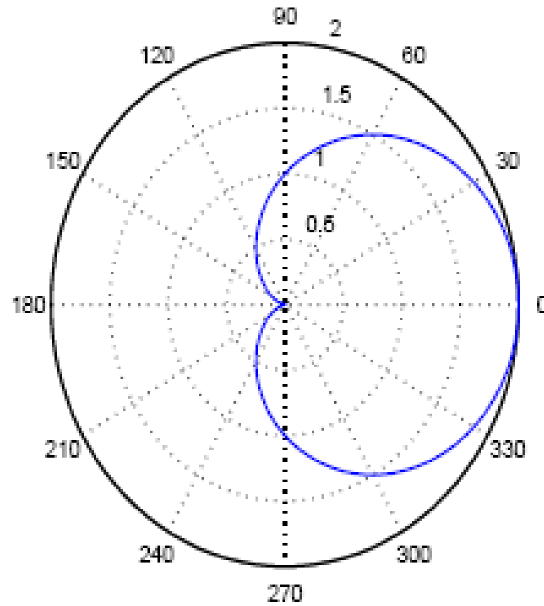
Empezamos recordando cómo se relacionan las coordenadas polares con las cartesianas. Vemos en la figura que ω es el ángulo que forman el vector de posición del punto *P* con la dirección positiva del eje OX y *r* es el módulo de dicho vector. Por un lado, el Teorema de Pitágoras nos dice que $r2 = x2 + y2$ y, por otro, usando las definiciones de sen *ω* y cos ω, obtenemos $x = r \cos \omega$ e $y = r$ sen ω. Si de una curva plana sabemos que las coordenadas polares de sus puntos, $(r; \omega)$, verifican la igual dad $r = r(\omega)$, para ω *2* $[\omega 1; \omega 2]$, diremos que $r = r(\omega)$ es la ecuación de la curva en coordenadas polares. La ecuación de la circunferencia unidad en cartesianas es $x2 + y2 = 1$ y en coordenadas polares $r = 1$. En general, para obtener la ecuación en polares, conocida la ecuación de una curva en cartesianas, basta sustituir en esta última ecuación *x* e *y* por *r* cos ω y *r* sen ω, respectivamente.

Ejemplo. *Dibujar la curva de ecuación* $r = 1 + \cos \omega$*, para* $0 \cdot \omega \cdot 2\pi$.

Por comodidad, vamos a usar *w* en lugar de ω.

```
>> w=linspace(0,2*pi,60);

r=1+cos(w);

polar(w,r)
```

pulsando enter se abre una ventana gráfica que muestra la curva siguiente (denominada cardioide).



7.2.- Gráficos tridimensionales

Quizás sea ésta una de las características de MATLAB que más admiración despierta entre los usuarios no técnicos (cualquier alumno de ingeniería sabe que hay ciertas operaciones algebraicas – como la descomposición de valores singulares, sin ir más lejos– que tienen dificultades muy superiores, aunque "luzcan" menos).

Tipos de funciones gráficas tridimensionales

MATLAB tiene posibilidades de realizar varios tipos de gráficos 3D. Para darse una idea de ello, lo mejor es verlo en la pantalla cuanto antes, aunque haya que dejar las explicaciones detalladas para un poco más adelante.

La primera forma de gráfico 3D es la función ***plot3***, que es el análogo tridimensional de la función ***plot***. Esta función dibuja puntos cuyas coordenadas están contenidas en 3 vectores, bien uniéndolos mediante una línea continua (defecto), bien mediante ***markers***. Asegúrese de que no hay ninguna ventana gráfica abierta y ejecute el siguiente comando que dibuja una línea espiral en color rojo:

```
>> fi=[0:pi/20:6*pi]; plot3(cos(fi),sin(fi),fi,'r'), grid
```

Ahora se verá cómo se representa una función de dos variables. Para ello se va a definir una función de este tipo en un fichero llamado *test3d.m*. La fórmula será la siguiente:

$$z = 3(1-x)^2 e^{-x^2-(y+1)^2} - 10\left(\frac{x}{5} - x^3 - y^5\right)e^{-x^2-y^2} - \frac{1}{3}e^{-(x+1)^2-y^2}$$

El fichero *test3d.m* debe contener las líneas siguientes:

```
function z=test3d(x,y)
z = 3*(1-x).^2.*exp(-(x.^2) - (y+1).^2) ...
- 10*(x/5 - x.^3 - y.^5).*exp(-x.^2-y.^2) ...
- 1/3*exp(-(x+1).^2 - y.^2);
```

Ahora, ejecútese la siguiente lista de comandos (directamente, o mejor creando un fichero llamado *test3dMain.m* que los contenga):

```
>> x=[-3:0.4:3]; y=x;
>> close
>> subplot(2,2,1)
>> figure(gcf),fi=[0:pi/20:6*pi];
>> plot3(cos(fi),sin(fi),fi,'r')
>> grid
>> [X,Y]=meshgrid(x,y);
>> Z=test3d(X,Y);
>> subplot(2,2,2)
>> figure(gcf), mesh(Z)
>> subplot(2,2,3)
>> figure(gcf), surf(Z)
>> subplot(2,2,4)
>> figure(gcf), contour3(Z,16)
```

En la figura resultante (Figura 3) aparece una buena muestra de algunas de las posibilidades gráficas tridimensionales de MATLAB. En las próximas secciones se realizará una explicación más detallada de qué se ha hecho y cómo se ha hecho.

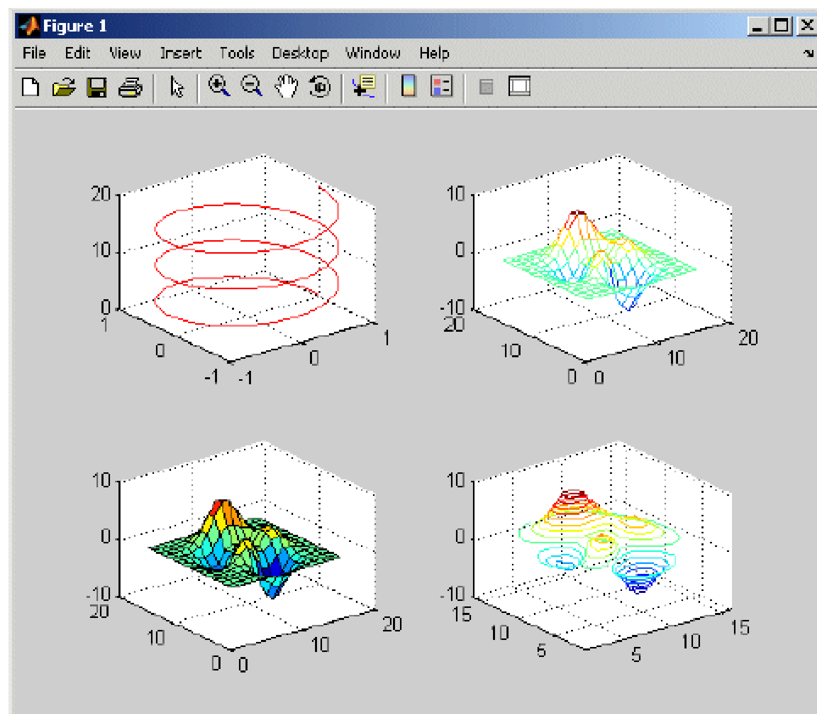


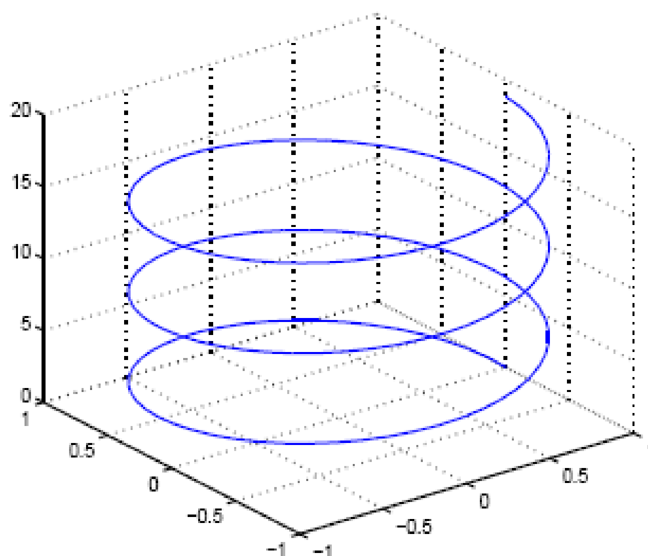
Figura 3. Gráficos 3D realizados con MATLAB.

Por ejemplo: Curvas en el espacio

Supongamos que se quiere dibujar la curva de ecuaciones paramétricas x = cos t; y = sen t; $z = t$, para t 2 [0; 6π]. Podemos usar el comando plot3ó el comando ezplot3.

a) Con plot3:

```
>>t=linspace(0,6*pi,150);
plot3(cos(t),sin(t),t)
grid on
```
y el resultado es



b) Con ezplot3:

```
ezplot3('cos(t)','sin(t)','t', [0,6*pi])
```

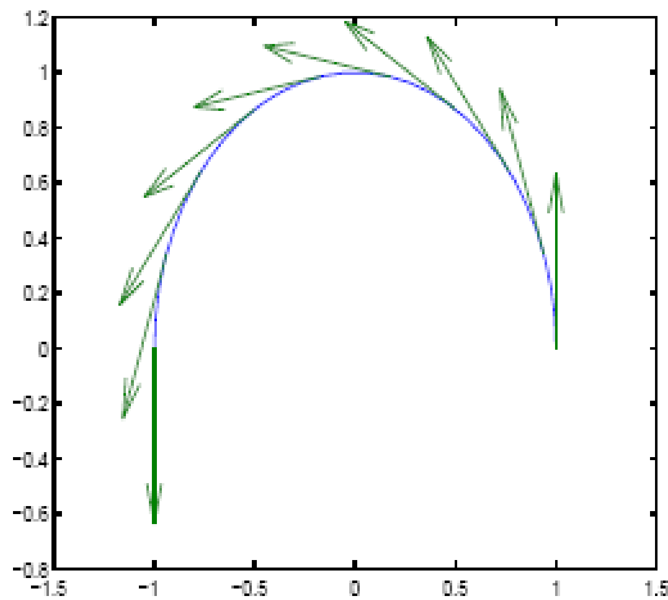
Terminamos esta sección indicando cómo puede conseguirse que aparezcan los vectores tangentes al dibujan curvas en paramétricas .Ejemplo. Representar la curva de ecuaciones paramétricas x = cos(t)*;* y = sen(t)*; t 2* [0*; pi*]*:*

```
>>t=linspace(0,pi,30);
plot(cos(t),sin(t))
```

Si se quiere que aparezcan los vectores tangente, se usa la función quiver.

```
>>t=linspace(0,pi,30);
plot(cos(t),sin(t))
hold on
```

>> t=linspace(0,pi,10);%Dibujamos el vector tangente en sólo 10 puntos % intermedios de la curva quiver(cos(t),sin(t),-sin(t),cos(t)) y se obtiene



Superficies

Para dibujar una superficie de ecuación $z = z(x; y)$, se comienza por establecer los intervalos de variación de x e y. Con la orden [x,y]=meshgrid(-2:.1:2;-1:.1:1) Matlab crea una matriz x con todas sus filas iguales a -2:.1:2 y una matriz y con todas sus columnas iguales a -1:.1:1. De este modo resultan dos matrices con la misma dimensión. Veamos un ejemplo simple. Consideramos la función $z = x2 + y2$ y escribimos la orden >> [x,y]=meshgrid(-1:.5:1,0:.5:1); que produce las matrices

$$x = \begin{pmatrix} -1 & -.5 & 0 & .5 & 1 \\ -1 & -.5 & 0 & .5 & 1 \\ -1 & -.5 & 0 & .5 & 1 \end{pmatrix}, \quad y = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ .5 & .5 & .5 & .5 & .5 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

De esta forma, cuando escribamos z=x.^2+y.^2 en la ventana de comandos, z sería la matriz 3 £ 5 que contiene los valores de z en todos y cada uno de los puntos $(x; y)$ con x igual a uno de los valores -1,-.5,0,.5,1 e y igual a uno de los valores 0,.5,1. Ejemplo. *Dibujar la superficie z* = px2 + *y2 en el dominio* [-3; 3]£[-3; 3]

```
>>[x,y]=meshgrid(-3:.1:3,-3:.1:3);

z=sqrt(x.^2+y.^2);

surf(x,y,z)
```

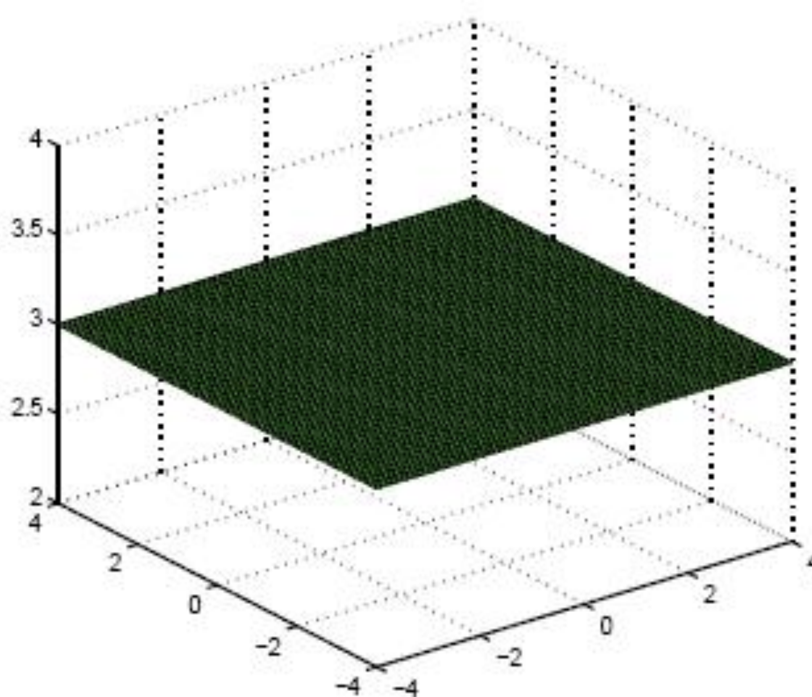
Además de la orden surf(x,y,z), para dibujar una superficie, podemos emplear plot3(x,y,z) o mesh(x,y,z) (la diferencia con surf es que esta rellena los espacios entre líneas). La orden rotate3d permite girar la superficie con el ratón. La orden colormap permite cambiar el color de la superficie. Hay diversas opciones: colormap(pink), colormap(summer), colormap(winter), etc.

En ele ejemplo siguiente mostramos cómo se dibuja una superficie plana paralela a z = 0.Ejemplo. Dibujar la superficie z = 3 en el primer octante.

>> [x,y]=meshgrid(-4:.1:4,-4:.1:4); [m,n]=size(x);% Recordar lo que hemos dicho sobre el comando meshgrid,% usamos size(x) para determinar las dimensiones de x de una manera segura.

z=3*ones(m,n);

surf(x,y,z)



. Curvas de nivel

Supongamos que hemos dibujado una superficie, $z = f(x; y)$, y queremos que en el plano z = 0 aparezcan dibujadas las curvas de nivel. Podemos proceder como sigue

>>hold on

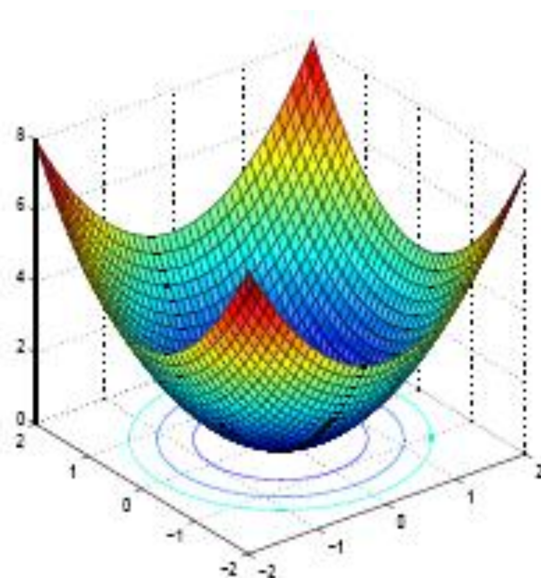
Contour (x,y,z,[c1,c2,c3,...,cn])

y aparecen representadas las curvas de nivel $f(x; y) = ci$, para $i = 1; ::; n$.

Ejemplo. Representar la superficie de ecuación $z = x2 + y2$, para $(x; y)$ *2* [-2; 2] є [-2;]. Además, deseamos que aparezcan representadas las curvas de nivel $f(x; y) = c$, para $c = 1; 2; 3$.

```
>>[x,y]=meshgrid(-2:.1:2,-2:.1:2);

z=x.^2+y.^2;

surf(x,y,z)

hold on

contour(x,y,z,[1,2,3])
```

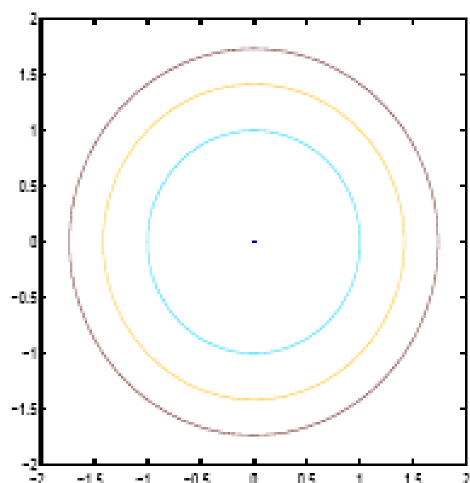
y se obtiene



Si sólo se buscan las curvas de nivel (no se necesita dibujar la superficie), basta escribir

```
>>[x,y]=meshgrid(-2:.1:2,-2:.1:2);

z=x.^2+y.^2;

contour(x,y,z,[1,2,3])
```

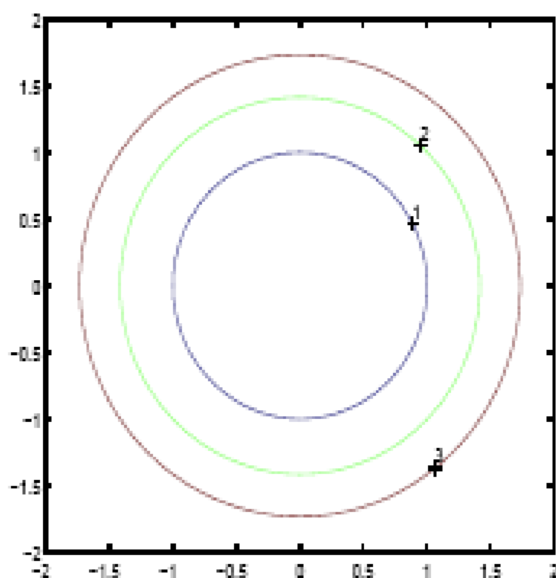
y se obtiene



Si se quiere que aparezcan etiquetas sobre cada curva de nivel que reflejen el valor de la constante *c*, se usa la función clabel

```
>>[x,y]=meshgrid(-2:.1:2,-2:.1:2);

z=x.^2+y.^2;

clabel( contour(x,y,z,[1,2,3]) )
```

resultando

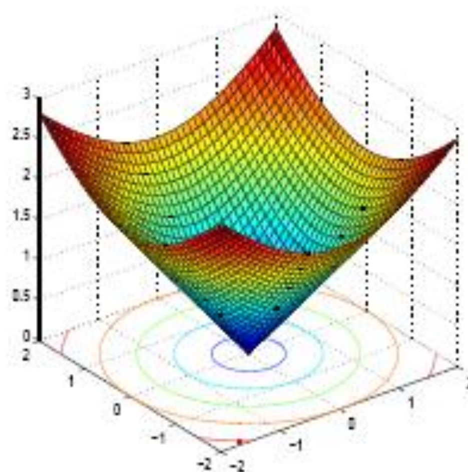


Cuando necesitamos representar una superficie y deseamos que aparezcan las curvas de nivel, hay una forma muy cómoda que consiste en emplear la función surfc(x,y,z).

Ejemplo. Gráfica de la superficie *z* = p*x*2 + *y*2 con las correspondientes curvas de nivel.

```
>>[x,y ]=meshgrid(-2:.1:2,-2:.1:2);

z=sqrt(x.^2+y.^2);

surfc(x,y,z)
```

y resulta



8.- Programación de matlab

Como ya se ha dicho varias veces –incluso con algún ejemplo– MATLAB es una aplicación que se puede programar muy fácilmente. De todas formas, como lenguaje de programación pronto verá que no tiene tantas posibilidades como otros lenguajes (ni tan complicadas...). Se comenzará viendo las bifurcaciones y bucles, y la lectura y escritura interactiva de variables, que son los elementos básicos de cualquier programa de una cierta complejidad.

Archivos .M

Vamos a ver que podemos crear dos tipos de archivos con extensión .m, que llamaremos archivos .M. Se denominan archivos de función y archivos de guión (o de instrucciones). Con los primeros podemos definir nuevas funciones que se añadirán a las que ya trae Matlab (como sin(x), exp(x),sqrt(x),etc.). Los archivos de instrucciones se llaman así porque pueden consistir en una serie de instrucciones a ejecutar. Veremos que en éstos puede ocurrir que en el momento de su ejecución nos pida una serie de valores (inputs). En todos los casos, se deberá ir a File-New-M-File y aparece una ventana (Editor-Untitled) donde podemos

escribir el programa correspondiente. Una vez terminado, vamos a File de dicha ventana y hacemos clic en Save As y podemos guardar el archivo .M con el nombre que le hayamos dado (nombre.m).

Archivos de función

Las dos primeras líneas de un archivo de función tienen la forma function [y,z,..]=nombre(a,b,..)% Una explicación que sirva para reconocer la función en cualquier otro momento a,b,.. denotan las variables de entrada (las variables independientes), mientras que y,z,.. son las variables de salida (dependientes), en ambos casos separadas por comas. A continuación van todas las órdenes que se necesitan para definir la nueva función. Ejemplo. *Crear un archivo de función cuya entrada sea una matriz fila x y cuyas salidas sean la media de x y su desviación típica.*

```
function [media,dest]=estadisticos(x) % Esta función determina la media y la desviación típica de una fila x

n=length(x);

media=sum(x)/n;

s=0;

for k=1:n

s=s+(x(k)-media)^2;

end

dest=sqrt(s/n);

[media,dest]
```

El nombre de todo archivo de función debe coincidir con el nombre de la función y tiene extensión .m. En nuestro caso, sería: nombre.m. El nombre de la función debe empezar con una letra y, para evitar confusiones, debemos asegurarnos que no coincide con el nombre de alguna de las funciones de que dispone Matlab.

Archivos de instrucciones

En Programación nos encontramos a menudo con la necesidad de ejecutar varias veces una misma serie de instrucciones. Por ello, puede resultar conveniente crear un archivo de guión con dichas instrucciones, al que podremos recurrir cada vez que lo necesitemos. El nombre de uno de tales archivos sólo tiene la restricción, como ocurre con los archivos de función, de que debe comenzar por una letra. La estructura de estos archivos es la siguiente: Una primera linea explicativa que describa brevemente el objetivo del archivo. Esta línea de comentarios debe comenzar con %. De esta forma se

consigue que el ordenador no considere esta línea. A continuación se escriben todas las instrucciones que deben ejecutarse en el orden que corresponda. Ejemplo. Vamos a crear un archivo de guión (que llamaremos raices) y que nos pedirá los coeficientes de una ecuación de segundo grado y determinará las raíces reales cuando las haya.

Usaremos el comando if que se estudiará con más detenimiento más adelante. % encuentra las raíces de una ecuación de segundo grado $ax2+bx+c = 0$ y nos pide a,b y c

```
a=input('dame a');

b=input('dame b');

c=input('dame c');

Delta=b^2-4*a*c;

if Delta>=0

[x1,x2]=[-b/(2*a)+sqrt(Delta)/(2*a),-b/(2*a)-sqrt(Delta)/(2*a)]

end
```

Este archivo funciona de la siguiente forma: Cuando tecleamos raíces el ordenador nos pide sucesivamente los valores de a, b y c. Calcula el discriminante de la ecuación, Delta, y, al encontrarse con el comando if, si Delta no es negativo, procede a calcular las raíces. Si Delta es negativo, no nos da ningún resultado, pues no ejecuta las instrucciones que hay entre if y end.

9.- Ficheros creados en Matlab

1. CREAR UNA FUNCIÓN EN MATLAB Y DIBUJARLA

%La función que queremos dibujar es f(x)=x sen(x), por lo tanto creamos el fichero que llamaremos f.m

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

```
function y=f(x)

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

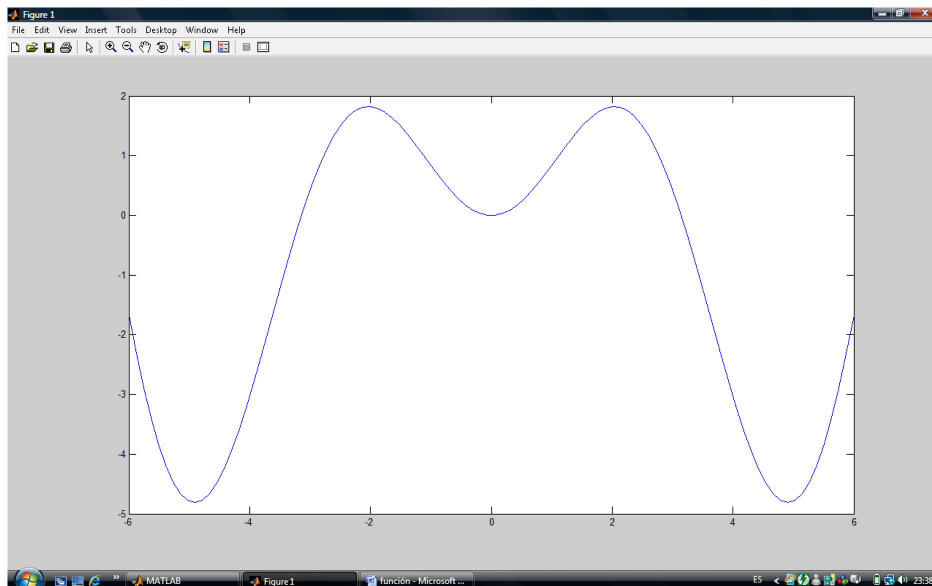
```
>> f(0)% le pedimos que nos dé el valor de la función cuando x=0

ans =

    0

>> f(1)% y cuando x=1

ans =

   0.8415
```

%creamos un programa sencillito para poder dibujarla, al que llamaremos dibujo.m

```
>> edit dibujo.m
```

```
x=linspace(-6,6,100); %ponemos las condiciones a la que queremos quenos dibuje

y=f(x);%le damos el nombre de la función que queremos que nos dibuje

plot(x,y) %añadimos el comando para que nos la dibuje
```

```
>> dibujo% ejecutamos en matlab y nos da el dibujo
```



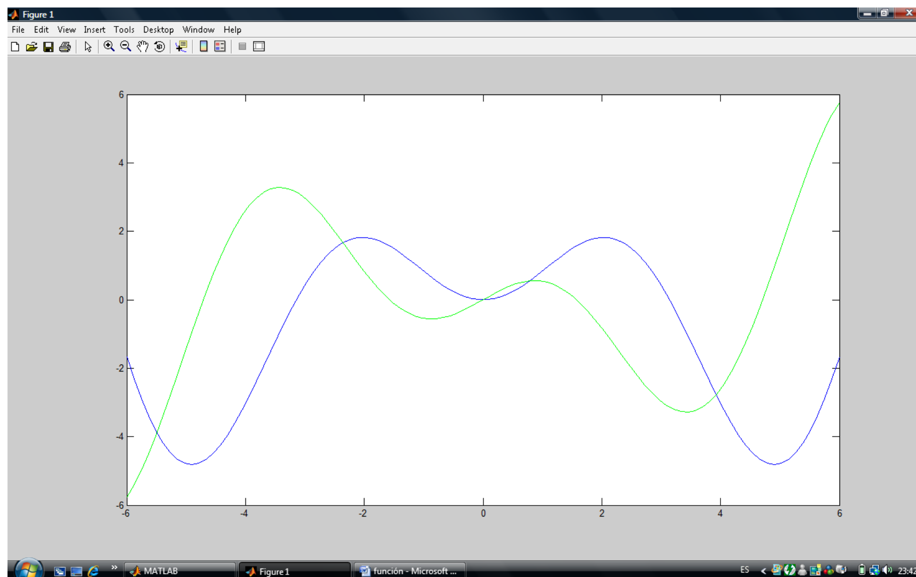
>> edit g.m%creamos otro archive con otra función que llamaremos g.m. La función es f(x)=xcos(x)

```
function y=g(x)

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

>> edit dibujo.m% modificamos el programa de dibujo.m para añadir ahora la nueva función creada y así dibujar las dos funciones a la vez

```
x=linspace(-6,6,100);

y=f(x);z=g(x);%añadimos el nombre de la segunda función

plot(x,y,x,z,'g')
```

>> dibujo%ejecutamos en matlab



%como vemos nos ha dibujado las dos funciones, siendo la azul f(x)=xsen(x) y la verde f(x)=xcos(x)

2. PROGRAMA PARA EL DIBUJO DE UNA FUNCIÓN

Haremos un programa en el que pondremos una función y nos la dibuje, sin tener que estar modificando el programa

%creamos el fichero en matlab dibujogeneral.m

>> edit dibujogeneral.m

```
function[ ]=dibujogeneral(f,a,b,n)%creamos el fichero
x=linspace(a,b,n);%ponemos los comandos para poder dibujar
y=feval(f,x);
plot(x,y)
```

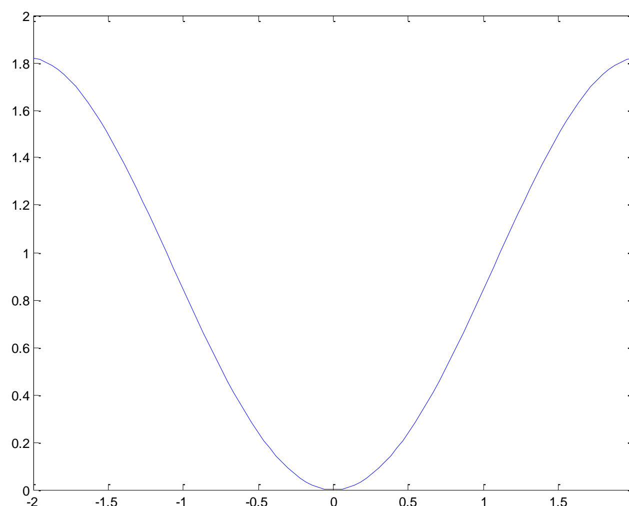
%definimos la función que queremos dibujar, que es f(x)=xsen(x)

>> edit fun.m

```
function y=fun(x)

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

% introducimos en matlab los parámetros y la función que quermos dibujar

>> dibujogeneral('fun',-2,2,100) % se dibuja al darle a enter



3. PROGRAMA PARA EL CÁLCULO DE RAÍCES

Vamos a emplear dos métodos para el cálculo de raíces. Uno es el método de bisección y otro es el método de Newton-Raphson.

MÉTODO DE BISECCIÓN:

En matemática, el método de bisección es un algoritmo de búsqueda de raíces que trabaja dividiendo el intervalo a la mitad y seleccionando el subintervalo que tiene la raíz.

Supóngase que queremos resolver la ecuación $f(x) = 0$ (donde f es continua. Dados dos puntos a y b tal que $f(a)$ y $f(b)$ tengan signos distintos, sabemos por el Teorema de Bolzano que f debe tener, al menos, una raíz en el intervalo $[a, b]$. El método de bisección divide el intervalo en dos, usando un tercer punto $c = (a+b) / 2$. En este momento, existen dos posibilidades: $f(a)$ y $f(c)$, ó $f(c)$ y $f(b)$ tienen distinto signo. El algoritmo de bisección se aplica al subintervalo donde el cambio de signo ocurre.

El método de bisección es menos eficiente que el método de Newton, pero es mucho más seguro asegurar la convergencia.

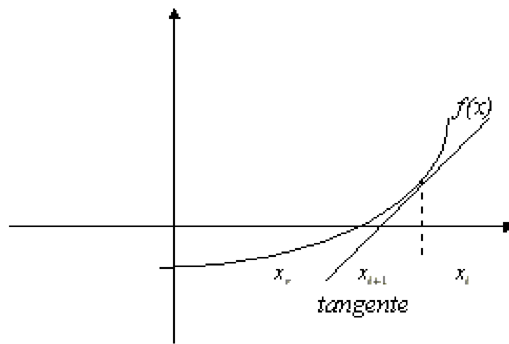
Si *f* es una función continua en el intervalo [*a*, *b*] y *f*(*a*)*f*(*b*) < 0, entonces este método converge a la raíz de *f*. De hecho, una cota del error absoluto es:

$$\frac{|b-a|}{2^n}$$

en la *n*-ésima iteración. La bisección converge linealmente, por lo cual es un poco lento. Sin embargo, se garantiza la convergencia si f(*a*) y f(*b*) tienen distinto signo.

MÉTODO DE NEWTON-RAPHSON:

Este método, el cual es un método iterativo, es uno de los más usados y efectivos. A diferencia de los métodos anteriores, el método de Newton-Raphson no trabaja sobre un intervalo sino que basa su fórmula en un proceso iterativo. Supongamos que tenemos la aproximación x_i a la raíz x_r de $f(x)$,



Trazamos la recta tangente a la curva en el punto $(x_i, f(x_i))$; ésta cruza al eje x en un punto x_{i+1} que será nuestra siguiente aproximación a la raíz x_r.

Para calcular el punto x_{i+1}, calculamos primero la ecuación de la recta tangente. Sabemos que tiene pendiente

$$m = f'(x_i)$$

Y por lo tanto la ecuación de la recta tangente es:

$$y - f(x_i) = f'(x_i)(x - x_i)$$

Hacemos $y = 0$:

$$-f(x_i) = f'(x_i)(x - x_i)$$

Y despejamos x :

$$x = x_i - \frac{f(x_i)}{f'(x_i)}$$

Que es la fómula iterativa de Newton-Raphson para calcular la siguiente aproximación:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}, \text{ si } f'(x_i) \neq 0$$

Note que el método de Newton-Raphson no trabaja con intervalos donde nos asegure que encontraremos la raíz, y de hecho no tenemos ninguna garantía de que nos aproximaremos a dicha raíz. Desde luego, existen ejemplos donde este método no converge a la raíz, en cuyo caso se dice que el método diverge. Sin embargo, en los casos donde si converge a la raíz lo hace con una rapidez impresionante, por lo cual es uno de los métodos preferidos por excelencia.

También observe que en el caso de que $f'(x_i) = 0$, el método no se puede aplicar. De hecho, vemos geométricamente que esto significa que la recta tangente es horizontal y por lo tanto no intersecta al eje x en ningún punto, a menos que coincida con éste, en cuyo caso x_i mismo es una raíz de $f(x)$!

%creamos el fichero en matlab biseccion.m

>> edit biseccion.m

```
function p=biseccion(f,a,b,TOL) %creamos el programa, a y b son dos puntos cercanos a la raíz, f es la función y TOL es la tolerancia

while abs(a-b)>TOL

p=(a+b)/2;

if feval(f,p)==0 return %significa que termina el while y retorna

end

if feval (f,a)*feval(f,p)<0

   b=p;
```

```
else
   a=p;
end
end
p=(a+b)/2;
```

Para aplicar el método de bisección primero tenemos que definir una función, luego dibujarla para sacar el parámetro a y b, y una vez definida la función, podemos llamar al programa y ya nos da el resultado.

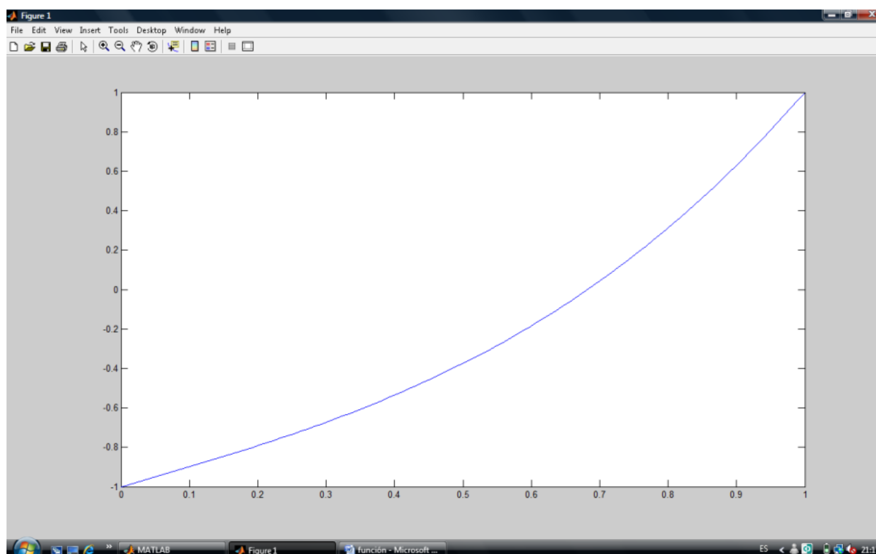
Ejemplo 1: Sacar las raíces de la función $f(x)=x^3+x-1$

\>> edit f1.m % creamos el fichero de la función, la cual la hemos llamado f1

```
function y=f1(x)
y=x.^3+x-1;
```

%utilizamos el programa dibujo general ya creado para dibujarla y ver donde puede estar la raíz para los parámetros a y b

\>> dibujogeneral('f1',0,1,100)



>> biseccion('f1',0,1,10^(-6))% aplicamos el programa bisección, desde 0 a 1, que como vemos en el dibujo es por donde se encuentra la raíz

ans =

0.68232774734497

>> f1(0.68232774734497)·% para comprobar que es una raíz, vemos que al sustituir el valor anterior en la función es próximo a cero, y es así

ans =

-1.353736912568238e-007

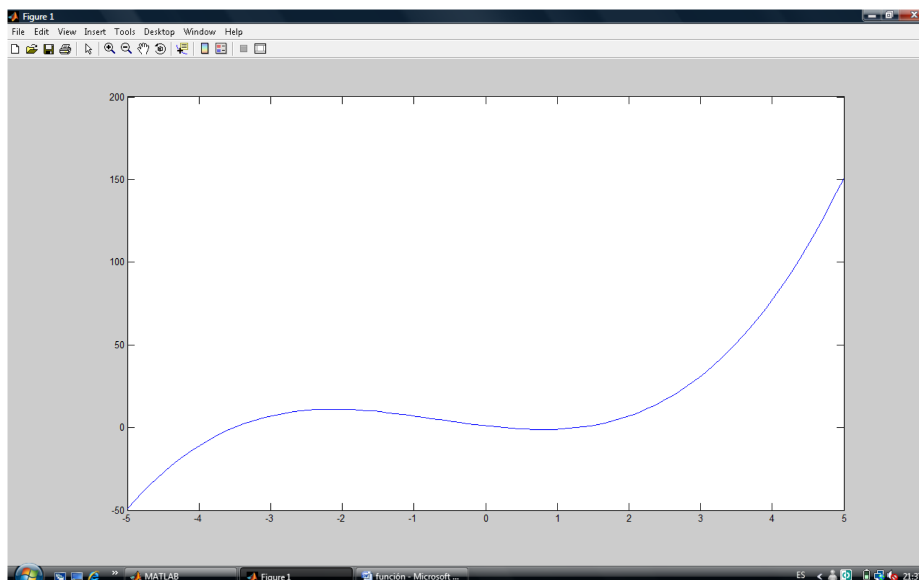
Ejemplo 2: calcular las raíces de la función $f(x)=x^3+2x^2-5x+1$

>> edit f1.m % creamos el fichero de la función y le ponemos el nombre f1

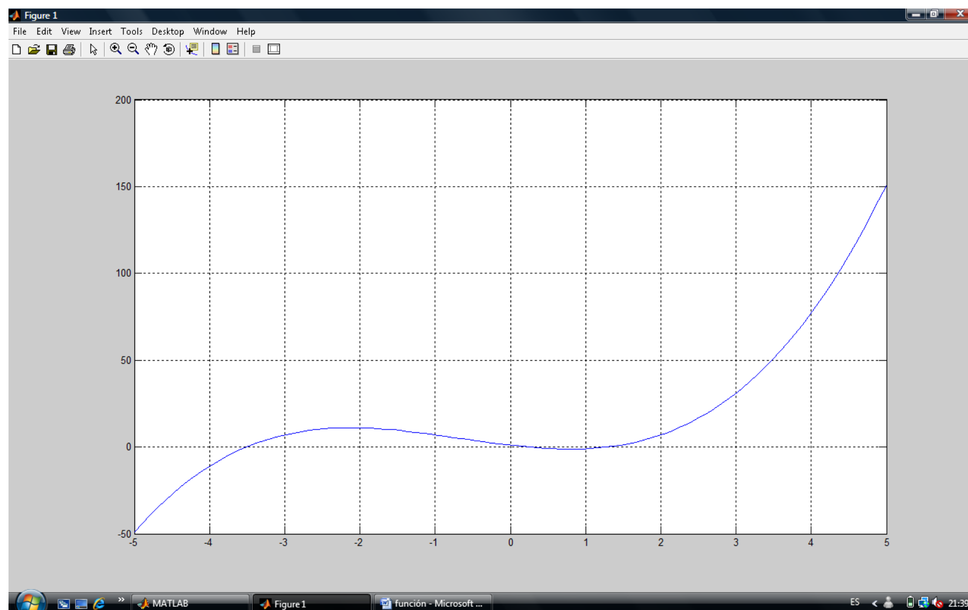
```
function y=f1(x)

y=x.^3+2.*x.^2-5.*x+1;
```

%dibujamos la función para ver donde se encuentra la raiz

>> dibujogeneral('f1',-5,5,100)



>> grid %le ponemos una malla para así ver mejor donde está la raiz



>> biseccion('f1',-5,-3,10^(-6)) %aplicamos el método de bisección entre -5 y -3 que es donde se encuentra una raíz

ans =

  -3.50701856613159

>> biseccion('f1',-1,0.5,10^(-6)) %aplicamos el método de bisección entre -1 y 0.5 que es donde se encuentra otra raíz

ans =

   0.22187602519989

>> biseccion('f1',0.5,2,10^(-6)) %aplicamos el método de bisección entre 0.5 y 2 que es donde se encuentra otra raíz

ans =

   1.28514230251312

%estos tres valores los sustituimos en la función para comprobar que son raíces

>> f1(-3.50701856613159)

ans =

    1.393128190585458e-006

```
>> f1(0.22187602519989)

ans =

   5.434297573048141e-007

>> f1(1.28514230251312)

ans =

  -9.136798224673726e-007
```

% como vemos todos los valores son muy próximos a 0, por lo tanto las tres son raíces

%creamos el fichero en matlab newtonsol.m

Es un programa para encontrar raíces de una función, pero como argumentos de entrada tenemos que poner la función, la derivada de esa función, y un intervalo aproximado del punto donde está la raíz. Por ello, lo primero que debemos de hacer es dibujar la función para buscar el punto aproximado.

```
>>edit newtonsol.m
```

```
function x1=newtonsol(f,fprime,x0,TOL)% creamos el programa para el cálculo de raíces, aquí añadimos la función, la deribada de la función y un valor que es aproximado a la raíz, este valor se ve con el dibujo

while abs(feval(f,x0))>TOL

   x0=x0-feval(f,x0)/feval(fprime,x0);

end

x1=x0;
```

% Como el método de Newton necesita tener definida la función y la derivada de la función, podemos editar un programa que defina la derivada de la función dada. Si la función es compleja como para calcular la derivada fácilmente de cabeza, podemos utilizar los siguientes comandos de Matlab:

>>syms x

>>f=x*sin(x);

>>diff(f)

% Una vez editado el método Newton, para aplicarlo ponemos:

>> newtonsol('fun','funp',0.75,10^(-15)) % El 0.75 es el valor aproximado de la raíz (que lo encontramos al dibujar previamente la función); y el dato 10^(-15) corresponde al número de decimales que queremos obtener.

ans =

0.75487766624669

% Para comprobar que el programa no falla, podemos calcular la función de la raiz que hemos obtenido con el método Newton, y si el número que nos da es muy cercano a 0, podemos verificar que el método es bastante aproximado.

>> fun(0.75487766624669)

ans =

-9.103828801926284e-015

% como vemos, el valor que obtenemos es bastante próximo a 0.

Ejercicio: sacar las raíces de la función f(x)=xsen(x)-cos(x) por los dos métodos anteriores que hemos propuesto en el intervalo [-2π, 2π]

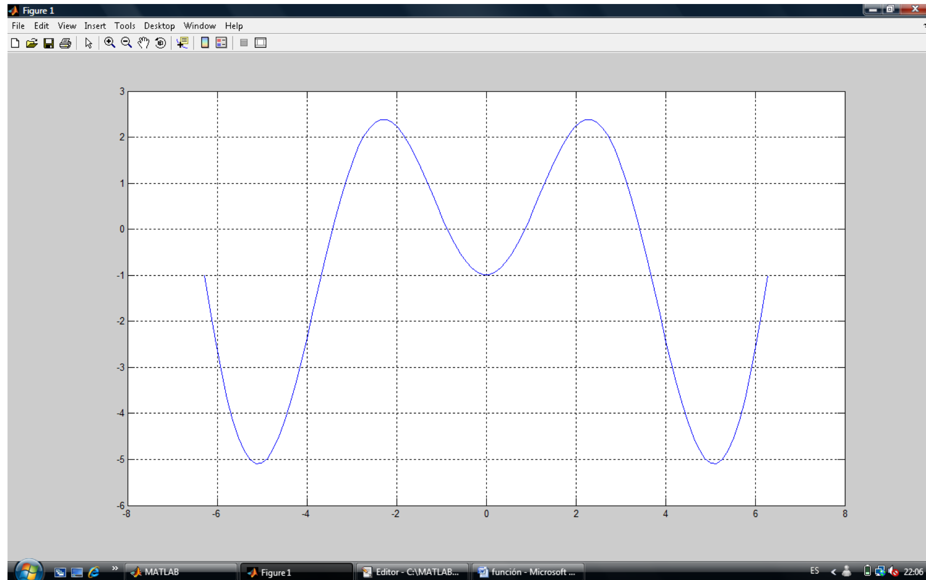
% primero creamos el fichero en matlab de la función a la que llamaremos fta.m

>> edit fta.m

```
function y=fta(x)

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

>> dibujogeneral('fta',-2*pi,2*pi,100)%ejecutamos el programa para dibujarla en el intervalo que piden

>> grid%le ponemos la malla para ver mejor donde se encuentran las raices



>> biseccion('fta',-4,-2.5,10^(-9))%ejecutamos el programa de bisección para que nos de las raíces, como tiene dos primero que nos de la raíz que está entre -4, -2.5

ans =

-3.42561845981982

>> biseccion('fta',2.5,4,10^(-9))%para que nos de la raíz que está entre 2.5, 4

ans =

3.42561845981982

% para saber la raicez con el método de newton, tenemos que crear el fichero de la función derivada

>> edit ftaprima.m

```
function y=ftaprima(x)

y=sin(x)+x.*cos(x)-sin(x);
```

```
>> newtonsol('fta','ftaprima',2*pi,10^(-9))%ejecutamos el programa para el método de newton, con el valor aproximado de una de las raices

ans =

    6.4373


>> newtonsol('f','fprima',-4,10^(-9))% añadimos el otro valor próximo a la otra raíz para ejecutar el programa

ans =

   -3.1416
```

4. PROGRAMA PARA EL AJUSTE EXPONENCIAL

Este programa sirve para muchos problemas. Por ejemplo, si tenemos una nube de puntos y queremos saber cuál es la curva exponencial que mejor se adapta.

```
%creamos el fichero ajusteexp.m

>>edit ajusteexp.m
```

```
function [a,b]=ajusteexp(x,y); % los valores de x e y son los valores de la nube de puntos para asi poderla dibujar

Y=log(y);

p=polyfit(x,Y,1);

a=exp(p(2));

b=p(1);

m=length(x);

xx=linspace(x(1),x(m),100);

yy=a.*exp(b.*xx);

plot(xx,yy,x,y,'og');% ponemos la herramienta para que nos la dibuje
```

%Para aplicar este programa, primero tenemos que definir x e y:

>>x = [ ]

>>y = [ ]

% Como el programa tiene más de un argumentos de salida, el programa por defecto sólo nos da el parámetro a, por ello tenemos que especificar a la hora de aplicar el programa los argumentos de salida que queremos obtener:

>>[a b]=ajusteexp(x,y)

%Con esto obtenemos los dos parámetros, pero además, también obtenemos la gráfica de la función.

5. PROGRAMA PARA EL AJUSTE RACIONAL

Este programa también sirve para muchos problemas, pero el ajuste de la curva se ajusta a una función racional.

%creamos el fichero en matlab que lo llamamos ajusteracional.m

>>edit ajusteracional.m

```
function [a,b]=ajusteracional(x,y) % para aplicar las x y las y y te de la nube de valores y asi dibujarla

Y=y;

X=(1/(x-1));

p=polifit(X,Y,1);

B=(p(1));

A=p(2);

a=a;

b=B-a;
```

```
m=length(x);

xx= linspace (x(1),x(m),100);

yy=(a.*xx+b)./(xx-1);

plot (xx,yy,x,y,'og')% dibujamos la función
```

6. INTERPOLACIÓN

En el subcampo matemático del análisis numérico, se denomina **interpolación** a la construcción de nuevos puntos partiendo del conocimiento de un conjunto discreto de puntos.

En ingeniería y algunas ciencias es frecuente disponer de un cierto número de puntos obtenidos por muestreo o a partir de un experimento y pretender construir una función que los ajuste.

Otro problema estrechamente ligado con el de la interpolación es la aproximación de una función complicada por una más simple. Si tenemos una función cuyo cálculo resulta costoso, podemos partir de un cierto número de sus valores e interpolar dichos datos construyendo una función más simple. En general, por supuesto, no obtendremos los mismos valores evaluando la función obtenida que si evaluásemos la función original, si bien dependiendo de las características del problema y del método de interpolación usado la ganancia en eficiencia puede compensar el error cometido.

En todo caso, se trata de, a partir de n parejas de puntos (x_k,y_k), obtener una función f que verifique

$$f(x_k) = y_k \ , \ k = 1, \ldots, n$$

a la que se denomina función interpolante de dichos puntos. A los puntos x_k se les llama nodos. Algunas formas de interpolación que se utilizan con frecuencia son la interpolación lineal, la interpolación polinómica (de la cual la anterior es un caso particular), la interpolación por medio de spline o la interpolación polinómica de Hermite.

```
>> x=[1,1.5,1.8,2,2.5]; % añadimos los valores de la x

>> y=[1.3,1.6,0.8,1,3]; %añadimos los valores de la y

>> p=polyfit(x,y,4) % nos da los coeficientes del polinomio que mejor ajusta al dibujo de los valores de x e y
```

p =

1.0e+002 *

Columns 1 through 4

-0.09642857142855   0.72166666666652 -1.93960714285677   2.21058333333294

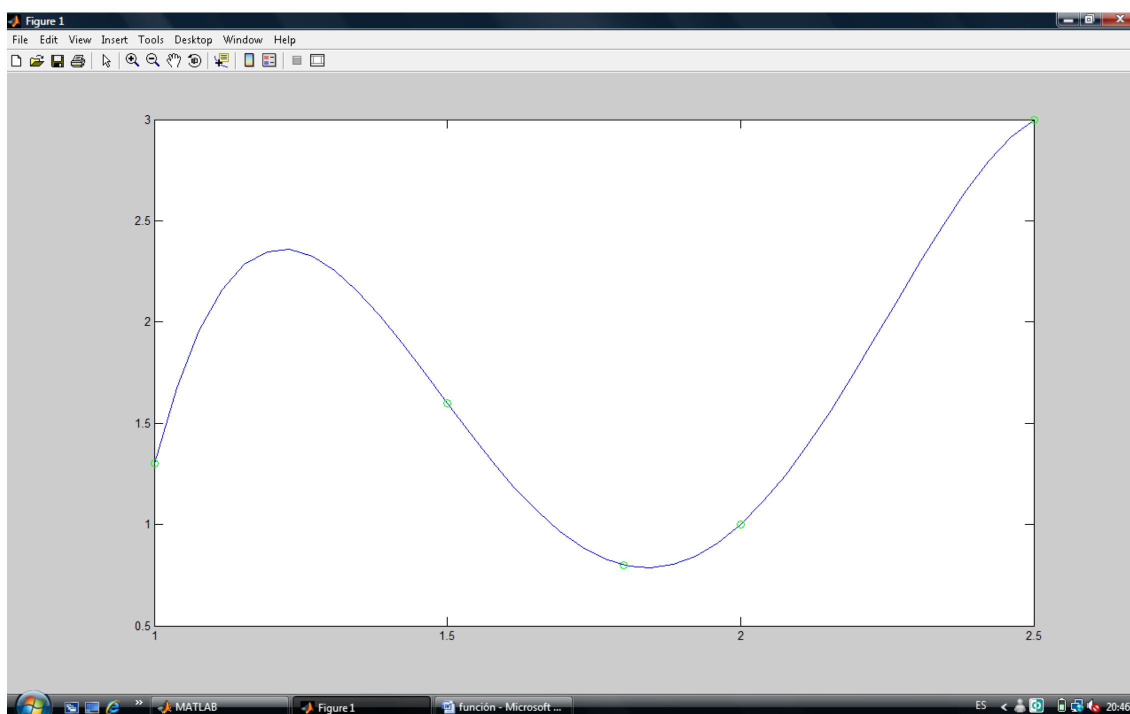
Column 5

-0.88321428571414

>> xx=linspace(1,2.5,40); %con estos comandos dibujamos la función

>> yy=polyval(p,xx);

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



>> polyval(p,1.2) % añadimos un valor de x y nos da el valor de y correspondiente en este caso lo hacemos para x=1.2

ans =

2.35371428571408

>> polyval(p,1.37)% buscamos el valor del polinomio cuando x=1.37

ans =

2.08018847499986

Ejemplo 1: encontrar el polinomio más acertado para los siguientes valores de x e y :

x=[0,0.5,1,1.5,2]; y=[0.1,0.7,-1,0.2,0.7]

%añadimos al programa los valores de x e y

>> x=[0,0.5,1,1.5,2];

>> y=[0.1,0.7,-1,0.2,0.7];

>> p=polyfit(x,y,4)% añadimos el comando para que nos de los coeficientes del polinomio que mejor se ajusta

p =

Columns 1 through 4

-5.86666666666671 24.53333333333350 -31.13333333333351 11.36666666666671

Column 5

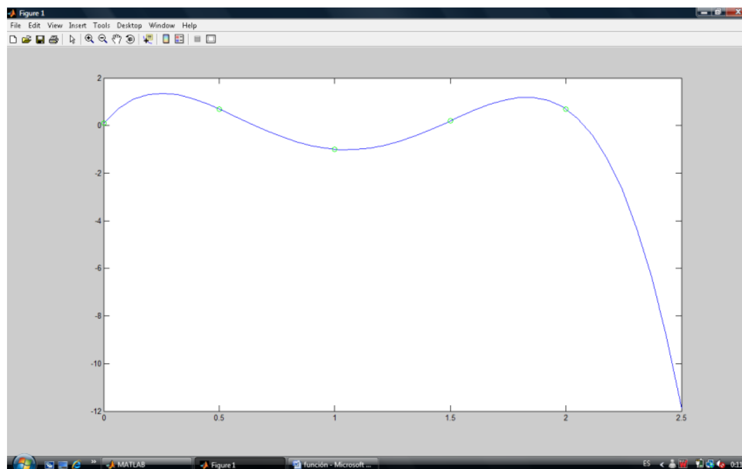
0.10000000000001

%dibujamos nuestros puntos

>> xx=linspace(0,2.5,40);

>> yy=polyval(p,xx);

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



```
>> polyval(p,0.7)%le pedimos que nos de el valor del polinomio cuando x=0.7

ans =

  -0.19232000000000


>> polyval(p,1.2)%cuando x=1.2

ans =

  -0.86352000000000


>> polyval(p,1.7)%cuando x=1.7

ans =

   0.98128000000000
```

7. AJUSTE POR MÍMINOS CUADRADOS

Mínimos cuadrados es una técnica de optimización matemática que, dada una serie de mediciones, intenta encontrar una función que se aproxime a los datos (un "mejor ajuste").

Intenta minimizar la suma de cuadrados de las diferencias ordenadas (llamadas residuos) entre los puntos generados por la función y los correspondientes en los datos. Específicamente, se llama mínimos cuadrados promedio (LMS) cuando el número de datos medidos es 1 y se usa el método de descenso por gradiente para minimizar el residuo

cuadrado. Se sabe que LMS minimiza el residuo cuadrado esperado, con el mínimo de operaciones (por iteración). Pero requiere un gran número de iteraciones para converger.

Un requisito implícito para que funcione el método de mínimos cuadrados es que los errores de cada medida estén distribuidos de forma aleatoria. El teorema de Gauss-Markov prueba que los estimadores mínimos cuadráticos carecen de sesgo y que el muestreo de datos no tiene que ajustarse, por ejemplo, a una distribución normal. También es importante que los datos recogidos estén bien escogidos, para que permitan visibilidad en las variables que han de ser resueltas (para dar más peso a un dato en particular, véase mínimos cuadrados ponderados).

La técnica de mínimos cuadrados se usa comúnmente en el ajuste de curvas. Muchos otros problemas de optimización pueden expresarse también en forma de mínimos cuadrados, minimizando la energía o maximizando la entropía

%creamos el fichero en matlab el cual lo llamamos EMC.m

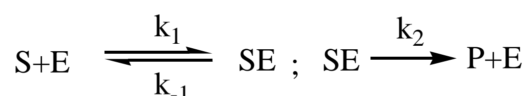
>>edit EMC.m

```
function z=EMC(x,y)% asi al ejecutarlo en matlab añadimos los valores de x e y asi nos da el ajuste por minimos cuadrados

m=length(x);

p=polyfit(x,y,1);

s=0

for i=1:m

   s=s+(polyval(p,x(i)-y(i))^2);

end

E=sqrt(s)
```

8. CINÉTICA DE REACCIONES QUÍMICAS

Utilizaremos un modelo básico que es el propuesto por Michaelis y Menten (1913) para reacciones encimáticas. La cinética de Michaelis-Menten describe la velocidad de reacción de muchas reacciones enzimáticas. Su nombre es en honor a Leonor Michaelis y Maud Menten. Este modelo solo es válido cuando la concentración del sustrato es mayor que

la concentración de la enzima, y para condiciones de estado estacionario, o sea que la concentración del complejo enzima-sustrato es constante. Estas reacciones constan:



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 s, e, c y p las concentraciones de los reactantes S, E, SE y P respectivamente. Dicha ley nos proporciona una ecuación para cada una de las sustancias involucradas en la reacción:

$$\frac{ds}{dt} = -k_1 es + k_{-1} c$$
$$\frac{de}{dt} = -k_1 es + \left(k_{-1} + k_2\right) c$$
$$\frac{dc}{dt} = k_1 es - \left(k_{-1} + k_2\right) c$$
$$\frac{dp}{dt} = k_2 c$$

Conocidas las concentraciones iniciales de los sustratos, hay que ver cómo evoluciona el sistema con el tiempo. Para resolver el sistema con matlab debemos de poner las condiciones iniciales que son s(0)=1.5, e(0)=2.7, c(0)=0 y p(0)=0 y a las constantes K_1, K_{-1} y K_2 le damos los valores de 2, 1 y 3 respectivamente.

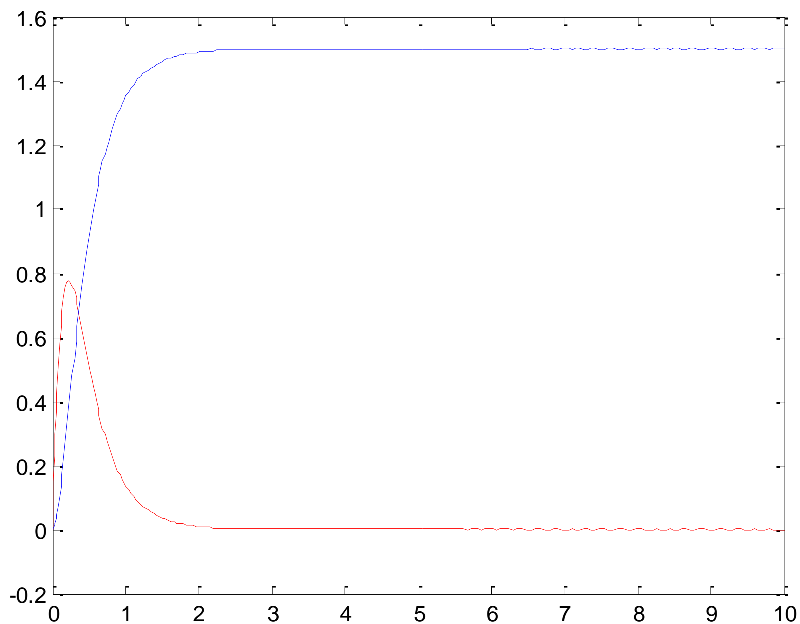
% Creamos en matlab el fichero cinética.m

>>edit cinetica.m

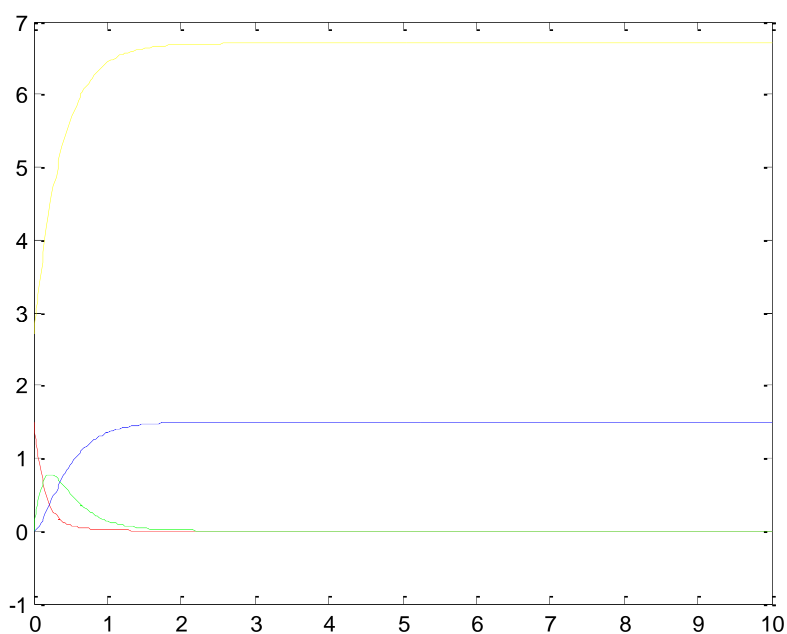
```
function z=cinetica(t,yy) % ponemos nombre de cinetica a la función

s=yy(1); e=yy(2); c=yy(3); p=yy(4); % indicamos las cuatro ecuaciones

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';
```

>>[t,y]=ode45('cinetica',[0,10],[1.5,2.7,0,0]); % añadimos las condiciones iniciales

>>plot (t,y(:,3),'r',t,y(:,4),'b')% dibujamos la variación de la concentración de c y p que son de SE y P respectivamente

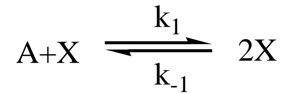


>>plot (t,y(:,1),'r',t,y(:,2),'y',t,y(:,3),'g',t,y(:,4),'b') % añadimos al dibujo las concentraciones de los sustratos



9. AUTOCATÁLISIS: ACTIVACIÓN E INHIBICIÓN

El modelo autocatalítico más simple posible es el representado por la ecuación:



La concentración de A permanece constante e igual a a, puesto que se gasta muy poco, la ley de acción de masas aplicada a esta ecuación nos conduce a las expresiones:

$$\frac{da}{dt} = -k_1 ax + k_{-1}x^2 = 0 (a = cons\tan te)$$

$$\frac{dx}{dt} = k_1 ax - k_{-1}x^2$$

Por lo tanto, nos quedamos solo con la ecuación:

$$\frac{dx}{dt} = k_1 ax - k_{-1}x^2$$

Para resolver el sistema con matlab debemos de poner las condiciones iniciales de las concentraciones de los sustratos que son x(0)=5 y a(0)=3 y a las constantes K_1 y K_{-1} le damos los valores de 2 y 1 respectivamente.

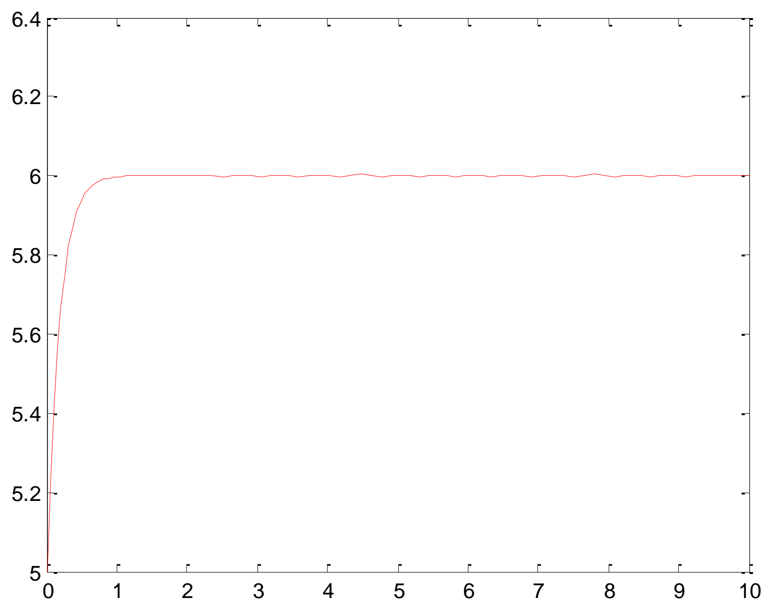
% Creamos en matlab el fichero autocatalisis.m

>>edit autocatalisis.m

```
function z=autocatalisis(t,x) % ponemos nombre de autocatalisis a la función

 z=(6.*x)-x.^2; % indicamos la ecuación, ya que solo tenemos una
```

>>[t,x]=ode45('autocatalisis',[0,10],[5]); % añadimos la condición inicial para la concentración de x

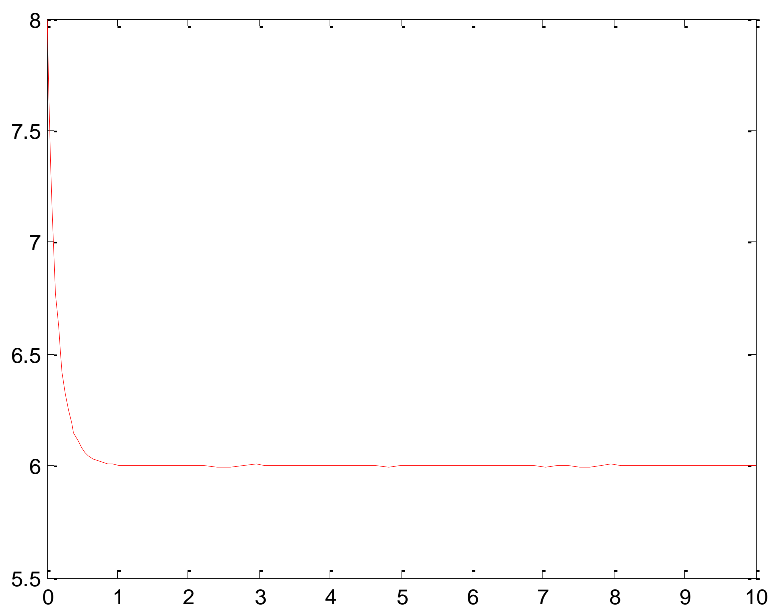
>>plot (t,x,'r') % dibujamos la variación de la concentración de x



%cambiamos la condición inicial para la concentración de x, siendo esta ahora de 8. Volvemos a proceder de la misma forma

>>[t,x]=ode45('autocatalisis',[0,10],[8]);   % añadimos la nueva condición inicial para la concentración de x

>>plot (t,x,'r') % dibujamos la variación de la concentración de x con esta nueva condición inicial



Como conclusión podemos decir que la reacción tiende a estabilizarse a medida que se acerca a 6.

10. OSCILADORES QUÍMICOS BIOLÓGICOS. HISTORIA Y ECUACIONES.

Muchas de los sistemas biológicos son de tipo oscilatorio bien sea con periodos de horas, minutos, segundos o, incluso, semanas. La reacción de este tipo fue descubierta por Belousov, cuyos trabajos fueron continuados por Zhabotinski (1964). Por eso este tipo de reacciones oscilantes son conocidas como **la reacción de Belousov-Zhabotinsky.**

Definición de reacción química oscilante

El concepto básico de reacción química nos dice que ésta consiste en una serie de sustancias químicas llamadas reactivos que puestas en contacto reaccionan entre sí dando lugar a los productos. Esperando un tiempo necesario, entre reactivos y productos se alcanza un equilibrio donde la relación de concentraciones entre reactivos y productos permanece constante. Es decir, en general solo un cierto porcentaje de reactivos se convierten en productos, de manera que en el equilibrio tenemos una mezcla de reactivos y productos.

En el caso de la reacción oscilante nos encontramos en una situación fuera del equilibrio, es decir antes de que pase el tiempo necesario para llegar al equilibrio, donde la mezcla reactiva oscila entre contenter prácticamente solo reactivos y prácticamente solo productos.

Si las oscilaciones son periódicas nos encontramos en el régimen regular. En caso contrario nos encontramos en régimen caótico.

Historia

El descubrimiento del fenómeno se le acredita a Boris Belousov, quien se dio cuenta en la década de los 50 (los datos cambian dependiendo de la fuente pero contenidos en un rango de 1951 a 1958), que en una mezcla de bromato potásico, sulfato de cerio (IV), ácido malónico y ácido cítrico, la concentración de los iones Ce(IV) y Ce(III) oscilaba, notándose esto mediante la oscilación de color de la reacción de un color amarillo a incoloro. Esto es debido a que los iones de Ce (IV) son reducidos por el ácido malónico a Ce(III), que son oxidados de nuevo a Ce(IV) por los ions de bromo (V).

Belousov hizo dos intentos de publicar su hallazgo, pero fue rechazado al no ser capaz de explicar sus resultados de forma que satisficieran a los editores de las revistas en las que lo presentó. Su trabajo fue publicado finalmente en una revista menos respetable.

Más tarde, en 1961, un estudiante llamado A. M. Zhabotinsky redescubrió la secuencia de esta reacción, aunque los resultados de su trabajo no fueron ampliamente diseminados, y él no era conocido hasta una conferencia en Praga en 1968.

Diferentes reacciones de Belousov-Zhabotinsky

La reacción de BZ es en esencia una reacción redox en la que se oxida el ácido malónico por bromatos en un medio ácido.

Constituye una mezcla reactiva muy compleja:

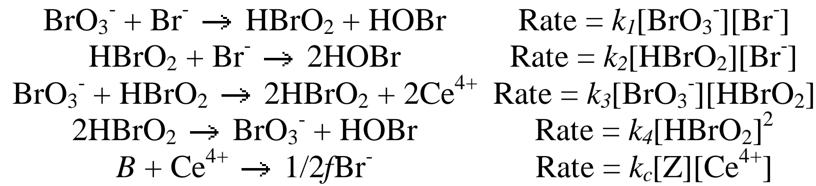
- Ácido malónico
- Ácido inorgánico, que puede ser el sulfúrico
- Una sal que aporte iones bromato (BrO_3^-)
- Una sal que aporte iones bromuro (Br^-)
- Una sal de hierro (Fe^{2+})
- Una solución acuosa

Precisemos que un ion es un átomo o grupo de átomos cargados eléctricamente y que sus cargas son el resultado de la ganancia o pérdida de uno o más electrones. Cuando un ion se oxida pierde electrones y cuando se reduce los gana. Se utiliza un indicador redox llamado ferroina, así, los iones de hierro en el estado reducido (Fe^{2+}) dan una coloración roja, y al oxidarse se convierten en Fe^{3+} (con ferroina), de coloración azul.

Para realizar la medida más precisa de las oscilaciones y poder comprobar su carácter caótico, se emplean electrodos conectados a un dispositivo capaz de medir las diferencias de potencial electrostático (un voltíemtro). Dichas referencias de potencial están relacionadas con las concentraciones a través de la ecuación de Nerst, de manera que las oscilaciones en potencial son equivalentes en cuanto al comportamiento caótico se refiere, a las concentraciones de las diferentes especies químicas implicadas en la reacción.

En la reacción oscilante nos encontramos en una situación fuera del equilibrio, es decir antes de que pase el tiempo necesario para llegar al equilibrio, donde la mezcla reactiva oscila entre contenter prácticamente solo reactivos y prácticamente solo productos.

Si las oscilaciones son periódicas nos encontramos en el régimen regular. En caso contrario nos encontramos en régimen caótico. UN ejemplo de reacción oscilante aplicada a matlab es:



$BrO_3^- + Br^- \rightarrow HBrO_2 + HOBr$	Rate = $k_1[BrO_3^-][Br^-]$
$HBrO_2 + Br^- \rightarrow 2HOBr$	Rate = $k_2[HBrO_2][Br^-]$
$BrO_3^- + HBrO_2 \rightarrow 2HBrO_2 + 2Ce^{4+}$	Rate = $k_3[BrO_3^-][HBrO_2]$
$2HBrO_2 \rightarrow BrO_3^- + HOBr$	Rate = $k_4[HBrO_2]^2$
$B + Ce^{4+} \rightarrow 1/2fBr^-$	Rate = $k_c[Z][Ce^{4+}]$

Siendo las concentraciones de cada especie las respectivas letras:

A	BrO_3^-
B	All oxidizable organic species
P	HOBr
X	$HBrO_2$
Y	Br^-
Z	Ce^{4+}

Se puede presentar estas ecuaciones como un sistema de ecuaciones diferenciales:

$$\frac{dX}{dt} = k_1 AY - k_2 XY + k_3 AX - 2k_4 X^2$$

$$\frac{dY}{dt} = -k_1 AY - k_2 XY + \frac{1}{2} f k_c BZ$$

$$\frac{dZ}{dt} = 2k_3 AX - k_c BZ$$

Las constantes han sido determinadas experimentalmente y sus valores son:

k_1	$1.28M^{-1}s^{-1}$
k_2	$2.4 \times 10^6 M^{-1}s^{-1}$
k_3	$33.6M^{-1}s^{-1}$
k_4	$3 \times 10^3 M^{-1}s^{-1}$
k_c	$1M^{-1}s^{-1}$

Para adimensionalizar las ecuaciones diferenciales anteriores, debemos de tener en cuenta las siguientes igualdades:

$$x = \frac{2k_4 X}{k_3 A};\quad y = \frac{k_4 Y}{k_3 A};\quad z = \frac{k_c k_4 BZ}{(k_3 A)^2};\quad \tau = k_c Bt$$

$$\varepsilon = \frac{k_c B}{k_3 A};\quad \varepsilon' = \frac{2k_c k_4 B}{k_2 k_3 A};\quad q = \frac{2k_1 k_4 B}{k_2 k_3 A}$$

Quedando las ecuaciones finales:

$$\frac{dx}{d\tau} = \frac{qy - xy + x(1-x)}{\varepsilon}$$

$$\frac{dy}{d\tau} = \frac{-qy - xy + fz}{\varepsilon'}$$

$$\frac{dz}{d\tau} = x - z$$

Las condiciones iniciales para las concentraciones de cada especie son de 0.06M para A y de 0.02M para B y el factor estequiométrico f es 1.

% Creamos en matlab el fichero oscilacion.m

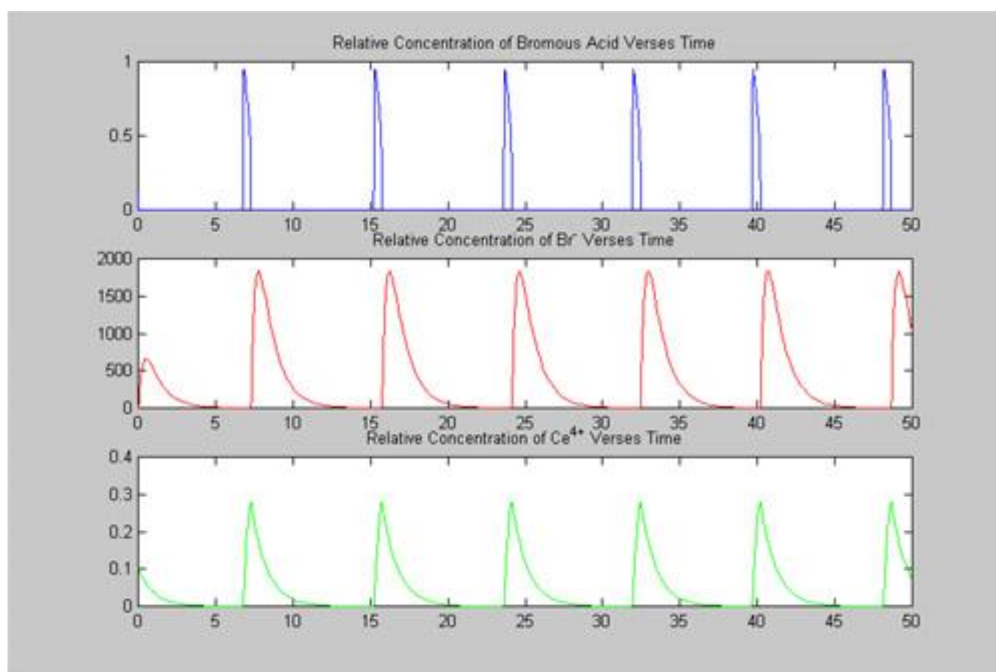
>>edit oscilacion.m

```
function d=oscilacion(t,yy) % ponemos nombre de oscilacion a la función

x=yy(1); y=yy(2); z=yy(3); % indicamos las tres ecuaciones

d(1)=(3.17.*10.^(-5).*y-x.*y +x.*(1-x))/(9.92.*10.^(-3));

d(2)=(-3.17.*10.^(-5).*y-x.*y +1.*z)/(2.48.*10.^(-5));

d(3)=x-z;

d=d';
```

>>[t,y]=ode45('oscilacion',[0,10],[0,0,0]); % añadimos las condiciones iniciales para las concentraciones de x, y y z

>>plot (t,y(:,1),'r',t,y(:,2),'y',t,y(:,3),'g') % dibujamos las concentraciones de los sustratos que se reprsentan por x, y y z

% vemos el comportamiento oscilatorio en la representación



11. MECANISMO PROPUESTO POR 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 ~ X^2Y), puede soslayarse tal dificultad introduciendo una tercera variable que se elimina adiabáticamente . En todo caso sus ecuaciones diferenciales son:

$$\dot{X} = k_1 - k_2X - k_3X + k_4X^2Y$$
$$\dot{Y} = k_5X - k_6X^2Y$$

Tiene estado estacionario en

$$X_0 = \frac{k_1}{k_2 + k_3 - \frac{k_4k_5}{k_6}}, Y_0 = \frac{k_5}{k_6k_1}(k_2 + k_3 - \frac{k_4k_5}{k_6})$$

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:

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

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 $T_0 = B - 1 - A^2$, y $Det_0 = A^2 > 0$. Para $B > 1 + A^2$, se tiene $T_0 > 0$, y hay un ciclo límite rodeando al estado estacionario inestable.

Teniendo en cuenta la difusión, el sistema anterior se escribe

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

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 $T_n = B - 1 - A^2 - (D_x + D_y)n^2$, y cuyo determinante es:

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

Se puede tener inestabilidad del estado estacionario homogéneo, desde luego, si $T_n > 0$, o sea si $B > B_1(n) = 1 + A^2 + (D_x + D_y)n^2$. El valor crítico B_1 depende de n; su valor mínimo se alcanza ($dB_1/d_n = 0$) cuando n = 0 (fluctuación homogénea), y vale $B_1min = 1 + A^2$, 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 $Det_n < 0$, lo que implica que

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

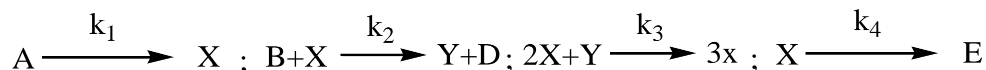
el mínimo valor del límite crítico B_2 se obtiene ($dB_2/d_n = 0$) para

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

Es obvio que si $D_x = D_y$, será $B_{1min} = 1 + A^2 < B_{2min} = (1 + A)^2$, 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 $B_2 > B > B_1$ implica $T_0 > 0$). Pero si $D_y >> D_x$, entonces puede ocurrir que $B_{2min} < B_{1min}$ (en el caso extremo $(D_x/D_y) \rightarrow 0$, $B_{2min}= 1 < B_{1min}= 1 + A^2$), y al aumentar B acaece antes la bifurcación en B_{2min}, y aparece (orden a través de fluctuaciones) una estructura disipativa correspondiente a $n^2 = (A^2/D_xD_y)^{1/2}$.

Por ejemplo, si A = 2, B = 2 ($T_0 = -3$, $Det_0 = 4 > 0$, $\Delta_0 = -7 < 0$, el estado estacionario homogéneo era un foco estable), para $D_x = 1$, $D_y = 25$, el valor de p = 0,4 satisface todas las condiciones, haciendo $Det_n = -0,4$; la parte real de ω_n será 0,0596/2, mayor que la de ω_o, que sería $T_0/2 < 0$.

Las ecuaciones propuestas por este mecanismo son:



$$A \xrightarrow{k_1} X \;;\; B+X \xrightarrow{k_2} Y+D \;;\; 2X+Y \xrightarrow{k_3} 3x \;;\; X \xrightarrow{k_4} E$$

% Creamos en matlab el fichero bruselator.m

>>edit bruselator.m

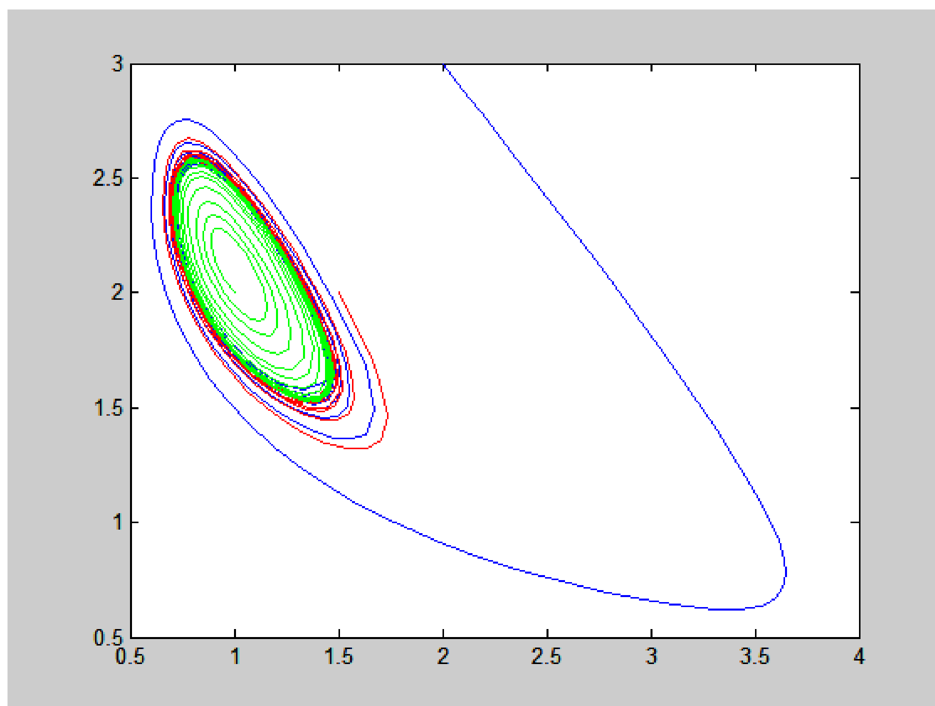
```
function z=bruselator(t,yy)% ponemos nombre de oscilacion a la función

X=yy(1),Y=yy(2); % indicamos las tres ecuaciones

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';
```

>>[t,y]=ode45('bruselator',[0,20],[0.5;2.2]);%ponemos las condiciones iniciales

>>plot (y(:,1),'r',y(:,2),'g')% la dibujamos

>>[t,y]=ode45('bruselator',[0,100],[1.53769848008897;2.01410542603064],options);% buscamos el ciclo limite

>> plot (y(:,1),y(:,2)) %la dibujamos cerca del ciclo límite



Como conclusión podemos decir que el ciclo limite es estable y el equilibrio es inestable.

12. PROBLEMAS DE TANQUE

Ejemplo 1.- Tenemos un tanque X cuya superficie es de 300 m^3 (x) y la cantidad inicial de contaminante es de 0.1%. En él, entra una cantidad de contaminante de 1 m^3/minuto al 0.01%. De X se saca 0.7 m^3/minuto y se añade a otro tanque Y cuya superficie es de 200 m^3 (y) y la cantidad inicial de contaminante en Y es de 0.1 %. De Y se saca 0.7 m^3/minuto y se añade a otro tanque Z cuya superficie es de 400 m^3 (z) y la cantidad inicial de contaminante en Z es de 0.7 %. De Z se saca 1 m^3/minuto. Determinar la cantidad de contaminante en cada tanque que hay en un intervalo de tiempo y ver cómo evolucionan en un diagrama.

% Creamos en matlab el fichero p.m donde añadimos las ecuaciones para ver cómo evoluciona el sistema.

>>edit p.m

```
function w=p(t,xx) % ponemos nombre de p a la función

x=xx(1);y=xx(2);z=xx(3); % indicamos las tres ecuaciones

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

w(2)=0.7.*x./(300+0.3.*t)-0.7.*y./200;

w(3)=0.7.*y./200-z./(400-0.3.*t);

w=w';
```

>> p(1,[1,2,3]) % nos da el vector en vertical, aquí lo que pedimos es la cantidad de contaminante en el tanque x= xx(1), en el y= xx(2)=1 y en el z= xx(3)=1 a t=1.

```
ans =

   -0.0022

   -0.0047

   -0.0005
```

>> [t,w]=ode45('p',[0,30],[0.3;0.2;2.8])% para que nos de la cantidad de contaminante en los tres tanques en el intervalo de tiempo [0,30], añadiendo las condiciones iniciales

```
t =

        0

   0.7500

   1.5000

   2.2500

   3.0000

   3.7500

   4.5000

   5.2500

   6.0000

   6.7500

   7.5000

   8.2500

   9.0000

   9.7500

  10.5000

  11.2500

  12.0000

  12.7500

  13.5000

  14.2500

  15.0000

  15.7500

  16.5000

  17.2500

  18.0000

  18.7500
```

```
   19.5000

   20.2500

   21.0000

   21.7500

   22.5000

   23.2500

   24.0000

   24.7500

   25.5000

   26.2500

   27.0000

   27.7500

   28.5000

   29.2500

   30.0000


w =

    0.3000    0.2000    2.8000

    0.2996    0.2000    2.7953

    0.2991    0.2000    2.7906

    0.2987    0.2000    2.7859

    0.2982    0.2000    2.7811

    0.2978    0.2000    2.7764

    0.2973    0.2000    2.7718

    0.2969    0.2000    2.7671

    0.2964    0.2000    2.7624

    0.2960    0.2000    2.7577
```

0.2956	0.1999	2.7530
0.2951	0.1999	2.7484
0.2947	0.1999	2.7437
0.2942	0.1999	2.7391
0.2938	0.1999	2.7344
0.2934	0.1999	2.7298
0.2929	0.1999	2.7252
0.2925	0.1998	2.7205
0.2921	0.1998	2.7159
0.2917	0.1998	2.7113
0.2912	0.1998	2.7067
0.2908	0.1997	2.7021
0.2904	0.1997	2.6975
0.2900	0.1997	2.6929
0.2895	0.1997	2.6883
0.2891	0.1996	2.6837
0.2887	0.1996	2.6791
0.2883	0.1996	2.6746
0.2878	0.1996	2.6700
0.2874	0.1995	2.6654
0.2870	0.1995	2.6609
0.2866	0.1995	2.6563
0.2862	0.1994	2.6518
0.2858	0.1994	2.6472
0.2854	0.1994	2.6427
0.2849	0.1993	2.6382
0.2845	0.1993	2.6337

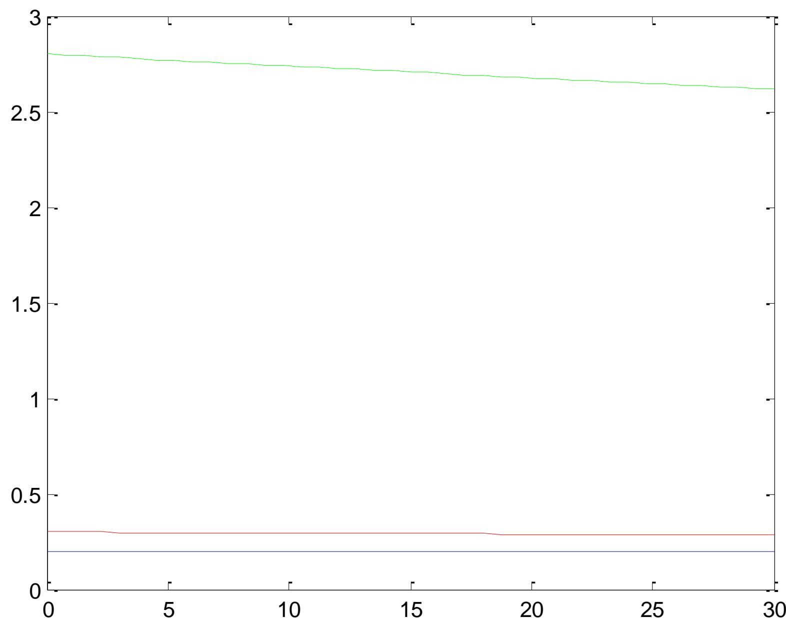
0.2841 0.1992 2.6292

0.2837 0.1992 2.6246

0.2833 0.1992 2.6201

0.2829 0.1991 2.6156

>> plot (t,w(:,1),'r',t,w(:,2),'b',t,w(:,3),'g') %dibujamos el sistema para ver como evoluciona a lo largo del intervalo de tiempo la cantidad de contaminante en los tres tanques, la línea roja es la evolución para el tanque X, la azul para el tanque Y y la verde para el Z.



Ejemplo 2.- Tenemos un tanque cuya superficie es de 400 m^3, con dos contaminantes distintos x e y que son inmiscibles. La cantidad inicial de contaminante x es de 0.1% y de la de y es de 0.2%. En él, entra una cantidad de ambos contaminantes de 1 m^3/minuto. De este tanque, se saca 0.7 m^3/minuto y se añade a otro tanque cuya superficie es de 500 m^3 y la cantidad inicial del contaminante x es de 0.3 % y del contaminante y es de 0.4%, sacándose 0.7 m^3/minuto. Determinar las cantidades de contaminantes x e y en cada tanque en un intervalo de tiempo y ver cómo evolucionan en un diagrama.

% Creamos en matlab el fichero p1.m donde añadimos las ecuaciones para ver cómo evoluciona el sistema.

>>edit p1.m

```
function w=p1(t,xx) % ponemos nombre de p a la función

x1=xx(1);x2=xx(2);y1=xx(3);y2=xx(4); % indicamos las cuatro ecuaciones para los dos contaminantes en los dos tanques, x1 e y1 son para el tanque primero y x2 e y2 para el segundo tanque. Esto se puede hacer porque son inmiscibles

w(1)=0.001-0.7.*x1./(400+0.3.*t);

w(2)=0.7.*x1./(400+0.3.*t)-0.8.*x2./(500-0.1.*t);

w(3)=0.002-0.7.*y1./(400+0.3.*t);

w(4)=0.7.*y1./(400+0.3.*t)-0.8.*y2./(500-0.1.*t);

w=w';
```

>> [t,w]=ode45('p1',[0,30],[0.1;0.2;0.3;0.4]) % para que nos de las cantidades de los dos contaminantes en los dos tanques en el intervalo de tiempo [0,30], añadiendo las condiciones iniciales que son las concentraciones de ambos contaminantes en los dos tamques

```
t =

     0

  0.7500

  1.5000

  2.2500

  3.0000

  3.7500

  4.5000

  5.2500

  6.0000

  6.7500

  7.5000

  8.2500

  9.0000
```

9.7500

10.5000

11.2500

12.0000

12.7500

13.5000

14.2500

15.0000

15.7500

16.5000

17.2500

18.0000

18.7500

19.5000

20.2500

21.0000

21.7500

22.5000

23.2500

24.0000

24.7500

25.5000

26.2500

27.0000

27.7500

28.5000

29.2500

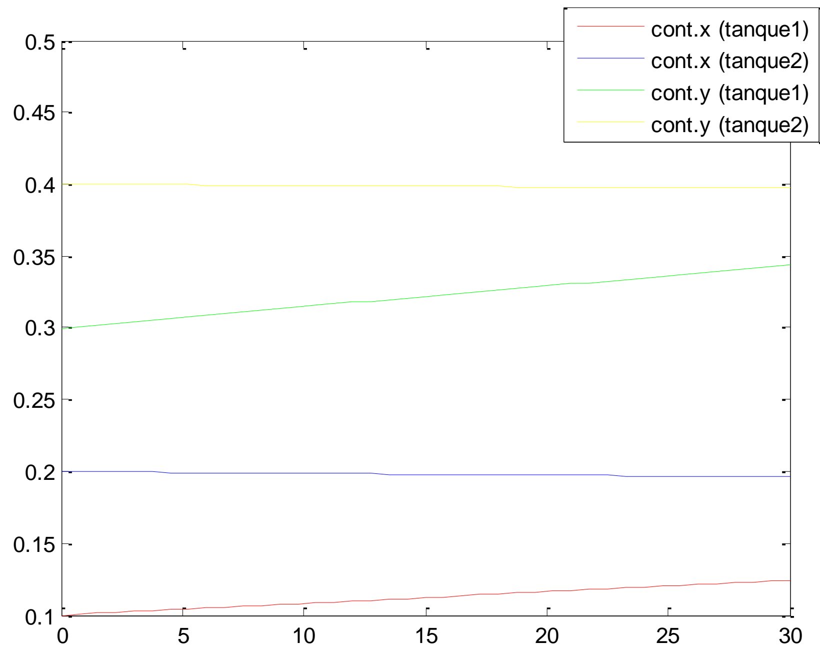
30.0000

w =

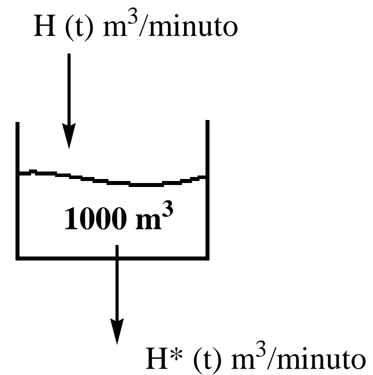
```
    0.1000    0.2000    0.3000    0.4000
    0.1006    0.1999    0.3011    0.3999
    0.1012    0.1998    0.3022    0.3998
    0.1019    0.1997    0.3033    0.3997
    0.1025    0.1996    0.3044    0.3997
    0.1031    0.1995    0.3055    0.3996
    0.1037    0.1994    0.3066    0.3995
    0.1043    0.1993    0.3077    0.3994
    0.1049    0.1992    0.3088    0.3994
    0.1055    0.1991    0.3099    0.3993
    0.1062    0.1990    0.3110    0.3992
    0.1068    0.1989    0.3121    0.3991
    0.1074    0.1988    0.3132    0.3991
    0.1080    0.1987    0.3143    0.3990
    0.1086    0.1986    0.3154    0.3989
    0.1092    0.1985    0.3165    0.3988
    0.1098    0.1984    0.3175    0.3988
    0.1104    0.1983    0.3186    0.3987
    0.1110    0.1982    0.3197    0.3986
    0.1116    0.1981    0.3208    0.3986
    0.1122    0.1980    0.3219    0.3985
    0.1128    0.1979    0.3230    0.3985
    0.1134    0.1978    0.3240    0.3984
    0.1140    0.1977    0.3251    0.3983
```

```
0.1146    0.1976    0.3262    0.3983

0.1152    0.1975    0.3273    0.3982

0.1158    0.1974    0.3284    0.3982

0.1164    0.1974    0.3294    0.3981

0.1170    0.1973    0.3305    0.3981

0.1176    0.1972    0.3316    0.3980

0.1182    0.1971    0.3326    0.3980

0.1188    0.1970    0.3337    0.3979

0.1194    0.1969    0.3348    0.3979

0.1200    0.1968    0.3359    0.3978

0.1206    0.1968    0.3369    0.3978

0.1212    0.1967    0.3380    0.3977

0.1218    0.1966    0.3391    0.3977

0.1224    0.1965    0.3401    0.3976

0.1230    0.1964    0.3412    0.3976

0.1236    0.1964    0.3422    0.3976

0.1242    0.1963    0.3433    0.3975
```

>> plot (t,w(:,1),'r',t,w(:,2),'b',t,w(:,3),'g',t,w(:,4),'y') %dibujamos el sistema para ver como evoluciona a lo largo del intervalo de tiempo las cantidades de contaminantes en los dos tanques, la línea roja es del contaminante x en el tanque primero, la azul es del contaminante x en el tanque segundo, la línea verde es del contaminante y en el primer tanque y la amarilla del contaminante y en el segundo tanque.



Ejemplo 3.- Tenemos un tanque cuya superficie es de 1000 m^3. Por arriba, se añade una cantidad H(t) en m^3/ minuto de dos contaminantes, uno llamado x al 0.01% y el otro llamado y al 0.02%. Se deja de añadir H(t), y se comienza a vaciar el tanque por debajo, sacando H*(t) en m^3/minuto, como vemos en la figura. Las condiciones iniciales para cada contaminantes son x(0)=100·0.005 m^3 e y(0)=100·0.007 m^3.



% Creamos en matlab el fichero tanqueH.m donde añadimos las ecuaciones para ver cómo evoluciona el sistema.

>> edit tanqueH.m

```
function y=tanqueH(t) % ponemos nombre de tanqueH a la función

if fix(fix(t)/2)==fix(t)/2 y=1;%fix(t)/2 te va a decir si es par

   else y=0;% te dice que si lo de ariiba no es, y=0

end
```

%comprobamos el programa que hemos hecho

```
>> tanqueH(0.5)

ans =

    1

>> tanqueH(1.7)

ans =

    0
```

% como vemos con 1.7 no se cumple y por eso y=0

% para ver el volumen del tanque creamos el archivo volumentanque.m

```
>> edit volumentanque.m
```

```
function y=volumentanque(t) % ponemos nombre de volumenetanque a la función

if fix(fix(t)/2)==fix(t)/2 y=t-fix(t);

  else y=fix(t)+1-t;
```

%El archivo volumentanque.m solo tiene un ciclo, lo modificamos para que contengan ciclos de tres y lo llamamos volumen tanque tres

```
>>edit volumentanque3.m
```

```
function y=volumentanque3(t) % ponemos nombre de volumenetanque3 a la función

if fix(fix(t)/3)==fix(t)/3 y=0; end

   if fix((fix(t)-1)/3)==(fix(t)-1)/3 y=1; end

      if fix((fix(t)-2)/3)==(fix(t)-2)/3 y=2; end
```

%Si le añadimos un tiempo en matlab, nos da el volumen que contiene el tanque a ese tiempo

>>volumentanque3(3.5)

ans =

0

% en el tiempo 3.5, el volumen del tanque permanece constante

>>volumentanque3(2.7)

ans =

2

%sin embargo, a tiempo 2.7, el volumen va aumentando con la variación que da el tercer ciclo.

13. PROBLEMA DE SUMA DE CUADRADOS

El objetivo del ejercicio es obtener todos los números de un intervalo que sean suma de dos números al cubo.

%creamos el fichero en matlab, el cual llamaremos descomposición.m

>>edit descomposicion.m

```
function [lista]=descomposicion(N1,N2)%para  cuando se ejecuta en matlab, al añadir los números N1 yN2 que ES el intervalo al que queremos que se cumpla la condición

lista=[];

for m=N1:N2

  s=0;sublista=[];

  for a=1:(m/2)^(1/3)+0.0000001

    b=fix((m-a^3)^(1/3)+0.0000001);

    if m==a^3+b^3 s=s+1; sublista=[sublista,m,a,b]; end

  end

  if s==2 lista=[lista;sublista]; end

end
```

%Una vez editado el programa, tan sólo hay que poner el intervalo de los números que queremos buscar que sean suma de dos números.

```
>> [lista]=descomposicion(100,10000)

lista =

    1729     1    12    1729     9    10

    4104     2    16    4104     9    15
```

% en el intervalo entre 100 y 1000, el primer número que se descompone en dos números al cuadrado es el 1729, y el siguiente es el 4104

```
>> [lista]=descomposicion(10000,100000)%ampliamos el intervalo

lista =

   13832     2    24   13832    18    20

   20683    10    27   20683    19    24

   32832     4    32   32832    18    30

   39312     2    34   39312    15    33

   40033     9    34   40033    16    33

   46683     3    36   46683    27    30

   64232    17    39   64232    26    36

   65728    12    40   65728    31    33
```

%como vemos hay más números que cumplen la condición

14. PROBLEMA DE SUMA DE CUBOS

Ahora el objetivo del ejercicio es obtener todos los números de un intervalo que se puedan expresar como dos combinaciones de una suma de dos números al cubo.

%creamos el fichero en matlab, al que llamamos descomposicion3.m

```
>>edit descomposicion3.m
```

```
function [lista]=descomposicion(N1,N2)%N1 y N2 es el intervalo al que queremos
lista=[];
for m=N1:N2
   s=0;sublista=[];
   for a=1:(m/2)^(1/3)+0.0000001%lo que cambia com el otro es que ahora es el cubo
      b=fix((m-a^3)^(1/3)+0.0000001);
      if m==a^3+b^3 s=s+1; sublista=[sublista,m,a,b]; end
   end
   if s==3 lista=[lista;sublista]; end
end
%si no se cumple la condición, que salga La lista vacía
```

```
>>[lista]= descomposicion3(1,10000)%ejecutamos en el intervalo al que queremos saber los números

lista =

    []%quiere decir que no hay ningún número que cumpla la condición, es decir que se descompongan en el cubo de dos números

>> [lista]= descomposicion3(1,100000)

lista =

    []%hemos ampliado el intervalo y sigue sin haber ningún número
```

Por lo tanto, no existen en estos intervalos números que cumplan la condición que hemos impuesto.

15. PROBLEMA DE SUMA DE CUBO Y CUADRADO

El objetivo del ejercicio es obtener todos los números de un intervalo que sean suma de dos números, uno al cuadrado y el otro al cubo.

```
%creamos el fichero en matlab con el nombre de ejerc2.m

>>edit ejerc2.m
```

```
n=2;s=0;%debemos de tener en cuenta que ahora es el cuadrado y el cubo
while s<2
   lista=[ ];s=0;n=n+1;
   for a=1:(n/2)^(1/3);
      b=fix((n-a^3)^(1/2)+0.00001);
      if n==a^3+b^3 lista=[lista,n,a,b]; s=s+1;end
   end
end
lista n=2;s=0;
while s<2
   lista=[ ];s=0;n=n+1;%si no se cumple la condición, que salga La lista vacía
   for a=1:(n/2)^(1/3);
      b=fix((n-a^3)^(1/2)+0.00001);
      if n==a^3+b^3 lista=[lista,n,a,b]; s=s+1;end
   end
end
lista
```

16. PROBLEMA DE LA HIPOTECA

El objetivo es crear un archivo que introduciendo el capital a hipotecar, el interés, y el pago mensual; obtengamos como argumentos de salida lo que se paga en total y el número de meses que hay que pagar.

%creamos el fichero en matlab con el nombre de hipoteca.m

>>edit hipoteca.m

```
function [T,M]=hipoteca (C,I,P);

%T-lo que pago en total, M-meses, C-capital a hipotecar, I-interes, P-pago mensual

S=C; M=0;

i=I/1200;

while S>P

   S=S+(S.*i)-P;

   M=M+1;

end

T=(M-1).*P+S;%ecuación que da lo que pago en total
```

>>[T,M]=hipoteca(300000,3,1000)%metiendo las condiciones que quiero, el capital que quiero hipotecar es 300000, el interés que es al 3% y el pago mensual que es 1000.

T =

  5.5421e+005

M =

   555

% nos da el programa T que es lo que tengo que pagar y los meses que tengo que pagar

17. EJERCICIO INTEGRAL

El objetivos es calcular la integral de una función aplicando la definición de integral, dibujándola en [-2,2] y obteniendo cuáles son sus raíces. La función es $f(x)=x^5-2x^2+3x-7$.

%creamos el fichero en matlab para introducir la ecuación

>> edit ejercintegral.m %llamamos el fichero de la función ejercintegral.m

```
function y=ejercintegral(x);

y=(x.^5)-(2.*x.^2)+(3.*x)-7;
```

% Hacemos el dibujo general y buscamos la raíz por el método bisección. Por último calculamos el área que ay debajo de la curva.

>>area=-quadl('ejercintegral',-2,1.4723)+quadl('ejercintegral',1.4723,4)

18. OTRO EJEMPLO DE EJERCICIO INTEGRAL

Calcular el área que forma la curva y= x^3 + 2x^2 -1 con el eje x, entre x=-3 y x=4

%creamos el fichero de la función en matlab con el nombre de ejercintegral1.m

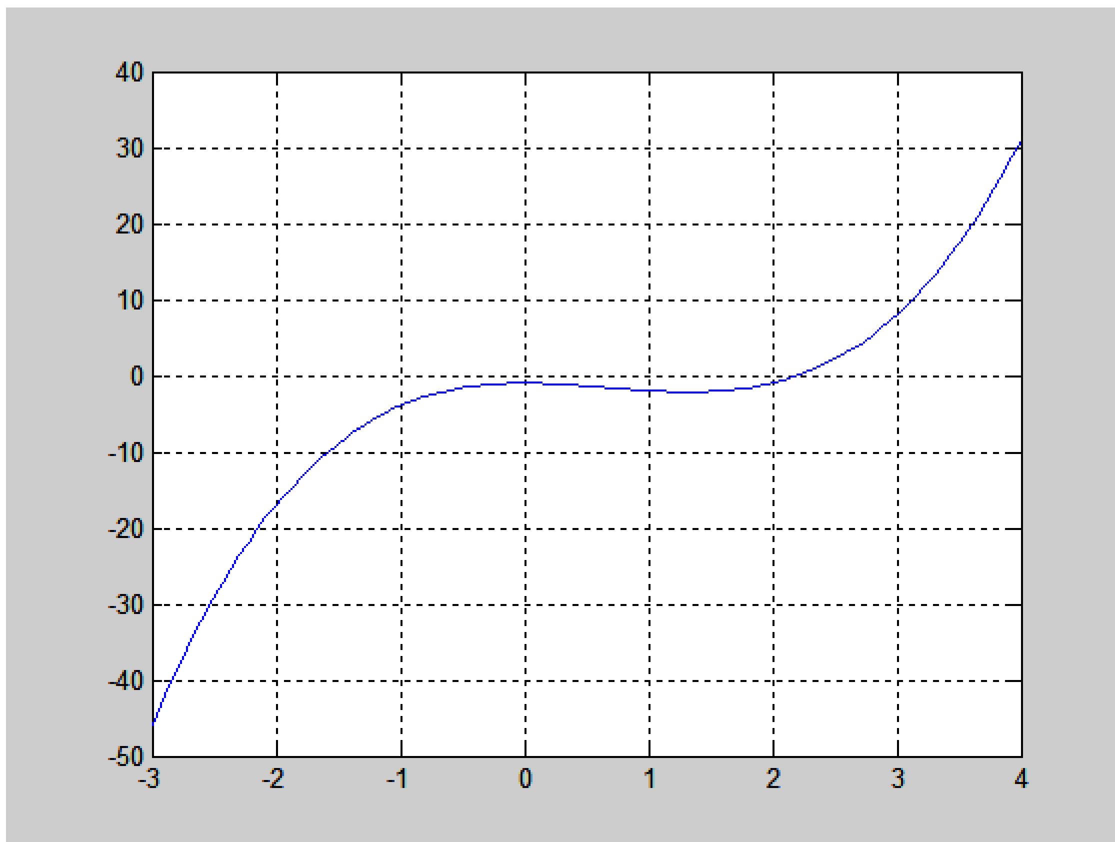
>>edit ejercintegral1.m

```
function y= ejercintegral1(x)

y=x.^3-2.*x.^2-1
```

%Se buscan las raices por el método de bisección, por lo que se dibuja la función para saber por dónde se encuentra los números próximos a la raíz.

>> dibujogeneral ('ejercintegral1',-3,4,100);

>>grid %le ponemos la malla para ver mejor donde está la raíz



%Como se observa, el número próximo a la raíz es 2, por lo que ahora se buscan las raíces.

>> biseccion('ejercintegral1',1.46,1.48,10^-6) %calculámos la raíz por el método de bisección

ans =

1.48

>> area=-quadl('ejercintegral',-2,1.48)+quadl('ejercintegral',1.48,4)% calculamos el area por debajo de la curva que se hace con integrales

area =

686.9681

19. EJERCICIO MÁXIMOS Y MÍNIMOS

Calcular los máximos y los mínimos de la función

% para ello primero creamos el fichero de la función en matlab que llamamos ejercicioa.m

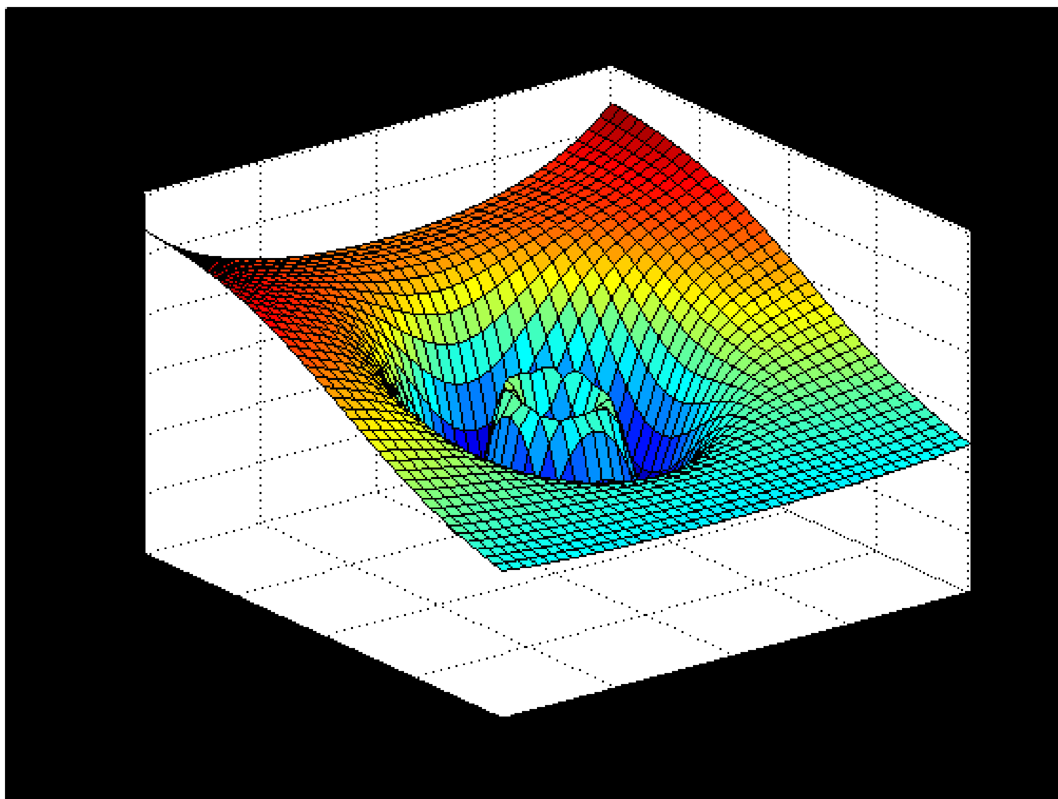
```
>>edit ejercicioa.m
```

```
function z=ejercicioa(x,y)

z=exp(x.^2+y).*sin(1./(0.1+x.^2+y.^2));
```

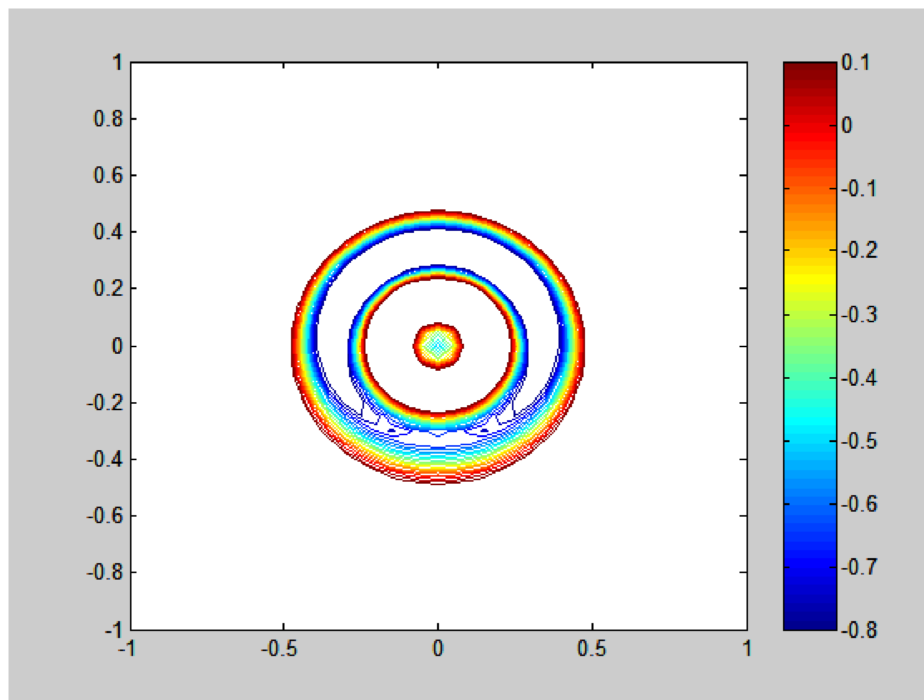
>>[x,y]=meshgrid(-1:0.05:1, -1:0.05:1)% para poner las condiciones en el intervalo que queremos que nos dibuje la superficie

>>z=ejercicioa(x,y)

>>surf(x,y,z) % para que nos dibuje la función en forma de superficie



>>contour(x,y,z,-0.8:0.05:0)% para que nos lo de en forma de isotermas es con el comando contour



En la figura se ven perfectamente los máximos y los mínimos. Para buscar un mínimo, en Matlab existe un comando específico, ‘fminsearch’, pero sin embargo, no existe un parámetro análogo para buscar un máximo. Pero si lo que se busca es un máximo, si cambiamos la función de signo, podríamos buscarlo con ‘fminsearch’.

%Para utilizar este comando, tenemos que espresar las funciones como vectores, por lo que creamos otro fichero que defina la función como vectores.

%creamos el fichero ejercicioa1.m de la función con vectores

>>edit ejercicioa1.m

```
function z=ejercicioa1(xx)% al ponerle las dos x introducimos los vectores

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

z=exp(x.^2+y).*sin(1./(0.1+x.^2+y.^2));
```

>> xx=fminsearch('ejercicioa1',[0;0.01])% así nos da todos los valores de los mínimos

y = 0.0100; y =0.0100; y =0.0105; y =0.0095; y = 0.0090; y =0.0090; y =0.0085; y = 0.0075; y =0.0062¸ y =0.0057; y =0.0041; y =0.0019; y =-2.5000e-004; y =0.0046; y = 0.0069; y =0.0031; y =0.0026; y =0.0016; y =0.0035; y =0.0040; y =0.0030; y =0.0033; y =0.0035; y =0.0040; y =0.0032; y =0.0032; y =0.0030; y =0.0034; y =0.0033; y = 0.0030; y =0.0033; y =0.0034; y =0.0032; y =0.0033; y =0.0033;

xx =

-0.0000

0.0032

%Como puede observarse, hemos obtenido todos los puntos donde la función alcanza un mínimo. Para los máximos, bastaría con cambiar la función de signo.

20. EJERCICIO DEL LAGO

Un lago que tiene 475 km^3 y actualmente tiene un 0.05% de contaminante, por lo que hay que hacer una limpieza del lago. Hay una aportación anual de 175 km^3 de agua con una contaminación de 0.01 % de contaminante. Del mismo lago, sale anualmente una caudal de 175 km^3. ¿Qué tiempo tardará en llegar al 0.02% de contaminante?

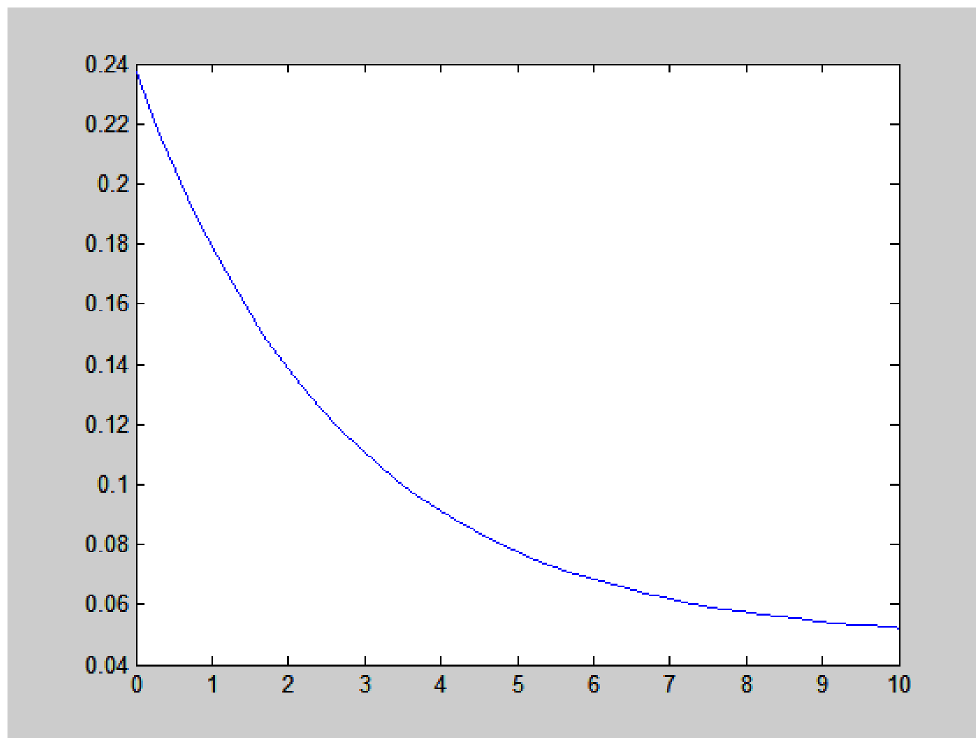
% creamos el fichero en matlab para añadir la función a la que llamamos lago.m

>>edit lago.m

```
function z=lago(t,x) % t es el tiempo y x concentración de contaminante

z=175*0.0001-175.*x/475;
```

>>[t,x]=ode45('lago',[0,10],475*0.0005);%ponemos las condiciones iniciales de contaminante y que nos de los valores de este en el intervalo de tiempo de 0 a 10

>>plot (t,x)% dibujamos estos valores obteniendo la gráfica



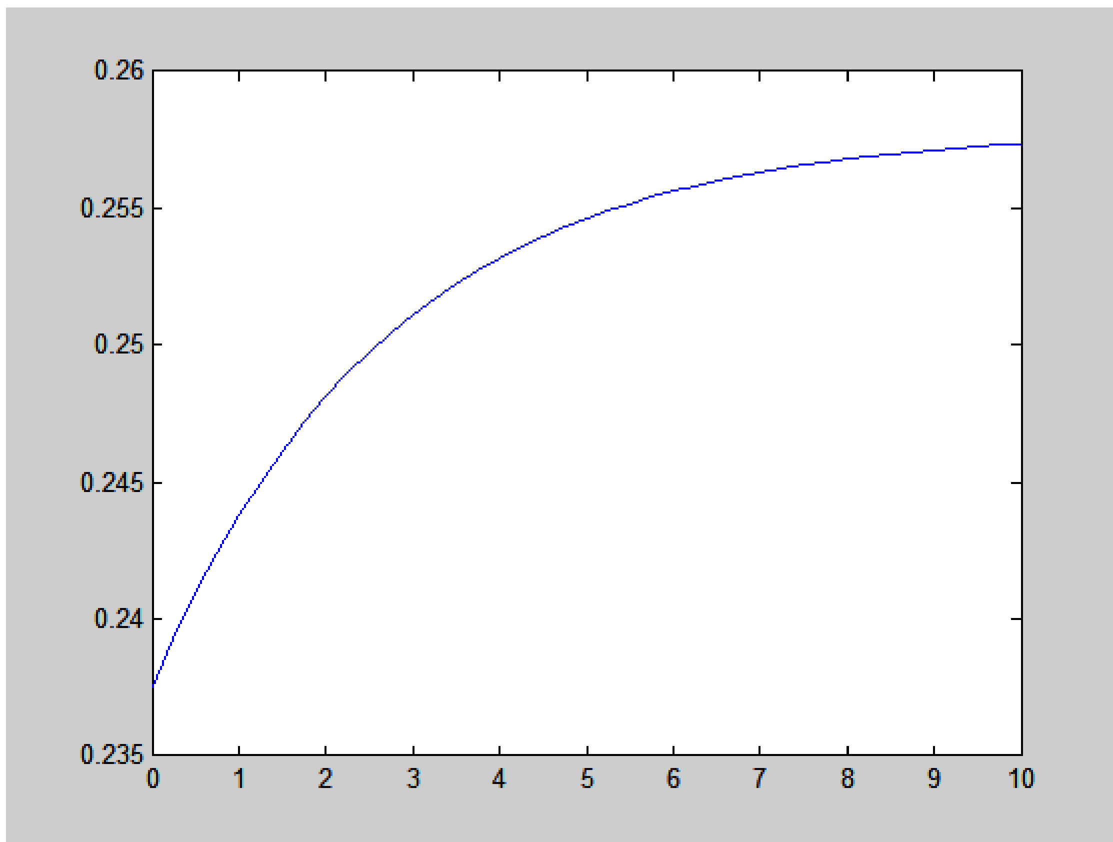
%Para saber cuando el lago está al 0.02%, tenemos que modificar el programa que hemos editado, el lago.m

>>edit lago.m

```
function z=lago(t,x)% lo modificamos

z=475*0.0002-175.*x/475;
```

>> [t,x]=ode45('lago',[0,10],475*0.0005);% vovemos a poner el intervalo de tiempo al que queremos y la condición inicial de contaminante

>> plot (t,x) % lo dibujamos con este comando



21. EJERCICIO DE SOLUBILIDAD DE UN PRECIPITADO EN PRESENCIA DE UN IÓN COMÚN

De acuerdo con el principio de Le Chatelier, la presencia de un ión común con algunos de los iones del precipitado ejerce influencia desfavorable sobre la solubilidad. Si se considera la solubilidad del yoduro de plomo en agua pura (K_{ps}=7·10^{-9}) se tiene que:

$$K_{ps} = 7.1\cdot 10^{-9} = \left|Pb^{2+}\right|\left|I^{-}\right|^{2} = \left|Pb^{2+}\right|\cdot\left(2\left|Pb^{2+}\right|\right)^{2} = 4\left(Pb^{2+}\right)^{3}$$

$$S_M = \left|Pb^{2+}\right| = \sqrt[3]{K_{ps}/4}$$

$$S_M = 0.0012 mol/L$$

Cuando se le añade una disolución de KI la solubilidad disminuye:

Si la concentración del ión común corresponde a C, el producto de solubilidad se calcula asi:

$$K_{ps} = 7.1 \cdot 10^{-9} = |Pb^{2+}||I^{-}|^{2} = |Pb^{2+}|\left(C + 2|Pb^{2+}|\right)^{2}$$

Para el caso en que C>> $|Pb^{2+}|$, las ecuaciones se reducen a:

$$K_{ps} = C^{2}|Pb^{2+}|^{2}$$

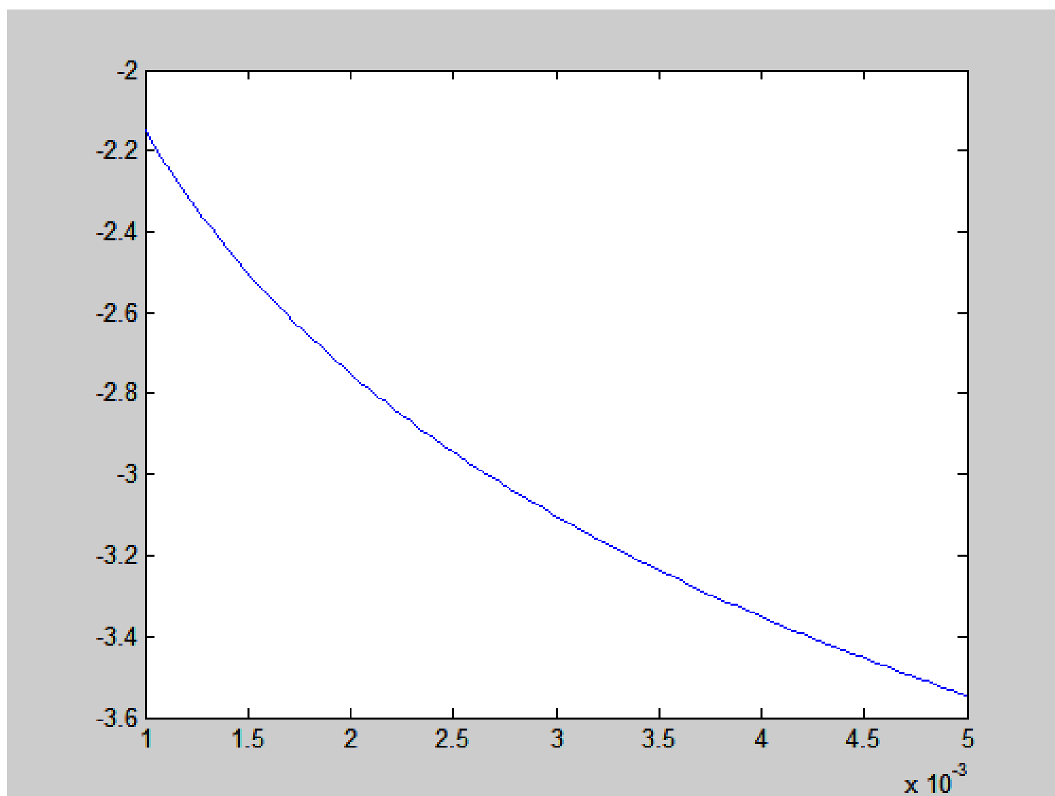
$$S_{M} = K_{ps} / C^{2}$$

%creamos el fichero en Matlab para concentraciones de ión común de 0.00 a 0.05 mol/L

```
>>edit solubilidad.m
```

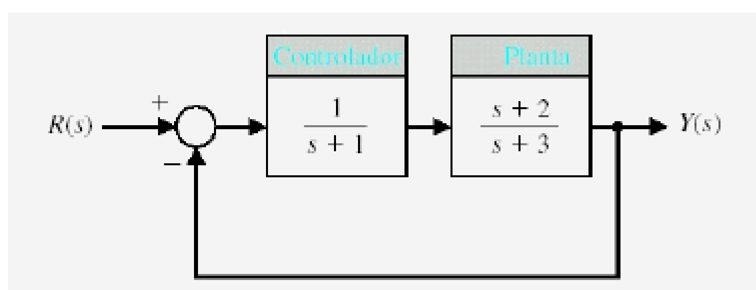
```
%Datos

Kps=7.1e-09;

C=linspace(0.001,0.005,100);

Sm=Kps./C.^2;

pSm=log10(Sm);

%resultados

fprintf('C,mol/L     Sm,mol/L\n\n');

for i=1:5:length(C)

    fprintf('%6.3f',C(i));

    fprintf('%12.2e\n',Sm(i));

end

plot(C,pSm)%añadimos el commando para que nos dibuje
```

C,mol/L	Sm,mol/L
0.001	7.10e-003
0.001	4.91e-003
0.001	3.60e-003
0.002	2.75e-003
0.002	2.17e-003
0.002	1.76e-003
0.002	1.45e-003
0.002	1.22e-003
0.003	1.04e-003
0.003	8.94e-004
0.003	7.78e-004
0.003	6.84e-004
0.003	6.06e-004
0.004	5.40e-004
0.004	4.84e-004
0.004	4.37e-004
0.004	3.96e-004
0.004	3.61e-004
0.005	3.30e-004
0.005	3.03e-004



22. PROBLEMA DE REALIMENTO

Considere el sistema realimentado mostrado en la figura:



a). Calcule la función de transferencia de lazo cerrado.

b). Obtenga la respuesta al impulso y la respuesta al escalón del sistema.

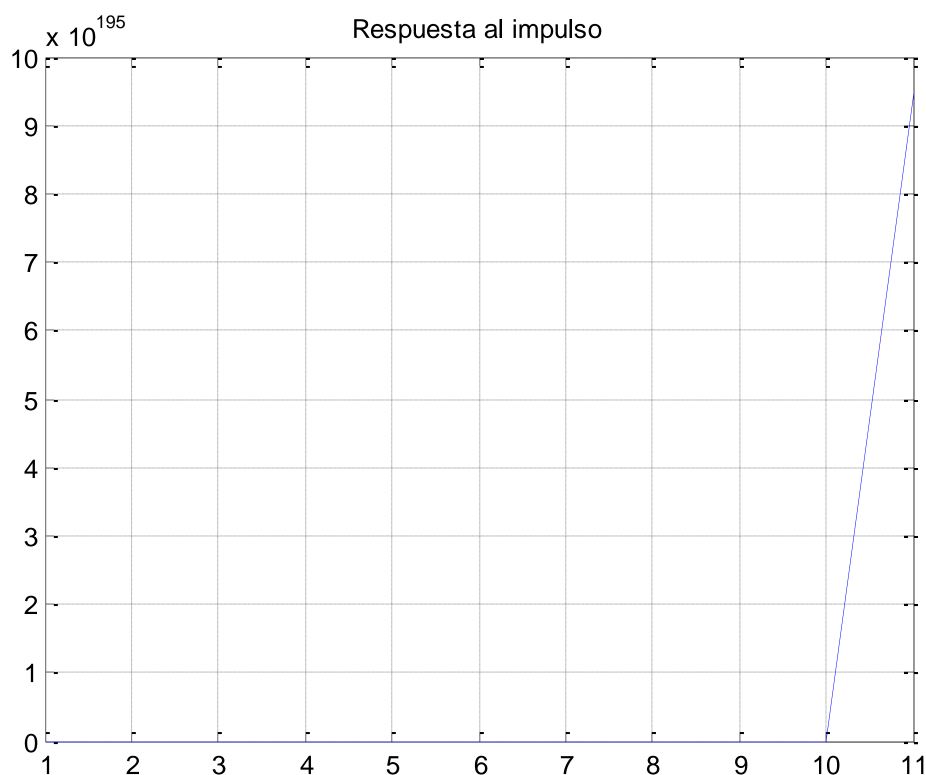
Grafique dichas respuestas y halle el valor final en el caso de la respuesta al escalón.

%Función de transferencia de lazo cerrado

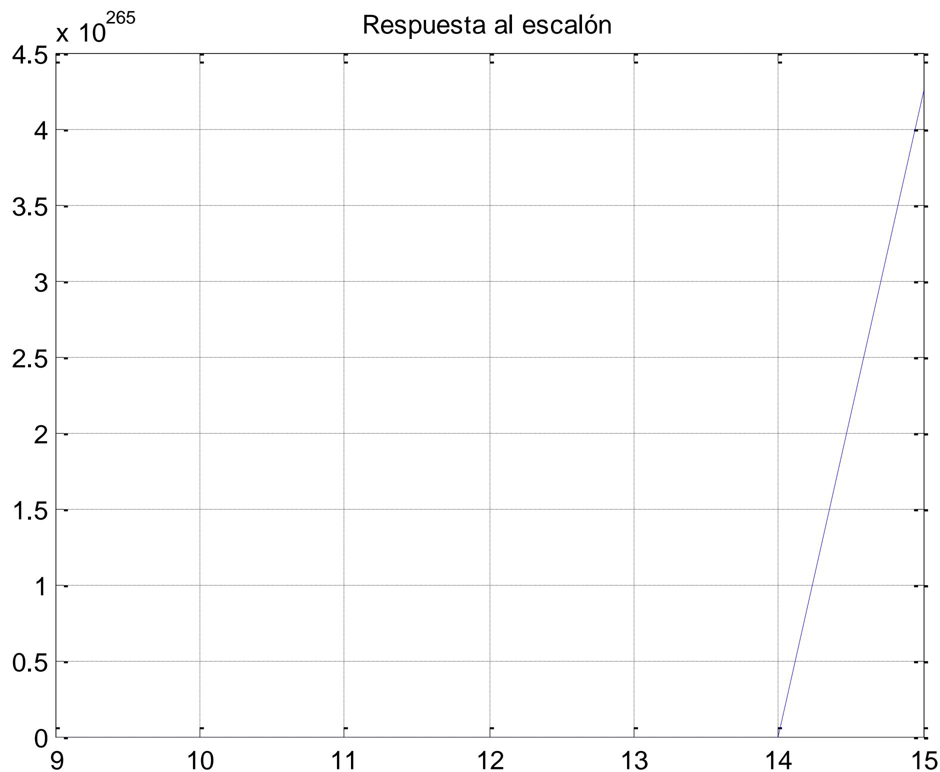
```
>> a2=feedback((series((tf(1,[1 1])),(tf([1 2],[1 3])))),1,'+');
```

%Respuesta al impulso

```
>> b2i=impulse(a2,[1:11]);

>> figure

>> plot([1:11],b2i), title('Respuesta al impulso'), grid, pause;
```




```
b2e=step(a2,[9:15]);

>> figure

>> plot([9:15],b2e), title('Respuesta al escalón'), grid, pause;
```



23. PROBLEMA ESCALÓN UNITARIO

Considere la ecuación diferencial:

$$\ddot{y} + 2\dot{y} + y = u$$

Donde $y(0) = \dot{y}(0) = 0$ y u(t) es un escalón unitario. Determine la solución y(t) solucionando por transformada inversa de Laplace y por el comando step. Compare los resultados, dibujando simultáneamente las dos respuestas.

%Por transformada inversa de Laplace

```
>> syms ys s

>> ys=1/(s*(s+1)^2);

>> yt=ilaplace(ys);

>> pretty(ys)
```

```
         1
     ----------
           2
     s (s + 1)

>> pretty(yt), pause;


    1 + (-1 - t) exp(-t)

>> figure

>> ezplot(yt,[0:0.1:6]), grid, pause;
```