1. Comandos Básicos y Matrices

1.1. Introducción

Matlab es un lenguaje de alto nivel orientado al desarrollo de cálculos técnicos. Integra cálculo, visualización y programación en un entorno interactivo de fácil manejo. Los problemas y las soluciones se expresan en la notación matemática habitual.

El elemento básico de información es una matriz a la que no hace falta asignar dimensiones con anterioridad. Por tanto, pueden abordarse problemas que requieren una formulación vectorial o matricial de un modo más fácil que en un lenguaje tipo FORTRAN o C. El nombre MATLAB es una abreviatura de Matrix Laboratory.

1.2. Iniciación a Matlab

Para entrar en Matlab, debemos hacer clic dos veces en el icono correspondiente. Para salir, tenemos dos opciones:

1) en File podemos hacer clic en exit

2) podemos teclear quit y pulsar enter.

Una vez que hemos entrado en Matlab, escribiremos las instrucciones en la ventana de comandos (command window) a partir del símbolo¶ >> (denominado prompt). Si se quiere salvar en un archivo toda una sesión de trabajo, podemos proceder como sigue:

Iniciamos la sesión tecleando:
>> diary nombre.txt
y terminar la sesión tecleando diary, automáticamente Matlab crea un archivo con la denominación nombre.txt que se encuentra en el directorio de trabajo. El nombre es opcional, pero un nombre adecuado puede ser poner el mes seguido de la fecha del día en curso. Si queremos guardar el archivo en un disquete, CD,.., lo abrimos y en File hacemos clic en save as. Ahora pulsando en Mi PC podemos guardarlo en la unidad que deseemos.

Matlab nos ofrece una ayuda que puede ser de gran utilidad. Si tenemos alguna duda sobre el funcionamiento concreto de determinada función, de la que conocemos su denominación en Matlab, podemos teclear en la ventana de comandos >> Help nombre y aparece en pantalla la ayuda de Matlab explicando el funcionamiento de la función. Si no fuera suficiente, podemos pulsar en doc nombre que nos ofrece una ayuda más completa y ejemplos. Supongamos que tenemos alguna duda sobre el funcionamiento de la función det (determinante). Tecleamos >> Help det. Si tecleamos >> lookfor nombre aparece en pantalla una relación con todos los comandos que contienen nombre. Si no queremos que la ayuda aparezca en la ventana de comandos sino en una ventana específica, debemos teclear >> helpwin nombre y se abre una ventana con la ayuda. Finalmente, indicar que también podemos buscar la ayuda acudiendo a Help y haciendo clic en Matlab Help.

1.3.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 $\sqrt{3}$, 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

1.4. 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 vectoró 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 numero 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).

2. Gráficos

2.1. 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

*c*onvenientemente 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*π; 2π*].

>> x=linspace(-2*pi,2*pi,60); % Tomamos 60 valores de x igual % mente espaciados en [-2*π; 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 y*1 = *x*2 *e y*2 = *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);*

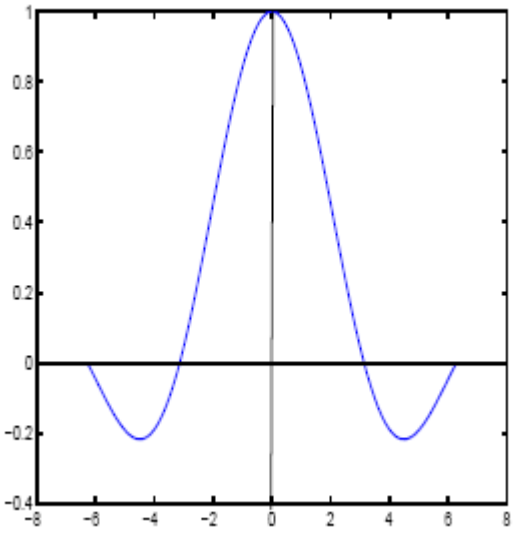
plot(x2,y2)

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 ϵ [-2π; 2π]. Se presenta el problema de que para x = 0 no está definida la función, aunque $\lim_{x\to 0} f(x)$ 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:



Otras opciones de la función plot

1. Etiquetas sobre los ejes: el comando xlabel('texto') se usa para que en el eje OX aparezca el *texto* que se desea. Análogamente, y label('texto') para el eje OY. Para colocar cualquier cadena de texto en el punto que queramos de la ventana gráfica, se usa el comando text(x,y,'texto'), donde (x,y) son las coordenadas del punto donde queremos situar el centro izquierda del texto. Si usamos el comando gtext, entonces podemos colocar el texto donde queramos con el propio ratón.

2. Color de la curva y estilo de línea: Se puede dibujar la curva del color y con el estilo que se desee. Por ejemplo, si se escribe >>plot(x,y,'.',x,z,'- -') De esta forma se consigue que la curva $y = y(x)$ aparezca con trazo punteado y la $z = z(x)$ como una línea de trazo discontinuo. Si se quiere escoger el color y estilo:

```
>> plot(x,y,'r - -')
```

La curva aparece en rojo y con línea de trazo discontinuo. La siguiente tabla muestra la sintaxis de los diferentes colores y tipo de línea:

Símbolo	Color	Símbolo	Estilo de línea
y	amarillo	.	línea de puntos
m	magenta	$\circ$	círculo
r	rojo	$+$	más
g	verde	$*$	estrella
b	azul	$--$	trazo discontinuo
k	negro	$-$	línea sólida

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.

Dibujo de poligonales

Supongamos que se desea dibujar la poligonal de vértices (*xi; yi*) 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).

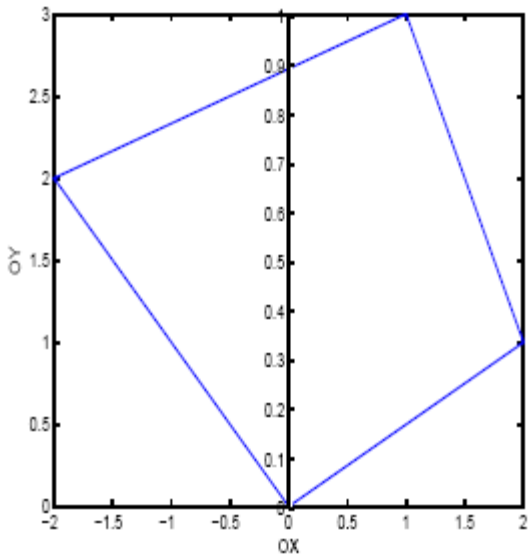
Ejemplo. *Dibujar la poligonal cerrada de vértices* (0*;* 0)*;* (2*;* 1)*;* (1*;* 3) y (-2*;* 2).

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

y=[0 1 3 2 0];

plot(x,y)

y el resultado sería

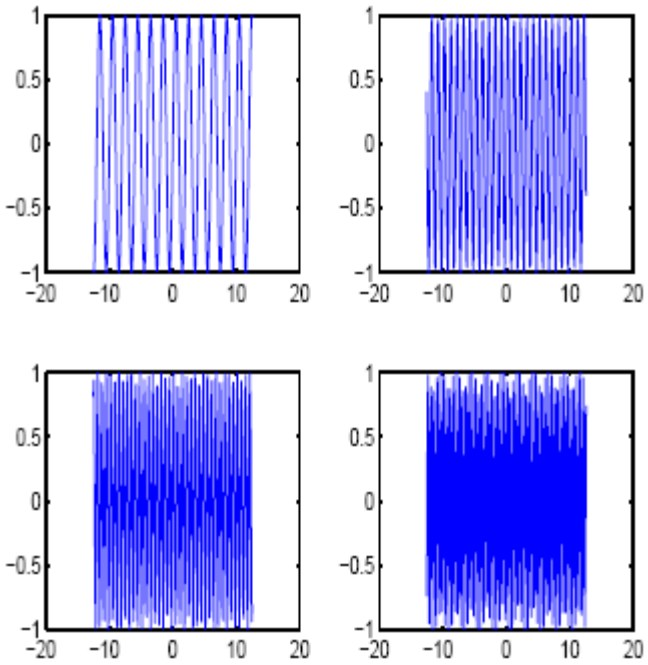


2.2. El comando subplot

A veces nos interesaría disponer en una misma ventana gráfica de varias subvén tañas para dibujar en cada una de ellas una curva distinta, con el objetivo de poder compararlas más cómodamente. Veamos un ejemplo. Ejemplo. Queremos dibujar en una misma ventana gráfica las curvas y = sen($k\pi x$), para k = 1*;... ;* 4 y $x \in$ [-4π*;* 4π], cada una en una subventana diferente.

```
>>x=linspace(-4*pi,4*pi,240);
subplot(2,2,1)
plot(x,sin(pi*x))
subplot(2,2,2)
plot(x,sin(2*pi*x))
subplot(2,2,3)
plot(x,sin(3*pi*x))
subplot(2,2,4)
plot(x,sin(4*pi*x))
```

Y se obtiene



En general, si se necesitan *m* ϵ *n* subventanas, se tendrá en cuenta que se numeran de izquierda a derecha y de arriba hacia abajo. Cuando tecleamos subplot(m,n,k), estamos indicando que vamos a dibujar en la subventana que ocupa el lugar *k*. En el ejemplo anterior, todas las subventanas responden a la forma subplot(2,2,k), porque teníamos 4 curvas y parece lo más adecuado disponerlas en dos filas y dos columnas.

2.3. Coloreado de polígonos

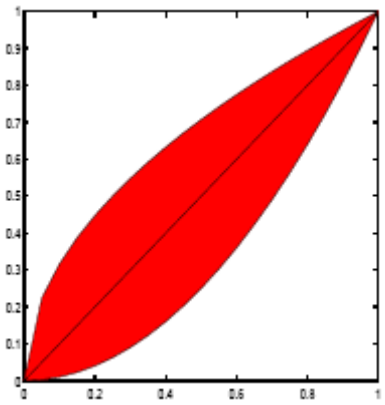
En la sección anterior hemos visto cómo se dibuja una poligonal con Matlab. Ahora vamos a ver cómo se colorea la región interior del color que queramos.

Ejemplo. Se desea dibujar las curvas $y = x2$ y $x = y2$ en el primer cuadrante. La región que encierran debe aparecer en color rojo.

```
>> x=0:.1:1;
y1=x.^2;y2=sqrt(x);
plot(x,y1,x,y2)
```

De esta forma se han dibujado las curvas en cuestión. Para colorear de rojo la región encerrada, Matlab dispone de la función fill. Su sintaxis es la siguiente: fill(x,y,'r') (*x* e *y* son matrices fila que contienen las coordenadas *x* e *y* de los vértices de la poligonal. El programa anterior se continuaría de la forma siguiente:

```
X=[x x];
y=[y1 y2];
fill(X,y,'r')
```



Si se desea escoger un color indicando las coordenadas (en el sistema RGB) se escribe >>fill(x,y,[r g b]) Los números *r; g* y *b* pertenecen a (0,1) y representan las proporciones en que se deben tomar los colores principales (rojo, verde y azul) para crear el color en cuestión.

2.4. 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 ω 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 $\omega\ 2\ [\omega 1; \omega 2]$, 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.

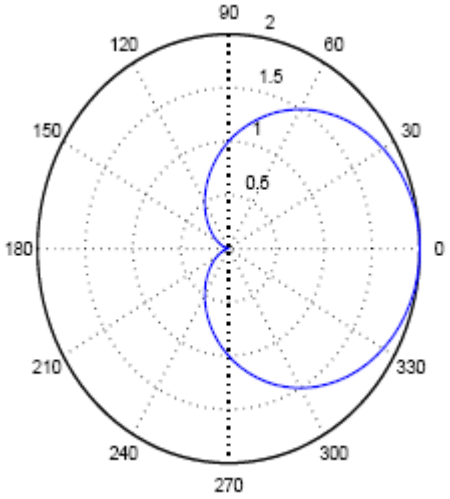
Ejemplo. *Dibujar la curva de ecuación* $r = 1 + \cos \omega$, *para* $0 \cdot \omega \cdot 2\pi$.
Por comodidad, vamos a usar *w* en lugar de ω.

```
>> 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).



2.5. Curvas en el espacio

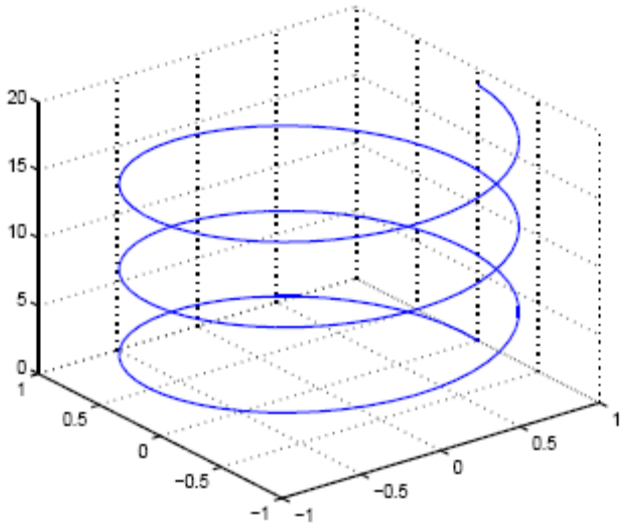
Supongamos que se quiere dibujar la curva de ecuaciones paramétricas $x = \cos t$; $y = \text{sen } t$; $z = t$, para $t \in [0; 6\pi]$. Podemos usar el comando plot3ó el comando ezplot3.

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))

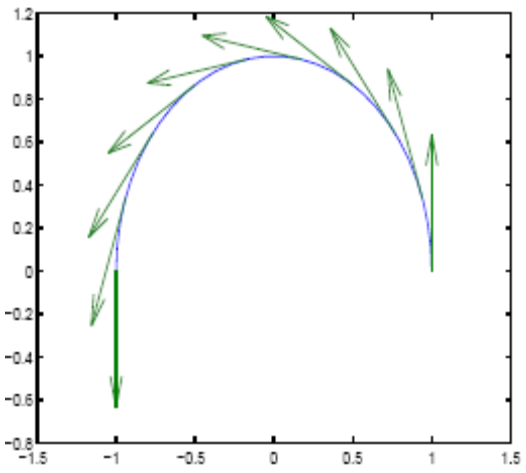
Si se quiere que aparezcan los vectores tangente, se usa la función quiver.

>>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



2.6. 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);

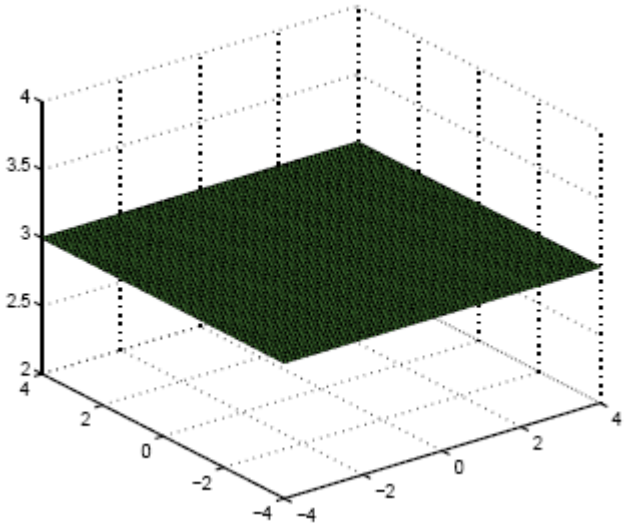
surf(x,y,z)

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)
```



2.7. 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])
```

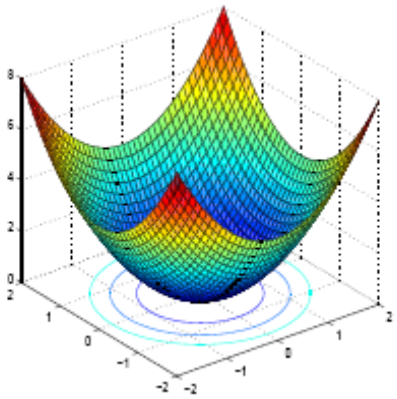
y aparecen representadas las curvas de nivel $f(x; y) = ci$, para $i = 1; :::; n$.

Ejemplo. Representar la superficie de ecuación $z = x2 + y2$, para $(x; y)$ *2* [-2*;* 2] ϵ [-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



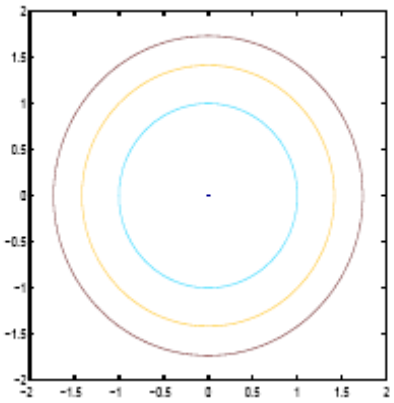
Si sólo se buscan las curvas de nivel (no se necesita dibujar la superficie), basta escribir

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

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

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

y se obtiene



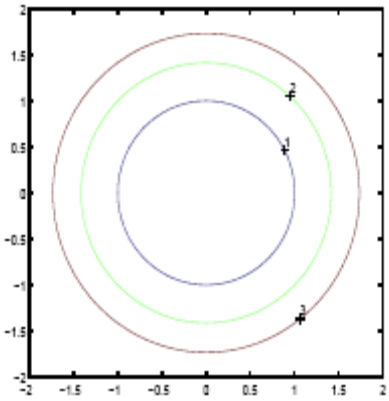
Si se quiere que aparezcan etiquetas sobre cada curva de nivel que reflejen el valor de la constante *c*, se usa la función clabel

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

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

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

resultando

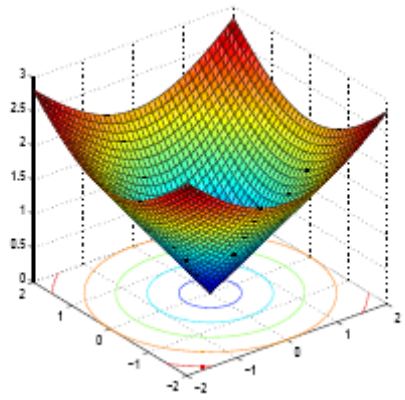


Cuando necesitamos representar una superficie y deseamos que aparezcan las curvas de nivel, hay una forma muy cómoda que consiste en emplear la función surfc(x,y,z).

Ejemplo. Gráfica de la superficie $z = px2 + y2$ 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



2.8. >Cómo exportar gráficos?

a) Word. Una vez que tenemos el gráfico en la ventana gráfica, vamos a Edit (en le ventana gráfica) y hacemos clic en Copy figure. A continuación abrimos el documento word y pegamos.

b) Latex. Primero debemos salvar el gráfico de forma adecuada. Vamos a File (en la ventana gráfica) y hacemos clic en Save as. Se abre una ventana (Save as) en la que debemos dar el nombre con que vamos a designar el archivo que contendrá el gráfico. Automáticamente, Matlab le añade la extensión que corresponda. Normalmente,

elegiremos el tipo EPS file, en cuyo caso la extensión que Matlab añade es eps. Ahora es el momento de guardar el archivo creado, nombre.eps, en la carpeta que contiene el archivo Latex donde queremos incluir el gráfico. Ahora debemos preparar el archivo Latex, lo que exige realizar los siguientes dos pasos:

1) En el archivo principal y antes de la instrucción *n*begin*f*document*g*, debemos incluir *N*usepackage*f*graphicx*g*

2) En el lugar donde vamos a incluir el gráfico:
*n*begin*f*center*g*
*n*includegraphics[width=5cm,height=6cm]*f*nombre.eps*g*
*n*end*f*center*g*

3. Programación

3.1. 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).

3.2. 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.

3.3. Cálculo de integrales usando un archivo de función

Vamos a ver una forma de calcular integrales simples y reiteradas creando un archivo de función con el integrando. Este método nos servirá, además, para resolver otro tipo de problemas que tienen en común que necesitamos hacer manipulable una función. El primer paso consiste en crear un archivo de función con una función que iremos cambiando en cada problema concreto. Este archivo puede llamarse f.m. Ejemplo.

Deseamos calcular $\int_0^1 e^{x^2}\,dx$.

Creamos el archivo f.m:

```
function y=f(x)
```

%y=F(x) es el integrando de una integral simple

y=exp(x.^2);

Ahora usamos la función quad para calcular una integral simple.

>>integrando=@f;% Hacemos manipulable la función quad(integrando,0,1) ans 1.4627

Veamos ahora cómo se determina una integral reiterada con la función dblquad.

Ejemplo. Calcular $\int_0^1 \left[\int_{-1}^{1} xy\,dx \right] dy.$

En primer lugar, creamos un archivo de función con el integrando:

function z=fun(x,y)

% z=FUN(x,y) es el integrando de una integral reiterada

z=x.*y;

Finalmente, usamos la función dblquad:

>>integrando=@fun;

a=dblquad(integrando,*xmin; xmax; ymin; ymax*)

ans 1.2309e-017

En nuestro caso, *xmin* = -1, *xmax* = 1, *ymin* = 0 e *ymax* = 1.

Finalizamos esta sección indicando que si necesitamos evaluar $f(a)$ y disponemos de un archivo de función f.m con la función $y = f(x)$, podemos usar la función feval:

g=@f;

feval(g,a)

y, pulsando enter, obtenemos el valor $f(a)$.

3.4. La función fplot

Para dibujar la curva $y = y(x)$ con el comando fplot, debemos crear un fichero de función con la función $y(x)$. Supongamos que a este fichero le hemos llamado fun.m. Para manipular la función, pondremos f=@fun. Veamos esto con un ejemplo. Ejemplo. *Usar fplot para dibujar la curva* $y = x2$ sen x

Empezamos creando un archivo de función para $y(x)$:

function y=fun(x)

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

Una vez creado este archivo con la funcin, procedemos a dibujar la curva $y = y(x)$ con fplot

>>f=@fun;

fplot(f,[xmin,xmax])

grid on

Si se quiere que la gráfica recoja con mayor detalle el intervalo [ymin,ymax]

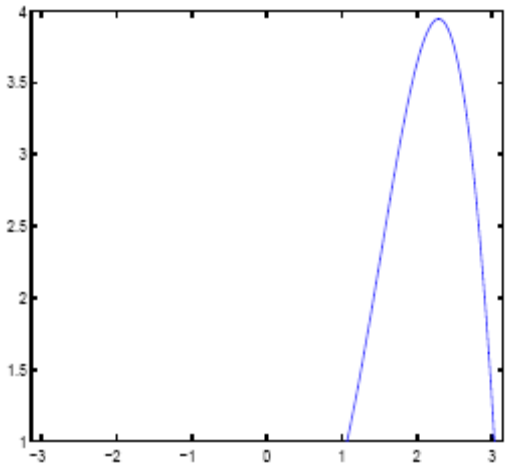
>>fplot(f,[xmin,xmax ymin ymax])

Ejemplo. *Dibujar la curva y = x*2 sen *x en* [-π; π] *con un zoom en [1,4]*.

>>f=@fun;

fplot(f,[-pi,pi,1,4])

Al pulsar enter, obtenemos



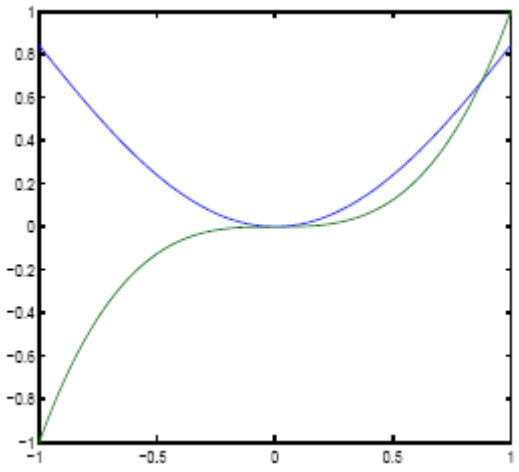
Otra forma de manejar la función fplot es la siguiente

>>fplot('x*sin(x)',[0,2])

Con la función fplot también se pueden dibujar dos curvas a la vez

>>fplot('[x*sin(x),x^3]',[-1,1])

y resulta



3.5. 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.

3.6. Subfunciones

Un archivo de función puede contener el código de más de una función. La primera función (la función principal) que aparece en el archivo es la que da el nombre

a éste y las otras se llamarán subfunciones (sólo son visibles para la función principal o para otra subfunción en el mismo archivo. Ejemplo. Un archivo de función que determina la media y la mediana de un vector fila.

```
function [media,mediana]=estadistico(v)
% Estadistico encuentra la media y la mediana con subfunciones internas
n=length(v);
media=med(v,n);
mediana=medn(v,n);
function a=med(v,n)
% Calcula la media
a=sum(v)/n;
function b=medn(v,n)
%Calcula la mediana
w=sort(v);
if rem(n,2)==1
b=w((n+1)/2);
else
b=(w(n/2)+w((n/2)+1))/2;
end
```

4. Ficheros creados en Matlab

Dibujogeneral.m

Es un programa para dibujar funciones

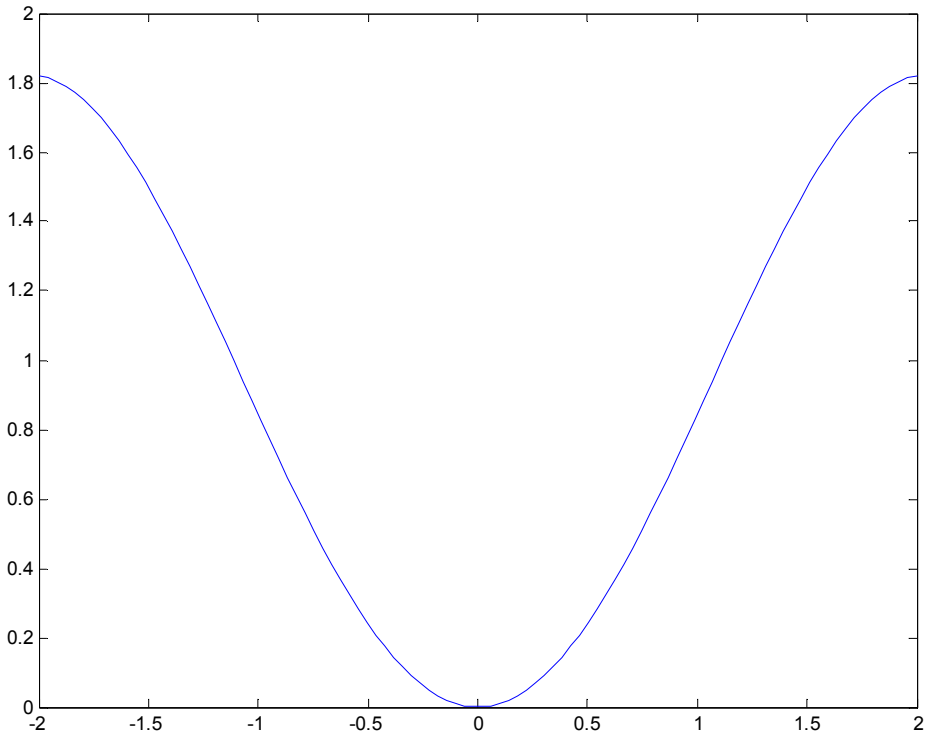
```
>> edit dibujogeneral.m
>> %definimos la funcion que queremos dibujar
```

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

>> edit fun.m

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

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



Bisección.m

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.

Cálculo con Matlab

>> edit biseccion.m

```
function p=biseccion(f,a,b,TOL)
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;
```

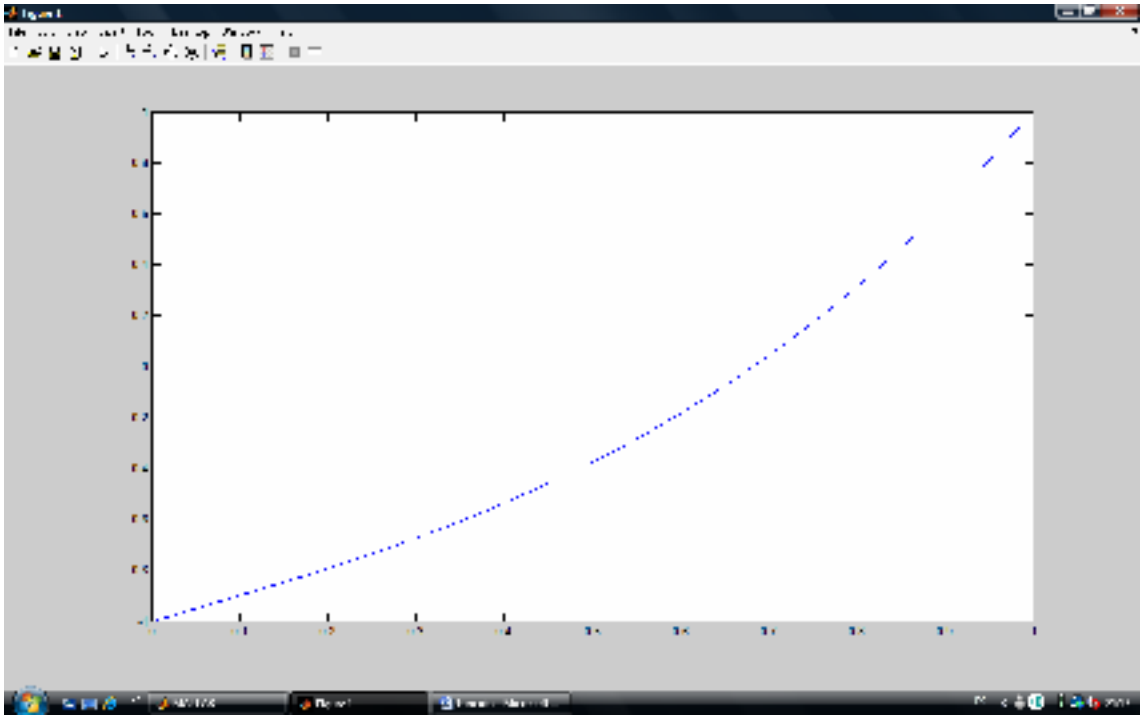
Para aplicar el método de bisección primero tenemos que definir una función, y una vez definida la función, podemos llamar al programa y ya nos da el resultado.

Ejercicio 1

>> edit f1.m

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

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



>> biseccion('f1',0,1,10^(-6))

ans =

0.68232774734497

>> f1(0.68232774734497)

ans =

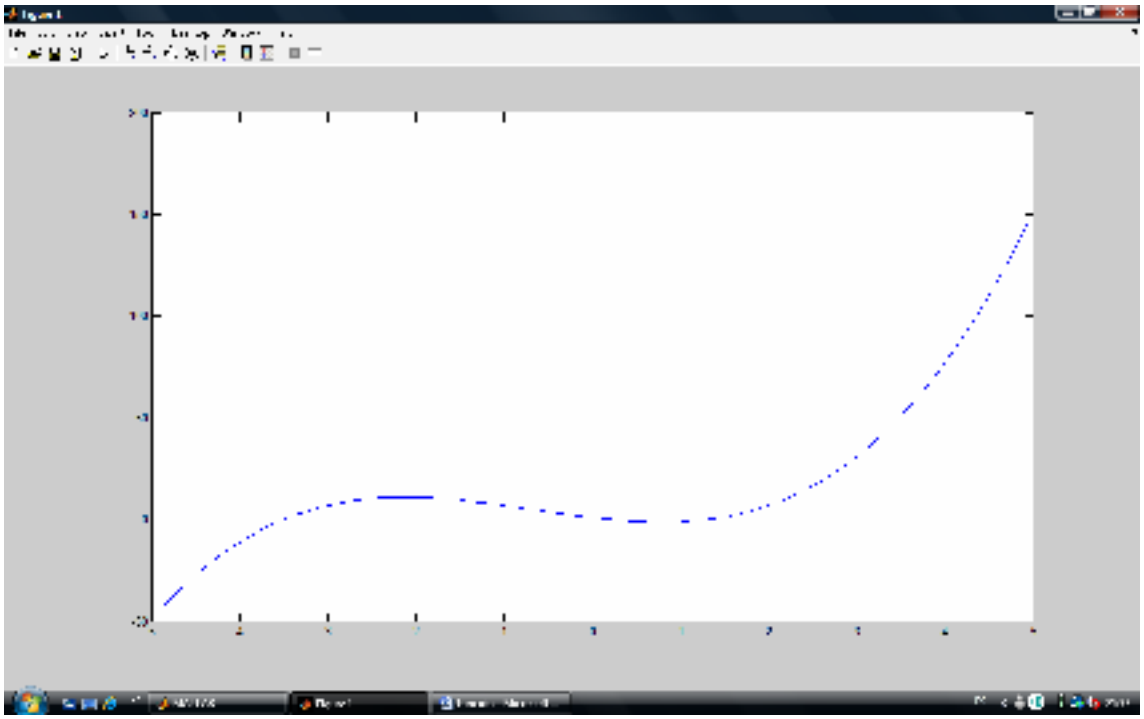
-1.353736912568238e-007

Ejercicio 2

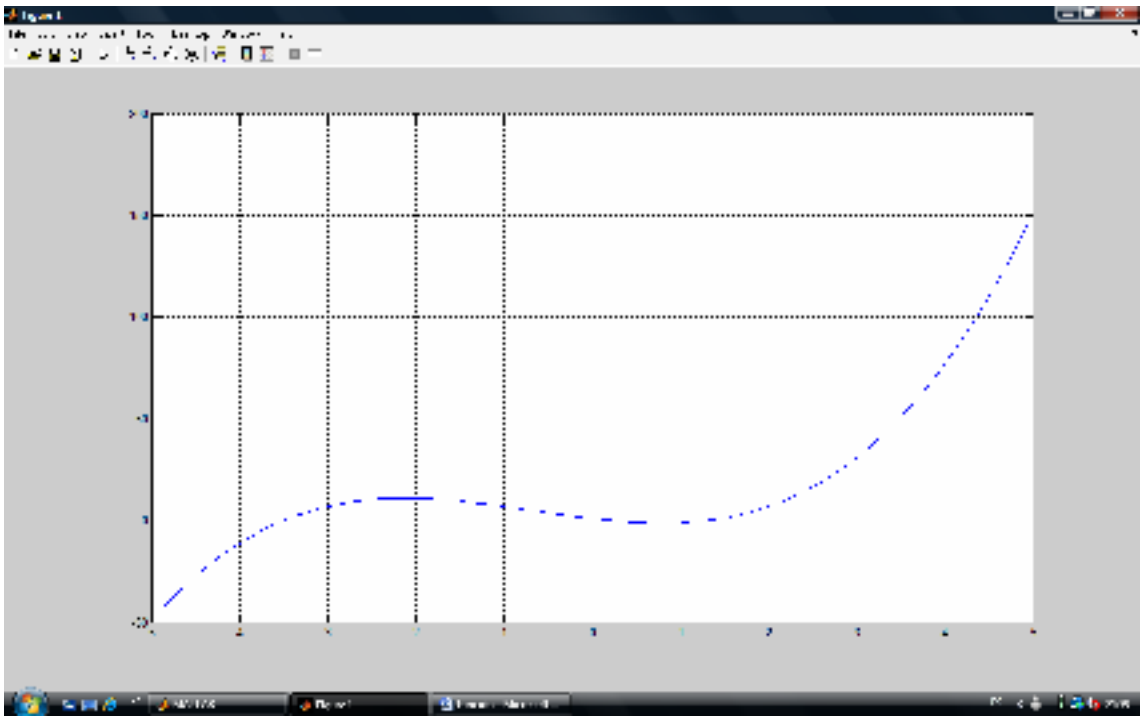
>> edit f1.m

```
function y=f1(x)
y=x.^3+2.*x.^2-5.*x+1;
```

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



>> grid



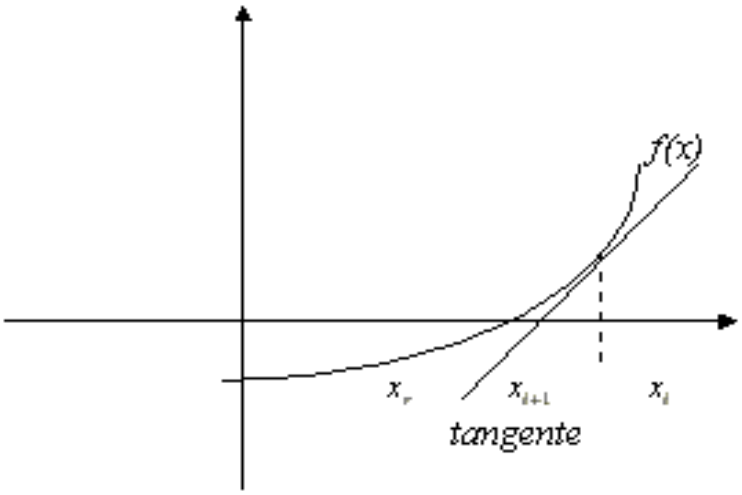
>> biseccion('f1',-5,-3,10^(-6))

```
ans =
  -3.50701856613159

>> biseccion('f1',-1,0.5,10^(-6))
ans =
   0.22187602519989

>> biseccion('f1',0.5,2,10^(-6))
ans =
   1.28514230251312

>> f1(-3.50701856613159)
ans =
    1.393128190585458e-006

>> f1(0.22187602519989)
ans =
    5.434297573048141e-007

>> f1(1.28514230251312)
ans =
   -9.136798224673726e-007
```

Newtonsol.m

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 $f(x)$!

Cálculo con Matlab

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 esta 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)
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 raiz (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
```

El valor que obtenemos es bastante próximo a 0.

Ajusteexp.m

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.

>>edit ajusteexp.m

```
function [a,b]=ajusteexp(x,y);
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');
```

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.

Ajusteracional.m

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

>>edit ajusteracional

```
function [a,b]=ajusteracional(x,y)
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')
```

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.

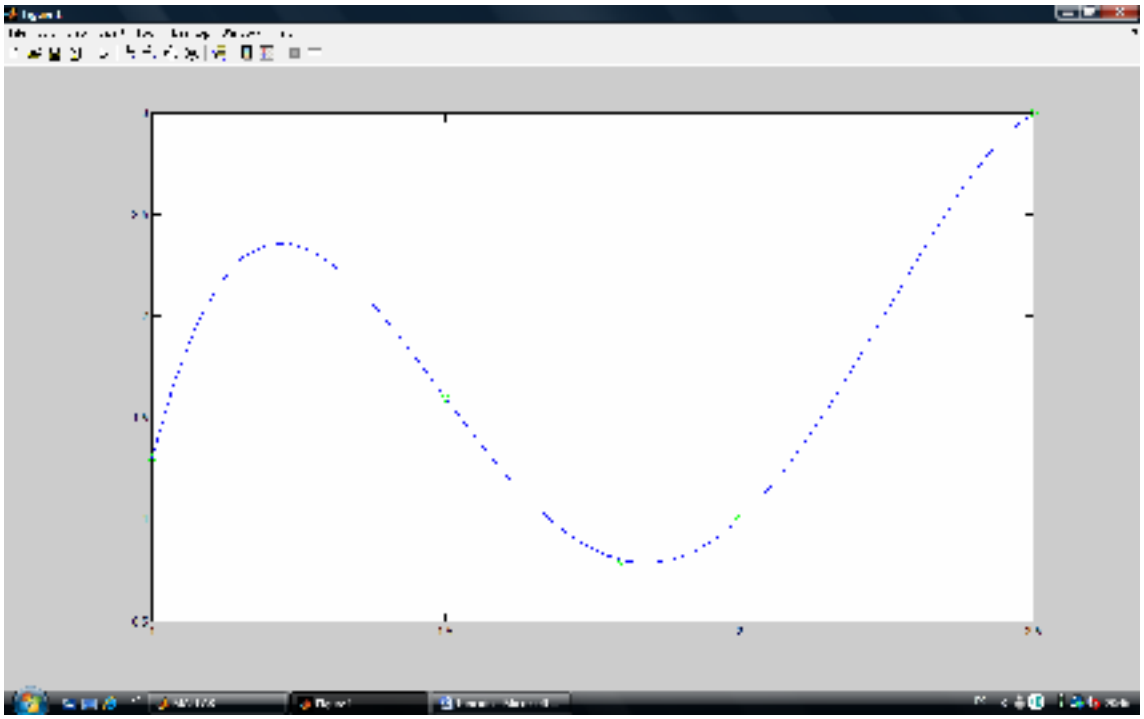
Otro problema estrechamente ligado con el de la interpolación es la aproximación de una función complicada por una más simple. Si tenemos una función cuyo cálculo resulta costoso, podemos partir de un cierto número de sus valores e interpolar dichos datos construyendo una función más simple. En general, 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];
>> y=[1.3,1.6,0.8,1,3];
>> p=polyfit(x,y,4)
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);
>> yy=polyval(p,xx);
>> plot(xx,yy,x,y,'og')
```



```
>> polyval(p,1.2)
ans =
   2.35371428571408
>> polyval(p,1.37)
ans =
   2.08018847499986
```

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

Cálculo con Matlab

>>edit EMC.m

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

Ejercicios

Suma de cubos

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

```
>>edit descomposicion.m
function [lista]=descomposicion(N1,N2)
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

>> [lista]=descomposicion(10000,100000)

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

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.

>>edit descomposicion3.m

```
function [lista]=descomposicion(N1,N2)
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==3 lista=[lista;sublista]; end
end
```

>>[lista]= descomposicion3(1,10000)

lista =

[]

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

lista =

[]

>> [lista]= descomposicion3(100000,10000000) [lista]= descomposicion3(1,10000)

Pero no existen en estos intervalos números que cumplan la condición que hemos impuesto.

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.

>>edit ejerc2

```
n=2;s=0;
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;
   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
```

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.

>>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;
```

>>[T,M]=hipoteca1(300000,3,1000)

```
T =

  5.5421e+005


M =

   555
```

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.

>> edit ejercintegral.m

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

Hacemos el dibujo general y buscamos la raiz por el método bisección. Por último calculamos el área.

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

Ejercicio integral 1

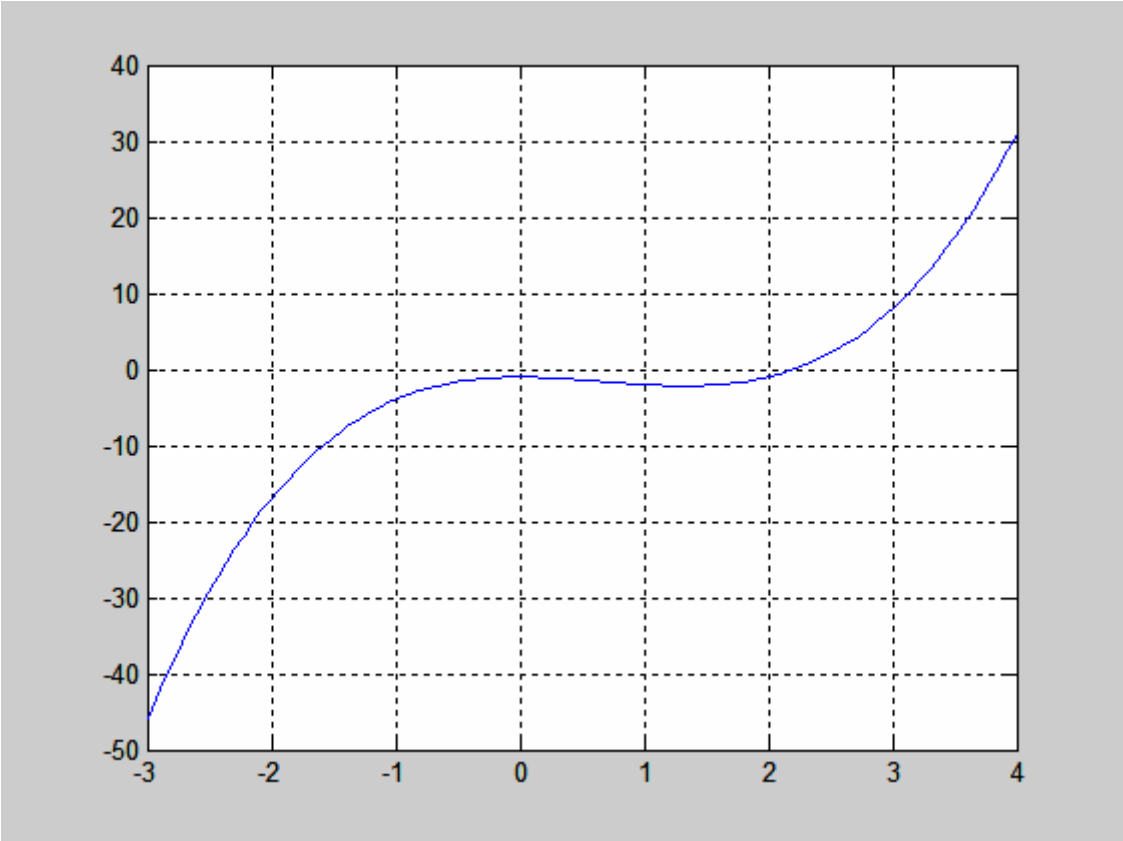
Calcular el área que forma la curva y= $x^3 + 2x^2 - 1$ con el eje x, entre x=-3 y x=4

>>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 donde se encuentra los números próximos a la raíz.

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



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)
ans =

    1.48
>> area=-quadl('ejercintegral',-2,1.48)+quadl('ejercintegral',1.48,4)

area =

  686.9681
```

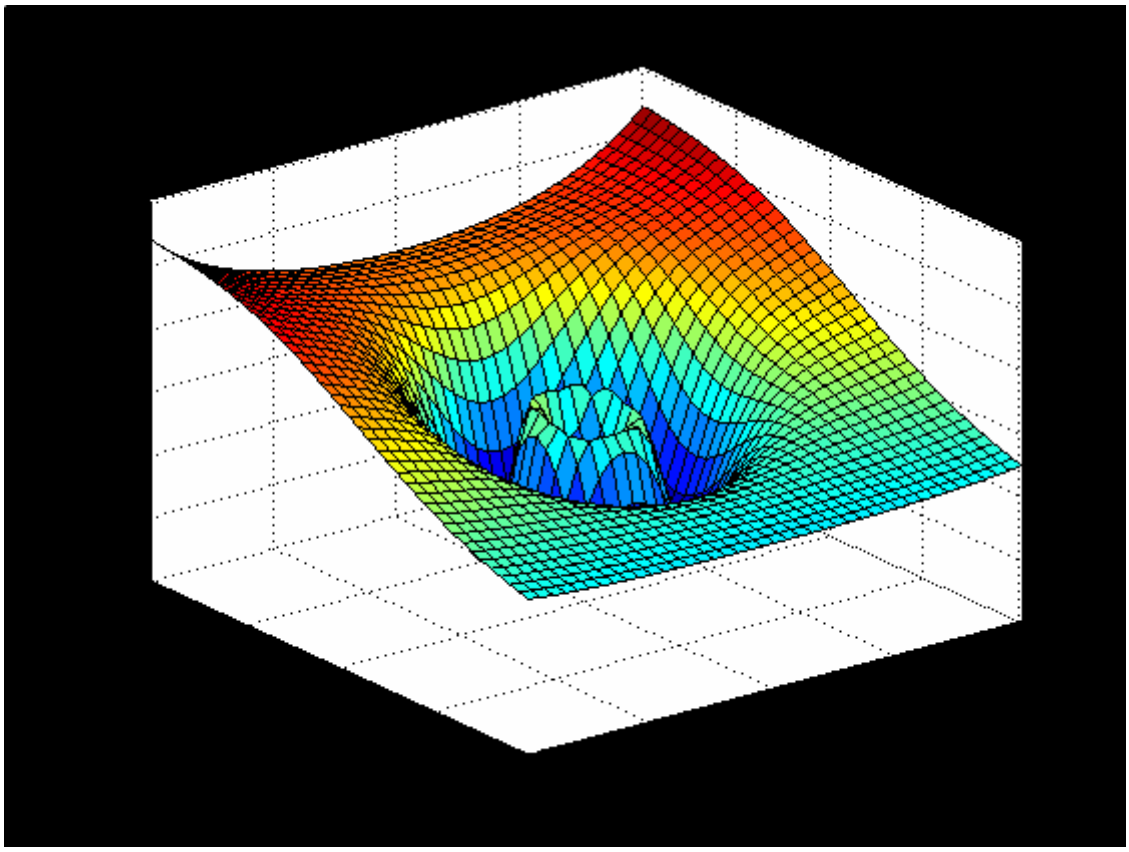
Ejercicio máximos y mínimos

Calcular los máximos y los mínimos de la función

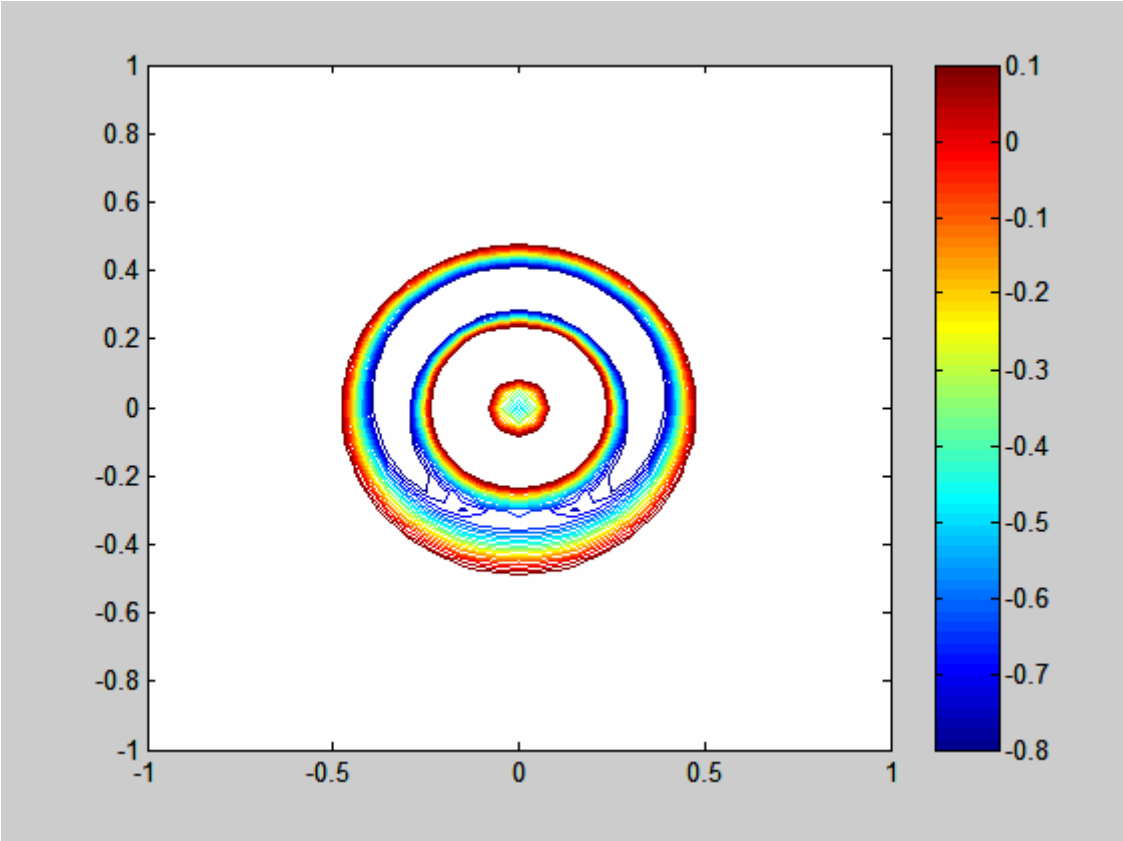
```
>>edit ejercicioa.m
```

```
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)
>>z=ejercicioa(x,y)
>>surf(x,y,z)
```



```
>>contour(x,y,z,-0.8:0.05:0)
```



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.

>>edit ejercicioa1.m

```
function z=ejercicioa(xx)
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])

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.

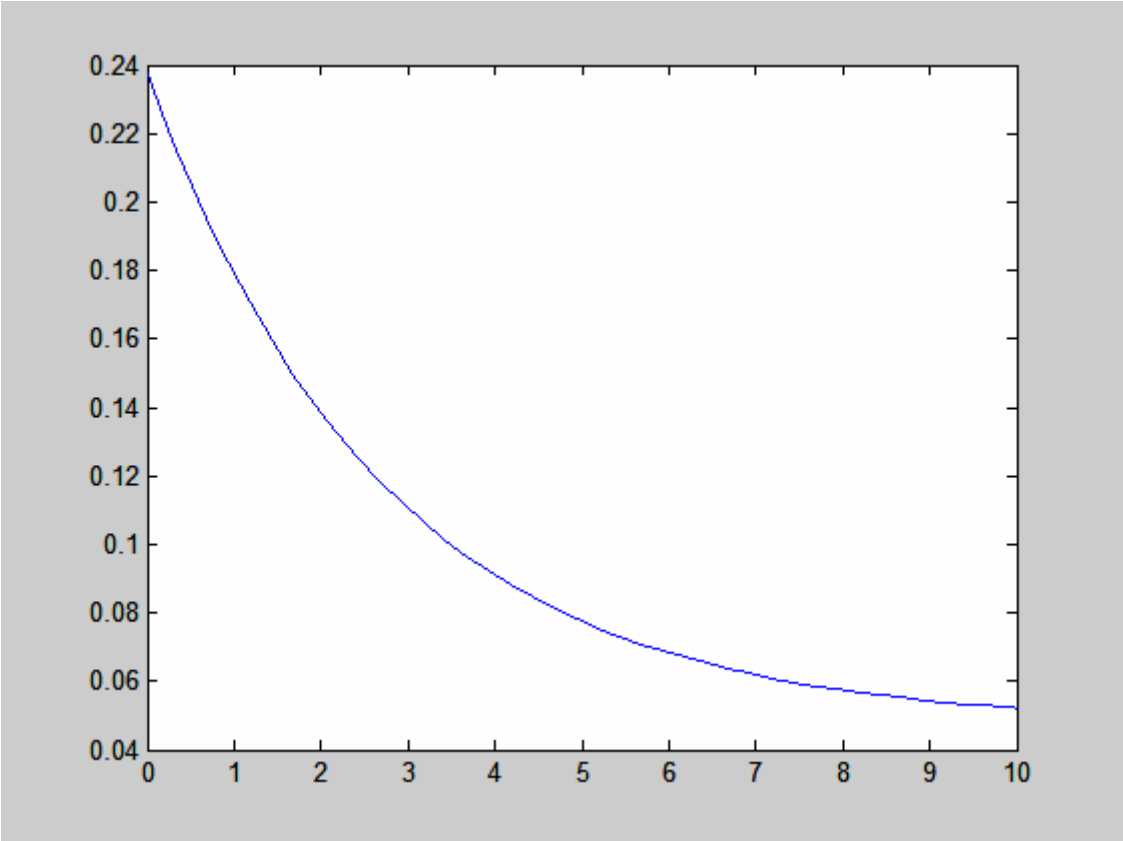
Ejercicio del lago

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?

>>edit lago.m

```
function z=lago(t,x)
z=175*0.0001-175.*x/475;
```

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

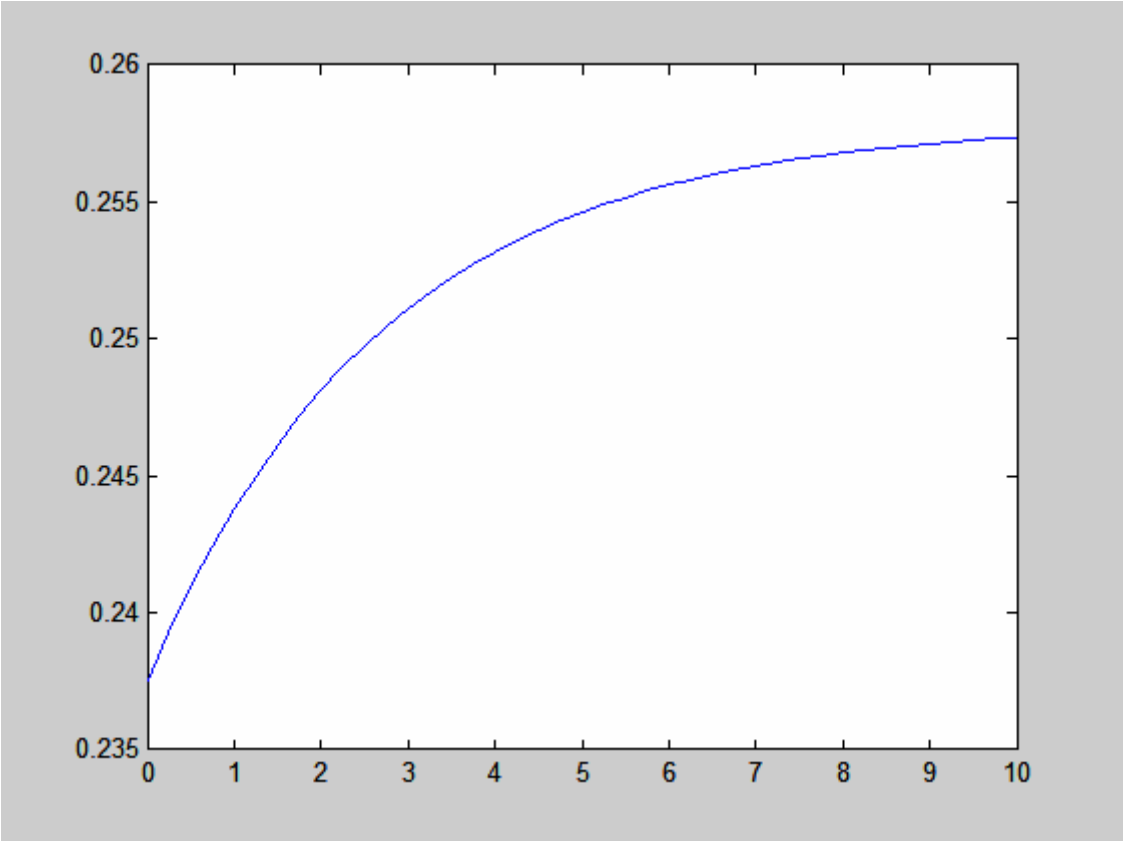


Para saber cuando el lago está al 0.02%, tenemos que modificar el programa que hemos editado, el lago.m

>>edit lago.m

```
function z=lago(t,x)
z=475*0.0002-175.*x/475;
```

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



Cinética de reacciones químicas

Utilizaremos un modelo básico que es el propuesto por Michaelis y Menten (1913) para reacciones encimáticas. Estas reacciones constan:

$$S+E \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} SE \;;\; SE \xrightarrow{k_2} P+E$$

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.

Lenguaje Matlab

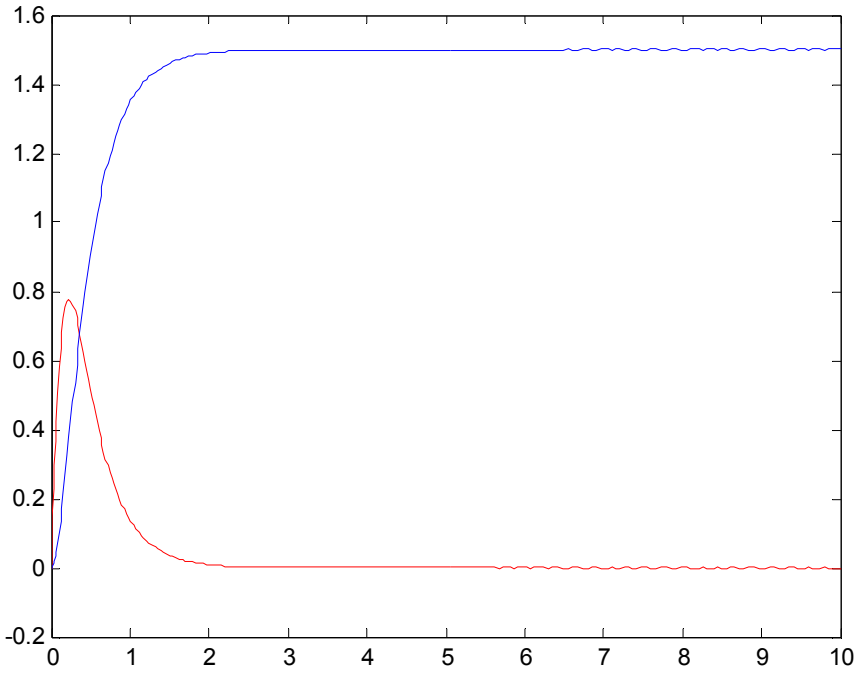
Creamos en matlab el fichero cinética.m

>>edit cinetica.m

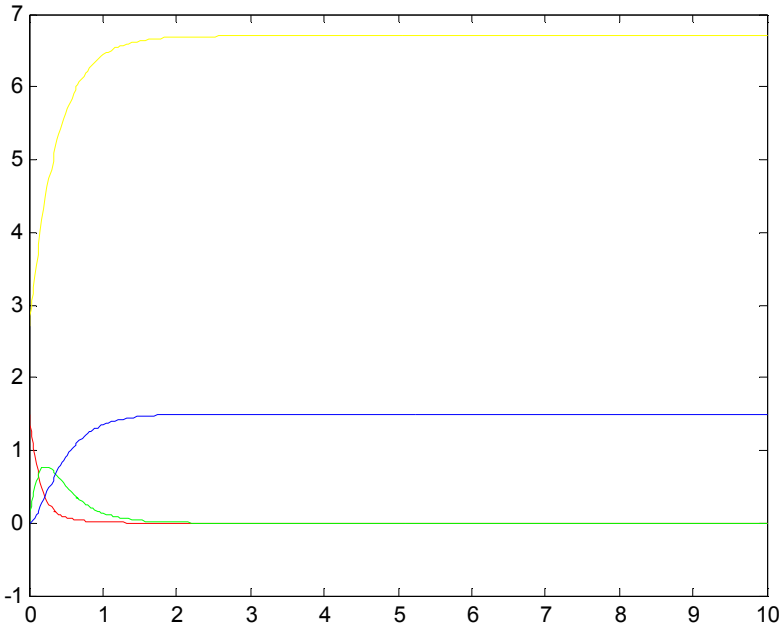
```
function z=cinetica(t,yy)
s=yy(1); e=yy(2); c=yy(3); p=yy(4); % indicamos las cuatro ecuaciones
z(1)=(-2.*e.*s)+c;
z(2)=(2.*e.*s)+(4.*c);
z(3)=(2.*e.*s)-(4.*c);
z(4)=3.*c;
z=z';
```

>>[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



Autocatálisis: activación e inhibición

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

$$A+X \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} 2X$$

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.

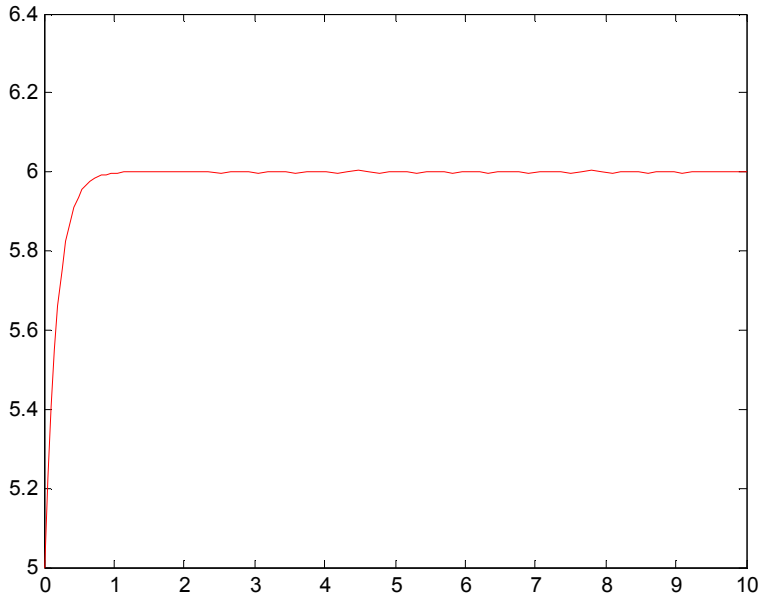
% Creamos en matlab el fichero autocatalisis.m

>>edit autocatalisis.m

```
function z=autocatalisis(t,x)
 z=(6.*x)-x.^2;
```

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

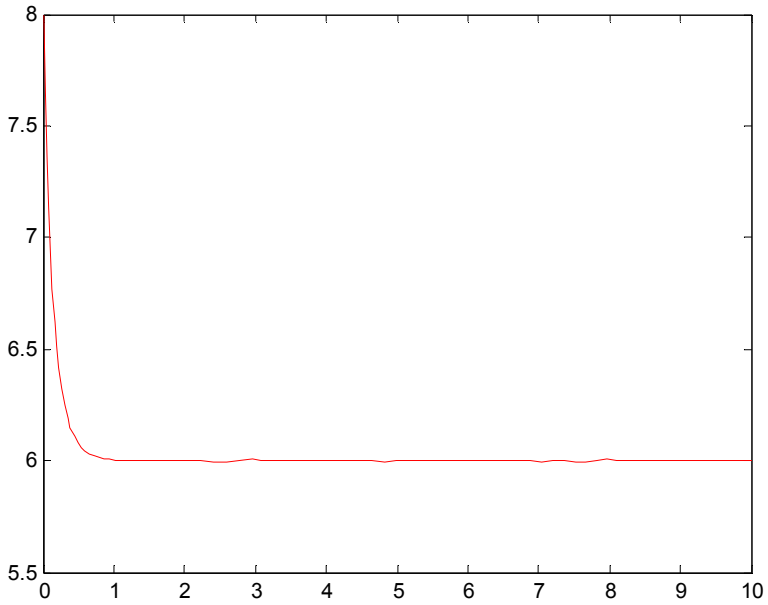
>>plot (t,x,'r') % dibujamos la variación de la concentración de x



Cambiamos la condición inicial para la concentración de x, siendo esta ahora de 8; por lo que se cambia en el comando ode45.

>>[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 puede observarse, la reacción tiende a estabilizarse a medida que se acerca a 6.

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.**

Boris Belousov se dio cuenta en la década de los 50, 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 iones de bromo (V). Cuando un ion se oxida pierde electrones y cuando se reduce los gana. Se utiliza un indicador redox llamado ferroína, así, los iones de cerio en el

estado reducido (Ce^{3+}) dan una coloración roja, y al oxidarse se convierten en Ce^{4+} (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 voltímetro). 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.

Más tarde, en 1961, un estudiante graduado llamado AM Zhabotinsky redescubierto esta reacción secuencia; [2] Sin embargo, los resultados de estos hombres del trabajo todavía no se difundió ampliamente, y no era conocido en Occidente hasta una conferencia en Praga en 1968.

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.

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 contener 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. UnN ejemplo de reacción oscilante 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	$1.28 M^{-1}s^{-1}$
K_2	$2.4 \times 10^6 M^{-1}s^{-1}$
K_3	$33.6 M^{-1}s^{-1}$
K_4	$3 \times 10^3 M^{-1}s^{-1}$
k_c	$1 M^{-1}s^{-1}$

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

$$x = \frac{2k_4X}{k_3A}\,;\quad y = \frac{k_4Y}{k_3A}\,;\quad z = \frac{k_c k_4 BZ}{(k_3A)^2}\,;\quad \tau = k_c Bt$$

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

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.

Lenguaje Matlab

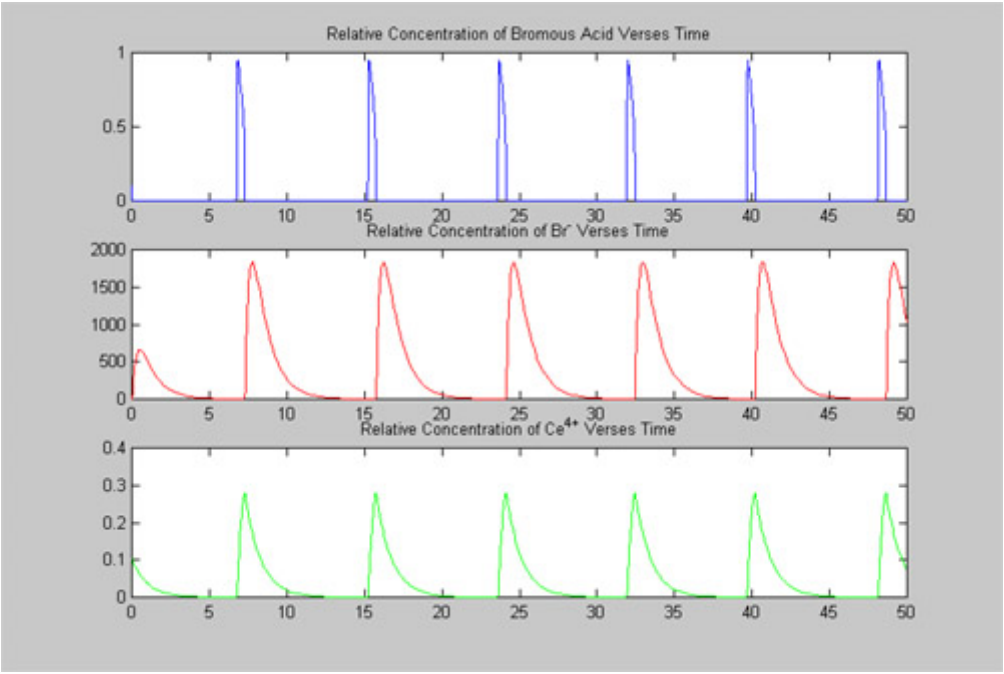
>>edit oscilacion.m

```
function d=oscilacion(t,yy)
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, z.

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

Puede observarse el comportamiento oscilatorio en la representación



Modelo de Prigogine-Lefever-Bruselador.

Las ecuaciones propuestas por este mecanismo son:

$$\rightarrow A \underset{k_{-1}}{\overset{k_1}{\rightleftharpoons}} X$$

$$\rightarrow B + X \underset{k_{-2}}{\overset{k_2}{\rightleftharpoons}} Y + D \rightarrow$$

$$2X + Y \underset{k_{-3}}{\overset{k_3}{\rightleftharpoons}} 3X$$

$$X \underset{k_{-4}}{\overset{k_4}{\rightleftharpoons}} E \rightarrow$$

El planteamiento general de flujos y afinidades es el siguiente:

$$J_1 = k_1 A - k_{-1} X \qquad \mathbb{A}_1 = RT \ln K_1 - RT \ln \frac{X}{A}$$

$$J_2 = k_2 BX - k_{-2} DY \qquad \mathbb{A}_2 = RT \ln K_2 - RT \ln \frac{Y}{B}\frac{D}{X}$$

$$J_3 = k_3 X^2 Y - k_{-3} X^3 \qquad \mathbb{A}_3 = RT \ln K_3 - RT \ln \frac{X}{Y}$$

$$J_4 = k_4 X - k_{-4} E \qquad \mathbb{A}_4 = RT \ln K_4 - RT \ln \frac{E}{X}$$

donde se supone que A, B, D y E se mantienen constantes. Asimismo, igual que en el modelo anterior, se considerar´a que los procesos de retorno ocurren con velocidad despreciable, es decir:

$$k_{-1}, k_{-2}, k_{-3}, k_{-4} \to 0$$

con lo cual, además de simplificar el tratamiento matemático, se garantiza que el sistema se encuentra lejos del equilibrio. Si se supone, además, que las constantes cinéticas:

$$k_1 = k_2 = k_3 = k_4 = 1$$

se consigue simplificar el análisis sin que el resultado obtenido pierda generalidad.

Análisis de la estabilidad

$$J_X = J_1 + J_3 - J_2 - J_4 = \frac{dX}{dt} = A + X^2 Y - (B+1)X$$

$$J_Y = J_2 - J_3 = \frac{dY}{dt} = BX - X^2 Y$$

Los valores de X e Y en el estado estacionario son:

$$X_s = A \qquad Y_s = \frac{B}{A}$$

Cuando estos valores se ven perturbados:

$$X = X_s + \delta X \qquad Y = Y_s + \delta Y$$

La evolución cinética de las fluctuaciones viene descrita por las siguientes ecuaciones:

$$\frac{d(\delta X)}{dt} = (B-1)(\delta X) + A^2(\delta Y) + \frac{B}{A}(\delta X)^2 + 2A(\delta X)(\delta Y) + (\delta X)^2(\delta Y)$$
$$\frac{d(\delta Y)}{dt} = -B(\delta X) - A^2(\delta Y) - \frac{B}{A}(\delta X)^2 - 2A(\delta X)(\delta Y) - (\delta X)^2(\delta Y)$$

Si se tiene en cuenta que los términos no lineales resultan despreciables si se consideran fluctuaciones muy pequeñas, las ecuaciones se reducen a:

$$\frac{d(\delta X)}{dt} \simeq (B-1)(\delta X) + A^2(\delta Y)$$
$$\frac{d(\delta Y)}{dt} \simeq -B(\delta X) - A^2(\delta Y)$$

cuya integración determinaría la evolución de las fluctuaciones con el tiempo. La solución general es de la forma:

$$\delta X(t) = \delta X(0)\ e^{\omega t}$$
$$\delta Y(t) = \delta Y(0)\ e^{\omega t}$$

Sustituyendo estas funciones y sus derivadas en las ecuaciones anteriores se llega al establecimiento del siguiente sistema de ecuaciones:

$$\delta X(0)[\omega - (B-1)] - \delta Y(0)A^2 = 0$$
$$\delta X(0)B + \delta Y(0)(\omega + A^2) = 0$$

cuya soluciones compatibles están determinadas por los valores de ω que anulan el determinante de los coeficientes:

$$\delta X(0)\delta Y(0)\begin{vmatrix} [\omega - (B - 1)] & A^2 \\ B & (\omega + A^2) \end{vmatrix} = 0$$

es decir, las raíces de la ecuación:

$$\omega^2 + (A^2 + 1 - B)\omega + A^2 = 0$$

que viene dados por la siguiente ecuación:

$$\omega = -\frac{A^2 + 1 - B}{2} \pm \frac{\sqrt{(A^2 + 1 - B)^2 - 4A^2}}{2}$$

Las soluciones de la ecuación $\omega^2 + (A^2 + 1 - B)\omega + A^2 = 0$ pueden ser reales o imaginarias según el signo del discriminante en $\omega = -\frac{A^2 + 1 - B}{2} \pm \frac{\sqrt{(A^2 + 1 - B)^2 - 4A^2}}{2}$. El comportamiento será estable o inestable dependiendo de la parte real; de modo que será estable si $(A^2 + 1 - B) > 0$, e inestable en el caso contrario, en el que $(A^2 + 1 - B) < 0$.

En relación con los valores del discriminante pueden darse las situaciones siguientes:

1. $A^2 + 1 - B = 0$

En este caso solamente existen raíces imaginarias puras, $\omega = \pm Ai$, con lo que las funciones que determinan la evolución de las fluctuaciones con el tiempo son de carácter periódico. Se observaría un comportamiento inestable con oscilaciones no amortiguadas, caracterizadas por el estado inicial y los valores de los parámetros A y B.

2. $(A^2 + 1 - B)^2 < 4A^2$

Las raíces en este caso poseen parte real y parte imaginaria. Si se extrae la raíz cuadrada a ambos miembros de la desigualdad, se obtiene: $\pm(a^2 + 1 - B) < \pm 2A$, de donde resulta que es preciso que los valores de B se encuentren en el intervalo:

$$(A^2 - 2A + 1) < B < (A^2 + 2A + 1)$$

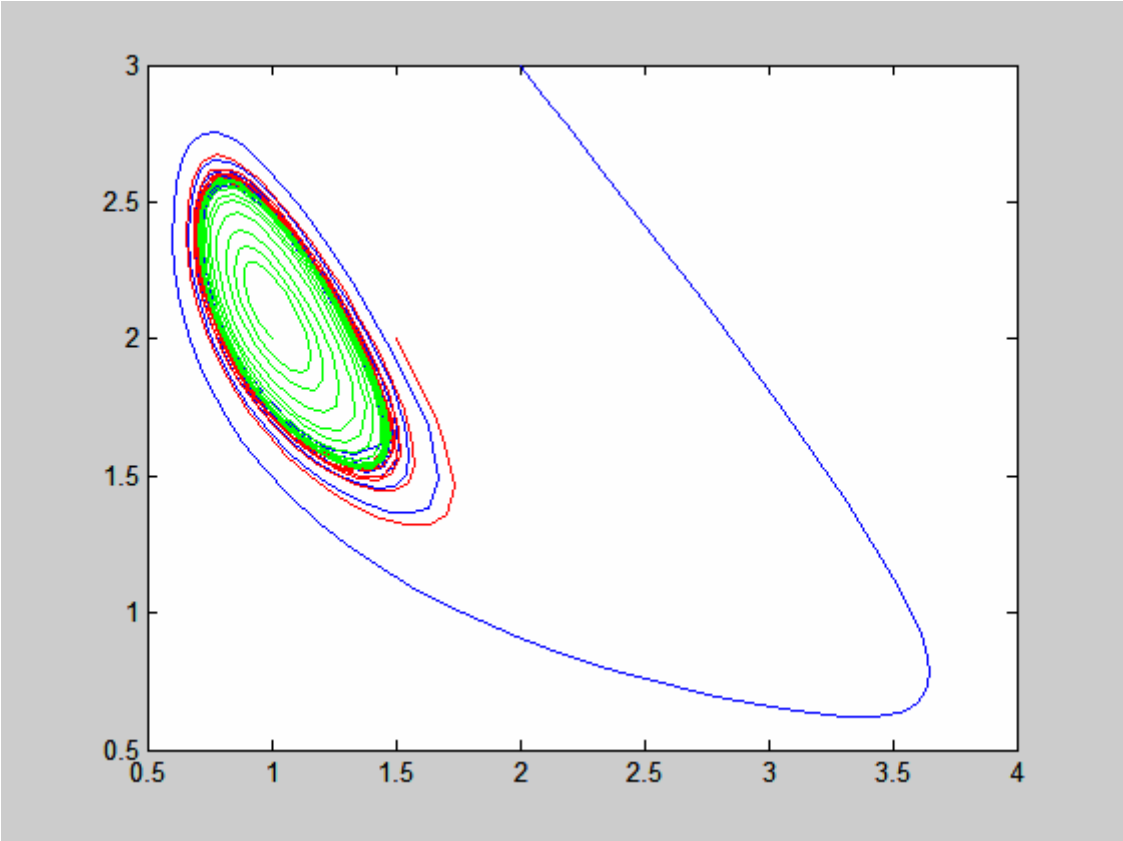
A continuación se pasa a programarlo en Matlab, para representar el ciclo límite.

Lenguaje Matlab

>>edit bruselator.m

```
function z=bruselator(t,yy)
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],[1.9;2.2]); );% buscamos el ciclo limite
>> plot (y(:,1),y(:,2))
>> plot (y(:,1),y(:,2),'g')
>> [t,y]=ode45('bruselator',[0,20],[2;3]);
>> plot (y(:,1),y(:,2),'r')
>> [t,y]=ode45('bruselator',[0,100],[2;3]);
>> hold off
>> plot (y(:,1),y(:,2))
>> [t,y]=ode45('bruselator',[0,100],[1.5;2]);
>> hold on
>> plot (y(:,1),y(:,2),'r')
>> [t,y]=ode45('bruselator',[0,100],[1;2]);
>> plot (y(:,1),y(:,2),'g') %la dibujamos cerca del ciclo límite
```



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

Problema 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.

Lenguaje Matlab

>>edit p.m

```
function w=p(t,xx)
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
```

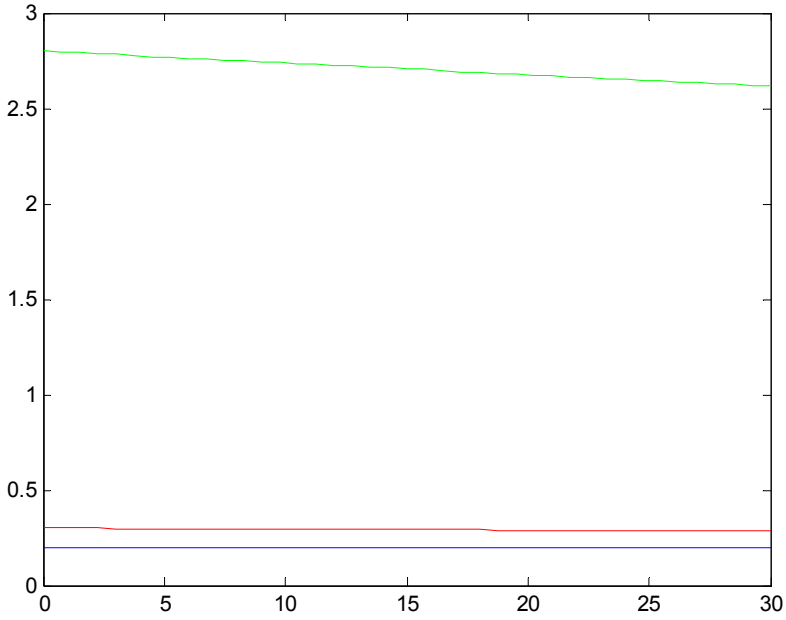
12.0000
12.7500
13.5000
14.2500
15.0000
15.7500
16.5000
17.2500
18.0000
18.7500
19.5000
20.2500
21.0000
21.7500
22.5000
23.2500
24.0000
24.7500
25.5000
26.2500
27.0000
27.7500
28.5000
29.2500
30.0000

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.

Lenguaje Matlab

>>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 =

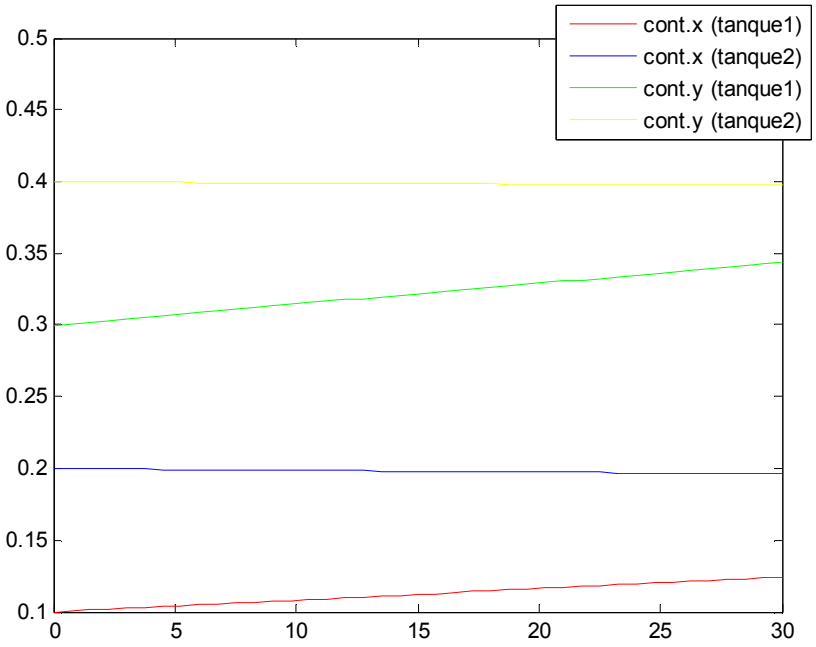
```
         0
    0.7500
    1.5000
    2.2500
    3.0000
    3.7500
    4.5000
    5.2500
    6.0000
    6.7500
    7.5000
    8.2500
    9.0000
    9.7500
   10.5000
   11.2500
   12.0000
   12.7500
   13.5000
   14.2500
```

```
   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.

Lenguaje Matlab

>> 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 que el programa que hemos hecho nos da un resultado coherente

>> tanqueH(0.5)

ans =

```
    1
>> tanqueH(1.7)
ans =
    0
```

El programa con 1.7 no se cumple, por ello y=0. Para ver el volumen del tanque creamos el archivo volumentanque.m

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

```
function y=volumentanque(t)
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, por ello, lo modificamos para que contengan ciclos de tres y lo llamamos volumen tanque tres

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

```
function y=volumentanque3(t)
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 al tiempo 3.5, el volumen del tanque permanece constante.

```
 >>volumentanque3(2.7)
ans=
        2
```

Sin embargo, a tiempo 2.7, el volumen del tanque va aumentando con la variación que da el tercer ciclo.

Aplicaciones en química analítica

pH de disoluciones de un ácido poliprótico

Ejercicio 1- Muchos ácidos débiles contienen protones ionizables y forman equilibrios múltiples en disolución acuosa. La solución del sistema de ecuaciones simultáneas, que resulta del análisis de todos los equilibrios, requiere de un método numérico. Sin embargo, cuando los valores de las constantes de equilibrio difieren en un factor superior de 10^3, se hacen algunas aproximaciones para obtener algoritmos más simples.

Se desea determinar el pH de una disolución de H_2SO_3, donde K_1=0.017, para concentraciones que se encuentren en un rango de 0.0010-0.1 M. También se requiere evaluar, a través de un gráfico, el error que se comete al utilizar una ecuación aproximada que surge de considerar que $C_a >>> [H_3O^+]$.

A partir del primer equilibrio del H_2SO_3

$$H_2SO_3 + H_2O \rightleftharpoons HSO_3^- + H_3O^+$$

Se obtiene que:

$$K_1 = \frac{\left|HSO_3^-\right|\left|H_3O^+\right|}{\left|H_2SO_3\right|}$$

La expresión de equilibrio conduce a la ecuación cuadrática:

$$\left|H_3O^+\right|^2 + K_1\left|H_3O^+\right| - K_1C_a = 0$$

Cuya solución aproximada es:

$$\left|H_3O^+\right| = \sqrt{K_1 C_a}$$

Para editar un programa tenemos que utilizar la ecuación anterior y la ecuación cuadrática:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

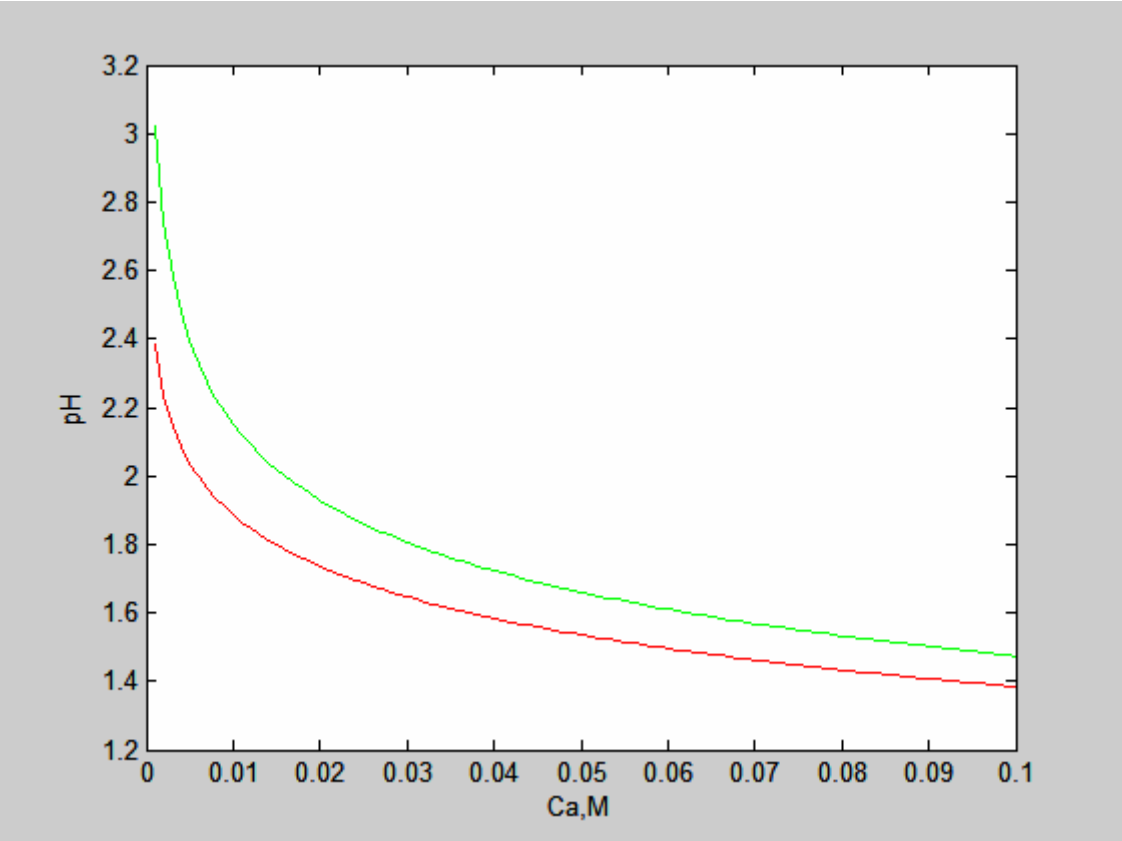
Lenguaje Matlab

>>edit acidodiprotico.m

```
%pH de disoluciones de un acido diprotico
%Datos del problema
K1=0.017;
Cao=linspace(0.001,0.1,100); %rango de concentraciones
H1=(-K1+sqrt(K1^2+4*K1*Cao))/2;
pH1=-log10(H1);
H2=sqrt(K1*Cao);
pH2=-log10(H2);
%resultados
fprintf('Ca,M Ph1 pH2n\n');
for i=1:5:length(Cao)
   fprintf('%5.3f',Cao(i));
   fprintf('%12.3f',pH1(i));
   fprintf('%12.3f\n',pH2(i));
end
plot(Cao,pH1,'g',Cao,pH2,'r');
xlabel('Ca,M')
ylabel('pH');
```

Ca,M pH1 pH2n

0.001	3.024	2.385
0.006	2.328	1.996
0.011	2.119	1.864
0.016	1.998	1.783
0.021	1.913	1.724
0.026	1.848	1.677
0.031	1.796	1.639
0.036	1.753	1.607
0.041	1.716	1.578
0.046	1.683	1.553
0.051	1.655	1.531
0.056	1.629	1.511
0.061	1.605	1.492
0.066	1.584	1.475
0.071	1.564	1.459
0.076	1.546	1.444
0.081	1.529	1.431
0.086	1.513	1.418
0.091	1.498	1.405
0.096	1.484	1.394



Al emplear la ecuación aproximada se obtiene un valor de pH más ácido que al emplear la ecuación cuadrática

Equilibrio de ácidos débiles monoprótico

Ejercicio 2- considérese el equilibrio un ácido débil en disolución de pCa= 3 a pCa=8 para el ácido yódico, HIO_3(K=0.169). Calcular los valores de pH para el rango de concentraciones mencionado.

Las reacciones que se dan son:

$$HIO_3 + H_2O \rightleftharpoons IO_3^- + H_3O^+$$

$$H_2O + H_2O \rightleftharpoons OH^- + H_3O^+$$

Las expresiones del equilibrio son:

$$K_1 = \frac{\left|IO_3^-\right|\left|H_3O^+\right|}{\left|HIO_3\right|} \quad (1)$$

$$K_W = \left|OH^-\right|\left|H_3O^+\right| \quad (2)$$

El balance de masa es:

$$C_{HIO_3} = C_a^0 = \left|IO_3^-\right| + \left|HIO_3\right| \quad (3)$$

El balance de carga es:

$$\left|H_3O^+\right| = \left|IO_3^-\right| + \left|OH^-\right| \quad (4)$$

La combinación de todas estas expresiones permita calcular el valor de $[H_3O^+]$. Al poner las concentraciones de HIO_3 y IO_3^- en función de H_3O^+ y combinar las ecuaciones se obtiene:

$$Ka = \frac{\left(\left|H_3O^+\right| - \frac{K_W}{\left|H_3O^+\right|}\right)\left|H_3O^+\right|}{C_a^0 - \left|H_3O^+\right| + \frac{K_W}{\left|H_3O^+\right|}} = \frac{\left|H_3O^+\right|^2 - K_W\left|H_3O^+\right|}{C_a^0\left|H_3O^+\right| - \left|H_3O^+\right|^2 + K_W} \quad (5)$$

Esta ecuación se transforma en una ecuación cúbica:

$$\left|H_3O^+\right|^3 + K_W\left|H_3O^+\right|^2 - \left(K_aC_a^0 + K_W\right)\left|H_3O^+\right| - K_aK_W = 0 \quad (6)$$

Sin embargo, bajo ciertas condiciones, esta ecuación se simplifica. Se supone que el equilibrio de protonación-desprotonación del agua es despreciable, pero para que esta suposición sea válida, el ácido debe de ser más fuerte que el agua protonada, por lo que la ecuación queda:

$$K_e = \frac{\left|H_3O^+\right|^2}{C_a^0 - \left|H_3O^+\right|} \quad (7)$$

Y al reorganizar queda:

$$\left|H_3O^+\right|^2 + K_a\left|H_3O^+\right| - K_aC_a^0 = 0 \quad (8)$$

Otra suposición viable es suponer que la concentración del ácido es mucho mayor que la concentración del agua protonada, algo factible para ácidos débiles con concentraciones no muy bajas. Con esta suposición, las ecuaciones quedan:

$$K_a = \frac{\left|H_3O^+\right|^2}{C_a^0} \quad (9)$$

Y al reordenar se queda:

$$\left|H_3O^+\right| = \sqrt{K_aC_a^0} \quad (10)$$

Lenguaje Matlab

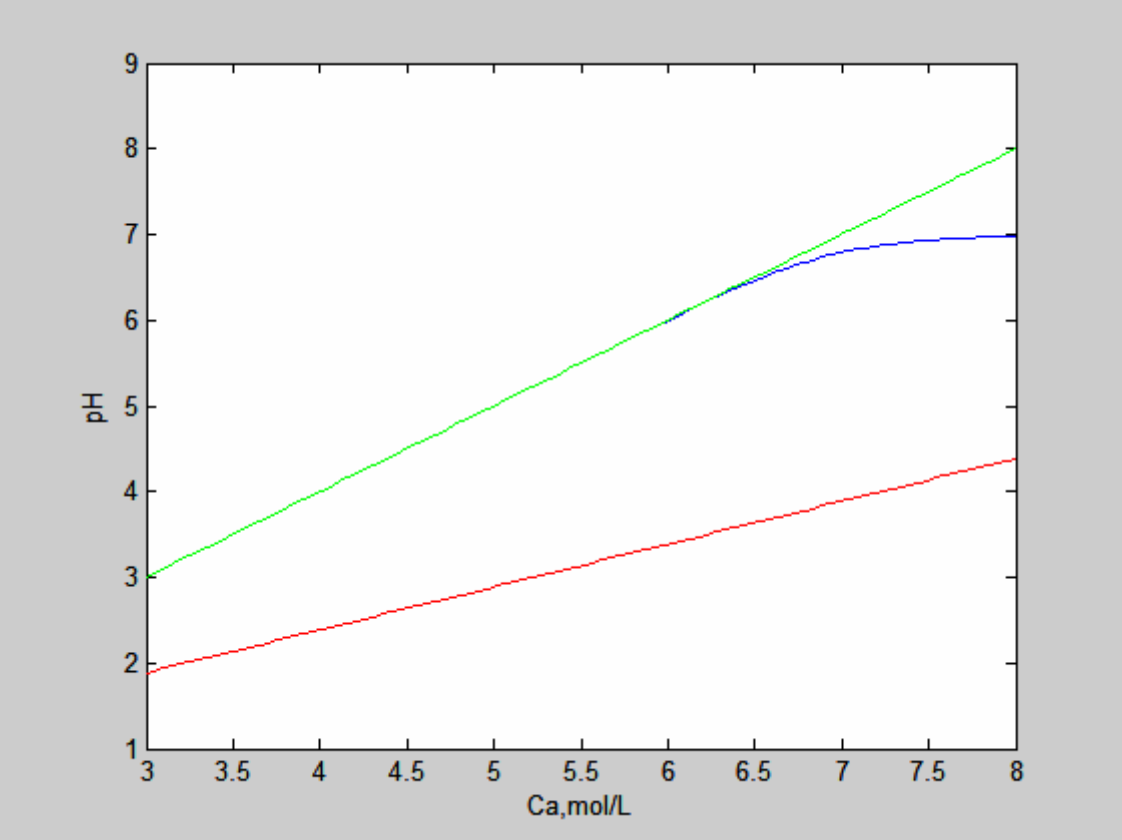
>>edit acidodebil.m

```
%equilibrio de un acido debil monoprotico
%Datos
Ka=0.169; Kw=1.0e-14;
pCa=linspace(3,8,100);
Ca=10.^(-pCa);   %Rango de concentracion del acido
%ecuacion 10
H3=sqrt(Ka*Ca)
pH3=-log10(H3);
%ecuacion 9
H2=(-Ka+sqrt(Ka^2+4*Ka*Ca))/2;
```

```
pH2=-log10(H2);
%ecuaccion 6
for i=1:length(Ca)
   Ho=H3(i);
   for j=1:200
      F=Ho^3+Ka*Ho^2-(Ka*Ca(i)+Kw)*Ho-Ka*Kw;
      D=3*Ho^2+2*Ka*Ho-(Ka*Ca(i)+Kw);
      H=Ho-F/D;
      pHo=-log10(Ho); pH=-log10(H);
      if abs((pHo-pH)/pHo)<=0.00001
         pH1(i)=pH;
         break;
      else
         Ho=H;
      end
   end
end
% Resultados
fprintf('Ca,mol/L      pH1      pH2      pH3\n\n');
for i=1:10:length(Ca)
   fprintf('%5.2e',Ca(i));
    fprintf('%12.3f',pH1(i));
    fprintf('%12.3f',pH2(i));
    fprintf('%12.3f\n',pH3(i));
 end
 plot(pCa,pH1,'b',pCa,pH2,'g',pCa,pH3,'r');
 xlabel('Ca,mol/L')
 ylabel('pH')
```

Ca,mol/L	pH1	pH2	pH3
1.00e-003	3.003	3.003	1.886
3.13e-004	3.506	3.506	2.139

9.77e-005	4.010	4.010	2.391
3.05e-005	4.515	4.515	2.644
9.55e-006	5.020	5.020	2.896
2.98e-006	5.525	5.525	3.149
9.33e-007	6.025	6.030	3.401
2.92e-007	6.491	6.535	3.654
9.11e-008	6.808	7.040	3.906
2.85e-008	6.938	7.545	4.159



En la figura se observa que al utilizar la ecuación 10 (línea roja) se comete un error mucho mayor al calcular los pH que al utilizar las otras dos ecuaciones. Las otras dos aproximaciones dan resultados idénticos de valores de pH, pero sin embargo a concentraciones diluidas, la representación de la ecuación 9 se desvia, debido a que la C_a^0 se asemeja a la concentración de H_3O^+ (línea azul). La línea verde representa la ecuación 6.

Curva de valoración de un ácido fuerte con una base fuerte.

Ejercicio 3- Considérese la valoración de un volumen Ve de HCl de concentración inicial C_a^0 con una dilsolución de NaOH de concentración inicial C_b^0. El volumen de equivalencia Ve se alcanza cuando la concentración de NaOH se iguala a la concentración de HCl.

$$V_e = \frac{V_a C_a^0}{C_b^0} \quad (1)$$

Antes de iniciar la valoración, cuando, cuando V_b=0, se tiene que $\left|H_3O^+\right| = C_a^0$. Cuando el proceso comienza y hasta antes del punto de equivalencia, la concentración de H_3O^+ corresponde a la concentración de ácido sobrante en la reacción de neutralización (se presume que la concentración de agua protonada es despreciable).

$$\left|H_3O^+\right| = \frac{V_e C_e^0 - V_b C_b^0}{V_e + V_b} \quad (2)$$

En el punto de equivalencia, $\left|H_3O^+\right| = \left|OH^-\right| = 1{\cdot}10^{-7} M$, después del punto de equivalencia, la concentración de reactivo en exceso se calcula mediante la ecuación:

$$\left|OH^-\right| = \frac{V_b C_b^0 - V_e C_e^0}{V_a + V_b} \quad \text{y} \quad \left|H_3O^+\right| = \frac{K_w}{\left|OH^-\right|} \quad (3)$$

Lenguaje Matlab

>> edit valoración.m

```
%Valoracion de un acido fuerte con una base fuerte
%Datos
Cao=0.12; Va=25;
Cbo=0.10;
```

```
Kw=1e-14;
Ve=Cao*Va/Cbo;
Vb=linspace(0,1.5*Ve,100);%Rango de volumende base, Vb
for i=1:length(Vb)
   if Vb(i)<Ve
      Ca=(Cao*Va-Cbo*Vb(i))/(Va+Vb(i));
      H=Ca;
   end
   if Vb(i)==Ve
      H=1e-07;
   end
   if Vb(i)>Ve
      Cb=Cbo*(Vb(i)-Ve)/(Va+Vb(i));
      OH=Cb; H=Kw/OH;

   end
    pH(i)=-log(H)
end
%Resultados
fprintf('Vb,ml      pH\n\n');
for i=1:10:length(Vb)
   fprintf('%5.2f',Vb(i));
   fprintf('%12.2f\n',pH(i));
end
plot(Vb,pH)
title('Valoracion de un acido fuerte con una base fuerte')
xlabel('Vb,ml')
ylabel('pH')
```

Vb,ml	pH
0.00	2.12
4.55	2.45
9.09	2.79

13.64 3.16

18.18 3.60

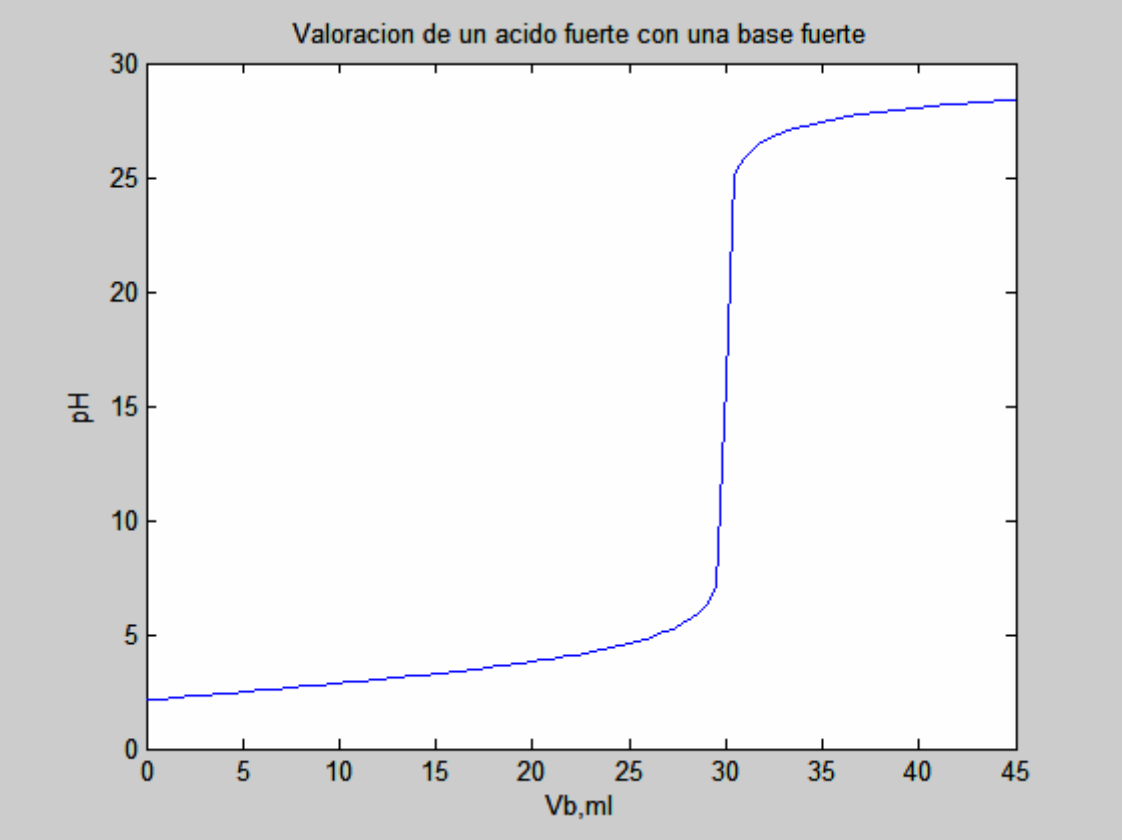
22.73 4.18

27.27 5.26

31.82 26.49

36.36 27.67

40.91 28.13



Gráficamente obtenemos la curva de valoración típica de un ácido fuerte con una base fuerte, sin embargo, ni gráficamente ni en la tabla de valores, nose puede decir exactamente cuál es el V_e, por lo que habría que tomar más puntos para que ver el salto de pH.

>>edit valoración.m

```
%Valoracion de un acido fuerte con una base fuerte
```

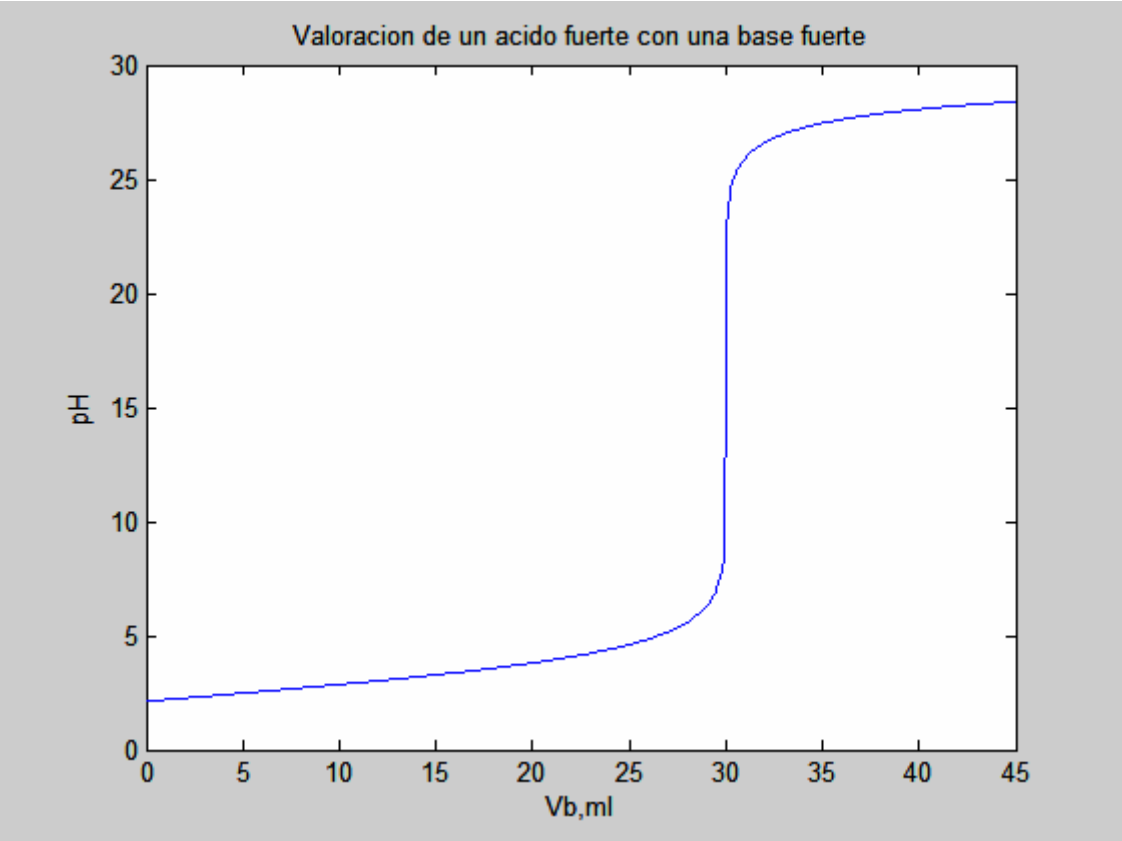
```
%Datos
Cao=0.12; Va=25;
Cbo=0.10;
Kw=1e-14;
Ve=Cao*Va/Cbo;
Vb=linspace(0,1.5*Ve,1000);%Rango de volumende base, Vb
for i=1:length(Vb)
   if Vb(i)<Ve
      Ca=(Cao*Va-Cbo*Vb(i))/(Va+Vb(i));
      H=Ca;
   end
   if Vb(i)==Ve
      H=1e-07;
   end
   if Vb(i)>Ve
      Cb=Cbo*(Vb(i)-Ve)/(Va+Vb(i));
      OH=Cb; H=Kw/OH;

   end
    pH(i)=-log(H)
end
%Resultados
fprintf('Vb,ml      pH\n\n');
for i=1:10:length(Vb)
   fprintf('%5.2f',Vb(i));
   fprintf('%12.2f\n',pH(i));
end
plot(Vb,pH)
title('Valoracion de un acido fuerte con una base fuerte')
xlabel('Vb,ml')
ylabel('pH')
```

```
Vb,ml       pH
```

0.00	2.12
0.45	2.15
0.90	2.19
1.35	2.22
1.80	2.25
2.25	2.28
2.70	2.32
3.15	2.35
3.60	2.38
4.05	2.42
4.50	2.45
4.95	2.48
5.41	2.51
5.86	2.55
6.31	2.58
6.76	2.61
7.21	2.65
7.66	2.68
8.11	2.72
8.56	2.75
9.01	2.79
9.46	2.82
9.91	2.86
10.36	2.89
10.81	2.93
11.26	2.96
11.71	3.00
12.16	3.04
12.61	3.07
13.06	3.11
13.51	3.15
13.96	3.19
14.41	3.23
14.86	3.27

15.32	3.31
15.77	3.35
16.22	3.40
16.67	3.44
17.12	3.49
17.57	3.53
18.02	3.58
18.47	3.63
18.92	3.68
19.37	3.73
19.82	3.78
20.27	3.84
20.72	3.90
21.17	3.96
21.62	4.02
22.07	4.08
22.52	4.15
22.97	4.22
23.42	4.30
23.87	4.38
24.32	4.46
24.77	4.56
25.23	4.66
25.68	4.76
26.13	4.88
26.58	5.02
27.03	5.16
27.48	5.34
27.93	5.54
28.38	5.80
28.83	6.13
29.28	6.62
29.73	7.61
30.18	24.21

30.63	25.45
31.08	25.98
31.53	26.33
31.98	26.57
32.43	26.77
32.88	26.93
33.33	27.07
33.78	27.19
34.23	27.30
34.68	27.39
35.14	27.47
35.59	27.55
36.04	27.62
36.49	27.68
36.94	27.74
37.39	27.80
37.84	27.85
38.29	27.90
38.74	27.95
39.19	27.99
39.64	28.03
40.09	28.07
40.54	28.11
40.99	28.14
41.44	28.17
41.89	28.21
42.34	28.24
42.79	28.27
43.24	28.29
43.69	28.32
44.14	28.35
44.59	28.37



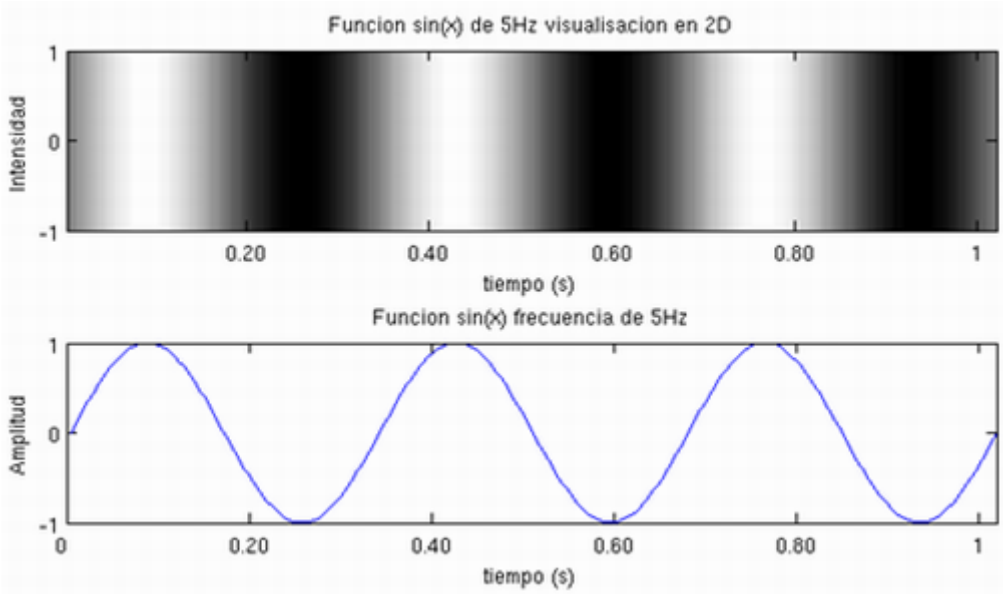
El volumen de equivalencia se encuentra entre 29.73-30.18 ml, y el valor de pH salta desde 7.61 hasta 24.21.

Visualización de Ondas en 2D y el fenómeno de interferencia

Visualización de ondas

Una onda es una forma como se propaga la energía en el espacio, por ejemplo golpear con un martillo una tabla de madera genera una vibración. Un foco es una fuente generadora de calor y energía luminosa.

La función *sin(x)* puede representarse en 2D como lo muestra la siguiente imagen, observa como la intensidad varia periódicamente (forma un patrón), El color negro representa zonas con un valor de amplitud igual a -1 y las zonas de color blanco representado con el valor de amplitud igual a 1.

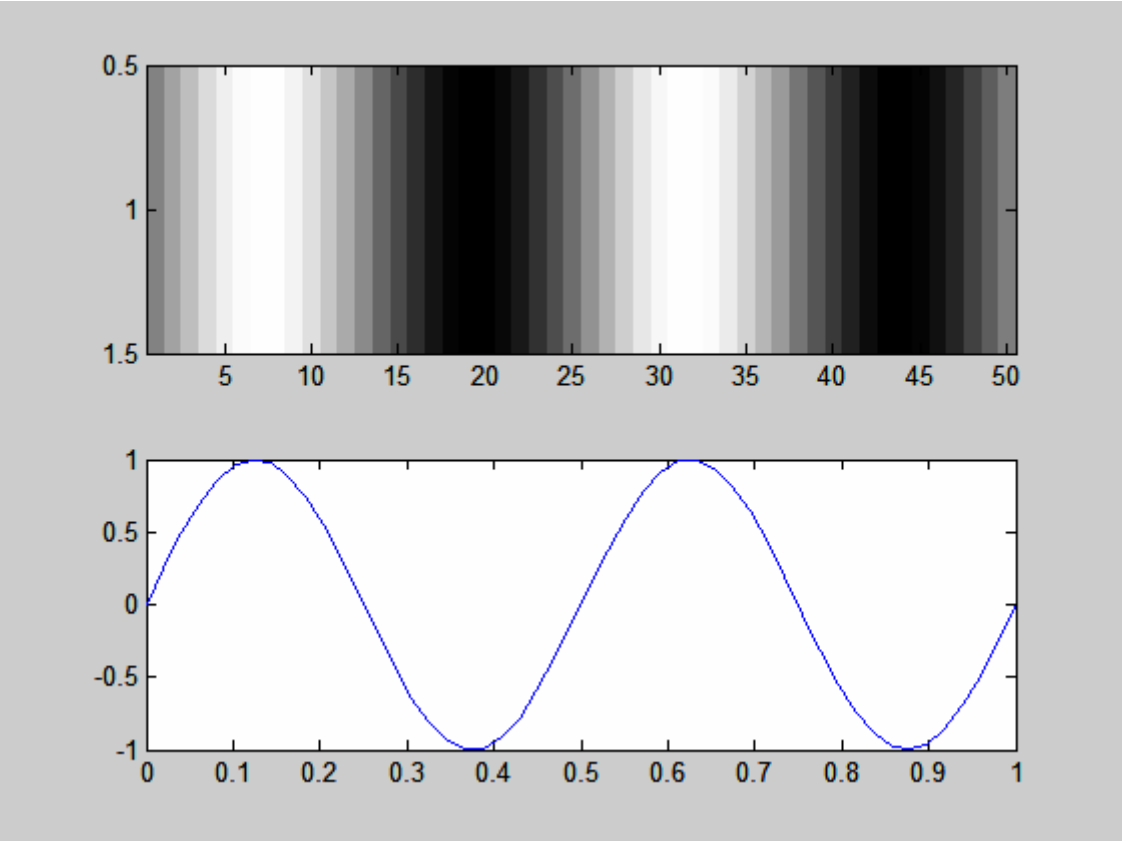


Lenguaje Matlab

>>edit onda.m

```
function x=onda(n,f)
w=2*pi*f;%donde f es la frecuencia
t=0:1/(n-1):1;
x=sin(w*t);%n es el numero de segmentaciones
subplot(2,1,1);image(x,'CDataMapping','scale')
colormap(gray)
subplot(2,1,2);plot(t,x)
```

>>onda(50,5000)



Interferencia destructiva y constructiva

En la mecánica ondulatoria la interferencia es lo que resulta de la superposición de dos o mas ondas, resultando en la creación de un nuevo patrón de ondas.

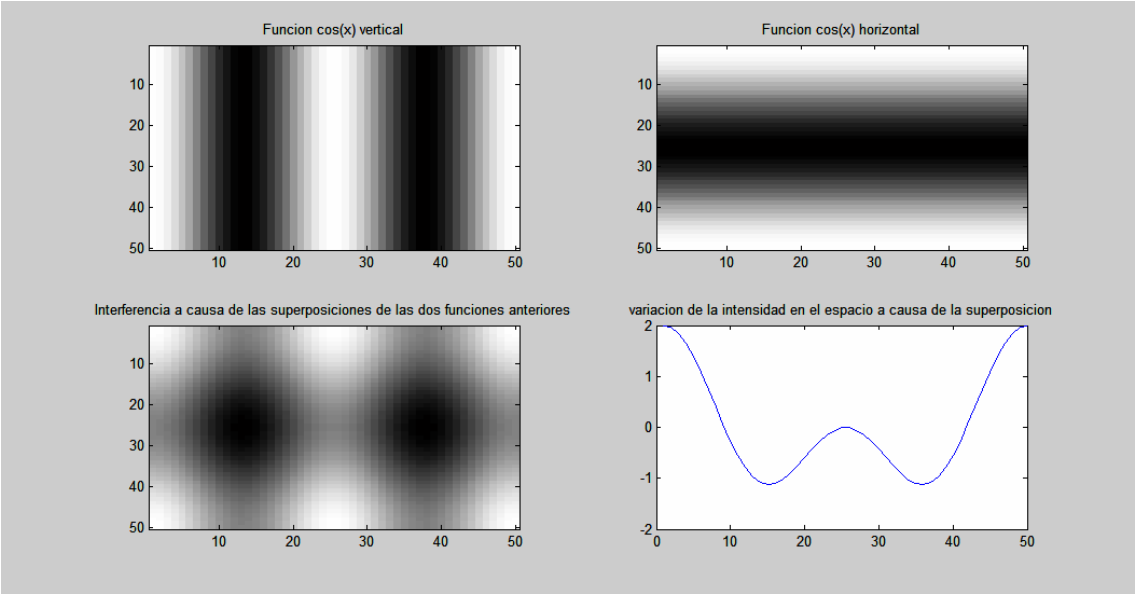
Aunque la acepción mas usual para interferencia se refiere a la superposición de dos o mas ondas de frecuencia idéntica o similares principio de superposición de ondas establece que la magnitud del desplazamiento ondulatorio en cualquier punto del medio es igual a la suma de los desplazamientos en ese mismo punto de todas las ondas presentes. Esto es consecuencia de que la Ecuación de onda es lineal, y por tanto si existen dos o mas soluciones, cualquier combinación lineal de ellas será también solución.

Lenguaje Matlab

>>edit interferencia.m

```
function x=interferencia(n,f,f1)
w=2*pi*f;%donde f es la frecuencia horizontal
d=2*pi*f1;%donde f es la frecuencia vertical
p=255*(0:1/(n-1):1);%n es el numero de segmentaciones
for i=1:n%se genera una matriz imagen de nxn pixeles
x(i,:)=cos(w*p);%la onda horizontal en 2d
y(i,:)=cos(d*p);%la onda vertical en 2d
end
subplot(2,2,1);image(x,'CDataMapping','scaled'),title('Funcion cos(x) vertical')
subplot(2,2,2);image(y','CDataMapping','scaled'),title('Funcion cos(x) horizontal')
subplot(2,2,3);image(x+y','CDataMapping','scaled'),title('Interferencia a causa de las superposiciones de las dos funciones anteriores')
colormap(gray)
subplot(2,2,4);plot(x(1,:)+y(1,:)),title('variacion de la intensidad en el espacio a causa de la superposicion')
function x=interferencia(n,f,f1)
w=2*pi*f;%donde f es la frecuencia horizontal
d=2*pi*f1;%donde f es la frecuencia vertical
p=255*(0:1/(n-1):1);%n es el numero de segmentaciones
for i=1:n%se genera una matriz imagen de nxn pixeles
x(i,:)=cos(w*p);%la onda horizontal en 2d
y(i,:)=cos(d*p);%la onda vertical en 2d
end
subplot(2,2,1);image(x,'CDataMapping','scaled')
subplot(2,2,2);image(y','CDataMapping','scaled')
subplot(2,2,3);image(x+y','CDataMapping','scaled')
colormap(gray)
subplot(2,2,4);plot(x(1,:)+y(1,:))
```

```
>> interferencia(50,500,250)
```



Podemos observar cada una de las ondas, tanto la horizontal como la vertical; la interferencia de la superposición de ambas ondas, y la representación de la intensidad de la superposición de las ondas.

Objetivo

El objetivo de este manual no es mas que enseñar la utilización del lenguaje Matlab desde un punto de vista práctico, y las aplicaciones que puede tener. Como primer punto, se comienzan a editar programas con aplicaciones meramente matemáticas, ya que serán una posterior herramienta para otras muchas aplicaciones. Posteriormente, se comienza a hacer ejercicios que son de un interés medio, para finalmente culminar el aprendizaje con la aplicación a la modelización matemática de sistemas físicos, químicos…

La elección de los ejercicios finales que se han propuesto, desde el punto de vista químico, son de gran interés; con la finalidad de aplicar la modelización matemática a sistemas químicos como es una cinética, equilibrio ácido-base, una valoración de un ácido, o la representación de ondas y su superposición.

Bibliografía

- Apuntes de Matlab. Cándido Piñeiro. Universidad de Huelva
- Fundamentos de informática y programación científica .Resolución en C y Matlab
- Análisis de sistemas dinámicos lineales. Oscar G. Duarte V. Profesor asociado del Departamento de Ingeniería Eléctrica y Electrónica
- Métodos numéricos en química con Matlab .Autor Rubén Darío Osorio Giraldo
- http://www.monografias.com/
- Apuntes de Biofísica. Licenciatura de Bioquímica. Universidad de Salamanca