Graficas en 2 dimensiones

Control de los ejes

Por defecto, MATLAB ajusta la escala de cada uno de los ejes de modo que varíe entre el mínimo y el máximo valor de los vectores a representar.

MATLAB tiene la función “axis” para controlar la escala y apariencia de los ejes. Así,

axis(‘auto’): regresa todos los valores al defecto

axis([xmin, xmax, ymin, ymax]): establece la escala para los ejes x e y para la figura activada

v = axis: devuelve en el vector v los valores xmin, xmax, ymin, ymax de los ejes x e y

axis(axis): mantiene los ejes con sus valores actuales, en el caso de añadir nuevas gráficas a la figura activa, usando “hold on”

axis(‘equal’): escalado es igual en ambos ejes

axis(‘square’): la ventana de la figura será cuadrada

axis(‘normal’): elimina las restricciones introducidas por “equal” y “square”

axis(‘off’) y axis(‘on’): elimina y restituye los ejes

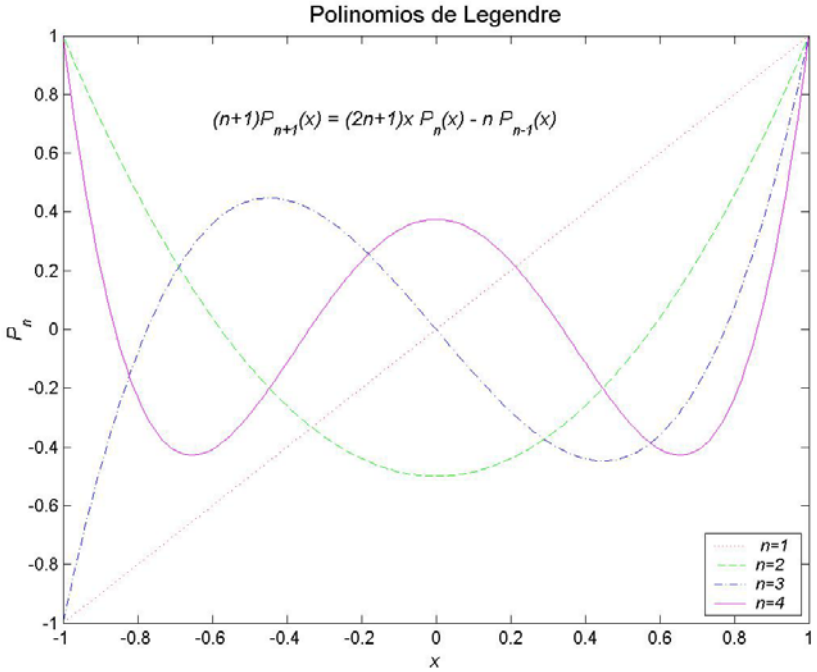
Graficas en 2 dimensiones

Ejemplo: polinomios de Legendre $P_n(x)$

$$(n+1)P_{n+1}(x) = (2n+1)xP_n(x) - nP_{n-1}(x)$$

con $P_0 = 1$ y $P_1 = x$

```
>> x = -1 : .01 : 1;
>> p1 = x;
>> p2 = (3/2)*x.^2 - 1/2;
>> p3 = (5/2)*x.^3 - (3/2)*x;
>> p4 = (35/8)*x.^4 - (15/4)*x.^2 + 3/8;
>> plot(x,p1,'r:', x,p2,'g--', x,p3,'b-.', x,p4,'m-')
>> legend('\it n=1','n=2','n=3','n=4',4);
>> xlabel('x', 'FontSize',12, 'FontAngle','italic')
>> ylabel('P_n', 'FontSize',12, 'FontAngle','italic')
>> title('Polinomios de Legendre', 'FontSize',14);
>> text(-.6,.7,'(n+1)P_{n+1}(x) = (2n+1)x P_n(x) - n P_{n-1}(x)', ...
     'FontSize',12,'FontAngle','italic')
```



graf_Legendre.m

-1 derecha del gráfico
0 escogencia automática
1 esquina superior derecha (defecto)
2 esquina superior izquierda
3 esquina inferior izquierda
4 esquina inferior derecha

Legendre.m

Graficas en 2 dimensiones

Función “ezplot”

ezplot(f) grafica la función $y = f(x)$ sobre el domino $-2\pi \leq x \leq 2\pi$

ezplot(f, [a, b]) grafica la función $y = f(x)$ sobre el domino $a \leq x \leq b$

Si la función está dada en forma inplícita, es decir, $f = f(x,y) = 0$

ezplot(f) grafica la función f sobre el domino $-2\pi \leq x \leq 2\pi$, $-2\pi \leq y \leq 2\pi$

ezplot(f, [xmin, xmax, ymin, ymax]) grafica la función f para

$xmin \leq x \leq xmax$, $ymin \leq y \leq ymax$

ezplot(f, [a, b]) grafica la función f sobre el domino $a \leq x \leq b$, $a \leq y \leq b$

Si la función está dada en forma paramétrica, es decir, $x = x(t)$, $y = y(t)$

ezplot(x, y) grafica la función para $0 \leq t \leq 2\pi$

ezplot(x, y, [tmin, tmax]) grafica la función para $tmin \leq t \leq tmax$

Función “inline”: construye una función a partir de una cadena de caracteres,

Por ejemplo f = inline('cos(x)+2*sin(x) ') o f = @(x) cos(x)+2*sin(x)

y puede ser usado con “ezplot”

Graficas en 2 dimensiones

Función “ezplot”

Ejemplos:

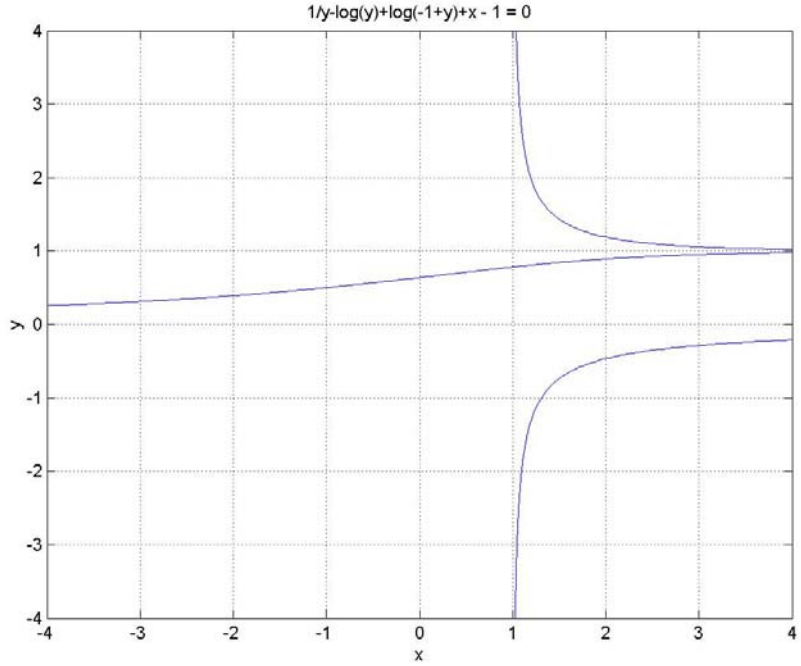
```
>> f = inline('1/y-log(y)+log(-1+y)+x - 1')
```

- f = '1/y-log(y)+log(-1+y)+x - 1'

```
>> ezplot( f, [-4, 4])
```

grafica en [-4, 4]× [-4, 4] la función implícita

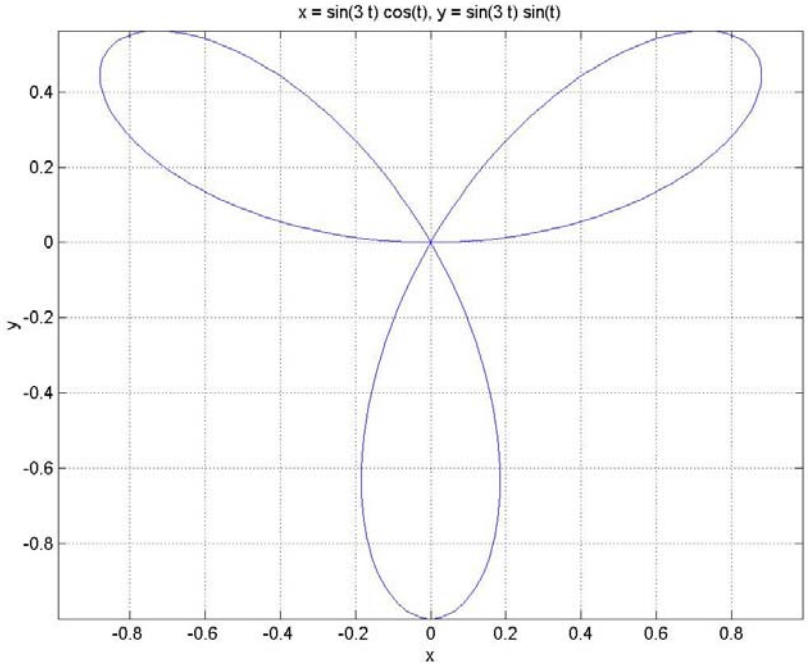
$$\frac{1}{y} - \log(y) + \log(-1+y) + x - 1 = 0$$




```
>> x = inline('sin(3*t)*cos(t) ');
>> y = inline('sin(3*t)*sin(t) ');
>> ezplot(x, y, [0,pi])
```

grafica para $0 \le t \le \pi$ la función paramétrica

$$x(t) = \sin(3t)\cos(t), \quad y(t) = \sin(3t)\sin(t)$$



Graficas en 2 dimensiones

Función “ginput”

Permite la entrada de puntos usando el ratón

>> [xv, yv] = ginput(n)

Captura n puntos del gráfico y devuelve las coordenadas x e y de los puntos en los vectores xv e yv; xv e yv son de longitud n.

Para introducir la información se usa los botones del ratón o una tecla del teclado.

Para capturar un número ilimitado de puntos hasta que se presione la tecla “return” se tiene la instrucción:

>> [xv, yv] = ginput

Graficas en 2 dimensiones

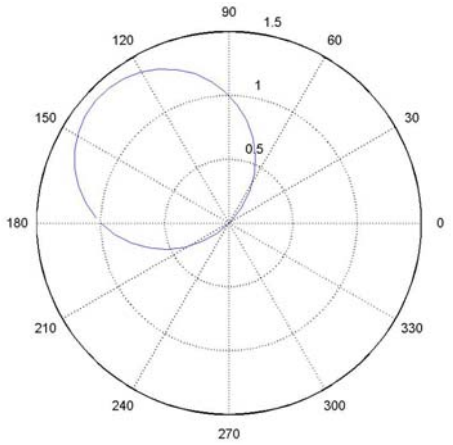
Otras funciones disponibles en MATLAB para gráficos especiales

polar(theta, rho, s)	Grafica los vectores theta y rho en coordenadas polares, usando el tipo de línea especificado en s
bar(x, y, s) o barh(x, y, s)	Grafica las columnas de la matriz y de dimensión m×n como m grupos de n barras verticales cada uno. El vector x (opcional) debe ser monótono creciente o decreciente; si x no se especifica se usa como defecto es x = 1 : m. El parámetro s puede ser 'grouped' o 'stacked'. La función barh dibuja barras horizontales.
hist(y) o hist(y, m)	Agrupa los elementos de y en 10 subgrupos igualmente espaciados, retorna en n el número de elementos en cada subgrupo y el histograma. Si m está dado, corresponde al número de subgrupos deseados.
pie(x, y, etiquetas)	Grafica las entradas del vector x usando un diagrama de torta. Los valores de x se normalizan via x/suma(x) para determinar el área de cada pedazo de la torta. El vector y (opcional) indica que pedazo de la torta se separa de esta (entrada diferente de cero). El arreglo de cadena de caracteres "etiquetas" asigna nombre a cada pedazo de la torta.
scatter(x, y, s ,c, 'filled')	Grafica círculos en las localizaciones especificadas por los vectores x e y (vectores de igual longitud). El área de cada círculo está determinada por los valores de las entradas del vector s y el color por el vector c. Los vectores s y c son opcionales. 'filled' indica que cada círculo se rellena (opcional).

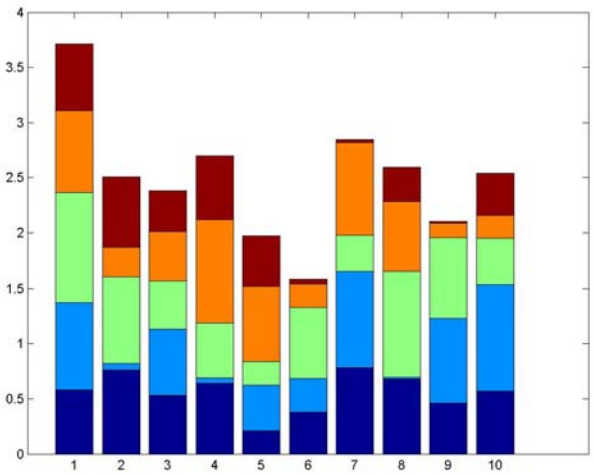
Graficas en 2 dimensiones

Otras funciones disponibles en MATLAB para gráficos especiales (cont.)

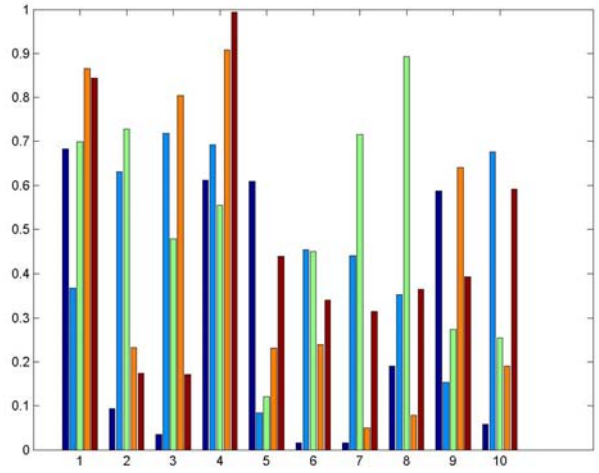
```
>> theta = 0: 0.1: pi;
>> r = sin(theta) - cos(theta);
>> polar(theta, r)
```




```
>> bar(rand(10,5),'stacked')
```



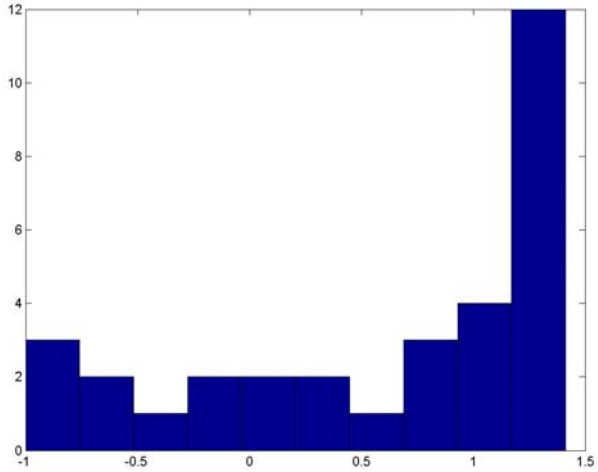

```
>> bar(rand(10,5),‘grouped')
```



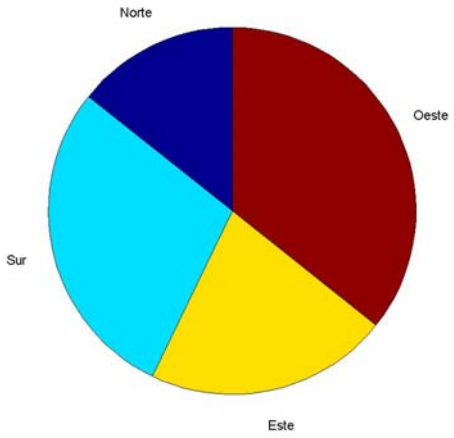
Graficas en 2 dimensiones

Otras funciones disponibles en MATLAB para gráficos especiales (cont.)

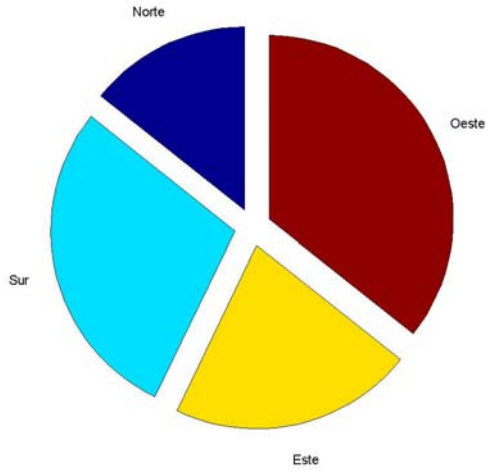
```
>> x = 0: 0.1: pi;
>> y = sin(x) - cos(x);
>> n = hist(y)
→   3  2  1  2  2  2  1  3  4  12
```




```
>> pie([2 4 3 5], …
{'Norte','Sur','Este','Oeste'})
```



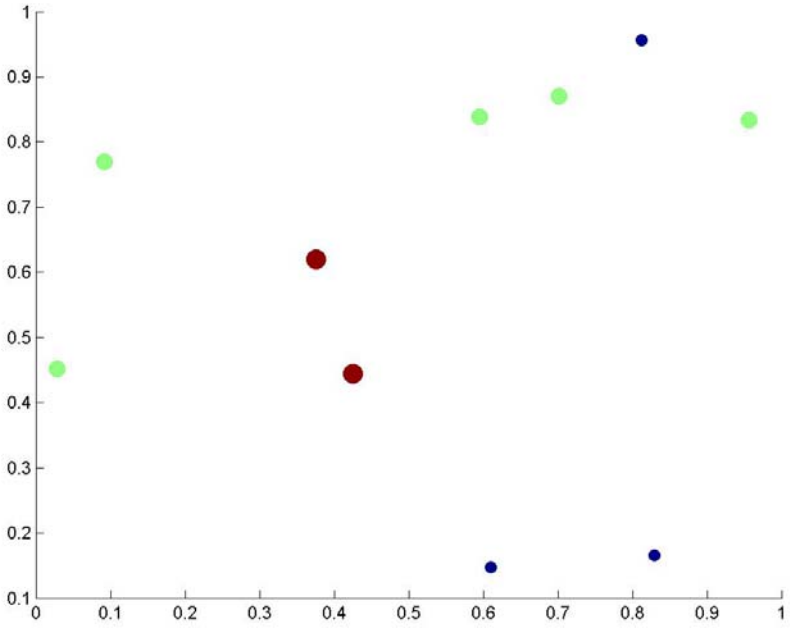

```
>> pie([2 4 3 5], [1 1 1 1], …
{'Norte','Sur','Este','Oeste'})
```



Graficas en 2 dimensiones

Otras funciones disponibles en MATLAB para gráficos especiales (cont.)

```
>> x = rand(1,10); y = rand(1,10);
>> c = [ 1  2  2  2  1  1  2  2  3  3 ]; s = c*50;
>> scatter(x, y, s, c, 'filled')
```



x = [0.8295 0.9561 0.5955 0.0287 0.8121 0.6101 0.7015 0.0922 0.4249 0.3756]
y = [0.1662 0.8332 0.8386 0.4516 0.9566 0.1472 0.8699 0.7694 0.4442 0.6206]

Graficas en 2 dimensiones

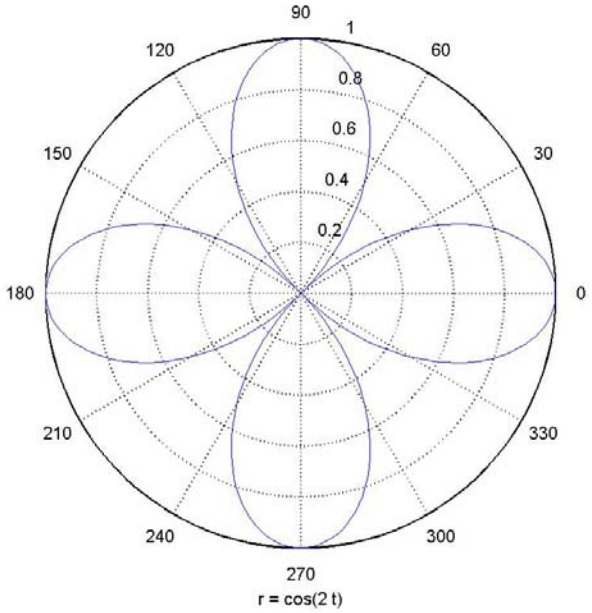
Función “ezpolar”

Grafica funciones en coordenadas polares.

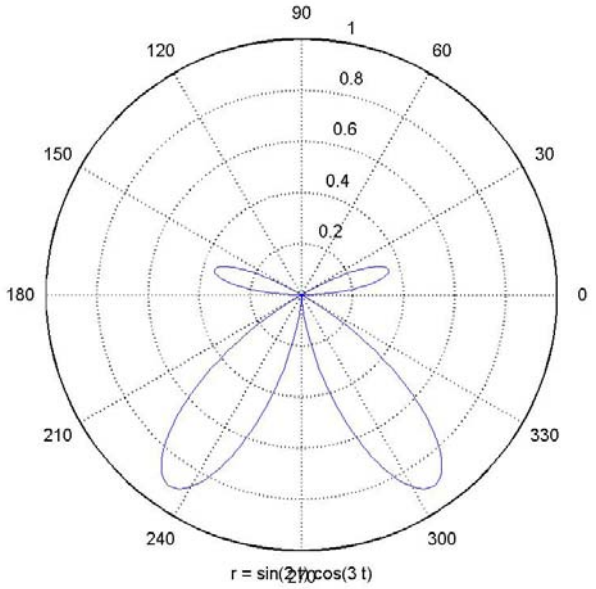
ezpolar(f) grafica f en el intervalo [0, 2π].

ezpolar(f,[a,b]) grafica f en el intervalo [a, b].

>> ezpolar('cos(2*t)')



>> ezpolar('sin(2*t)*cos(3*t)', [0,pi])



progreso_polar.m

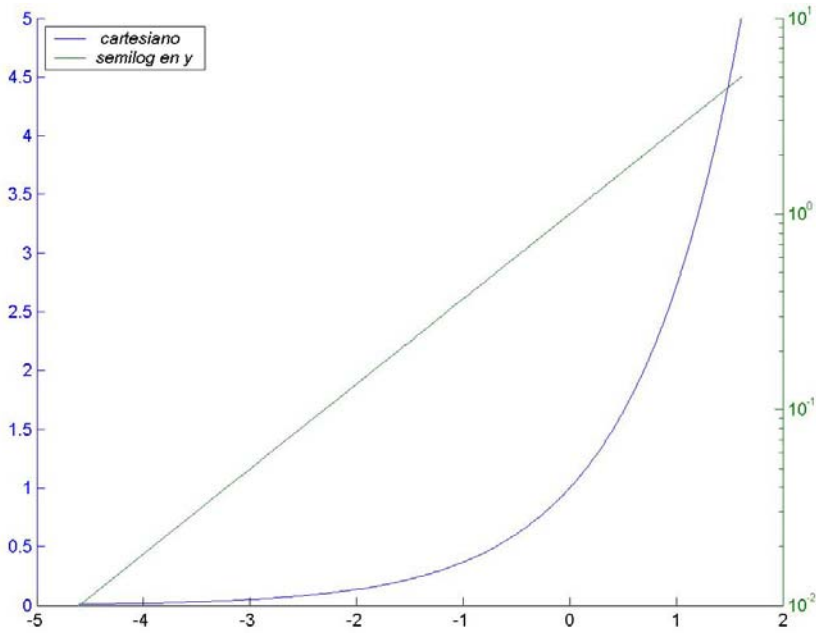
Graficas en 2 dimensiones

Función “plotyy”

plotyy(x1,y1,x2,y2,fun1,fun2) grafica los puntos (x1,y1) contra el eje y de la izquierda y (x2,y2) contra el eje y de la derecha, usando las opciones especificadas por fun1 y fun2. Las opciones pueden ser:

@plot, @semilogx, @semilogy, @loglog, @stem, @stairs

```
>> y = 0.01 : 0.01 : 5;
>> x = log(y);
>> plotyy(x, y, x, y, @plot, @semilogy);
>> [ax, h1, h2] = ...
    plotyy(x, y, x, y, @plot, @semilogy);
>> h = [h1, h2];
>> legend(h, '\it cartesiano', …
      'semilog en y', 2);
```



h1 y h2 son las referencias a las curvas, las cuales son usadas en “legend”

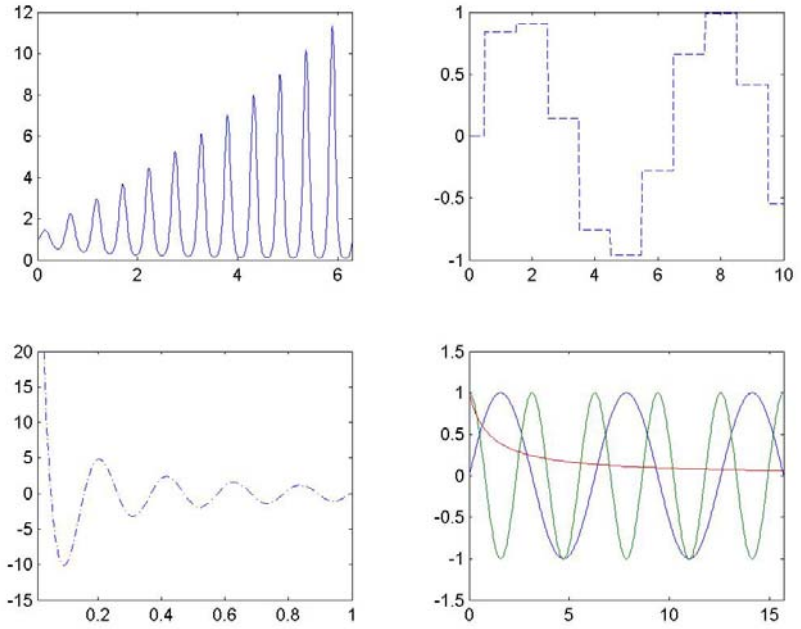
Graficas en 2 dimensiones

Función “subplot”

Una ventana gráfica se puede dividir en m particiones horizontales y n verticales, con el objeto de representar múltiples gráficas en ella.

Así, subplot(m,n,i): divide la ventana gráfica en m filas y n columnas, siendo i la subdivisión activa, para $1 \leq i \leq mn$.

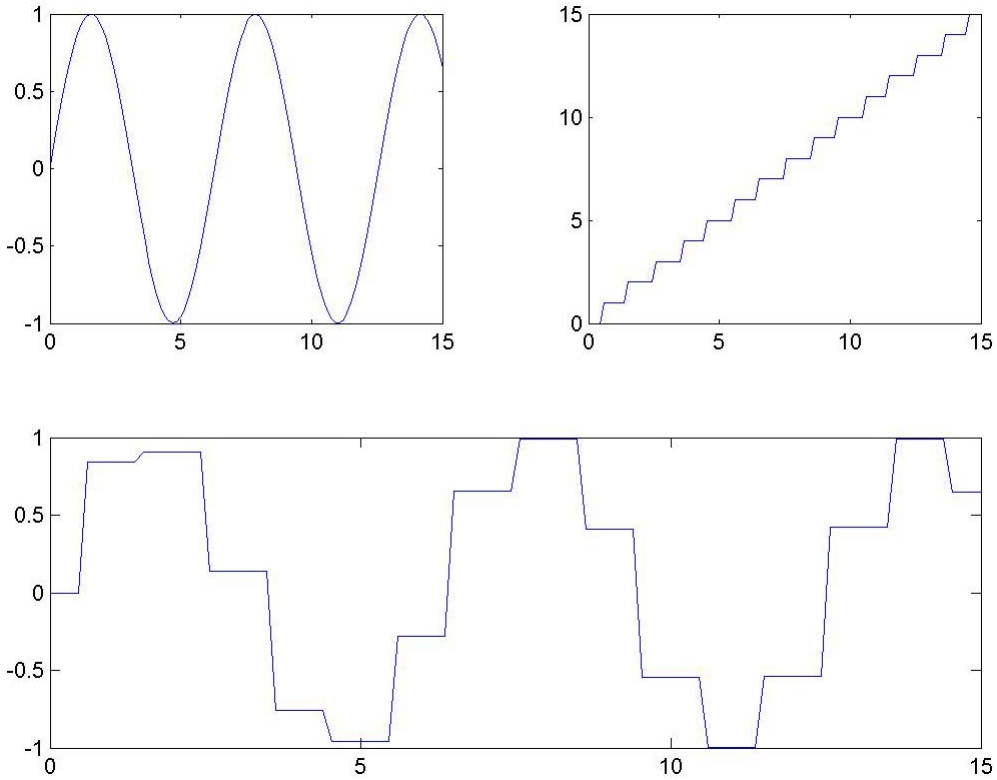
```
>> subplot(2, 2, 1), …
fplot('exp(sqrt(x)*sin(12*x))', [0, 2*pi])
>> subplot(2, 2, 2), …
fplot('sin(round(x))', [0, 10], '--')
>> subplot(2, 2, 3), …
fplot('cos(30*x)/x',[0.01  1  -15  20],'-.')
>> subplot(2, 2, 4), …
fplot('[sin(x),cos(2*x),1/(1+x)]', [0  5*pi
-1.5  1.5])
```



Graficas en 2 dimensiones

Función “subplot” (cont.)

```
>> x = linspace(0,15,100);
>> subplot(2,2,1), plot(x,sin(x))
>> subplot(2,2,2), plot(x,round(x))
>> subplot(2,2,3:4), plot(x,sin(round(x)))
```



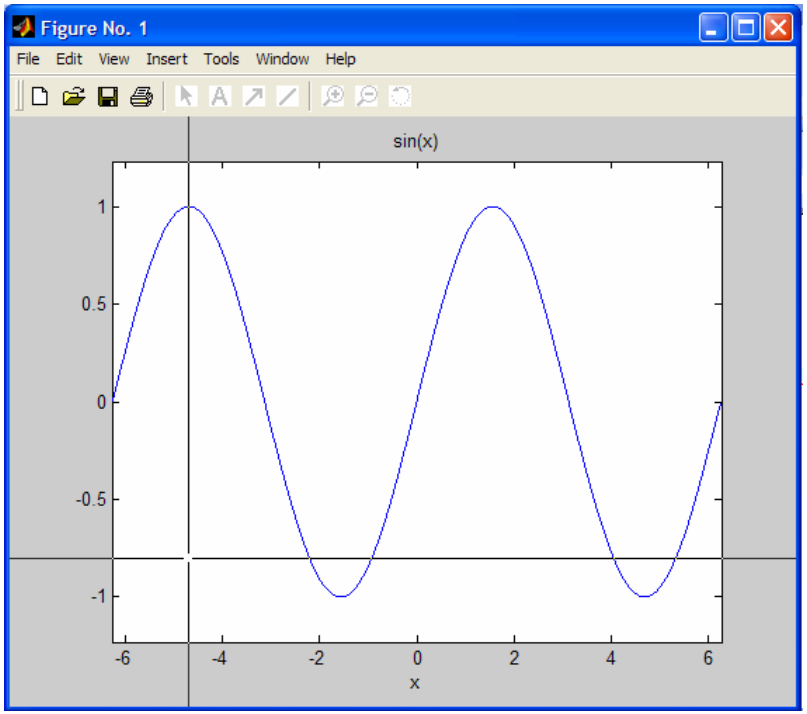
Graficas en 2 dimensiones

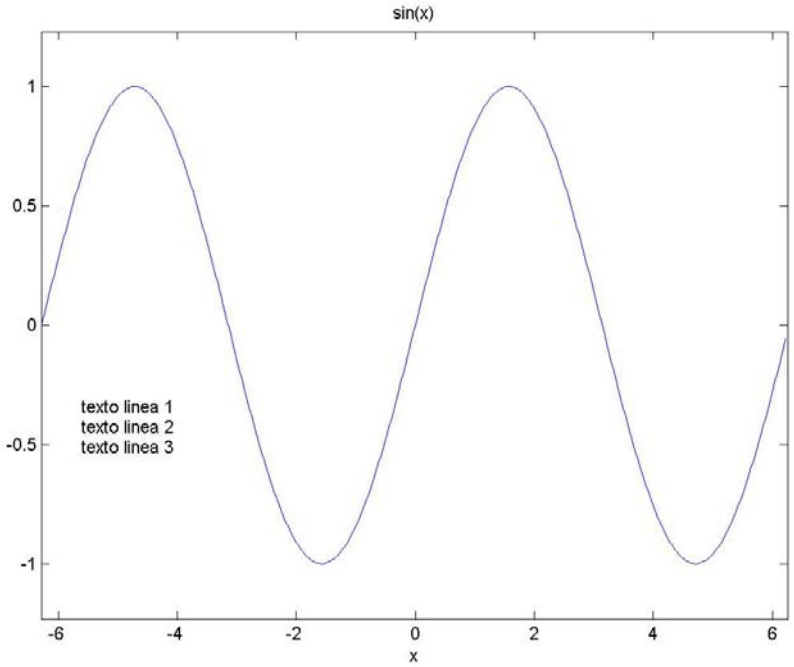
Función “gtext”

También puedo insertarle texto al grafico indicando con el " mouse" el lugar donde deseo colocar el texto.

Con la ayuda del botón izquierdo del “mouse” podemos insertar un texto

```
>> ezplot(‘sin(x)’);
>> gtext({‘texto linea 1‘, ‘texto linea 2’, ‘texto linea 3’});
```





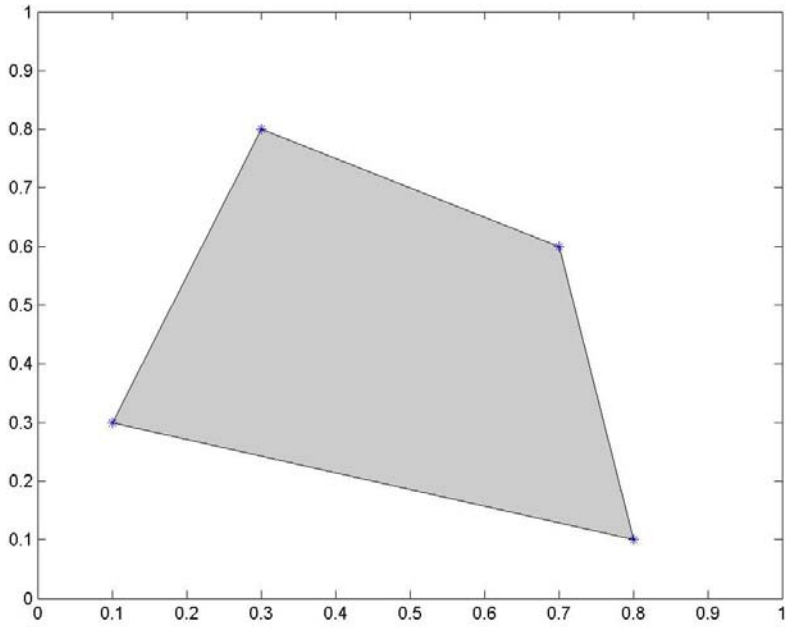
Graficas en 2 dimensiones

Función “fill”

La función “fill trabaja de manera similar que “plot”. Así fill(x,y, [r, g, b]) dibuja un polígono cuyos vértices son los puntos x(i), y(i) y rellena la región interna del color indicado. Los elementos r, g, b que deben estar en [0,1], determina los niveles de rojo, verde y azul respectivamente cuando se rellena el polígono. Los puntos se toman en orden, y el último se une al primer vértice.

Así, fill(x,y,[0,1,0]) rellena el polígono con verde puro y fill(x,y,[1,0,1]) lo rellena con igual cantidad de rojo y azul. El color negro se consigue con [0,0,0] y el blanco con [1,1,1].

```
>> x = [0.1, 0.3, 0.7, 0.8];
>> y = [0.3, 0.8, 0.6, 0.1];
>> plot(x, y, '*');
>> axis([0,1,0,1]);
>> hold on
>> fill(x, y, [0.8, 0.8, 0.8]);
```



Métodos numéricos en MATLAB

Polinomios

Los polinomios en MATLAB se representan con vectores filas, así

$$p(x) = x^2 - x - 1 \quad \leftrightarrow \quad p = [1 \quad -1 \quad -1]$$

Evaluación de polinomios: esta se lleva a cabo siguiendo el método de Horner la cual corresponde a la representación anidada siguiente

$$p(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_{n-1} x^2 + a_n x + a_0$$
$$= (((a_1 x + a_2)x + a_3)x + \cdots + a_n)x + a_{n+1}$$

La función correspondiente a este método se denomina “polyval”. Así, y = polyval(p,x) corresponde a evaluar el polinomio p en x.

La variable x puede ser un arreglo (vector o matriz), y en este caso evaluar el polinomio en x corresponde a evaluarlo en cada una de sus entradas.

>> y = polyval([1, -1, -1], [0, 1]) $\rightarrow$ y = [-1, -1]

Métodos numéricos en MATLAB

Polinomios

Las raíces (o ceros) del polinomio p, se calculan con la función "roots"; por supuesto algunas de las raíces pueden ser complejas, a pesar de que los coeficientes de p sean reales.

>> p = [1, -1, -1]; r = roots(p) $\rightarrow$ r = -0.6180 1.6180

También se tiene la función "poly(r)" que construye el polinomio p cuyas raíces son los elementos del vector r. Aquí, siempre se lleva a cabo la normalización de los coeficientes tal que $a_1 = 1$.

>> r = [-0.6180, 1.6180]; p = poly(r) $\rightarrow$ p = 1 -1 -1

La función "poly" puede ser aplicada a una matriz cuadrada A, en cuyo caso retorna el polinomio característico p asociado a la matriz A, es decir el polinomio p(x) = det(I x – A).

>> A = [0 1; 1 1]; p = poly(A) $\rightarrow$ p = 1.0000 -1.0000 -1.0000

Métodos numéricos en MATLAB

Polinomios

Evaluar un polinomio p en un argumento matricial A, donde A es una matriz cuadrada, se puede llevar a cabo con la función “polyvalm”, dando como resultado una matriz de igual dimensión que A.

>> A = [0 1; 1 1]; p = [1, -1, -1]; B = polyvalm(p, A)

$\rightarrow$ retornando la matriz B de dimensión 2×2

```
B =
     0     0
     0     0
```

$$A = \begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}, \quad p(x) = \det(Ix - A) = x^2 - x - 1$$

$$p(A) = A^2 - A - 1 = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

Este resultado confirma el teorema de Cayley-Hamilton:

“toda matriz satisface su propio polinomio característico”,

porque *p* es el polinomio característico de A.

Métodos numéricos en MATLAB

Multiplicación de polinomios

Dados 2 polinomios p y q, su producto m se calcula usando la función m = conv(p,q).

>> p = [1, -3, 2]; q = [1, 3 ,2]; m = conv(p,q)

$\rightarrow$ retorna el polinomio m de coeficientes 1 0 -5 0 4

$$m(x) = (x^2 - 3x + 2)(x^2 + 3x + 2) = x^4 - 5x^2 + 4$$

División de polinomios

Cuando un polinomio g se divide por un polinomio h, existen polinomios q y r tal que g(x) = h(x) q(x) + r(x), donde el grado de r es menor que el grado de h. La función disponible en MATLAB para esta operación es “deconv”.

>> g = [1 -6 12 -8]; h = [1, -2]; [q,r] = deconv(g,h)

$\rightarrow$ retorna los polinomios q = 1 -4 4 y r = 0 0 0 0

$$x^3 - 6x^2 + 12x - 8 = (x - 2)(x^2 - 4x + 4) + 0$$

Métodos numéricos en MATLAB

Descomposición en fracciones parciales

$$\frac{2x-5}{x^2-5x+6}=\frac{1}{x-3}+\frac{1}{x-2} \qquad \frac{x^2-3x+1}{x^2-5x+6}=\frac{1}{x-3}+\frac{1}{x-2}+1$$

MATLAB dispone de la función “residue” para realizar esta descomposición:

dados los polinomios a y b, esta calcula los polinomios r, p y k tal que

$$\frac{b}{a}=\frac{r(1)}{x-p(1)}+\frac{r(2)}{x-p(2)}+\cdots+\frac{r(n)}{x-p(n)}+k$$

es decir, retorna la descomposición en fracciones parciales de b/a.

>> b = [2, -5]; a = [1, -5, 6]; [r, p, k] = residue(b,a)

→ retorna los vectores r = [1 1], p = [3 2], k = [ ]

>> b = [1, -3, 1]; a = [1, -5, 6]; [r, p, k] = residue(b,a)

→ retorna los vectores r = [1 1], p = [3 2], k = [1]

Métodos numéricos en MATLAB

Descomposición en fracciones parciales

Ejercicios:

$$\frac{4/3\,x^2+13/3\,x+2}{x^3+4x^2+3x}=\frac{1/6}{x+3}+\frac{1/2}{x+1}+\frac{2/3}{x}$$

$$\frac{x^2-2x+1}{x^3+3x^2+4x+2}=\frac{-1.5+2i}{x+1-i}+\frac{-1.5-2i}{x+1+i}+\frac{4}{x+1}$$

$$\frac{5x-1}{x^3-3x-2}=\frac{1}{x-2}-\frac{1}{x+1}+\frac{2}{(x+1)^2}$$

$$\frac{x^3+2x-4}{x^2+4x-2}=x-4+\frac{20.6145}{x+4.4495}-\frac{0.6145}{x-0.4495}$$

Métodos numéricos en MATLAB

Derivación de polinomios

- La función “polyder” calcula la derivada de un polinomio dado. La derivada del polinomio p = [1, 0, -2, -5] es el polinomio q

  >> q = polyder([1, 0, -2, -5]); $\rightarrow$ retorna el polinomio q = [3, 0, –2]

  $$p(x) = x^3 - 2x - 5 \quad \Rightarrow \quad q(x) = p'(x) = 3x^2 - 2$$

- Cálculo de la derivada del producto y cociente de 2 polinomios dados:

  Derivada de a*b es c

  >> a = [1, 3, 5]; b = [2, 4, 6]; c = polyder(a,b) $\rightarrow$ c = [8 30 56 38]

  $$a(x) = x^2 + 3x + 5, \;\; b(x) = 2x^2 + 4\mathrm{x} + 6$$

  $$\Rightarrow \quad \mathrm{c(x)} = (a(x) \cdot b(x))' = 8x^3 + 30x^2 + 56x + 38$$

  Derivada de a/b es la fracción racional q/d

  >> a = [1, 3, 5]; b = [2, 4, 6]; [q, d] = polyder(a,b) $\rightarrow$ q = [-2 -8 -2] y d = [4 16 40 48 36]

  $$a(x) = x^2 + 3x + 5, \;\; b(x) = 2x^2 + 4\mathrm{x} + 6$$

  $$\Rightarrow \quad \mathrm{c(x)} = (a(x)/b(x))' = \frac{-2x^2 - 8x - 2}{4x^4 + 16x^3 + 40x^2 + 48x + 36}$$

Métodos numéricos en MATLAB

Ajuste de un polinomio a un conjunto de puntos

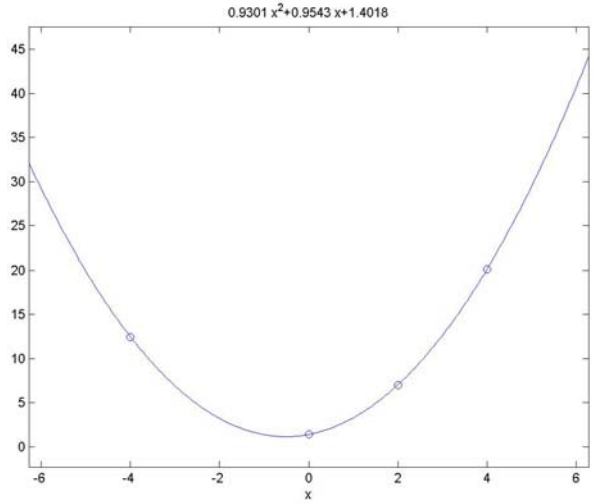
La función “polyfit” ajusta un polinomio p(x) de grado n a un conjunto de puntos (x_i, y_i) para i = 1, …, m con m > n, es decir, $p(x_i) \approx y_i$ en el sentido de mínimos cuadrados.

polyfit(x, y, n) devuelve los coeficientes del polinomio p

```
>> x = [-4, 0, 2]; y = [13, 1, 7]
>> p = polyfit(x, y, 2);
```

→ retorna el polinomio p = [1, 1, 1]

En este caso, p(-4) = 13, p(0) = 1, p(2) = 7




```
>> x = [-4, 0, 2, 4]; y = [12.5, 1.2, 7.3, 20]
>> p = polyfit(x, y, 2);
```

→ retorna p = [0.9301, 0.9543, 1.4018]

En este caso,

p(-4) = 12.4664, p(0) = 1.018, p(2) = 7.0309, p(4) = 20.1009

Métodos numéricos en MATLAB

Ejercicios con polinomios.

```
x = 0:0.1:1;
y = [-0.447, 1.978, 3.28, 6.16, 7.08, 7.34, 7.66, 9.56, 9.48, 9.30, 11.2];
```

Interpolar polinomios de grado 2 y 10 para los datos.
Representar sobre una misma gráfica los puntos y ambos polinomios.
¿Qué se observa?

Métodos numéricos en MATLAB

Determinación de los ceros de una función de una variable

MATLAB dispone de la función “fzero” para intentar determinar un cero de una función “fun” de una variable cerca de un punto “x0” dado.

x = fzero(fun, x0)

>> fzero('cos(x)-x', 0) $\rightarrow$ retorna ans = 0.7391

Dado que “fzero” busca puntos donde la función cambia de signo, este no funciona para ceros de multiplicidad par. Cuando “fzero” falla, esta retorna un NaN.

>> fzero('x^2 + 4*x + 4', 0) $\rightarrow$ retorna ans = NaN

Si “x0” es un vector de 2 elementos, tal que fun(x0(1)) y fun(x0(2)) tienen signos opuestos, “fzero” trabaja en el intervalo definido por x0.

>> fzero('cos(x)-x', [0, 1]) $\rightarrow$ retorna ans = 0.7391

Métodos numéricos en MATLAB

Determinación de los ceros de una función de una variable

"fzero" adicionalmente puede retornar el valor de la función en la raíz x calculada: [x, fval] = fzero(fun, x0).

Esto permite determinar casos atípicos.

```
>> [x, fval] = fzero('x-tan(x)',1)
```

→ retorna

x = 1.5708

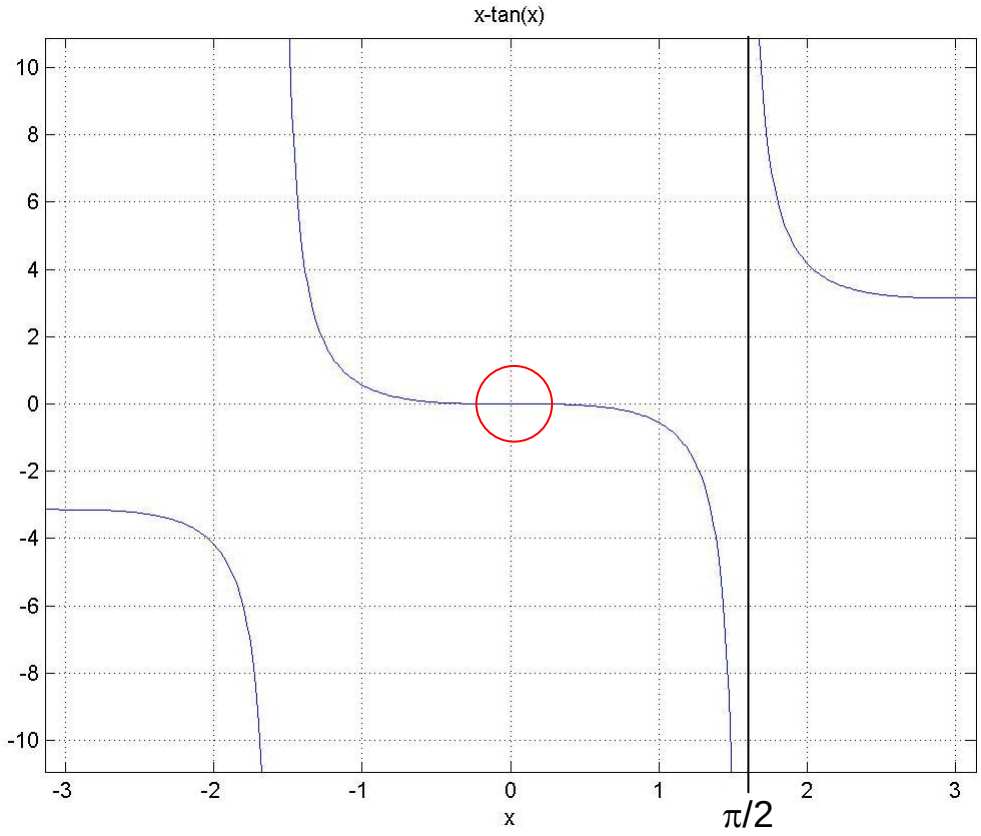
fval = 1.2093e+015

```
>> [x, fval] = fzero('x-tan(x)', [-1 1])
```

→ retorna

x = 0 fval = 0

```
>> ezplot('x-tan(x)', [-pi,pi])
```



Métodos numéricos en MATLAB

Determinación de los ceros de una función de una variable

Ejemplo:

Graficación de la familia de curvas cos(a+x) - (a+x) para diferentes valores de a y cálculo de los ceros de cada curva:

Definimos la función de 2 parámetros myfunc

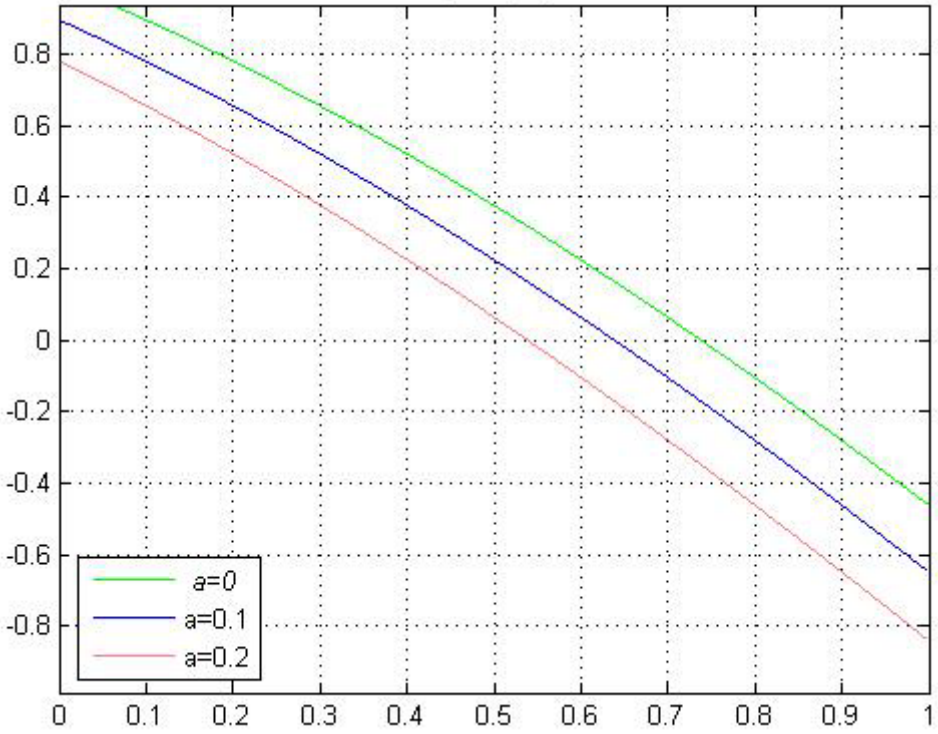
```
function f = myfun(x,a)
        f = cos(a+x) - (a+x);
end
```

```
>> a=0; ezplot(@(x) myfun(x,a),[0,1]);
>> hold on
>> x = fzero(@(x) myfun(x,a),1);

>> a=0.1; ezplot(@(x) myfun(x,a),[0,1]);
>> x = fzero(@(x) myfun(x,a),1);

>> a=0.2; ezplot(@(x) myfun(x,a),[0,1]);
x = fzero(@(x) myfun(x,a),1)

>> grid on
>> legend('\it a=0','a=0.1','a=0.2',3);
```



Métodos numéricos en MATLAB

Otras funciones en Matlab

[x,fval] = fminbnd(fun,x1,x2)	Trata de encontrar el mínimo local x de la función fun en el intervalo [x1,x2]. El valor mínimo alcanzado retorna en fval.
[x,fval] = fminsearch('fun',x0)	Minimización no lineal multidimensional sin restricciones basado en el método de Nelder-Mead, de la función fun con punto inicial x0. Retorna el punto mínimo x y su valor mínimo fval.
z = trapz(y)	Calcula una aproximación de la integral de y vía el método de trapecios (usa espaciamiento uniforme de 1). Para calcular la integral para un espaciamiento uniforme diferente de 1, multiplicamos z por el espaciamiento.
q = quad(fun,a,b)	Trata de aproximar la integral de la función fun entre a y b con un error de 10-6 usando el método recursivo adaptativo de cuadratura de Simpson.
[t,y] = ode23(odefun,[t0 tfinal],y0)	Integra numéricamente el sistema de ecuaciones diferenciales y'=f(t,y) desde t0 hasta tfinal con la condición inicial y0. odefun(t,y) corresponde a la función f(t,y). Está basado en el método de Runge-Kutta de orden 2 y 3.
[t,y] = ode45(odefun,[t0 tfinal],y0)	Integra numéricamente el sistema de ecuaciones diferenciales y'=f(t,y) desde t0 hasta tfinal con la condición inicial y0. odefun(t,y) corresponde a la función f(t,y). Está basado en el método de Runge-Kutta de orden 4 y 5.

Métodos numéricos en MATLAB

Otras funciones en Matlab

[x,fval] = fsolve(F,x0,…)	Intenta resolver un sistema de la forma F(x)=0, donde x es un vector, comenzando en el vector x0. Retorna en x el vector solución y en fval los valores de F en x.
pcg(A,b,tol,maxit)	Trata de resolver el sistema Ax=b, para A una matriz n×n simétrica y definida positiva, b un vector columna de longitud n, tol la tolerancia del método, y maxit el número máximo de iteraciones. pcg usa el método de gradiente conjugado precondicionado.
yi = interp1 (x,y,xi,'metodo')	Interpola los puntos dados por x, y para determinar el valor yi para el valor dado xi, usando el algoritmo especificado por 'metodo', por ejemplo: linear, cubic, splines, nearest. La opción por defecto es 'linear'.

Métodos numéricos en MATLAB

Minimizando funciones de una función de una variable

Dada una función de 1 variable codificada en un archivo tipo M, se puede usar la función de MATLAB fminbnd para encontrar su mínimo en un intervalo dado.

Ejemplo: determinar un mínimo de la función “humps” en el intervalo (0.3, 1)

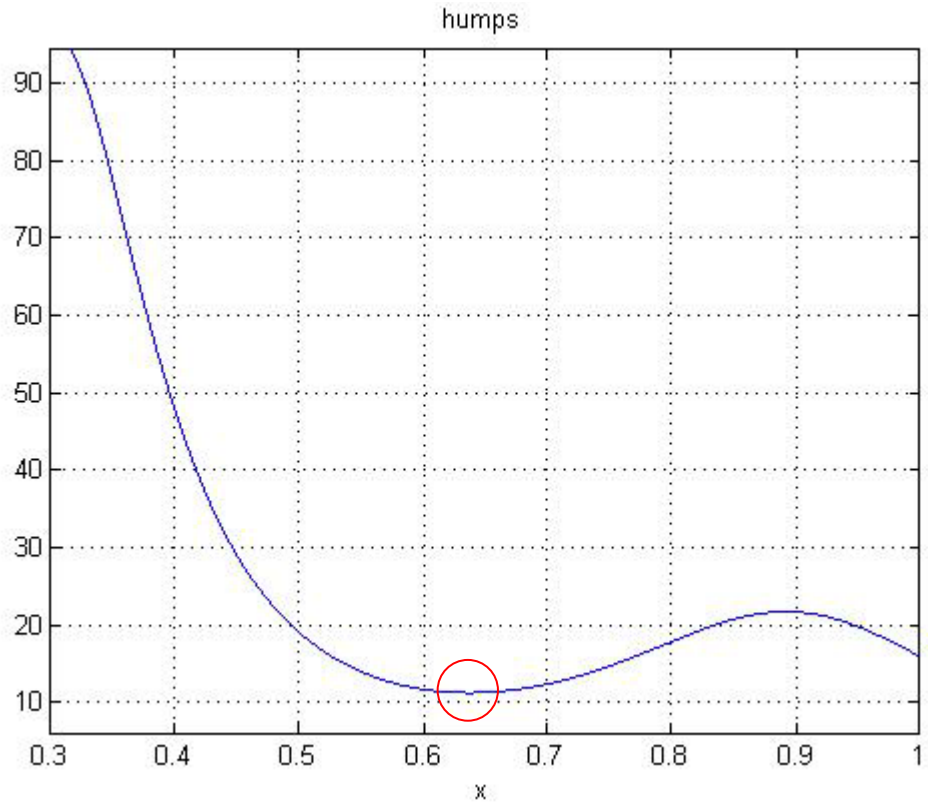
```
>> x = fminbnd(@humps,0.3,1)
```

el cual retorna

x = 0.6370

y = humps(x) es una función con máximos cerca de
x = .3 y x = .9.

```
>> ezplot(@humps,0.3,1)
```



Determinar estos máximos.

Usar f = @(x) -1*humps(x)

Métodos numéricos en MATLAB

Minimizando funciones de una función de una variable

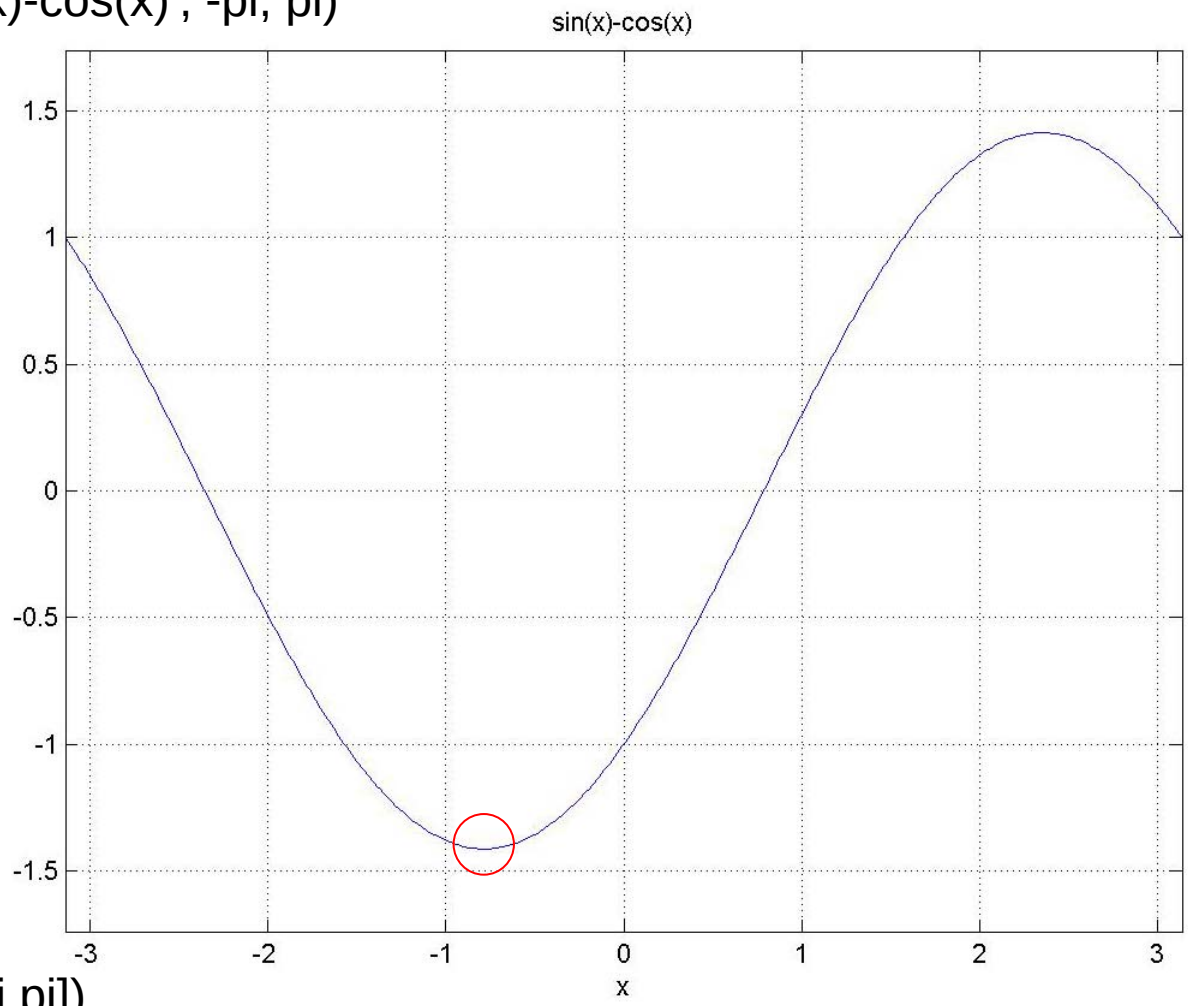
Ejemplo:

```
>> [x,fval] = fminbnd('sin(x)-cos(x)', -pi, pi)
```

retorna

x = -0.7854,

fval = -1.4142




```
>> ezplot('sin(x)-cos(x)', [-pi,pi])
```

Métodos numéricos en MATLAB

Minimizando funciones de una función de una variable

Ejemplo:

```
>> x = fminbnd(@humps,0.3,1,optimset('Display','iter'))

 Func-count     x          f(x)         Procedure
    1       0.567376      12.9098       initial
    2       0.732624      13.7746       golden
    3       0.465248      25.1714       golden
    4       0.644416      11.2693       parabolic
    5         0.6413      11.2583       parabolic
    6       0.637618      11.2529       parabolic
    7       0.636985      11.2528       parabolic
    8       0.637019      11.2528       parabolic
    9       0.637052      11.2528       parabolic

Optimization terminated:
 the current x satisfies the termination criteria using OPTIONS.TolX of 1.000000e-004

x =
    0.6370
```

Probar con:

[x,fval] = fminbnd(@humps,0.3,1,optimset('Display','iter'))

Métodos numéricos en MATLAB

Aproximando una integral

Función quad

Trata de aproximar la integral de la función “fun” entre a y b con un error de 10^{-6} usando el método recursivo adaptativo de cuadratura de Simpson.

Sintaxis: q = quad(fun,a,b,tol)

fun: función a integrar, [a,b] intervalo de integración, tol: tolerancia para el error.

Ejemplo:

```
>> q = quad('1./(x.^3-2*x-5)', 0, 1);
```

→ q = -0.1745

```
>> ezplot('1./(x.^3-2*x-5)', [0, 1])
```

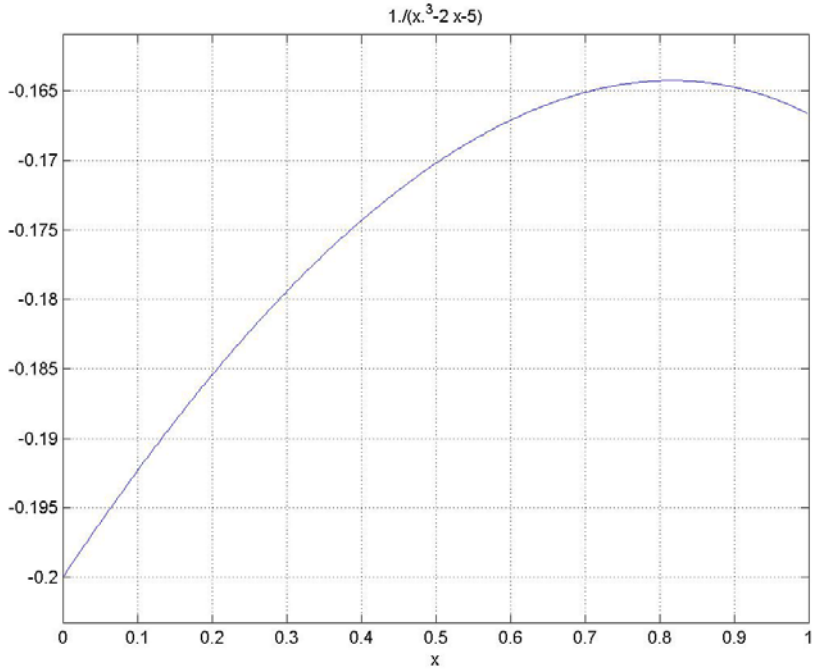
Obs. Se puede definir la función como

```
f = @(x) 1./(x.^3-2*x-5)
```

o

```
f = inline('1./(x.^3-2*x-5)')
```

```
>> ezplot( f, [0, 1])
```



Métodos numéricos en MATLAB

Aproximando una integral

Ejemplo: Aproximar las integrales siguientes:

$$\int_0^1 \frac{1}{\sqrt{1+x^4}}\,dx$$

= 0.9270

```
f = inline('1./sqrt(1+x.^4)')
q = quad(f,0,1)
```

$$\int_0^{\pi} \frac{\sin x}{x}\,dx$$

= 1.8519

```
f = inline('sin(x)./x')
q = quad(f,0,pi)

comparar con
q = quad(f,realmin,pi)
```

Métodos numéricos en MATLAB

Resolución numérica de ecuaciones diferenciales ordinarias

Encontrar la solución de $\frac{d}{dt} y(t) = f(t, y(t)), \quad a \le t \le b$

en conjunto con la condición inicial $y(t_0) = y_0$

- `ode45`. Nonstiff problems, medium accuracy. Use most of the time. This should be the first solver you try.
- `ode23`. Nonstiff problems, low accuracy. Use for large error tolerances or moderately stiff problems.
- `ode113`. Nonstiff problems, low to high accuracy. Use for stringent error tolerances or computationally intensive ODE functions.
- `ode15s`. Stiff problems, low to medium accuracy. Use if `ode45` is slow (stiff systems) or there is a mass matrix.
- `ode23s`. Stiff problems, low accuracy, Use for large error tolerances with stiff systems or with a constant mass matrix.
- `ode23t`. Moderately stiff problems, low accuracy, Use for moderately stiff problems where you need a solution without numerical damping.
- `ode23tb`. Stiff problems, low accuracy. Use for large error tolerances with stiff systems or if there is a mass matrix.

Métodos numéricos en MATLAB

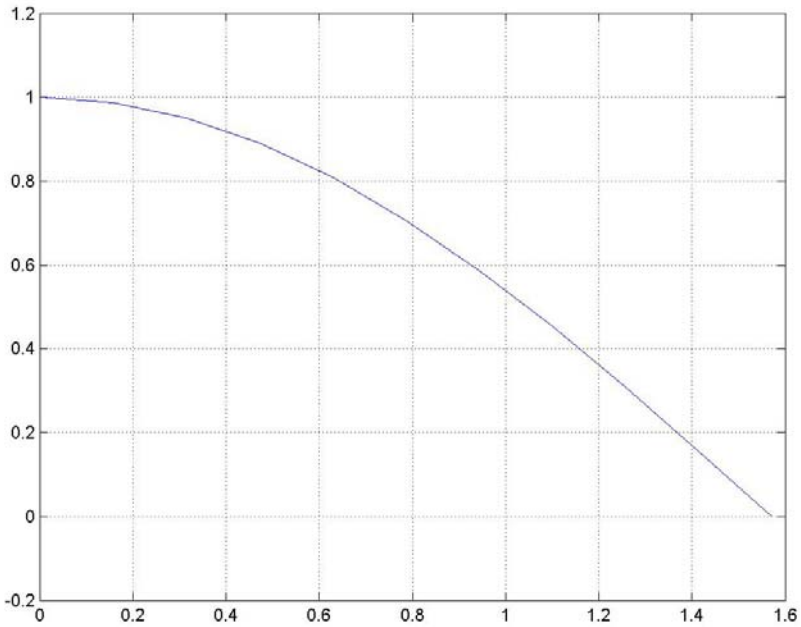
Resolución numérica de ecuaciones diferenciales ordinarias

Ejemplo: $\frac{d}{dt}y(t) = -\sin(t), \quad y(0) = 1, \quad 0 \le t \le \pi/2$

```
1  % funcion que evalua la funcion yprime = -sin(x);
2
3  function yprime = odefun(x,y)
4
5  yprime = -sin(x);
```

>> [t, y] = ode23('odefun', [0 pi/2], 1)

t =	y =
0	1.0000
0.1571	0.9877
0.3142	0.9511
0.4712	0.8910
0.6283	0.8090
0.7854	0.7071
0.9425	0.5878
1.0996	0.4540
1.2566	0.3090
1.4137	0.1564
1.5708	-0.0000



>> plot(t,y); grid on

Métodos numéricos en MATLAB

Resolución numérica de ecuaciones diferenciales ordinarias

Ejemplo:

Dado el siguiente diagrama de reacciones:

$$H + NO_2 \xrightarrow{k_1} OH + NO$$
$$OH + OH \xrightarrow{k_2} H_2O + O$$
$$O + OH \xrightarrow{k_3} O_2 + H$$

Estas reacciones se caracterizan por unas *constantes de reacción*, denotadas por k_i en este ejemplo, las cuales determinan la *velocidad* con la que se producen/consumen las diferentes sustancias (especies) químicas que intervienen en una reacción determinada. Cada vez que una sustancia aparece en el lado izquierdo de una reacción, la misma se consume proporcionalmente al producto de las concentraciones de las sustancias reaccionantes, siendo la constante de proporcionalidad justamente la constante asociada a la reacción. Para expresar matemáticamente que una sustancia se consume, se utiliza la constante de reacción con signo negativo. Análogamente, si una sustancia aparece en el lado derecho de una reacción, entonces dicha sustancia se produce proporcionalmente al producto de las concentraciones de las sustancias reaccionantes, y la constante de proporcionalidad es la constante asociada a la reacción. Luego, si se denota por $[S]$ la concentración de la sustancia S, es posible escribir expresiones matemáticas para la variación en el tiempo de las concentraciones de todas las sustancias que intervienen en el reactor, lo cual se traduce en el siguiente *sistema de ecuaciones diferenciales ordinarias* (las concentraciones sólo dependen del tiempo):

Métodos numéricos en MATLAB

Resolución numérica de ecuaciones diferenciales ordinarias

Ejemplo:

Dado el siguiente diagrama de reacciones:

$$H + NO_2 \xrightarrow{k_1} OH + NO$$
$$OH + OH \xrightarrow{k_2} H_2O + O$$
$$O + OH \xrightarrow{k_3} O_2 + H$$

se tiene el sistema de ecuaciones ordinarias asociado siguiente:

$$\begin{cases} \frac{d[H]}{dt} = -k_1[H][NO_2] + k_3[O][OH], \\ \frac{d[NO_2]}{dt} = -k_1[H][NO_2], \\ \frac{d[OH]}{dt} = k_1[H][NO_2] - k_2[OH][OH] - k_3[O][OH], \\ \frac{d[NO]}{dt} = k_1[H][NO_2], \\ \frac{d[HO_2]}{dt} = k_2[OH][OH], \\ \frac{d[O]}{dt} = k_2[OH][OH] - k_3[O][OH], \\ \frac{d[O_2]}{dt} = k_3[O][OH], \end{cases}$$

Métodos numéricos en MATLAB

Resolución numérica de ecuaciones diferenciales ordinarias

Ejemplo (cont.):

denotamos las concentraciones de cada especie como

$$x_1 = [H], \quad x_2 = [NO_2], \quad x_3 = [OH], \quad x_4 = [NO],$$
$$x_5 = [H_2O], \quad x_6 = [O], \quad x_7 = [O_2],$$

el sistema se reescribe como

$$\begin{cases} \dfrac{dx_1}{dt} = -k_1 x_1 x_2 + k_3 x_6 x_3, \\ \dfrac{dx_2}{dt} = -k_1 x_1 x_2, \\ \dfrac{dx_3}{dt} = k_1 x_1 x_2 - k_2 x_3^2 - k_3 x_6 x_3, \\ \dfrac{dx_4}{dt} = k_1 x_1 x_2, \\ \dfrac{dx_5}{dt} = k_2 x_3^2, \\ \dfrac{dx_6}{dt} = k_2 x_3^2 - k_3 x_6 x_3, \\ \dfrac{dx_7}{dt} = k_3 x_6 x_3, \end{cases}$$

Se plantea el sistema

$$\frac{d}{dt} x(t) = F(t, y(t))$$

$$t \in [0, 0.01]$$

$$x(t) = (x_1, x_2, x_3, x_4, x_5, x_6, x_7)'$$

$$F(t, x(t)) = (-k_1 x_1 x_2 + k_3 x_6 x_3, -k_1 x_1 x_2,$$
$$k_1 x_1 x_2 - k_2 x_3^2 - k_3 x_6 x_3, k_1 x_1 x_2, k_2 x_3^2, k_2 x_3^2 - k_3 x_6 x_3, k_3 x_6 x_3)'$$

el cual se complementa con las condiciones iniciales

$$x(0) = (4.5 \cdot 10^{-10}, 5.6 \cdot 10^{-10}, 0, 0, 0, 0, 0)'$$

Métodos numéricos en MATLAB

Resolución numérica de ecuaciones diferenciales ordinarias

Ejemplo (cont.):

Función que evalúa F (término de la derecha en la EDO)

```
function y = cinetica(t,x)
  k1 = 2.9e13;
  k2 = 1.55e12;
  k3 = 1.1e13;

  y(1,1) = -k1*x(1)*x(2) + k3*x(6)*x(3);
  y(2,1) = -k1*x(1)*x(2);
  y(3,1) = k1*x(1)*x(2) -k2*x(3)^2 - k3*x(6)*x(3);
  y(4,1) = k1*x(1)*x(2);
  y(5,1) = k2*x(3)^2;
  y(6,1) = k2*x(3)^2 - k3*x(6)*x(3);
  y(7,1) = k3*x(6)*x(3);
```

El sistema es rígido (“stiff”). Se usará la función “ode23s” de MATLAB

```
>> [t,y] = ode23s(@cinetica, [0:0.0001:0.01], [4.5e-10,5.6e-10,0,0,0,0,0]);
```

Métodos numéricos en MATLAB

Resolución numérica de ecuaciones diferenciales ordinarias

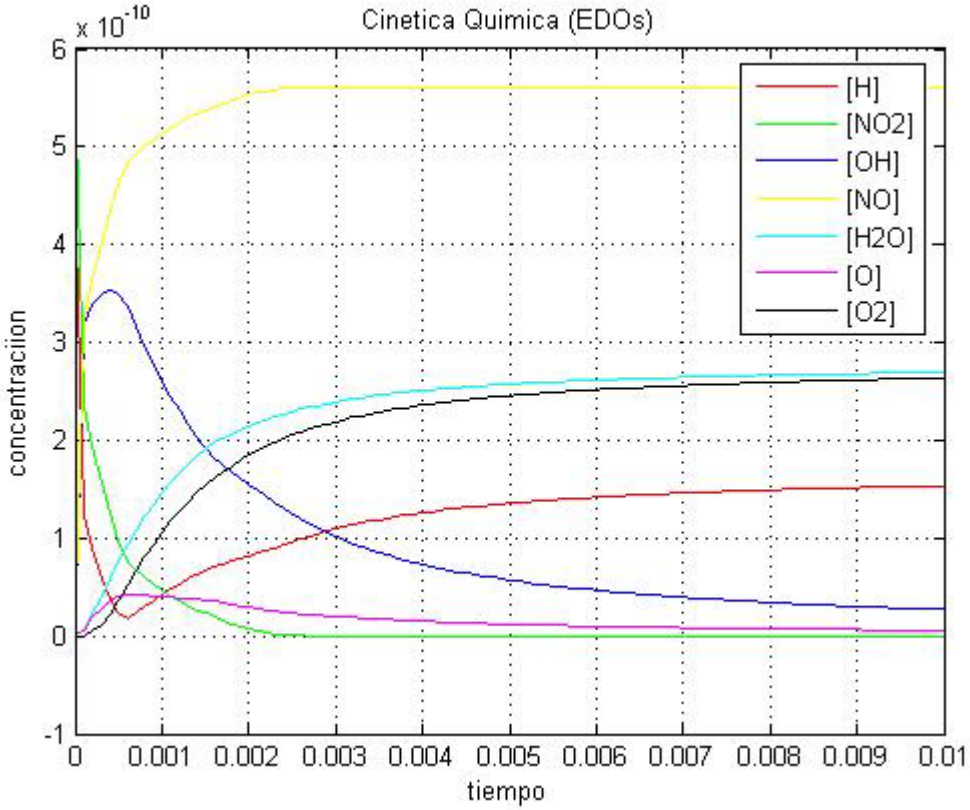
Ejemplo (cont.):

se procede a graficar los valores de las concentraciones en el tiempo

```
>> plot(t,y(:,1),'r', t,y(:,2),'g', t,y(:,3),'b', t,y(:,4),'y', ...
   t,y(:,5),'c', t,y(:,6),'m', t,y(:,7),'k');
```

Se agregan los títulos

```
>> title('Cinetica Quimica (EDOs)');
>> xlabel('tiempo');
>> ylabel('concentracion');
>> legend('[H]', '[NO2]', '[OH]', …
          '[NO]', '[H2O]', '[O]', '[O2]');
```



cinetica_driver.m

Graficas en 3 dimensiones

Función “ezmesh”

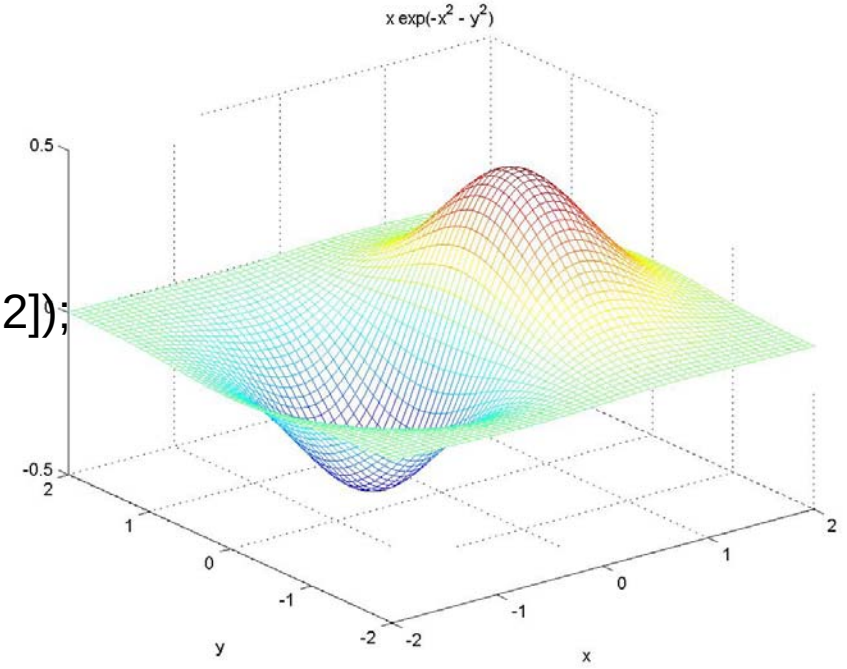
ezmesh(f): dibuja la función f(x,y) en el dominio predeterminado

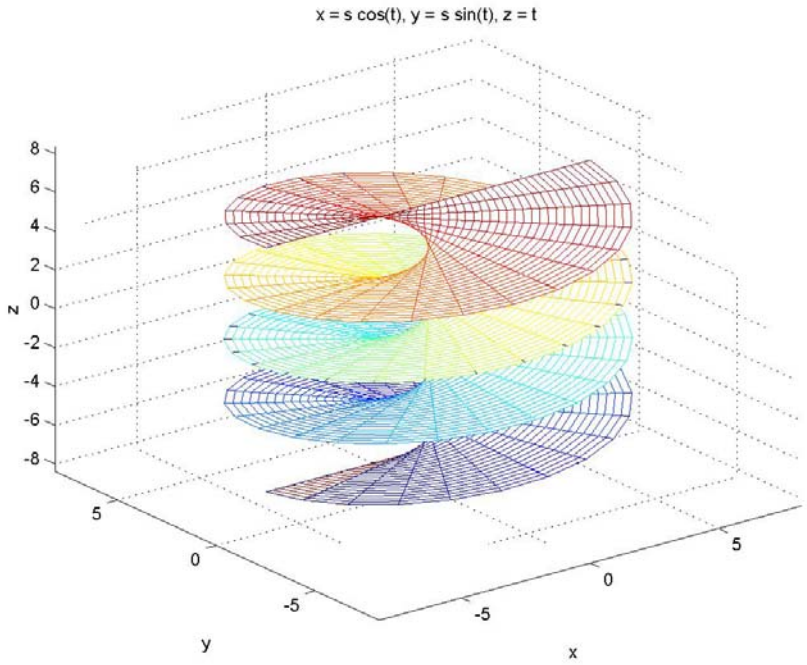
-2*pi ≤ x ≤ 2*pi, -2*pi ≤ y ≤ 2*pi.

```
>> ezmesh('x*exp(-x^2 - y^2)', [-2,2]);
```

o

```
>> ezmesh(@(x,y) x*exp(-x^2 - y^2), [-2,2]);
```





También se pueden dibujar funciones paramétricas, es decir,

x = x(s,t), y = y(s,t), z = z(s,t) con

smin ≤ s ≤ smax, tmin ≤ t ≤ smat

```
>> ezmesh('s*cos(t)','s*sin(t)','t');
```

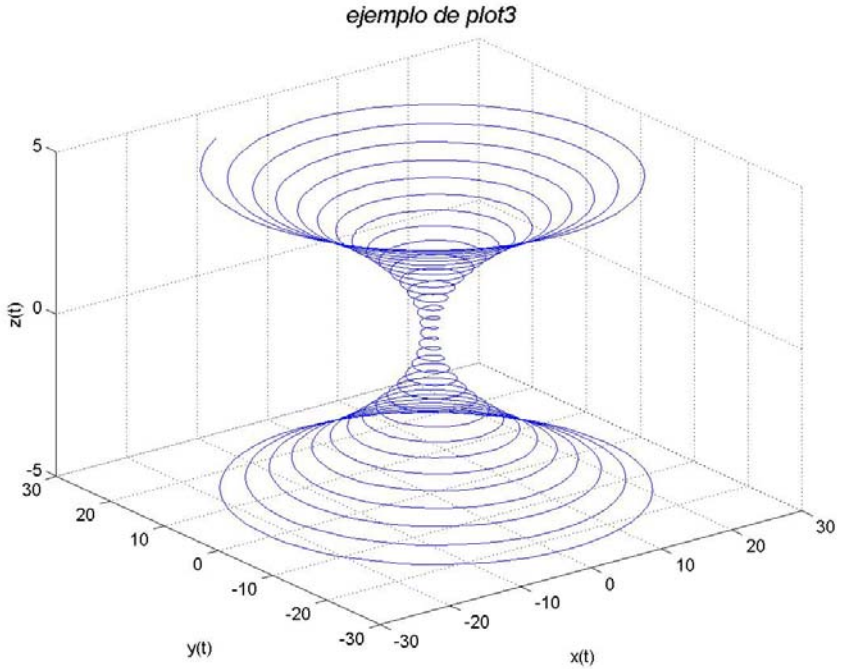
Graficas en 3 dimensiones

Función “plot3”

Esta función es el análogo en 3 dimensiones a “plot”, es decir, dibujar secuencia de puntos en el espacio. Así, plot3(x,y,z) dibuja los puntos x(i), y(i), z(i) en el espacio en el orden especificado por los vectores x, y, z uniéndolos por segmentos.

$$x(t) = \left(1 + t^2\right)\sin(20t), \quad y(t) = \left(1 + t^2\right)\cos(20t), \quad z(t) = t$$

```
>> t = -5 : 0.005 : 5;
>> x = (1+t.^2).*sin(20*t);
>> y = (1+t.^2).*cos(20*t);
>> z = t;
>> plot3(x, y, z);
>> grid on
>> xlabel('x(t)'); ylabel('y(t)'); zlabel('z(t)');
>> title('\it {ejemplo de plot3}', 'fontsize', 14)
```



Obs: noten las operaciones matriciales (.^ , .*).

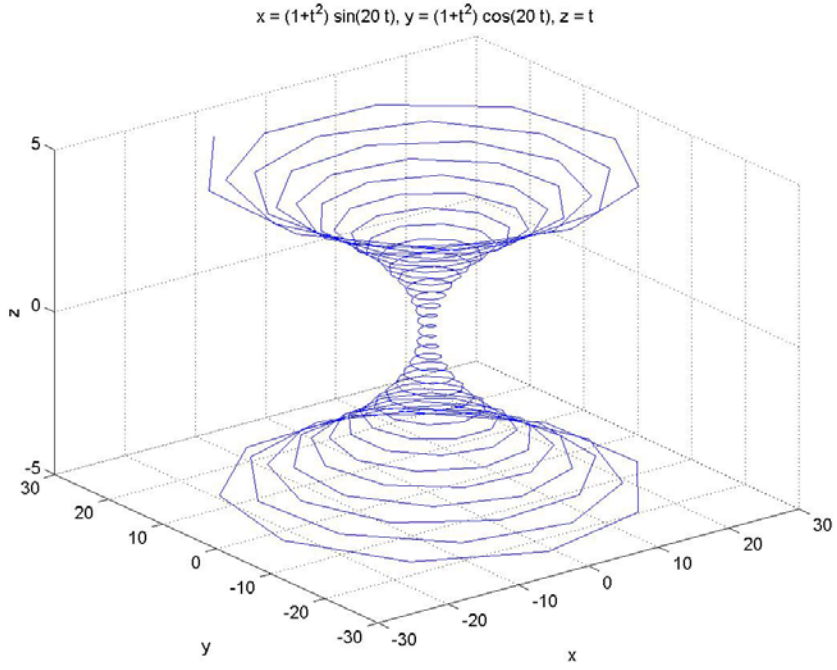
Graficas en 3 dimensiones

Función “ezplot3”

Esta función es el análogo en 3 dimensiones a “ezplot”, es decir, dibuja la curva descrita en forma paramétrica por las funciones x(t), y(t), z(t) sobre el domino 0 < t < 2*pi.

$$x(t) = \left(1 + t^2\right)\sin(20t), \quad y(t) = \left(1 + t^2\right)\cos(20t), \quad z(t) = t$$

```
>> x = @(t) (1+t.^2).*sin(20*t);
>> y = @(t) (1+t.^2).*cos(20*t);
>> z = @(t) t;
>> ezplot3(x,y,z, [-5,5]);
```




```
>> ezplot3(x,y,z, [-5,5], ’animate’);
```

produce una animación del recorrido de la curva