FUNCIONES MATLAB

help elfun : te da todas las funciones elementales de matemáticas

Trigonometric.
sin - Sine.
sind - Sine of argument in degrees.
sinh - Hyperbolic sine.
asin - Inverse sine.
asind - Inverse sine, result in degrees.
asinh - Inverse hyperbolic sine.
cos - Cosine.
cosd - Cosine of argument in degrees.
cosh - Hyperbolic cosine.
acos - Inverse cosine.
acosd - Inverse cosine, result in degrees.
acosh - Inverse hyperbolic cosine.
tan - Tangent.
tand - Tangent of argument in degrees.
tanh - Hyperbolic tangent.
atan - Inverse tangent.
atand - Inverse tangent, result in degrees.
atan2 - Four quadrant inverse tangent.
atan2d - Four quadrant inverse tangent, result in degrees.
atanh - Inverse hyperbolic tangent.
sec - Secant.
secd - Secant of argument in degrees.
sech - Hyperbolic secant.
asec - Inverse secant.
asecd - Inverse secant, result in degrees.
asech - Inverse hyperbolic secant.
csc - Cosecant.
cscd - Cosecant of argument in degrees.
csch - Hyperbolic cosecant.
acsc - Inverse cosecant.
acscd - Inverse cosecant, result in degrees.
acsch - Inverse hyperbolic cosecant.
cot - Cotangent.
cotd - Cotangent of argument in degrees.
coth - Hyperbolic cotangent.
acot - Inverse cotangent.
acotd - Inverse cotangent, result in degrees.
acoth - Inverse hyperbolic cotangent.
hypot - Square root of sum of squares.
deg2rad - Convert angles from degrees to radians.
rad2deg - Convert angles from radians to degrees.

Exponential.
exp - Exponential.
expm1 - Compute exp(x)-1 accurately.
log - Natural logarithm.
log1p - Compute log(1+x) accurately.
log10 - Common (base 10) logarithm.
log2 - Base 2 logarithm and dissect floating point number.
pow2 - Base 2 power and scale floating point number.
realpow - Power that will error out on complex result.
reallog - Natural logarithm of real number.
realsqrt - Square root of number greater than or equal to zero.
sqrt - Square root.
nthroot - Real n-th root of real numbers.

nextpow2 - Next higher power of 2.

Complex.
abs - Absolute value.
angle - Phase angle.
complex - Construct complex data from real and imaginary parts.
conj - Complex conjugate.
imag - Complex imaginary part.
real - Complex real part.
unwrap - Unwrap phase angle.
isreal - True for real array.
cplxpair - Sort numbers into complex conjugate pairs.

Rounding and remainder.
fix - Round towards zero.
floor - Round towards minus infinity.
ceil - Round towards plus infinity.
round - Round towards nearest integer.
mod - Modulus (signed remainder after division).
rem - Remainder after division.
sign - Signum.

help elmat : te da las operaciones con matrices

Elementary matrices.
zeros - Zeros array.
ones - Ones array.
eye - Identity matrix.
repmat - Replicate and tile array.
repelem - Replicate elements of an array.
linspace - Linearly spaced vector.
logspace - Logarithmically spaced vector.
freqspace - Frequency spacing for frequency response.
meshgrid - X and Y arrays for 3-D plots.
accumarray - Construct an array with accumulation.
: - Regularly spaced vector and index into matrix.

Basic array information.
size - Size of array.
length - Length of vector.
ndims - Number of dimensions.
numel - Number of elements.
disp - Display matrix or text.
isempty - True for empty array.
isequal - True if arrays are numerically equal.
isequaln - True if arrays are numerically equal, treating NaNs as equal.

Matrix manipulation.
cat - Concatenate arrays.
reshape - Reshape array.
diag - Diagonal matrices and diagonals of matrix.
blkdiag - Block diagonal concatenation.
tril - Extract lower triangular part.
triu - Extract upper triangular part.
fliplr - Flip matrix in left/right direction.
flipud - Flip matrix in up/down direction.
flip - Flip the order of elements.
rot90 - Rotate matrix 90 degrees.
: - Regularly spaced vector and index into matrix.

find - Find indices of nonzero elements.
end - Last index.
sub2ind - Linear index from multiple subscripts.
ind2sub - Multiple subscripts from linear index.
bsxfun - Binary singleton expansion function.

Multi-dimensional array functions.
ndgrid - Generate arrays for N-D functions and interpolation.
permute - Permute array dimensions.
ipermute - Inverse permute array dimensions.
shiftdim - Shift dimensions.
circshift - Shift array circularly.
squeeze - Remove singleton dimensions.

Array utility functions.
isscalar - True for scalar.
isvector - True for vector.
isrow - True for row vector.
iscolumn - True for column vector.
ismatrix - True for matrix.

Special variables and constants.
eps - Floating point relative accuracy.
realmax - Largest positive floating point number.
realmin - Smallest positive floating point number.
intmax - Largest positive integer value.
intmin - Smallest integer value.
flintmax - Largest consecutive integer in floating point format.
pi - 3.1415926535897....
i - Imaginary unit.
inf - Infinity.
nan - Not-a-Number.
isnan - True for Not-a-Number.
isinf - True for infinite elements.
isfinite - True for finite elements.
j - Imaginary unit.
true - True array.
false - False array.

Specialized matrices.
compan - Companion matrix.
gallery - Test matrices.
hadamard - Hadamard matrix.
hankel - Hankel matrix.
hilb - Hilbert matrix.
invhilb - Inverse Hilbert matrix.
magic - Magic square.
pascal - Pascal matrix.
rosser - Classic symmetric eigenvalue test problem.
toeplitz - Toeplitz matrix.
vander - Vandermonde matrix.
wilkinson - Wilkinson's eigenvalue test matrix.

help ops : operadores lógicos

Arithmetic operators.
plus - Plus +
uplus - Unary plus +
minus - Minus -
uminus - Unary minus -

*mtimes - Matrix multiply *
*times - Array multiply .**
mpower - Matrix power ^
*power - Array power .^*
*mldivide - Backslash or left matrix divide *
mrdivide - Slash or right matrix divide /
ldivide - Left array divide .
rdivide - Right array divide ./
idivide - Integer division with rounding option.
kron - Kronecker tensor product

Relational operators.
eq - Equal ==
ne - Not equal ~=
lt - Less than <
gt - Greater than >
le - Less than or equal <=
ge - Greater than or equal >=

Logical operators.
relop - Short-circuit logical AND &&
relop - Short-circuit logical OR ||
and - Element-wise logical AND &
or - Element-wise logical OR |
not - Logical NOT ~
punct - Ignore function argument or output ~
xor - Logical EXCLUSIVE OR
any - True if any element of vector is nonzero
all - True if all elements of vector are nonzero

Special characters.
colon - Colon :
paren - Parentheses and subscripting ( )
paren - Brackets [ ]
paren - Braces and subscripting { }
punct - Function handle creation @
punct - Decimal point .
punct - Structure field access .
punct - Parent directory ..
punct - Continuation ...
punct - Separator ,
punct - Semicolon ;
punct - Comment %
punct - Invoke operating system command !
punct - Assignment =
punct - Quote '
punct - Double quote "
transpose - Transpose .'
ctranspose - Complex conjugate transpose '
horzcat - Horizontal concatenation [,]
vertcat - Vertical concatenation [;]
subsasgn - Subscripted assignment ( ),{ },.
subsref - Subscripted reference ( ),{ },.
numArgumentsFromSubscript - Number of arguments for indexing methods
subsindex - Subscript index
metaclass - Metaclass for MATLAB class ?

Bitwise operators.
bitand - Bit-wise AND.
bitcmp - Complement bits.

bitor - Bit-wise OR.
bitxor - Bit-wise XOR.
bitset - Set bit.
bitget - Get bit.
bitshift - Bit-wise shift.

Set operators.
union - Set union.
unique - Set unique.
intersect - Set intersection.
setdiff - Set difference.
setxor - Set exclusive-or.
ismember - True for set member.

See also arith, relop, slash, function_handle.

help lang : programación

Control flow.
if - Conditionally execute statements.
else - Execute statement if previous IF condition failed.
elseif - Execute if previous IF failed and condition is true.
end - Terminate scope of control statements.
for - Repeat statements a specific number of times.
parfor - Parallel FOR-loop.
while - Repeat statements an indefinite number of times.
break - Terminate execution of WHILE or FOR loop.
continue - Pass control to the next iteration of a loop.
switch - Switch among several cases based on expression.
case - SWITCH statement case.
otherwise - Default SWITCH statement case.
try - Begin TRY block.
catch - Begin CATCH block.
return - Return to invoking function.
error - Display message and abort function.
MException - Constructs MATLAB exception object.
assert - Generate an error when a condition is violated.
rethrow - Reissue error.

Evaluation and execution.
eval - Execute string with MATLAB expression.
evalc - Evaluate MATLAB expression with capture.
feval - Execute the specified function.
evalin - Evaluate expression in workspace.
builtin - Execute built-in function from overloaded method.
assignin - Assign variable in workspace.
run - Run script.

Scripts, functions, classes, and variables.
script - About MATLAB script files.
function - Add new function.
global - Define global variable.
persistent - Define persistent variable.
mfilename - Name of currently executing MATLAB code file.
lists - Comma separated lists.
exist - Check if variables or functions are defined.
mlock - Prevents clearing function from memory.
munlock - Allow clearing function from memory.
mislocked - Determine if function is locked in memory.
precedence - Operator Precedence in MATLAB.

isvarname - True for valid variable name.
iskeyword - Check if input is a keyword.
javachk - Validate level of Java support.
genvarname - Construct a valid MATLAB variable name from a string.
classdef - Define a new class or subclass.

Argument handling.
nargchk - Validate number of input arguments.
nargoutchk - Validate number of output arguments.
nargin - Number of function input arguments.
nargout - Number of function output arguments.
varargin - Variable length input argument list.
varargout - Variable length output argument list.
inputname - Input argument name.
inputParser - Construct input parser object.
ans - Most recent answer.

Message display.
warning - Display warning message.
lastwarn - Last warning message.
disp - Display array.
display - Display array.
details - Display array.

Interactive input.
input - Prompt for user input.
keyboard - Invoke keyboard from MATLAB code file.

Abstract superclasses.
handle - Superclass of all handle classes.
dynamicprops - Support for dynamic properties.

See also general, iofun, ops, datatypes, matfun, funfun, elfun, polyfun.

INTRODUCCIÓN

clc : limpia la pantalla

format long : muchos decimales

format short : pocos decimales

format rat : te deja los números decimales en forma de fracción

vpa 'operación' n : p.e. vpa ‘sqrt(23)’ 20 : te da el resultado con n (20) decimales

a = 1 ; b = 2 : declaración de variables

a + b : suma de variables

a - b : resta de variables

a / b : división de variables

a * b : multiplicación de variables

Reglas para el nombre de las variables :

- Pueden tener una longitud de hasta 63 caracteres en MATLAB7
- Pueden tener letras, y dígitos
- Deben empezar por una letra
- MATLAB es un lenguaje que distingue letras mayúsculas y minúsculas. Por ejemplo: AA,aa,Aa,aA son cuatro variables distintas
- Hay que evitar poner a las variables el nombre de funciones del sistema

sqrt(a) : raiz cuadrada

e == exp(1) : el número e lo escribimos como exp(1)
exp(x) : función exponencial en base e de x

log(x) : logaritmo neperiano (en base e) de x
log10(x) : logaritmo en base 10 de x
log2(x) : logaritmo en base 2 de x

pow2(x) : potencia en base 2 de x (pow2(2) = 4)

round(x) : redondea el entero más proximo

fix(x) : redondea hacia el 0 el entero más proximo

floor(x) : redondea hacia menos infinito el entero más proximo

ceil(x) : redondea hacia infinito el entero más proximo

pi : = 3.1416…

Inf : infinito (∞)

NaN : indeterminación (= 0/0)

i
j : == sqrt(-1)

eps : mínima diferencia entre dos números (= 2^(-52))

realmin : mínimo número utilizable (= 2.2251*10^(-308))
realmax : máximo número utilizable (= 1.7977*10^(308))

sin(x) : seno	asin(x) : arcoseno
cos(x) : coseno	acos(x) : arcococoseno
tan(x) : tangente	atan(x) : arcotangente
csc(x) : cosecante	acsc(x) : arcocosecante
sec(x) : secante	asec(x) : arcosecante
cot(x) : cotangente	acot(x) : arcocotangente

rem(x,y)
mod(x,y) : te dan el resto de x entre y (la diferencia entre ambas es que actúan diferente dependiendo de si los dos números tienen o no el mismo signo)

sign(x) : te da el signo de x

gcd(x,y) : máximo común divisor entre x e y : de 100 y 36 te dará el 4

lcm(x,y) : mínimo común divisor entre x e y : de 100 y 36 te dará el 900

factorial(x) : te da x!

factor(n) : descompone n en factores primos : de 242 te dará 2, 11 y 11

Operadores relacionales:

== : igual
~= : diferente
< : menor que
> : mayor que
>= : mayor o igual
<= : menor o igual
en todos saldrá 1 si se cumple y 0 si no se cumple

ejemplo :

r=[8 12 9 4 23 19 10]; : con un vector r…
s=r<=10 : …creamos uno s con los indices de los valores mayores o iguales a 10 (de r)
t=r(s) : …creamos uno t con los valores correspondientes a los indices de s
w=r(r<=10) : …creamos uno w con los valores de r mayores o iguales a 10

x&y = and(x,y)
x|y = or(x,y)
~(x+y) = not(x+y)

xor(x,y) : te da un 1 donde sen diferentes ambos vectores

all(v) : te saldrá 1 si no hay ningún cero
any(v) : te saldrá 1 si hay alguno diferente a cero

find(x>6) : te buscará dentro del vector v todos los que son mayores que 6 (en este caso)

fprintf('lo que quieres escribir',x) : siendo x una variable con un valor asignado
fprintf(',\n') : salto de linea

VECTORES

v = 1:10 : crea un vector de 1 a 10 de 1 en 1
v = 1:2:10 : crea un vector de 1 a 10 de 2 en 2
v = [1 3 5 4 3] : crea un vector (1 3 5 4 3) llamado v

size(v) : te da las dimensiones del vector (como si fuera una matriz, filas y columnas)
length(v) : te da el número de valores que tiene el vector

v + w : (siendo vectores) opera con ellos

x = input(‘cuanto vale?’) : te lo escribe y tienes que meter el valor
y = input(‘nombre?’ , ’s’) : te lo escribe y tienes que meter el valor como string

disp (‘La solución es’) : pega ese texto antes

w’ : ese símbolo hace que se trasponga

v .* w : multiplica componente a componente

1 ./ x : dividir componente a componente

v(3) : valor de v en la posición 3

v(3) = 5 : le asigna a la posición 3 el valor 5

vv = v(3,4) : creamos un nuevo vector vv con los valores de el vector v en las posiciones 3 y 4

vvv = linespace(3,14,7) : vector de 3 a 14 con 7 componentes

ejemplo : V1=linespace(3,14,7),V1(6) : te saldrán los resultados de ambas separadas por comas

prod(v) : da el producto de todos los elementos del vector

sum(v) : suma todos los elementos del vector

max(v) : máximo valor del vector v
min(v) : mínimo valor del vector v

poly(v) : v serán las raíces de un polinomio (posibles resultados : x -), poly lo que nos dará será el polinomio correspondiente a estas
poly([1 1]) == (x - 1)*(x - 1) == x^2 - 2*x + 1 → resultado = (1 -2 1)

poly2sym(v,’x’) : nos convierte el vector en la base de un polinomio con variable, en este caso, x

sym2poly(po) : nos convierte el polinomio ‘po’ en un vector con sus bases

roots(v) : nos da los posibles resultados (las raíces : x -) del polinomio cuyos coeficientes son de mayor a menor orden los elementos del vector v()

sort(v) : ordena de forma creciente (de menor a mayor) los valores de un vector

Dos vectores serán linealmente independientes cuando uniéndolos por columnas en una matriz y haciendo un rref() se escalona adecuadamente

MATRICES

A = [1 2 3 4 ; 1 2 3 4 ; 5 6 7 8 ; 5 6 7 8] : crea una matriz (metemos de fila en fila)

A’
traspose(A) : hace la traspuesta

det(A) : determinante

rank(A) : rango de la matriz

trace(A) : traza de la matriz (suma de los valores de la diagonal)

A^2 : multiplica A por A
A.^2 : eleva cada componente de A al cuadrado

A(2,3) : elemento de A en la fila 2, columna 3
A(:,2) : todas las filas, columna 2

AA = A([1,2] , [2,3]) : crea una nueva matriz con las filas 1 y 2 y las columnas 2 y 3 de la matriz A

A2 = [A ; [6 7 8 9]] : añade una fila fila nueva a A
A3 = [A [6 7 8 9]'] : añade una fila columna nueva a A

teniendo :
M=[1 3 5 7 9 11;2 3 4 5 6 7;2 4 6 8 10 12;1 0 1 0 1 0;1 1 1 1 1 1]
MM=[1 1 1;2 2 2;3 3 3;4 4 4;5 5 5]

M(:,7:9)=MM : une M a MM

size(A) : dimensión de la matriz A

[f c] = size(A) : le da nombre a los valores del vector que size(A)

A2 (3 , :) = [ ] : elimina la fila 3
A2 (: , 2) = [ ] : elimina la columna 2

A ([2 3 1] , :) : mueve de Fila 1 a Fila 2
Fila 2 Fila 3
Fila 3 Fila 1

A (: , [3 2 1]) : mueve de Columna 1 Columna 2 Columna 3
a Columna 3 Columna 2 Columna 1

eye(n) : te crea una matriz unidad con dimensión n x n

zeros(3,4) : crea una matriz de ceros con dimensión 3x4
ones(3,4) : crea una matriz de unos con dimensión 3x4

AA2 : [A2 A3] : junta dos matrices en una nueva

blkdiag (B,C) : teniendo dos matrices B=[1 1;1 1] y C=[2 2 2;2 2 2;2 2 2], las junta por la diagonal, es decir, nos daría:

1	1	0	0	0
1	1	0	0	0
0	0	2	2	2
0	0	2	2	2
0	0	2	2	2

max(A) : te dará un vector con los máximos de cada columna
max(max(A)) : máximo de toda la matriz

min(A) : te dará un vector con los mínimos de cada columna
min(min(A)) : mínimo de toda la matriz

prod(A) crea un vector con los productos de los elementos columna a columna

sum(A) : crea un vector con la suma de los elementos columna a columna

diag(A) : (siendo A una matriz) diagonal de la matriz A convertida en vector
diag(v) : (siendo v un vector) crea una matriz con el vector v como matriz

diag(v,k) : (siendo v un vector y k un entero) crea una matriz con diagonal v dejando k columnas con ceros antes de empezar con la diagonal

rand(n,m) : crea una matriz aleatoria de dimensión nxm

flipud(A) : ordena de forma inversa los índices de fila, pasando de F1 a F3
F2 F2
F3 F1

fliplr(A) : ordena de forma inversa los índices de columna, pasando de C1 C2 C3
a C3 C2 C1

sort(A) : ordena los valores de forma creciente por columnas (de arriba a abajo)
sort(A,'descend') : de forma descendiente
sort(A,2,'descend') : descendiente por filas
sort(A,1,'ascend') : ascendiente por columnas

tril(A) : devuelve la parte triangular inferior de la matriz
triu(A) : devuelve la parte triangular superior de la matriz
ambas dos haciendo ceros en el resto de la matriz

inv(A) : hace la inversa de la matriz

svd(A) : Vector V de valores singulares de A que son las raíces cuadradas de los autovalores de la matriz simétrica A'*A

Z = null(A) : da una base ortogonal del nucleo de A
Z = null(A,'r') : da una base ortogonal y racional

Q = orth(A) : nos da una base ortonormal para el rango de A

subspace(A,B) : nos da el ángulo formado entre los vectores de A y B columna a columna

dot(A,B) : producto escalar de dos matrices (se multiplican elemento a elemento y luego se suman los valores por columnas)

cross(A,B) Producto vectorial

magic(n) : nos proporciona una matriz nxn construida por los enteros positivos de 1 a n^2, en ella cada columna y cada fila (al igual que la diagonal) suman lo mismo

pascal(n) : nos proporciona una matriz de orden n simétrica cuyas elementos son los números enteros del triángulo de Pascal. Su inversa tiene elementos enteros

A/B == A*inv(B)
A\B == inv(A)*B

rref(A) : reduce la matriz a la forma escalonada por el método de eliminación de Gauss Jordan (haciendo ceros por debajo de la diagonal). Esta función es muy útil para resolver sistemas de ecuaciones ya que p.e. creando una matriz A acumulada (matriz_variables y matriz_resultados), al aplicar el comando rref(A), en la esquina derecha nos saldrá el valor de una de las variables, y con este podremos hallar el resto.

ejemplo de cambio y operación de filas :

Hallar la matriz R obtenida a partir de C realizando las operaciones elementales :
fila1 <—> fila3 : C = C([3,2,1],:)
fila2 —> fila2 + 6*fila1 : C(2,:) = C(2,:) + 6*C(1,:)
fila3 —> fila3 - 2*fila2 : C(3,:) = C(3,:) - 2*C(2,:)
fila3 —> 11*fila3 + 2*fila2 : C(3,:) = 11*C(3,:) + 2*C(2,:)

poly(A) : hallamos el polinomio característico de la matriz A (pA(λ)=det(A-λ*I))
ejemplo :

Supongamos que queremos encontrar el polinomio característico de la matriz A = [2 1,-1 0]. Debemos calcular el determinante de A - t*I, el cual es (t-1)^2. Siendo el polinomio característico de A = [1 -2 1]

jordan(A) : halla la matriz canónica de Jordan de la matriz A
[P,J]=jordan(A) : halla la matriz canónica de Jordan de la matriz A y la matriz de paso P cuyas columnas son los autovalores de cumpliéndose inv(P) * A * P = J

eig(A) : nos da los autovalores de A
[P,D]=eig(A) Halla la matriz diagonal D de autovalores de A y una matriz P cuyas columnas son los autovectores correspondientes. Cumple A*P=P*D

¡¡¡Recuerda: AUTOVALORES!!!

Sea A matriz cuadrada nxn, λ es autovalor real si existe v vector no nulo tal que A*v= λ*v (equivalentemente, (A-λ*I)v=0, siendo I la matriz identidad nxn).
Por tanto debe cumplir det(A-λ*I)=0

¡¡¡Recuerda: ORTOGONAL!!!

Una matriz Q de nxn es ortogonal si sus columnas forman una base ortonormal de R^n.

Sea una matriz Q de nxn. Entonces:

- **Q es ortogonal si y solo si traspose(Q)*Q=I**
- **Q es ortogonal si y solo si inv(Q)=traspose(Q) (es decir la inversa coincide con la traspuesta)**
- **Si Q1 y Q2 son ortogonales entonces Q1*Q2 es ortogonal**

CADENAS DE CARACTERES

char('var1','var2','var3') : crea una cadena de caracteres en forma de matriz

BUCLES

- FOR :

ejemplo :

```
suma = 0;                  : declaras variables que vayas a utilizar dentro
for i = 1:10               : crea una i de 1 a 10
        suma = suma + i    : aquí dentro escribes las operaciones
end                        : sentencias el bucle con un ' end '
```

- WHILE :

ejemplo :

```
while a < 5                : mientras se cumpla esta condición…
        a = a + 1          : …se van a repetir estas operaciones
end                        : sentencias el bucle con un ' end '
```

- IF :

ejemplo :

```
x=input('Dame un numero: ');
if x>0                          : primer if con su condición…
   disp('Es positivo')          : …y lo que ocurre
elseif x==0                     : …además si ocurre esto…
   disp('Es nulo')              : …que salga esto…
else                            : …el resto…
   disp('Es negativo')          : …esto
end                             : y se acabó
```

- SWITCH : (este es un caso particular del bucle IF)

ejemplo :

```
metodo = 'bis'                  : creamos la variable sobre la que vamos a trabajar
switch metodo                   : switch nombre_de_la_variable
    case 'bis'                  : en el caso de que metodo='bis'…
       disp('Biseccion')        : …sacamos 'Biseccion'
    case 'rgf'                  : en el caso de que metodo='rgf'…
       disp('Regula Falsi')     : …sacamos 'Regula Falsi'
    case 'new'                  : en el caso de que metodo='new'…
       disp('Newton')           : …sacamos 'Newton'
    otherwise                   : en otro caso…
       disp('Ni idea')          : …sacamos 'Ni idea'
end                             : sentenciamos el bucle con un ' end '
```

Algoritmo de la burbuja : Ordenar un vector :

```
v=input('Introduce un vector:')          : lo introducimos modo [1 2 3 4]
n= length(v);
v0=v;
cont=0;
cambios=0;

for k=1:n-1
        for j=1:n-k
                cont=cont+1;
                if v0(j) > v0(j+1)
                        aux=v0(j);
                        v0(j)=v0(j+1);
                        v0(j+1)=aux;
                        cambios=cambios+1;
                end
                cambios;
        end
        fprintf('En la iteración ,k= %i, solo se necesitan %i cambios. Pero ....\n,'k, cambios)
        fprintf('Con la iteración ,k= %i, se estudian ,%i, posibles cambios y se obtiene el vector de
                componentes\n ', k, cont)
        disp(v0)
end
```

POLINOMIOS (VARIABLES SIMBÓLICAS)

syms x y z : convierte en simbólicas las variables x y z (si hubiera solo una variable sería 'sym')

ejemplo : f1=3*x+3*y-2*z; f2=-2*x+4*y-z; g=f1-f2 : resuelve la función g tomando x y z como variables que no conocemos

y=sym('y') : convierte la variable en simbólica

pretty(S) : pone bonito un polinomio S

double(S) : hace que sea un número con decimales

Suma de polinomios :

```
P1=[3 15 0 -10 -3 15 -40]; P2=[3 0 -2 -6];
P1=poly2sym(P1); P2=poly2sym(P2);
Suma=P1+P2
```

Producto de polinomios :

```
P1=[3 15 0 -10 -3 15 -40]; P2=[3 0 -2 -6];
poly2sym(conv(P1,P2))
```

División de polinomios :

```
u=[2 9 7 -6];v=[1 3];
[q r]=deconv(u,v)
```

Derivada de un polinomio :

```
f1=[3 -2 4], f2=[1 0 5]
k=polyder(f1)              : derivada del polinomio f1 (tiene que estar en modo vector)
k=polyder(f1,f2)           : derivada del producto de los polinomios
[n,d]=polyder(f1,f2)       : derivada de la división de los polinomios
```

Curva de ajuste mediante polinomios :

polyfit(x,y,n) : se utiliza para calcular polinomios de ajuste sobre n puntos
- x vector con las variables independientes
- y vector con las variables dependientes
- n es el grado el polinomio de ajuste deseado

ejemplo :

```
x=[0.9 1.5 3 4 6 9 9.5];
y=[0.9 1.5 2.5 5.1 4.5 4.9 6.3];
p=polyfit(x,y,3)           : de grado 3
```

ejemplo gráfica de la curva de ajuste :

```
x=[0.9 1.5 3 4 6 9 9.5];
y=[0.9 1.5 2.5 5.1 4.5 4.9 6.3];        : 1- declaramos los puntos
p1=polyfit(x,y,1)                       : 2- creamos el polinomio de ajuste en grado n (1)
                                           sustituyendo la x en este polinomio te dará el valor
                                           estimado de la y
p2=polyfit(x,y,2)
p3=polyfit(x,y,3)
p4=polyfit(x,y,4)
p5=polyfit(x,y,5)
p6=polyfit(x,y,6)
xp=0.9:0.1:9.5;                         : creamos un intervalo de min(x) a max(x) más preciso
```

```
yp1=polyval(p1,xp);                    : creamos el polyval entre el polyfit y el xp
yp2=polyval(p2,xp);
yp3=polyval(p3,xp);
yp4=polyval(p4,xp);
yp5=polyval(p5,xp);
yp6=polyval(p6,xp);
plot(x,y,'or',xp,yp1,':k')             : representamos las gráfica
hold on
plot(xp,yp2,'c')
hold on
plot(xp,yp3,':r')
hold on
plot(xp,yp4,'y')
hold on
plot(xp,yp5,':b')
hold on
plot(xp,yp6,'g')
hold off
xlabel('x');
ylabel('y');
legend('P1','P2','P3','P4','P5','P6');
title('POLINOMIOS DE AJUSTE DE DATOS');
```

polyval(p,xp) : valores del polinomio p en cada punto xp

POLINOMIOS (FUNCIONES SIMBÓLICAS)

- Se pueden crear gráficas de funciones simbólicas solo con el comando ezplot

Comandos subs() y solve() :

subs() :

```
syms x a b c                  : teniendo unas variables simbólicas x a b c…
f=a*x^3+b*x^2+c*x+sin(x)      : …con una función f…
subs(f,x,2)                   : …aplica la función f en el punto x=2
g=x^2-4*x-5;
```

solve() :

```
syms x a b c                  : teniendo unas variables simbólicas x a b c…
h=a*x^2+b*x+c;                : …con una función h…
solve(h)                      : …despeja la x
solve(h,a)                    : …despeja la a
```

Derivada de una función :

```
syms x a b c                  : teniendo unas variables simbólicas x a b c…
f=a*x^3+b*x^2+c*x+sin(x)      : …con una función f
diff(f),                      : primera derivada
diff(f,1),                    : primera derivada
```

```
diff(diff(f)),                  : segunda derivada
diff(f,2),                      : segunda derivada
diff(f,3)                       : tercera derivada
```

Integral de una función :

```
int(f)                          : integral en base a la primera variable
int(f,x)                        : integral en base a x
int(f,a)                        : integral en base a a
```

_____

```
syms x
g=2*x^3+7*x^2+9*x+sin(x)
int(g,x)
int(g,x,0,3)                    : integral de g en base a x desde 0 hasta 3
```

Composición de funciones : (g ∘ f) (x) = f(g(x)) :

```
f(x)=sin(cos(x^(1/2)));
g(x)=sqrt(tan(x^2));
compose(f,g),                   : f ∘ g
compose(g,f)                    : g ∘ f
```

Comandos factor(), expand() y simplify() :

```
syms x ,
f=(x^4-1)*(x^4-4),
g=((x^4 - 4*x^6)*x^2 - x^3+ 4*x^2),
factor(f),                      : te da los factores de la función
expand(f),                      : te lo opera (alargándolo)
simplify(g)                     : te lo acorta
```

FUNCIONES

ejemplo :

```
                                : en un script escribiremos esta información
function[ m d ] = dame(x)       : escribimos :
                                function[ lo que devolvemos ] = nombre ( variable que damos )
m = x+1
d = x+2                         : mediante operaciones hallamos lo que devolveremos
```

f = inline('x^3+x-1') : mediante esto creamos una función que metiendo un valor para x te sacará el resultado de la función que asignemos

f = inline('x^3+y','x','y') : si tiene dos variables o más habrá que meterlas en el inline tras la función, y ese será el orden con el que las meteremos en la f()

*¡¡¡ no se puede hacer un fplot para una función creada con inline (un ez****sí) !!!*

Si x es una variable vectorial :

f = inline(sym('x^2')) o f = inline('x.^2')

GRÁFICAS

- REPRESENTACION 2-D

Puntos :

x=1:10; y=[3.2 5.4 4.3 6 7.1 8 7.5 6.9 7.3 8.2]; : declaras los puntos
plot(x,y,'*r') : crea una gráfica con las coordenadas x e y dibujando sobre ellas asteriscos rojos, si no ponemos nada saldrá una linea negra uniendo los puntos de x e y
plot(x,y,'g:d','LineWidth',2,'markersize',9,'markeredgecolor','r','markerfacecolor','b')
  : plot(x,y,'especificadores de linea','propiedades' , 'valores')
line(x,y,'LineStyle','--','color','r','marker','o')
  : line(x,y,'propiedades' , 'valores')

teniendo :
x=[-2:0.01:4]; : puntos
y=3*x.^3-26*x+6; : función
yd=9*x.^2-26; : primera derivada
ydd=18*x; : segunda derivada

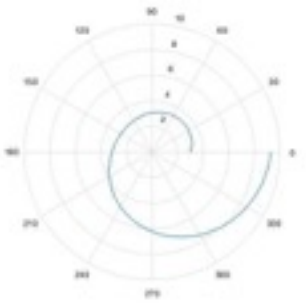
mediante plot :
plot(x,y,'-b',x,yd,'--r',x,ydd,':k')

mediante line :
plot(x,y,'LineStyle','-','color','g') : '-' hace lineas continuas
line(x,yd,'LineStyle','--','color','c') : '- -' hace lineas discontinuas
line(x,ydd,'LineStyle',':','color','r') : ':' hace lineas con puntos

Curva :

- Anónima
  f1=@(x) cos(2*x)-3;
  fplot(f1, [3,7], 'g') : crea una gráfica de la función (en este caso cos(2*x)-3) que escribamos en el intervalo, con unos límites [3,7]. color = verde, color por defecto = azul. Hay que obligatoriamente determinar unos límites
  fplot(@(x) cos(x+1),[-pi,pi])
- Implícita
  ezplot('x^2+y^2-4',[-2,2]) : se puede o no especificar limites
    - para cambiar color con ezplot hay que dar dos pasos:
    dibujo1=ezplot('sin(x)',[-pi,pi]) : primero le damos nombre y lo pinta en azul
    set(dibujo1,'color','magenta') : luego en el color que digamos
- Paramétrica
  ezplot('cos(t)','sin(t)',[0,3*pi/2])
- **Con coordenadas polares**
  t=linspace(0,2*pi,200); : 200 puntos equidistantes
  r=3*cos(0.5*t).^2+t; : función
  polar(t,r) : gráfica —>

  ezpolar('sin(3*t)*cos(t)',[0,pi]) : por defecto pone el



intervalo [0,2*pi]

- REPRESENTACION 3-D

Puntos :

```
x=1:6;
y=[3.2 5.4 4.3 6.5 7.1 8.2];
z=[7.5 6.9 7.3 8.2 4.8 7.9];
plot3(x,y,z,'+k')
grid on          : añade cuadrícula
grid off         : quita cuadrícula
```

Curva :

```
ezplot3('sin(t)','cos(t)','t',[0,6*pi],'animate')     : animación opcional
- Si quiero decidir yo cuantos puntos pintar :
t=0:pi/20:pi;              : declaras un vector con los puntos que quieras pintar
plot3(sin(t),cos(t),t,'og')
```

Superficie :

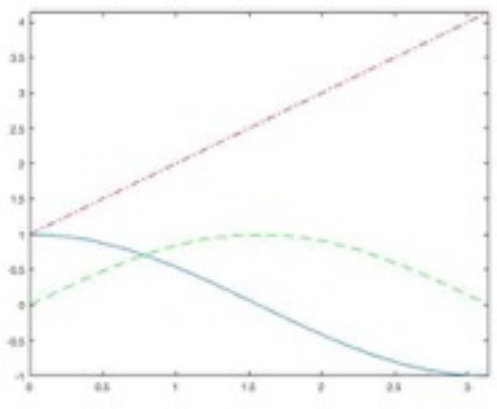
- Explícita
  ```
  ezsurf('x^2+y^2')      : expresion=z despejada;se pueden o no especificar limites
  ezmesh('x^2+y^2')      : igual que ezsurf pero con un tono más claro
  ezsurfc('x^2+y^2')     : curvas de nivel o isocontornos
  ```
- Paramétrica
  ```
  ezsurf('r*cos(t)','r*sin(t)','r',[0, 5, pi, 2*pi])     : [rmin, rmax, tmin, tmax]
  ```

  - Con curvas de nivel sobre una superficie :

    ejemplos :
    ```
    ezmeshc('x*exp(-x^2-y^2)',[-pi,pi,-pi,pi])
    ezsurfc('x*exp(-x^2-y^2)',[-pi,pi,-pi,pi])
    ```
  - Solo curvas de nivel :
    ```
    ezcontour('x*exp(-x^2-y^2)',[-pi,pi,-pi,pi])
    ```

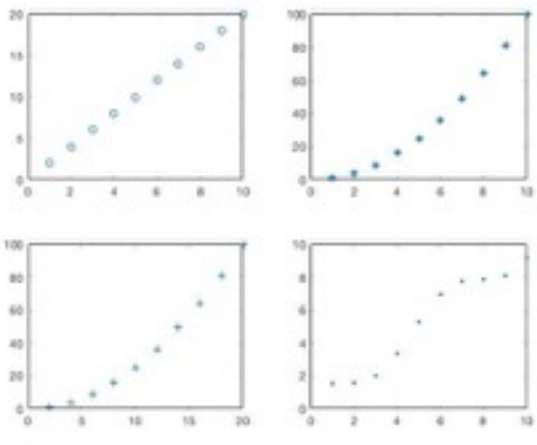
Para juntar curvas en un mismo dibujo

```
fplot('cos(x)',[0 pi])
hold on
fplot('sin(x)',[0 pi],'--g')
fplot('x+1',[0 pi],'-.r')
hold off
```



Para juntar dibujos en una misma página

```
x=1:10;y=2*x;z=x.^2;t=cos(x)+x;
subplot(2,2,1); plot(x,y,'o')
subplot(2,2,2); plot(x,z,'*')
subplot(2,2,3); plot(y,z,'+')
subplot(2,2,4); plot(x,t,'.')
```

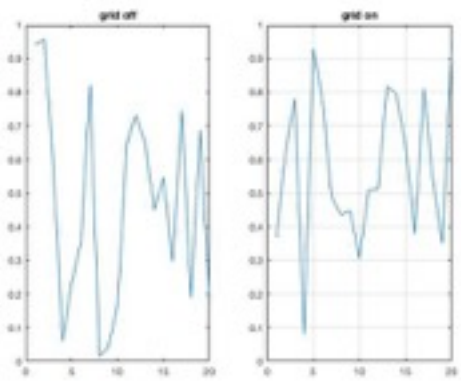


pasos:
1. dibujamos la primera gráfica
2. escribimos : hold on
3. dibujamos el resto de gráficas
4. escribimos : hold off

pasos:
1. declaramos las funciones a representar
2. escribimos subplot(n,m,p) antes de cada gráfica siendo nxm la dimensión de la matriz de gráficas y la p la posición

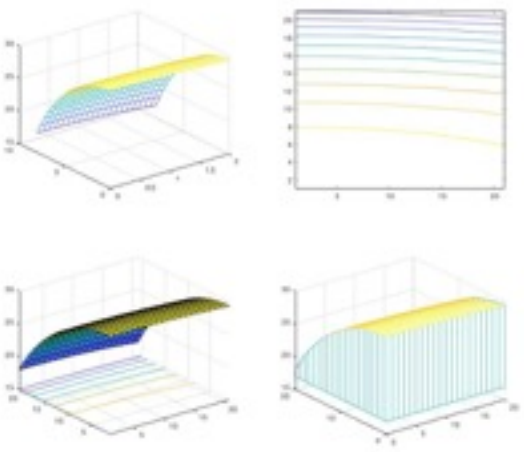
Título :

```
figure                          : no es obligatorio
x=1:20;
y=rand(1,length(x));            : declaramos las variables…
subplot(1,2,1);                 : …decimos donde la queremos,…
plot(x,y)                       : …la ponemos y…
title('grid off')               : …le damos un título…
y=rand(1,length(x));            : …and again
subplot(1,2,2);plot(x,y)
grid on
title('grid on')
```



Crear un mallado :

```
x=0:0.1:2;
y=4*x;
[X,Y]=meshgrid(x,y);
Z=3*sqrt(100-X.^2-Y.^2);
subplot(2,2,1);mesh(X,Y,Z)
subplot(2,2,2);contour(Z,10)
subplot(2,2,3);surfc(Z)
subplot(2,2,4);meshz(Z)
```



Función que saca unas determinadas lineas de nivel :
(en un script a parte con nombre igual que la función)

```
function [ x,y,z ] =lineas_nivel
x1=linspace(-3,3,15);
y1=linspace(-3,13,17);
[x,y]=meshgrid(x1,y1);
z=x.^4+3*x.^2-2*x+6-2*y.^2-2*y;
subplot(1,2,1);contourf(x,y,z)
subplot(1,2,2);contour3(x,y,z)
end
```

Colocación de texto en la gráfica :

```
fplot('sin(x)',[0 2*pi])
gtext('sen(x)')          : mediante el puntero seleccionamos donde queremos el texto
```

Nombre de los ejes :

```
xlabel('EJE X');
ylabel('EJE Y');
```

Leyenda :

legend('funcion','1a derivada','2a derivada') : en el orden en el que has asignado las curvas anteriormente

SERIES NUMÉRICAS

Si la serie es convergente (tiende a un punto) : limit(an,n,Inf) = 0
Si la serie es divergente (tiende a ∞) : limit(an,n,Inf) = 1

```
syms n;
an=(n+3)/(n-2);
L=limit(an,n,Inf)
```

symsum(S,k,a,b) : suma los componentes de la serie S en base a la variable k desde a hasta b
ejemplo :

```
syms k;
S=symsum(3^k/factorial(k),k,0,inf)
```
: si solo hay una variable simbólica no hace falta que la escribamos en el comando : (symsum(3^k/factorial(k),0,inf)

Si te pidieran analizar la convergencia de una serie $\sum(n/2^n)$ $n\in(0,1,2,\ldots,n)$:

a) Analizar la convergencia de la serie symsum(n/2^n,n,0,inf)
b) Si es convergente, calcula la suma de los k primeros números

a)

```
syms n
f=n/2^n;
limit(n/2^n,n,inf)
```
: da 0, es decir, es convergente (tiende a un punto)

```
L=limit(subs(f,n+1)/subs(f,n),inf)
```
: aplicas el Criterio del cociente (F_{n+1} / F_n), si esta es mayor que 1, la serie diverge, en este caso en 1/2 así que converge

b)

```
syms n k;
Sparcialk=symsum(n/2^n,0,k-1);
```
: creas un comando que signifique la suma de los k primeros sumandos (k - 1 y el 0)

```
S2=limit(Sparcialk,k,inf)
```
: buscas el límite del anterior comando, lo cual te dará la suma total (=2)

```
S1=symsum(n/2^n,0,inf)
```
: suma del total (de 0 a Inf) (=2)
: S1 == S2

Criterios de convergencia :

Criterio de Raabe :

symsum(f,n,1,inf) es convergente si limit(n*(1-(subs(f,n+1)/subs(f,n)))) > 1
es divergente si limit(n*(1-(subs(f,n+1)/subs(f,n)))) < 1

En el caso de que sea 1 no se podrá asegurar nada a cerca del sentido de la convergencia de la serie

Criterio de la raiz :

symsum(f,n,1,inf) es convergente si limit(f^(1/n),inf) < 1
es divergente si limit(f^(1/n),inf) > 1

- Series alternadas :

Una serie numérica alternada es aquella que tiene por término general a :

ejemplo :
bn = (-1)^n*an : siendo an>0

Criterio de Leibniz :

Sea la serie alternada bn cumple:
- limit(bn,inf) = 0
- l b(n+1) l < l b(n) l , es decir, para todo n natural, bn es decreciente, es decir, es convergente

- Series de términos arbitarios

Criterio de Dirichlet :

Consideremos la serie de término general an*bn y la de término general an. Si las sumas parciales de la serie de términos an está acotada y la sucesión {bn} es decreciente con limit(bn,n,inf)==0 entonces la serie de término general an*bn es convergente (estarás multiplicando a an cada vez por números más pequeños).

Criterio de Abel :

Si la serie de término general an es convergente y la sucesión {bn} es monotona y acotada, entonces la serie de término general a(n)*b(n) es convergente.

Diferencia entre sum y symsum a la hora de hallar las sumas parciales :
ejemplo :

```
syms k;

S_symsum = symsum(1/k^2, k, 1, 10);
AproxS_symsum=double(S_symsum)

S_sum = sum(subs(1/k^2, k, 1:10));
AproxS_sum=double(S_sum)
```

A la hora de hacer una suma el tiempo de ejecución es diferente según el comando que utilizas:

```
syms k
tic                                          : con tic empiezas a contar
S_sum = sum(subs(k^2, k, 1:100000))
toc                                          : con toc acabas de contar
tic
S_sumvector =sum(sym(1:100000).^2)
toc
tic
S_symsum = symsum(k^2, k, 1, 100000)
toc

S_sum = 333338333350000
Elapsed time is 8.169955 seconds.

S_sumvector = 333338333350000
Elapsed time is 2.655626 seconds.

S_symsum = 333338333350000
Elapsed time is 0.111731 seconds.            : la forma más rápida es la de symsum
```

SERIES DE POTENCIAS

Los criterios para hallar la convergencia son los mismos que hemos utilizamos anteriormente con las series numéricas.

Intervalo de convergencia : fn = an*bn

```
syms x n
fn=(4^(2*n)/(n+2))*(x-3)^n
an=(4^(2*n)/(n+2))

RadioConvg=limit(simplify(an/subs(an,n,n+1)),n,Inf)
L=limit(simplify(subs(fn,n,n+1)/subs(fn,n,n)),n,inf)
```

La serie será convergente cuando |16*x - 48| < 1

Para calcular los extremos del intervalo de convergencia de la serie resolvemos las siguientes ecuaciones:

```
extremos=[solve('16*x - 48=-1'), solve('16*x - 48=1')]
extremos = [47/16, 49/16]                    : No es el intervalo de convergencia, únicamente
                                               podemos asegurar por tanto que para todo x que
                                               cumple 47/16 < x < 49/16 la serie es convergente.
                                               Falta por estudiar el comportamiento de la serie en
                                               los extremos del intervalo de convergencia
```

Estudio de la serie cuando x=49/16 :

```
fnu=simplify(subs(fn,x,49/16))
```

fnu=
1/(n + 2)

- Se estudia la serie numérica de términos positivos que tiene por término general an=1/(n+2) :

Lcociente=limit(simplify(subs(fnu,n+1)/subs(fnu,n)),n,inf)
Lraabe=limit(simplify(n*(1-(subs(fnp,n+1)/subs(fnu,n)))),n,inf)

- Observamos que tanto el criterio del cociente como el de Raabe dan limite 1 y por tanto debe utilizarse un criterio alternativo. Aplicamos entonces el criterio de comparación de segunda especie, comparando la serie del problema con la serie armónica divergente 1/n :

LCC2=limit(subs(fnu,n)/(1/n),Inf)

- (=1) Como el limite es finito y mayor que 0. La serie diverge para x=49/16

Estudio de la serie cuando x=47/16 :

fnl=simplify(subs(fn,x,47/16))

- Se obtiene una serie alternada de término general an=(-1)^n/(n + 2). Como esta serie tiene sus sumas parciales acotadas ,el término general 1/(n + 2) es decreciente y converge a cero por el criterio de Dirichlet, es decir, la serie alternada converge. Su suma es:

Suma_fnl=symsum(fnl,1,Inf) : =1/2 - log(2)

El intervalo de convergencia es por tanto [47/16,49/16).

taylor(f) : realiza el desarrollo de McLaurin de quinto grado para la función f

$$f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f^{(3)}(a)}{3!}(x-a)^3 + \cdots$$

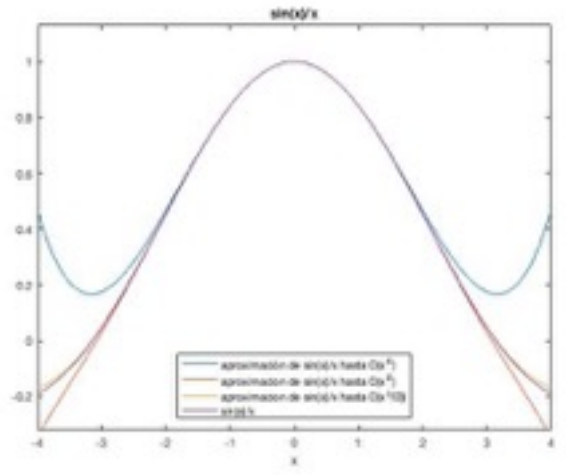
taylor(f,x,'ExpansionPoint',a) : realiza el desarrollo de Taylor de quinto grado para la función f centrado en x=a

taylor(f,x,'Order',n) : realiza el desarrollo de Taylor de orden grado n-1

ejemplo : aproximamos sin(x)/x

```
syms x
f = sin(x)/x,
t6 = taylor(f, 'Order',6)
t8 = taylor(f, 'Order', 8)
t10 = taylor(f, 'Order', 10)
dibujo=ezplot(t6, [-4, 4])
hold on
```

```
dibujo1=ezplot(t8, [-4, 4]);
dibujo2=ezplot(t10, [-4, 4]);
dibujo3=ezplot(f, [-4, 4]);
legend('aproximación de sin(x)/x hasta
O(x^6)','aproximacion de sin(x)/x hasta
O(x^8)','aproximacion de sin(x)/x hasta
O(x^(10))','sin(x)/x','Location','South');
hold off
```



taylortool(f) : este comando es la serie de taylor de la función f de orden 18, la cual coincide con el desarrollo de la función

ESTUDIO DE FUNCIONES

Recordatorio :

fplot('sqrt(x^2-4)') : como anónima
: ó

```
f1=@(x) sqrt(x^2-9);
fplot(f1,[3,7],'g')
```

fplot('curvaejemplo',[0,pi]) : como una función en un fichero a parte

subs(f,x,pi/2)
eval('sin',pi/2) : evalúan la función en un punto (pi/2 en este caso)

Límites de funciones :

limit(función,x,a) : Calcula el límite de una función, indicada por su expresión analítica, cuando x tiende hacia el valor de a

limit(función,x,a,'right') : Calcula el límite de una función, indicada por su expresión analítica, cuando x tiende hacia el valor de a por la derecha

limit(función,x,a,'left') : Calcula el límite de una función, indicada por su expresión analítica, cuando x tiende hacia el valor de a por la izquierda

Derivadas e integrales de funciones :

```
syms x a b c;
f=a*x^3+b*x^2+c*x+sin(x);
g=3*cos(5*x);
```

```
primera = diff(f)                    : diff(f) == diff(f,1)
segunda = diff(diff(f))              : diff(diff(f)) == diff(f,2)
tercera = diff(diff(diff(f)))        : diff(diff(diff(f))) == diff(f,3)

int(f)                               : integral de f en base a la primera variable
int(f,0,3)                           : integral de f en base a la primera variable entre 0 y 3
int(f,a)                             : integral de f en base a a
```

En polinomios :

```
k=polyder(p)                         : derivada del polinomio
k=polyder(a,b)                       : derivada del producto de los polinomios
[n d]=polyder(u,v)                   : derivada del cociente de los polinomios
```

——————————————— recordar los comandos “ eval ”, “ feval ”, “ subs ” y “ sign ” ———————————————

Ejemplo de estudio de una función : f(x)=(x^3)/(x^2-1)

1. Función

```
syms x
f=(x^3)/(x^2-1)
```

2. Dominio

```
denominador0=solve(x^2-1)            : este comando iguala la ecuación a 0 y
                                       saca los posibles resultados, estos los
                                       eliminaremos del dominio
```

Dominio=R-(denominador0) (R-{-1,1})

3. Simetría

```
subs(f,x,-x)==f                      : si se cumple tiene simetría par
subs(f,x,-x)==-f                     : si se cumple tiene simetría impar
```

La función f tiene simetría impar

4. Cortes con los ejes

```
subs(f,x,0)                          : resolvemos para x = 0
solve(f)                             : resolvemos para y = 0
```

El único punto de corte con los ejes será en el (0,0)

5. Asíntota horizontal

limit(f,x,inf) : buscamos el límite de f en inf, si nos da un número, esa será la asíntota horizontal

limit(f,x,-inf) : buscamos el límite de f en -inf, si nos da un número, esa será la asíntota horizontal

No hay asíntota horizontal

6. Asíntota vertical

: sabiendo los valores en los que podría haber una asíntota, hacemos límite por la derecha y por la izquierda de ambos, si en ambos sale +inf ó -inf, es decir, tiende hacia un punto inexistente, habrá asíntota vertical en ese punto

limit(f,x,denominador0(1),'right') : = inf, es asíntota vertical por la derecha
limit(f,x,denominador0(1),'left') : = -inf, es asíntota vertical por la izquierda
limit(f,x,denominador0(2),'right') : = inf, es asíntota vertical por la derecha
limit(f,x,denominador0(2),'left') : = -inf, es asíntota vertical por la izquierda

Hay dos asíntotas verticales, una en x=-1 y otra en x=1

7. Asíntota oblicua y=x*m+n

: sabiendo que toda asíntota oblicua cumple que y=x*m+n, hemos de hallar m y n, siendo :

m=limit(f/x,x,inf)
n=limit(f-a*x,x,inf)

: estos puntos los sustituiremos en y=x*m+n y de ahí sacaremos la asíntota oblicua

solve(f-x) : resolvemos la asíntota (y=x → y-x=0), si sale algún valor de x entero, ese punto existirá en la asíntota

Hay una asíntota oblicua en y=x (siendo (0,0) el único punto existente en esta

8. Puntos críticos

deriv1=simplify(diff(f)) : lo primero que hacemos es derivar la función
criticos=solve(deriv1) : tras esto la resolvemos (igualándola a 0 con el comando solve())

: mediante la función double() te da números decimales y, sustituimos estos en la función para hallar los puntos críticos

Los puntos críticos son (0,0) (1.7321,2.5981) (-1.7321,-2.5981)

9. Estudio del crecimiento de una función

: recordar ”sign" : ans= 1 → es positiva
ans= -1→ es negativa
ans= 0 → es nula
: estudiaremos los intervalos que delimitan los puntos críticos :

: 0 < 0.5 < 1
sign(subs(deriv1,0.5)) : = -1 => f es decreciente en (0,1)
: 1 < 1.5 < 1.73
sign(subs(deriv1,1.5)) : = -1 => f es decreciente en (1,3^(1/2))
: 1.73 < 2 < Inf
sign(subs(deriv1,2)) : = 1 => f es creciente en (3^(1/2),inf)

: Por ser impar se conoce el crecimiento en la parte negativa
f es decreciente en (-1,0)
f es decreciente en (-1.7321,-1)
f es creciente en (-inf, -1.7321)

10. Máximos y mínimos

deriv2=diff(f,2) : hacemos la segunda derivada…
D=simplify(deriv2) : …la simplificamos…
subs(D,x,criticos) : …y hallamos los puntos críticos en ella
: nos dará 4 puntos, ahora hay que estudiarlos sabiendo que si son < 0 será un máximo y si son > 0 será un mínimo:
(3*3^(1/2))/2 > 0
-(3*3^(1/2))/2 < 0

En x=3^(1/2) f tiene un mínimo
En x=-3^(1/2) f tiene un máximo

11. Concavidad y puntos de inflexión

: teniendo los máximos y mínimos estudiamos el punto anterior y siguiente a estos, si el anterior decrece y el siguiente crece es cóncava, si es al revés será convexa

min1=3^(1/2)
max1=-3^(1/2)
sign(subs(deriv1,min1-0.1)) : decrece
sign(subs(deriv1,min1+0.1)) : crece
: = concava

sign(subs(deriv1,max1-0.1)) : crece
sign(subs(deriv1,max1+0.1)) : decrece
: = convexa

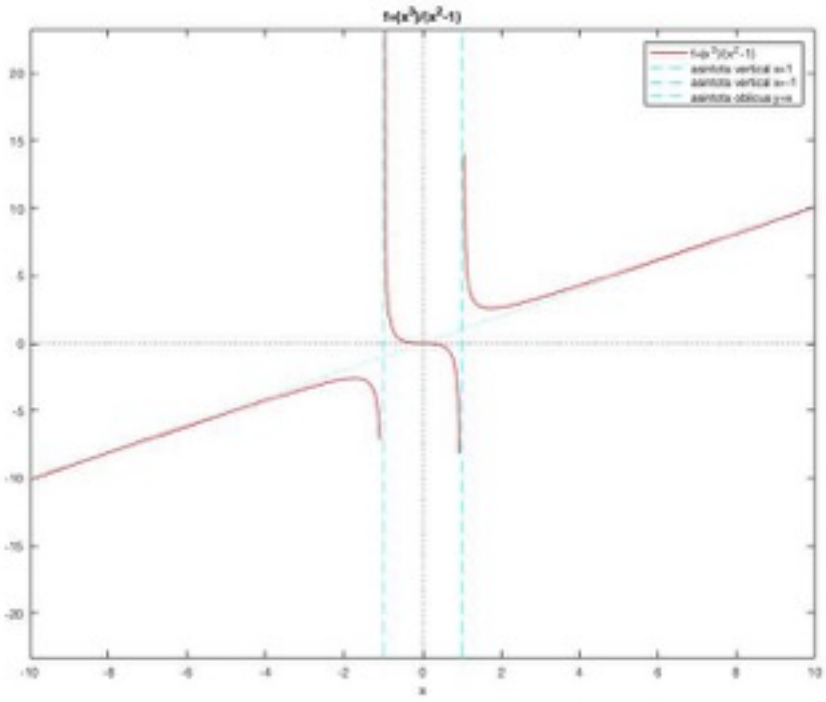
En el máximo es convexa y en el mínimo es concava

deriv2=diff(f,2)
solve(deriv2,x,0)

Hay un punto de inflexión en (0,0)

12. Gráfica

```
syms x
f=(x^3)/(x^2-1);
dibujo=ezplot(f,[-10,10]);
set(dibujo,'Color','r')
hold on
y=-25:0.5:25;
x=ones(length(y)),
plot(x,y,'--c');
y=-25:0.5:25;
x=(-1)*ones(length(y)),
plot(x,y,'--c');
x=-25:0.5:25;
y=x;
plot(x,y,':c');
x=-25:0.1:25;
y=zeros(length(x))
plot(x,y,':k');
y=-25:0.1:25;
x=zeros(length(y))
plot(x,y,':k');
hold off
title('f=(x^3)/(x^2-1)');
legend('f=(x^3)/(x^2-1)','asintota vertical x=1','asintota vertical x=-1','asintota oblicua y=x')
```



Hallar la ecuación de la recta tangente a la curva y=(x^2-4)^2 cuando x=1

```
syms x;
f=(x^2-4)^2
f1=subs(f,x,1)
derivf=diff(f)
derivf1=subs(derivf,x,1)

tangente = f1 + derivf1*(x-1)
```

DERIVADAS

RECORDATOTIO

k=polyder(f1)	: derivada del polinomio f1 (tiene que estar en modo vector)
k=polyder(f1,f2)	: derivada del producto de los polinomios
[n,d]=polyder(f1,f2)	: derivada de la división de los polinomios
diff(f)	: primera derivada de una función
diff(f,1)	: primera derivada de una función
diff(diff(f))	: segunda derivada de una función
diff(f,2)	: segunda derivada de una función
diff(f,3)	: tercera derivada de una función

Derivadas parciales :

gradient(f,[x,y]) : nos devolverá la derivada de f respecto a x y respecto a y

- Derivada parcial de f respecto de x : diff(f(x,y,z,..),x)

```
syms x y z;
fx=diff('x^3+y^2+z^3+2*x*y^2+3*x*z+4*x',x)%3*x^2 + 2*y^2 + 3*z + 4   : derivas respecto a x
```

- Derivada parcial enésima de f respecto a x : diff(f(x,y,z,..),x,n)

```
syms x y z;
fxx=diff('x^3+y^3+z^3+2*x*y^2+3*x*z+4*x',x,2)   : derivas respecto a x dos veces
fxxx=diff('x^3+y^3+z^3+2*x*y^2+3*x*z+4*x',x,3)   : derivas respecto a x tres veces
```

- Derivada parcial enésima de f respecto a y : diff(f(x,y,z,..),y,n)

```
syms x y z;
fyy=diff('x^3+y^3+z^3+2*x*y^2+3*x*z+4*x',y,2)
```

- Derivada parcial de f respecto a x e y sucesivamente (fxy) : diff(diff(f(x,y,z,…),x),y)

```
syms x y ;
fxy=diff(diff('x^3+y^3+z^3+2*x*y^2+3*x*z+4*x',x),y)
fxxxyy=diff(diff('x^4*y+x^3*y^3+z^3+2*x*y^2+3*x^2*y*z^3+4*x',x,3),y,2)
fxxyzz=diff(diff(diff('z^3*x^3*y^2+2*x*y^2+3*x^2*y*z^3+4*x',x,3),y),z,2)
```

EXTREMOS LOCALES

```
syms x y p
T=25+4*x^2-4*x*y+y^2      : función a optimizar
g=x^2+y^2-25              : condición
F=T+p*g                   : función de los extremos locales
[L X Y]=solve(gradient(F))
subs(T,{x,y},{X,Y})       : aquí hallamos los puntos en la función que hemos optimizado
```

Clasificación de puntos extremos de una función (f(x,y,z)=x^2+y^2+z^2+x*y):

```
syms x y z;
f=x^2+y^2+z^2+x*y

fx=diff(f,x)
fy=diff(f,y)
fz=diff(f,z)
[a b c]=solve(fx,fy,fz,x,y,z)
criticos=[a b c]
hess=hessian(f,[x y z])
detH=det(hess)
fxx=diff(f,x,2)                    : sustituimos a y b en ella
 : si l H l > 0, fxx(a,b) > 0, f alcanza mínimo relativo en (a,b)
 : si l H l > 0, fxx(a,b) < 0, f alcanza máximo relativo en (a,b)
 : si l H l < 0, (a,b) es punto de silla
 : si l H l = 0, el criterio no concluye el resultado
```

ÁLGEBRA

Resolver el sistema por una sustitución regresiva (o de regresión) :

```
A=[1 0 0 0;2 4 0 0;4 -2 -2 0;1 2 3 -3]
b=[-1 6 2 3]

n=length(b);
x=zeros(n,1);
x(1)=b(1)/A(1,1);
for i=2:n
    x(i)=(b(i)-A(i,1:i-1)*x(1:i-1))/A(i,i);
end
```

Triangularización por Gauss :

```
A=[1 0 0 0;2 4 0 0;4 -2 -2 0;1 2 3 -3]
b=[-1 6 2 3]
[n,n]=size(A)

for p=1:n-1
  for k=1:n-1
     a = Ab(k,p)/Ab(p,p);
     Ab(k,:) = Ab(k,:) - a*Ab(p,:)
  end
end
NAb=Ab
NA=Ab(1:n,1:n)
Nb=Ab(1:n,n+1)
```

Cramer :

```
x=zeros(n,1)
M=A
for i=1:n
  M(:,i) = b;
  x(i)=det(M)/det(A);
  M=A;
end
```

Ejemplo álgebra importante:
Hallar las coordenadas de v1, v2, v3, v4 respecto de B y la inversa de P=[1 -1 0 3;1 -1 0 1;1 -1 1 1;0 1 1 -1]' (todo a la vez) :

```
B=[1 -1 0 3;1 -1 0 1;1 -1 1 1;0 1 1 -1]';
v1=[1 -1 2 1]'; v2=[2 -1 1 1]'; v3=[1 0 3 -1]'; v4=[0 1 1 -1]';
Ab=[B v1 v2 v3 v4 eye(4)]
C=rref(Ab)

coordv1=C(:,5)
coordv1=C(:,5)
coordv1=C(:,5)
coordv1=C(:,5)
inversaP=C(:,[9 10 11 12])
```

Obtener la proyección ortogonal de un vector b :

```
A=input ('Sistema generador del S.V. en las columnas de A: ');
b=input ('Vector que vaya a proyectar: ');
Q=orth(A)
P=Q*Q'*b
```

1. sacamos autovalores con eig(M)
2. restamos M - diag(eig(M))
3. convertimos en sistema de ecuaciones
4. las soluciones son los autovectores