Preliminares: sobre cálculo diferencial, cálculo integral y álgebra.

Hoja de problemas 1 **+** ***Resumen de comandos***

Resumen de contenidos:

- **Variables, funciones y formatos**
- **Derivación e integración: una y varias variables**
- **Entornos gráficos: curvas y superficies**
- **Cálculos con matrices y vectores**

% Este cuaderno tiene un carácter introductorio a comandos Matlab que nos serán de utilidad a lo largo del curso para la computación simbólica y numérica en problemas de ecuaciones diferenciales. Seguimos el esquema de la hoja de problemas 1.

% Estas prácticas se hacen fundamentalmente con la versión de Matlab 2011 (e.g., R2011b): los resultados, en el contexto que nos movemos, pueden cambiar con respecto a los obtenidos con las versiones 2007-2008, y anteriores. No obstante, si no hay diferencia importante en el resultado, se puede utilizar cualquier versión a partir de Matlab 5.3, y a lo largo de los cuadernos de clase se irá comentando. Se ha procurado que el software proporcionado funcione (o funcione con mínimas modificaciones) con todas estas versiones de Matlab: ver funciones del curso en http://personales.unican.es/meperez/OCW_CSyNenED/

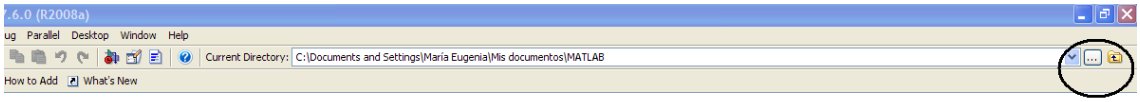
% Por ejemplo, ecuaciones diferenciales que se resolvían con versiones anteriores, Matlab 2011 no las resuelve, o las deja en términos integrales, o las resuelve en el campo complejo (sin explicación aparente). Por el contrario, hay otras ecuaciones diferenciales que versiones anteriores de Matlab no resolvían (o dejaban el resultado en una expresión larga y compleja de interpretar) y Matlab 2011 lo hace.

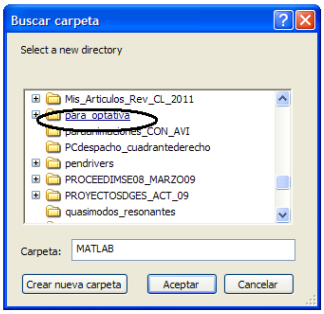
% Un resumen de comandos útiles se encuentra en el enlace de la cabecera. Empezamos desde un nivel cero, con comandos útiles para nuestro curso de ED, que sigue el libro de apuntes [Ecuaciones Diferenciales. Una introducción, ISBN 84-89627-28-2] o, para temas nuevos no tratados en el libro, las notas complementarias que se proporcionan (ver enlaces a capítulos o resúmenes en las cabeceras de los cuadernos). Observamos que, de manera general, distintos comandos Matlab nos pueden llevar a un mismo resultado.

% Al iniciar Matlab, el llamado "prompt" >> indica que el programa está esperando instrucciones. De manera general, comenzamos guardando la sesión de trabajo con diary

>> diary nombre

% Matlab nos guarda en un fichero "nombre" todo lo que va pasando en pantalla: instrucciones y respuestas de Matlab (salvo gráficas). Dicho fichero se puede editar. Para ejecutar los comandos de nuevo, e.g., se pueden copiar en pantalla. También puede resultar útil para esto el llamado "Command History" (en el desplegable "Desktop", en la cabecera). Por defecto nos guarda el fichero "nombre" en el directorio de trabajo (directorio "Work" en diversas versiones de Matlab); si queremos que nos lo guarde en otro sitio, o utilizar funciones que tengamos en un directorio, al empezar debemos indicarle el camino de dicho directorio: por ejemplo, pinchar en los puntos "…" de la barra de herramientas, y luego indicar la carpeta en la que queremos trabajar





% Nos guarda todo en este fichero hasta la instrucción

>> diary off

% En principio Matlab se puede utilizar como una calculadora para hacer las operaciones básicas de sumas, restas, etc., pero es "una calculadora potente", que tiene un lenguaje de programación propio, intuitivo desde el punto de vista matemático, y que nos permite crear nuestras funciones, guardarlas y utilizarlas en otra sesión. Asimismo, permite modificar algunas de sus funciones y adaptarlas a nuestras necesidades.

>> diary preliminares

% nos guarda todo lo que hagamos desde este momento hasta que ejecutemos "diary off" o nos salgamos de Matlab. En este caso el fichero se llama "preliminares" y está en el directorio en que estemos trabajando en la sesión en curso.

% Comenzamos distinguiendo entre variables simbólicas y numéricas

% Variables numéricas: asignado valores numéricos a variables

>> a=2

a =

2

>> b=3

b =

3

>> c=a+b

c =

5

>> a+b

% a b c son variables numéricas para Matlab (números para nosotros).
Si no almacenamos a+b en c, se almacena en otra variable numérica interna

>> a+b

ans =

5

% "ans" variable respuesta por defecto: "ans" es la variable respuesta que guarda el resultado de la última operación

% Para definir funciones del tipo de las que se utilizan en Cálculo o en Ecuaciones Diferenciales hay que introducir las llamadas variables simbólicas, por ejemplo, la variable simbólica t

>> syms t

% nos permite crear funciones ("variables simbólicas") utilizando otras funciones internas que tiene Matlab como senos, cosenos, exponenciales, …

>> f=2*sin(t)

f =

2*sin(t)

>> g=t^2+exp(-t)

g =

1/exp(t) + t^2

>> h=f+g

h =

1/exp(t) + 2*sin(t) + t^2

>> f+g

ans =

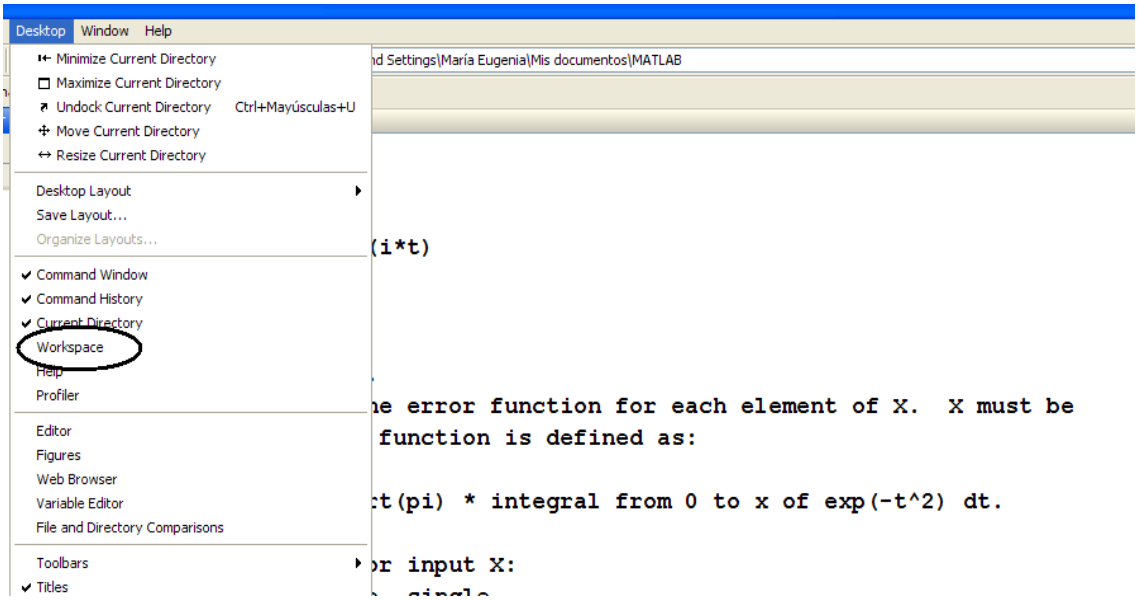
1/exp(t) + 2*sin(t) + t^2

% f, g, h, ans son ahora funciones de la variable t: variables simbólicas para Matlab; como antes "ans" es la variable respuesta por defecto

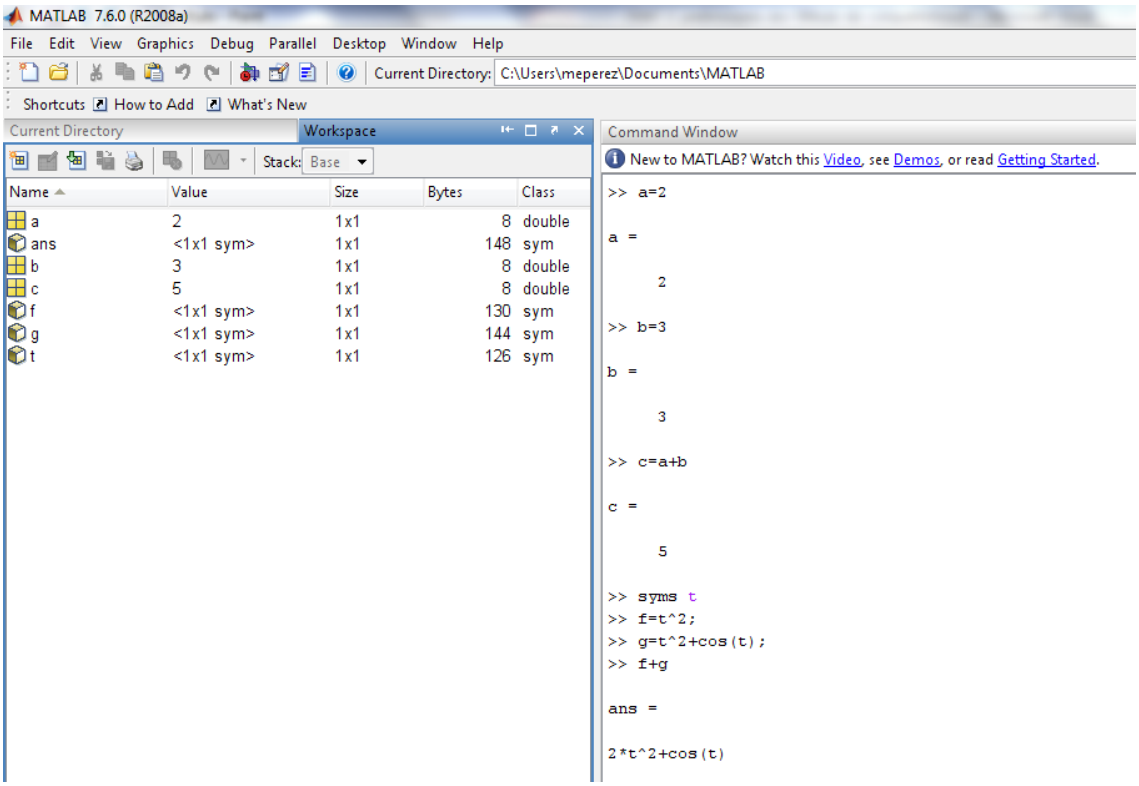
% "ans" es la variable respuesta que pasa de número a función dependiendo del resultado de la operación anterior (que no hemos guardado en una variable con un nombre). Operaciones con números dan "ans" variable numérica; operaciones con funciones o variables simbólicas dan "ans" variable simbólica.

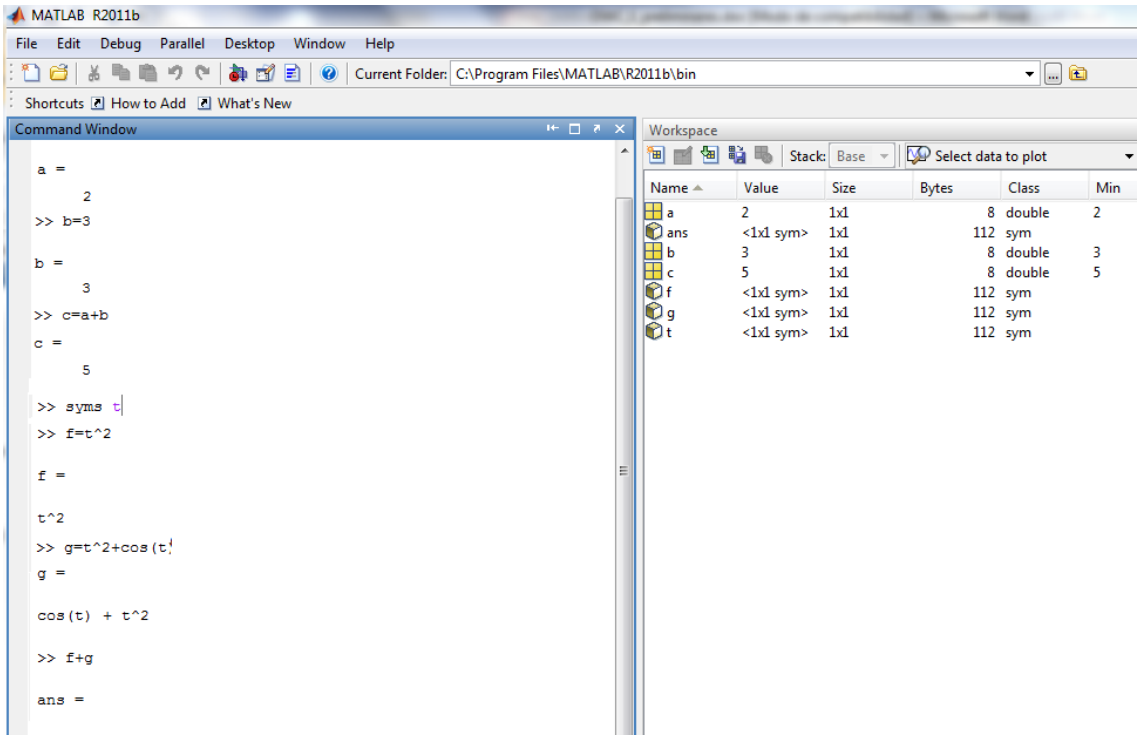
% El distinto tratamiento que Matlab da a las variables simbólicas y numéricas, se puede ver en el área de trabajo, "Workspace" (en los desplegables de la cabecera), donde nos indica cada variable de la sesión con el tipo de variable, el tamaño, el espacio que ocupa, etc…

% En el desplegable Desktop,

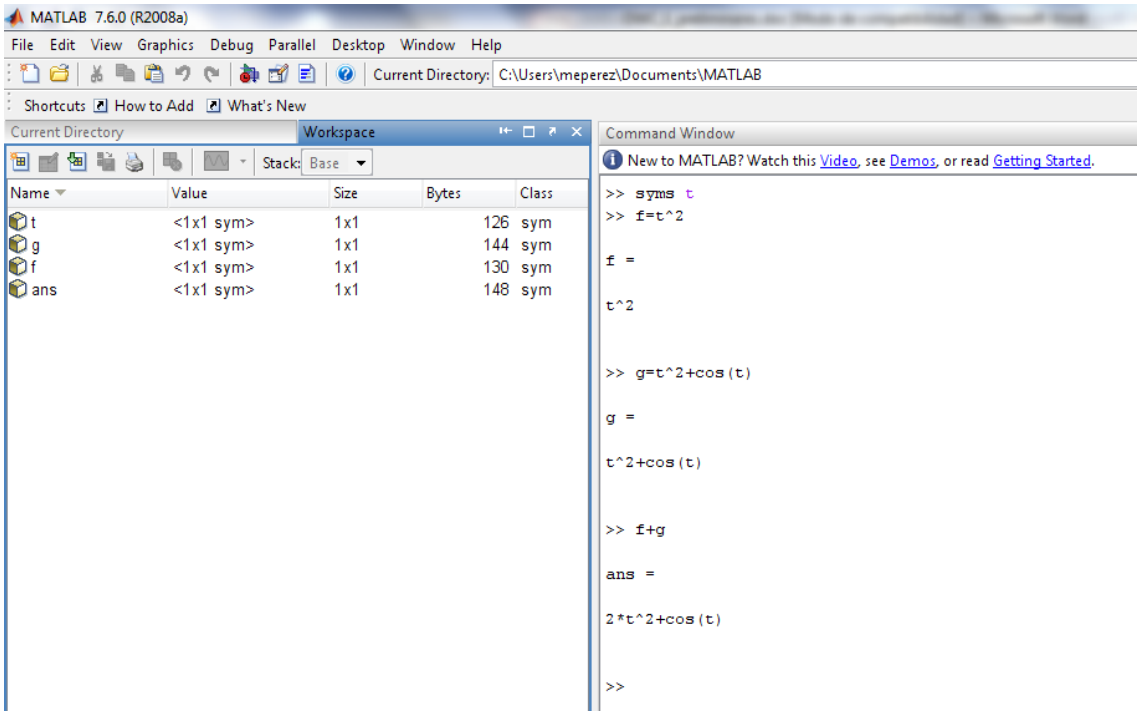


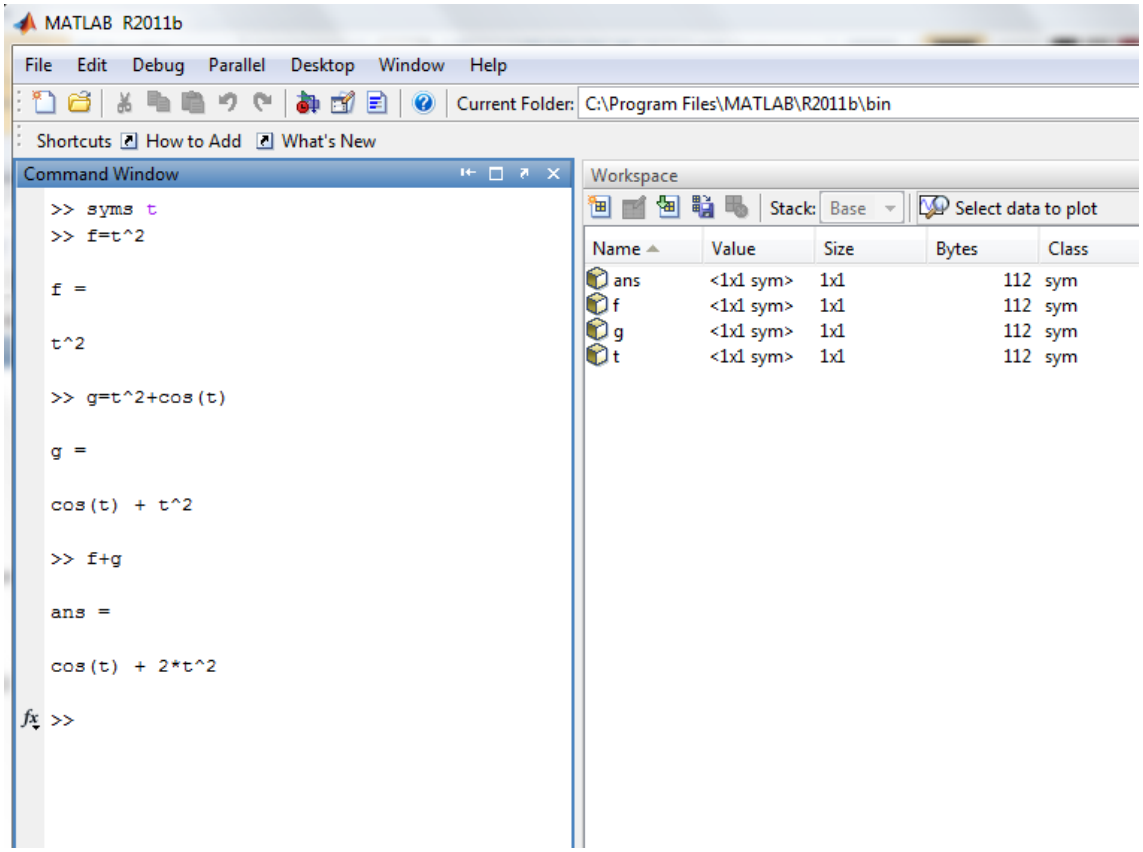
% y pinchando en Workspace vemos que el almacenamiento de las variables numéricas y simbólicas es muy distinto, y para las simbólicas varía con la versión de Matlab. Esto se ve en las fotos de "workspaces" abajo. El tratamiento que Matlab da a estas variables es también muy distinto





% Las variables simbólicas en Matlab son de la clase "sym", dependiendo de la versión, ocupan distinto espacio. Las variables numéricas son de la clase "double" y ocupan 8 bytes.





% Listas de funciones internas de Matlab nos las dan los comandos "help elfun", "help elmat" y "help matfun" entre otros

>> help elfun

```
Elementary math functions.

 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.
   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.

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" nos da ayuda sobre algo. Para saber qué es o como se utiliza la función "sin":

>> help sin

```
SIN    Sine of argument in radians.
   SIN(X) is the sine of the elements of X.

   See also asin, sind.

   Overloaded methods:
     distributed/sin
     sym/sin

   Reference page in Help browser
     doc sin
```

>> help elmat

```
Elementary matrices and matrix manipulation.

  Elementary matrices.
    zeros       - Zeros array.
    ones        - Ones array.
    eye         - Identity matrix.
    repmat      - Replicate and tile array.
    rand        - Uniformly distributed random numbers.
    randn       - Normally distributed random numbers.
    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.
    isequalwithequalnans - True if arrays are numerically equal.

  Matrix manipulation.
    cat         - Concatenate arrays.
    reshape     - Change size.
    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.
    flipdim     - Flip matrix along specified dimension.
    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.

Special variables and constants.
  ans         - Most recent answer.
  eps         - Floating point relative accuracy.
  realmax     - Largest positive floating point number.
  realmin     - Smallest positive floating point number.
  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.
  why         - Succinct answer.

Specialized matrices.
  compan      - Companion matrix.
  gallery     - Higham 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.
```

% Utilizando la variable t y las funciones internas podemos definir otras funciones o variables simbólicas que se pueden derivar, integrar, dibujar,… Así lo hemos hecho para definir h=h(t):

>> h

h =

1/exp(t) + 2*sin(t) + t^2

% derivando: si solo hay una variable simbólica ("variable independiente" en ED) lo hace con respecto a ella, si no hay que decirle con respecto a qué tiene que derivar

>> diff(h)

ans =

2*t - 1/exp(t) + 2*cos(t)

>> diff(diff(h)) *% derivada segunda*

ans =

1/exp(t) - 2*sin(t) + 2

>> diff(h,2) *% nos da la derivada segunda también*

ans =

1/exp(t) - 2*sin(t) + 2

>> diff(h,2)-diff(diff(h)) *% comparando los dos últimos comandos*

ans =

0

% integrando: si solo hay una variable simbólica, lo hace con respecto a ella, si no hay que decirle con respecto a qué tiene que integrar

>> h

h =

1/exp(t) + 2*sin(t) + t^2

>> int(h)

ans =

t^3/3 - 2*cos(t) - 1/exp(t)

% de manera general siempre falta la constante de integración

>> diff(int(h))-h *% comprobando integración*

ans =

0

% atención, integra lo que puede: de la función exp(-t^2) no nos devuelve una primitiva sino que nos la deja como una integral definida

>> int(exp(-t^2))

ans =

(pi^(1/2)*erf(t))/2

>> help erf

```
erf Error function.
   Y = erf(X) is the error function for each element of X.  X must be
   real. The error function is defined as:

     erf(x) = 2/sqrt(pi) * integral from 0 to x of exp(-t^2) dt.
```

% Además de "ans", y de estas funciones como "erf" que se muestran con "help", otras variables internas de Matlab que pueden resultar útiles son: "pi", "realmax", "realmin", "eps", "NaN", "Inf", etc.

>> realmax

ans =

1.7977e+308

>> realmin

ans =

2.2251e-308

>> NaN

ans =

NaN

>> help NaN

```
NaN    Not-a-Number.
    NaN is the IEEE arithmetic representation for Not-a-Number.
    A NaN is obtained as a result of mathematically undefined
    operations like 0.0/0.0  and inf-inf.

    NaN('double') is the same as NaN with no inputs.

    NaN('single') is the single precision representation of NaN.

    NaN(N) is an N-by-N matrix of NaNs.

    NaN(M,N) or NaN([M,N]) is an M-by-N matrix of NaNs.

    NaN(M,N,P,...) or NaN([M,N,P,...]) is an M-by-N-by-P-by-... array of NaNs.

    NaN(...,CLASSNAME) is an array of NaNs of class specified by CLASSNAME.
    CLASSNAME must be either 'single' or 'double'.

    Note: The size inputs M, N, and P... should be nonnegative integers.
    Negative integers are treated as 0.

    See also inf, isnan, isfinite, isfloat.

    Overloaded methods:
      distributed/nan
      codistributor2dbc/nan
      codistributor1d/nan
      codistributed/nan

    Reference page in Help browser
       doc nan
```

>> 0/0

ans =

  NaN

>> 1/0

ans =

  Inf

>> help eps

```
 EPS  Spacing of floating point numbers.
D = EPS(X), is the positive distance from ABS(X) to the next larger in
magnitude floating point number of the same precision as X.
X may be either double precision or single precision.
For all X, EPS(X) is equal to EPS(ABS(X)).

EPS, with no arguments, is the distance from 1.0 to the next larger double
precision number, that is EPS with no arguments returns 2^(-52).

EPS('double') is the same as EPS, or EPS(1.0).
EPS('single') is the same as EPS(single(1.0)), or single(2^-23).
```

```
Except for numbers whose absolute value is smaller than REALMIN,
if 2^E <= ABS(X) < 2^(E+1), then
   EPS(X) returns 2^(E-23) if ISA(X,'single')
   EPS(X) returns 2^(E-52) if ISA(X,'double')

For all X of class double such that ABS(X) <= REALMIN, EPS(X)
returns 2^(-1074).   Similarly, for all X of class single such that
ABS(X) <= REALMIN('single'), EPS(X) returns 2^(-149).

Replace expressions of the form
   if Y < EPS * ABS(X)
with
   if Y < EPS(X)

Example return values from calling EPS with various inputs are
presented in the table below:

     Expression                Return Value
    ==========================================
     eps(1/2)                   2^(-53)
     eps(1)                     2^(-52)
     eps(2)                     2^(-51)
     eps(realmax)                2^971
     eps(0)                     2^(-1074)
     eps(realmin/2)              2^(-1074)
     eps(realmin/16)             2^(-1074)
     eps(Inf)                   NaN
     eps(NaN)                   NaN
    ------------------------------------------
     eps(single(1/2))            2^(-24)
     eps(single(1))             2^(-23)
     eps(single(2))             2^(-22)
     eps(realmax('single'))      2^104
     eps(single(0))             2^(-149)
     eps(realmin('single')/2)    2^(-149)
     eps(realmin('single')/16)   2^(-149)
     eps(single(Inf))           single(NaN)
     eps(single(NaN))            single(NaN)

    See also realmax, realmin.

Overloaded methods:
   codistributed/eps
   qfft/eps

Reference page in Help browser
   doc eps
```

>> pi

ans =

3.1416

% parece que cualquier calculadora nos da más cifras decimales de precisión de las que vemos para el número "pi", por ejemplo, o para cualquier operación. Ocurre que Matlab por defecto trabaja en doble precisión y esto significa más cifras decimales de precisión. (ver, e.g., http://www.mathworks.es/es/help/symbolic/double.html). Se pueden visualizar cambiando el formato con "format"

>> help format

```
FORMAT Set output format.
   FORMAT with no inputs sets the output format to the default appropriate
   for the class of the variable. For float variables, the default is
   FORMAT SHORT.

   FORMAT does not affect how MATLAB computations are done. Computations
   on float variables, namely single or double, are done in appropriate
   floating point precision, no matter how those variables are displayed.
   Computations on integer variables are done natively in integer. Integer
   variables are always displayed to the appropriate number of digits for
   the class, for example, 3 digits to display the INT8 range -128:127.
   FORMAT SHORT and LONG do not affect the display of integer variables.

   FORMAT may be used to switch between different output display formats
   of all float variables as follows:
     FORMAT SHORT     Scaled fixed point format with 5 digits.
     FORMAT LONG      Scaled fixed point format with 15 digits for double
                      and 7 digits for single.
     FORMAT SHORT E   Floating point format with 5 digits.
     FORMAT LONG E    Floating point format with 15 digits for double and
                      7 digits for single.
     FORMAT SHORT G   Best of fixed or floating point format with 5
                      digits.
     FORMAT LONG G    Best of fixed or floating point format with 15
                      digits for double and 7 digits for single.
     FORMAT SHORT ENG Engineering format that has at least 5 digits
                      and a power that is a multiple of three
     FORMAT LONG ENG  Engineering format that has exactly 16 significant
                      digits and a power that is a multiple of three.

   FORMAT may be used to switch between different output display formats
   of all numeric variables as follows:
     FORMAT HEX     Hexadecimal format.
     FORMAT +       The symbols +, - and blank are printed
                    for positive, negative and zero elements.
                    Imaginary parts are ignored.
     FORMAT BANK    Fixed format for dollars and cents.
     FORMAT RAT     Approximation by ratio of small integers.  Numbers
                    with a large numerator or large denominator are
                    replaced by *.

   FORMAT may be used to affect the spacing in the display of all
   variables as follows:
     FORMAT COMPACT Suppresses extra line-feeds.
     FORMAT LOOSE   Puts the extra line-feeds back in.

   Example:
      format short, pi, single(pi)
   displays both double and single pi with 5 digits as 3.1416 while
      format long, pi, single(pi)
   displays pi as 3.141592653589793 and single(pi) as 3.1415927.


      format, intmax('uint64'), realmax
   shows these values as 18446744073709551615 and 1.7977e+308 while
      format hex, intmax('uint64'), realmax
   shows them as ffffffffffffffff and 7fefffffffffffff respectively.
   The HEX display corresponds to the internal representation of the value
   and is not the same as the hexadecimal notation in the C programming
   language.

   See also disp, display, isnumeric, isfloat, isinteger.

   Overloaded methods:
      quantizer/format
```

% Cualquier manera de expresar el formato puede resultar útil en un momento dado, especialmente los formatos 'g' que muestran resultados de forma agradable para leer

>> pi

ans =

3.1416

>> format long

>> pi

ans =

3.141592653589793

% "vpa" puede darnos el número dígitos que queramos si la variable tiene sentido como número (en este caso, pi es una variable interna). Abajo 1000 cifras del número pi o más….

>> vpa(pi,1000)

ans =

3.14159265358979323846264338327950288419716939937510582097494459230781
64062862089986280348253421170679821480865132823066470938446095505822 3
17253594081284811174502841027019385211055596446229489549303819644288 1
09756659334461284756482337867831652712019091456485669234603486104543 2
66482133936072602491412737245870066063155881748815209209628292540917 1
53643678925903600113305305488204665213841469519415116094330572703657 5
95919530921861173819326117931051185480744623799627495673518857527248 9
12279381830119491298336733624406566430860213949463952247371907021798 6
09437027705392171762931767523846748184676694051320005681271452635608 2
77857713427577896091736371787214684409012249534301465495853710507922 7
96892589235420199561121290219608640344181598136297747713099605187072 1
13499999983729780499510597317328160963185950244594553469083026425223 0
82533446850352619311881710100031378387528865875332083814206171776691 4
73035982534904287554687311595628638823537875937519577818577805321712 2
68066130019278766111959092164201 99

>> vpa(pi,10000)

ans =

3.14159265358979323846264338327950288419716939937510582097494459230781
64062862089986280348253421170679821480865132823066470938446095505822317253594081284811174502841027019……
……
……..677646575237428802109276457931065792295524988727584610126483699989225695968815920560010165525637568

% a veces Matlab nos devuelve resultados de cálculos simbólicos usando un formato racional, para ver los valores numéricos de lo que se pueda, se puede utilizar también "vpa"

>> 28/500*cos(5)

ans =

0.015885082385941

>> 28/500*cos(t)

ans =

(7*cos(t))/125

>> vpa(ans,3)

ans =

0.056*cos(t)

% así vpa "convierte a número" lo que tiene sentido numérico en la expresión, y nos muestra el número de cifras dependiente del entero que le indiquemos después de "," (usar "help vpa" o ver http://es.mathworks.com/help/symbolic/vpa.html para más detalles)

>> cos(2)*exp(t)-7*exp(2)*sin(t^2)

ans =

*- (909927547095103*sin(t^2))/17592186044416 - (7496634952020485*exp(t))/18014398509481984*

>> vpa(cos(2)*exp(t)-7*exp(2)*sin(t^2),2)

ans =

- 52.0*sin(t^2) - 0.42*exp(t)

>> format short *% cambiamos formato*

% Integrales definidas: integral definida de h entre 0 y 2

>> int(h,0,2)

ans =

17/3 - 1/exp(2) - 2*cos(2)

>> r=int(h,0,2)

r =

17/3 - 1/exp(2) - 2*cos(2)

% "r" es una variable simbólica que tiene significado de número. Se convierte a número con "double"

>> double(r)

ans =

6.3636

>> format long

>> double(r)

ans =

6.363625056524339

>> vpa(r,50) *% "vpa" nos da los dígitos que queramos*

ans =

6.3636250565243387487678036306957066427910366629082

>> format short

% Evaluando una función

>>sin(2) *% se trata de la evaluación en t=2 de una función interna*

ans =

0.9093

>> h(2) *% vemos que no se puede evaluar de la misma forma*

Error using mupadmex
Error in MuPAD command: Index exceeds matrix dimensions.

Error in sym/subsref (line 1410)
B = mupadmex('symobj::subsref',A.s,inds{:});

% se puede pedir a Matlab que lo evalúe con "subs" (substituir):

>> subs(h,t,2)

ans =

5.9539

% Otra forma de evaluar funciones que creemos, como si fueran funciones internas, y sin necesidad de definir variables simbólicas, es utilizando ficheros Matlab o utilizando @:

>> hh=@(x)[1/exp(x) + 2*sin(x) + x^2]

hh =

@(x)[1/exp(x)+2*sin(x)+x^2]

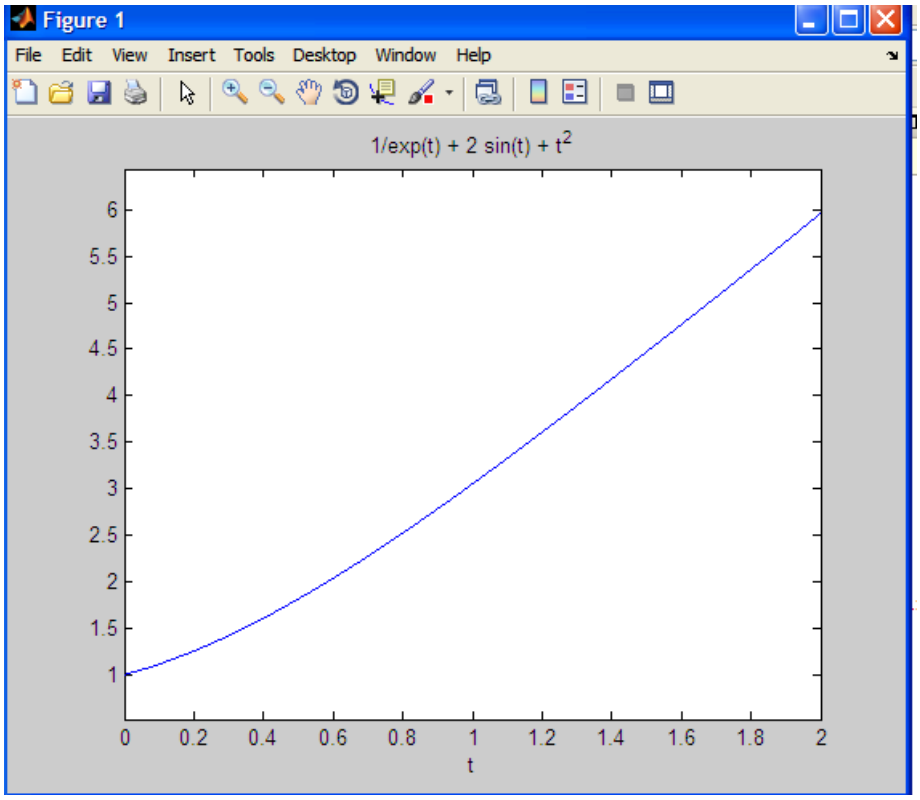
>> hh(2)

ans =

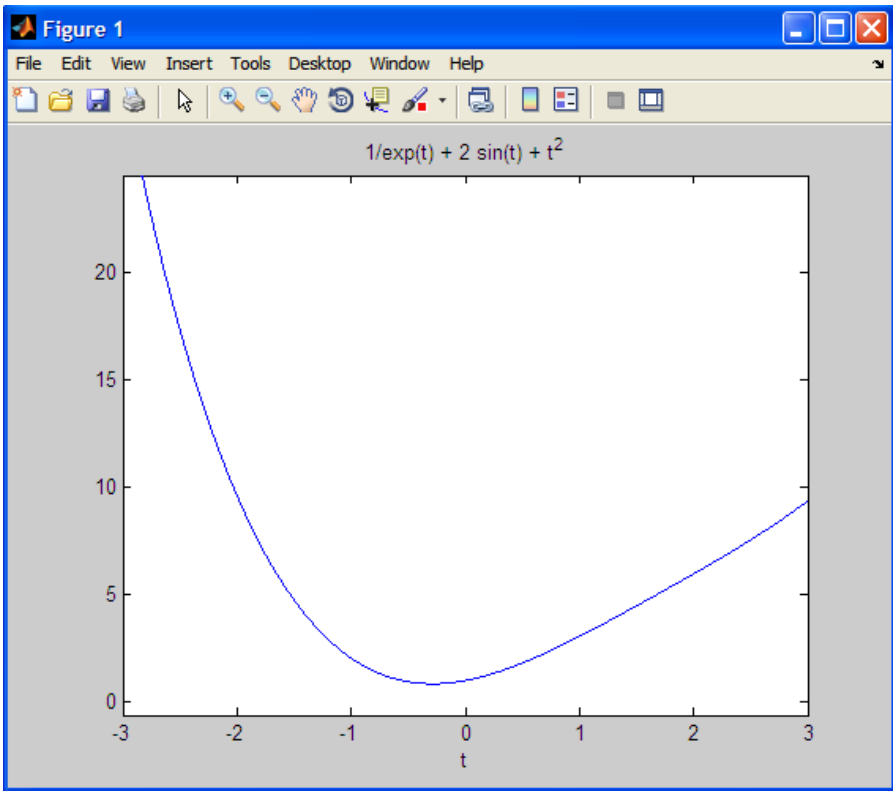
5.9539

% Dibujando una función de variable simbólica en distintos intervalos

>> ezplot(h,[0,2])



>> ezplot(h,[-3,3])



% dibujando punto a punto una función con el comando "plot": hay que definir los puntos con abscisa y ordenada. Abajo una forma de hacerlo:

% vector de abscisas (con elementos entre 0 y 2)

>> dt=0:0.2:2 *% empezando en 0, terminando en 2, con incrementos de 0.2*

```
dt =

  Columns 1 through 8

       0    0.2000    0.4000    0.6000    0.8000    1.0000    1.2000    1.4000

  Columns 9 through 11

    1.6000    1.8000    2.0000
```

% vector de ordenadas

>> dh=subs(h,t,dt)

```
dh =

  Columns 1 through 8

    1.0000    1.2561    1.6092    2.0381    2.5240    3.0508    3.6053    4.1775

  Columns 9 through 11

    4.7610    5.3530    5.9539
```

% hacer pero no mostrar con ";"

>> dt=0:0.2:2;

>> dh=subs(h,t,dt);

% para mostrar los pares abscisa y ordenada:

>> [dt;dh]

ans =

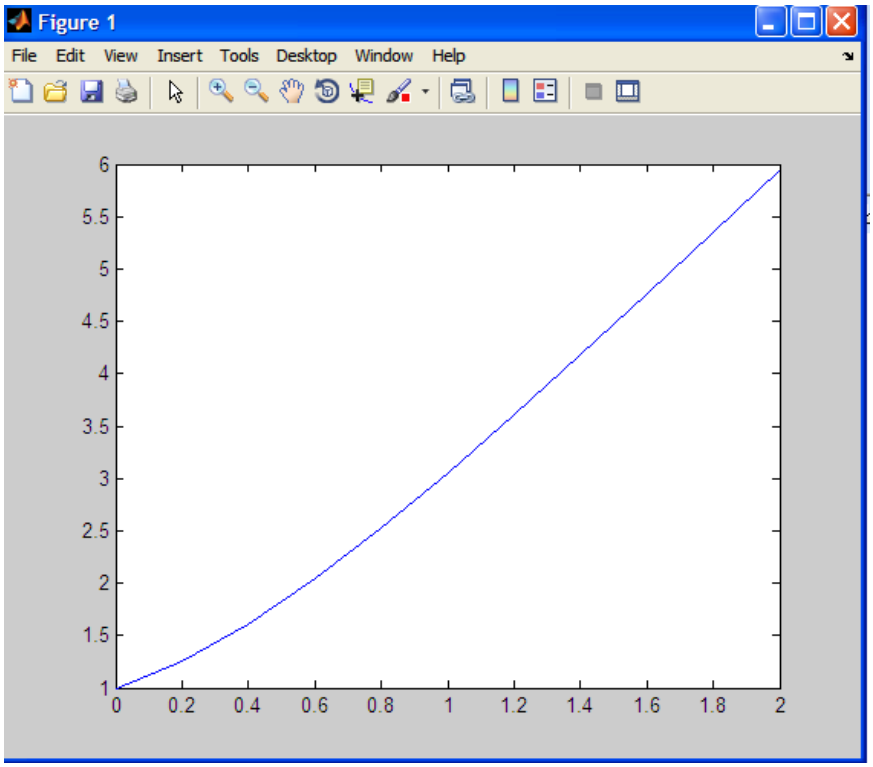
Columns 1 through 8

0	0.2000	0.4000	0.6000	0.8000	1.0000	1.2000	1.4000
1.0000	1.2561	1.6092	2.0381	2.5240	3.0508	3.6053	4.1775

Columns 9 through 11

1.6000	1.8000	2.0000
4.7610	5.3530	5.9539

>> plot(dt,dh)



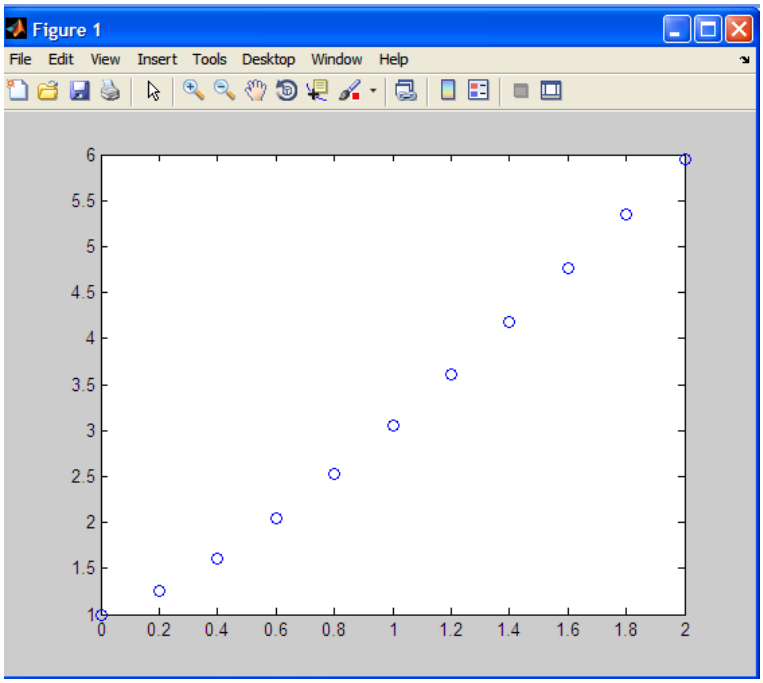
% "plot" admite opciones de dibujo: un círculo, en rojo, en verde, etc.

>> help plot

```
plot   Linear plot.
   plot(X,Y) plots vector Y versus vector X. If X or Y is a matrix,
   then the vector is plotted versus the rows or columns of the matrix,
   whichever line up.  If X is a scalar and Y is a vector, disconnected
   line objects are created and plotted as discrete points vertically at  X.
```

```
    plot(Y) plots the columns of Y versus their index.
If Y is complex, plot(Y) is equivalent to plot(real(Y),imag(Y)).
In all other uses of plot, the imaginary part is ignored.

Various line types, plot symbols and colors may be obtained with
plot(X,Y,S) where S is a character string made from one element
from any or all the following 3 columns:

    b     blue          .     point              -     solid
    g     green         o     circle             :     dotted
    r     red           x     x-mark             -.    dashdot
    c     cyan          +     plus               --    dashed
    m     magenta       *     star             (none)  no line
    y     yellow        s     square
    k     black         d     diamond
    w     white         v     triangle (down)
                        ^     triangle (up)
                        <     triangle (left)
                        >     triangle (right)
                        p     pentagram
                        h     hexagram

For example, plot(X,Y,'c+:') plots a cyan dotted line with a plus
at each data point; plot(X,Y,'bd') plots blue diamond at each data
point but does not draw any line.

plot(X1,Y1,S1,X2,Y2,S2,X3,Y3,S3,...) combines the plots defined by
the (X,Y,S) triples, where the X's and Y's are vectors or matrices
and the S's are strings.

For example, plot(X,Y,'y-',X,Y,'go') plots the data twice, with a
solid yellow line interpolating green circles at the data points.

The plot command, if no color is specified, makes automatic use of
the colors specified by the axes ColorOrder property.  By default,
plot cycles through the colors in the ColorOrder property.  For
monochrome systems, plot cycles over the axes LineStyleOrder property.

Note that RGB colors in the ColorOrder property may differ from
similarly-named colors in the (X,Y,S) triples.  For example, the
second axes ColorOrder property is medium green with RGB [0 .5 0],
while plot(X,Y,'g') plots a green line with RGB [0 1 0].

If you do not specify a marker type, plot uses no marker.
If you do not specify a line style, plot uses a solid line.

plot(AX,...) plots into the axes with handle AX.

plot returns a column vector of handles to lineseries objects, one
handle per plotted line.

The X,Y pairs, or X,Y,S triples, can be followed by
parameter/value pairs to specify additional properties
of the lines. For example, plot(X,Y,'LineWidth',2,'Color',[.6 0 0])
will create a plot with a dark red line width of 2 points.

Example
  x = -pi:pi/10:pi;
  y = tan(sin(x)) - sin(tan(x));
  plot(x,y,'--rs','LineWidth',2,...
               'MarkerEdgeColor','k',...
               'MarkerFaceColor','g',...
               'MarkerSize',10)

See also plottools, semilogx, semilogy, loglog, plotyy, plot3, grid,
title, xlabel, ylabel, axis, axes, hold, legend, subplot, scatter.

Overloaded methods:
    ……..
    …….
```

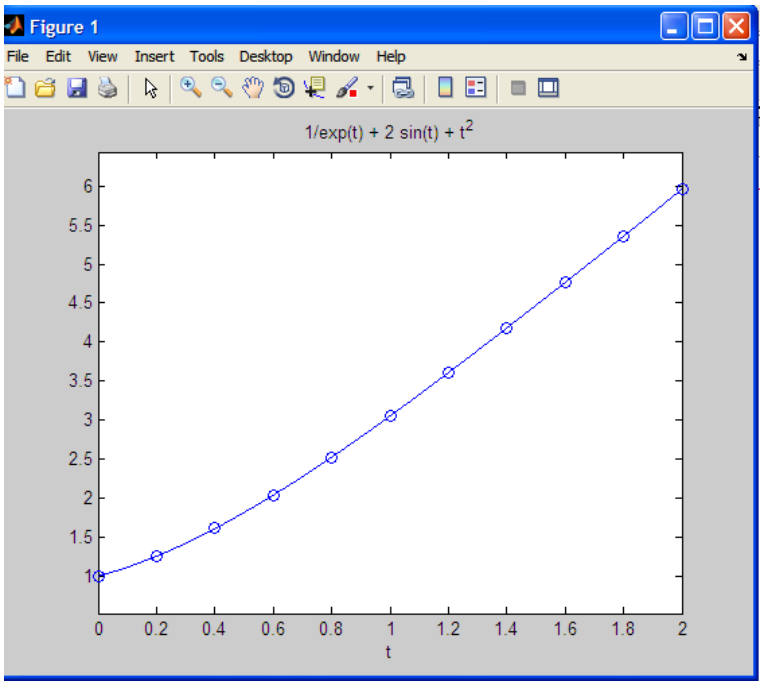
>> plot(dt,dh,'o')



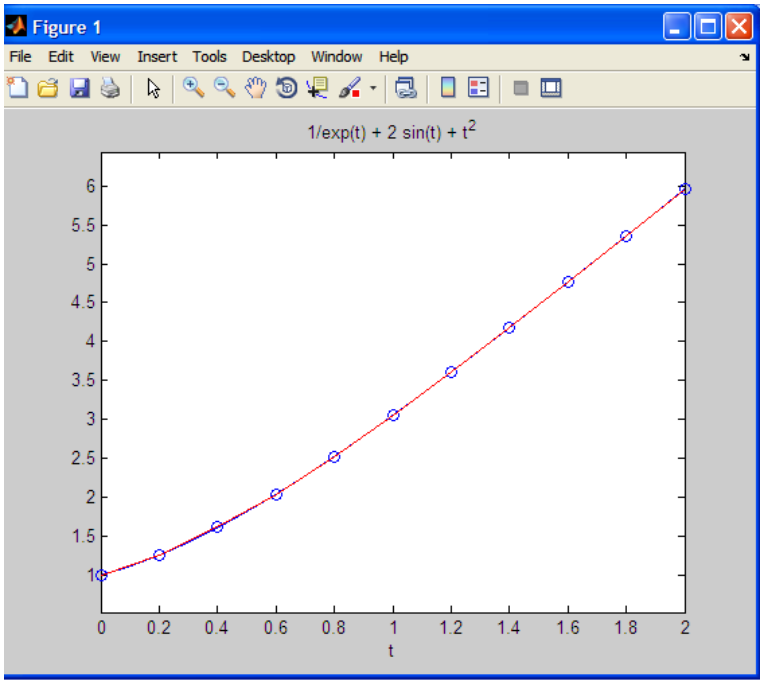
% comparando con "ezplot": utilizamos el comando "hold on" que almacena gráficas hasta que ejecutemos "hold off". Esto es, todas las gráficas se hacen en la misma figura (superponiéndose en este caso), lo que nos permite comparar

>> hold on

>> ezplot(h,[0,2])

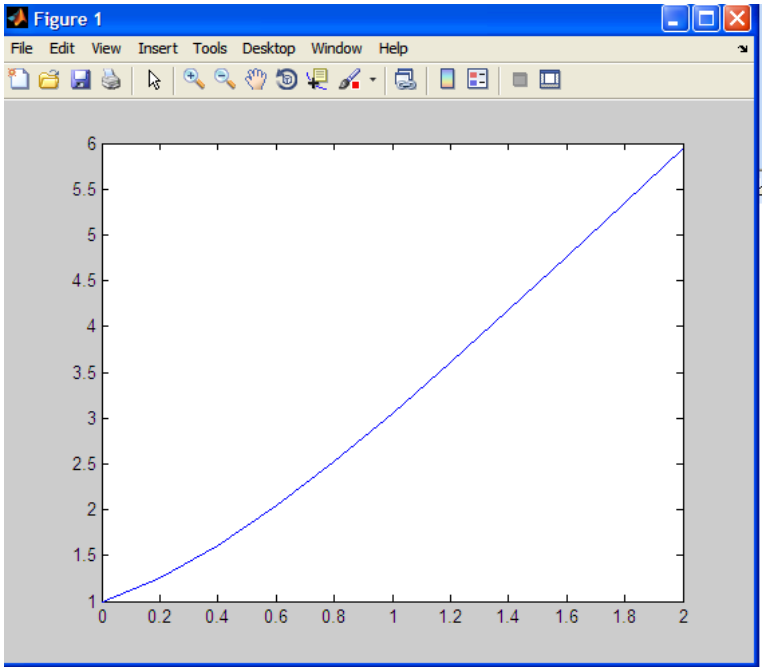


>> plot(dt,dh,'r')



>> hold off *% deja de almacenar gráficas*

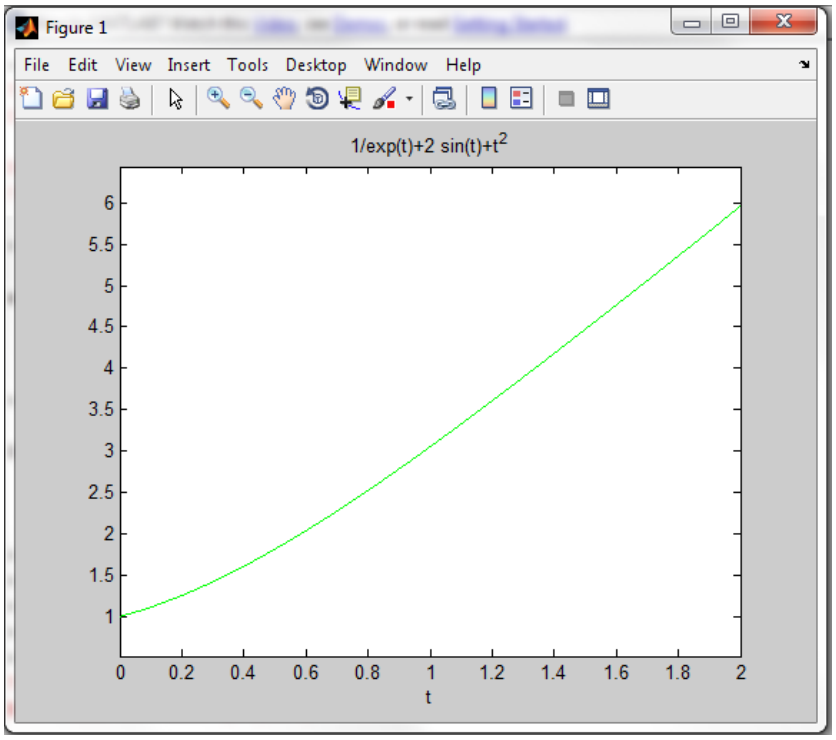
>> plot(dt,dh)



% una manera de cambiar el color con el comando ezplot es el siguiente

>> GR=ezplot(h,[0,2]);

>> set(GR,'Color','g')



>> clear all *% borra todas las variables definidas en la sesión*

>> clc *% borra la pantalla*

% funciones de dos variables simbólicas

>> syms t x

>> u=sin(x)*cos(t)+t^2

u =

cos(t)*sin(x) + t^2

% derivadas parciales: hay que indicar con respecto a que variable se deriva

>> diff(u,t) *% con respecto a t*

ans =

2*t - sin(t)*sin(x)

>> diff(u,x) *% con respecto a x*

ans =

cos(t)*cos(x)

>> diff(diff(u,t),x) *% derivadas parciales segundas cruzadas*

ans =

-cos(x)*sin(t)

>> diff(u,t,2) *% derivada parcial segunda con respecto a t*

ans =

2 - cos(t)*sin(x)

>> u

u =

cos(t)*sin(x) + t^2

% integrando con respecto a t o x

>> int(u,t)

ans =

sin(t)*sin(x) + t^3/3

% integral definida

>> int(u,t,0,2) *% función de x*

ans =

sin(2)*sin(x) + 8/3

>> int(int(u,t),x) *% integral doble*

ans =

(t^3*x)/3 - cos(x)*sin(t)

% evaluando u en (2,3)

>> subs(subs(u,t,2),x,3)

ans =

3.9413

>> subs(u,t,x,2,3) *% no funciona, nos da error*

{Error using <a href="matlab:helpUtils.errorDocCallback('sym.subs')" style="font

% lo que funciona es

>> subs(u,{t,x},{2,3})

ans =

3.9413

% cortando por planos ; por ejemplo por t=2

>> w=subs(u,t,2)

w =

4 - (7496634952020485*sin(x))/18014398509481984

% incomodo de leer. Parece que Matlab intenta aproximar un número por otro con formato racional. Visualizamos mejor con:

>> vpa(w,6)

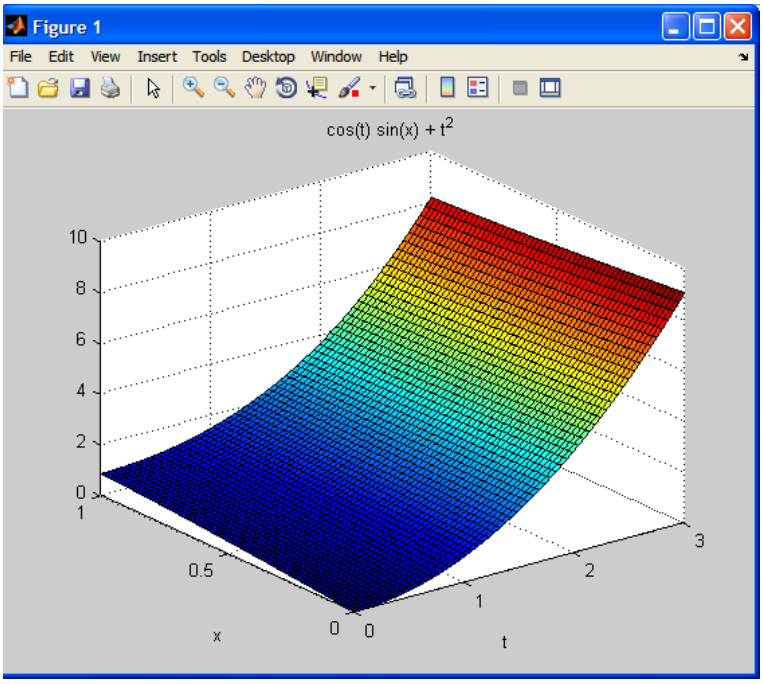
ans =

4.0 - 0.416147*sin(x)

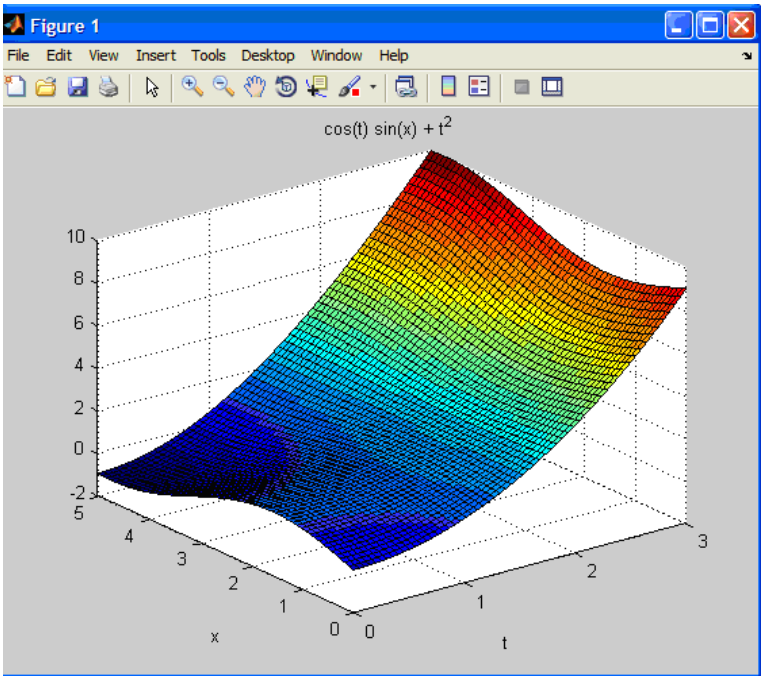
>> ezplot(w,[0,3])

% dibujando la superficie z=u con "ezsurf" cuando u está definida como función de dos variables simbólicas t y x variando en [0,3]x[0,1] (y luego en [0,3]x[0,5])

>> ezsurf(u,[0,3],[0,1])



>> ezsurf(u,[0,3],[0,5])



% cortando por planos y dibujando de manera que nos muestre/conserve varias gráficas en distintas figuras sobre la pantalla. Se introducen los comandos "figure(1)", "figure(2)","figure(3)", …

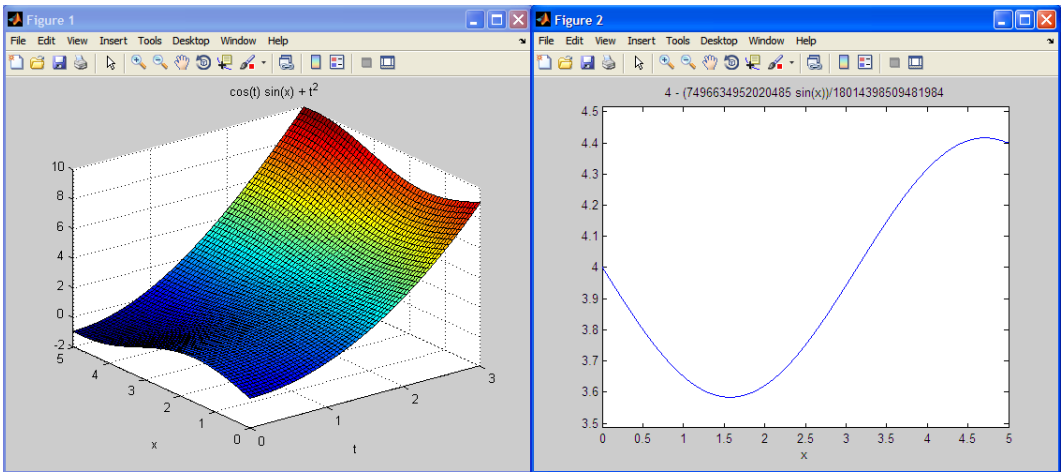
>> w=subs(u,t,2) *% cortando por el plano t=2*

w =

4 - (7496634952020485*sin(x))/18014398509481984

>> figure(2)

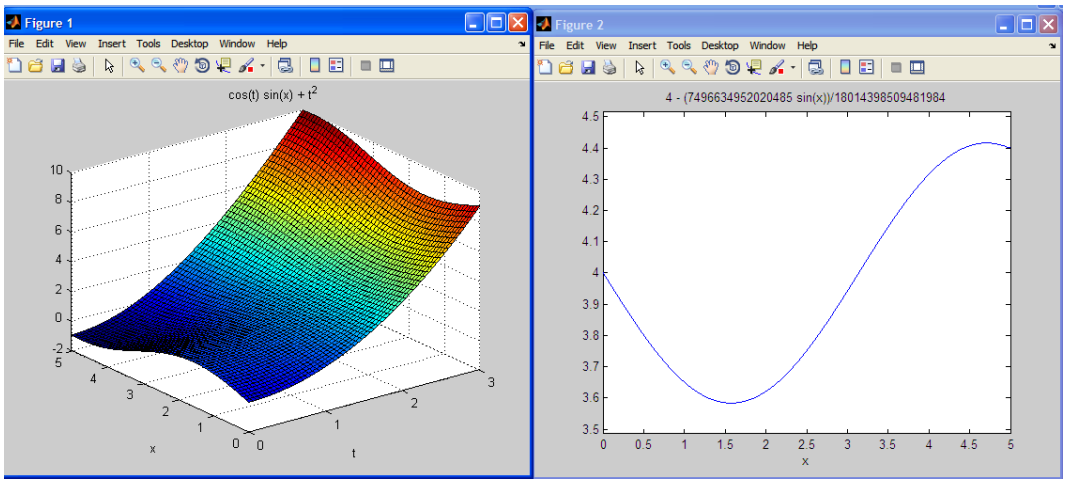
>> ezplot(w,[0,5])



% cortando por el plano x=1 y visualizando el perfil de la superficie con el corte

>> figure(3)

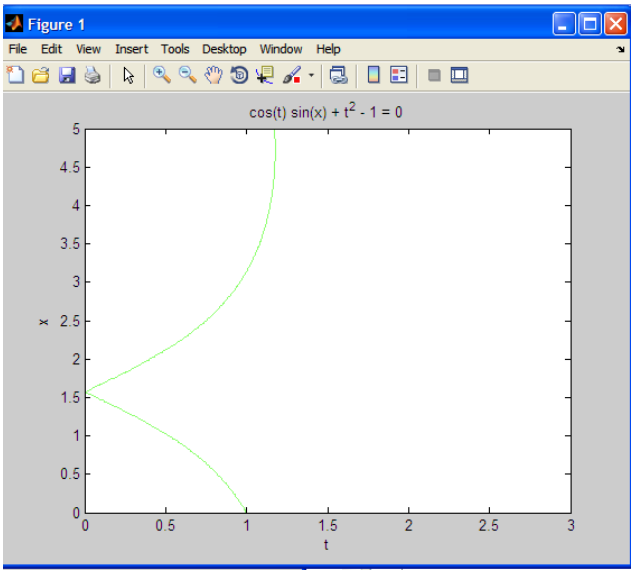
>> ezplot(subs(u,x,1),[0,3])



% Dibujando curvas de nivel con "ezplot" y "ezcontour"

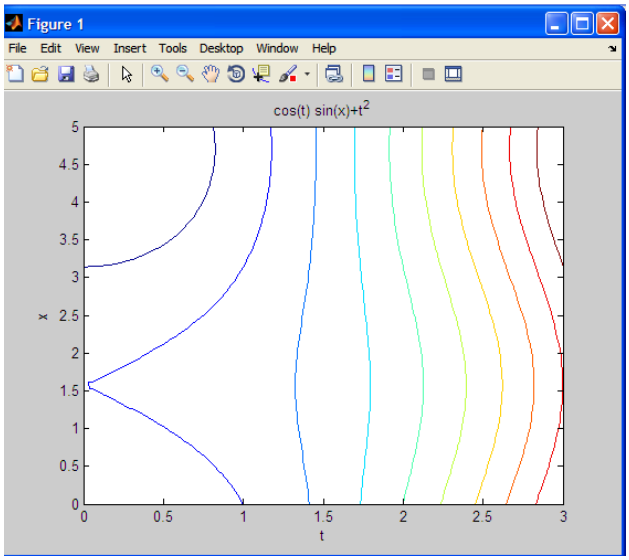
% "ezplot" dibuja también curvas definidas en forma implícita. Por ejemplo, la curva de nivel u=1

>> ezplot(u-1,[0,3],[0,5])



% diversas curvas de nivel con "ezcontour"

>> ezcontour(u,[0,3],[0,5])



>> clc

% A continuación, introducimos nociones y operaciones del cálculo vectorial y matricial que son útiles en los programas para resolución numérica de ecuaciones diferenciales y en derivadas parciales (por ejemplo, en las funciones "elementosfinitos.m", "galerkin.m" y "ffinitge.m", entre otras).

% Definimos una matriz 3x3 y un vector columna, y hacemos operaciones básicas de multiplicación, inversión, cálculo de determinantes y resolución de sistemas, e introducimos algunas matrices especiales. Las variables en que se almacenan estas matrices nxm son de la clase "double", ocupan 8 x n x m bytes, de tamaño nxm (ver "size" en el Workspace). Dependiendo de la operación "ans" puede ser ahora un vector, una matriz o un número, si no se introducen variables simbólicas.

% Para definir una matriz, se separan los elementos de una fila por ",", y al cambiar de fila por ";".

>> A=[1,2,3;0,1,0;6,7,8]

A =

```
   1   2   3
   0   1   0
   6   7   8
```

>> b=[1;2;3]

b =

```
   1
   2
   3
```

% otras formas de definir el mismo vector columna

>> bb=1:1:3

bb =

```
   1   2   3
```

% transposición

>> bb'

ans =

1
2
3

>> bb(:)

ans =

1
2
3

% multiplicando elemento por elemento en un vector

>> bb(:).^2

ans =

1
4
9

% multiplicación de matrices

>> A*b

ans =

14
2
44

*% para elevar al cuadrado A^2 o A*A. Abajo la variable ans es una matriz*

```
>> A^2-A*A

ans =

     0     0     0
     0     0     0
     0     0     0

>> det(A) % determinante de A

ans =

   -10

>> inv(A) % inversa de A

ans =

   -0.8000   -0.5000    0.3000
         0    1.0000         0
    0.6000   -0.5000   -0.1000

% verificación

>> A*inv(A)

ans =

    1.0000         0         0
    0.0000    1.0000         0
    0.0000         0    1.0000

% resolver el sistema Ac=b

>> A\b

ans =

   -0.9000
    2.0000
   -0.7000

% comprobación : llamando 'c' al vector 'ans' de arriba
```

```
>> c=A\b

c =

  -0.9000
   2.0000
  -0.7000

>> A*c

ans =

   1.0000
   2.0000
   3.0000

% definiendo matrices especiales: de ceros, de unos, diagonales, identidad, etc.

>> eye(5) % matriz identidad 5x5

ans =

   1   0   0   0   0
   0   1   0   0   0
   0   0   1   0   0
   0   0   0   1   0
   0   0   0   0   1

>> ones(5,5) % matriz 5x5 de unos

ans =

   1   1   1   1   1
   1   1   1   1   1
   1   1   1   1   1
   1   1   1   1   1
   1   1   1   1   1

>> zeros(2,3) % matriz 2x3 de ceros

ans =

   0   0   0
   0   0   0
```

>> diag(b) *% matriz con b en la diagonal principal*

```
ans =

    1    0    0
    0    2    0
    0    0    3
```

>> diag([11,12,13,14])) *% matriz 4x4 con [11,12,13,14] en la diagonal*

```
ans =

   11    0    0    0
    0   12    0    0
    0    0   13    0
    0    0    0   14
```

>> diag(b,1) *% matriz con b en la diagonal por encima de la principal*

```
ans =

    0    1    0    0
    0    0    2    0
    0    0    0    3
    0    0    0    0
```

>> diag(b,-1) *% matriz con b en la diagonal por debajo de la principal*

```
ans =

    0    0    0    0
    1    0    0    0
    0    2    0    0
    0    0    3    0
```

% matriz tridiagonal simétrica 4x4

>> diag([11,12,13,14])+diag(b,1)+diag(b,-1)

```
ans =

   11    1    0    0
    1   12    2    0
    0    2   13    3
    0    0    3   14
```

% nos referimos a unos elementos de un vector como sigue

>> dt=0:0.2:2; *% definiendo el vector*

```
dt=0:0.2:2

dt =

  Columns 1 through 6

       0    0.2000    0.4000    0.6000    0.8000    1.0000

  Columns 7 through 11

    1.2000    1.4000    1.6000    1.8000    2.0000
```

>> dt(3:5) *% nos da los elementos 3, 4 y 5 de dicho vector*

```
ans =

    0.4000    0.6000    0.8000
```

>> dt(1:8) *% nos da los elementos del primero al octavo de dt*

```
ans =

       0    0.2000    0.4000    0.6000    0.8000    1.0000    1.2000    1.4000
```

% Todas estas instrucciones, ordenadas convenientemente, se pueden introducir en un fichero Matlab para resolver, por ejemplo, problemas de contorno numéricamente

% En relación al cálculo de los valores propios y vectores propios, para una matriz cuadrada y simétrica, por ejemplo,

>> A=[1,2,3;2,1,4;3,4,8]

```
A =

     1     2     3
     2     1     4
     3     4     8
```

>> eig(A) *% nos da un vector columna que contiene los valores propios*

```
ans =

  -1.1536
   0.0783
  11.0753
```

% para sacar los vectores propios, hay que introducirlos en una matriz

>> [vp,l]=eig(A)

```
vp =

   0.4965   0.8013   0.3338
  -0.8433   0.3541   0.4043
   0.2058  -0.4822   0.8515


l =

  -1.1536        0        0
        0   0.0783        0
        0        0  11.0753
```

% arriba, l es una matriz que tiene en la diagonal principal los valores propios y vp es una matriz cuyas columnas son los vectores propios. Por ejemplo, vp(1:3) nos da el vector propio asociado con el valor propio -1.1536. Como se comprueba a continuación, se trata de un cálculo numérico que involucra un error

>> (A+1.1536*eye(3))*vp(1:3)'

```
ans =

  1.0e-005 *

  -0.2229
   0.3787
  -0.0924
```

% se han tomado sólo cuatro cifras decimales para primer valor propio; si escribimos más cifras decimales de precisión, vemos que el error disminuye:

>> format long

>> [vp,l]=eig(A)

```
vp =

   0.496477270727066   0.801300714843487   0.333807555400894
  -0.843302009616461   0.354066166486433   0.404325203674516
   0.205796113248626  -0.482238855901221   0.851524307128599

l =

  -1.153604490133523                   0                   0
                   0   0.078268344144607                   0
                   0                   0  11.075336145988917
```

>> (A+1.153604490133523 *eye(3))*vp(1:3)'

```
ans =

  1.0e-015 *

   0.555111512312578
   0.777156117237609
                   0
```

% vp(1:3) nos da el vector transpuesto del vector propio asociado al primer valor propio

>> vp(1:3)

```
ans =

    0.4965   -0.8433    0.2058
```

>> vp(1:3,1)

```
ans =

    0.4965
   -0.8433
    0.2058
```

>> vp(1:3)'

ans =

0.4965
-0.8433
0.2058

>> vp(:,1)

ans =

0.4965
-0.8433
0.2058

% Así, v(1:3,1)=vp(1:3)'= vp(:,1) es el vector propio asociado al primer valor propio. El vector propio asociado al segundo valor propio

>> vp(1:3,2)

ans =

0.8013
0.3541
-0.4822

% El vector propio asociado al tercer valor propio

>> vp(:,3)

ans =

0.3338
0.4043
0.8515

% Cambiamos el formato y hacemos verificaciones para el segundo y tercer valor propio y los vectores propios asociados.

>> format short g

```
>> (A-l(2,2)*eye(3))*vp(1:3,2)

ans =

   2.2204e-016
  -2.2204e-016
   1.3323e-015

>> (A-l(3,3)*eye(3))*vp(1:3,3)

ans =

   8.8818e-016
             0
  -8.8818e-016
```

% Para evitar errores numéricos, en ocasiones puede ser conveniente utilizar "sym(A)" como nos muestran las siguientes verificaciones (por brevedad evitamos mostrar resultados intermedios; usar "help sym/eig" para más detalles)

```
>> [VP,L]=eig(sym(A));

>> simplify((A-L(1,1)*eye(3))*VP(:,1))

ans =

  0
  0
  0

>> simplify((A-L(2,2)*eye(3))*VP(:,2))

ans =

  0
  0
  0

>> simplify((A-L(3,3)*eye(3))*VP(:,3))

ans =

  0
  0
  0
```

% Estas operaciones de cálculo vectorial pueden ser de utilidad en el cálculo numérico de los valores propios de una ecuación diferencial, también en la resolución de sistemas cuando quedan expresiones complicadas para un cálculo exacto.

% La aplicación de fórmulas integrales para resolución de ecuaciones diferenciales de orden superior a uno, o de sistemas diferenciales, involucra (e.g, el método de variación de parámetros) definir funciones vectoriales o matriciales de una variable simbólica t y hacer las operaciones elementales de multiplicación, inversión e integración. Ejemplo (ejercicio 14 de la hoja de problemas):

```
>> syms t

>> A=[t,t^2;1,2*t]

A =

    [   t, t^2]
    [   1, 2*t]

>> b=[t;t^3]

b =

  t
 t^3

>> det(A)

ans =

  t^2

>> inv(A)

ans =

    [    2/t,    -1]
    [ -1/t^2,    1/t]

>> inv(A)*b

ans =

   2-t^3
 -1/t+t^2
```

>> int(ans)

ans =

```
    2*t-1/4*t^4
 -log(t)+1/3*t^3
```

*% ahora A*ans nos da lo mismo que*

>> A*int(inv(A)*b)

ans =

```
 t*(2*t-1/4*t^4)+t^2*(-log(t)+1/3*t^3)
    2*t-1/4*t^4+2*t*(-log(t)+1/3*t^3)
```

>> pretty(ans)

```
[                4      2                    3 ]
[t (2 t - 1/4 t ) + t   (-log(t) + 1/3 t )]
[                                           ]
[             4                         3   ]
[ 2 t - 1/4 t  + 2 t (-log(t) + 1/3 t )   ]
```

% "pretty" en general permite visualizar mejor un resultado o una expresión matemática, pero depende de la versión de Matlab: arriba lo que se tiene con la versión 2008 y abajo con la 2011

>> pretty (ans)

```
+-                                                -+
|    /    4        \      /           3 \         |
|    |   t         |    2 |          t  |         |
|  t | - -- + 2 t  | - t  | log(t) - -- |         |
|    \   4         /      \          3  /         |
|                                                 |
|                 /           3 \    4            |
|                 |          t  |   t             |
|      2 t - 2 t  | log(t) - -- | - --            |
|                 \          3  /   4             |
+-                                                -+
```

% Para terminar la sesión de trabajo, y que Matlab nos guarde lo que hemos hecho desde el comienzo (fichero "preliminares"), usamos

>> diary off

---

*Diseño y Maquetación: **Antonio J. González Pérez***