# Curso de Programación en Matlab y Simulink

Alberto Herreros (albher@eis.uva.es)
Enrique Baeyens (enrbae@eis.uva.es)

Departamento de Ingeniería de Sistemas y Automática (DISA)
Escuela de Ingenierías Industriales (EII)
Universidad de Valladolid (UVa)

Curso 2010/2011



## Contenidos

# Contenidos





# ¿Qué es MATLAB?

- Es un lenguaje de alto nivel para computación e ingeniería. Integra computación, visualización y programación.
- Aplicaciones típicas de MATLAB son:
  - Matemáticas y computación
  - Desarrollo de algoritmos
  - Modelado, simulación y prototipado
  - Análisis de datos, exploración y visualización
  - Gráficos científicos y de ingeniería.
  - Desarrollo de aplicaciones
- Matlab es un sistema interactivo cuyo elemento básico son las matrices y no requiere dimensionamiento.
- El nombre proviene de "laboratorio de matrices".
- Originalmente fue escrito en FORTRAN y hacía uso de las librerías LINPACK y EISPACK
- Las últimas versiones están desarrolladas en C y utilizan las librerías LAPACK y BLAS.
- Sobre la base de MATLAB se han construido conjuntos de funciones específicas para diferentes problemas, denominadas "toolboxes".

- Lista explícita de elementos.
- Desde un fichero de datos externo.
- Utilizando funciones propias.
- Creando un fichero .m

Comenzaremos introduciendo manualmente la matriz de Dürer. Para ello utilizamos las siguientes reglas:

- Separar elementos de una fila con espacios o comas.
- Usar "punto y coma"; para indicar final de fila.
- Incluir la lista completa de elementos dentro de corchetes, [ ].





Para introducir la matriz de Dürer hacemos:

```
A = [16 3 2 13; 5 10 11 8; 9 6 7 12; 4 15 14 1]
```

Como resultado se obtiene

```
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

Una vez introducida una matriz, queda guardada en el entorno de trabajo de MATLAB.
La matriz `A` es un cuadrado mágico: Todas sus filas, columnas y diagonales suman lo mismo. Para comprobarlo hacemos

```
sum(A)
ans =
    34    34    34    34
```

El comando `sum(A)` calcula la suma de las columnas de la matriz A, obteniéndose un vector de dimensión el número de columnas.

Para calcular la suma de las filas, podemos calcular la transpuesta de la matriz.

```
A'
```

obteniendo

```
ans =
    16     5     9     4
     3    10     6    15
     2    11     7    14
    13     8    12     1
```

la suma de las filas, en formato vector columna es

```
sum(A')'
```

```
ans =
    34
    34
    34
    34
```





La función `diag` permite obtener un vector con los elementos de la diagonal principal.

```
diag(A)
```

Se obtiene

```
ans =
    16
    10
     7
     1
```

y la suma de los elementos de la diagonal principal es

```
sum(diag(A))
```

obteniéndose

```
ans =
    34
```

La antidiagonal de una matriz no suele ser muy importante, por lo que no hay ninguna función para extraerla. No obstante, puede invertirse la disposición de las columnas de la matriz con la función `fliplr`, así la suma de la antidiagonal es

```
sum(diag(fliplr(A)))
ans =
     34
```

Otra forma de obtener la suma de los elemento de la antidiagonal es sumando elemento a elemento.





Un elemento de la matriz A se referencia como `A(i,j)`, siendo `i` la fila y `j` la columna. La suma de la antidiagonal podría haberse obtenido también como sigue:

```
A(1,4)+A(2,3)+A(3,2)+A(4,1)
ans =
     34
```

También es posible acceder a cada elemento de una matriz con un solo índice, así `A(k)` corresponde al elemento `k` de un vector ficticio que se formara colocando las columnas de la matrix `A` una debajo de otra: Comprobar que `A(4,2)` y `A(8)` corresponden al mismo elemento de la matriz `A`.

Si se intenta acceder a un elemento que excede las dimensiones de la matriz, se obtiene un error

```
t = A(4,5)
Index exceeds matrix dimensions.
```

Si se inicializa un elemento que excede las dimensiones de la matriz, la matriz se acomoda en dimensión al nuevo elemento, con el resto de nuevos elementos inicializados a cero.

```
X = A;
X(4,5) = 17

X =
    16     3     2    13     0
     5    10    11     8     0
     9     6     7    12     0
     4    15    14     1    17
```



## El operador :

El operador : es uno de los más importantes de MATLAB. Tiene diferentes utilidades. La expresión

```
1:10
```

indica un vector que contiene los números enteros desde 1 hasta 10.

```
1   2   3   4   5   6   7   8   9  10
```

Para obtener un espaciado no unitario, se utiliza un incremento.

```
100:-7:50
```

es

```
100    93    86    79    72    65    58    51
```

y

```
0:pi/4:pi
```

es

```
0    0.7854    1.5708    2.3562    3.1416
```

Cuando el operador : aparece en los subíndices de una matriz se refiere a las filas o columnas y permite extraer submatrices. Por ejemplo, `A(1:k,j)` es el vector formado por los primeros `k` elementos de la columna `j` de la matriz `A` y

```
sum(A(1:4,4))
```

calcula la suma de todos los elementos de la cuarta columna. Otra forma más compacta y elegante de hacer lo mismo es

```
sum(A(:,end))
```

los dos puntos : (sin otros números) significan todas las filas y `end` se refier a la última columna.
Pregunta: ¿Qué esta calculando la siguiente expresión?

```
sum(A(end,:))
```





Matlab dispone de una función `magic` que permite calcular cuadrados mágicos
Haciendo

```
B = magic(4)
B =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

La matriz obtenida es casi la misma que la matriz de Dürer, solo se diferencia en que las columnas 2 y 3 están intercambiadas. Se pude obtener de nuevo la matriz de Dürer haciendo la siguiente operación

```
A = B(:,[1 3 2 4])
A =
    16     3     2    13
     5    10    11     8
     9     6     7    12
     4    15    14     1
```

Al igual que muchos otros lenguajes de programación, MATLAB dispone de expresiones matemáticas, pero al contrario que en la mayoría de los lenguajes de programación, estas expresiones hacen referencia a matrices.
Los bloques constructivos de las expresiones son

- Variables
- Números
- Operadores
- Funciones





- MATLAB no requiere ningún tipo de declaración o indicación de la dimensión. Cuando MATLAB encuentra un nuevo nombre de variable la crea automáticamente y reserva la cantidad de memoria necesaria. Si la variable ya existe, MATLAB cambia su contenido y si es necesario modifica la reserva de memoria.

  Por ejemplo, la expresión

  ```
  num_est = 15
  ```

  crea una matriz 1 por 1 llamada `num_est` y almacena el valor 25 en su único elemento.
- Los nombres de variables deben comenzar siempre por una letra y pueden incluir otras letras, números y el símbolo de subrayado, hasta un total de 31 caracteres.
- Se distingue entre mayúsculas y minúsculas. `A` y `a` no son la misma variable.
- Para ver el contenido de una variable, simplemente escribir el nombre de la variable.

- MATLAB utiliza notación decimal convencional, con punto decimal opcional y signo + ó -
- Es posible utilizar notación científica. La letra e especifica un factor de escala de potencia de 10.
- Los números imaginarios puros se especifican con la letra i ó j
- Los siguientes ejemplos son todos números válidos en MATLAB

```
3              -99            0.0001
9.6397238      1.60210e-20    6.02252e23
1i             -3.14159j      3e5i
```

- Internamente, los números se almacenan en formato largo utilizando la norma IEEE de punto flotante. La precisión es aproximadamente de 16 cifras decimales significativas y el rango está entre $10^{-308}$ y $10^{+308}$.





Las expresiones de MATLAB utilizan los operadores aritméticos usuales, así como sus reglas de precedencia

| | |
|---|---|
| + | Suma |
| - | Resta |
| * | Producto |
| / | División |
| \ | División por la izquierda (se explicará) |
| ^ | Potencia |
| ' | Transposición y conjugación compleja |
| ( ) | Orden de evaluación |

- MATLAB proporciona un gran número de funciones matemáticas elementales, por ejemplo, abs, sqrt, exp, sin, cos, etc.
- Por defecto, MATLAB utiliza números complejos:
- La raíz cuadrada o el logaritmo de un número negativo no producen error, sino que dan como resultado u número complejo.
- Los argumentos de las funciones pueden ser números complejos
- MATLAB proporciona también funciones avanzadas: Funciones de Bessel o funciones gamma.
- Una lista de todas las funciones elementales puede obtenerse con el comando

```
help elfun
```

- Funciones más avanzadas y funciones de matrices se obtienen con

```
help specfun
help elmat
```





- Algunas funciones están compiladas con el núcleo de MATLAB y son muy rápidas y eficientes. Ej. sqrt, sin
- Otras funciones están programadas en lenguaje de MATLAB (ficheros m). Pueden verse y modificarse
- Algunas funciones proporcionan el valor de ciertas constantes útiles.

| | |
|---|---|
| `pi` | 3.14159265 |
| `i` | $\sqrt{-1}$ |
| `j` | $\sqrt{-1}$ |
| `eps` | Precisión relativa de punto flotante $2^{-52}$ |
| `realmin` | Número en punto flotante más pequeño $2^{-1022}$ |
| `realmax` | Número en punto flotante más grande $(2-\epsilon)2^{+1023}$ |
| `Inf` | Infinito |
| `NaN` | Not-a-Number (no es un número) |

Infinito se obtiene al dividir un número no nulo por cero, o como resultado de evaluar expresiones matemáticas bien definidas.
NaN se obtiene al tratar de evaluar expresiones como 0/0 ó Inf-Inf que no tienen valores bien definidos

- Los nombres de las funciones no están reservados. Puede definirse una variable eps=1e-6 y utilizarla. Para restaurar su valor original

```
clear eps
```

Ya se han visto varios ejemplos de expresiones. Algunos otros ejemplos son los siguientes:

```
rho = (1+sqrt(5))/2
rho =
    1.6180

a = abs(3+4i)
a =
     5

z = sqrt(besselk(4/3,rho-i))
z =
   0.3730+ 0.3214i

huge = exp(log(realmax))
huge =
  1.7977e+308

toobig = pi*huge
toobig =
   Inf
```





- MATLAB proporciona cuatro funciones para generar matrices

| | |
|---|---|
| `zeros` | Matriz de ceros |
| `ones` | Matriz de unos |
| `rand` | Matriz de elementos uniformemente distribuidos |
| `randn` | Matriz de elementos normalmente distribuidos |

- Ejemplos

```
Z = zeros(2,4)
Z =
     0     0     0     0
     0     0     0     0

F = 5*ones(3,3)
F =
     5     5     5
     5     5     5
     5     5     5

N = fix(10*rand(1,10))
N =
     4     9     4     4     8     5     2     6
           8     0

R = randn(4,4)
R =
    1.0668    0.2944   -0.6918   -1.4410
    0.0593   -1.3362    0.8580    0.5711
   -0.0956    0.7143    1.2540   -0.3999
   -0.8323    1.6236   -1.5937    0.6900
```





- El comando load permite leer ficheros binarios que contienen matrices generadas en sesiones anteriores de MATLAB
- También permite leer ficheros de texto que contienen datos.
  El fichero debe estar organizado como una tabla de numeros separados por espacios, una línea por cada fila, e igual número de elementos en cada fila.
- Ejemplo: Crear utilizando un editor de texto un fichero llamado `magik.dat` que contenga los siguientes datos

```
16.0    3.0    2.0   13.0
 5.0   10.0   11.0    8.0
 9.0    6.0    7.0   12.0
 4.0   15.0   14.0    1.0
```

El comando

```
load magik.dat
```

crea una variable llamada magik conteniendo la matriz.

- Los ficheros m son ficheros de texto que contienen código de MATLAB.
- Para crear una matriz haciendo uso de un fichero m, editar un fichero llamado magik.m con el siguiente texto

```
A = [ ...
16.0     3.0     2.0    13.0
 5.0    10.0    11.0     8.0
 9.0     6.0     7.0    12.0
 4.0    15.0    14.0     1.0 ];
```

Ejecutar ahora el comando

```
magik
```

Comprobar que se ha creado la matriz A.





- Es el proceso de unir dos o más matrices para formar otra matriz de mayor dimensión
- El operador concatenación es []
- Ejemplo:

```
B = [A   A+32; A+48   A+16]
B =
    16     3     2    13    48    35    34    45
     5    10    11     8    37    42    43    40
     9     6     7    12    41    38    39    44
     4    15    14     1    36    47    46    33
    64    51    50    61    32    19    18    29
    53    58    59    56    21    26    27    24
    57    54    55    60    25    22    23    28
    52    63    62    49    20    31    30    17
```

Comprobar que las columnas de esta matriz suman todas lo mismo, pero no ocurre lo mismo con sus filas.

## Borrado de filas y columnas

- Se pueden borrar filas y columnas utilizando el operador [].
  es la matriz vacía (concatenación de nada).
- El proceso es sustituir una fila o una columna por la matriz vacía [].
- Ejemplo: Borrado de la segunda columna de una matriz

```
X = A;
X(:,2) = []
X =
    16     2    13
     5    11     8
     9     7    12
     4    14     1
```

- No se pueden borrar elementos, por que el resultado ya no sería una matriz

```
X(1,2) = []
```

  produciría un error.
  Sin embargo, utilizando un único subíndice es posible borrar elementos, aunque el resultado ya no sería una matriz, sino un vector.

```
X(2:2:10) = []
X =
    16     9     2     7    13    12     1
```



## El comando format

- Este comando controla el formato numérico de los resultados que muestra MATLAB.
- Afecta sólo a la presentación en pantalla, no al formato interno ni a los cálculos.
- Ejemplos:

```
x = [4/3 1.2345e-6]
format short
   1.3333    0.0000
format short e
   1.3333e+000  1.2345e-006
format short g
   1.3333  1.2345e-006
format long
   1.33333333333333   0.00000123450000
format long e
   1.333333333333333e+000    1.234500000000000e
       -006
format long g
   1.33333333333333                  1.2345e-006
```

```
format bank
   1.33            0.00

format rat
   4/3           1/810045

format hex
   3ff5555555555555    3eb4b6231abfd271
```

- Además `format compact` suprime espacios y líneas en blanco. Para obtener más control sobre la presentaciín en pantalla se pueden utilizar las funciones `sprintf` y `fprintf`.
- Para que no aparezca el resultado de un cálculo en la pantalla, se utiliza ;

```
A = magic(100);
```

- Para dividir expresiones que no caben en una única línea, se usan tres puntos ...

```
s = 1 -1/2 + 1/3 -1/4 + 1/5 - 1/6 + 1/7 ...
      - 1/8 + 1/9 - 1/10 + 1/11 - 1/12;
```





| | | |
|---|---|---|
| ↑ | ctrl-p | Comando anterior |
| ↓ | ctrl-n | Comando siguiente |
| ← | ctrl-b | Carácter atrás |
| → | ctrl-f | Carácter adelante |
| ctrl-→ | ctrl-r | Palabra adelante |
| ctrl-← | ctrl-l | Patabra atrás |
| home | ctrl-a | Ir a comienzo de línea |
| end | ctrl-e | Ir a fin de línea |
| esc | ctrl-u | Borrar línea |
| del | ctrl-d | Borrar carácter actual |
| backspace | ctrl-h | Borrar carácter anterior |
| | ctrl-k | Borrar hasta fin de línea |

MATLAB dispone de recursos para mostrar vectores y matrices en gráficos, así como para incluir texto en los gráficos e imprimirlos.

- La función básica de creación de gráficos es `plot`.
- Si `y` es un vector, `plot(y)` dibuja un gráfico de los valores de los elementos de `y` frente a sus índices.
- Si `x` e `y` son dos vectores de igual tamaño, `plot(x,y)` dibuja un gráfico de los valores de los elementos de `y` frente a los de `x`.
- Ejemplo:

```
t = 0:pi/100:2*pi;
y = sin(t);
plot(t,y)
```





- Se pueden crear gráficos múltiples con una única llamada a `plot`. MATLAB elige los colores automáticamente siguiendo una tabla predefinida.
- Ejemplo:

```
y2 = sin(t-.25);
y3 = sin(t-.5);
plot(t,y,t,y2,t,y3)
```

- Se puede especificar el color, tipo de línea, y marcas con el comando

```
plot(x,y,'color_style_marker')
```

- `color_style_marker` es una cadena de tres caracteres, que indican respectivamente el color, tipo de línea y marca.
- La letra que indica el color puede ser: `'c'`, `'m'`, `'y'`, `'r'`, `'g'`, `'b'`, `'w'`, `'k'`, que indican cyan, magenta, amarillo, rojo, verde, azul, blanco y negro.
- La letra que indica el tipo de línea puede ser: `'-'` para línea continua, `'--'` para línea de trazos, `':'` para línea de puntos, `'-.'` para punto y raya, `'none'` sin línea.
- Las marcas más comunes son `'+'`, `'o'`, `'*'` y `'x'`.
- Ejemplo: El comando

```
plot(x,y,'y:+')
```

dibuja el gráfico en línea continua amarilla y situa marcas `'+'` en cada punto.





Existen varias formas de obtener ayuda en línea de MATLAB.

- El comando `help`
- La ventana de ayuda
- El escritorio de ayuda (MATLAB help desk)
- Páginas de referencia en línea
- Página Web de The Mathworks, Inc. (`www.mathworks.com`)

- Es el comando más básico para obtener información de la sintaxis y actuación de una función.
- La información aparece directamente sobre la ventana de comandos.
- Ejemplo:

```
help magic

MAGIC  Magic square.
    MAGIC(N) is an N-by-N matrix constructed from
    the integers 1 through N^2 with equal row,
    column, and diagonal sums.
    Produces valid magic squares for N =
        1,3,4,5....
```

- El nombre de la función siempre aparece en mayúsculas, pero en realidad debe escribirse en minúsculas al llamar a la función



- Las funciones están organizadas en grupos lógicos, así como la estructura de directorios de MATLAB.
- Las funciones de álgebra lineal están en el directorio matfun. Para listar todas las funciones de este grupo

```
help matfun

Matrix functions - numerical linear algebra.

Matrix analysis.
  norm          - Matrix or vector norm.
  normest       - Estimate the matrix 2-norm
...
```

- El comando `help` lista todos los grupos de funciones

```
help

matlab/general
matlab/ops
...
```

## La ventana de ayuda

- Disponible seleccionando la opción `Help Window` del menú `Help` o bien pulsando la interrogación de la barra de menú.
- Puede invocarse desde la ventana de comandos con `helpwin`
- Para obtener ayuda sobre un comando `helpwin comando`
- La información obtenida es la misma que con el comando `help` pero permite hipertexto y navegación



## El comando lookfor

- Conveniente cuando buscamos una función pero no recordamos su nombre.
- Busca todas las funciones que en la primera línea de texto de la ayuda (línea H1) contienen la palabra clave.
- Ejemplo: Estamos buscando una función para invertir matrices, hacemos

```
help inverse
inverse.m not found.
```

entonces bucamos con `lookfor`

```
lookfor inverse
INVHILB  Inverse Hilbert matrix.
ACOSH    Inverse hyperbolic cosine.
ERFINV   Inverse of the error function.
INV      Matrix inverse.
PINV     Pseudoinverse.
IFFT     Inverse discrete Fourier transform.
IFFT2    Two-dimensional inverse discrete Fourier
    transform.
ICCEPS   Inverse complex cepstrum.
IDCT     Inverse discrete cosine transform.
```

- Con la opción `-all` busca en todo el texto de la ayuda, no solo en H1.

- El escritorio de ayuda de MATLAB permite acceder a mucha información de referencia almacenada en el disco duro o en el CD-ROM en formato HTML mediante un navegador.
- Se accede a través de la opción `Help Desk` del menú `Help`.
- También se accede escribiendo `helpdesk` en la ventana de comandos.
- Para acceder a la página de referencia en formato HTML de un comando específico, se utiliza el comando `doc`. Ejemplo: `doc eval`.
- Las páginas de referencia se encuentran también disponibles en formato PDF y pueden ser consultadas e impresas con Acrobat Reader.
- Finalmente, desde el escritorio de ayuda se puede acceder a la Página Web the The MathWorks, Inc.





- El entorno de MATLAB incluye el conjunto de variables definidas durante una sesión de MATLAB y el conjunto de ficheros del disco que contienen programas y datos y que permanecen entre sesiones.
- El espacio de trabajo (workspace) es el área de memoria accesible desde la línea de comandos de MATLAB.
- Los comandos `who` y `whos` muestran el contenido del espacio de trabajo, `who` proporciona una lista reducida, `whos` incluye además información sobre tamaño y almacenamiento.

```
whos

  Name        Size              Bytes  Class
  A           4x4                 128  double array
  D           5x3                 120  double array
  M          10x1                3816  cell array
  S           1x3                 442  struct array
  h           1x11                 22  char array
  n           1x1                   8  double array
  s           1x5                  10  char array
  v           2x5                  20  char array

Grand total is 471 elements using 4566 bytes.
```

- Para borrar variables del espacio de trabajo, usar el comando `clear`.

## El comando save

- Permite almacenar los contenidos del espacio de trabajo en un fichero MAT (binario).

```
save 15oct02
```

  salva el espacio de trabajo en el fichero `15oct02.mat`. Para salvar únicamente ciertas variables

```
save 15oct02 nombres_variables
```

- Para recuperar el espacio de trabajo se utiliza el comando `load`.

```
load 15oct02
```

- El formato MAT es binario y no puede leerse, si se desea un fichero que pueda leerse pueden utilizarse las siguientes alternativas

| | |
|---|---|
| -ascii | Formato de texto de 8 bits. |
| -ascii -double Use | Formato de texto de 16 bits. |
| -ascii -double -tabs | Delimita los elementos de una matriz con tabuladores |
| -v4 | Crea un fichero MAT de la versión 4 |
| -append | Añade datos a un fichero MAT ya existente |

- En formato texto no puede salvarse todo el espacio de trabajo de una vez, y debe hacerse indicando el nombre de las variables.



## La trayectoria de búsqueda

- La trayectoria de búsqueda (search path) es la lista ordenada de directorios en los que MATLAB va buscando las funciones.
- El comando `path` muestra la trayectoria de búsqueda.
- Si hubiera varios ficheros con el mismo nombre de función en diferentes directorios, MATLAB ejecuta el primero que encuentra al seguir la trayectoria de búsqueda.
- Para modificar la trayectoria de búsqueda, ir a `Set Path` en el menú `File`.

MATLAB dispone de los comandos `dir`, `type`, `delete`, `cd`, para realizar las operaciones usuales de manipulación de ficheros de un sistema operativo.

| MATLAB | MS-DOS | UNIX | VAX/VMS |
|---|---|---|---|
| dir | dir | ls | dir |
| type | type | cat | type |
| delete | del ó erase | rm | delete |
| cd | chdir | cd | set default |





- Crea un diario de la sesión MATLAB en un fichero de texto.
- El fichero puede editarse con cualquier editor o procesador de textos.
- Para crear un fichero llamado `midiario.txt` que contenga todos los comandos de la sesión y sus resultados en la ventana de comandos, hacer

```
diary midiario.txt
```

  si no se incluye ningún nombre de fichero, el diario de la sesión se almacena por defecto en el fichero `diary`.
- Para parar la grabación del diario

```
diary off
```

- Para volver a activar/desactivar la grabación del diario

```
diary on/off
```

Para ejecutar programas externos a MATLAB desde la línea de comandos, se antepone el carácter de escape `!`. Por ejemplo, en UNIX

```
! vi
```

Ejecuta el editor de texto visual.



## Ejercicios 1

`Fichero Aejer1.m :` Las siguientes expresiones describen las tensiones principales de contacto en las direcciones $x$, $y$ y $z$ que aparecen entre dos esferas que se presionan entre sí con una fuerza $F$.

$$\sigma_x = \sigma_y = -p_{\text{máx}} \left[ \left(1 - \frac{z}{a}\tan^{-1}\left(\frac{a}{z}\right)\right)(1-\nu_1) - 0{,}5\left(1+\frac{z^2}{a^2}\right)^{-1} \right]$$

$$\sigma_z = \frac{-p_{\text{máx}}}{1+z^2/a^2}$$

siendo

$$a = \left( \frac{3F}{8} \frac{(1-\nu_1^2)/E_1 + (1-\nu_2^2)/E_2}{1/d_1 + 1/d_2} \right)^{1/3}$$

$$p_{\text{máx}} = \frac{3F}{2\pi a^2}$$

$\nu_j$ son los coeficientes de Poisson, $E_j$ los módulos de Young de cada esfera y $d_j$ son los diámetros de las dos esferas.
Escribir las ecuaciones en notación de MATLAB y evaluarlas para los siguientes valores: $\nu_1 = \nu_2 = 0{,}3$, $E_1 = E_2 = 3 \cdot 10^7$, $d_1 = 1{,}5$, $d_2 = 2{,}75$, $F = 100$ lb. y $z = 0{,}01$ in.

## Ejercicio 2

`Fichero Aejer2.m :` El número de carga de un rodamiento hidrodinámico esta dado por la siguiente expresión:

$$N_L = \frac{\pi\epsilon\sqrt{\pi^2(1-\epsilon^2)+16\epsilon^2}}{(1-\epsilon^2)^2}$$

siendo $\epsilon$ el coeficiente de excentricidad. Escribir la ecuación en notación de MATLAB y evaluarla para $\epsilon = 0{,}8$.



## Ejercicio 3

`Fichero Aejer3.m :` Un tubo largo con radio interior $a$ y radio exterior $b$ y diferentes temperaturas en la superficie interior $T_a$ y en la exterior $T_b$ está sometido a tensiones.
Las tensiones radial y tangencial se obtienen mediante las siguientes ecuaciones:

$$\sigma_r = \frac{\alpha E(T_a - T_b)}{2(1-v)\ln(b/a)}\left[\frac{a^2}{b^2-a^2}\left(\frac{b^2}{r^2}-1\right)\ln\left(\frac{b}{a}\right)-\ln\left(\frac{b}{r}\right)\right]$$

$$\sigma_t = \frac{\alpha E(T_a - T_b)}{2(1-v)\ln(b/a)}\left[1-\frac{a^2}{b^2-a^2}\left(\frac{b^2}{r^2}+1\right)\ln\left(\frac{b}{a}\right)-\ln\left(\frac{b}{r}\right)\right]$$

siendo $r$ la coordenada radial del tubo, $E$ el módulo de Young del material del tubo y $\alpha$ el coeficiente de dilatación. La distribución de temperaturas a lo largo de la pared del tubo en la dirección radial es:

$$T = T_b + \frac{(T_a - T_b)\ln(b/r)}{\ln(b/a)}$$

Escribir las ecuaciones en notación de MATLAB y evaluarlas para los siguientes valores: $\alpha = 1{,}2\cdot 10^{-5}$, $E = 3\cdot 10^7$, $v = 0{,}3$, $T_a = 500$, $T_b = 300$, $a = 0{,}25$, $b = 0{,}5$, $r = 0{,}375$.

## Ejercicio 4

`Fichero Aejer4.m :` La fórmula siguiente, propuesta por el matemático S. Ramanujan permite aproximar el valor de $\pi$.

$$\frac{1}{\pi} = \frac{\sqrt{8}}{9801} \sum_{n=0}^{N\to\infty} \frac{(4n)!(1103 + 26390n)}{(n!)^4 396^{4n}}$$

Evaluar la formula anterior para $N = 0, 1, 2, 3$ y comparar el resultado obtenido con el valor de $\pi$ que proporciona MATLAB. Para calcular el factorial, utilizar la función `gamma` que satisface `gamma(n+1)=n!`.



## Ejercicio 5

`Fichero Aejer5.m :` Introducir en el espacio de trabajo de MATLAB dos vectores $a$ y $b$ siendo $a_j = 2j - 1$ y $b_j = 2j + 1$, $j = 1, \ldots, 7$. Se pide:

1. Calcular la suma de $a$ y $b$
2. Calcular la diferencia de $a$ y $b$.
3. Calcular el producto $a^T b$ y el valor de su traza y determinante.
4. Calcular el producto $ab^T$.

## Ejercicio 6

`Fichero Aejer6.m :` Sea `z=magic(5)`. Realizar las siguientes operaciones ordenadamente y mostrar los resultados:

1. Dividir todos los elementos de la segunda columna por $\sqrt{3}$.
2. Sustituir la última fila por el resultado de sumarle los elementos de la tercera fila.
3. Sustituir la primera columna por el resultado de multiplicarle los elementos de la cuarta columna.
4. Hacer que todos los elementos de la diagonal principal sean 2.
5. Asignar el resultado obtenido a la variable `q` y mostrarla por pantalla.
6. Mostrar la diagonal principal de $qq^T$.
7. Mostar el cuadrado de todos los elementos de la matriz $q$.



## Ejercicio 7

`Fichero Aejer7.m :` En análisis de regresión lineal multivariante aparece la siguiente cantidad:

$$H = X(X^T X)^{-1} X^T$$

Sea

$$X = \begin{bmatrix} 17 & 31 & 5 \\ 6 & 5 & 4 \\ 19 & 28 & 9 \\ 12 & 11 & 10 \end{bmatrix}$$

Calcular la diagonal de $H$.

## Ejercicio 8

`Fichero Aejer8.m :` Dibujar el resultado de la suma de las siguientes series para los rangos indicados de valores de $\tau$. Utilizar 200 puntos para realizar la gráfica.

1. Onda cuadrada

$$f(\tau) = \frac{4}{\pi}\sum_{k=1}^{\infty}\frac{1}{n}\sin(2(2k-1)\pi\tau), \quad -\frac{1}{2} \leq \tau \leq \frac{1}{2}$$

2. Diente de sierra

$$f(\tau) = \frac{1}{2} + \frac{1}{\pi}\sum_{k=1}^{\infty}\frac{1}{n}\sin(2k\pi\tau), \quad -1 \leq \tau \leq 1$$

3. Diente de sierra

$$f(\tau) = \frac{1}{2} - \frac{1}{\pi}\sum_{k=1}^{\infty}\frac{1}{n}\sin(2k\pi\tau), \quad -1 \leq \tau \leq 1$$

4. Onda triangular

$$f(\tau) = \frac{\pi}{2} - \frac{4}{\pi}\sum_{k=1}^{\infty}\frac{1}{(2k-1)^2}\cos((2k-1)\pi\tau), \quad -1 \leq \tau \leq 1$$



## Ejercicio 9

`Fichero Aejer9.m :` Dibujar las siguientes curvas. Utilizar `axis equal` para una correcta visualización.

1. Cicloide ($-\pi \leq \phi \leq 3\pi$, $r = 0{,}5, 1, 1{,}5$)

$$\begin{aligned} x &= r\phi - \sin\phi \\ y &= r - \cos\phi \end{aligned}$$

2. Lemniscata ($-\pi/4 \leq \phi \leq \pi/4$)

$$\begin{aligned} x &= \cos\phi\sqrt{2\cos(2\phi)} \\ y &= \sin\phi\sqrt{2\cos(2\phi)} \end{aligned}$$

3. Espiral ($0 \leq \phi \leq 6\pi$)
   - de Arquímedes

$$\begin{aligned} x &= \phi\cos\phi \\ y &= \phi\sin\phi \end{aligned}$$

   - Logarítmica ($k = 0{,}1$)

$$\begin{aligned} x &= e^{k\phi}\cos\phi \\ y &= e^{k\phi}\sin\phi \end{aligned}$$

## Ejercicio 10

`Fichero Aejer10.m :` Dibujar las siguientes curvas. Utilizar `axis equal` para una correcta visualización.

1. Cardioide ($0 \leq \phi \leq 2\pi$)

$$\begin{aligned} x &= 2\cos\phi - \cos 2\phi \\ y &= 2\sin\phi - \sin 2\phi \end{aligned}$$

2. Astroide ($0 \leq \phi \leq 2\pi$)

$$\begin{aligned} x &= 4\cos^3\phi \\ y &= 4\sin^3\phi \end{aligned}$$

3. Epicicloide ($R = 3$, $a = 0{,}5, 1ó2$, y $0 \leq \phi \leq 2\pi$)

$$\begin{aligned} x &= (R+1)\cos\phi - a\cos(\phi(R+1)) \\ y &= (R+1)\sin\phi - a\sin(\phi(R+1)) \end{aligned}$$

4. Epicicloide ($R = 2{,}5$, $a = 2$, y $0 \leq \phi \leq 6\pi$)

$$\begin{aligned} x &= (R+1)\cos\phi - a\cos(\phi(R+1)) \\ y &= (R+1)\sin\phi - a\sin(\phi(R+1)) \end{aligned}$$



## Ejercicio 11

`Fichero Aejer11.m :` Dibujar las siguientes curvas tridimensionales. Utilizar `axis equal` para visualizar correctamente.

1. Hélice esférica ($c = 5{,}0$, $0 \leq t \leq 10\pi$)

$$\begin{aligned} x &= \sin(t/2c)\cos(t) \\ y &= \sin(t/2c)\sin(t) \\ z &= \cos(t/2c) \end{aligned}$$

2. Senoide sobre cilindro ($a = 10{,}0$, $b = 1{,}0$, $c = 0{,}3$, $0 \leq t \leq 2\pi$)

$$\begin{aligned} x &= b\cos(t) \\ y &= b\sin(t) \\ z &= c\cos(at) \end{aligned}$$

3. Senoide sobre esfera ($a = 10{,}0$, $b = 1{,}0$, $c = 0{,}3$, $0 \leq t \leq 2\pi$)

$$\begin{aligned} x &= \cos(t)\sqrt{b^2 - c^2\cos^2(at)} \\ y &= \sin(t)\sqrt{b^2 - c^2\cos^2(at)} \\ z &= c\cos(at) \end{aligned}$$

4. Espiral toroidal ($a = 0{,}2$, $b = 0{,}8$, $c = 20{,}0$, $0 \leq t \leq 2\pi$)

$$\begin{aligned} x &= [b + a\sin(ct)]\cos(t) \\ y &= [b + a\sin(ct)]\sin(t) \\ z &= a\cos(ct) \end{aligned}$$

## Introducción a entornos de trabajo (I)

- Entornos de trabajo con Matlab:
  - Espacio de trabajo *workspace*.
  - Ficheros de escritura *scripts* (`*.m`).
  - Ficheros de funciones de Matlab (`*.m`) y compiladas.
  - Objetos desarrollados en Matlab y en Java.
- *Workspace* en Matlab.
  - Uso de variables globales, *scripts*, funciones y objetos.
  - Ejemplo:

```
>> a=[1,2;3,4];
>> whos
>> inv(a);
```

- Ficheros de escritura *scripts*:
  - Ficheros (`*.m`) con ordenes iguales a las dadas en el *workspace*.
  - Las variables que utiliza son las globales del *workspace*.
  - Útiles para repetir la misma operación varias veces.

- Ficheros (*.m) de funciones:
  - Son ficheros en lenguaje interpretado de Matlab.
  - Sus variables son locales por defecto.
  - Paso a la función del valor de las variables.
  - Funciones propias de Matlab y Toolbox.
  - Ejemplo:

```
function c = myfile1(a,b)
c = sqrt((a.^2)+(b.^2))
```

  - Uso desde el *workspace*:

```
>> x = 7.5
>> y = 3.342
>> z = myfile(x,y)
>> whos
...
```

- Ficheros de funciones compilador en C/C++ o FORTRAN. Contienen cabecera especial para conexión con Matlab.
- Clases y objetos definidos en Matlab y Java. Una clase es un tipo de dato al que se puede asociar funciones propias y redefinir operadores.

```
>> s = tf('s'); get(s)
>> P= 1/(s+1); bode(P);
```





- Tipos de variables:

| Clase | Ejemplo | Descripción |
|---|---|---|
| array | `[1,2;3,4]; 5+6i` | Datos virtual ordenado por índices cuyos componentes son datos del mismo tipo. |
| char | `'Hola'` | Array de caracteres (cada carácter tiene 16 bits). |
| celda | `{17, 'hola', eye(2)}` | Dato virtual ordenado por índices cuyos componentes son arrays de distinto tipo. |
| struct | `a.dia=1; a.mes='julio'` | Dato virtual para almacenar datos por campos (estructura). Cada campo es un array o celda. |
| objeto | `tf(1,[1,1])` | Datos definido por el usuario con base a una estructura y con funciones asociadas. |

- Operadores:

| Oper. aritméticos | |
|---|---|
| + | Suma. |
| - | Resta. |
| .* | Multiplicación. |
| ./ | División derecha. |
| .\ | División izquierda. |
| : | Operador dos puntos. |
| .^ | Potencia. |
| .' | Transpuesta. |
| ' | Conjugada transpuesta. |
| * | Multiplicación de matrices. |
| / | División derecha de matrices. |
| \ | División izquierda de matrices. |
| ^ | Potencia de matrices. |

| Oper. de relación | |
|---|---|
| > | Menor que |
| > | Mayor que |
| <= | Menor que o igual a |
| >= | Mayor que o igual a |
| == | Igual a |
| = | No igual a |

| Operadores lógicos. | |
|---|---|
| & | Y |
| \| | OR |
| ~ | NO |





- Operaciones aritméticas en el *workspace* o `Ascript1.m`:

```
>> a=[1,2;3,4];   b=[4,5;6,7];
>> c= 3*a
...
>> a(1,:)*b(:,1)
...
```

```
>> c= a*b
...
>> c= a.*b
```

- Operaciones de relación en el *workspace* o `Ascript1.m`:

```
>> a=[1,2,3];   b=[1,3,2];
>> a==b
...
```

```
>> a > b
...
>> a=[]; isempty(a)
```

...

- Operaciones lógicas en el *workspace* o `Ascript1.m`:

```
>> a=[1,2,3]; b=[1,0,3];
>> a==b
...
>> a|b
...
```

```
>> a&b
...
>> ~a
...
```

- Funciones del directorio `elmat` que devuelven valores importantes,

| | |
|---|---|
| `ans` | Variable a la que se asigna el resultado de una expresión que no ha sido asignada. |
| `eps` | Tolerancia con la que trabaja Matlab en sus cálculos. |
| `realmax` | Mayor número en coma flotante que puede representar el computador. |
| `realmin` | Menor número en coma flotante que puede representar el computador. |
| `pi` | 3.1415926535897... |
| `i, j` | Números imaginarios puros. |
| `Inf` | Infinito. Se obtiene de divisiones entre cero. |
| `NaN` | Indeterminación. Se obtiene de divisiones `0/0`, `inf/inf` o `n/0` cuando n es imaginario. |
| `flop` | Cuenta las operaciones en coma flotante realizadas. |
| `version` | Indica la versión de Matlab usada. |





- En el *workspace*:

```
>> 3+2
...
>> pi*3
...
>> realmax
...
```

```
>> a= 5+3*i
...
>> a= i*(4+3*i)
...
>> a= 3/0
...
```

```
>> a= 0/0
...
>> a= NaN*3
...
>> a= 3*Inf
...
```

- Sentencias de control en Matlab:
  - `if`, `else` y `elseif`: Ejecuta un grupo de sentencias basándose en condiciones lógicas.
  - `switch`, `case` y `otherwise`: Ejecuta diferentes grupos de sentencias en función de condiciones lógicas.
  - `while`: Ejecuta un número de sentencias de forma indefinida en función de una sentencia lógica.
  - `for`: Ejecuta un número de sentencias un número determinado de veces.
  - `try...catch`: Cambia el control de flujo en función de los posibles errores producidos.
  - `break`: Termina de forma directa la realización de un bucle `for` o `while`.
  - `return`: Sale de la función.
- Nota: Los bucles `for` y `while` pueden ser modificados por código vectorizado para aumentar la velocidad de ejecución.





- Forma general `Asript2.m`,

```
if sent_lóg_1,
   % bloque_1
elseif sent_lóg_2,
   % bloque_2
else
   % bloque_3
end
```

```
if n < 0
% Si n negativo error.
  disp('Entrada debe se positiva');
elseif rem(n,2) == 0
% Si es par se divide entre 2.
  A = n/2;
else
% Si es impar se incrementa y divide.
  A = (n+1)/2;
end
```

## Sentencia de control *switch*, *case* y *otherwise*.

- Formulación general `Ascript3.m`,

```
switch expression
case value1
    %  bloque_1
case value2
    %  bloque-2
. . .
otherwise
    %  bloque-n
end
```

```
switch input_num
case {-1, -2, -3}
    disp('-1 ó -2 ó -3');
case 0
    disp('cero');
case {1, 2, 3}
    disp('1 ó 2 ó 3');
otherwise
    disp('otro valor');
end
```



## Sentencia de control *while* y *for*.

- Formulación general `Ascript4.m`,

```
while expresión
    % bloque
end
```

```
n = 1;
while prod(1:n) < 1e100,
  n = n + 1;
end
```

- Formulación general `Ascript5.m`,

```
for índice= inicio:paso:fin,
    % bloque
end
```

```
for ii = 2:6,
  x(ii) = 2*x(ii-1);
end
```

## Sentencia de control: *break*, *try-catch* y *return*

- Sentencia `break`:
  - Sirve para salir de forma automática del último bucle `while` o `for` abierto sin tener en cuenta la condición o índice de salida.
- Sentencia de control `try`, `catch`:
  - Formulación general,
    `try` *bloque-1* `catch` *bloque-2* end
  - Ejecuta el *bloque-1* mientras no haya un error. Si se produce un error en *bloque-1* se ejecuta *bloque-2*.
- Sentencia `return`:
  - Se sale de la función en la que se trabaja.
  - Si se llega al final de la función (`*.m`), Matlab sale de ella automáticamente.



## Funciones en matlab (I): Cabecera

- Se define el nombre y las variables de entrada y salida:

```
function c = myfile(a,b)
```

- Las líneas de comentario se inician con el carácter %.
- Las líneas de comentario posteriores a la función son de ayuda.

```
function c = myfile(a,b)
% Output: c. Input: a y b
```

- Usando la función `help`.

```
>> help myfile
Output: c. Input: a y b
```

- Variables de entrada-salida:
  - Libertad en su número. La variable `nargin` y `nargout` indican su número.
  - Variables locales por defecto sin tipo determinado.
  - Ejemplo: `a`, `b` y `c` pueden ser double o array `myfile2.m`.

```
function c = myfile2(a,b,c)
%Output: c. Input: a, b y c
if nargin<2,
  error('c = myfile(a,b,[c])');
elseif nargin==2,
  c= sqrt(a.^2+b.^2);
else
  c= sqrt(a.^2+b.^2+c.^2);
end
```





- Una celda `varargin` como variable de entrada y otra `varargout` como salida. Ejemplo `myfile3.m`:

```
function c = myfile3(varargin)
%Output: c. Input: a, b y c
if nargin<2,
  error('c = myfile(a,b,[c])');
elseif nargin==2,
  c= sqrt(varargin{1}.^2+ varargin{2}.^2);
else
  c= sqrt(varargin{1}.^2+ varargin{2}.^2+ varargin
      {3}.^2);
end
```

## Funciones en matlab (IV): Variables globales y estáticas

- Variable estática: No se pierde su valor y sólo se puede usar en la función definida.
- Variable global: No se pierde su valor y se puede usar en todas las funciones donde esté definida. Debe estar definida en el *workspace*.
- Ejemplo función `myfile4.m`:

```
function c = myfile4(a)
% Output: c. Input: a
global P;
if isempty(P), P=1; end
c= P*a; P= P+1;
```

- Variable estática:

```
>> P= 10; z= myfile(3); % repetir
```

- Variable global:

```
>> global P; P=10; z= myfile(3); % repetir
```



## Funciones en matlab (V): Sub-funciones y funciones privadas

- Varias funciones contenidas en un mismo fichero.
- La función principal es la primera. Equivalente a la función `main` del lenguaje C.
- Ejemplo `myfile5.m`:

```
function c = myfile5(a,b)
% Output: c. Input: a y b
  c= fun(a,b);

function z= fun(x,y)
z=sqrt(x.^2+y.^2);
```

- Funciones privadas: Están en sub-carpeta `private` y sólo se pueden usar por las funciones de la carpeta.
- Prioridades en la llamada a funciones: Sub-función, función en misma carpeta, función en carpeta `private`,función en las carpetas del `path`.

- Son el equivalente a los punteros a funciones de lenguaje C.
- `eval()`: Una cadena de caracteres es interpretada como orden,

```
>> cad= 'myfile'; a= 1;
>> c= eval([cad, '(a,', int2str(2), ')']);
```

- `feval`: Se llama a una función por su nombre o comodín `Ascript6.m`,

```
>> cad= 'myfile'; a= 1, b=2;
>> c= feval(cad, a, b);
>> cad= @myfile; a=1, b=1;
>> c= feval(cad, a, b);
>> cad= @(x,y) sqrt(x.^2+y.^2);
>> c= feval(cad, a, b);
```





- `input()`: Introducción de datos `Ascript7.m`,

```
n= input('Intr. dato:'); %Double.
n= input('Intr. dato:','s'); %Cadena de caracteres.
```

- `ginput()`: Localizar puntos en una gráfica con el ratón,

```
figure; plot(1:1000);
[x,y]= ginput(1) %localizar un punto x, y en gráfica.
[x,y,tecla]= ginput(1) %tecla da la tecla del ratón
    usada.
```

- `pause()`: La función para el programa durante un periodo de tiempo,

```
pause(n); %Para el programa durante n segundos.
pause; %Para el programa hasta que se pulse una tecla.
```

- Llamada a la *shell* (MS-DOS o LINUX): Iniciar sentencia con `!`,

```
! copy fich1.c fich2c. %Si el sistema fuera msdos
```

- `save`: Grabar datos en ficheros `Ascript8.m`,
  - `-mat`: Código binario (por defecto).
  - `-ascii`: Código ASCII.
  - `-append`: Graba al final del fichero.

```
>> save datos.dat a b c % Graba 'datos.dat' las
    variables a b c
>> a= rand(10,5);
>> save -ascii -append datos.dat a %Graba al final del
    fichero en código ASCII.
```

- `load`: Recupera las variables guardadas con la sentencia `save`.

```
>> load datos.dat % Recupera las variables de datos.dat
```



## FUNCIONES DE LIBRERÍA ENTRADA/SALIDA DE LENGUAJE C (I)

- Algunas de las funciones de entrada/salida:

| **Clase** | **Función** | **Descripción** |
|---|---|---|
| Abrir/Cerrar | `fopen()` | Abrir fichero. |
| | `fclose()` | Cerrar fichero. |
| Binarios I/O | `fread()` | Lectura binaria de fichero (defecto enteros). |
| | `fwrite()` | Escritura binaria en fichero (defecto enteros). |
| Con formato | `fscanf()` | Lectura con formato de fichero. |
| | `fprintf()` | Escritura con formato en fichero. |
| Conversión cadenas | `sscanf()` | Lee de cadena con un determinado formato. |
| | `sprintf()` | Escribe en cadena con formato. |

- Ejemplos de apertura y cierre. Permisos:
  - 'r': Lectura. Puntero al inicio del fichero.
  - 'w': Escritura. Se borra el fichero si existe.
  - 'a': Añadir. Puntero al final del fichero.
  - 'r+': Lectura/escritura. Puntero al inicio.

```
>> fic= fopen('fich.dat','r'); % Abre fichero para
    lectura.
>> fclose(fic);  % Cierra fichero 'fich.dat'.
>> fclose('all'); % Cierra todos los ficheros.
```

- Principales usos:
  - Ficheros de texto con formato
  - Ficheros binarios para guardar o extraer matrices en su forma vectorial.





- Ejemplo Ascript9.m:

```
>> a= rand(3,3)
>> fich= fopen('datos.txt','w'); % Guardar en texto
>> fprintf(fich, '%.2f %.2f %.2f\n', a);
>> fclose(fich);
>> fich= fopen('datos.txt','r'); % Recuperar de fichero
     texto
>> b= fscanf(fich,'%f')
>> fclose(fich);
```

```
>> fich= fopen('datos.txt','w'); % Guardar en binário,
    formato real*4
>> fwrite(fich, a, 'real*4');
>> fclose(fich);
>> fich= fopen('datos.txt','r'); % Recuperar en binário
>> b= fread(fich,inf,'real*4')
>> fclose(fich);
```

- `Fichero Bejer1.m :` Generar una función (`*.m`) para obtener las siguientes series matemáticas. Los argumentos son tres: El primero es el nombre de la serie deseada (obligatorio). El segundo es el número de datos de $\tau$, por defecto 200 (opcional). El tercero es límite superior de sumatorio, por defecto 1000 (opcional). Si el número de argumentos de salida es uno se devuelve los datos, si es cero se dibuja la gráfica correspondiente.
  - Señal cuadrada:
  $$f(\tau) = \frac{4}{\pi} \sum_{n=1,3,5,\ldots} \frac{1}{n} \sin(2n\pi\tau) \quad -\frac{1}{2} \leq \tau \leq \frac{1}{2}$$
  .
  - Dientes de sierra:
  $$f(\tau) = \frac{1}{2} + \frac{1}{\pi} \sum_{n=1} \frac{1}{n} \sin(2n\pi\tau) \quad -1 \leq \tau \leq 1$$
  .
  - Señal triangular:
  $$f(\tau) = \frac{\pi}{2} - \frac{4}{\pi} \sum_{n=1} \frac{1}{(2n-1)^2} \cos((2n-1)\pi\tau) \quad -1 \leq \tau \leq 1$$
  .





- `Fichero Bejer2.m :` El desplazamiento de una onda propagada a lo largo de una cuerda tiene una velocidad inicial cero y un deplazamiento inicial,
$\{u(\eta,0) = \frac{\eta}{a} \mid 0 \leq \eta \leq a\} \quad \{u(\eta,0) = \frac{1-\eta}{1-a} \mid a \leq \eta \leq 1\}$,
siendo su ecuación,
$u(\eta,\tau) = \frac{2}{a\pi(1-a)} \sum_{n=1}^{N\to\infty} \frac{\sin n\pi a}{n^3} \sin(n\pi\tau)\cos(n\pi\eta)$.
Crear una función `*.m` para mostrar en gráfico $u(\eta,\tau)$. La entrada de la función será el valor de $a$, opcional defecto $a = 0{,}25$, el de $N$, opcional defecto $N = 50$, y el de $\Delta\tau$, opcional defecto $\Delta\tau = 0{,}05$, donde $0 \leq \tau \leq 2$. La función dibuja la gráfica si el usuario no pide variables de salida y devuelve el valor de $u(\eta,\tau)$ sin dibujar la gráfica en caso contrario.

## Optimización de programas: Indexado de arrays y celdas (I)

- Para la optimización de un programa con matlab se debe reducir el número de bucles y cambiarlo por álgebra matricial.
- Formato externo: Filas y columnas. Formato interno: vector de columnas, `Ascript10.m`.

```
>> a= [1,2,3; 4,5,6];
>> a(2,1), a(2),
```

- Llamada parcial a un array, `end` cuenta el número de filas o columnas,

```
>> ii= 1:2:3; % vector de 1 a 3 con paso 2.
>> a(1, ii) % primera fila, columnas ii
>> a(1,2:end) % primera fila, columnas de 2 al final
>> a(1,:) % primera fila, todas las columnas
```

- Composición de arrays,

```
>> b= [a(:,1),[5,7]'] % Primera columna de a y [5,7]
    vector columna.
```

- Borrado de matrices,

```
>> b(:,1)=[]; % Borrado de la primera columna.
```

- Matrices ceros, unos y aleatorias,

```
>> a= zeros(2,3); b= ones(3,2); c= rand(2,3);
```

- Espacios lineales y logarítmicos,

```
>> a= linspace(1,10,100); % De 1 a 10, 100 puntos, paso
     lineal
>> a= logspace(1,5,100); % De 1e1 a 1e5, 100 puntos,
    paso log.
```

- Funciones de tamaño y repetición.

```
>> [nfil,ncol]= size(a); % Tamaño fila columna,
>> nfil= size(a,1); % Tamaño fila.
>> ncomp= size(a(:),1); % Número de componentes,
    formato interno.
>> b= repmat(a, [3,1]); % Repetir matriz 'a' tres veces
     en columna.
```





- Inicialización de matrices como matrices cero.
- Sustitución de bucles por productos matriciales, `Ascript11.m`
  - Función en diferentes puntos, $y(n) = \sin(n) * n$, $0 < n < 10$, 100 puntos:

```
>> n=linspace(0,10,100); y= sin(n).*n;
```

  - Sumatorio de función, $y = \sum_{n=0}^{10} \sin(n) * n$.

```
>> n= [0:10]'; y= sum(sin(n).*n);
```

  - Función de dos dimensiones en varios puntos, $y(i,j) = i^2 + j^2 + i * j$, $i \in [0,5]$, $j \in [0,7]$,

```
>> ii=0:5; jj=[0:7]';
>> sii= size(ii,2); sjj= size(jj,1);
>> ii= repmat(ii,[sjj,1]); jj= repmat(jj,[1,sii]);
>> [ii,jj]= meshgrid(ii,jj); % equivalente
>> y= ii.^2+jj.^2+ii.*jj;
```

- Sustitución de bucles por productos matriciales, `Ascript11.m`
  - Función de una dimensiones con sumatorio, $y(i) = \sum_{n=1}^{10} n * i^2 + i, \quad i \in [0, 5]$,

```
>> ii=0:5; n=[1:10];
>> y= n*ones(size(n))'*ii.^2+ii;
```

- `Práctica de optimización de programas:` Volver a escribir el código de las práctica de generación de señales sin usar bucles, `Ficheros Bejer1bis.m, Bejer2bis.m`.





- Se componen de filas, columnas y páginas.
- Generación de matrices tridimensionales, `Ascript12.m`

```
>> a=[1,2;3,4];   % Matriz de dos dimensiones.
>> a(:,:,2)= [5,6;7,8];  % Matriz de tres dimensiones.
>> a= cat(3,[2,3;4,5],[5,6;7,8]);  % encadena en dim 3
>> a= repmat([2,3;4,5],[1,1,2]);   % repite en páginas
```

- Re-dimensión: El array es tomado como vector y re-dimesionado,

```
>> a= reshape(a, [2,4]);  % Convierte dos páginas a
    cuatro columnas.
```

- Borrado de parte de la matriz,

```
>> a(:,:,2)=[];  % Borrado de la página 3.
```

## Matrices tri-dimensionales (II)

- Eliminación de dimensiones,

```
>> b= squeeze(a(:,1,1)); %Se obtiene un vector dim
    (2*1)
>> b= squeeze(a(1,:,1)); %Se obtiene un vector dim
    (1*2)
>> b= squeeze(a(1,1,:)); %Se obtiene un vector dim
    (2*1)
```

- Cambio de índices en dimensiones,

```
>> b= permute(a,[2,1,3]); %Las filas pasan a ser
    columnas.
>> a= ipermute(b,[2,1,3]); %Es la inversa de permute.
```



## Matrices multidimensionales (III)

- Celdas multidimensionales: Se puede trabajar con ellas de forma similar a como se trabaja con las matrices.

```
>> A= {[1,2;3,4],'hola'; [1,2,3],'2'}; %celda de dim
    (2*2)
>> B= {'hola',[1,2,3]; '2',2}; %celda de dim (2*2)
>> C= cat(3,A,B); %celda de dim (2*2*2)
```

- Estructuras multidimensionales: Se puede trabajar con ellas de la forma similar a como se trabaja con matrices.

```
>> clase(1,1,1).alum= 'pepe'; clase(1,1,1).nota=10;
>> clase(1,1,2).alum= 'juan'; clase(1,1,2).nota=10;
>> clase= squeeze(clase); %Se reduce a dos dimensiones
    .
>> clase.alum %Muestra los nombres de todos los
    alumnos.
```

## Funciones para estructuras y celdas (I)

- Funciones especificas para structuras.

| Función | Descripción |
|---|---|
| `getfield()` | Muestra los campos de la estructura. |
| `isfield()` | Verdadero si un campo de la estructura. |
| `isstruct()` | Verdadero si es una estructura. |
| `rmfield()` | Borra el campo marcado de la estructura. |
| `setfield()` | Cambia los contenidos de campo. |
| `struct()` | Crea o convierte en una matriz de estructuras. |
| `struct2cell()` | Convierte una matriz de estructuras en celdas. |



## Funciones para estructuras y celdas (II)

- Ejemplos de funciones para estructuras,

```
>> clase(1).alum= 'pepe'; clase(1).nota=10;
>> clase(2).alum= 'juan'; clase(2).nota=10;
>> clase(3)= struct('alum', 'josé', 'nota', 7) %Otra
    forma de definir
>> getfield(clase) %Muestra los campos de clase
>> isstruct(clase) %Afirmativo
>> isfield(clase, 'nota') %Afirmativo
>> rmfield(clase, 'nota') %Elimina campo nota.
>> setfield(clase, 'alum', 'pepe'); %Introduce 'pepe'
    en campo alum
>> p= struc2cell(clase)
>> %Pone un elemento de struct en una columna de la
    celda.
>> %De un vector estructura sale una matriz de celdas
```

## Funciones para estructuras y celdas (III)

- Funciones especificas de celdas.

| Función | Descripción |
|---|---|
| `cell()` | Crea una matriz de celda. |
| `cell2struct()` | Convierte celdas en estructuras. |
| `celldisp()` | Muestra el contenido de la celda. |
| `cellfun()` | Aplica una celda función a matriz. |
| `cellplot()` | Muestra una gráfica de la celda. |
| `iscell()` | Verdadero en caso de que sea celda. |
| `num2cell()` | Conversión de matriz numérica en celda. |



## Funciones para estructuras y celdas (IV)

- Ejemplos de funciones de estructuras.

```
>> a= cell(2,2) %Se crea una celda vacía.
>> a={'pepe', 'juan'; 10, 10}; %Se llena celda
>> iscell(a) %Afirmativo
>> celldisp(a) %Muestra el contenido de la celda
>> cellplot(a) %Muestra el contenido en ventana.
>> cellfun('isreal', a) %Diferentes funciones
    aplicadas a celdas.
>> cell2struc(a, {'alum', 'nota'}) %Pasa de celda a
    estructura.
>> %Toma los campos por filas.
>> num2cell([1,2;3,4]) %Convierte matriz en celda.
```

- Escribir un fichero `Fichero Bejer3.m` las siguientes operaciones:
  - Generar una matriz aleatoria de dimensiones $\{10 \times 5 \times 20\}$.
  - Obtener la matriz correspondiente a la segunda página.
  - Obtener el vector correspondiente a la fila 2, columna 3.
  - Obtener una celda cuyos componentes sean los elementos de la matriz.
  - Agregar dicha celda al campo `datos` de una estructura. Introducir otro campo llamado `nombre` que corresponda a una cadena de caracteres.
  - Salvar la matriz, celda y estructura en un fichero de nombre `datos.dat`.
  - Salvar los elementos de la matriz en un fichero binario usando `fwrite()`.
  - Recuperar dichos datos e introducirlos en una matriz de dimensión $\{5 \times 10 \times 20\}$.
  - Meter la primera página de esta matriz en un fichero de texto con formato 5 datos por línea.
  - Recoger estos datos línea a línea y reconstruir la matriz.



## Contenidos

## Funciones para gráficas en dos dimensiones (I)

| Función | *Comentario* |
|---|---|
| `figure` | Crea una figura |
| `subplot` | Crea varios ejes en la misma figura |
| `hold` | Superpone diferentes plots |
| `plot` | Plot lineal |
| `loglog` | Plot logarítmico |
| `semilogx, semilogy` | Plot semilogarítmico en eje x e y |
| `xlim, ylim , zlim` | Márgenes en cada uno de los ejes |
| `tit, xlabel, ylabel` | Texto en título y ejes |
| `legend, text, gtext` | Añadir texto en figura |
| `ginput` | Marcar posición en figura |
| `grid, box` | Mallado y caja en figura |



## Funciones para gráficas en dos dimensiones (II)

| Función | *Comentario* |
|---|---|
| `bar, bar3, bar3h` | Gráficas de barras |
| `errorbar` | Gráficas con barras que marcan el error |
| `compass` | Gráficas en forma de compás |
| `ezplot, ezpolar` | Gráfica sencillas de funciones |
| `fplot` | Gráficas de funciones |
| `hist, pareto` | Histograma y carta de pareto |
| `pie, pie3` | Pastel de dos o tres dimensiones |
| `stem, stairts` | Gráficas con impulsos y escaleras |
| `scatter, plotmatrix` | Gráficas de dispersión de datos y matrices |

## Ejemplos con gráficos de dos dimensiones, *Escript1.m* (I)

- Barras:

```
>> x= -2.9:0.2:2.9; bar(x,exp(-x.*x));
>> barh(x,exp(-x.*x));
>> y= round(rand(5,3)*10);
>> bar(y,'group'); bar(y,'stack');
```

- Histogramas:

```
>> y= randn(1e4, 1); hist(y); hist(y,20);
```

- Pasteles:

```
>> x=[1,3,0.5,2.5,2]; pie(x);
```

- Escaleras:

```
>> x= -3:0.1:3; stairs(x,exp(-x.^2));
```

- Barras con error:

```
>> x= -4:.2:4; y= (1/sqrt(2*pi))*exp(-(x.^2)/2);
>> e=rand(size(x))/10;
>> errorbar(x,y,e);
```



## Ejemplos con gráficos de dos dimensiones, *Escript1.m* (II)

- Puntos:

```
>> y=randn(50,1); stem(y);
```

- Histograma de los ángulos.

```
>> y= randn(1000,1)*pi; rose(y);
```

- Representación de números complejos:

```
>> z= eig(randn(20,20)); compass(z);
>> feather(z);
```

## Gráficas para funciones y complementos (I), *Escript2.m*.

- Gráficas funciones: `plot()` para números, `ezplot()`, `fplot()` para funciones:

```
>> x= 0:0.05:10; y= sin(x).*exp(-0.4*x);
>> figure;
>> subplot(3,1,1); plot(x,y); title('plot');
>> subplot(3,1,2);
>> ezplot('sin(x).*exp(-0.4*x)', [0,10]); title('ezplot
    ');
>> subplot(3,1,3);
>> fplot('sin(x).*exp(-0.4*x)', [0,10]); title('fplot')
    ;
```

`subplot(n,m,p)` divide la gráfica en $n \times m$ partes y va a la $p$.

- Texto y ejes en las gráficas: Títulos, legendas, cajas, mallado:

```
>> disp('Texto y ejes en gráficas:')
>> t= 0:0.1:2*pi; r=sin(2*t).*cos(2*t);
>> figure; subplot(2,1,1); polar(t,r);
>> title('polar')
>> subplot(2,1,2);
>> fplot('[sin(x), sin(2*x), sin(3*x)]', [0,2*pi],'-','
    o','*');
>> title('fplot')
>> legend('sin(x)', 'sin(2*x)', 'sin(3*x)');
```



## Gráficas para funciones y complementos (II), *Escript2.m*.

```
>> disp('Título, nombre en ejes, legenda:')
>> x= linspace(0,2,30); y= sin(x.^2); figure; plot(x,y);
>> text(1,.8,'y=sin(x^2)'); xlabel('Eje X'); ylabel('Eje Y')
    ;
>> title('Gráfico senoidal');

>> disp('Subplot, tamaño de letra, tamaño ejes, caja, grid:'
    )
>> x=0:.1:4*pi; y= sin(x); z=cos(x);
>> figure; subplot(1,2,1); plot(x,y);
>> axis([0,2*pi,-1,1]);
>> set(gca,'FontSize',12);
>> grid on; box on;
>> title('sin(x)','FontWeight','bold','FontSize',12);
>> subplot(1,2,2); plot(x,z);
>> axis([0,2*pi,-1,1]);  grid on; box on
>> set(gca,'FontSize',12);
>> title('cos(x)','FontWeight','bold','FontSize',12);
```

| Función | *Comentario* |
|---|---|
| plot3 | Plot lineal en tres dimensiones |
| mesh, meshc, meshz | Plot de mallados en tres dimensiones |
| surf, surfc, surfl | Plot de superfiecie en tres dimensiones |
| meshgrid, ndgrid | Preparación de datos para gráficas de superficie |
| hidden | Ocultar líneas y superficies ocultas |
| contour, contour3 | Curvas de nivel |
| trimesh, trisurf | Plot de mallado triangular |
| scatter3, stem3 | Diagramas de dispersión y impulsos en 3 dimensiones |
| slice | Gráficos de volumen |
| surfnorm | Normales de las superficies |
| quiver3 | Puntos y normales en vectores |
| patch | Parches de superficies |



## Ejemplos de gráficas en tres dimensiones, *Escript3.m* (I)

- Gráfica de tres dimensiones por puntos:

```
>> t= 0:pi/50:10*pi;
>> figure; plot3(sin(t),cos(t),t); grid on; axis square
>> figure; plot3(sin(t),cos(t),t,'-',cos(t),sin(t),t,'*
    ');
```

- Gráfica de tres dimensiones por polígonos:

```
>> z=0:0.01:8; x=cos(z); y=sin(z);
>> figure; fill3(x,y,z,'r');
```

- Gráficas de tres dimensiones con barras:

```
>> figure; y= cool(7);
>> subplot(1,3,1); bar3(y,0.2,'detached');
>> subplot(1,3,2); bar3(y,'grouped');
>> subplot(1,3,3); bar3(y,0.1,'stacked');
```

- Gráficas de tres dimensiones con puntos con base:

```
>> figure; x= linspace(0,1,10);
>> y=x./2; z=sin(x)+sin(y);
>> stem3(x,y,z,'fill');
```

## Ejemplos de gráficas en tres dimensiones, *Escript3.m* (II)

- Superficies tres dimensiones y contornos en dos y tres:

```
>> [X,Y]= meshgrid(-7.5:.5:7.5);
>> Z= sin(sqrt(X.^2+Y.^2))./sqrt(X.^2+Y.^2);
>> surf(X,Y,Z); figure; surfc(X,Y,Z);
>> figure; surfl(X,Y,Z);
>> contour(Z); contour3(Z,50);
```

- Superficie con velocidad:

```
>> figure; [U,V,W]= surfnorm(X,Y,Z);
>> quiver3(X,Y,Z,U,V,W,0.5);
```

- Contornos:

```
>> [X,Y]= meshgrid(-2:.2:2,-2:.2:3); Z= X.*exp(-X.^2-Y
    .^2);
>> contour(X,Y,Z); figure; contour(X,Y,Z,50);
>> figure; contourf(X,Y,Z);
```

- Cambio de color y perspectiva:

```
>> figure; sphere(16); axis square; shading flat;
>> set(gca, 'Zlim', [-0.6,0.6]); set(gcf,'Color','w');
```



## Ejemplos de gráficas en tres dimensiones, *Escript3.m* (III)

- Rotación de la figura:

```
>> h= surf(peaks(20)); rotate(h,[1,0,0],15);
>> view([10,10]);
```

- Mallado triangular de la base, no homogéneo. Superficie en función de ese mallado:

```
>> figure; x= rand(1,50); y= rand(1,50);
>> z= peaks(6*x-3, 6*x-3);
>> tri= delaunay(x,y); trimesh(tri,x,y,z);
>> figure; trisurf(tri,x,y,z);
```

- Representación en cuatro dimensiones, la cuarta es el color:

```
>> figure; load wind; cav= curl(x,y,z,u,v,w);
>> slice(x,y,z,cav,[90,134],[59],[0]);
```

- Escribir en un fichero `Fichero Cejer1.m` el código para obtener las siguientes gráficas,
  - Visualizar sobre el rango $-2$ a 2 la función $v = e^{-x^2-y^2-z^2}$.
  - Representar en el intervalo $[-8, 8]$ la función $f(x) = \frac{x^3}{x^2-4}$.
  - Graficar sobre los mismos ejes las funciones $bessel(1, x)$, $bessel(2, x)$ y $bessel(3, x)$ para valores entre 0 y 12, separados uniformemente entre sí dos décimas. Colocar tres leyendas y tres tipos de trazo diferentes (normal, asteriscos y círculos) respectivamente para las tres funciones.
  - Representar la curva en polares $r = 4(1 + cos(a))$ para $a$ entre 0 y $2\pi$ (cardiode). Representar también la curva en polares $r = 3a$ para $a$ entre $-4\pi$ y $4\pi$ (espiral).
  - Representar la curva alabeada de coordenadas paramétricas $x = \cos^2(t)$, $y = \sin(t)\cos(t)$ y $z = \sin(t)$ para $t$ entre $-4\pi$ y $4\pi$.
- Escribir en un fichero `Fichero Cejer2.m` el código para obtener las siguientes gráficas,
  - Representar la superficie, su gráfico de malla y su gráfico de contorno cuya ecuación es la siguiente:

$$x = xe^{-x^2-y^2} \qquad -2 < x, y < 2$$

  - Representar en un gráfico de curvas de nivel con 20 líneas la superficie de la ecuación $z = \sin(x)\sin(y)$ con $-2 < x, y < 2$.
  - Representar el paraboloide $x^2 + y^2$ seccionado por el plano $z = 2$.



## Contenidos

## Definición de clase y objeto

- Una clase es un nuevo tipo de dato, a una estructura, para el que se pueden definir funciones especificas y redefinir los operadores.
- Un objeto es un caso particular de una clase.
- Los campos de la estructura asociada a una clase se llamarán propiedades de la clase, y las funciones asociadas a una clase funciones método.

```
>> s= tf('s'); % Clase 'tf', objeto 's'.
>> P= 1/(s+1); % Operador '+' y '/' redefinidos para P
>> K= zpk([],[-1,-2],1); % Clase 'zpk', objeto K
>> bode(P); %'bode' método de la clase 'tf'
```

- Características de la programación a objeto:
  - Redefinición de operadores específicos para la clase.
  - Datos encapsulados: Las propiedades de un objeto no son visibles y sólo se puede acceder a ellas desde las funciones método de la clase.
  - Herencia: Una clase se puede crear a partir de otra, heredando todas sus funciones método. Las clases `tf`, `zpk` y `ss` derivan de la clase `lti`.
  - Agregación: Un objeto puede contener otros objetos.
- Toda la información de una clase, `tf`, está en el directorio asociado `@tf`.
- Las propiedades de clase usada quedan en memoria. Se deben limpiar para poder modificarlas,

```
>> clear tf
```



## Funciones método principales (I)

- Constructor: Genera un objeto a partir de datos.

```
>> P= tf(1,[1,2,3]); % Constructor 'tf', objeto P.
>> K= zpk([],[-1,-2],1); % Constructor 'zpk', objeto K
>> Pzpk= zpk(P); %P objeto 'tf', Pzpk objeto 'zpk'
>> isa(P,'tf') %afirmativo
>> isa(P,'zpk') %negativo
>> P %Llama a función 'display'
```

- Visualizador: Método `display` que muestra la información del objeto.

```
>> P %Llama a función 'display' de 'tf'
>> Pzpk %Llama a función 'display' de 'zpk'
```

## Funciones método principales (II)

- Obtención de información:
  - General: Método `get()`, muestra propiedades del objeto.

```
>> get(P)
```

  - Por índices: Método `B=subref(A,S)`

```
>> P.num{1}  %A=P,  S(1).type='.',S(1).type='{}',
>>  %S(1).subs={'num'},S(2).subs={1}
```

- Introducción de información:
  - General: Método `set()`, cambia propiedades del objeto.

```
>> set(P,'num',[1,2])
```

  - Por índices: Método `A= subsasign(A,S,B)`

```
>> P.num{1}= 1  %A=P,  S(1).type='.',S(2).type='{}'
>>  %S(1).subs={'num'},  S(2).subs={1},  B=1
```



## Redefinición de operadores (I)

| Operación | M-fichero | Descripción |
|---|---|---|
| `a + b` | `plus(a,b)` | Suma |
| `a - b` | `minus(a,b)` | Resta |
| `-a` | `uminus(a)` | Menos unitario |
| `+a` | `uplus(a)` | Más unitario |
| `a.*b` | `times(a,b)` | Multiplicación por elemento |
| `a*b` | `mtimes(a,b)` | Multiplicación matricial |
| `a./b` | `rdivide(a,b)` | División derecha por elemento |
| `a.\b` | `ldivide(a,b)` | División izquierda por elemento |
| `a/b` | `mrdivide(a,b)` | División matricial derecha |

## Redefinición de operadores (II)

| Operación | M-fichero | Descripción |
|---|---|---|
| a\b | mldivide(a,b) | División matricial izquierda |
| a.^b | power(a,b) | Potencia por elemento |
| a^b | mpower(a,b) | Potencia matricial |
| a < b | lt(a,b) | Menor que |
| a > b | gt(a,b) | Mayor que |
| a <= b | le(a,b) | Menor que o igual a |
| a >= b | ge(a,b) | Mayor que o igual a |
| a ~= b | ne(a,b) | Distinto de |
| a == b | eq(a,b) | Igual a |
| a & b | and(a,b) | Y lógico |
| a \| b | or(a,b) | O lógico |



## Redefinición de operadores (III)

| Operación | M-fichero | Descripción |
|---|---|---|
| ~a | not(a) | NO lógico |
| a:d:b<br>a:b | colon(a,d,b)<br>colon(a,b) | Operador dos puntos |
| a' | ctranspose(a) | Traspuesta conjugada compleja |
| a.' | transpose(a) | Matriz transpuesta |
| | display(a) | Visualización pantalla |
| [a b] | horzcat(a,b,...) | Concatenación horizontal |
| [a; b] | vertcat(a,b,...) | Concatenación vertical |
| a(s1,s2,...sn) | subsref(a,s) | Referencia por subíndices |
| a(s1,...,sn) = b | subsasgn(a,s,b) | Asignamiento por subíndices |
| (a) | subsindex(a) | Conversión al ser índice |

- Prioridades entre objetos ante métodos y operadores:
  - Un objeto creado tiene prioridad sobre una variable de Matlab,

```
>> P= 1/(s+1); % operador '/' y '+' de clase 'tf'.
```

  - Entre dos objetos creados hay que asignar prioridades,

```
>> inferiorto('class1', 'class2', ...)
>> superiorto('class1', 'class2', ...)
```

- Ejemplo de objetos, métodos y operadores:

```
>> s= tf('s') % Constructor
>> P(1)= 1/(s+1); % Operadores '/', '+', método '
    subsasign'
>> P(2)= 1/(s+2);
>> K= P(1) % método 'subsref'
```



## Ejemplo: Una clase de polinomios (I)

- Objetivos: Se pretende realizar una clase para trabajar con polinomios. Para ello se definen las siguientes funciones método:
  - Método constructor `polynom`: Se crea un objeto a partir de los coeficientes del polinomio.
  - Método `double`: El polinomio se podrá convertir a un vector.
  - Método `display`: El objeto se verá en la pantalla en forma de cadena de caracteres.
  - Sobrecarga de operadores: Los operadores suma (`+`), resta (`-`) y multiplicación (`*`) son redefinidos para polinomios.

## Ejemplo: Una clase de polinomios (II)

- Método constructor,

```
function p = polynom(a)
%polynom Constructor de la clase polynom.
%p = polynom(v) crea un polinomio de un vector.
%Los coeficiente están en orden decreciente
%de las potencias de x.

if nargin == 0
  p.c = [];
  p = class(p,'polynom');
elseif isa(a,'polynom')
  p = a;
else
  p.c = a(:).';
  p = class(p,'polynom');
end
```



## Ejemplo: Una clase de polinomios (III)

- Método `display`:

```
function display(p)
%polynom\display. Comando ventana para ver el objeto.
disp(int2str(p.c));
```

- Método `double`:

```
function c = double(p)
%polynom\double. Convierte polynom a un vector double.
%c = double(p). Convierte un polinomio en vector.
c = p.c;
```

- Operador `+`:

```
function r = plus(p,q)
%polynom\plus. Define p + q para polinomios.
p = polynom(p); q = polynom(q);
k = length(q.c) - length(p.c);
r = polynom([zeros(1,k) p.c] + [zeros(1,-k) q.c]);
```

## Ejemplo: Una clase de polinomios (IV)

- Operador -:

```
function r = minus(p,q)
%polynom\minus. Implementa p - q entre polinomios.
p = polynom(p); q = polynom(q);
k = length(q.c) - length(p.c);
r = polynom([zeros(1,k) p.c] - [zeros(1,-k) q.c]);
```

- Operador *:

```
function r = mtimes(p,q)
%polynom\mtimes. Implementa p * q entre polinomios.
p = polynom(p);
q = polynom(q);
r = polynom(conv(p.c,q.c));
```

- Ejemplo de su uso en *workspace*:

```
>> p= polynom %Crea un objeto vacio
>> p= polynom(p) %Devuelve el objeto que se manda
>> p= polynom([1,2,3]) %Crea un objeto p lleno
>> q= p+p
>> t= q-p
>> a= double(p)
```



## Clases derivadas: Herencia (I)

- Muchas veces se desea crear una nueva clase con las mismas propiedades y funciones métodos que otra ya existente a la que se añaden nuevas propiedades y funciones método.
- Esto se puede conseguir añadiendo un objeto de la clase existente `ClasePadre` en la definición de la nueva clase.
- Los objetos de la nueva clase serán `ObjetoHijo`, y los de la clase existente `ObjetoPadre`.
- Un `ObjetoHijo` puede acceder a todos las funciones método de la `ClasePadre` que no estén definidos en su clase.
- La forma de definir un objeto hijo es la siguiente:

```
ObjetoHijo= class(ObjetoHijo,'ClaseHijo',ObjetoPadre);
```

- Con esta definición Matlab crea un componente `ObjetoHijo.ClasePadre` donde se guardará la información de la parte del `ObjetoHijo` con las mismas propiedades que el `ObjetoPadre`.

## Clases derivadas: Herencia (II)

- Un objeto hijo puede recibir herencia de varios objetos padres,

```
ObjetoHijo= class(ObjetoHijo,'ClaseHijo',ObjetoPadre1,
    ObjetoPadre2);
```

  Una función método que no posea la `ClaseHijo` será buscada en las funciones de la clase `ClasePadre1` y de no ser encontrada entre las de la clase `ClasePadre2`.
- Un ejemplo muy sencillo de una clase derivada es el de una clase de funciones, cuyas propiedades son:
  - Nombre de la función.
  - Polinomio característico.
- Está claro que esta clase `funcion` puede ser propuesta como derivada de la clase polinomio `polymon`, añadiendo a la misma una propiedad donde se escriba el nombre de la función.
- Todas las funciones método de la clase `polynom` pueden ser usadas en la clase `funcion` excepto el método constructor, el método `display` y el `subsref`, que van a ser redefinidos.
- Cuando los objetos `funcion` use funciones método de la clase `polynom`, se está trabajando con la parte del objeto `funcion` heredada, y el resultado de la operación podrá ser un objeto `polynom` o de otra clase ya definida, pero nunca de la clase `funcion`.



## Clases derivadas: Herencia (III)

- Función método constructor de `funcion`. Código.

```
function p = funcion(varargin)
%FUNCION Construtor de la clase funcion.
switch nargin
case 0
  poly= polynom;
  p.nombre = '';
  p = class(p, 'funcion', poly);
case 1
  if isa(varargin{1},'funcion')
     p = varargin{1};
  else
     error('Tipo de argumento erróneo'); end
case 2
  if ischar(varargin{1}),
     p.nombre= varargin{1};
  else
     error('Arg: nombre, polinomio'); end
  poly = polynom(varargin{2});
  p = class(p,'funcion', poly);
otherwise
  error('Número de argumentos erróneo'); end
```

- Función método `display`. Código.

```
function display(p)
%POLYNOM/DISPLAY Comando ventana para ver el objeto.
disp(' ');
disp(['Función ', p.nombre,' = '])
disp(' ');
disp([' ' char(p)]);
```

- Ejemplos en el *workspace*:

```
>> p= funcion %Objeto funcion nulo
>> p1= funcion('zeta', [1,2,3])
>> %Objeto funcion con nombre y parámetros.
>> p2= funcion('eta', [2,3,4])
>> p3= p1+p2
>> %Se usa un método del padre. El resultado en un
    objeto polynom.
```





- Función método `subsref`: Muestra por campos el nombre y polinomio de la función, y por índice el valor de la función en un punto. Código.

```
function b = subsref(a,s)
%SUBSREF Muestra, por campos el contenido del
%objeto, por índices el valor en un cierto punto.
switch s.type,
case '.',
   switch s.subs
   case 'nombre',
      b= a.nombre;
   case 'poly',
      b= char(a.polynom);
   otherwise
      error('Campos: nombre, poly.');
   end
case '()',
   ind= s.subs{:};
   b= a.polynom(ind);
otherwise
   error('Campo o índice erróneo.');
end
```

## Clases agregadas (I)

- Una clase puede tener de componentes objetos de otras clases ya definidas.
- Con ello, no se heredan directamente sus funciones método, pero estas funciones podrán ser usadas en la definición de las nuevas funciones método.
- Ejemplo: La clase `transfer` tiene las siguientes propiedades:
  - Polinomio del numerador.
  - Polinomio del denominador.
- Está claro que se puede aprovechar la clase `polynom` para crear esta nueva clase ya que sus dos componentes son polinomios.
- Las funciones método de la nueva clase no tienen nada que ver con las de `polynom`, pero en su construcción serán empleadas.
- Las funciones método de la nueva clase son la función constructora, `display`, `subsref`, `plus`, `minus`, `mtimes` y `mrdivide`.



## Clases agregadas (II)

- Función método constructor. Código.

```
function p = transfer(varargin)
%FUNCION Construtor de la clase transfer
%Crea una función de transferencia compuesta de
%dos polinomios, uno en numerador y otro en
    denominador.
switch nargin
case 0
  p.num= polynom; p.den= polynom;
  p = class(p, 'transfer');    %Objeto nulo
case 1
  if isa(varargin{1},'transfer')
     p = varargin{1};    %Objeto funcion
  else
     p.num= polynom(varargin{1}); p.den= polynom(1);
     p= class(p,'transfer');   %Objeto sólo numerador
  end
case 2
  p.num= polynom(varargin{1}); p.den= polynom(varargin
      {2});
  p = class(p,'transfer');   %Objeto numerador y
      denominador
otherwise
  error('Número de argumentos erróneo');
end
```

## Clases agregadas (III)

- Función método `display`. Código:

```
function display(p)
%transfer\display. Comando ventana para ver el objeto.
num= char(p.num);
den= char(p.den);
disp(' ');
disp([inputname(1),' = '])
disp(' ');
disp([num]);
disp(repmat('-',[1,max([size(num,2),size(den,2)])]));
disp(den);
```

- Ejemplos en el *workspace*:

```
>> g= transfer; %Objeto nulo
>> g= transfer([1,2,3]) %Objeto con num y den= 1.
>> p= polynom([1,2,3]) %Objeto polynom
>> g= transfer([1,2,3],[3,4,5]); %Objeto con num y den
>> g= transfer(p, [1,2,3]); %Objeto con num y den
```



## Clases agregadas (IV)

- Función método `subsref`. Código:

```
function b = subsref(a,s)
%transfer\subsref. Por campos, la representación del
    objeto.
%Por índices, el valor de la función en un punto.

switch s.type,
case '.',
   switch s.subs,
   case 'num',
      b= char(a.num); %polinomio num
   case 'den',
      b= char(a.den); %polinomio den
   otherwise
      error('Campos: num, den');
   end
case '()',
   ind = s.subs{:};
   b= a.num(ind)./a.den(ind); %Valor en x
otherwise
  error('Dar campo o valor de x en p(x)')
end
```

## Clases agregadas (V)

- Sobrecarga de operadores. Código:

```
function r = plus(p,q)
%transfer\plus. Define p + q para transfer.
p = transfer(p);
q = transfer(q);
r.num= p.num*q.den + p.den*q.num;
r.den= p.den*q.den;
r= class(r,'transfer');

function r = minus(p,q)
%transfer\minus. Define p - q para transfer.
p = transfer(p);
q = transfer(q);
r.num= p.num*q.den - p.den*q.num;
r.den= p.den*q.den;
r= class(r,'transfer');

function r = mtimes(p,q)
%transfer\mtime. Define p * q para transfer.
p = transfer(p);
q = transfer(q);
r.num= p.num*q.num;
r.den= p.den*q.den;
r= class(r,'transfer');
```



## Clases agregadas (VI)

- Sobrecarga de operadores. Código:

```
function r = mtimes(p,q)
%transfer\mrdivide Define p / q para transfer.
p = transfer(p);
q = transfer(q);
r.num= p.num*q.den;
r.den= p.den*q.num;
r= class(r,'transfer');
```

- Ejemplos en el *workspace*:

```
>> g1= transfer([1,2,3],[2,3,4])
>> g2= transfer([1,4,3])    %den=1
>> g3= g1+g2; %Objeto transfer
>> g3= g1/g2 %Objeto transfer
>> g3.num %Cadena de caracteres
>> g3(10) %Valor de cociente en x=10
```

- Modificar la función método `subsref`, `Fichero Dejer1.m` de la clase `funcion`, de forma que los índices sirva para devolver el valor del coeficiente correspondiente. Por ejemplo, `p(3)` debe devolver el tercer coeficiente.
- Modificar la función método `subsref`, `Fichero Dejer2.m`, de la clase `transfer` de forma que devuelva los índices de numerador y denominador correspondientes. Por ejemplo, `g(1,2)` debe devolver el primer coeficiente del numerador, y segundo del denominador.
- Definir una función método `subasgn`, de las clases `funcion` y `transfer`, `Fichero Dejer3.m y Dejer4.m` con un criterio similar a los empleados en los dos apartados anteriores. Ejemplo en la clase `funcion`, `p(3)=5`, introduce un `5` en la posición tercera del polinomio. Ejemplo en la clase `transfer`, `g(2,3)=[1,2]`, introduce un `1` en la posición segunda del numerador, y un `2` en la posición tercera de denominador.

## Definición de sistema continuos.

- Un sistema es la relación entre una señal de entrada y una de salida,

$$y(t) = F(u(t)).$$

- Todo sistema físico es causal, es decir, la señal de salida depende en el tiempo de la señal de entrada.
- Un sistema continuo en el tiempo puede ser representado matemáticamente mediante una ecuación diferencial ordinaria (ODE),

$$y^{(n)} = f(t, y, y', \ldots, y^{(n-1)}).$$

Nota: La entrada es una función del tiempo $u(t)$.



## Definición de sistemas continuos lineales

- Sistema lineal:
  - Si $u(t) \rightarrow y(t)$, entonces $\alpha u(t) \rightarrow \alpha y(t)$.
  - Si $\{u_1(t), u_2(t)\} \rightarrow \{y_1(t), y_2(t)\}$, entonces $\{u_1(t) + u_2(t)\} \rightarrow \{y_1(t) + y_2(t)\}$.
- Un sistema lineal se rige por una ecuación diferencial lineal,

$$y^{(n)} + a_{n-1}y^{(n-1)} + \ldots + a_0 y = b_n u^{(n)} + b_{n-1}u^{(n-1)} + \ldots + b_0 u.$$

Nota: Ver que de esta forma se cumple con su definición.

- Función de transferencia de un sistema lineal es la transformada de Laplace de su ecuación diferencial,

$$\frac{Y(s)}{U(s)} = \frac{s^n + a_{n-1}s^{n-1} + \ldots + a_0}{b_n s^n + b_{n-1}s^{n-1} + \ldots + b_0}.$$

- Un sistema muestreado puede ser representado por una ecuación en diferencias,

$$y(k) = f(k, y(k), y(k-1), \ldots, y(k-n)).$$

- Un sistema muestreado lineal puede ser representado por un una ecuación en diferencias lineal,

$$y(k+n)+a_1 y(k+n-1)+\ldots+a_0 y(k) = b_n u(k+n)+b_{n-1} u(k+n-1)+\ldots+b_0 u(k).$$

- Función de transferencia de un sistema muestreado lineal es la transformada $\mathcal{Z}$ de su ecuación en diferencias,

$$\frac{Y(z)}{U(z)} = \frac{z^n + a_1 z^{n-1} + \ldots + a_0}{b_n z^n + b_{n-1} z^{n-1} + \ldots + b_0}.$$

- Un sistema, en general puede estar compuesto por partes continuas, muestreadas, lineales y no lineales.



## Simulación en Matlab y Simulink: Comparación

- La simulación de un sistema consiste en predecir los datos de salida del mismo frente a los datos de entrada.
- Simulación desde Matlab:
  - Creación de un fichero con la ecuación diferencial del sistema en forma de derivadas de primer orden.
  - Resolución del ODE por métodos similares a los de Runge-Kutta.
- Simulación desde Simulink (interface gráfico):
  - Dibujo del sistema en un entorno gráfico, donde se dispone de iconos para sus partes lineales, no lineales, continuas y discretas.
  - Creación de un fichero con la información de la planta y uso de las funciones de simulación de matlab.

- La ecuación ODE del sistema $y^{(n)} = f(t, y, y', \ldots, y^{(n-1)})$ se debe transformar a una ecuación ODE vectorial de primer orden $y' = F(t, y)$.
- Una forma sencilla de conseguirlo es mediante el cambio,

$$y_1 = y, \quad y_2 = y', \quad \ldots, \quad y_n = y^{(n-1)},$$

y por tanto

$$\begin{bmatrix} y_1' \\ y_2' \\ \vdots \\ y_n' \end{bmatrix} = \begin{bmatrix} y_2 \\ y_3 \\ \vdots \\ f(t, y_1, y_2, \ldots, y_n) \end{bmatrix}.$$

- Condiciones iniciales: Valores iniciales de

$$\begin{bmatrix} y_1(t_0), \ldots, y_n(t_0) \end{bmatrix}^T.$$



## ODE de la función de Val der Pol

- Ejemplo: Función de Val der Pol, (dinámica no lineal masa-muelle-amortiguador)

$$y_1'' - \mu(1 - y_1^2)y_1' + y_1 = 0,$$

en ecuaciones de estado,

$$\begin{bmatrix} y_1' \\ y_2' \end{bmatrix} = \begin{bmatrix} y_2 \\ \mu(1 - y_1^2)y_2 - y_1 \end{bmatrix}.$$

- Fichero con la relación de las ecuaciones de estado,

```
function dy= pol(t,y)
%t es el tiempo
%y es el valor del vector para un t.
%dy es la derivada de y para un t dado.
%y(3) parámetro modificable con condiciones iniciales
dy = [y(2); y(3)*(1-y(1)^2)*y(2)-y(1); 0]; %Columna
```

- Esta función será llamada por el programa ODE en los sucesivo puntos $t$ para obtener la derivada.
- La entrada se debe poner en función de $t$.

- Funciones ODE para sistema suaves basados en métodos de Ruge-Kutta:
  - `ode45`, `ode23` y `ode113`.
- Funciones ODE para problema con cambios de alta frecuencia:
  - `ode15s`, `ode23s` y `ode23t`.
- Formato de la llamada a la función ODE:

```
>> [t, y]= solver(@F, tspan, y0, option)
```

  - `@F`: Nombre o puntero del fichero `.m` donde se guarda la función.
  - `tspan`:
    - `[ti,tf]`: Límite inferior y superior. Paso y número de valores de salida variables.
    - `linspace(ti,tf,Npuntos)`: Se fija el número de puntos y tiempo cuya salida se desea conocer. El programa internamente tiene paso variable.
  - `y0`: Valores iniciales deseados (vector columna).
  - `option`: Especificaciones del algoritmo. Si se pone `[]` se toman por defecto.





- Consultar con la ayuda:

```
>> help odeset
>> option= odeset; %datos por defecto.
>> x0= [0.1, 1.1, 0.1]; %dos estados, un parámetro
>> sol= ode45(@pol, x0 ); %Versión 7
>> y= deval(sol, linspace(0,10,100)); %sol. en puntos
```

Ejemplos:

- Número de datos de salida:

```
>> option= odeset('Refine', 4); %por defecto.
```

- Jacobiano del ODE en función `jacpol.m`:

```
>> option= odeset('Jacobian', @jacpol);
```

Se precisa una función de la forma,

```
function jac= jacpol(t,y)
jac = [0,       1, 0;
      -2*y(1)*y(2), 1-y(1)^2, (1-y(1)^2)*y(2);
       0, 0, 0];
```

## Ecuaciones diferenciales con valores de frontera (I)

- El problema puede ser planteado por las ecuaciones como para $a < t < b$,

$$y' = f(t, y, p)$$
$$g(y(a), y(b), p) = 0$$

Se resuelve con la función:

```
>> sol= bvp4c(@F, @bc, solinit, option, p1, p2, ...)
```

- `@F` nombre o puntero a función que define el problema.
- `@bc` nombre o puntero a función que define los valores frontera.
- `solinit`: Fijar el mallado en $t$ y puntos iniciales para $y$.
- `option` opciones de resolucion `bvpset`, `bvpget`.
- `pi` parámetros extras.



## Ecuaciones diferenciales con valores de frontera (II)

- Ejemplo: Solución de la ecuación $y'' + |y| = 0$, sabiendo que $y(0) = 0$ e $y(4) = -2$.
  - Función diferencial:

```
function dydx= F(t,y)
dydx= [y(2); -abs(y(1))];
```

  - Función frontera:

```
function res= bc(ya,yb)
rec= [ya(1); yb(1)+2];
```

  - Operaciones a realizar:

```
>> solinit= bvpinit(linspace(0,4,5), [1,0]);
>> sol= bvp4c(@F, @bc, solinit);
>> t=linspace(0,4); y= deval(sol, t);
>> plot(t,y);
```

- Una ecuación en derivadas parciales puede formularse como:

$$c\left(x,t,u,\frac{\partial u}{\partial x}\right)\frac{\partial u}{\partial t}=x^{-m}\frac{\partial}{\partial x}\left(x^{m}f\left(x,t,u,\frac{\partial u}{\partial x}\right)\right)+s\left(x,t,u,\frac{\partial u}{\partial x}\right)$$

donde $a \leq x \leq b$, $t_0 \leq t \leq t_f$.
Condiciones iniciales: Para $t = T_0$,

$$u(x,t_0)=u_0(x).$$

Condiciones frontera: Para $x = a$ o $x = b$,

$$p(x,t,u)+q(x,t)f\left(x,t,u,\frac{\partial u}{\partial x}\right)=0.$$





- Se resuelve con la función:

```
>> sol= pdepe(m, @F, @init, @front, xmesh, tspan,
    options, p1, p2, ...)
```

  - `m`: Simetría de la ec. diferencial, bloques (`m=1`), cilíndrica (`m=2`) y esférica (`m=3`).
  - `F` Nombre o puntero a la definición de función.

```
>> [c,f,s]= F(x,t,u,dudx)
```

  - `init`: Nombre o puntero a las condiciones iniciales.

```
>> u= init(x)
```

  - `front`: Nombre o puntero a las condiciones frontera.

```
>> [pl,ql,pr,pr]= front(xl,ul,xr,ur,t)
```

  - `xmesh`: Mallado de los valores de $x$.
  - `tspan`: Mallado de los valores de $t$.

- Ejemplo: Resolver la ecuación diferencial

$$\pi^2 \frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left(\frac{\partial u}{\partial x}\right),$$

sujeto a las condiciones iniciales $u(x,0) = \sin(\pi x)$ y condiciones frontera

$$u(0,t) = 0, \quad \pi e^{-t} + \frac{\partial u}{\partial x}(1,t) = 0$$

```
function [c,f,s]= F(x,t,u,dxdu)
c= pi^2; f= dxdu; s=0;
```

```
function u0= init(x)
u0= sin(pi*x);
```

```
function [pl,ql,pr,qr]= front(xl,ul,xr,ur,t)
pl=ul; ql=0; pr= pi*exp(-t); qr= 1;
```

```
>> m= 0; x= linspace(0,1,20); t= linspace(0,2,5);
>> sol= pdepe(m,@F,@init,@front,x,t);
>> u= sol(:,:,1);
>> figure; surf(x,t,u);
>> figure; plot(x,u(end,:));
```





- Resolver la ecuación de Van del Pol $y'' + \mu(1-y^2)y' + y = 0$ para $\mu = 1$, con valores iniciales $y(0) = 2$ e $y'(0) = 0$, en el intervalo $t = [0, 20]$ (usando su Jacobiano) por el `ode45` y `ode23`. Generalizar el resultado para una $\mu$ cualquiera. El fichero "script" con la resolución del problema se llamará `Eejer1.m`.
- Resolver la ecuación de Lorenz, usadas en la descripción de sistemas caóticos, para los puntos iniciales y valores de $\sigma$, $r$ y $b$ que el usuario desee, por ejemplo $\sigma = 10$, $r = 28$ y $b = 8/3$. El fichero "script" con la resolución del problema se llamará `Eejer2.m`.

$$\begin{aligned} x' &= \sigma(y - x) \\ y' &= x(r - z) - y \\ z' &= xy - bz \end{aligned}$$

- Dada la ecuación $y'' + (\lambda - 2q\cos(2t))y = 0$, con condiciones de frontera $y(0) = 1$, $y'(0) + y'(\pi) = 0$ encontrar una solución para $q = 15$ y $\lambda = 15$, basándose en una solución inicial para diez puntos de $t$ en el intervalo $[0, \pi]$. Dibujar la gráfica de la primera componente en 100 puntos igualmente espaciados entre $[0, \pi]$. El fichero "script" con la resolución del problema se llamará `Eejer3.m`.

## Simulink: Tabla de bloques propios

- Bloques específicos de Simulink:
  - `Continuous:` Bloques de sistemas continuos escritos en base a su funciones de transferencia; sus polos, ceros y ganancias; y sus ecuaciones en espacio de estado.
  - `Discrete:` Bloques de sistemas discretos escritos en base a su funciones de transferencia; sus polos, ceros y ganancias; y sus ecuaciones en espacio de estado.
  - `Function & Tables:` Funciones y tablas de Matlab. Especial importancia las S-Function.
  - `Math:` Bloque de operaciones matemáticas entre señales.
  - `Nonlinear:` Bloque de no linealidades.
  - `Signal & Systems:` Entradas y salidas de datos hacia el espacio de trabajo de Matlab (bloques `in` y `out`), y hacia ficheros. Bloque `subsystem` que permite generar un diagrama de bloque dentro de otro.
  - `Sinks:` Bloques que muestran los datos simulados en pantallas o los guardan en ficheros.
  - `Sources:` Bloques que generan diferentes tipos de señales.



## Simulink: Tabla de bloques pertenecientes a toolbox

- Bloques pertenecientes a toolbox de Matlab:
  - `Control System Toolbox:` Bloques de sistemas continuos y discretos en la formulación orientada a objeto LTI específica de esa toolbox.
  - `Real-Time:` Bloques de comunicación entre el sistema y una tarjeta de adquisición de datos.
  - En general, todas las toolbox de matlab han desarrollado funciones de simulink en la versión 7 o posterior.

- Interface gráfico para modelar un sistema.
- Simulación desde Matlab: Entradas bloques `in` y salidas bloques `out`.

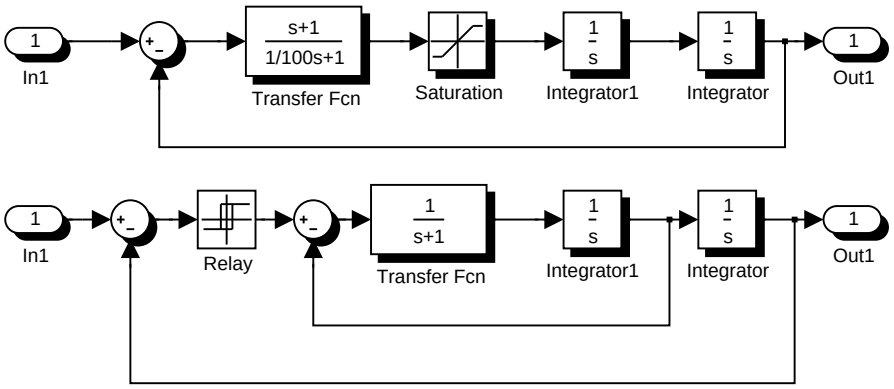
```
>> [t, x, y]= sim('FUN', tspan, option, [t,u])
```

  - `'FUN'`: Nombre del fichero `.mdl` del fichero Simulink.
  - `tspan`:
    - `[ti,tf]`: Límite inferior y superior. Paso y número de valores de salida variables.
    - `linspace(ti,tf,Npuntos)`: Se fija el número de puntos y tiempo cuya salida se desea conocer. El programa internamente tiene paso variable.
  - `x0`: Valores iniciales de las variables de estado (vector columna).
  - `option`: Especificaciones del algoritmo. Si se pone `[]` se toman por defecto.
  - `[t,u]`: Tiempo y entradas al modelo Simulink desde el espacio de trabajo.





- Formular en un fichero `*.m` los dos modelos planteados en Simulink y demostrar que la simulación con la función `ode45` y con Simulink es equivalente. El nombre del fichero "script" será `Eejer4.m`.



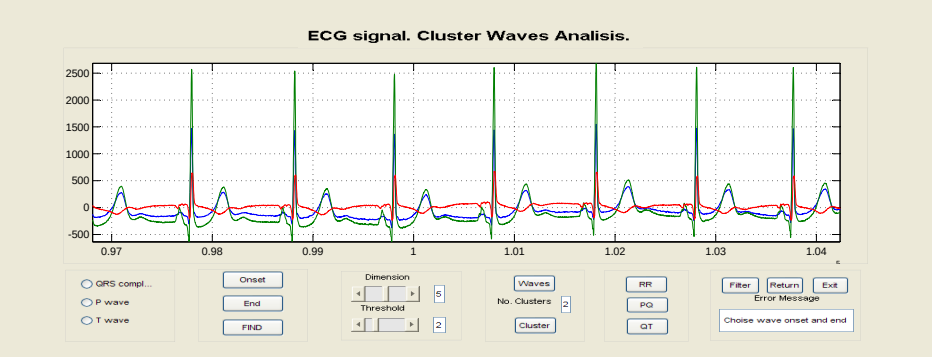
# Contenidos





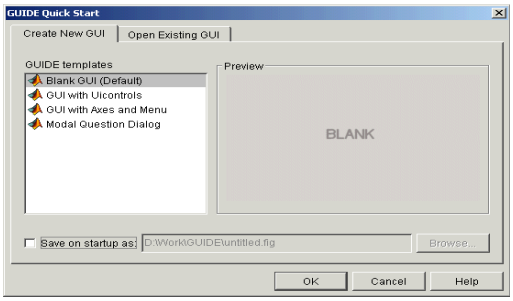
# GUIDE: Interface gráfico de matlab (I)

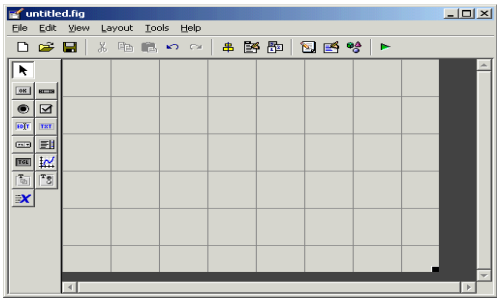
- `Definición:` Es una herramienta para construir interfaces gráficos con botoneras, figuras, texto, y más elementos.
  - `Construcción de gráficos:` Se realiza con un interface del programa que permite colocar cada elemento donde se desee. Tras ello se exporta la información a un fichero `.m`.
  - `Programación de funciones:` Cada elemento del gráfico tiene asociado en el fichero `.m` una función donde el programador escribe las instrucciones de cada elemento.
- `Ejemplo:` La siguiente gráfica muestra un inteface para el análisis de las señales cardíacas. Se compone de,
  - `Pantalla:` para visualizar los datos.
  - `Botones:` para marcar las operaciones que se desean realizar.
  - `Pantallas de texto:` Para mandar mensajes al programa.



## GUIDE: Interface gráfico de matlab (II)

- **Entorno gráfico:** El comando `GUIDE` abre una pantalla con la que se puede dibujar el esquema gráfico del interface,
  - Cada elemento añadido es un objeto con un nombre y propiedades que se pueden modificar en la pantalla.
  - Los elementos se pueden alinear, formar bloques y otra serie de operaciones para conseguir una gráfica bonita.
  - Cuando la figura se haya terminado se procede a exportar la información a un fichero `.m`.
- **Pantalla GUIDE y paleta de trabajo:**






## GUIDE: Interface gráfico de matlab (III)

- **Programación de los objetos:** En el fichero `.m` generado con el interface cada objeto tiene asociado dos funciones, una de inicialización y otra de llamada.
- **Variables de las funciones:** Son dos objetos, `hObject` para los gráficos y `handles` para la información.
- **Ejemplo:** Barra para mandar datos ("slider"),

  Función creación de un "slider":

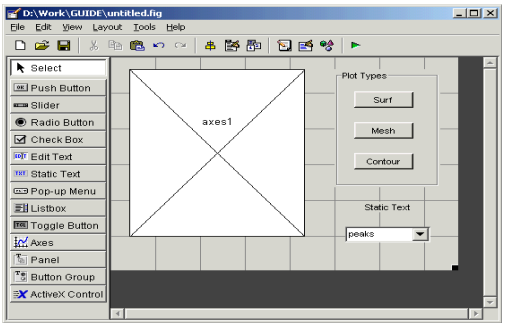
```
function S_Dim_CreateFcn(hObject, eventdata, handles)
% Introduce en el objeto gráfico los valores iniciales
set(hObject,'Value', 5); set(hObject,'Min', 0);
set(hObject,'Max', 10);
```

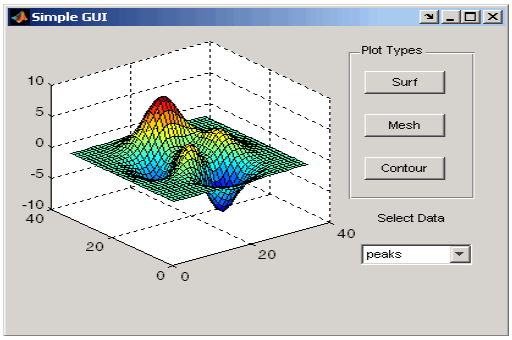
  Función de llamada de un "slider":

```
function S_Dim_Callback(hObject, eventdata, handles)
% Extrae del objeto gráfico el valor
N= get(hObject, 'Value');
% Introduce dicho valor en otro objeto
% E_Dim, casilla de texto
set(handles.E_Dim, 'String', num2str(floor(N)));
```

- `Ejercicio:` Realizar un interface de usuario con la herramienta GUIDE que consiga mostrar en una pantalla gráficas elegidas por el usuario en diferentes formatos, superficie, mallado o contorno. La función donde debe ser guardado el programa se llamará `Fejer1.m`

  Ver las explicaciones del manual de matlab del interface de usuario, `builgui.pdf`, donde se explica este ejemplo con detalle.






## Contenidos

## Funciones matemáticas básicas (I)

| Función | *Comentario* |
|---|---|
| abs | Valor absoluto |
| acos, acosh | Arco coseno y arco coseno hiperbólico |
| acot, acoth | Arco cotangente y arco cotangente hiperbólico |
| acsc, acsch | Arco cosecante y arco cosecante hiperbólico |
| angle | Argumento |
| asec, asech | Arco secante y arco secante hiperbólico |
| asin, asinh | Arco seno y arco seno hiperbólico |
| atan, atanh | Arco tangente y arco tangente hiperbólico |
| atan2 | Arco tangente en el cuarto cuadrante |
| ceil | Redondeo al entero más próximo |
| complex | Forma un número complejo |



## Funciones matemáticas básicas (II)

| Función | *Comentario* |
|---|---|
| conj | Complejo conjugado |
| cos,cosh | Coseno y coseno hiperbólico |
| cot,coth | Cotangente y cotangente hiperbólica |
| csc,csch | Cosecante y cosecante hiperbólica |
| exp | Exponencial |
| fix | Elimina la parte decimal |
| floor | Mayor entero menor o igual que un real dado |
| gcd | Máximo común divisor |
| imag | Parte imaginaria de un número complejo |
| lcm | Máximo común múltiplo |
| log | Logaritmo neperiano |
| log2 | Logaritmo base 2 |
| log10 | Logaritmo base 10 |
| mod | Módulo |

| Función | *Comentario* |
|---|---|
| nchoosek | Coeficiente binomial |
| real | Parte real de un número complejo |
| rem | Resto de la división |
| round | Redondeo al entero más cercano |
| sec,sech | Secante y secante hiperbólica |
| sign | Signo |
| sin,sinh | Seno y seno hiperbólico |
| sqrt | Raíz cuadrada |
| tan,tanh | Tangente y tangente hiperbólica |

- Pueden consultarse con

```
>> help elfun
```

- MATLAB tiene también funciones matemáticas especiales

```
>> help specfun
```



## Ejemplo de funciones matemáticas básicas DElfun1.m

- Combinaciones de 10 elementos tomadas de 4 en 4 $\binom{10}{4}$:

```
>> nchoosek(10,4)
```

- Seno y coseno de los ángulos entre 0 y $2\pi$, incrementando de $\pi/2$ en $\pi/2$.

```
>> sin(0:pi/2:2*pi)
>> cos(0:pi/2:2*pi)
```

- Algunas propiedades de las funciones exponencial y logarítmica

```
>> exp(2*pi*i)
>> exp(log(2))
>> 2*exp(i*pi)
>> 2*(cos(pi)+i*sin(pi))
>> log(3+2*i)
```

- Algunas propiedades de las funciones trigonométricas

```
>> sin(pi/4)^2+cos(pi/4)^2
>> (exp(5)+exp(-5))/2
>> cosh(5)
>> cosh(pi)^2-sinh(pi)^2
>> 1+tan(pi/4)^2
>> sec(pi/4)^2
```

| Función | *Comentario* |
|---|---|
| `cart2pol,pol2cart` | Transforma cartesianas a polares (cilíndricas 3D) |
| `cart2sph,sph2cart` | Transforma cartesianas a esféricas |

- Ejemplo de transformación de coordenadas `DCoor1.m`
  - Transforma el punto $(3, 2, 5)$ de cilíndricas a cartesianas:

```
>> [x,y,z]=pol2cart(3,2,5)
```

  - Transforma el punto $(1, 1, 1)$ de cartesianas a cilíndricas y a esféricas:

```
>> [c1,c2,c3]=cart2pol(1,1,1)
>> [c1,c2,c3]=cart2sph(1,1,1)
```

  - Transforma el punto $(5, \pi/3)$ de cilíndricas a cartesianas:

```
>> [x,y]=pol2cart(5,pi/3)
```





| Función | *Comentario* |
|---|---|
| `max` | Máximo de vector |
| `mean` | Media |
| `median` | Mediana |
| `min` | Máximo |
| `perms` | Permuta las filas de una matriz |
| `sort` | Datos ordenados |
| `sortrows` | Ordena filas de una matriz |
| `std` | Desviación estandar. |
| `var` | Varianza |

## Funciones estadísticas básicas (II)

| Función | *Comentario* |
|---|---|
| corr | Correlación entre variables |
| cov | Matriz de covarianzas |
| corrcoef | Matriz de correlaciones |
| xcorr | Correlación cruzada entre variables |
| xcov | Covarianzas cruzadas entre variables |
| cumprod | Producto acumulativo |
| cumsum | Suma acumulativa |
| cumtrapz | Integración acumulativa trapezoidal |
| diff | Función diferencial y aproximación acumulativa |
| find | Busca datos en vectores |
| hist,histc | Histograma y contaje de histograma |



## Ejemplo funciones básicas estadística *DStat1.m* (I)

- Generamos dos series de 1000 números cada una que se almacenan en los vectores x e y. Estos vectores representan un conjunto de medidas obtenidas de muestrear dos variables aleatorias X e Y.

```
>> randn('seed', 1);
>> x = randn(1000,1);
>> y = randn(1000,1);
```

- El valor medio de x se calcula con el comando:

```
>> mean(x)
```

- Si hubiera algún valor NaN en el vector x, el comando mean(x) devuelve NaN como media, para descontar estos valores se utiliza el comando NaN

```
>> xn=x;
>> xn(200)=NaN;
>> mean(xn)
>> nanmean(xn)
```

- La mediana se calcula con el comando:

```
>> median(x)
```

## Ejemplo funciones básicas estadística *DStat1.m* (II)

- La desviación típica se calcula con el comando

```
>> std(x)
```

- La varianza se calcula con el comando

```
>> var(x)
```

- El valor más grande de la serie se obtiene con el comando

```
>> max(x)
```

- El valor más pequeño de la serie se obtiene con el comando

```
>> min(x)
```

- El rango de valores de la serie se obtiene con el comando

```
>> range(x)
```



## Ejemplo funciones básicas estadística *DStat1.m* (III)

- La matriz de covarianza cruzada entre las dos variables aleatorias X e Y se obtiene con el comando:

```
>> cov(x,y)
```

- La matriz de correlación cruzada entre las dos variables aleatorias X e Y se obtiene con el comando:

```
>> corrcoef(x,y)
```

- Para obtener la posición o índice del mayor o menor valor dentro del vector x, se puede utilizar el comando max o min con argumentos de salida.

```
>> [a i] = max(x)
```

El mayor valor es a, y su posición dentro del vector x queda almacenado en la posición i.

## Ejemplo funciones básicas estadística *DStat1.m* (IV)

- Los valores del vector x pueden ordenarse con el comando sort

```
>> xs = sort(x);
```

- Se puede obtener el índice de ordenación utilizando sort con un segundo argumento de salida

```
>> [xs i] = sort(x);
```

- Tanto xs, como x(i) contienen los valores ordenados de menor a mayor, para ver los que van de la posición 201 a 210 se hace:

```
>> [xs(201:210) x(i(201:210))]
```

- El histograma de los datos se calcula con el comando

```
>> hist(x)
```

- Por defecto el comando hist utiliza 10 intervalos. Para utilizar un número diferente de intervalos, por ejemplo 50, hacer

```
>> hist(x, 50)
```

- La cuenta de elementos h por intervalo i se obtiene con el comando

```
>> [h i] = hist(x,50);
```

  i contiene el valor medio del intervalo y h la cuenta de elementos



## Ejercicios de tratamiento de datos

- En un fichero script de nombre `Gejer1.m` realizar las siguientes operaciones:
  - Generar una variable aleatoria $x$ con distribución normal y otra $y$ con distribución uniforme, ambas con 1000 elementos.
  - Hallar la media, varianza y mediana de ambas variables.
  - Hallar el histograma de ambas variables.
  - Representar la función de distribución acumulada de ambas variables a partir de los datos ordenados.
  - Representar la función de distribución de ambas variables a partir de la diferencia de los datos obtenidos en el apartado anterior.
  - Hallar el diagrama Q-Q entre ambas variables, es decir, el diagrama de los datos ordenados de una variable con respecto a la otra.
  - Hallar la correlación y convarianza entre ambas variables.
  - Hallar la correlación y covarianza cruzadas de las variables consigo mismas y entre ellas para un tiempo de $[-\tau, \tau]$.

## Funciones básicas de álgebra matricial (I)

| Función | *Comentario* |
|---|---|
| `expm` | Exponencial de una matriz $e^A$ |
| `logm` | Logaritmo neperiano de una matriz |
| `sqrtm` | Raíz cuadrada de una matriz |
| `funm` | Cualquier función matemática aplicada a una matriz |
| `transpose, ()'` | Transpuesta de una matriz |
| `inv` | Inversa de una matriz |
| `det` | Determinante de una matriz |
| `rank` | Rango de una matriz |
| `trace` | Traza de una matriz |

## Funciones básicas de álgebra matricial (II)

| Función | *Comentario* |
|---|---|
| `eig` | Valores propios de una matriz |
| `svd` | Valores singulares de una matriz |
| `cond` | Número de condición de una matriz |
| `rcond` | Recíproco del número de condición (estimado) |
| `norm` | Norma de una matriz |
| `null` | Base ortonormal del núcleo de una matriz |
| `orth` | Base ortonormal de la imagen de una matriz |
| `subspace` | Ángulo entre los subespacios de dos matrices |



## Ejemplo de Matlab de funciones de álgebra de matrices *DAlg1.m* (I)

- Formamos una matriz cuadrada aletoria de dimensión 3

```
>> A=randn(3)
```

- Calculamos su traspuesta

```
>> A'
```

- Calculamos su rango con rank

```
>> rank(A)
```

- Calculamos su determinante con det

```
>> det(A)
```

- Calculamos sus autovalores con eig

```
>> eig(A)
```

- Calculamos su traza con eig

```
>> eig(A)
```

- Comprobamos que la traza es la suma de los autovalores

```
>> [sum(eig(A)) trace(A)]
```

## Ejemplo de Matlab de funciones de álgebra de matrices *DAlg1.m* (II)

- Comprobamos que el determinante es el producto de los autovalores

```
>> [prod(eig(A)) det(A)]
```

- Calculamos el número de condición

```
>> cond(A)
```

- Comprobamos que el número de condición es el cociente entre el máximo y el mínimo autovalor

```
>> sqrt(max(eig(A'*A))/min(eig(A'*A)))
>> max(svd(A))/min(svd(A))
```

- Estimamos el recíproco del número de condición con rcond

```
>> rcond(A)
```

- Obtenemos el error relativo de estimación obtenido con rcond')

```
>> abs(cond(A)-1/rcond(A))/cond(A)
```



## Ejemplo de Matlab de funciones de álgebra de matrices *DAlg2.m* (I)

- Formamos una matriz cuadrada compleja aleatoria de dimensión 3

```
>> B=randn(3)+j*randn(3)
```

- Calculamos B elevada al cubo

```
>> B^3
```

- Calculamos 2 elevado a B

```
>> 2^B
```

- Calculamos la exponencial de B por dos métodos

```
>> expm(B)
>> exp(1)^B
```

- Calculamos el logaritmo neperiano de B por dos métodos

```
>> logm(B)
>> funm(B,'log')
```

- Calculamos la raíz cuadrada de B por tres métodos

```
>> sqrtm(B)
>> funm(B,'sqrt')
>> B^.5
```

- Calculamos el seno y coseno de B

```
>> funm(B,'sin')
>> funm(B,'cos')
```



## Descomposición de matrices

| Función | *Comentario* |
|---|---|
| `[V,D]=eig(A)` | $AV = VD$, $D$ diagonal |
| `[T,B]=balance(A)` | $TB = AT$, $\mathrm{eig}(A) \approx \mathrm{eig}(\mathrm{B})$ |
| `[U,T]=schur(A)` | $UT = AU$, $U'U = I$, $T$ triangular superior |
| `[L,U,P]=lu(A)` | $PA = LU$, $P$ permutación, $L$ triangular inferior, $U$ triangular superior |
| `R=chol(A)` | $R'R = A$ para $A$ definida positiva, $R$ triangular superior |
| `[Q,R,P]=qr(A)` | $AP = QR$, $P$ permutación, $Q$ ortogonal, $R$ triangular superior |
| `[V,J]=jordan(A)` | $AV = VJ$, $J$ matriz de Jordan |
| `pinv` | Pseudoinversa de una matriz |
| `poly` | Polinomio característico de una matriz |

## Ejemplo de descomposición de matrices *DAlg3.m* (I)

- Formamos una matriz cuadrada aletoria de dimensión 3

```
>> A=randn(3)
```

- Calculamos su descomposición en valores propios con `svd`

```
>> [V,D]=svd(A)
```

- Comprobamos la descomposición:

```
>> A*V-V*D
```

- Calculamos la matriz balanceada de A

```
>> [T,B]=balance(A)
```

- Comprobamos la descomposición

```
>> [B T\A*T]
>> eig(A)
>> eig(B)
```



## Ejemplo de descomposición de matrices *DAlg3.m* (II)

- Calculamos la descomposición de Schur de A

```
>> [U,T]=schur(A)
```

- Comprobamos la descomposición

```
>> [U*T*U' A]
>> U*U'
```

- Calculamos la descomposición QR de A

```
>> [Q,R,E]=qr(A)
```

- Comprobamos la descomposición

```
>> [Q*R A*E]
>> Q*Q'
```

- Calculamos la descomposición LU de A

```
>> [L,U,P]=lu(A)
```

- Comprobamos la descomposición

```
>> [L*U P*A]
```

## Ejemplo de descomposición de matrices *DAlg3.m* (III)

- Calculamos la descomposición SVD de A

```
>> [U,S,V]=svd(A)
```

- Comprobamos la descomposición

```
>> [U*S*V' A]
>> U'*U
>> V*V'
```

- Para calcular el factor de Choleski necesitamos una matriz definida positiva que calculamos premultiplicando A por su transpuesta

```
>> AA=A'*A
>> R=chol(AA)
```

- Comprobamos la descomposición

```
>> [R'*R AA]
```

- Comprobamos que si la matriz no es definida positiva no tiene factor de Choleski y se obtiene un error

```
>> R=chol(A)
```



## Solución de sistemas de ecuaciones

| Función | *Comentario* |
|---|---|
| `X=A/B` | Resuelve $XA = B$ |
| `X=A\B` | Resuelve $AX = B$ por mínimos cuadrados |
| `X=lsqnonneg(A,b)` | Solución de mínimos cuadrados de $Ax = b$, $x \geq 0$ |
| `X=linsolve(A,B)` | Resuelve $AX = B$, $A$ matriz cuadrada, $B$ matriz |
| `r=roots(p)` | Raíces de un polinomio $p$ |
| `p=poly(r)` | Polinomio de raíces $v$ |
| `x=fzero(fun,x0)` | Calcula un cero de la función fun próximo a x0 |

## Ejemplo solución de sistemas *DAlg4.m* (I)

- Obtenemos las raíces del polinomio $p(x) = x^3 + 2x^2 - 3x + 1$

```
>> v = roots([1 2 -3 1])
```

- Obtenemos el polinomio que tiene raíces $-1$, $+2$, $+j$ y $-j$

```
>> p = poly([-1 2 j -j])
```

- Sea el sistema de ecuaciones lineales

$$\begin{aligned} x + 2y + 3z &= 3 \\ 2x + 3y + z &= 1 \\ x + y + 5z &= 5 \end{aligned}$$

- Para resolverlo se forman las matrices A y b

```
>> A = [1 2 3; 2 3 1; 1 1 5]
>> b = [3 1 5]'
```

- La solución es:

```
>> X=A\b
```



## Ejemplo solución de sistemas *DAlg4.m* (II)

- La solución de mínimos cuadrados no negativa es:

```
>> Xn=nlsqnoneg(A,b)
```

- La ecuación $x\sin(x) = 1/2$ puede resolverse con `fzero` en el entorno de los puntos 2, 4 y 6:

```
>> [fzero('x*sin(x)-.5',2) fzero('x*sin(x)-.5',4) ...
>>  fzero('x*sin(x)-.5',6)]
```

- En un fichero "script" de nombre `Gejer1.m` realizar el siguiente ejercicio. Dada la siguiente matriz:

$$A = \begin{bmatrix} 2/3 & 2/5 & 2/7 & 2/9 & 2/11 \\ 2/5 & 2/7 & 2/9 & 2/11 & 2/13 \\ 2/7 & 2/9 & 2/11 & 2/13 & 1/15 \\ 2/9 & 2/11 & 2/13 & 2/15 & 2/17 \\ 2/11 & 2/13 & 2/15 & 2/17 & 1/19 \end{bmatrix}$$

  - Autovalores, autovectores, polinomio característico y número de la condición.
  - Hallar al inversa de la matriz.
  - Hallar la descomposición por los siguientes métodos: Jordan, Schur, LU, QR, Choleski y SVD. Comprobar si es posible la descomposicióny si los valores obtenidos son ciertos.
  - Si $b = [1, 3, 5, 7, 9]'$ resolver el valor de debe tener $x$ para que se cumpla la ecuación $Ax = b$.
- En un fichero "script" de nombre `Gejer2.m` realizar el siguiente ejercicio. Sea $x$ y $n$ dos vectores aleatorios de distribución uniforme entre $[0, 1]$ de 100 elementos.
  - Fijar un valor para los parámetros $[a, b, c]$.
  - Obtener el valor de $y$ de la formula $y = a * x + b * x^2 + c * x^3 + 0{,}1 * n$.
  - Estimar el valor de los parámetros $[a, b, c]$ a partir del valor de $x$ e $y$ usando mínimos cuadrados. Se considera que $n$ es un ruido.

| Función | *Comentario* |
|---|---|
| `cov` | Varianza de un vector |
| `corrcoef` | Coeficientes de correlación (normalizados) |
| `conv` | Convolución de datos, producto de polinomios |
| `diff` | Diferencias entre elementos de un vector |
| `gradient` | Derivadas parciales numéricas de una matriz |
| `del2` | Laplaciano discreto de una matriz |
| `filter` | Filtro FIR y IIR de datos |
| `ltitr` | Respuesta lineal |



## Prácticas con funciones de relación y filtros, *Bscript2.m* (I)

- Varianza de un vector

```
>> load count.dat
>> a= cov(count(:,1));
```

- Coeficiente de correlación:

```
>> b= corrcoef(count);
```

- Convolución entre dos vectores:

```
>> c= conv([1,2,3],[4,5,6]);
>> d= conv(count(:,1),count(:,2));
```

- Diferencial., Derivada aproximada:

```
>> a=diff(count);
>> size(a), size(count)
```

- Gradientes, derivada parcial aproximada:

```
>> [px,py]= gradient(count);
```

- Laplaciano discreto de un vector $del^2u = (d^2u/dx^2 + d^2/dy^2)$:

```
>> lp= del2(count)
```

- Filtro FIR y IIR de vectores:

```
>> b=[1,1,1]/3; a=1;
>> f= filter(b,a,count);
```

- Simulación de un sistema lineal en ecuaciones de estado:

```
>> A= [0.9,0; 0,0.9]; B=[0,1]';
>> x= ltitr(A,B,ones(100,1));
```



## Análisis en frecuencia

| Función | *Comentario* |
|---|---|
| fft | Transformada de Fourier discreta |
| fft2 | Transformada de Fourier en dos dimensiones |
| ifft | Inversa transformada de Fourier |
| ifft2 | Inversa transformada de Fourier en dos dimensiones |
| abs | Magnitud |
| angle | Ángulo |
| fftshift | Mueve el retraso cero al centro del espectro |

- Toma de datos:

```
>> load sunspot.dat;
>> year= sunspot(:,1); wolfer= sunspot(:,2);
```

- Transformada de Fourier, se le quita el primer dato:

```
>> y= fft(wolfer); y(1)=[];
```

- Gráficas con eje frecuencia $[0, \pm$ ínf$, 0]$ y $[-$ ínf$, 0,$ ínf$]$:

```
>> figure; subplot(2,1,1); plot(abs(y))
>> subplot(2,1,2); plot(fftshift(abs(y)));
```

- Gráficas en función de la frecuencia de Nyquist:

```
>> N= lenght(y); power = abs(Y(1:N/2)).^2;
>> nyquist = 1/2; freq = (1:N/2)/(N/2)*nyquist;
>> figure; subplot(2,1,1); plot(freq,power);
>> subplot(2,1,2); plot(freq,unwrap(angle(y(1:N/2))));
```



## Ejercicios de filtro y transformada de Fourier

- En un fichero "script" de nombre `Hejer1.m` responder a las siguientes preguntas.
  - Obtener una señal de $[0, 3]$ segundos con periodo de muestreo 0,001 s. $y = sin(2 * \pi * 2 * t) + 0{,}5 * sin(2 * \pi * 5 * t + \pi/3) + 0{,}1 * sin(2 * \pi * 50 * t)$
  - Filtrar la señal para eliminar el componente de alta frecuencia, producto del acoplamiento con la red a 50 Hz. Proponer para ello diferentes tipos de filtros.
  - Obtener la transformada de Fourier de la señal filtrada y sin filtrar viendo las diferencias. Se precisa que la frecuencia cero esté en el centro de la gráfica.

## Funciones para polinomios

| Función | *Comentario* |
|---|---|
| `conv` | Producto de polinomios |
| `deconv` | División de polinomios |
| `poly` | Definición de polinomios por raíces |
| `polyder` | Derivada de polinomios |
| `polyfit` | Interpola por mínimos cuadrados |
| `polyval` | Valor polinomio en un punto |
| `polyvalm` | Valor polinomio con matrices |
| `residue` | Fracciones parciales |
| `roots` | Raíces de un polinomio |

## Prácticas con funciones de polinomios, *Cscript1.m* (I)

- Polinomio y sus raíces. Coeficientes a partir de raíces:

```
>> p = [1 0 -2 -5]; r= roots(p);
>> poly(r)
```

- Polinomio característico:

```
>> A = [1.2 3 -0.9; 5 1.75 6; 9 0 1];
>> poly(A)
```

- Valor del polinomio en un número o matriz:

```
>> polyval(p,5)
>> polyvalm(p,A)  % p(A)= A^3-2*A-5*I
```

- Convolución (producto) y deconvolución:

```
>> a= [1,2,3]; b= ones(1,5); c= conv(a,b);
>> [q, r]= deconv(c, a);
```



## Prácticas con funciones de polinomios, *Cscript1.m* (II)

- Derivada de un polinomio, del producto y de la división:

```
>> q = polyder(p)
>> c = polyder(a,b)  % derivada del producto
>> [q,d] = polyder(a,b)  % derivadas de la división
```

- Residuos de un polinomio:

```
>> b = [-4 8]; a = [1 6 8];
>> [r,p,k] = residue(b,a)
```

## Prácticas regresiones basadas mínimos cuadrados, *Cscript2.m* (I)

- Datos para la regresión:

```
>> t = [0 .3 .8 1.1 1.6 2.3]'; y = [0.5 0.82 1.14 1.25
    1.35 1.40]';
>> plot(t,y,'o'), grid on;
```

- Regresión, $y = a(0) + a(1) * t + a(2) * t^2$:

```
>> X= [ones(size(t)), t, t.^2]; a= X\y;
>> that = (0:0.1:2.5)'; yhat = [ones(size(that)) that
    that.^2]*a;
>> plot(that,yhat,'-',t,y,'o'), grid on;
```

Solución: $a = inv(X' * X) * X' * y$

- Regresión exponencial: $y = a(0) + a(1) * exp(-t) + a(2) * t * exp(-t)$:

```
>> X = [ones(size(t)) exp(-t) t.*exp(-t)]; a = X\y;
>> that = (0:0.1:2.5)';
>> yhat = [ones(size(that)) exp(-that) that.*exp(-that)
    ]*a;
>> figure; plot(that,yhat,'-',t,y,'o'), grid on;
```



## Prácticas regresiones basadas mínimos cuadrados, *Cscript2.m* (II)

- Regresiones multiples: $y = a(0) + a(1) * x1 + a2 * x2$:

```
>> x1 = [.2 .5 .6 .8 1.0 1.1]'; x2 = [.1 .3 .4 .9 1.1
    1.4]';
>> y = [.17 .26 .28 .23 .27 .24]';
>> X = [ones(size(x1)) x1 x2]; a = X\y;
```

- $y$ estimada y error máximo:

```
>> yhat = X*a; MaxErr = max(abs(yhat - y))
```

- Obtención de los coeficientes de un polinomio que se aproxime:

```
>> x = [1 2 3 4 5]; y = [5.5 43.1 128 290.7 498.4];
>> p = polyfit(x,y,3)
>> x2 = 1:.1:5; y2 = polyval(p, x2);
>> plot(x, y, 'o', x2, y2); grid on
```

- Sea $x$ un vector aleatorio uniforme de 1000 componentes. Comprobar en un fichero "script" de nombre `Iejer1.m` que la convolución de ese vector con el polinomio $[1,1,1]/3$ da el mismo resultado que la señal obtenida con `y= filter([1,1,1]/3,1,x)`. Dar una explicación a este hecho.
- En un fichero "script" de nombre `Iejer2.m` realizar el siguiente ejercicio. Sea $t$ un intervalo de tiempo entre $[0,3]$ con un periodo de muestreo de 0,1 s y $n$ un vector aleatorio de distribución uniforme entre $[0,1]$ del mismo número de elementos.
  - Fijar un valor para los parámetros $[a,b,c]$.
  - Obtener el valor de $y$ de la formula $y = a + b * x + c * x^2 + 0{,}1 * n$.
  - Estimar el valor de los parámetros $[a,b,c]$ a partir de la función `polyfit` y con mínimos cuadrados.





| Función | *Comentario* |
|---|---|
| `interp1` | Interpolación en una dimensión |
| `inter2, inter3` | Interpolación en dos y tres dimensiones |
| `interpft` | Interpolación una dimensión fft. |
| `mkpp` | Compone un spline a partir de propiedades |
| `spline` | Genera splines cubicos |
| `pchip` | Genera splines cúbico de Hermite |
| `ppval` | Valor de un spline en puntos |
| `unmkpp` | Propiedades de un spline |
| `mmppint, mmppder` | Spline integral y derivada |

- Interpolación de datos en una dimensión por distintos métodos:

```
>> x = [1 2 3 4 5]; y = [5.5 43.1 128 290.7 498.4];
>> s= interp1(x, y, 2.5, 'linear')
>> s= interp1(x, y, 2.5, 'cubic')
>> s= interp1(x, y, 2.5, 'spline')
>> s= interp1(x, y, 2.5, 'nearest')
>> xhat= linspace(1,5,100)';
>> yhat= interp1(x, y, xhat, 'spline');
>> plot(xhat, yhat, '-', x, y, 'o');
```

- Interpolación de datos en dos dimensiones por distintos métodos:

```
>> [x,y] = meshgrid(-3:1:3);
>> z = peaks(x,y); surf(x,y,z)
>> [xi,yi] = meshgrid(-3:0.25:3);
>> zi1 = interp2(x, y, z, xi, yi, 'nearest');
>> surf(xi, yi, zi1);
>> zi2 = interp2(x, y, z, xi, yi, 'bilinear');
>> surf(xi, yi, zi2);
>> zi3 = interp2(x, y, z, xi, yi, 'bicubic');
>> surf(xi, yi, zi1);
```



## Prácticas splines, *Cscript3.m* (I)

- Datos:

```
>> x= 0:12; y= tan(pi*x/25);
>> xi= linspace(0,12,100);
```

- Interpola datos:

```
>> yp= spline(x,y,xi);
>> plot(x,y,'o',xi,yp);
```

- Genera los coeficientes de spline, interpola y propiedades:

```
>> pp= spline(x,y)
>> yp= ppval(pp, xi);
>> [break, coefs, npolys, ncoefs, dim]= unmkpp(pp)
>> pp= mkpp(breaks, coefs) % composición
```

- Spline con el método de Hermite:

```
>> x=[0,2,4,5,7.5,10]; y= exp(-x/6).*cos(x);
>> ch= pchip(x,y); ych= ppval(ch, xi);
>> plot(x, y, 'o', xi, yp, ':', xi, ych);
```

- Spline integral y derivada de otro:

```
>> ppi= mmppint(pp); ppd= mmppder(pp);
>> yi= ppval(ppi,xi); yd= ppval(ppd,xi);
>> plot(x, y, 'o', xi, yp, '-', xi, yi, '--', xi, yd, '
    -.');
```

- Splines para dos dimensiones:

```
>> t= linspace(0,3*pi,15);
>> x= sqrt(t).*cos(t); y= sqrt(t).*sin(t);
>> ppxy= spline(t,[x;y])
>> ti= linspace(0, 3*pi, 1000);
>> xy= ppval(ppxy,ti);
>> plot(x, y, 'd', xy(1,:), xy(2,:));
```



## Funciones para interpolar superficies e hiperplanos

| Función | *Comentario* |
|---|---|
| meshgrid | Mallado de dos o tres dimensiones |
| ndgrid | Mallado de dimensión n |
| surf, mesh | Dibuja superficies y mallados |
| slide | Dibuja cortes dentro de un volumen |
| griddata | Interpolación una superficie |
| griddata3 | Interpolación una hipersuperficie, datos orden 3 |
| griddatan | Interpolación una hipersuperficie, datos orden n |
| interpn | Interpolación en n dimensiones |

- Datos $X, Y$ reales e interpolados:

```
>> x1 = -2:0.2:2; x2 = -2:0.25:2;
>> [X1, X2] = ndgrid(x1,x2);
>> xi1 = -2:0.1:2; xi2 = -2:0.1:2;
>> [Xi1, Xi2] = ndgrid(xi1,xi2);
```

- $Z$ real e interpolada:

```
>> Z = X2.*exp(-X1.^2 -X2.^2);
>> Zi = griddata(X1, X2, Z, Xi1, Xi2)
>> subplot(2,1,1); mesh(X1, X2, Z);
>> subplot(2,1,2); mesh(Xi1, Xi2, Zi);
```



## Prácticas de interpolación de datos.

- Representar en un "script" de nombre `Iejer3.m` la diferencia entre el polinomio interpolador cúbico hermitiano a trozos y el polinomio interpolador spline cuando x y t varían entre $-3$ y 3 ($t$ varia entre décima y décima) y $x = [-1, -1, -1, 0, 1, 1]$.
- Se considera un conjunto de temperaturas medidas sobre las cabezas de los cilindros de un motor que se encuentra en período de pruebas para utilizar en coches de carreras. Los tiempos de funcionamiento del motor en segundos y las temperaturas en grados Fahrenheit son las siguientes:

$$Tiempo = [0, 1, 2, 3, 4, 5]$$
$$Temperaturas = [0, 20, 60, 68, 77, 110]$$

Realizar una regresión lineal en un fichero `Iejer4.m` que ajuste las temperaturas en función de los tiempos. Realizar también el ajuste mediante regresiones polinómicas de grados 2, 3 y 4 representando los resultados.

## Funciones de optimización e integración

| Función | *Comentario* |
|---|---|
| `fplot` | Dibuja la función |
| `fminbnd` | Minimiza función con una variable con restricciones |
| `fminsearch` | Minimiza función con varias variables |
| `fzero` | Encuentra el cero en función con una variable |
| `optimset, optimget` | Parámetros de resolución |
| `quad` | Integración numérica, Simpson |
| `quadl` | Integración numérica, Lobatto |
| `dblquad` | Integración numérica, doble integral |
| `triplequad` | integración numérica, triple integral |

## Prácticas funciones de optimización, *Dscript1.m* (I)

- Parámetros de la optimización:

```
>> help optimset
```

- Función definida como anónima (versión 7):

```
>> humps= @(x) 1./((x - 0.3).^2+0.01)+1./((x-0.9)
    .^2+0.04)-6;
```

- Función definida como anónima (versiones anteriores):

```
>> humps= inline('1./((x - 0.3).^2+0.01)+1./((x-0.9)
    .^2+0.04)-6');
```

- Entradas y salida de la función, representación:

```
>> x= linspace(-.5,1.5,100); y= humps(x);
>> fplot(humps,[-5 5]); grid on;
```

- Ejemplo de modificación de parámetros (ver valor en cada iteración):

```
>> option= optimset('Display','iter'); optimget
```



## Prácticas funciones de optimización, *Dscript1.m* (II)

- Algorithmo: Nelder-Mead simplex, $x \in [0{,}3 < x < 1]$, espacio de búsqueda de una dimensión:

```
>> x = fminbnd(humps, 0.3, 1, option);
```

- Espacio de búsqueda de varias dimensiones:

```
>> t_var= @(x) x(1).^2 + 2.5*sin(x(2)) - x(3)^2*x(1)^2*
    x(2)^2;
```

- Mínimo cercano a $v$, valor desde donde se empieza a buscar:

```
>> v = [-0.6 -1.2 0.135];
>> [vmin, values, flag, output] = fminsearch(t_var,v)
```

- Punto $f(x) = \sin(3 * x) = 0$ cercano a $x = 2$, donde se empieza a buscar:

```
>> x = fzero(@(x) sin(3*x),2)
```

## Prácticas funciones de optimización, *Dscript1.m* (III)

- Función de control del algoritmo:

```
>> options = optimset('OutputFcn', @outfun);
```

- Forma de la función de control, saca gráficos en cada iteración:

```
function stop = outfun(x, optval, state)
%optval campos: funcount, fval, interation, procedure
%state: 'init', 'interrupt', 'iter', 'done'
%stop: false, true
   stop = []; hold on;
   plot(x(1),x(2),'.');
   drawnow
```

- Prueba con la minimización anterior:

```
>> v = [-0.6 -1.2 0.135]; %Empieza a buscar en v
>> [vmin, values, flag, output] = fminsearch(t_var,v,
    options)
```



## Prácticas funciones de integración, *Dscript1.m*

- Integral simple:

```
>> quad(humps, -1, 2) %Integración simple
>> quadl(humps, -1, 2) %Integración, mayor exactitud
```

- Integral doble:

```
>> out= @(x,y) y*sin(x) + x*cos(y);
>> xmin= pi; xmax= 2*pi; ymin= 0; ymax= pi;
>> result = dblquad(out, xmin, xmax, ymin, ymax)
```

- Restricción del problema: $\text{mín}_x\{f(x)\}$, $g(x) > 0$,

$$\begin{array}{l} \text{if} \quad g(x) < 0, cost = f(x), \\ \text{else} \quad cost = f(x) * P(g(x)), \quad P(g(x)) > 0. \end{array}$$

- Problema minimax: $\text{mín}_x\{\text{máx}_f\{[f_1(x), \ldots, f_n(x)]\}\}$,

$$cost = \text{máx}([f_1(x), \ldots, f_n(x)]).$$

- Sistema de ecuaciones no lineales: $\{f_1(x) = 0, \ldots, f_n(x) = 0\}$:

$$cost = \text{máx}(\text{abs}([f_1(x), \ldots, f_n(x)])).$$

- Problema multiobjetivo: $\text{mín}_x[f_1(x), \ldots, f_n(x)]$.

$$cost = \sum_i p_i * f_i(x).$$

  La solución depende de los $p_i$ elegidos.



## Prácticas con funciones de optimización (I)

- Minimizar la función $f(x) = (x-3)^2 - 1$ en el intervalo $(0, 5)$.
- Encontrar el valor de x que minimiza el valor máximo de $[f_1(x), \ldots, f_5(x)]$,

$$\begin{array}{c} f_1(x) = 2x_1^2 + x_2^2 - 48 * x_1 - 40x_2 + 304 \\ f_2(x) = -x_2^2 - 3x_2^2 \\ f_3(x) = x_1 + 2x_2 - 18 \\ f_4(x) = -x_1 - x_2 \\ f_5(x) = x_1 + x_2 - 8 \end{array}$$

- Minimizar la función siguiente $f(x) = 3x_1^2 + 2x_1 * x_2^2$.

- Encontrar en un fichero "script" de nombre `Jejer1.m` los valores de x que minimizan la función $f(x)$ sujeta a las restricciones $k_1(x, w_1)$ y $k_2(x, w_2$ con $w_1$ y $w_2$ en $[1, 100]$. La función y las restricciones se definen en el problema y el punto inicial es $(0{,}5, 0{,}2, 0{,}3)$,

$$f(x) = (x_1 - 0{,}5)^2 + (x_2 - 0{,}5)^2 + (x_2 - 0{,}5)^2$$
$$k(x, w_1) = \sin(w_1 x_1)\cos(w_2 x_2) - 1/100(w_1 - 50)^2 - sin(w_1 x_3) - x_3 \leq 1$$
$$k(x, w_2) = \sin(w_2 x_2)\cos(w_2 x_1) - 1/100(w_2 - 50)^2 - sin(w_2 x_3) - x_3 \leq 1$$

- Dado el conjunto de datos:

$$xdata = [3{,}6, 7{,}7, 9{,}3, 4{,}1, 8{,}6, 2{,}8, 1{,}3, 7{,}9, 10{,}0, 5{,}4]$$
$$ydata = [16{,}5, 150{,}6, 263{,}1, 24{,}7, 208{,}5, 9{,}9, 2{,}7, 163{,}9, 325{,}0, 54{,}3]$$

se trata de encontrar los coeficientes $x$ que minimizan la función $ydata(i)$ del tipo,

$$ydata(i) = x(1)xdata(i)^2 + x(2)\sin(xdata(i)) + x(3)xdata(i)^2$$

Los resultados se escribirán en un fichero "script" de nombre `Jejer2.m`.



## Contenidos

- Matlab y sus aplicaciones en la ciencia y la ingeniería, (César Pérez). Prentice Hall.
- Mastering Matlab 7, (Duane Hanselman, Bruce Littlefield). Prentice Hall, Internaltional Edition.