Manejo básico de Matlab

Mariano Mateos Alberdi

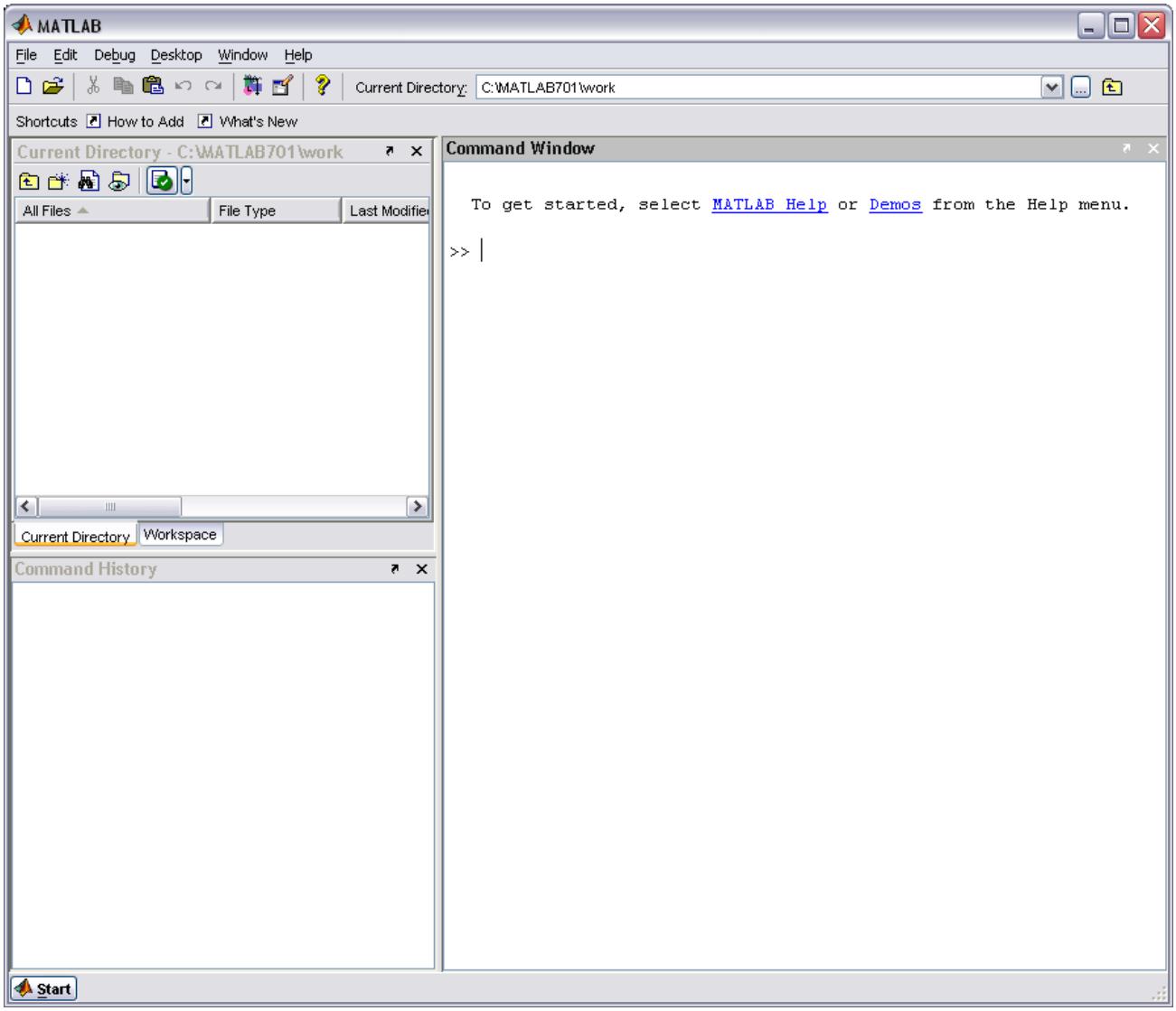
Complementos de Matemática Aplicada
Ingeniería Técnica Industrial
Universidad de Oviedo

Curso 2007-2008

Índice

1. Manejo básico

Según abrimos MATLAB, versión 7 para Windows, nos aparecen 3 ventanas. La ventana de la derecha según miramos (*Command Window*) es la que utilizamos para escribir los comandos. En la de arriba a la izquierda *Current Directory* aparecen los ficheros que hay en la carpeta de trabajo. Bajo ésta, está oculta una ventana con el título *Workspace*, donde nos aparecen las variables que estamos utilizando y en la de abajo a la izquierda (*Command History*), un historial con los comandos que hemos escrito. Encima de estas ventanas hay un recuadro (*Current Directory*) donde aparece la carpeta o directorio de trabajo, es decir, el lugar donde MATLAB buscará nuestros programas. Si durante el trabajo perdemos esta estructura de ventanas y la queremos recuperar, basta con pinchar en Desktop → Desktop Layout → Default.



1.1. Calculadora

Para efectuar una operación, la escribimos en la *Command Window* y le damos a la tecla Enter

Ejemplo 1.1.1 *Si queremos calcular* $2+2$, $1256-34$, 15×365, $1/23$, 2^{10}, $\cos(1)$, $\sqrt{2}$, e^2, $\ln(3)$, $\log(3)$ *escribiremos:*

```
>> 2+2
ans =
     4
>> 1256-34
ans =
        1222
>> 15*365
ans =
        5475
>> 1/23
ans =
    0.0435
>> 2^10
ans =
        1024
>> cos(1)
ans =
    0.5403
>> sqrt(2)
ans =
    1.4142
>> exp(2)
ans =
    7.3891
>> log(3)
ans =
    1.0986
>> log10(3)
ans =
    0.4771
```

Podemos usar paréntesis. No podemos saltarnos ningún símbolo. Para calcular $2(\sqrt[3]{2}+\operatorname{arc\,tg}(1))$ *escribiremos:*

```
>> 2*(2^(1/3)+atan(1))
ans =
    4.0906
```

Observa lo que ocurre si olvidamos el $*$*:*

```
>> 2(2^(1/3)+atan(1))
??? 2(2^(1/3)+atan(1))
```

```
     |
Error: Unbalanced or misused parentheses or brackets.
```

Corrección de errores

Si nos equivocamos al escribir una orden, no es necesario reescribirla por completo. Hay dos maneras rápidas de corregirla:

1. Darle a la tecla del cursor ↑ hasta que aparezca la orden a editar, y después editarla.
2. Arrastrarla desde la ventana *Command History* hasta la *Command Window*, y después editarla.

Notación científica

MATLAB escribe muchas veces números de la forma `1.2219e+003`. Esto quiere decir $1{,}2219 \times 10^3$, y es el número 1221,9. Nosotros también podemos escribir números así.

Ejemplo 1.1.2 *Para escribir un millón, un millardo, un billón y el número de Avogadro, escribiremos:*

```
>> 1e6
ans =
     1000000
>> 1e9
ans =
  1.0000e+009
>> 1e12
ans =
  1.0000e+012
  >> 6.022e23
ans =
  6.0220e+023
```

Y para escribir una millonésima y la constante de gravitación universal escribiremos:

```
>> 1e-6
ans =
  1.0000e-006
>> 6.67392e-11
ans =
  6.6739e-011
```

Número de decimales

¿Dónde ha ido la última cifra significativa de la constante de gravitación? Por defecto, MATLAB nos muestra 5 cifras significativas. Si queremos ver más, sólo hay que cambiar el formato. Observa que el formato corto redondea la salida.

Ejemplo 1.1.3

```
>> format long
>> 2.41396e-11
ans =
    2.413960000000000e-011
>> format short
>> 2.41396e-11
ans =
  2.4140e-011
```

Precisión

Con el formato largo vemos 16 cifras significativas. Esa es *grosso modo* la precisión de MATLAB. Esto quiere decir que dos números tienen órdenes de magnitud que difieren en por lo menos 16 cifras decimales, no se pueden sumar.

Ejemplo 1.1.4

```
>> 1+1e-16-1
ans =
     0
```

Además, esto da lugar a la aparición de **errores de redondeo** en operaciones que sí se pueden hacer.

Ejemplo 1.1.5 *Observa qué ocurre cuando sumamos y restamos 1 a una cantidad "pequeña":*

```
>> 1+1e-15-1
ans =
    1.110223024625157e-015
>> 1+1e-14-1
ans =
    9.992007221626409e-015
>> 1+1e-13-1
ans =
    9.992007221626409e-014
>> 1+1e-3-1
ans =
    9.999999999998899e-004
>> 1+1e-2-1
ans =
   0.01000000000000
```

He aquí algunas de las órdenes de calculadora científica que usa MATLAB.

MATLAB	**Función**	MATLAB	**Función**
`exp(x)`	e^x	`abs(x)`	$\|x\|$
`log(x)`	$\ln(x)$	`fix(x)`	Redondeo hacia cero
`log10(x)`	$\log_{10}(x)$	`floor(x)`	Redondeo hacia $-\infty$
`log2(x)`	$\log_2(x)$	`ceil(x)`	Redondeo hacia $+\infty$
`sqrt(x)`	$\sqrt{x}$	`round(x)`	Redondeo hacia el entero más próximo
		`rem(m,n)`	resto de dividir m entre n

y las funciones trigonométricas:

MATLAB	**Fun.**	MATLAB	**Fun.**	MATLAB	**Func.**	MATLAB	**Fun.**
`sin(x)`	$\text{sen}(x)$	`asin(x)`	$\text{asen}(x)$	`sinh(x)`	$\text{senh}(x)$	`asinh (x)`	$\text{asenh}(x)$
`cos(x)`	$\cos(x)$	`acos(x)`	$\text{acos}(x)$	`cosh(x)`	$\cosh(x)$	`acosh (x)`	$\text{acosh}(x)$
`tan(x)`	$\tan(x)$	`atan(x)`	$\text{atan}(x)$	`tanh(x)`	$\tanh(x)$	`atanh (x)`	$\text{atanh}(x)$
`cot(x)`	$\cot(x)$	`acot(x)`	$\text{acot}(x)$	`coth(x)`	$\coth(x)$	`acoth (x)`	$\text{acoth}(x)$
`sec(x)`	$\sec(x)$	`asec(x)`	$\text{asec}(x)$	`sech(x)`	$\text{sech}(x)$	`asech (x)`	$\text{asech}(x)$
`csc(x)`	$\csc(x)$	`acsc(x)`	$\text{acsc}(x)$	`csch(x)`	$\text{csch}(x)$	`acsch (x)`	$\text{acsch}(x)$

Ejercicio 1 *Calcula* $|\,\text{sen}(6)(\ln(4)+6{,}7)|$, $e^{-1{,}5}$. *Redondea al entero más próximo* $(2+3/7)^{1/5}$.

1.2. Uso de la ayuda

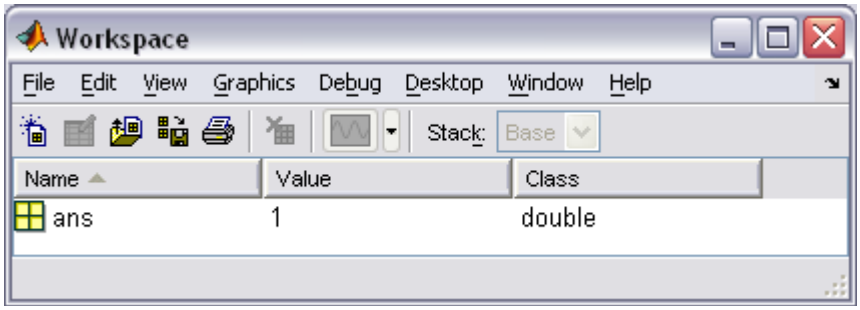
MATLAB es un programa muy bien documentado. Para obtener instrucciones sobre el uso de un comando de Matlab teclearemos `help comando` ó `doc comando`. La primera da una breve descripción en la ventana de comandos. La segunda abre una ventana de ayuda mucho más completa.

Ejemplo 1.2.1

```
>> help sin
 SIN    Sine.
    SIN(X) is the sine of the elements of X.

    See also asin, sind.

    Overloaded functions or methods (ones with the same name in other directorie
       help sym/sin.m

    Reference page in Help browser
       doc sin
```

1.3. Variables

Introducir variables nos ofrece nuevas posibilidades en MATLAB. Nos permiten recordar y usar los cálculos ya hechos. Si pinchas en la pestaña *Workspace* aparecerá una ventana donde se muestran las variables que hay guardadas en memoria, y su valor. Ahora mismo sólo está la variable `ans`, que es la variable que usa MATLAB por defecto para dar una respuesta (*answer*).



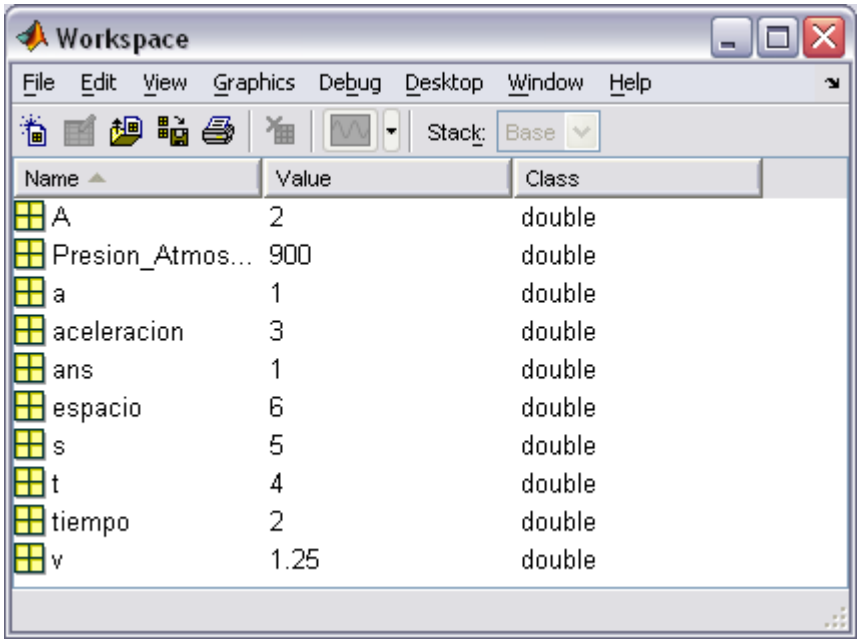
Según vamos metiendo variables, se van cargando en memoria:

```
>> a=1
a =
     1
>> A=2
A =
     2
>> Presion_Atmosferica=900
Presion_Atmosferica =
   900
```

Observa que podemos escribir **varias órdenes** en cada línea separándolas mediante una coma o un punto y coma. Al escribir tras una orden **punto y coma** (`;`), la orden se ejecuta, la variable se guarda en memoria, pero no aparece nada en pantalla.

```
>> s=5,t=4,v=s/t
s =
     5
t =
     4
v =
    1.2500
>> aceleracion=3;tiempo=2;espacio=(1/2)*aceleracion*tiempo^2
espacio =
     6
```

Ahora la ventana *Workspace* es así:



Las reglas que se utilizan para nombrar las variables son las siguientes:

- MATLAB distingue entre letras mayúsculas y minúsculas. Las variables `area`, `Area`, `AREA`, `arEa` serían variables distintas.
- El nombre de una variable puede contener un máximo de 31 caracteres ignorándose los posteriores.
- El nombre de una variable debe empezar necesariamente por una letra, aunque puede contener letras números y el guión de subrayado, nunca puede contener operadores (+,*,...), espacios en blanco ni signos de puntuación.
- No deben nombrarse variables con funciones con significado específico en MATLAB, por ejemplo `cos`=3 construiría una variable `cos` cuyo valor es 3, y a partir de este momento no podríamos calcular el coseno de un ángulo hasta que no borrásemos la variable `cos`.

Cómo borrar variables

Para borrar variables se utiliza la orden `clear` para borrar todas las variables definidas hasta el momento; si a la orden se le añade una lista de variables (separadas por espacios en blanco) sólo se borrarán las variables de la lista.

```
» clear t s
```

También se pueden borrar variables pinchando con el botón derecho del ratón sobre la variable en la ventana *Workspace* y seleccionando *Delete*.

Algunas variables predefinidas en MATLAB

Algunas variables ya están definidas en MATLAB:

Nombre	Significado
`ans`	Almacena el último resultado no asignado a una variable
`eps`	Epsilon de la máquina
`pi`	π
`i` y `j`	Unidad imaginaria
`inf`	∞
`NaN`	No es un número
`realmin`	Mínimo número mayor que cero
`realmax`	Número más grande
`date`	Fecha
`true`	1 (ocupa sólo 1 byte de memoria)
`false`	0 (ocupa sólo 1 byte de memoria)

El épsilon de la máquina es el número positivo más pequeño que sumado a 1 genera un número mayor que 1 en el ordenador, (en un PC `eps`=2.220446049250313e-016) y `NaN` (*Not a Number*) representa una expresión indeterminada, como puede verse en el siguiente ejemplo:

```
» (2-2)/(3-3)
```

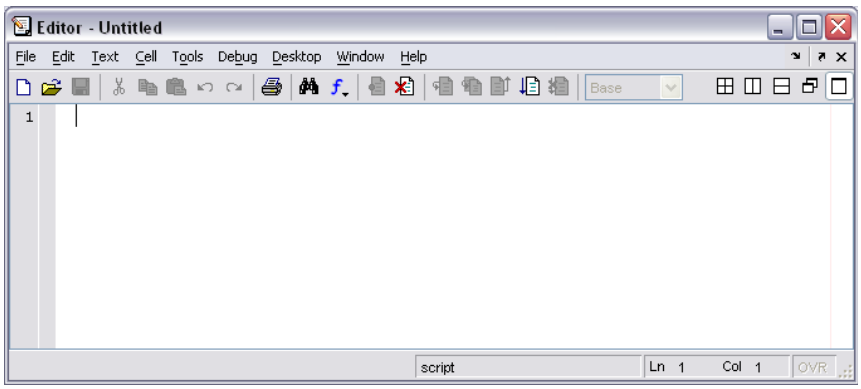
Ejercicio 2 *Sea* $M = 1{,}989 \cdot 10^{30} kg$ *la masa del sol,* $m = 5{,}972 \cdot 10^{18} kg$ *la masa de la tierra,* $G = 6{,}67392 \cdot 10^{-11} m^3/s^2 kg$ *la constante de gravitación universal y* $d = 1{,}427 \cdot 10^{11} m$ *la distancia tierra sol. Introduce todas las variables en memoria y calcula la fuerza de atracción* $F = GMm/d^2$

Ejercicio 3 MATLAB *trabaja de manera natural con complejos. Demuestra que* $e^{i\pi} = -1$.

1.4. Programas

Escribir los comandos directamente en la ventana de trabajo de MATLAB resulta cómodo para ejecutar unas pocas órdenes. Sin embargo, cuando tenemos que ejecutar un gran número de instrucciones de forma consecutiva, necesitamos crear un **programa**.

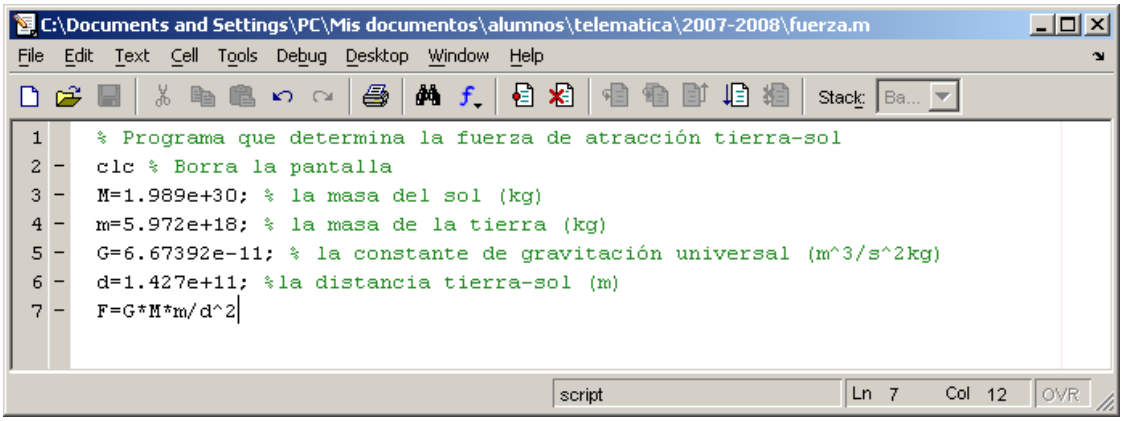
En MATLAB, un programa es un fichero con la extensión `.m` y en el cual hay escritos comandos que MATLAB pueda entender. Para crear un programa hay que usar un **editor** de texto. MATLAB trae uno incorporado que es especialmente útil, ya que resalta en diferentes colores comandos, variables, mensajes, ... Para activarlo se puede pinchar con el ratón en el icono de la barra de herramientas. Se abre el editor de textos de MATLAB.



El % sirve para insertar comentarios. Todo lo que escribamos detrás no será tenido en cuenta por MATLAB. Para hacer el ejercicio 2 escribimos:

```
% Programa que determina la fuerza de atracción tierra-sol
clc % Borra la pantalla
M=1.989e+30; % la masa del sol (kg)
m=5.972e+18; % la masa de la tierra (kg)
G=6.67392e-11; % la constante de gravitación universal (m^3/s^2kg)
d=1.427e+11; %la distancia tierra-sol (m)
F=G*M*m/d^2
```

Lo guardamos en nuestra carpeta, por ejemplo `z:\algebra`, con un nombre válido: `fuerza.m`. Para ello podemos darle al icono de salvar, seleccionar la carpeta adecuada y escribir el nombre sin extensión. En la barra superior del editor nos aparece el nombre completo del archivo: `z:\algebra\fuerza.m`



Para ejecutarlo tenemos dos opciones:

1. Volvemos a la ventana de trabajo y seleccionamos en la barra de arriba la carpeta donde está el fichero. Esto también puede hacerse en la línea de comandos mediante la orden

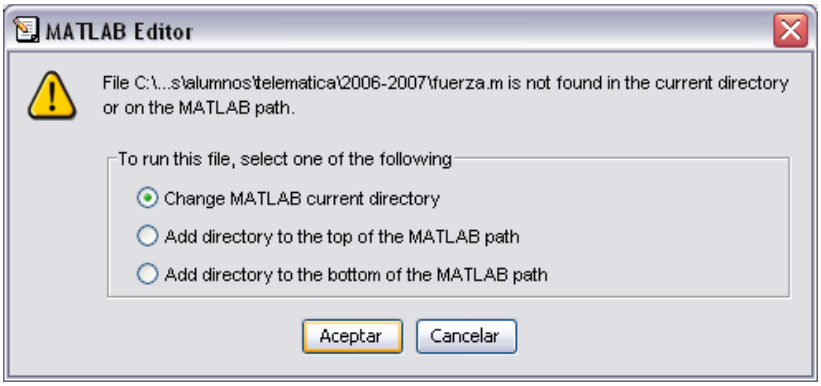
   ```
   >> cd z:\algebra
   ```

   Tecleamos el nombre del programa, sin extensión,

   ```
   » fuerza
   ```

   y MATLAB ejecuta todas órdenes de manera consecutiva (se salta los comentarios) y nos devuelve por pantalla sólo el valor de F.

2. Desde la ventana del editor, pinchar con el ratón sobre el icono  o dándole a la tecla F5. Si aún no hemos grabado el archivo, nos aparecerá el cuadro de diálogo de grabar. Si el archivo no se encuentra en la carpeta actual (*Current Directory*), nos aparecerá el siguiente cuadro de diálogo:



Aceptamos y el programa se ejecuta.

Uso de la memoria: Los valores de `M`, `m` `G` y `d` están en memoria aunque no aparezcan en pantalla. Aparecen en la ventana de las variables.

Ejercicio 4 *Se suelta un objeto desde un globo a 2000 metros de altura del suelo. ¿A qué altura estará al cabo de diez segundos? ¿Cuánto tiempo tarda en tocar el suelo? Nota:* $h = h_0 - \frac{1}{2}gt^2$, $g = 9{,}8m/s^2$. *Resuelve en un programa* `globo.m`.

Ejercicio 5 *¿Con qué fuerza se atraen dos partículas de signos opuestos y cargas* $q_1 = 1{,}602 \times 10^{-19}c$, $q_2 = -1{,}602 \times 10^{-19}c$, *si están a distancia* $d = 5 \times 10^{-11}m$? *Nota:* $F = \frac{1}{4\pi\varepsilon_0}|q_1||q_2|/d^2$ *y* $\varepsilon_0 = 8{,}854 \times 10^{-12}$. *Resuelve en un programa* `carga.m`.

2. Trabajando con matrices

El tipo básico de dato con el que MATLAB trabaja es la matriz. MATLAB significa MATrix LABoratory. Los escalares son considerados como matrices 1×1 y los vectores (que no son flechas en el plano o el espacio, sino ristras de números) son matrices fila o columna.

2.1. Escribiendo vectores y matrices

Los vectores se introducen escribiendo cada una de sus coordenadas entre corchetes, separadas por un espacio en blanco:

```
» v=[1 3 pi 1/3]
```

o bien separadas por comas:

```
» v=[1,3,pi,1/3]
```

Si separamos las componentes mediante punto y coma, obtendremos un vector columna:

```
» w=[1;3;pi;1/3]
```

Los elementos de una matriz se introducen, entre corchetes, por filas, separadas mediante un punto y coma (;) y con sus elementos separados por espacios en blanco o comas.

Ejemplo 2.1.1 *Para crear la matriz*

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

escribimos

```
>> A=[1 2 3; 3,1,2;1 1 0]
```

Ejercicio 6 *Crea en un programa* `vectores_y_matrices.m` *el vector* $cifras = (1,2,3,4,5,6,7,8,9,0)$*, el vector*

$$columna = \begin{pmatrix} 1 \\ \pi/3 \\ -\ln(2) \\ 0{,}03/(1-0{,}03) \\ \varepsilon_0 \end{pmatrix},$$

donde $\varepsilon_0 = 8{,}854 \times 10^{-12}$ *y las matrices*

$$B = \begin{pmatrix} 1 & 0 & 3 & -1 \\ -3{,}2 & 4 & 8 & \frac{1+\sqrt{5}}{2} \end{pmatrix},$$

$$C = \begin{pmatrix} 8 & -1 & 2 & 0 \\ 2 & 7 & 0 & -3 \\ -3 & 0 & 9 & 5 \\ 1 & 2 & 3 & 7 \end{pmatrix}.$$

2.2. Información sobre vectores y matrices.

Función	Salida
`size(A)`	Vector con las dimensiones de la matriz A
`size(A,1)`	Número de filas de la matriz A
`size(A,2)`	Número de columnas de la matriz A
`length(v)`	Número de coordenadas del vector v
`max(v)`	Máximo elemento del vector v
`[m, pos]=max(v)`	Guarda en m el valor `max(v)` y en pos la posición de m en v
`min(v)`	Mínimo elemento del vector v
`[m, pos]=min(v)`	Guarda en m el valor `min(v)` y en pos la posición de m en v

Ejemplo 2.2.1 *Sean A como en el ejemplo 2.3.1 y B como en el ejercicio 6. El número de filas de B es*

```
>> size(B,1)
ans =
     2
```

y el número de columnas de A es

```
>> size(A,2)
ans =
     4
```

Ejemplo 2.2.2 *Sea $v = (-6,0,4,2)$. Vamos a calcular su tamaño, su máximo, su mínimo y las posiciones que ocupan.*

```
>> v=[-6,0,4,2]
v =
    -6     0     4     2
>> length(v)
ans =
     4
>> [maximo,posicion_del_maximo]=max(v)
maximo =
     4
posicion_del_maximo =
     3
>> [minimo,posicion_del_minimo]=min(v)
minimo =
    -6
posicion_del_minimo =
     1
```

2.3. Operaciones elementales

Si A y B son matrices con las dimensiones adecuadas y λ es un escalar, las operaciones habituales se efectúan con las siguientes órdenes:

Operación	Resultado
A+B	Suma A y B
A-B	Resta B de A
A*B	Multiplica A por B
λ*A	Multiplica todos los elementos de A por λ
A^n	Eleva la matriz A al entero n
A'	Calcula la traspuesta de (la conjugada de) A

Ejemplo 2.3.1 *Sean B y C como en el ejercicio 6,*

$$A = \begin{pmatrix} -8 & 3 & 1 & 4 \\ 1 & 7 & 0 & 2 \\ 3 & 1 & 9 & 1 \\ 0 & 0 & 2 & 4 \end{pmatrix}, \; b = (1,2,3,4)^T.$$

Intentemos calcular $D1 = A + C$, $D2 = A + B$, $D3 = A * C$, $D4 = B * C$ $D5 = A * B$, $e1 = B * b$, $e2 = A * b$, $F1 = A^2$, $F2 = B^2$. En los casos en que no es posible, sale un mensaje de error indicando lo que ha pasado.

```
>> A=[-8,3,1,4;1,7,0,2;3,1,9,1;0,0,2,4]
A =
    -8     3     1     4
     1     7     0     2
     3     1     9     1
     0     0     2     4
>> b=[1,2,3,4]'
b =
     1
     2
     3
     4
>> D1=A+C
D1 =
     0     2     3     4
     3    14     0    -1
     0     1    18     6
     1     2     5    11
>> D2=A+B
??? Error using ==> plus Matrix dimensions must agree.

>> D3=A*C
D3 =
   -57    37     5    24
    24    52     8    -7
     0     6    90    49
    -2     8    30    38
>> D4=B*C
D4 =
```

```
   -2.0000   -3.0000   26.0000    8.0000
  -39.9820   34.4361   70.4541   39.3262
>> D5=A*B
??? Error using ==> mtimes Inner matrix dimensions must agree.

>> e1=B*b
e1 =
    6.0000
   35.2721
>> e2=A*b
e2 =
    17
    23
    36
    22
>> F1=A^2
F1 =
    70    -2     9    -9
    -1    52     5    26
     4    25    86    27
     6     2    26    18
>> F2=B^2
??? Error using ==> mpower Matrix must be square.
```

Ejercicio 7 *Sean*

$$H = \begin{pmatrix} 2 & 3 & 1 \\ 1 & 2 & 3 \\ 3 & 1 & 2 \end{pmatrix},\ J = \begin{pmatrix} 4 & 5 & 6 \\ 6 & 5 & 4 \end{pmatrix}\ y\ q = (7,8,9)^T.$$

Intenta calcular $J * H$, $(3 * H + H^3)J^T$, $J * q$ *y* $H * J$. *Haz los cálculos en el programa* `ejercicio7.m`.

2.4. Inversa de una matriz.

Las siguientes órdenes son útiles para calcular la inversa de una matriz y resolver sistemas lineales.

Operación	**Resultado**
`rank(A)`	Rango de la matriz A
`det(A)`	Determinante de la matriz A
`inv(A)`	Inversa de la matriz A
`A/B`	Si existe B^{-1}, calcula AB^{-1} de manera eficiente
`A\B`	Si existe A^{-1}, calcula $A^{-1}B$ de manera eficiente

Ejemplo 2.4.1 *Sean A como en el ejemplo 2.3.1 y B y C como en el ejercicio 6. Calculemos el rango, el determinante y la inversa de cada una de esas matrices. Observa los errores que aparecen cuando se intenta calcular el determinante o la inversa de una matriz no cuadrada.*

```
>> rank(A)
ans =
     4
>> rank(B)
ans =
     2
>> rank(C)
ans =
     4
>> det(A)
ans =
       -1994
>> det(B)
??? Error using ==> det
Matrix must be square.

>> det(C)
ans =
        3630
>> inv(A)
ans =
   -0.1214    0.0532   -0.0080    0.0968
    0.0231    0.1304    0.0181   -0.0928
    0.0401   -0.0341    0.1184   -0.0527
   -0.0201    0.0171   -0.0592    0.2763
>> inv(B)
??? Error using ==> inv
Matrix must be square.

>> inv(C)
ans =
    0.1074    0.0077   -0.0328    0.0267
   -0.0413    0.1201   -0.0105    0.0590
    0.0496    0.0292    0.1259   -0.0774
   -0.0248   -0.0479   -0.0463    0.1554
```

Ejemplo 2.4.2 *La siguiente matriz tiene determinante 0. Sin embargo el error de redondeo hace que salga 1. Observa que nos da una advertencia al calcular su inversa.*

```
M =
        2040      -19271      -15267        6513
       -8416      -80392      -62320       11920
     -246640    -2884194    -2239738      477982
        -368       -2616       -1985         335
>> det(M)
ans =
     1
>> inv(M)
```

```
Warning: Matrix is close to singular or badly scaled.
         Results may be inaccurate. RCOND = 1.642570e-019.
ans =
  1.0e+011 *
    2.2686    5.1853   -0.1620    2.5926
   -2.2153   -5.0635    0.1582   -2.5317
    2.1059    4.8134   -0.1504    2.4067
   -2.3289   -5.3232    0.1663   -2.6616
```

Ejemplo 2.4.3 *Sean A como en el ejemplo 2.3.1 y B y C como en el ejercicio 6. Calculemos $A^{-1}C$, CA^{-1}, $C^{-1}B^T$*

```
>> A\C
ans =
   -0.7437    0.6871   -0.0246    0.4779
    0.2984    0.7041   -0.0697   -0.9504
   -0.1550   -0.3842    0.9875    0.3255
    0.3275    0.6921    0.2563    1.5873
>> C/A
ans =
   -0.9137    0.2267    0.1545    0.7618
   -0.0211    0.9679    0.2879   -1.2849
    0.6249   -0.3811    0.7934    0.6174
   -0.0953    0.3310   -0.0311    1.6876
>> C\B'
ans =
   -0.0176   -0.5320
   -0.1317    0.6243
    0.5047    0.8400
   -0.3190   -0.2312
```

2.5. Resolución elemental de sistemas lineales.

Consideremos el sistema lineal de m ecuaciones con n incógnitas

$$\begin{cases} a_{11}x_1 & + & \dots & + & a_{1n}x_n & = & b_1 \\ \dots & & \dots & & \dots & & \dots \\ a_{m1}x_1 & + & \dots & + & a_{mn}x_n & = & b_m \end{cases}$$

Sea A su matriz de coeficientes, x el vector de incógnitas y b el vector del segundo miembro:

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}, \; x = \begin{pmatrix} x_1 \\ \cdots \\ x_n \end{pmatrix}, \; b = \begin{pmatrix} b_1 \\ \cdots \\ b_m \end{pmatrix}.$$

Si A es cuadrada e invertible, la solución del sistema es $x = A^{-1}b$. Esto en MATLAB se escribe

```
x=A\b
```

Este comando funciona aunque A sea rectangular o singular. Si el sistema es compatible determinado devuelve la única solución. Si es compatible indeterminado, devuelve una de las infinitas soluciones. Si es incompatible, devuelve la llamada *solución de mínimos cuadrados*. Podemos clasificar los sistemas utilizando el Teorema de Rouché-Frobenius y los rangos de la matriz A y la matriz ampliada $A|b$. La manera de escribir esta en MATLAB es `[A,b]`.

Ejemplo 2.5.1 *Resuelve el sistema*

$$\begin{cases} 2x + 3y + z = 7 \\ x + 2y + 3z = 8 \\ 3x + y + 2z = 9 \end{cases}$$

```
>> H=[2,3,1;1,2,3;3,1,2]
H =
     2     3     1
     1     2     3
     3     1     2
>> q=[7,8,9]'
q =
     7
     8
     9
>> solucion=H\q
solucion =
    1.6667
    0.6667
    1.6667
>> rank(H)
ans =
    3
>> rank([H,q])
ans =
    3
```

La *solución (con una precisión de 4 decimales) es* $x = 1{,}6667$, $y = 0{,}6667$, $z = 1{,}6667$.

Ejemplo 2.5.2 *Resuelve el sistema*

$$\begin{cases} 2x + 3y + z = 7 \\ x + 2y + 3z = 8 \end{cases}$$

```
>> H=[2,3,1;1,2,3]
H =
     2     3     1
     1     2     3
>> q=[7,8]'
q =
     7
```

```
     8
>> solucion=H\q
solucion =
         0
    1.8571
    1.4286
>> rank(H)
ans =
     2
>> rank([H,q])
ans =
    2
```

Una *solución es* $x = 0$, $y = 1{,}8571$, $z = 1{,}4286$.

Ejemplo 2.5.3 *Resuelve el sistema*

$$\begin{cases} 2x + 3y + z = 7 \\ x + 2y + 3z = 8 \\ 3x + y + 2z = 9 \\ -3x - 4y + z = 10 \end{cases}$$

```
>> H=[2,3,1;1,2,3;3,1,2;-3,-4,1]
H =
     2     3     1
     1     2     3
     3     1     2
    -3    -4     1
>> q=[7,8,9,10]'
q =
     7
     8
     9
    10
>> solucion=H\q
solucion =
    0.6296
   -1.2593
    4.1852
>> rank(H)
ans =
     3
>> rank([H,q])
ans =
     4
```

La solución de mínimos cuadrados *es* $x = 0{,}6296$, $y = -1{,}2593$, $z = 4{,}1852$. *Observa que realmente no es solución del sistema:* $H * solucion \neq q$.

```
>> H*solucion
ans =
    1.6667
   10.6667
    9.0000
    7.3333
```

Ejercicio 8 *Clasifica y resuelve los siguientes sistemas, en un programa* `sistemas.m`,

$$(1)\begin{cases} 5x + 5y - 2z - 2t = 6 \\ 5x - 10y + 7z + 7t = 9 \\ 5x + z + t = 7 \\ 5y - 3z - 3t = -1 \end{cases}$$

$$(2)\begin{cases} x + 2y + 3z = 6 \\ x + 3y + 8z = 19 \\ 2x + 3y + z = -1 \\ 5x + 6y + 4z = 5 \end{cases}$$

$$(3)\begin{cases} x + 2y + 3z = 6 \\ x + 3y + 8z = 19 \\ 2x + 3y + z = -1 \\ 5x + 6y + 4z = 0 \end{cases}$$

2.6. Construcción de vectores y matrices especiales.

Vectores con muchos elementos

Muchas veces nos aparecen vectores de muchos elementos y con alguna estructura especial. Disponemos de varios trucos para escribirlos rápidamente. Los dos esenciales son:

Orden	Salida
`v=a:h:b`, `w=a:b`	Vector $(a, a+h, a+2h, \ldots, a+nh)$, donde n es el mayor entero tal que $a+nh \in [a,b]$ si $h>0$ y $a+nh \in [b,a]$ si $h<0$. Si no se escribe h toma por defecto $h=1$. Puede resultar el vector vacío.
`linspace(a,b,n)`	Vector de n componentes cuyas coordenadas son los puntos de una partición uniforme del intervalo $[\text{mín}\{a,b\}, \text{máx}\{a,b\}]$

Ejemplo 2.6.1 *Construir* z *un vector con todos los números enteros del 1 al 10,* p *con todos los pares del 2 al 20,* ca *con los enteros del 10 al 1, contados marcha atrás,* x *con todas las centésimas entre -1 y 1,* v *un vector vacío,* t *un vector con 10 elementos entre 0 y* π *y* e *un vector con todos los enteros entre* 0 *y* pi

```
>> z=1:10
z =
     1     2     3     4     5     6     7     8     9    10
>> p=2:2:20
p =
     2     4     6     8    10    12    14    16    18    20
>> ca=10:-1:1
ca =
    10     9     8     7     6     5     4     3     2     1
>> x=-1:.01:1;
>> v=1:0
v =
   Empty matrix: 1-by-0
>> t=linspace(0,pi,10)
t =
  Columns 1 through 6
         0    0.3491    0.6981    1.0472    1.3963    1.7453
  Columns 7 through 10
    2.0944    2.4435    2.7925    3.1416
>> e=0:pi
e =
     0     1     2     3
```

Ejercicio 9 *Resuelve en el programa* `globo.m`*. Escribe un vector* t *con todos los segundos (enteros) desde 0 hasta* T*, el tiempo que tarda el objeto del ejercicio 4 en tocar el suelo.*

Ejercicio 10 *Escribe el vector* $ca2 = (10,8,6,4,2,0,-2)$*; calcula su tamaño. Almacena en memoria, pero no muestres por pantalla, un vector* r *de 736 elementos equiespaciados entre* -2π *y* $\pi/2$*. Haz los cálculos en el programa* `ejercicio10.m`*.*

Matrices especiales

Pueden definirse ciertas matrices con las siguientes órdenes:

Orden	Salida
`ones(n)`	Matriz cuadrada $n \times n$ de unos.
`ones(m,n)`	Matriz $m \times n$ de unos.
`zeros(n)`	Matriz cuadrada $n \times n$ de ceros.
`zeros(m,n)`	Matriz $m \times n$ de ceros.
`eye(n)`	Matriz identidad $n \times n$.
`eye(m,n)`	Matriz $m \times n$ con unos en la diagonal principal y el resto ceros.
`rand(n)`	Matriz cuadrada $n \times n$ de números aleatorios entre 0 y 1 y distribución uniforme.
`rand(m,n)`	Matriz $m \times n$ de números aleatorios entre 0 y 1 y distribución uniforme.
`randn(n)`	Matriz cuadrada $n \times n$ de números aleatorios, distribución N(0,1).
`randn(m,n)`	Matriz $m \times n$ de números aleatorios, distribución N(0,1).

Ejemplo 2.6.2 *Construyamos la* $I3$*, matriz identidad* 3×3*;* U*, una matriz* 4×2 *rellena de unos,* Z *una columna de 6 ceros y busquemos* r *un número aleatorio entre 0 y 1.*

```
>> I3=eye(3)
I3 =
     1     0     0
     0     1     0
     0     0     1
>> U=ones(4,2)
U =
     1     1
     1     1
     1     1
     1     1
>> Z=zeros(6,1)
Z =
     0
     0
     0
     0
     0
     0
>> r=rand(1)
r =
    0.3420
```

Ejercicio 11 *Construye I*34 *la matriz identidad* 3×4, *I*43 *la matriz identidad* 4×3, *u una fila de 20 unos y d una fila de 20 doses. Haz los cálculos en el programa* `ejercicio11.m`.

Ejercicio 12 *En un programa* `dado.m` *crea un dado, esto es, una orden que saque por pantalla aleatoriamente un entero entre 1 y 6. Usa para ello de manera adecuada la orden* `rand` *y una de las órdenes de redondeo* `round`, `ceil`, `floor`, `fix`. *Procura que el dado no salga cargado.*

Construcción de matrices por bloques

Ya hemos construido matrices por bloques al construir la matriz ampliada de un sistema. Dadas dos matrices A y B con el mismo número de filas, se puede definir una matriz C formada por todas las columnas de A y de B. Análogamente, se puede definir una matriz a partir de otras dos con el mismo número de columnas. Estas dos posibilidades pueden combinarse para formar matrices definidas por bloques.

Ejemplo 2.6.3 *Construyamos las matrices*

$$C=\begin{pmatrix}1&2&3&1&0\\4&5&6&0&1\\7&8&9&0&0\end{pmatrix},\ D=\begin{pmatrix}1&2&3\\4&5&6\\1&1&1\\1&1&1\\1&1&1\end{pmatrix},\ E=\begin{pmatrix}1&0&0&1&1&1\\0&1&0&1&1&1\\0&0&1&1&1&1\\1&2&3&4&5&6\\0&0&1&1&0&0\\0&0&1&0&1&0\end{pmatrix}$$

```
>> A=[1,2,3;4,5,6;7,8,9];
>> B=eye(3,2);
>> C=[A,B]
C =
     1     2     3     1     0
     4     5     6     0     1
     7     8     9     0     0
>> A=[1,2,3;4,5,6];
>> D=[A;ones(3)]
D =
     1     2     3
     4     5     6
     1     1     1
     1     1     1
     1     1     1
>> E=[eye(3) ones(3,3);1:6;zeros(2) ones(2,1) eye(2,3)]
```

Ejercicio 13 *Construye, usando bloques adecuados, la matriz*

$$A=\begin{pmatrix}2&1&0\\0&2&0\\0&0&3\end{pmatrix}.$$

Haz los cálculos en el programa `ejercicio13.m`.

2.7. Elementos dentro de una matriz. Submatrices.

Una vez definida una matriz o un vector, se puede acceder a sus elementos o submatrices con las órdenes:

Orden	Salida
v(i)	Coordenada i del vector v
A(i,j)	Elemento de la matriz A que ocupa la posición i,j
A(:,j)	Columna j de la matriz A
A(i,:)	Fila i de la matriz A
A(v,w)	Submatriz de A que contiene las filas indicadas en las coordenadas de v y las columnas indicadas en w
A(i,:)=[ ]	Elimina la fila i de la matriz A
A(:,j)=[ ]	Elimina la columna j de la matriz A

Ejemplo 2.7.1 *Sea* $v = (-6, 0, 4, 2)$. *El segundo elemento de* v *es*

```
>> v=[-6,0,4,2]
v =
    -6     0     4     2
>> v(2)
ans =
     0
```

El elemento de la quinta fila, tercera columna de la matriz E *del ejemplo 2.6.3 es*

```
>> E(5,3)
ans =
     1
```

Ejemplo 2.7.2 *Los elementos de las posiciones 2, 1, y 4 del vector* v *son*

```
>> v([2,1,4])
ans =
     0    -6     2
```

Los elementos de la tercera fila, columnas 2 y 5 de E *son*

```
>> E(3,[2,5])
ans =
     0     1
```

La submatriz de E *formada por los elementos de las fila 3 y 4, columnas 3, 4 y 5 es*

```
>> E([3,4],[3,4,5])
ans =
     1     1     1
     3     4     5
```

La segunda fila de E *es*

```
>> E(2,:)
ans =
     0     1     0     1     1     1
```

Ejemplo 2.7.3 *Son útiles los vectores de la forma* `a:b`*. Los elementos de la 3 fila de E, columnas de la 3 a la última, se pueden escribir como*

```
>> E(3,3:6)
ans =
     1     1     1     1
```

o también usando la palabra clave `end`

```
>> E(3,3:end)
ans =
     1     1     1     1
```

Los elementos de v que ocupan posición impar son

```
>> v(1:2:end)
ans =
    -6     4
```

Ejercicio 14 *Resuelve en un programa* `ejercicio14.m`*. Sea A la matriz ampliada del primer sistema del ejercicio 8. Encuentra*

1. *El elemento 2 de la cuarta fila.*
2. *La submatriz formada por los elementos de las filas tercera y primera, columnas 2 y 4.*
3. *La tercera fila.*
4. *Los elementos de la 2^a fila, columnas de la 2^a a la última.*
5. *Los elementos de la 3^a columna, filas de la 3^a a la última.*

Ejercicio 15 *Resuelve en un programa* `ejercicio15.m`*. La siguiente orden crea un vector donde se almacenan 100 tiradas de un dado (cargado para que el 1 y el 6 tengan la mitad de probabilidad de salir que cada uno de los otros números).*

```
jugada=round(rand(1,100)*5)+1
```

1. *¿Cuál fue la tirada número 50?*
2. *Encuentra las tiradas 12, 15 y 93.*
3. *Encuentra las tiradas de la 80 a la 95, ambas inclusiva.*
4. *Encuentra las tiradas de ocupan posición impar.*
5. *Encuentra las tiradas de la 91 a la última.*

Modificación de elementos y submatrices

Una vez localizado un elemento o submatriz, podemos modificarlo por otro número o matriz del tamaño adecuado. También podemos eliminar filas y columnas enteras.

Ejemplo 2.7.4 *Construye las matrices*

$$A=\begin{pmatrix}2&1&0\\0&2&0\\0&0&2\end{pmatrix},\ B=\begin{pmatrix}1&0&0&0&0&0\\0&1&0&0&0&0\\1&1&1&1&1&1\\0&0&0&1&0&0\\0&0&0&0&1&0\\0&0&0&0&0&1\end{pmatrix}$$

```
>> A=2*eye(3);
>> A(1,2)=1
A =
     2     1     0
     0     2     0
     0     0     2
>> B=eye(6);
>> B(3,:)=ones(6,1)
B =
     1     0     0     0     0     0
     0     1     0     0     0     0
     1     1     1     1     1     1
     0     0     0     1     0     0
     0     0     0     0     1     0
     0     0     0     0     0     1
```

Ejemplo 2.7.5 *Consideremos el sistema*

$$\begin{cases}2x+3y+z=7\\x+2y+3z=8\\3x+y+2z=9\end{cases}$$

y sea A la matriz ampliada del sistema. Vamos a aplicar el método de Gauss:

```
>> H=[2,3,1;1,2,3;3,1,2];
>> q=[7,8,9]';
>> A=[H,q]
A =
     2     3     1     7
     1     2     3     8
     3     1     2     9
>> A(2,:)=A(2,:)-(A(2,1)/A(1,1))*A(1,:)
A =
    2.0000    3.0000    1.0000    7.0000
         0    0.5000    2.5000    4.5000
```

```
    3.0000    1.0000    2.0000    9.0000
>> A(3,:)=A(3,:)-(A(3,1)/A(1,1))*A(1,:)
A =
    2.0000    3.0000    1.0000    7.0000
         0    0.5000    2.5000    4.5000
         0   -3.5000    0.5000   -1.5000
>> A(3,:)=A(3,:)-(A(3,2)/A(2,2))*A(2,:)
A =
    2.0000    3.0000    1.0000    7.0000
         0    0.5000    2.5000    4.5000
         0         0   18.0000   30.0000
```

Y hemos obtenido un sistema equivalente al original, cuya matriz es triangular superior.

Ejemplo 2.7.6 *Para construir la matriz*

$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

podemos hacer

```
>> A=eye(3);
>> A(:,2)=[]
A =
     1     0
     0     0
     0     1
```

Ejercicio 16 *Resuelve en línea de comandos. Intenta aplicar el método de Gauss a cada una de las matrices ampliadas de los sistemas del ejercicio 8. Ten cuidado con dividir entre 0.*

Para guardar este ejercicio utiliza la orden `diary`*. Con ella almacenamos en un fichero todo lo que pase en la ventana de comandos:*

```
>> diary ejercicio16
>> A=[5,5,-2,-2; ...
>> [... todos los comandos necesarios ...]
>> diary off
```

2.8. Operaciones elemento a elemento

Funciones de calculadora científica

Todas las funciones definidas para números, se pueden aplicar a un vector o a una matriz, y nos devolverá otro vector cuyos elementos son las imágenes del original por la función dada.

Ejemplo 2.8.1

```
>> x=[0,pi/2,pi,3*pi/2,2*pi]
x =
```

```
         0    1.5708    3.1416    4.7124    6.2832
>> y=sin(x)
y =
         0    1.0000    0.0000   -1.0000   -0.0000
>> z=round(x)
z =
     0     2     3     5     6
>> v=[-6,0,4,2]
v =
    -6     0     4     2
>> w=abs(v)
w =
     6     0     4     2
```

Operaciones elementales

MATLAB nos permite realizar operaciones que matemáticamente no están definidas o que tienen muy malas propiedades. Por ejemplo, sumar una matriz y un número o multiplicar dos matrices del mismo tamaño elemento a elemento. Salvo para la suma, **la regla general es añadir un punto "." delante del operador.** Se puede operar con dos matrices del mismo tamaño o con una matriz y un número.

Operación	Resultado
λ+A	Suma a cada elemento de A el escalar λ
A.*B	Calcula una matriz que en la posición (i,j) contiene el producto $a_{ij}b_{ij}$ de los elementos que en A y B ocupan dicha posición
n./A	Divide el número n entre cada elemento de A
A./B	Calcula una matriz que en la posición (i,j) contiene el cociente a_{ij}/b_{ij} de los elementos que en A y B ocupan dicha posición
A.\B	B./A
A.^n	Eleva cada elemento de la matriz A al número n
n.^A	Eleva el número n a cada elemento de A
A.^B	Calcula una matriz que en la posición (i,j) contiene $a_{ij}^{b_{ij}}$

Ejemplo 2.8.2 *Sea A la matriz del ejemplo 2.7.4. Observa la diferencia entre* A^2 *y* `A.^2`

```
>> A=2*eye(3);A(1,2)=1;
>> A^2
ans =
     4     4     0
     0     4     0
     0     0     4
>> A.^2
ans =
     4     1     0
     0     4     0
     0     0     4
```

Ejemplo 2.8.3 *Sea* $f(x) = e^{x^2} - (e^x)^2$*. Calcula las imágenes de todas las décimas en el intervalo* $[0,1]$*.*

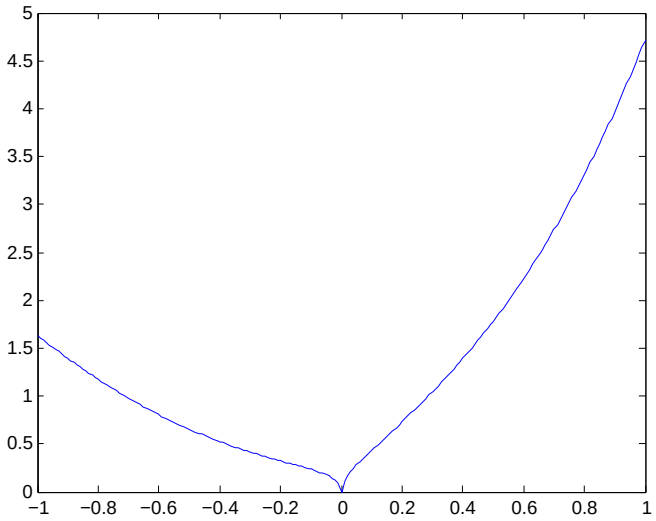
```
>> x=0:.1:1;
>> exp(x.^2)-exp(x).^2
ans =
  Columns 1 through 6
         0   -0.2114   -0.4510   -0.7279   -1.0520   -1.4343
  Columns 7 through 11
   -1.8868   -2.4229   -3.0566   -3.8017   -4.6708
```

Ejemplo 2.8.4 *Ya sabemos que con la estructura* `a:h:b` *podemos crear vectores cuyos elementos están en progresión aritmética. A partir de éstos, podemos crear otros tipos de progresiones:*

```
>> v=1:10
v =
     1     2     3     4     5     6     7     8     9    10
>> w=1./v
w =
  Columns 1 through 6
    1.0000    0.5000    0.3333    0.2500    0.2000    0.1667
  Columns 7 through 10
    0.1429    0.1250    0.1111    0.1000
>> x=v.^2
x =
     1     4     9    16    25    36    49    64    81   100
>> y=(1/2).^v
y =
  Columns 1 through 6
    0.5000    0.2500    0.1250    0.0625    0.0313    0.0156
  Columns 7 through 10
    0.0078    0.0039    0.0020    0.0010
```

Ejemplo 2.8.5 *Para pintar una función* $y = f(x)$ *se utiliza la orden* `plot(x,y)`*.* `x` *es un vector con las coordenadas* x *de los puntos a pintar, e* `y` *es un vector con las imágenes de cada uno de esos puntos.* MATLAB *pinta todos los puntos y los une mediante una poligonal. Si se ponen suficientes puntos, el efecto visual es una curva suave. Por ejemplo, para pintar* $f(x) = xe^x + x^2 + \sqrt{|x|}$ *en* $[-1,1]$*, podemos hacer*

```
>> x=-1:.01:1;
>> y=x.*exp(x)+x.^2+sqrt(abs(x));
>> plot(x,y)
```



Ejercicio 17 *Resuelve en el fichero* `ejercicio17.m`*:*

1. *Crea un vector* $d1$ *de 20 componentes de tal modo que* $d1_i = 2^i$.
2. *Crea un vector* $d2$ *de 10 componentes de tal modo que* $d2_i = 1/(i(i+1))$
3. *Escribe los 10 primeros términos de la sucesión* $(3n)^3$.

Ejercicio 18 *Construir un vector de 10 entradas que tenga un 1 en las posiciones pares y un -1 en las impares. Resuelve en el fichero* `ejercicio18.m`.

Ejercicio 19 *Dibuja las siguientes funciones en intervalos adecuados:* $y = \arg\tanh(x)$, $y = x^3 - x + 1$, $y = \sqrt{\frac{x+1}{x-1}}$. *Resuelve en los ficheros* `ejercicio19a.m`, `ejercicio19b.m`, `ejercicio19c.m`.

2.9. Otras funciones

MATLAB dispone también de las siguientes funciones para vectores:

Función	Salida
`dot(u,v)`	Producto escalar de los vectores u y v
`sum(v)`	Suma de las componentes del vector v
`prod(v)`	Producto de las componentes del vector v

Ejemplo 2.9.1 *Calcula* $20! = 1 \times 2 \times \cdots \times 20$

```
>> prod(1:20)
ans =
    2.432902008176640e+018
```

Ejemplo 2.9.2 *Aproximar con 100 términos la serie* $\sum_{n=1}^{\infty} \frac{1}{n^2}$ *de dos maneras distintas.*

```
>> n=1:100;
>> sum(1./n.^2)
ans =
   1.63498390018489
>> dot(1./n,1./n)
ans =
   1.63498390018489
```

Ejemplo 2.9.3 *Sea A la matriz de coeficientes del primer sistema del ejercicio 8. Calculemos la suma de los valores absolutos de los elementos de la segunda fila.*

```
>> A=[5,5,-2,-2;5,-10,7,7;5,0,1,1;0,5,-3,-3];
>> sum(abs(A(2,:)))
ans =
    29
```

Ejercicio 20 *Resuelve en un programa* `ejercicio20.m`*:. Utilizando la orden* `sum` *sobre vectores adecuados, calcular:*

$$\sum_{i=1}^{100} i,\ \sum_{i=1}^{100} i^2,\ \sum_{i=1}^{100} \frac{1}{i},\ \sum_{i=1}^{100} \frac{1}{2^i}.$$

Para hacerlo, es conveniente encontrar primero cada vector utilizando las operaciones elemento a elemento sobre el vector `1:100`*.*

Ejercicio 21 *Investiga que ocurre si aplicamos la orden* `sum` *a una matriz. Este uso será importante a la hora de calcular normas matriciales.*

3. Ficheros de función

La entrada y salida de datos en un programa no es ágil. Si queremos cambiar un dato en un programa, debemos modifícalo. Normalmente, un usuario no modifica un programa, sino que le puede enviar y pedir datos de alguna manera. En MATLAB hay dos maneras principalmente de realizar esta tarea:

1. Por pantalla, mediante las órdenes `input` y `disp`.
2. Mediante el uso de funciones.

Una **función** es un programa especial que pide unos datos y devuelve otros. El funcionamiento básico es como el de una función en matemáticas. La estructura de la cabecera del fichero es siempre es la misma:

```
function [salida]=nombre_funcion(entrada)
```

Ejemplo 3.0.4 *Creemos un fichero con la función* $f(x) = x^3 + 3x + 1$.

```
function y=f(x)
y=x.^3+3*x+1;
```

Lo guardamos con el nombre `f.m`.

No es obligatorio, pero es muy recomendable para evitar confusiones, que el nombre de la función coincida con el nombre del fichero.

Para calcular la imagen de un número por esa función, simplemente escribimos en la ventana de trabajo.

```
>> f(3)
ans =
    37
```

O, si queremos guardar el resultado en una variable,

```
>> z=f(3)
z =
    37
```

Importante:

1. A diferencia de los programas normales, los nombres de las variables dentro de la función son independientes de los nombres de las variables en la ventana de trabajo de MATLAB. `x` e `y` no aparecen en la ventana de las variables.
2. Dentro de la función, las órdenes suelen ir con punto y coma, para que no salga nada en pantalla. Ni siquiera el resultado final. Las funciones deben de ser como *cajas negras*. La salida se guardará en la variable `ans` si no ponemos nada o en la variable que asignemos.

Ejercicio 22 *Crea ficheros de función para las funciones del ejercicio 19. Llámalas respectivamente* `fun19a.m`, `fun19b.m`, `fun19c.m`.

Las variables de entrada y salida pueden por supuesto ser vectores o matrices.

Ejemplo 3.0.5

```
function jugada=lanza_cargado(n)
%
%       jugada=lanza_cargado(n)
%
%   Esta función simula n lanzamientos de un dado defectuoso
%   y los guarda en un vector
jugada=round(rand(1,n)*5)+1;
```

Para utilizarlo, desde la ventana de MATLAB, *escribimos*

```
>> j=lanza_cargado(9)
j =
     3     4     5     6     5     2     3     6     6
```

Los comentarios que hemos escrito aparecen al usar el comando de ayuda de MATLAB.

```
>> help lanza_cargado

        jugada=lanza_cargado(n)

    Esta función simula n lanzamientos de un dado defectuoso
    y los guarda en un vector
```

El programa que de lugar al cálculo de la salida puede ser complicado, pero en algún momento debemos asignar un valor a la variable de salida. Si se le asignan varios valores, tomará el último. Como regla general, no es conveniente modificar el valor de la variable de entrada.

Ejemplo 3.0.6 *Un mago le pide a un voluntario que piense un número, le sume 3, el resultado lo multiplique por 2, al resultado le reste 4, el resultado lo divida entre 2 y le diga el resultado final. Vamos a crear una función* `function m=voluntario(n)` *que haga los cálculos del voluntario y una función* `function n=mago(m)` *que adivine el número pensado por el voluntario, conocido el resultado final de las operaciones hechas.*

```
function m=voluntario(n)
m=n+3;
m=2*m;
m=m-4;
m=m/2;

function n=mago(m)
n=m*2;
n=n+4;
n=n/2;
n=n-3;
```

```
>> voluntario(8)
ans =
     9
>> mago(9)
ans =
     8
>> mago(voluntario(6))
ans =
     6
```

Ejercicio 23 *El mago se crece y le pide al voluntario que ahora eleve el número pensado al cubo, saque el logaritmo decimal, y encuentre el argumento cuya tangente hiperbólica (*`atanh`*) es el resultado. Crea una función* `function y=voluntario2(x)` *que ayude al voluntario a realizar los cálculos y otra* `function x=harrypotter(y)` *que ayude al mago a adivinar el número. Practica con tu compañero, a ver si eres capaz de adivinar los números que piensa.*

Ejercicio 24 *Crea una función* `x=sumapar(v)` *cuyo argumento de entrada sea un vector y cuya salida sea la suma de los valores absolutos de los elementos que ocupan posición par.*

Funciones de varias variables

Podemos pasar más de una variable de entrada o de salida. Las variables de entrada se escriben entre paréntesis y separadas por comas. Las de salida entre corchetes y separadas por comas también. No se pueden omitir las comas.

Ejemplo 3.0.7 *Crea una función* `n=norma_euclidea(x,y,z)` *donde se calcule la norma euclídea del vector de* (x, y, z).

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

Lo guardamos con el nombre `norma_euclidea.m` *y para ejecutarlo escribimos*

```
>> norma_euclidea(1,0,0)
ans =
     1
>> norma_euclidea(1,-1,2)
ans =
   2.44948974278318
```

Ejemplo 3.0.8

```
function [s,d,p]=sumaprod(x,y)
s=x+y;
p=x.*y;
d=x-y;
```

Lo guardamos con el nombre `sumaprod.m` *y para ejecutarlo escribimos*

```
>> [suma, diferencia, producto]=sumaprod(3,4)
suma =
     7
diferencia =
    -1
producto =
    12
```

Importante

1. De nuevo las variables de la función son **locales**. Es decir, sus nombres no afectan a los valores de las variables de MATLAB.

2. Para la entrada y salida de variables, MATLAB se fija en las **posiciones**. En el ejemplo anterior, el 3 está en la primera posición de entrada y entra en la variable `x`. El 4 está en la segunda posición y entra en la `y`. `suma` está en la primera posición de salida y toma el valor que sale de `s`. `diferencia` toma el valor que sale de `d` y `producto` el que sale de `p`.

Podemos omitir alguna o todas las variables de salida. En ese caso, MATLAB nos devolverá sólo las que pidamos y por orden:

Ejemplo 3.0.9

```
>> [suma, diferencia]=sumaprod(3,4)
suma =
     7
diferencia =
    -1
>> suma=sumaprod(3,4)
suma =
     7
>> sumaprod(3,4)
ans =
     7
```

Ejercicio 25 *Crea una función* `z=hemisferio_norte(x,y,R)` *donde* $z = \sqrt{R^2 - (x^2 + y^2)}$.

Ejercicio 26 *La función anterior tiene el siguiente problema: si* $x^2 + y^2 > R^2$ *entonces estamos sacando la raíz de un negativo y aparece un número imaginario. ¿Puedes arreglarlo para que salga 0? Utiliza la función* `max`. *Resuelve en* `hemisferio_norte2.m`.

Ejercicio 27 *Crea una función* `B=swap(A,i,j)` *cuyos datos de entrada sean una matriz* `A` *y dos números* i, j. *La matriz* B *tiene que ser como la* A *pero con las filas* i, j *intercambiadas.*

Ejercicio 28 *Crea una función* `[c,f]=sumcfabs(A)` *cuya entrada* A *sea una matriz* $n \times m$. c *será un vector fila de* m *componentes que tendrá en la posición* j *la suma de los valores absolutos de los elementos de la columna* j *de* A. f *será un vector columna de* n *componentes que tendrá en la posición* i *la suma de los valores absolutos de los elementos de la fila* i *de* A. *Utiliza lo aprendido en el ejercicio 21. Para calcular* f *utiliza la traspuesta de* A.

Entrada y salida de datos por pantalla

Programando ficheros `.m` es posible que queramos presentar algún mensaje o dato en la pantalla, e incluso que el propio programa nos pida parte de los datos por pantalla. Esto puede hacerse con las órdenes siguientes:

`x=input`	Permite introducir un dato desde el teclado La ejecución del programa continúa al pulsar ↩
`x=input('Cadena de caracteres')`	Realiza la misma operación que la instrucción anterior, pero mostrando en pantalla el mensaje *Cadena de caracteres*, a la espera de recibir datos
`disp('Cadena de caracteres')`	Muestra en pantalla la `Cadena de caracteres`. No espera datos.
`disp(x)`	Muestra en pantalla el valor de la variable x

4. Estructuras de control

En los programas de MATLAB se manejan principalmente tres estructuras de control:

- Decisión: `if ... elseif ... else ... end`
- Repetición un número fijo de veces: `for ... end`
- Repetición bajo condiciones: `while ... end`

Se puede utilizar estos comandos en la ventana de trabajo de MATLAB.

4.1. Toma de decisiones

Si queremos ejecutar un conjunto de instrucciones en el caso de que se cumpla una condición, usaremos una estructura `if`.

Manejo básico

La manera más sencilla de usarla es la siguiente:

```
if condición lógica
      ordenes de Matlab
end
```

Ejemplo 4.1.1 *Escribamos una función* `f=entero(n)` *que devuelva* $f = 1$ *si* n *es entero y* $f = 0$ *si no. Utilizaremos para ello 2 estructuras* `if ... end`.

```
function f=entero(n)
if n==round(n)
    f=1;
end
if n~=round(n)
    f=0;
end

>> entero(2)
ans =
     1
>> entero(1.5)
ans =
     0
```

Observa que para comprobar si dos números son iguales usamos el doble igual == y para comprobar si son distintos, usamos el símbolo ~=. Los operadores que se pueden usar para comparar números son:

Operador	Descripción
<	Menor
<=	Menor o igual
>	Mayor
>=	Mayor o igual
==	Igual
~=	Distinto

El símbolo ~ se obtiene manteniendo pulsado Alt Gr y dándole 2 veces a la tecla 4.

Es importante distinguir el símbolo = de asignación de un valor a una variable, del símbolo == que compara el valor de dos variables.

Ejercicio 29 *En el juego de la ruleta, se dice que "falta" si el número es menor o igual que 18 y que "pasa" si el número es mayor o igual que 19. Escribe una función* `f=ruleta(n)` *que devuelva* $f = 1$ *si* n *"pasa " y* $f = 0$ *si* n *"falta".*

Alternativas

Usando En los ejemplos anteriores se observa la necesidad de introducir una **alternativa** cuando no se cumpla la condición. Esto se hace mediante el comando `else`. La estructura habitual para este comando es:

```
if condición lógica
      ordenes de Matlab si la condición es verdadera
else
      ordenes de Matlab si la condición es falsa
end
```

Ejemplo 4.1.2 *Escribir una función* `f=par(n)` *que devuelva* $f = 1$ *si* n *es par y* $f = 0$ *si no.*

```
function f=par(n)
if n/2==round(n/2)
    f=1;
else
    f=0;
end

>> par(8)
ans =
     1
>> par(9)
ans =
     0
```

Otra manera de escribir esta función es

```
function f=par(n)
f=0;
```

```
if n/2==round(n/2)
    f=1;
end
```

Ejercicio 30 *Escribe una función* `ejercicio30.m` *para*

$$f(x) = \begin{cases} \operatorname{sen}\left(\frac{1}{x}\right) & si\ x \neq 0 \\ 0 & si\ x = 0. \end{cases}$$

Decisiones compuestas

Podemos relacionar varias expresiones lógicas entre sí usando, entre otros, los operadores `not`, `and`, `or`.

Operador	Forma corta	Descripción
`not(p)`	`~(p)`	Negación: toma el valor de verdad 1 cuando la expresión `p` toma el 0 y viceversa
`and(p,q)`	`(p) & (q)`	Y lógico: Sólo toma el valor 1 cuando `p` y `q` lo toman simultáneamente. 0 en los otros tres casos
`or(p,q)`	`(p) | (q)`	O lógico (inclusivo): Sólo toma el valor 0 cuando `p` y `q` lo toman simultáneamente. 1 en los otros tres casos

Nota: Al igual que en el lenguaje C, se pueden usar los símbolos `&&` y `||`. El comportamiento puede ser ligeramente distinto.

Ejemplo 4.1.3 *Escribamos una función* `f=natural(n)` *que nos devuelva* $f = 1$ *si* n *es natural y* $f = 0$ *si no. Un número es natural si es entero y positivo.*

```
function f=natural(n)
f=0;
if n==round(n) & n>0
    f=1;
end

>> natural(8)
ans =
     1
>> natural(-3)
ans =
     0
>> natural(3.5)
ans =
     0
```

Ejemplo 4.1.4 *Muchos algoritmos numéricos consisten en generar una sucesión* a_n *que converja hacia la solución del problema (llamémosla* s*, por ejemplo) que se quiere resolver. En general, llegar hasta el límite llevaría un tiempo infinito del que no disponemos, así que se para el algoritmo cuando* a_n *esté lo suficientemente cerca del límite. La distancia* $\|a_n - s\|$ *(el error absoluto* e_a *cometido) es normalmente imposible de medir, pero la mayoría de*

algoritmos proporcionan una manera de medirla aproximadamente. Por ejemplo, es muy frecuente hacer $e_a \approx \|a_n - a_{n-1}\|$. *Así, para parar un algoritmo fijamos una tolerancia* tol *y lo pararemos si*

$$|e_a| < tol.$$

Puede ocurrir que el algoritmo converja muy lentamente, o que simplemente no converja, con lo cual nunca se pararía el proceso. Hay que fijar un número máximo razonable de iteraciones. Llamémosle $maxiter$. *Así, una segunda condición que pararía el algoritmo es*

$$n > maxiter.$$

Construyamos una función `t=test_parada(ea,n)` *que pare un algoritmo con una tolerancia de* 10^{-6} *un número máximo de iteraciones de* 100.

```
function t=test_parada(ea,n)
if ea<1e-6 | n>100
   t=true;                  % true vale 1
else
   t=false;                 % false vale 0
end
```

```
 >> test_parada(1e-4,20)
ans =
     0
>> test_parada(1e-7,50)
ans =
     1
>> test_parada(1e-4,101)
ans =
     1
>> test_parada(1e-7,101)
ans =
     1
```

Ejercicio 31 *Escribe una función* `ejercicio31.m` *para*

$$f(x,y) = \begin{cases} \operatorname{sen}\left(\frac{1}{x^2+y^2}\right) & si\ (x,y) \neq (0,0) \\ 0 & si\ (x,y) = (0,0). \end{cases}$$

Ejercicio 32 *Escribe una función* `ejercicio32.m` *para*

$$g(x,y) = \begin{cases} \operatorname{sen}\left(\frac{1}{xy}\right) & si\ x \neq 0 \wedge y \neq 0 \\ 0 & si\ x = 0 \vee y = 0. \end{cases}$$

Ejercicio 33 *Cuando se está usando un algoritmo numérico para resolver una ecuación del tipo* $f(x) = 0$, *no es suficiente la condición de para* $ea < tol$. *Muchas veces se impone que, además de esa, se cumpla que el valor absoluto de* $f(x)$, *llamémosle* f_a, *sea menor que una tolerancia dada* $ftol$:

$$f_a < ftol$$

a) Escribe una función `t=test_parada2(ea,fa)` *que pare un algoritmo para resolución de ecuaciones con una tolerancia de* 10^{-6} *para el error en* x *y de* 10^{-8} *para el valor absoluto de la función.*

b) Escribe una función `t=test_parada3(ea,fa,n)` *que además establezca una salvaguarda para que el número de iteraciones no sea mayor que* 50.

Alternativas múltiples

Para tomar varias decisiones de maneras consecutivas, utilizaremos la orden `elseif`:

```
if condición lógica 1
      ordenes de Matlab si la condición 1 es verdadera
elseif condición lógica 2
      ordenes de Matlab si la condición 2 es verdadera y la condición 1 es
      falsa
else
      ordenes de Matlab si las condiciones 1 y 2 son falsas
end
```

Ejemplo 4.1.5 *Escribamos una función* `f=rouche_frobenius(A,b)` *que clasifique el sistema lineal* $Ax = b$. f *tomará el valor 0 si el sistema es compatible determinado, 1 si es compatible indeterminado y 2 si es incompatible.*

```
function f=rouche_frobenius(A,b)
if rank(A)==rank([A,b]) & rank(A)==size(A,2)
         f=0;
elseif rank(A)==rank([A,b]) & rank(A)<size(A,2)
         f=1;
else
         f=2;
end

>> A=eye(3);b=ones(3,1);
>> f=rouche_frobenius(A,b)
f =
     0

>> A=[eye(2);[1,1]];b=[0.7;-0.5;0.1];
>> f=rouche_frobenius(A,b)
f =
     2

>> A=[1,2,3;4,5,6];b=[8;9];
>> A(3,:)=-3*A(1,:)+4*A(2,:);b(3)=-3*b(1)+4*b(2);
>> f=rouche_frobenius(A,b)
f =
     1
```

Ejercicio 34 *Resuelve en* `ejercicio34.m`*. Clasifica los sistemas del ejercicio 8 utilizando la función* `f=rouche_frobenius(A,b)` *del ejemplo 4.1.5. Dentro de ejercicio34.m hay que escribir, para cada sistema, la matriz y el vector correspondiente y después llamar a la función, que deberá ir escrita en un fichero aparte. El programa debería sacar por pantalla solamente tres números (que serán ceros, unos y/o doses).*

4.2. Repetición un número fijo de veces

Si queremos repetir unas instrucciones un número fijo de veces, usamos una estructura `for ... end`. Para ello necesitamos un vector. Las ordenes se repetiran tantas veces como elementos tenga el vector. Además dispondremos de una `variable de iteración` que en cada paso irá tomando el valor correspondiente que se encuentre en el vector. El esquema general es el siguiente:

```
for k=v
     ordenes de Matlab
end
```

En este caso el vector es `v` y la variable de iteración es `k`.

Ejemplo 4.2.1 *Escribamos en el programa* `pruebafor.m`

```
for k=[1,2,3]
      k^2
end

>> pruebafor
ans =
     1
ans =
     4
ans =
     9
```

La variable de iteración `k` *va tomando consecutivamente los valores que encuentra en el vector y se van ejecutando las órdenes.*

Ejemplo 4.2.2 *Veamos como hacer un sumatorio con una orden* `for`*. Para calcular*

$$\sum_{k=1}^{100} \frac{1}{k^2}$$

escribimos en un programa `sumatorio.m`

```
resultado=0;
for k=1:100
    resultado=resultado+1/k^2;
end
resultado


>> sumatorio
resultado =
    1.6350
```

Recuerda que otra manera de hacerlo es:

```
>> sum(1./(1:100).^2)
ans =
    1.6350
```

Ejemplo 4.2.3 *La siguiente función calcula mediante un* `for` *el factorial de un número. Además comprueba que se ha introducido un entero no negativo.*

```
function f=factorial2(n)
% Función que calcula el factorial de un número
% (factorial es una función predefinida en Matlab)
if n~=round(n)|n<0
      error('Debe introducir un entero no negativo')
end
f=1;
for k=1:n
      f=f*k;
end

>> factorial2(3)
ans =
     6
>> factorial2(170)
ans =
  7.2574e+306
>> factorial2(171)
ans =
   Inf
```

Otra manera de hacerlo es con la orden `prod` *sobre un vector adecuado:*

```
>> prod(1:3)
ans =
     6
>> prod(1:170)
ans =
  7.2574e+306
>> prod(1:171)
ans =
   Inf
```

Ejercicio 35 *Calcula, usando la estructura* `for ... end` *los sumatorios del ejercicio 20. Resuelve en* `ejercicio35.m`.

Ejercicio 36 *Repite el ejercicio 28 sin utilizar la orden* `sum`. *Utiliza dos bucles* `for ... end` *anidados para calcular cada variable de salida. Resuelve en* `ejercicio36.m`.

4.3. Repetición condicional

En cálculo numérico muchas veces hay que repetir un determinado cálculo hasta que se alcance una precisión deseada. En principio no sabemos cuantas veces hay que hacer el cálculo, así que una estructura `for ... end` no es adecuada. Para repetir unas órdenes de manera condicional usamos la estructura `while ... end`. Su uso básico es el siguiente:

```
while condición
      ordenes de Matlab
end
```

Las órdenes se repetirán mientras la condición sea verdadera.

Ejemplo 4.3.1 *Vamos a calcular e usando la fórmula*

$$e = \sum_{k=0}^{\infty} \frac{1}{k!}$$

con una precisión de 10^{-6}. *Creamos un programa* `calculo_e.m`

```
e_anterior=0;
e=1;
k=1;
while abs(e-e_anterior)>1e-6
    e_anterior=e;
    e=e+1/prod(1:k);
    k=k+1;
end
e
iteraciones=k-1


>> calculo_e
e =
   2.71828180114638
iteraciones =
    10
```

Ejercicio 37 *Crea un programa* `calculo_pi.m` *que intente calcular* π *con 6 decimales usando la fórmula*

$$\frac{\pi}{4} = \sum_{k=0}^{\infty} \frac{(-1)^k}{2k+1}.$$

La condición de parada deberá incluir una salvaguarda para no realizar más de 1000 *iteraciones.*

Esta serie converge muy lentamente y en 1000 iteraciones sólo se consiguen 3 decimales de precisión. Una buena referencia para encontrar series que convergen rápidamente a π *es* `http://www.mcs.surrey.ac.uk/Personal/R.Knott/Fibonacci/fibpi.html`