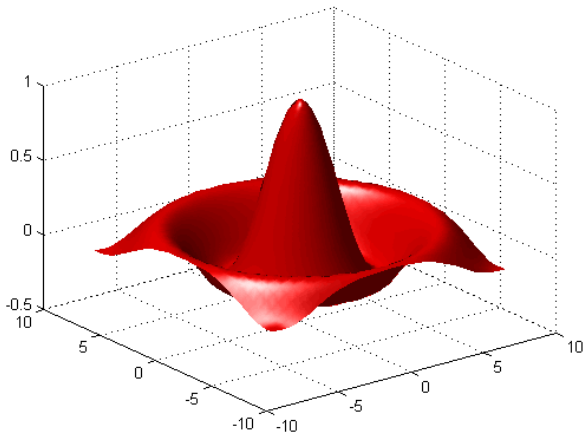
UNIDAD 2
Matrices, bucles, funciones, condicionales, superficies 3D.

- **Control de *flujo***: **bucles** `for`.
- Matrices.
- Representación gráfica de superficies 3D.
- **Funciones de usuario**.
- **Variables lógicas** y operadores de comparación.
- **Control de flujo**: **condicionales** (`if elseif else`).
- **Control de flujo**: **bucles** `while`.
- Control de flujo: `break` y `continue`.

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$



Control de flujo

Hasta el momento, los comandos del script se ejecutan secuencialmente desde la primera hasta la última línea.

Las instrucciones de control de flujo permiten alterar el orden de esta secuencia de ejecución, de dos maneras diferentes:

Bucles: repetición de la ejecución de un grupo de comandos. Esto se hace mediante las instrucciones `for` y `while`.

Ejecución condicional: la ejecución de uno o varios grupos de comandos está sujeta a ciertas condiciones. Esto se hace mediante las instrucciones `if`, `elseif` y `else`.

Control de flujo. Bucle `for`. (`help for`)

Los bucles `for` son útiles para realizar operaciones repetitivas. Esto es, se puede repetir un fragmento de código (una o más líneas o comandos) un número fijo de veces. Lo habitual es generar un índice (número entero) que va cambiando de valor para recorrer elementos de un vector según se va repitiendo la ejecución del bucle.

Sintaxis:

```
% Observar que n es una variable tipo vector, que debe nombrarse.
for n=valor_inicial:incremento:valor_final
% El bucle se ejecuta tantas veces como número de valores contenga n
% En cada ejecución n adquiere secuencialmente los valores especificados
% en valor_inicial:incremento:valor_final
        comando 1 ;
        comando 2 ;
        etc.
end  % la lista de comandos dentro del bucle se termina con end
```

Ejemplo: (bucle_for_01.m)

```
% Calcula la suma y producto de todos los elementos de un vector
x=linspace(1,10,300) ;
n_elem=length(x) ;
suma=0.0 ;
producto=1.0 ;
for n=1:1:n_elem
        suma=suma+x(n) ;
        producto=producto*x(n) ;
end
% equivalente a suma=sum(x) ; producto=prod(x) ;
```

Ejemplo: bucle `for` y obtención del tiempo de ejecución con funciones `tic` y `toc`.

```
% bucle_for_velocidad_01.m
% Calcula el módulo de un vector con la función interna norm y con un bucle for
% Compara los tiempos de ejecución

a=linspace(1,100,1E5) ;
n_elem=length(a) ;

% Cálculo con función interna norm
tic ;
modulo_norm=norm(a) ;
toc  % muestra el tiempo transcurrido desde el último tic

% Cálculo con bucle for
tic ;
modulo_for=0 ;
for n=1:1:n_elem
        modulo_for=modulo_for+a(n)^2 ;
end
modulo_for=sqrt(modulo_for) ;
toc % muestra el tiempo transcurrido desde el último tic

modulo_norm , modulo_for  % mostrar resultados
```

Ejemplo:

```
% Multiplica todos los elementos de un vector por dos y por tres
% Compara los tiempos de ejecución

a=linspace(1,100,1E5) ;

% Cálculo con operaciones elemento a elemento
tic ;
a2=2.*a ;
a3=3.*a ;
toc  % muestra el tiempo transcurrido desde el último tic

% Cálculo con bucle for
tic ;
for n=1:1:length(a)
        a2(n)=2*a(n) ;
        a3(n)=3*a(n) ;
end
toc % muestra el tiempo transcurrido desde el último tic
```

Ejemplo: generar la serie de Fibonnaci f(i)=f(i-2)+f(i-1), donde f(1)=0 y f(2)=1.
Comprobar que el cociente f(i)/f(i-1) converge hacia la proporción áurea (sqrt(5)+1)/2.

```
f(1)=0 ; f(2)=1 ; N=20 ;
for i=3:N
    f(i)=f(i-1)+f(i-2) ;
end
plot([1:N-1],f(2:end)./f(1:end-1),'o-') ;
disp(['Golden ratio = ',num2str((sqrt(5)+1)/2)]) ;
```

Matrices

MATLAB está especialmente diseñado para operar muy eficazmente con matrices; MATLAB=MATrix LABoratory.

Para MATLAB una matriz es un conjunto de números ordenados en forma de filas y columnas. Siempre tiene *forma rectangular*, esto es, todas las filas tienen el mismo número de columnas.

En realidad para MATLAB todas las variables son matrices: un valor numérico es una matriz de una fila y una columna (1x1), un vector fila es una matriz de una fila y el número de columnas es el número de elementos del vector fila.

La asignación de valores a los elementos de una matriz se hace mediante la ya conocida sintaxis para crear vectores: `[ ] , ;`

```
>> M=[1 2 3 4 ; 4 5 exp(6) 7 ; 8 pi 9^2 9] % 3 filas, 4 columnas
```

Espacios o comas separan columnas, punto y coma separa filas.

En nombre de las variables tipo matriz sigue las mismas reglas que el de los vectores. *Lo más habitual es que el nombre de las variables tipo matriz, aunque no es obligatorio, empiece por mayúscula. Por otra parte, el nombre de las variables tipo vector suele empezar por minúscula, aunque no es obligatorio.*

Matrices. Acceso a elementos, filas y columnas (`help colon`)

El acceso a elementos individuales por índice es mediante `()`, como en los vectores, aunque para matrices se emplean dos índices separados por coma.

```
Matriz(n_fila,n_columna)
>> A(2,3) % acceso al elemento fila 2 y columna 3 de la matriz M
```

Acceso a filas, serán vectores fila

```
>> A(2,1:end)     % toda la fila 2
>> A(2,:)         % ATENCIÓN a : que es equivalente a 1:end
>> A(2,2:end-1)   % parte de la fila 2
```

Acceso a columnas, serán vectores columna

```
>> A(1:end,3) ;   % toda la columna 3
>> A(:,3) ;       % ATENCIÓN a : que es equivalente a 1:end
>> A(2:3,3) ;     % parte de la columna 3
```

Acceso a sub-matrices, serán matrices

```
>> A(1:2,3:4) ;
>> A([1,2],[3,4]) ;
```

Matrices. Operadores aritméticos *elemento a elemento*. `help .*`

Suma, resta, multiplicación, división, exponenciación elemento a elemento: `+ - .* ./ .^` (ATENCIÓN al punto que precede). Las dos matrices deben tener mismo número de filas y columnas, esto es, iguales dimensiones.
Mensaje de error `>> Matrix dimensions must agree`.

Estas operaciones se hacen elemento a elemento. ATENCIÓN: los operadores `* / ^` (sin punto) entre matrices tienen un significado completamente diferente (se verá más adelante).

Matriz transpuesta
Se emplea para intercambiar filas con columnas, y vector fila en vector columna o viceversa. NOTA: válido para transformar vectores fila en vectores columna y viceversa.

```
>> A=transpose(M)   ,   B=M'
% el apóstrofe ' después de matriz es lo mismo que la función transpose
```

Matrices. Multiplicación de matrices (* ^) `help *`

El operador `*` (sin punto) multiplica dos matrices (`C=A*B`). El número de columnas de `A` debe ser igual al número de filas de `B`. La matriz `C` tendrá las mismas filas que `A` y las mismas columnas que `B`.
Mensaje de error: `>> Inner matrix dimensions must agree.`

```
>> M1=[1 2 3 ; 4 5 6] ; M2=[3 2 ; 4 5 ; 6 7] ; M1*M2
```

El operador de exponenciación (^) (sin punto) permite multiplicar `n` veces la matriz `A` por sí misma (`A^n`). La matriz debe ser cuadrada. ATENCIÓN a los exponentes no enteros o negativos, que no vamos a considerar en este curso.

Matrices predefinidas

```
>> zeros(n,m) ;   % matriz n*m con todos los elementos igual a cero
>> ones(n,m) ;    % matriz n*m con todos los elementos igual a uno
>> rand(n,m) ;    % matriz n*m con elementos aleatorios (entre cero y uno)
>> eye(n) ;       % matriz n*n 'identidad': unos en la diagonal, cero el resto
>> diag(v) ;      % matriz cuadrada con los elementos del vector v en la diagonal
```

Matriz inversa. Función `inv`

La matriz inversa `inv(A)` de una matriz cuadrada `A` cumple: $A^{-1}*A=A*A^{-1}=$`eye`

Ejercicio: comprobar esto último con una matriz aleatoria de dimensión 3x3

Determinante de una matriz. Función `det(A)`
Ejercicio: comprobar `det(A)=det(A') y det(A*B)=det(A)*det(B)`

Dimensiones de la matriz. Función `size`
`size(A)` da como resultado un vector fila cuyos elementos son el número de filas y el número de columnas de la matriz `A`. NOTA: no es lo mismo que la función `length` vista anteriormente.

```
>> A=[1 2 3;4 5 6;7 8 9] ; size(A) , length(A)
>> B=[3 5 6 7] ; size(B) , length(B)
```

Bucles *for* anidados.

Se trata de insertar un bucle *for* dentro de otro bucle *for*. Generalmente se emplea para acceder secuencialmente, por índice de fila y columna, a los elementos de una matriz.

Ejemplo: En este caso, para una matriz A, un bucle recorre las filas y el otro recorre todas las columnas de una fila.

```
A=zeros(5,4) ; % generar matriz con valores cero

[nfilas,ncolumnas]=size(A) ; % obtiene dimensiones de A

for i=1:nfilas % este bucle recorre las filas de A
       for j=1:ncolumnas % este bucle recorre las columnas de
                         % la fila número i de A

              % asigna el valor del elemento A(i,j)
              A(i,j)=2*i+4*j ;
       end % fin del bucle j (columnas de cada fila i)
end % fin del bucle i (filas)
```

Ejercicio: obtener la traspuesta de una matriz empleando dos bucles `for` anidados.

Representación gráfica de superficies

Una superficie puede estar definida por una función escalar de dos variables **`z=f(x,y)`**. Para que MATLAB pueda hacer una representación gráfica hace falta suministrar las matrices `XM` e `YM` que definen el rango (coordenadas x e y) en el que se dibuja la superficie y una matriz `Z` con los valores de `f(x,y)` en todos los pares de coordenadas de los elementos de `XM` e `YM`.

```
% Generar los vectores x e y
x=linspace(-3,3,100) ; y=linspace(-5,4,100) ;
% no es necesario, pero reservar espacio se ejecuta más rápido
Z=zeros(100,100) ;

% Generar la superficie (matriz) Z=f(x,y) mediante dos bucles for anidados
for i=1:100
    for j=1:100
        Z(i,j)=y(i)^2+x(j)^2 ;
    end
end
[XM,YM]=meshgrid(x,y) ;   % genera todos los pares de coordenadas x,y
surf(XM,YM,Z) ;           % genera el gráfico
shading interp ;          % opción de dibujo
```

Ejercicio: probar otras funciones f(x,y), y otras representaciones como `contour, surf, surfc, surfl, mesh` así como las opciones `colorbar, shading.` También `zlabel, title, zlim` etc.

Nota sobre el ejercicio anterior. Vectorización de bucles `for`

Por motivos internos de MATLAB los bucles `for` se ejecutan, en general, mucho más lentamente que las operaciones 'vectoriales' o 'vectorizadas'.

Los bucles anidados del ejemplo anterior pueden sustituirse por:

```
[XM,YM]=meshgrid(x,y) ; % genera todos los pares de coordenadas x,y
Z=XM.^2+YM.^2 ;         % operación 'vectorizada'
surf(XM,YM,Z) ;         % genera el gráfico
```

Ejemplo: potencial eléctrico (V) producido por una carga puntual (Q) a distancia (r)

V=K*Q/r % K=9E9 J*m^2*C^{-2} Cte. de Coulomb

Hacer un script que muestre un gráfico de V(x) para Q=1E-19 C, x=[-1E-8 hasta +1E-8] m

```
K_Coulomb=9E9 ; % J*m^2*C^{-2}
Q=1.6E-19 ; % C
x=abs(linspace(-1E8,1E8,100)) ; % metros
V= K_Coulomb*Q./abs(x) ; % Voltios
plot(x,V,'b*-') ;
% etiquetar los ejes
```

Ejercicio: Calcular la componente x del campo eléctrico (derivando numéricamente V) y representar gráficamente. Etiquetar los ejes.

Ejemplo 2D: potencial eléctrico (V) producido por una carga puntual (Q) a distancia (r)

```
%% Generar los vectores x e y
x=linspace(-1E-8,1E-8,20) ; y=linspace(-1E-8,1E-8,20) ; % metros
Z=zeros(100,100) ;

[XM,YM]=meshgrid(x,y) ;     % genera todos los pares de coordenadas x,y

% Generar la superficie (matriz) Z=f(x,y)
K_coulomb=9E9 ; Q=1.6E-19 ; % Coulombios
V=K_coulomb*Q./sqrt(XM.^2+YM.^2) ; % vectorizado, en lugar de for anidados
surf(XM,YM,V) ;           % genera el gráfico
shading interp ;          % opción de dibujo
% etiquetar ejes
```

Ejercicio: repetir lo anterior con una carga que no esté en el origen y también con **dos** cargas situadas en puntos diferentes.

Funciones de usuario (`help function`)

Además de las funciones predefinidas de MATLAB (`sin sum` etc.) se pueden definir funciones de usuario. Son *scripts* (ficheros *.m*) ligeramente modificados para que acepten argumentos de la función y devuelvan resultados de la función. La primera línea debe empezar por `function` y la última línea debe ser `end`

```
function [r1, r2, ...]=nombre_funcion(arg1, arg2, arg3,...)
        % secuencia de comandos...
end
```

arg1, arg2, arg3,... son los argumentos (puede haber cualquier número o ninguno) r1,r2,... son los resultados (puede haber cualquier número o ninguno). Pueden ser escalares, vectores o matrices. No todos los resultados han de ser del mismo tipo.

El código de la función debe guardarse en un archivo .m, cuyo nombre debe ser exactamente el mismo que el de la función.

```
function [r1]=cos_punto_cuadrado(angulo) % ATENCIÓN: debe ser el nombre del fichero .m
% Comentario para comando help
% Más comentario
          r1=cos(angulo).^2 ;    % ATENCIÓN: ángulo podría ser una matriz!
end
```

NOTA: las funciones definidas así pueden utilizarse desde MATLAB o un *script*. Deben estar en el directorio actual. Ejemplo: `a=cos_punto_cuadrado(pi/7) ;`
NOTA: el comentario inicial aparece con el comando `help cos_punto_cuadrado`
NOTA: si hay más de un resultado y/o argumento, ejecutar con `[a,b]=my_func(c,d)`

Funciones de usuario. Visibilidad de las variables.

Dentro de la función se pueden utilizar y modificar los valores de los argumentos, pero esto no repercutirá en el código que llama a la función, aunque el nombre coincida. Esto es, los argumentos son variables locales.

La función tiene su propio *workspace*, distinto al del código que llama a la función, de modo que TODA la comunicación con el *exterior* ha de ser con argumentos y resultados. En el código de la función pueden definirse variables (que serán locales), pero no serán visibles desde el código que llama a la función.

Dentro de la función se pueden utilizar y modificar los valores de los resultados, que tras terminar la función repercutirá en los valores del código que llama a la función.

Dentro de la función:
- Pueden no utilizarse todos los argumentos, pero sería extraño.
- Pueden no establecerse los valores de todos los resultados, pero sería extraño.

El código que llama a la función no tiene que aceptar todos los resultados:

```
function [a,b]=my_fun(c)
        a=2*c ;
        b=9*c ;
end ;

>> z=my_fun(3) ; % el resultado es z=6
>> [z,k]=my_fun(3) ; % el resultado es z=6 y k=27
```

Ejercicio: escribir una función charge_potential.m que obtenga el potencial eléctrico de una carga *Q* en el punto (*xQ,yQ*), en la posición (*x,y*) que pueden ser matrices, como las del resultado de la función `meshgrid`.

```
function [U_pot]=charge_potential(Q,xQ,yQ,x,y)
```

Ejercicio: repetir un ejercicio anterior del potencial producido por dos cargas en posiciones diferentes, pero ahora con **cinco** cargas no necesariamente del mismo signo ni magnitud.

Ejercicio: escribir una función charge_field.m que obtenga el vector campo eléctrico de una carga *Q* en el punto rQ=(*xQ,yQ,zQ*), en la posición r=(*x,y,z*).

```
function [E_field]=charge_field(Q,rQ,r)
```

Ejercicio: repetir el aparado anterior, pero obteniendo a la vez el vector campo eléctrico y el potencial electrostático:

```
function [E_field,U_pot]=charge_field(Q,rQ,r)
```

COMPROBAR que funcionan correctamente, empleando estas funciones para representar gráficamente el potencial electrostático de una o más cargas en una región del espacio XY, tal como se hizo en un ejercicio anterior.

Funciones de usuario

Ejercicio:
Escribir las funciones de usuario correspondientes a la derivada numérica y la integral *cumulativa* o *incremental* estudiadas en el capítulo 1.

```
function [dydx] = mi_derivada(x,y)

function [integ] = mi_integral(x,f)
% RECORDAR SUMAR EL VALOR INICIAL !!!!
```

Funciones, descartar resultados

En ciertas ocasiones, típicamente cuando una función devuelve varios resultados, y no estamos interesados en todos ellos, queremos ignorar o descartar alguno de los resultados. Para ello se utiliza el símbolo **~**.

Por ejemplo con la función `size`:

```
% Sea la matriz A
A=ones(5,2) ;
% queremos obtener el número de filas y de columnas
[nfilas,ncolumnas]=size(A) ;
% sólo estamos interesados en el número de columnas
[~,solo_n_columnas]=size(A) ;
disp(['columnas de A = ',num2str(solo_n_columnas)]) ;
```

Variables lógicas y operaciones de comparación y lógicas (`help relop`)

Una variable lógica elemental expresa si algo es verdadero o falso. Su valor se expresa mediante los números uno para verdadero y cero para falso.

Pueden ser el resultado de un operador de comparación numérica, por ejemplo *mayor que* >, o *menor que* <, cuyo resultado será verdadero o falso.

```
>> a=3.5 ; b=2.5 ; cc=a>b , dd=a<b
```

Lista de operadores lógicos de comparación entre dos valores numéricos: `==` (*igual a*), `>`, `>=` (*mayor o igual*), `<`, `<=`, `~=` (*diferente a)* (`~` tecla Alt-126)

Existen dos variables predefinidas de MATLAB: `true` (verdadero) y `false` (falso).

```
>> a=false
```

Operaciones lógicas entre variables lógicas:

~	NOT	negación de una sola variable lógica (lo contrario)	`~a`
&	AND	sólo true si la dos variables son true	`a&b`
|	OR	verdadero si al menos una de las variables es true	`a|b`

NOTA: el resultado es también un valor lógico
VÉASE https://es.wikipedia.org/wiki/Lógica_binaria

Existen los vectores y matrices lógicas, en los que cada uno de los elementos es un valor lógico. Las operaciones de comparación entre vectores son elemento a elemento (sin necesidad del *punto*). Por tanto los vectores o matrices deben tener las mismas dimensiones.

```
>> a=[3 4 5] ; b=[1 8 0] ; a>b
ans=[1 0 1]
```

es decir: `[true false true]`

Las operaciones entre vectores lógicos también son elemento a elemento (sin necesidad del *punto*). Por tanto los vectores o matrices deben tener las mismas dimensiones.

```
>> a=[true true false] ; b=[true false false] ;
>> c=a & b % debería ser [true false false]
```

Hay reglas de precedencia de operadores lógicos; utilizar paréntesis en caso de duda

```
>> d=(a | (b & c))
>> g=b & c ; d=a | g  % obtiene el mismo resultado
```

Control de flujo. Bucle `while`.

Permite ejecutar una secuencia de comandos repetidamente mientras se cumpla que un valor lógico sea `true`.

```
valor_logico=true ;
n=1 ;                       % índice opcional
while valor_lógico          % también podría ser una operación lógica
        comando1 ;
        comando2 ;
        etc.
        n=n+1 ;             % opcional
        valor_logico=??? % opcional decisión para continuar el bucle
end  % la lista de comandos dentro del bucle se termina con end
```

Diferencias bucle `while` con el bucle `for`:

- Se utiliza cuando se desconoce el número de veces que hay que ejecutar el bucle, que depende del resultado de alguno de los comandos internos del bucle.

- Podría ejecutarse indefinidamente, si el valor lógico es siempre verdadero.
- Podría no ejecutarse nunca, si al comienzo el valor lógico es falso.
- `help while`

Ejemplo: Considerar el cálculo del número de Euler (*e*) con la expresión:

$$F(n) = \lim_{n\to\infty} \left(1 + \frac{1}{n}\right)^n$$

Determinar el valor de *n* de manera que la diferencia entre *F(n)* y el número *e* sea menor que 1E-4.

```
euler=exp(1) ; % número de Euler
n=1 ; % valor inicial de n
F=(1+1/n)^n ; % primera aproximación

while abs(F-euler)>1E-4 % comprobar convergencia
   n=n+1 ; F=(1+1/n)^n ; % siguiente aproximación
end

disp(['n=',num2str(n)]) ;
```

Control de flujo. Ejecución condicional `if else`

Permite ejecutar condicionalmente una secuencia de comandos, dependiendo del resultado de una operación lógica.

```
if valor_logico  % los siguientes comandos se ejecutan si valor_logico==true
        comando1 ;
        etc.
else             % los siguientes comandos se ejecutan si valor_logico==false
        comando35 ;
        etc.
end
```

NOTA: sólo se ejecuta uno de los bloques (`if` o `else`)
NOTA: no es necesario el bloque else `>> if 4>3 , a=2*pi , end`

Ejemplo

```
% Evitar una división por cero
a=pi ;
if a~=0          % el valor_logico se obtiene de una comparación numérica
        b=1/a    % se ejecuta si a es distinto de cero
else
        b=0.0     % se ejecuta si a no es distinto de cero
end
```

NOTA: si no hay que ejecutar los comandos `else`, basta con terminar los comandos del `if` con `end`.

NOTA: `help if`

Comando `elseif`

```
if valor_logico_1  % los siguientes comandos se ejecutan si valor_logico_1==true
          comando1 ;
          etc.
elseif valor_logico_2 % se ejecuta si no se ha ejecutado ningún if o elseif anterior y
                      % valor_logico_2==true
          comando15 ;
          etc.
elseif valor_logico_3 % se ejecuta si no se ha ejecutado ningún if o elseif anterior y
                      % valor_logico_3==true
          comando21 ;
          etc.
else      % los siguientes comandos se ejecutan si no se ha ejecutado ningún if o elseif
          comando35 ;
          etc.
end
```

NOTA: puede haber tantos `elseif` como se quiera
NOTA: sólo se ejecuta uno de los `if, elseif` o `else`
NOTA: no es necesario el bloque `else`

Ejercicio: Combinación de bucles anidados y condicionales

a) Generar una matriz cuadrada *M* de 15x15 cuyos elementos $M_{i,j}$ sean:
$M_{i,j}$=0 si $i<j$
$M_{i,j}=m!/(n!(m-n)!)$ en caso contrario, donde m=*i-1,* n=*j-1.* (Son los coeficientes del binomio de Newton)

b) Generar una matriz *N* cuyos elementos son:
$N_{i,j}=M_{j,i}$ si $M_{j,i}$ es par (cero se considera par)
$N_{i,j}=-M_{j,i}$ si $M_{j,i}$ es impar.

Ejercicio: para evitar la singularidad del potencial eléctrico producido por una carga puntual, puede utilizarse el potencial de una *esfera* de radio R con carga total *Q*, distribuida uniformemente en su interior, a distancia *r*.

*V=K*Q*(3*R^2-r^2)/(2*R^3)* si *r<R*
*V=K*Q/r* si *r>=R*

Escribir una función *sphere_potential.m* que obtenga el potencial eléctrico de una esfera de carga *Q* y radio *R* en el punto (*xQ,yQ*), en la posición (*x,y*), que pueden ser matrices.

```
function [U_pot]=sphere_potential(Q,R,xQ,yQ,x,y)
```

Ejercicio: Escribir un *script* para representar gráficamente el potencial y el campo eléctrico producido por una distribución de *cargas esféricas*, empleando la función anterior.

Utilidades. Acceso a elementos de una matriz mediante matrices lógicas (en lugar de utilizar bucles `for` y condicionales `if`)

Para establecer los elementos de la diagonal de A igual a 3.6

```
A=rand(5,5)
B=(eye(size(A))~=0) % obtener matriz lógica con true en la diagonal
A(B)=3.6
```

Esto es, si el *argumento* de índices es una matriz lógica (en lugar de números) de la misma dimensión que A, se escogen los elementos que son true.

Encontrar los valores de los elementos de A menores que 0.5 e igualarlos a 0

```
A=rand(5,5) ;
B=A<0.5 ; % obtener matriz lógica
A(B)=0 ;
```

Comando `find`

Encontrar los índices de elementos de un vector que cumplen una condición: `help` **`find`**

```
a=rand(1,10) ;
i_vector=find(a<0.5)
```

Bucles. Los comandos `break` y `continue`.

Añaden control sobre la ejecución de un bucle `for` o `while`.

`break`: sale del bucle para seguir ejecutando los comandos después de `end`, esto es, termina forzadamente el bucle.

`continue`: deja de ejecutar los comandos posteriores y comienza la ejecución de otra iteración del bucle, esto es, salta hacia detrás hasta el comienzo del bucle.

```
a=0 ;
for n=1:20
       if a>=10
              break ;
       end
       a=a+1 ;
end ;
disp(['a=',num2str(a)]) ;
% el resultado debería ser a=10
```

Funciones anónimas (de usuario)

Se trata de una versión simplificada de las funciones de usuario, para las que no es necesario generar un fichero .m. Pueden utilizarse en un script o función después de haber sido definidas.

Sintaxis, ejemplo (atención a la arroba):

```
my_fun_name=@(arg1,arg2,...)  arg1+arg2 ;
```

- Sólo pueden contener una línea de código.
- Pueden tener ninguno o más argumentos.
- Sólo devuelven una respuesta, aunque puede ser un valor, un vector o una matriz.

Ejemplo:

```
sumar_dos_valores=@ (a,b) a+b ;
sumar_dos_valores(3,4)
```

Ejemplo (devuelve una matriz)

```
generar_diag_valor=@(n,d) d*eye(n) ;
generar_diag_valor(3,17)
```

NOTA: una función @ puede llamar a otra función @ (anidadas).

COMANDOS, FUNCIONES, OPERADORES

Matrices: `[] , ; ()`
Bucles: `for while break continue`
Funciones de usuario `function % %%`
Variables lógicas: `true false`
Condicionales: `if elseif else`
Operadores de comparación numérica: `== > >= < <= ~=`
Operadores entre variables lógicas: `~ | &`
Utilidades: `zeros ones rand eye diag`
Comando `find`
Matrices: operadores `+ - .* ./ .^` (elemento a elemento)
Matrices: operador `*` (producto matricial)
Funciones de matrices: `inv det gradient transpose (') length size`
Superficies 3D: `meshgrid surf contour, surf, surfc, surfl, mesh, quiver`
Utilidades 3D: `colorbar shading zlabel title zlim`

DESTREZAS

Matrices: construcción (:) y acceso a elementos, filas, columnas, etc.
Matrices: operaciones, funciones, matrices predefinidas.
Bucles (`for while`), anidamiento de bucles.
Funciones de usuario: parámetros y resultados (múltiples y con dimensión dispar).
Funciones anónimas de usuario
Variables y vectores lógicos, operadores comparación y lógicos.
Ejecución condicional (`if elseif else`).
Acceso a elementos mediante matrices lógicas.
Gráficos de superficies 3D, formato, etiquetado.

ATENCIÓN a los ejemplos y ejercicios propuestos en el aula

APÉNDICE: Cómo mostrar resultados de un cálculo en la ventana de comandos.

Ya se ha visto la utilización de las funciones `disp` y `num2str` para mostrar valores numéricos en la ventana de comandos.

Existe también otra función más avanzada (**`fprintf`**) que permite más posibilidades. Consultar la ayuda de Matlab para `fprintf`.

Ejemplo:

```
a=4.56 ; b=1.32E-2 ;
fprintf('valor de a = %.3f (m)\n',a) ; % \n es salto de línea
fprintf('valor a = %.3f (m), valor b = %.3E (m/s)\n',a,b) ;

% la función sprintf es igual que fprintf pero entrega el
% resultado como uma variable de texto
% s=sprintf(...) ; xlabel(s) ;
% xlabel(sprintf(...)) ;
```