Introducción a Matlab: Scripts, bucles de control, funciones y presentación de datos.

- En algunos de los apartados se incluye la respuesta **(en color azul)** a las cuestiones planteadas.
- Las respuestas de los ejercicios **en rojo** se deberán incluir en la Hoja de Entrega antes de la **fecha determinada en Moodle**.

Funciones

1. Escribir una función que reciba como argumentos de entrada un número x y un entero k y devuelva el valor de cos(kx):

```
function p = fun1(x,k)
  p= cos(k*x);
end
```

Aplicar para calcular el valor de cos(2*pi).

2. Escribir una función llamada cosn(x,n) que reciba como argumentos de entrada un vector x y un entero n y devuelva el valor (p) de $\cos^n(x)$. El segundo argumento de entrada puede ser opcional, en caso de omitirse n debe tomar el valor 2

```
function p = cosn (x,n)
if nargin==1, n=2; end
p= cos(x).^n;
end
```

Probad la función p=cosn(0.3,4). El resultado debería ser idéntico a cos(0.3)^4.
Observación: Notad que la función creada es válida si el primer argumento es un vector de x's y no un único número.

3. Escribir una función que reciba como argumentos dos valores a y b y devuelva su suma. Opcionalmente, si el usuario la llama con dos argumentos de salida, debe devolver también su resta.

```
function [S,R]=suma(a,b)
S = a+b;
if nargout==2, R = a-b; end
end
```

5. - Crear una función que reciba como argumento un vector x y un entero n y devuelva el valor de la función

$$f(x) = \frac{1}{1+|x|^n}.$$

- Usando la función anterior, hacer las gráficas de f(x) en el intervalo [-2,2] para n = 1,2,3 y 4 (superponer en el mismo gráfico). Para ello:
- Crear el vector x=-2:0.01:2 y calcular su longitud l (length).
- Calcular las potencias 1,2,3 y 4 de los valores de dicho vector (uilizando la función creada anteriormente) y guardar los resultados en una matriz 4xl (que llamaremos p) en la que las filas contienen los valores de la función para n=1,2,3 y 4 .
- Hacer el plot de las 4 salidas (en el mismo gráfico, hold on):

Volcado de datos formateados usando fprintf( )

1. Volcar el valor del número pi con 5 decimales: fprintf('%.5f\n',pi);

2. Volcar el número e = exp(1); dos veces: con 2 decimales en una línea y 8 decimales en la siguiente: e=exp(1); fprintf('%.2f\n%.8f\n',e,e);

3. Sea la variable x = 174. Usar fprintf( ) para volcar en una línea el valor de x con los siguientes formatos:
   - un entero (%d),
   - un entero reservando 4 columnas (%4d)
   - un entero, reservando 4 columnas y rellenando con 0's (%04d)
   - un número real con 2 decimales (%.2f)
   - un número real en notación científica (%e)

```
fprintf('%d %4d %04d %.2f %e\n',x,x,x,x,x)
```

4. Sea la matriz A = [2 -15 3 127; -97 32 0 3]. Usar fprintf para visualizar sus contenidos con un solo comando:

a) En cuatro líneas, cada una mostrando una columna de A como números enteros.
b) En dos líneas, cada una conteniendo una fila de la matriz como números enteros.
c) Idem antes, pero reservando 4 columnas de espacio para cada número para verlos alineados.

Bucles de control

1. Se va a calcular una estimación del límite de la siguiente sucesión:

$$\begin{cases} x_0 = 0 \\ x_{n+1} = \cos(x_n) \end{cases}$$

a) Calcular una estimación del límite de la sucesión dada implementando la iteración anterior 100 veces ¿Cuánto vale el límite de la sucesión? Pintar los resultado de cada iteración (plot …hold on hold off).

```
x=0;
for k=1:100,
x=cos(x);
 plot(k,x, '*r')
hold on
 end
hold off
 x
```

b) Se va a crear un script (trabajamos ya en el editor) para calcular (y guardar) los valores de la sucesión iterando 100 veces y estudiar los errores que cada iteración produce respecto del límite de la sucesión. Para ello: 1º Crea un vector x inicializado con ceros (en el que posteriormente se guardarán las sucesivas iteraciones). 2º En cada componente del vector x se van guardando los sucesivos valores de la sucesión. 3º Hacer una gráfica del vector x con asteriscos rojos. 4º Mostrar el valor de la última iteración y llamar l (asumiremos que es el límite de la sucesión). 4º Calcular el vector er que contiene los errores relativos de cada iteración respecto del valor límite l. 5º Calcular el vector ncif con el nº de cifras significativas que produce cada iteración. 6º En una ventana gráfica dividida en dos columnas (comando subplot(1,2,.)) incluir la gráfica del vector de errores relativos y la del nº de cifras significativas (en escalas adecuadas y ambas con *). Comentar los resultados.

```
% Crear vector x inicializado con ceros
x=zeros(101,1);    %x(1)=x0=0
% Calcular iteraciones y guarder en x:
for k=1:100,
   x(k+1)=cos(x(k));
end
% Dibujar gráfica de iteraciones:
plot(x,'*r')
%Valor estimado del límite sucesión:
l=x(end)
%Calculamos error relativo y nº de cifras decimales y representamos gráficamente:
er=abs(x(1:end)-l))/l);
ncif=floor(-log10(er));
subplot(121), semilogy(er,'*r')
subplot(122), plot(ncif,'*g')
```

2. Hacer un script que calcule las 50 primeras iteraciones de la sucesión

$$\begin{cases} x_0 = 0 \\ x_{n+1} = \frac{1}{2} e^{-x_n} \end{cases}$$

Dibujar una gráfica con los resultados. ¿Dar el último valor obtenido, a qué valor converge la sucesión x_n?

(Nota. El comando exp(x) de Matlab calcula el valor de la función exponencial de x).

3. Sea A=rand(5,7) la matriz 5x7 de números aleatorios entre 0 y 1.

a) Usar bucles anidados para asignar a los elementos de A que son menores o iguales a 0.4 el valor 0.2 y en otro caso (si son mayores que 0.4) el valor 0.7:

```
 for k=1:5, for j=1:7,
  if A(k,j)<=0.4, A(k,j)=0.2; else A(k,j)=0.7; end
end   end
```

b) Repetir usando indexado lógico.
A(A<=0.4)=0.2; A(A>0.4)=0.7;

4. Dado el vector x = rand(1,10), escribir código (usando un bucle for e if) que convierta cada valor del vector mayor o igual que 0.5 a 0.8, y en otro caso (los valores menores que 0.5) a 0. Repetir usando indexado lógico.

5. Se van a estudiar los valores aproximados del número 2 proporcionados al sumar términos de la siguiente serie

$$1+\frac{1}{2}+\frac{1}{2^2}+\frac{1}{2^3}+\frac{1}{2^4}+\ldots=2$$

1. Crear un script (trabajamos en editor) para calcular un valor aproximado de 2 (que llamaremos s) a partir de la serie anterior:
a) Sumando los primeros 61 sumandos de la serie anterior usando operaciones punto a punto y el comando sum, sin usar bucles.
b) Sumando los primeros 61 sumandos de la serie anterior usando un bucle for …. end . Nótese que cada término de la serie se puede calcular a partir del término anterior mediante

$$s_{n+1}=s_n+\frac{1}{2^n} \quad \text{con } s_1=1 \qquad (1)$$

c) Usando un bucle while … end que itere mientras que la solución aproximada s produzca un error (abs(s-2)>10^(-16)) mayor que 10^(-16) ¿Cuántas iteraciones realiza el bucle? Sabiendo que 10^(-16) es la precisión máxima que produce Matlab ¿tenía sentido programar la iteración (1) 60 veces?

2. Ahora vamos a crear un script en el que se generará un vector s con los valores aproximados obtenidos con n sumandos donde n varía desde 1 hasta 61 y estudiaremos los correspondientes errores. Para ello:
- Generamos un vector (denominado s) de ceros de la dimensión adecuada, donde se guardarán posteriormente los valores aproximados.
- Calculamos (bucle for) los valores aproximados obtenidos usando n sumandos con n de 1 a 61 y guardamos en el vector s (s(n)=valor aproximado utilizando n sumandos).
- Calculamos los errores relativos cometidos por las aproximaciones anteriores y los guardamos en un vector denominado erel.
Calcular cuál es el mínimo valor del vector erel e indicar para qué índice se alcanza. Comparar el resultado con el obtenido en el apartado 1c.
- Calculamos el nº de cifras decimales significativas que proporcionan las aproximaciones calculadas y guardamos en un vector denominado ncif. Mostrar los valores del vector ncif con n=50:60 ¿qué valores producen?
- En una ventana gráfica pintamos en gráficas separadas (comando subplot(1,2,)) la gráfica de los errores relativos (en una escala adecuada para que se aprecien los resultados y con asteriscos rojos '*r') y la de las cifras decimales (asteriscos verdes '*g'). Comentar qué se observa en las gráficas.
Nota: Incluir comentarios (%) en el script, indicando qué hace en cada etapa.

6. Se va a utilizar la relación

$$\sin(x)=x-\frac{x^3}{3!}+\frac{x^5}{5!}-\frac{x^7}{7!}+\ldots+(-1)^n\frac{(x)^{2n+1}}{(2n+1)!}+\ldots \quad \text{con n=0,1,…}$$

para estimar el valor del sin(x) en x =0.5:
1. Dar un valor aproximado de sin(0.5) usando los primeros 10 términos de la serie anterior y calcular el error relativo cometido usando los siguientes métodos:

a) Un script utilizando los comandos **.*** **./** **.^**
b) Un script utilizando un bucle **for end**.

2. Crear un script para estudiar y visualizar los errores producidos al aproximar sin(0.5) por la suma de n sumandos con n= 1:10. Para ello:

- Generamos un vector (denominado vap) de ceros de la dimensión adecuada, donde se guardarán posteriormente los valores aproximados. Calculamos los valores aproximados con n variando de 1 a 10 y los guardamos en el vector vap (vap(n)= suma de los n primeros términos de la serie dada).
- Calculamos los errores relativos cometidos para cada n y los guardamos en un vector denominado erel.
- Calculamos el nº de cifras decimales significativas que proporcionan las aproximaciones calculadas y guardamos en un vector denominado ncif ¿Era necesario utilizar tantos sumandos para calcular la solución con 14 cifras decimales?

3. Construir un script con un bucle **while** que sume los términos de la serie inicial hasta obtener un valor aproximado de sin(0.5) con 15 cifras decimales significativas (error relativo <1e-15) ¿Cuántas iteraciones efectúa?

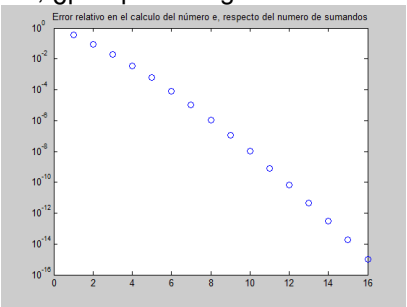
```
1.a)
x=0.5;
n=0:9;
term=((-1).^n).*((x).^(2*n+1))./(factorial(2*n+1)); %TERMINOS QUE HAY QUE SUMAR
v_apr=sum(term);
1b)
x=0.5; v_apr=0;
for n=0:9;
      v_apr=v_apr+((-1).^n).*((x).^(2*n+1))./(factorial(2*n+1));
end

2.
x=0.5;
vapr=zeros(1,10); vapr(1)=x;
for n=1:9;
      vapr(n+1)=vapr(n)+((-1).^n).*((x).^(2*n+1))./(factorial(2*n+1));
end
% Error relativo y el nº de cifras decimales significativas obtenidas:
erel=abs(sin(x)-vapr)/abs(sin(x));
ncif=floor(-log10(erel)) (1   3   5   7   10   13  Inf  Inf  Inf  Inf)
Se observa que con 7 o más sumandos se ha alcanzado la precisión de la máquina.
3.
x=0.5; vapr=0;  n=0; erel=10;
while(erel>=1e-15),
      vapr=vapr+((-1).^n).*((x).^(2*n+1))./(factorial(2*n+1));
      erel=abs(sin(x)-vapr)/abs(sin(x));
      n=n+1;
 end
```

7. Utilizar la relación $e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \ldots$

para estimar el valor del número e, sumando los primeros 30 términos de la serie. Calculamos la estimación de 5 formas distintas, unas más eficientes que otras, respecto del coste computacional (la más eficiente es la que menos operaciones utiliza).

a) Utilizando los comandos **.*, ./, .^** (poco eficiente, recalcula el factorial para cada valor de n).
   n=[0:29]; fact=factorial(n); s=sum(1./fact)
   Sabiendo que e=exp(1) ¿Cuál es el error cometido con la estimación?
b) Mediante un bucle **for** (poco eficiente ya que recalcula el factorial).
   s=1; for k=1:29, s=s+1/factorial(k); end
   ¿Obtenemos la misma estimación que en el apartado anterior?
c) Modificando el bucle anterior para sea más eficiente (calcula el factorial iterativamente).
   s=1; temp=1; for k=1:29, temp=temp/k; s=s+temp; end
   ¿Hay alguna diferencia con la solución anterior?
d) Repetir el bucle anterior pero de forma que el bucle termine (break) si el término a sumar es menor que 10^(-16). ¿Cuántas iteraciones del bucle se hacen? ¿Hay diferencias entre ambas soluciones?
   s=1; temp=1; for k=1:29, temp=temp/k; s=s+temp; if temp<1e-16; break; end; end
   ¿Cuántas iteraciones del bucle se hacen? ¿Por qué motivo se impone que el bucle termine si el término a sumar es menor que 10^(-16), tendría sentido imponer un error menor?
e) Implementar de forma más elegante el problema anterior usando un bucle while, con la condición de que el término a sumar sea mayor que 10^(-16). Calcular el número de iteraciones.
   s=0; temp=1; k=1; while(temp>1e-16), s=s+temp; temp=temp/k; k=k+1; end

f) Calcular el error relativo al estimar el número e con las aproximaciones anteriores para n=1:20 (número de sumandos), y dibujar una gráfica con los resultados. ¿Cuál es la precisión máxima obtenida? ¿cuántas cifras significativas se alcanzan para esa precisión máxima?, ¿para qué valor de n se alcanza esa precisión máxima?. Con cada nuevo sumando de la serie, ¿cuántas cifras más de precisión obtenemos (aproximadamente)? ¿Aparecen todos los datos en la gráfica?, ¿por qué?. La gráfica obtenida debe ser similar a la siguiente.



8. Calcular la suma de los inversos de los impares menores de 1000:

$$S = 1 + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \ldots + \frac{1}{999}$$

a) Implementarlo con un bucle for end
b) Repetir sin emplear bucles (utilizando comandos punto a punto)
c) Vamos a medir cuánto tiempo lleva la ejecución de los comandos utilizados en los apartados anteriores. Para ello, en cada caso usaremos tic (al principio del programa a medir tiempo) y toc (al final del programa a medir tiempo). ¿Qué implementación, la hecha en el apartado (a) o en el (b) consume menos tiempo?
d) Modificar el script anterior (versión sin bucles), convirtiéndolo en una función que reciba como argumento de entrada el índice máximo a sumar (999 en el ejemplo anterior).