APELLIDOS, NOMBRE: SERRANO ARRESE, JULIA

Adjuntar código utilizado, valores y gráficas pedidos

Ejercicio de funciones

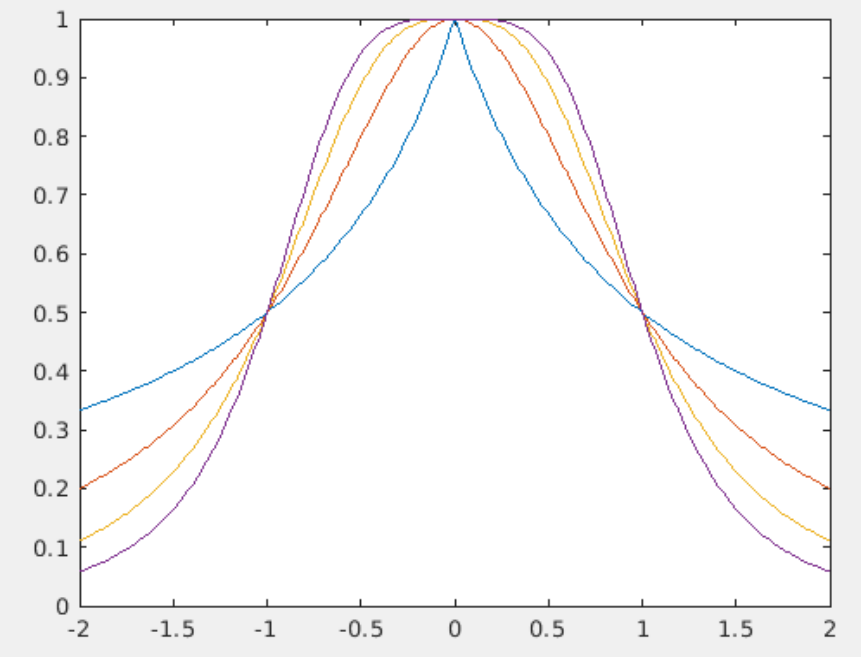
Crear una función que reciba como argumento un vector x y un entero n y devuelva el valor de la función

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

- 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):

```
function val = fun5(x,n)
    val = 1 ./ (1 + (abs(x).^ n));
end
  %5.
x = -2:0.01:2;      %intervalo
l = length(x);      %longitud
p = zeros(4,l);      %matriz 4xl
for i= 1:4
    p(i,:) = fun5(x,i);
end
plot(x,p)          %graficar en mismo gráfico
```



Ejercicio de volcado de datos

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.

```
fprintf('%d %d\n', A(:,1)', A(:,2)', A(:,3)', A(:,4)')
```

b) En dos líneas, cada una conteniendo una fila de la matriz como números enteros.

```
fprintf('%d %d %d %d\n',A(1,:),A(2,:))
```

c) Idem antes, pero reservando 4 columnas de espacio para cada número para verlos alineados.

```
fprintf('%4d %4d %4d %4d\n',A(1,:),A(2,:))
```

Ejercicio de bucles

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.

```
n = 0:60;

x1 = 1 ./ (2 .^ n);

vap1 = sum(x1);
```

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

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

```
x2 = ones(61,1);

for k = 1:60

  x2(k+1) = x2(k) + 1/ (2.^k);

end

vap2 = x2(end);
```

c) Usando un bucle while … end que itere mientras que la solución aproximada s produzca un error (abs(s2)>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?

```
n=1;                    %iteraciones

s = 1;                  %s1 = 1 v_aprox

v_real = 2;             %valor real

e_rel = abs(s - 2);     %error rel


while(e_rel>10^(-16))

        s(n+1) = s(n) + 1/ (2.^ n);             %calculo sol aprox

        n=n+1;                          %sig iteracion

        e_rel = abs(s(n) - 2);

        fprintf('Numero de iteraciones: %d Error: %d\n ',n,e_rel)

end
```

El bucle realiza 54 iteraciones. No tiene sentido iterar 60 veces ya que no se va a conseguir mayor precisión, al estar limitada por Matlab.

```
Numero de iteraciones: 2 Error: 5.000000e-01
Numero de iteraciones: 3 Error: 2.500000e-01
Numero de iteraciones: 4 Error: 1.250000e-01
Numero de iteraciones: 5 Error: 6.250000e-02
Numero de iteraciones: 6 Error: 3.125000e-02
Numero de iteraciones: 7 Error: 1.562500e-02
Numero de iteraciones: 8 Error: 7.812500e-03
Numero de iteraciones: 9 Error: 3.906250e-03
Numero de iteraciones: 10 Error: 1.953125e-03
Numero de iteraciones: 11 Error: 9.765625e-04
Numero de iteraciones: 12 Error: 4.882812e-04
Numero de iteraciones: 13 Error: 2.441406e-04
Numero de iteraciones: 14 Error: 1.220703e-04
Numero de iteraciones: 15 Error: 6.103516e-05
Numero de iteraciones: 16 Error: 3.051758e-05
Numero de iteraciones: 17 Error: 1.525879e-05
Numero de iteraciones: 18 Error: 7.629395e-06
Numero de iteraciones: 19 Error: 3.814697e-06
```

```
Numero de iteraciones: 20 Error: 1.907349e-06
Numero de iteraciones: 21 Error: 9.536743e-07
Numero de iteraciones: 22 Error: 4.768372e-07
Numero de iteraciones: 23 Error: 2.384186e-07
Numero de iteraciones: 24 Error: 1.192093e-07
Numero de iteraciones: 25 Error: 5.960464e-08
Numero de iteraciones: 26 Error: 2.980232e-08
Numero de iteraciones: 27 Error: 1.490116e-08
Numero de iteraciones: 28 Error: 7.450581e-09
Numero de iteraciones: 29 Error: 3.725290e-09
Numero de iteraciones: 30 Error: 1.862645e-09
Numero de iteraciones: 31 Error: 9.313226e-10
Numero de iteraciones: 32 Error: 4.656613e-10
Numero de iteraciones: 33 Error: 2.328306e-10
Numero de iteraciones: 34 Error: 1.164153e-10
Numero de iteraciones: 35 Error: 5.820766e-11
Numero de iteraciones: 36 Error: 2.910383e-11
Numero de iteraciones: 37 Error: 1.455192e-11
Numero de iteraciones: 38 Error: 7.275958e-12
Numero de iteraciones: 39 Error: 3.637979e-12
Numero de iteraciones: 40 Error: 1.818989e-12
Numero de iteraciones: 41 Error: 9.094947e-13
Numero de iteraciones: 42 Error: 4.547474e-13
Numero de iteraciones: 43 Error: 2.273737e-13
Numero de iteraciones: 44 Error: 1.136868e-13
Numero de iteraciones: 45 Error: 5.684342e-14
Numero de iteraciones: 46 Error: 2.842171e-14
Numero de iteraciones: 47 Error: 1.421085e-14
Numero de iteraciones: 48 Error: 7.105427e-15
Numero de iteraciones: 49 Error: 3.552714e-15
Numero de iteraciones: 50 Error: 1.776357e-15
Numero de iteraciones: 51 Error: 8.881784e-16
Numero de iteraciones: 52 Error: 4.440892e-16
Numero de iteraciones: 53 Error: 2.220446e-16
Numero de iteraciones: 54 Error: 0
```

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?

```
n = 1:61;

s = zeros(61,1);

erel = zeros(61,1);

ncif = zeros(61,1);

s(1) = 1;

erel(1) = abs(2 - s(1));

ncif(1) = floor(-log10(erel(1)));

for n = 1:61

    s(n+1) = s(n) + 1/ (2.^ n);

    erel(n+1) = abs(  2 - s(n+1));

    ncif(n+1) = floor(-log10(erel(n+1)))

end

erel;

[min_erel,pos] = min(erel);

fprintf('Numero cifras significativas: %d\n',ncif([50:60],:))
```

El mínimo error relativo es 0 y se encuentra en la posición 54. Hemos obtenido la misma solución que en el apartado 1c, solo que hemos realizado iteraciones innecesarias.

```
Numero cifras significativas: 14
Numero cifras significativas: 15
Numero cifras significativas: 15
Numero cifras significativas: 15
Numero cifras significativas: Inf
Numero cifras significativas: Inf
Numero cifras significativas: Inf
Numero cifras significativas: Inf
Numero cifras significativas: Inf
Numero cifras significativas: Inf
Numero cifras significativas: Inf
```

- 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.

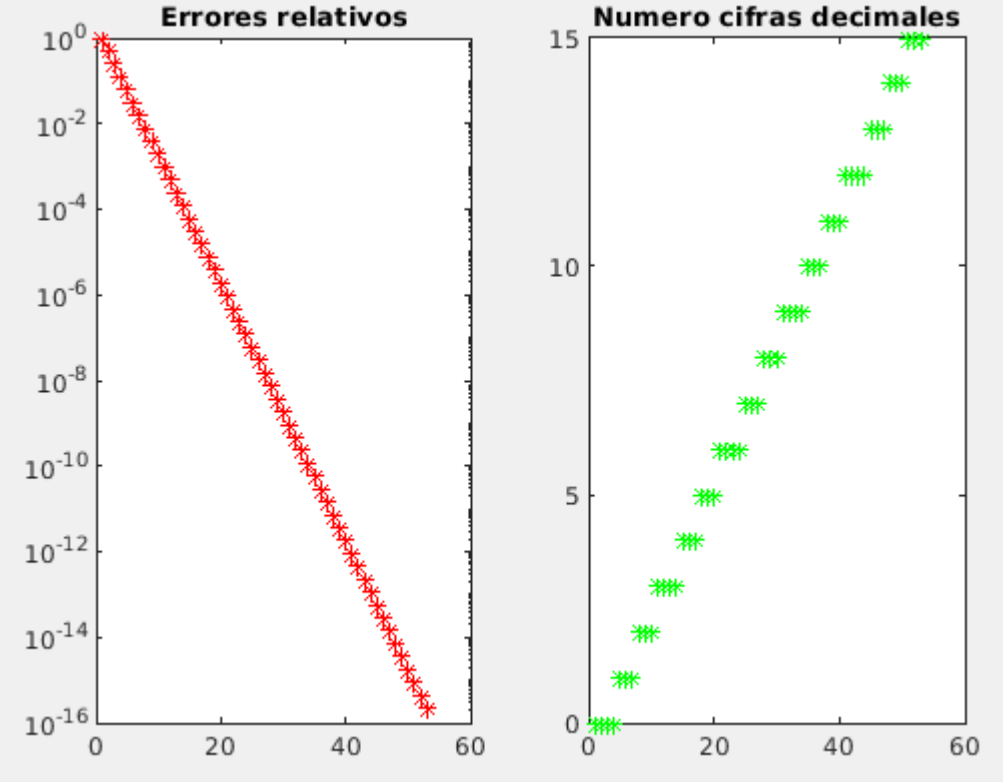
Nota: Incluir comentarios (%) en el script, indicando qué hace en cada etapa.

```
%grafica errores relativos

subplot(1,2,1), semilogy(erel,'*r'), title('Errores relativos')


%grafica cifras decimales

subplot(1,2,2), plot(ncif,'*g'), title('Numero cifras decimales')
```



Como se puede observar en las gráficas, el error relativo se va reduciendo según aumentamos el número de iteraciones y justo lo contrario pasa con el número de cifras decimales, que aumenta según el número de iteraciones es más alto.