Table of Contents

P8 : Cortes y García

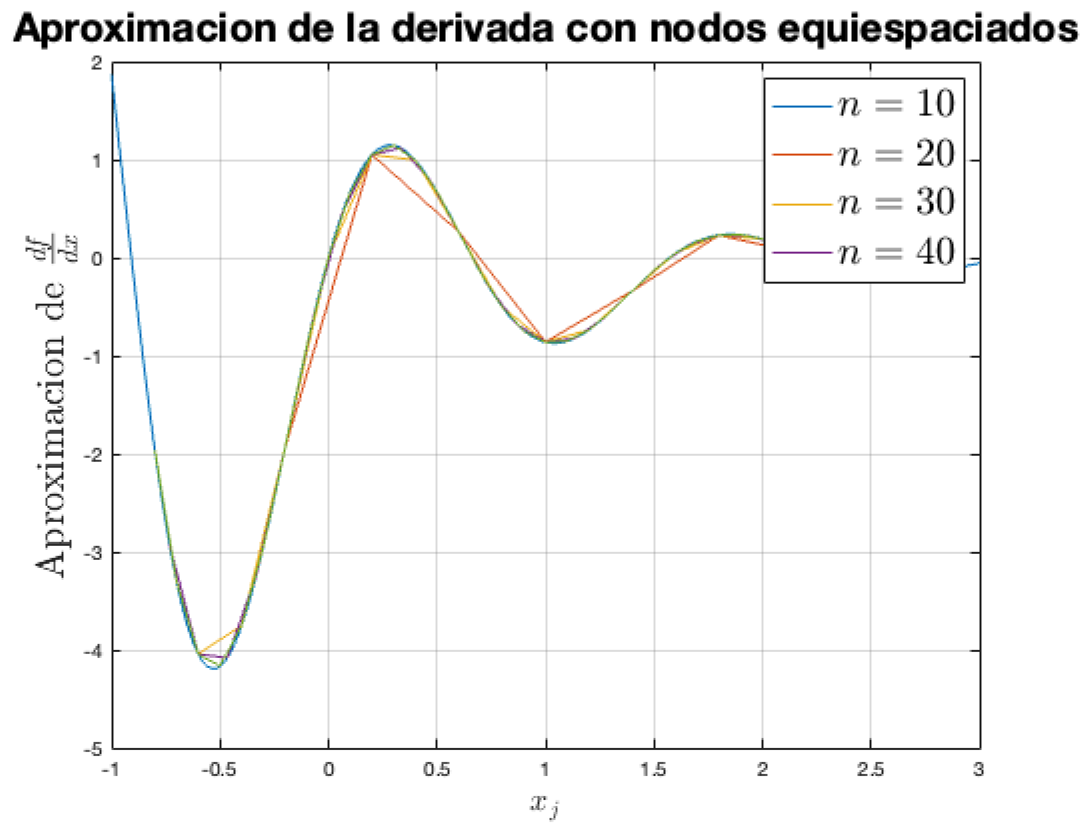
```
clear all
format long
```

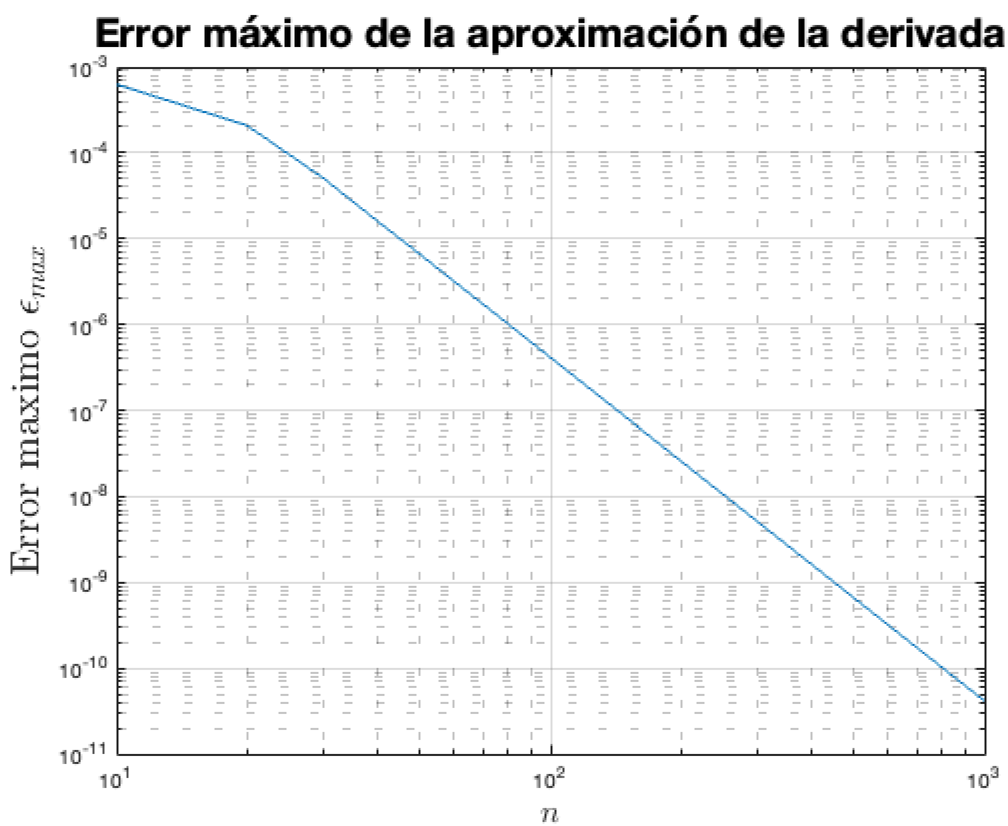
Ejercicio 1: Diferencia finita local

```
a = -1; b = 3; % Intervalo I = [a, b] = [-1, 3]
ff = @(x) exp(-x).*sin(2*x).^2;
ddff = @(x) exp(-x).*sin(2*x).*(4*cos(2*x) - sin(2*x));

z = linspace(-1, 3, 1000);

% Aproximamos el error para n = 10, 20, 30, 40
figure(1)
plot(z, ddff(z), 'LineWidth', 1)
hold on
for n = [10:10:40]
    % Creamos el vector de nodos equiespaciados
    h = 1/n;
    j = [0:n];
    xj = -1 + 4*h*j;

    df_aprox = (1 /(12*h))*(-ff(xj+2*h)+8*ff(xj+h)-8*ff(xj-
h)+ff(xj-2*h));

    % Puesto que para calcular la aproximación de f'(x) en x0, x1,
 x(n-1) y
    % x(n) estamos usando nodos que no existen (x(-2), x(-1), x(n+1),
    % x(n+2)) tenemos que eliminar los valores de los extresos del
 vector,
    % correspondientes a estos nodos.
    df_aprox = df_aprox(3:end-2);

    plot(xj(3:end-2), df_aprox, 'LineWidth', 1)
end
hold off
grid on
title('Aproximacion de la derivada con nodos
 equiespaciados', 'Fontsize', 20);
xlabel('$x_j$', 'Interpreter', 'Latex', 'FontSize', 16);
```

```
ylabel('Aproximacion de $\frac{df}
{dx}$', 'Interpreter', 'Latex', 'FontSize', 20);
legend('$n=10$', '$n=20$', '$n=30$', '$n=40$', 'Interpreter', 'Latex', 'FontSize',
 20);

% Calculamos errores máximos
e_max = [];
for n = [10:10:1000]
    % Creamos el vector de nodos equiespaciados
    h = 1/n;
    j = [0:n];
    xj = -1 + 4*h*j;

    df_aprox = (1 /(12*h))*(-ff(xj+2*h)+8*ff(xj+h)-8*ff(xj-
h)+ff(xj-2*h));

    % Puesto que para calcular la aproximación de f'(x) en x0, x1,
 x(n-1) y
    % x(n) estamos usando nodos que no existen (x(-2), x(-1), x(n+1),
    % x(n+2)) tenemos que eliminar los valores de los extresos del
 vector,
    % correspondientes a estos nodos.
    df_aprox = df_aprox(3:end-2);
    error = df_aprox - ddff(xj(3:end-2));
    e_max = [e_max max(error)];
end

figure(2)
loglog([10:10:1000], e_max, 'LineWidth', 1);
grid on;
title('Error máximo de la aproximación de la derivada', 'Fontsize',
 20);
xlabel('$n$', 'Interpreter', 'Latex', 'FontSize', 16);
ylabel('Error maximo $
\epsilon_{max}$', 'Interpreter', 'Latex', 'FontSize', 20);

r = polyfit(log10([10:10:1000]), log10(e_max), 1);
p = r(1)

% Obtenemos p = -3.91058, es decir, el error decae con una potencia
 p=-4
```

```
p =

  -3.910585393880240
```

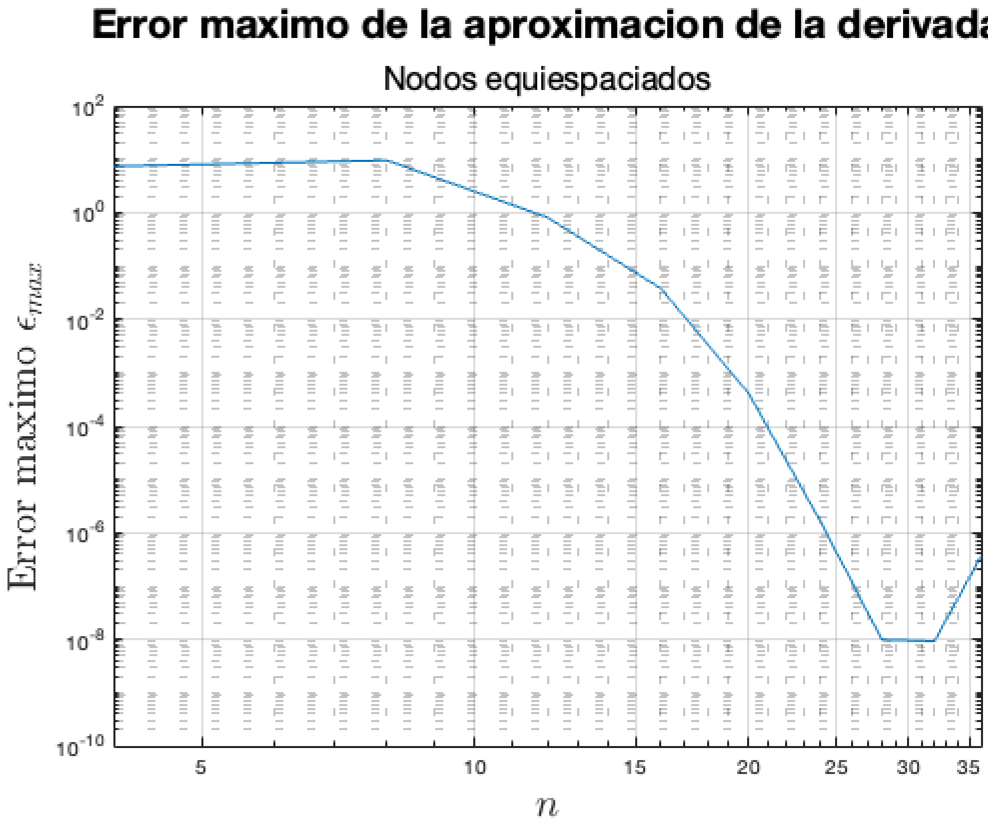

Aproximacion de la derivada con nodos equiespaciados
$n = 10$
$n = 20$
$n = 30$
$n = 40$
Aproximacion de $\frac{df}{dx}$
x_j
2
1
0
-1
-2
-3
-4
-5
-1
-0.5
0
0.5
1
1.5
2
2.5
3


Error máximo de la aproximación de la derivada
Error maximo ϵ_{max}
n
10^{-3}
10^{-4}
10^{-5}
10^{-6}
10^{-7}
10^{-8}
10^{-9}
10^{-10}
10^{-11}
10^{1}
10^{2}
10^{3}

Ejercicio 2: Derivación global - Nodos equiespaciados

```
e_max = [];
for nn = [4:4:36]
    % Creamos el vector de nodos equiespaciados. Utilizamos a= -1, b =
 3
    h = 1/nn;
    j = [0:nn];
    xj = -1 + 4*h*j;

    % Creamos la matriz D y inicializamos todos sus elementos a 0
    D = zeros(nn+1,nn+1);

    % Calculamos los pesos baricéntricos
    lamdas = ones(1,nn+1);

    for ii = 1:nn+1
        for kk = 1:nn+1
            if (kk ~= ii)
                lamdas(ii) = lamdas(ii)/(xj(ii)-xj(kk));
            end
        end
    end

    % Rellenamos la matriz D con los valores requeridos
    for ii = 1:nn+1
        for jj = 1:nn+1
            if (ii == jj)
                D(ii,jj) = sum(1./(xj(jj)-xj(1:jj-1)))+sum(1./(xj(jj)-
xj(jj+1:end)));
            else
                D(ii,jj) = (lamdas(jj)/lamdas(ii))/(xj(ii)-xj(jj));
            end
        end
    end

    % Calculamos la aproximación de la función
    fj = ff(xj)'; % Evaluación de f en los nodos
    df_aprox = D*fj; % Aproximación de la función
    df = ddff(xj)'; % Evaluación de la derivada en los nodos

    error = abs(df_aprox - df);

    e_max = [e_max max(error)];
end

figure(3)
loglog([4:4:36], e_max, 'LineWidth', 1);
grid on;
title('Error maximo de la aproximacion de la derivada', 'Fontsize',
 20);
```

```
subtitle('Nodos equiespaciados', 'Fontsize', 16);
xlabel('$n$', 'Interpreter', 'Latex', 'FontSize', 20);
ylabel('Error maximo $
\epsilon_{max}$', 'Interpreter', 'Latex', 'FontSize', 20);

% Observamos que de 4 a 32 el error máximo decrae de forma más rápida
 que
% en el método anterior. Sin embargo, solo con aumentar el número de
 nodos
% hasta 64 ya vemos que el error crece nuevamente y supera los valores
% alcanzados anteriormente. Esto se debe al fenómeno de Runge que se
% produce normalmente al usar nodos equiespaciados para evaluar
 algunas
% funciones.
```

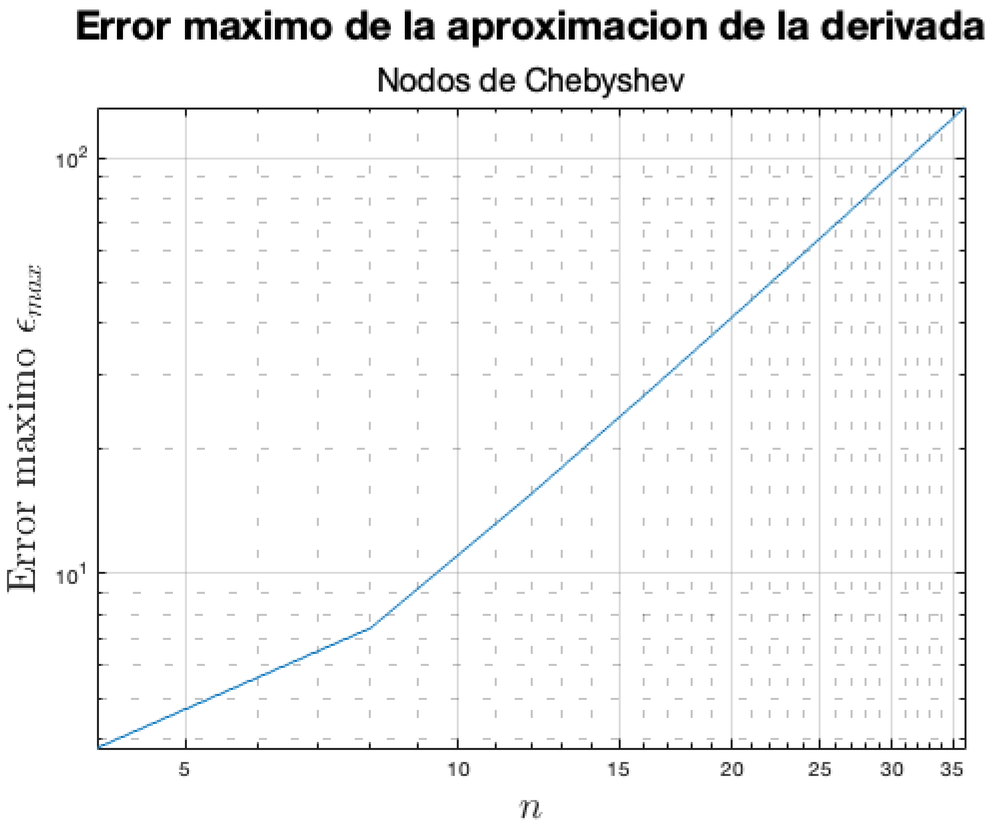


Ejercicio 2: Derivación global - Nodos de Chebyshev

A continuación repetiremos el ejercicio 2 usando los nodos de Chebyshev en vez de los nodos equiespaciados.

```
e_max = [];
for nn = [4:4:36]
    % Creamos la matriz D y inicializamos todos sus elementos a 0
    [D_cheb, xj] = chebdiff(nn);
```

```
    % Calculamos la aproximación de la función
    fj = ff(xj); % Evaluación de f en los nodos
    df_aprox = D_cheb.*fj; % Aproximación de la función
    df = ddff(xj); % Evaluación de la derivada en los nodos

    error = abs(df_aprox - df);
    m = max(error);

    e_max = [e_max m(1)];
end

figure(4)
loglog([4:4:36], e_max, 'LineWidth', 1);
grid on;
title('Error maximo de la aproximacion de la derivada', 'Fontsize',
 20);
subtitle('Nodos de Chebyshev', 'Fontsize', 16);
xlabel('$n$', 'Interpreter', 'Latex', 'FontSize', 20);
ylabel('Error maximo $
\epsilon_{max}$', 'Interpreter', 'Latex', 'FontSize', 20);

% Esta vez vemos que en general el error es menor que al usar los
 nodos
% equiespaciados. Por el contrario, vemos que el error crece, aunque
 de
% forma lineal y no tan bruscamente como lo hace en el caso de los
 nodos
% equiespaciados. Esto se debe a que, como ya habíamos constatado
% anteriormente, los nodos de Chebyshev ayudan a paliar el fenómeno de
% Runge al aproximar funciones.
```



Implementación de funciones usadas

```
% FUNCIÓN chebdiff
% Input: n
% Output: differentiation matrix D and Chebyshev nodes
% function [D,x] = chebdiff(n)
%     x =cos([0:n]'*pi/n);
%     d = [.5;ones(n-1,1);.5];
%     D = zeros(n+1,n+1);
%
%     for ii = 0:n
%         for jj = 0:n
%             ir = ii + 1; jc = jj + 1;
%             if ii == jj
%                 kk = [0:ii-1 ii+1:n]';
%                 num = (-1).^kk.*d(kk+1);
%                 D(ir,jc) =((-1)^(ir)/d(ir))*sum(num./(x(ir)-x(kk
+1)));
%             else
%                 D(ir,jc) = d(jc)*(-1)^(ii+jj)/((x(ir)-x(jc))*d(ir));
%             end
%         end
%     end
% end
```

Published with MATLAB® R2020b