Escuela Superior de Ingeniería y Tecnología (UNIR)

Máster Universitario en Ingeniería Matemática y Computación

- OPTIMIZACIÓN -

Actividad 1: Optimización de funciones

Alumno: Pablo Ruiz Molina

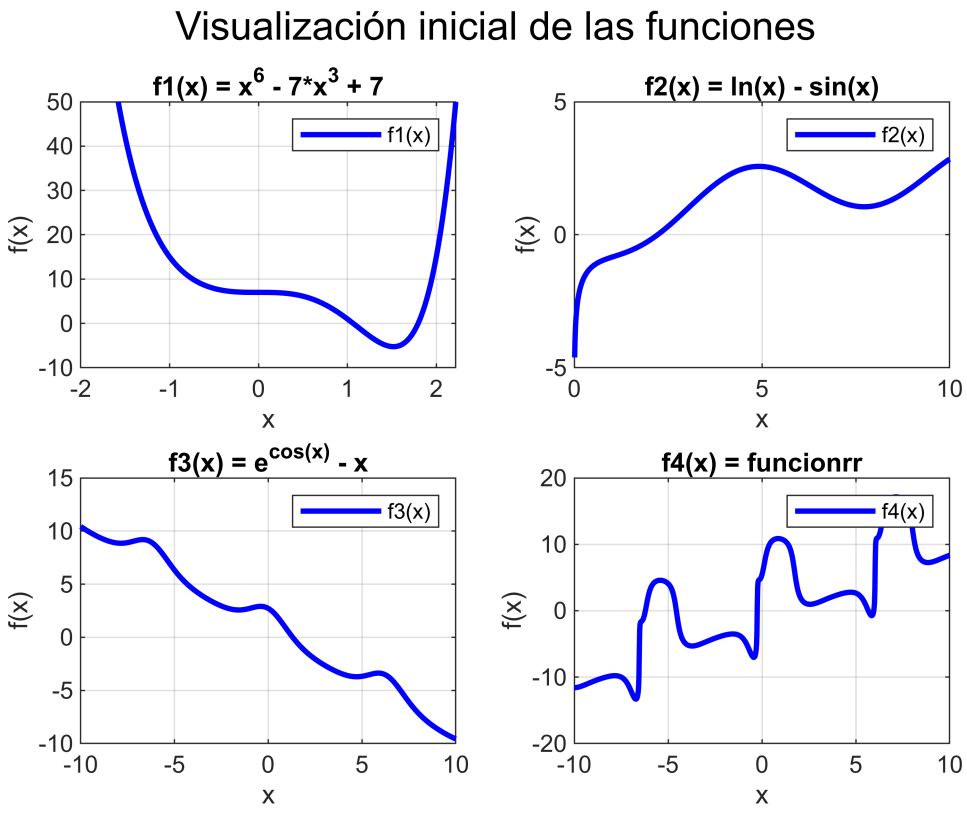
1. Selección de intervalo de longitud 1 de las funciones test

En el primer apartado de esta actividad se busca seleccionar un intervalo de longitud 1, en el cual la función sea unimodal y contenga el mínimo localizado en el valor positivo más pequeño.

Para entender qué tipo de funciones hay que enfrentar, el primer paso es visualizarlas. Se han escogido rangos ciertamente amplios para entender las funciones de manera general. El código usado para ello es el siguiente:

```
% Visualización inicial de las funciones

% Limpieza de la consola, variables y figuras
clc; clear; close all;

% Definición de las funciones objetivo
f1 = @(x) (x.^6) - 7*(x.^3) + 7;
f2 = @(x) log(x) - sin(x);
f3 = @(x) exp(1).^cos(x) - x;
f4 = @(x) funcionrr(x);
%f4 = @(x) x;

%% Visualización inicial
x1_values = linspace(-10, 10, 1000); y1_values = f1(x1_values);
x2_values = linspace(0, 10, 1000); y2_values = f2(x2_values);
x3_values = linspace(-10, 10, 1000); y3_values = f3(x3_values);
x4_values = linspace(-10, 10, 1000); y4_values = f4(x4_values);

figure (1);
subplot(2, 2, 1), plot(x1_values, y1_values, 'b-', 'LineWidth', 2);
title('f1(x) = x^6 - 7*x^3 + 7');
ylim([-10, 50]);
xlabel('x');
ylabel('f(x)');
legend('f1(x)');
hold on;
grid on;

subplot(2, 2, 2), plot(x2_values, y2_values, 'b-', 'LineWidth', 2);
title('f2(x) = ln(x) - sin(x)');
ylim([-5, 5]);
```

```
xlabel('x');
ylabel('f(x)');
legend('f2(x)');
hold on;
grid on;

subplot(2, 2, 3), plot(x3_values, y3_values, 'b-', 'LineWidth', 2);
title('f3(x) = e^{cos(x)} - x', 'Interpreter', 'tex');
xlabel('x');
ylabel('f(x)');
legend('f3(x)');
hold on;
grid on;

subplot(2, 2, 4), plot(x4_values, y4_values, 'b-', 'LineWidth', 2);
title('f4(x) = funcionrr');
xlabel('x');
ylabel('f(x)');
legend('f4(x)');
hold on;
grid on;

sgtitle('Visualización inicial de las funciones');
```

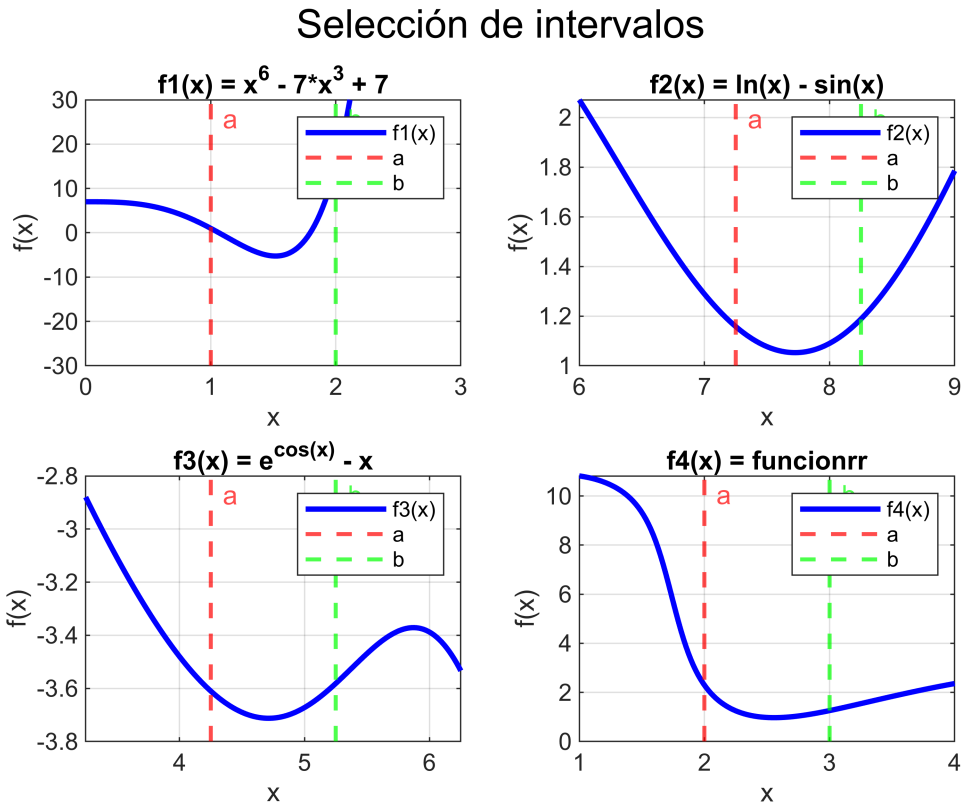


A destacar cómo la segunda función (arriba a la derecha) cuenta con un límite de $-\infty$ en $x = 0$. Es de hecho esperable según el rango donde se define la función en el enunciado: $(0, \infty)$.

Es fácil seleccionar para cada función un intervalo unidad en el que la función sea unimodal, con un mínimo, y que a su vez sea el más cercano a 0 por la parte positiva.

```
%% Selección del intervalo de búsqueda de cada función
% Calcula intervalos de manera automática
intervalo_f1 = [1, 2];
intervalo_f2 = [7.25, 8.25];
intervalo_f3 = [4.25, 5.25];
intervalo_f4 = [2, 3];

figure (2);
subplot(2, 2, 1), plot(x1_values, y1_values, 'b-', 'LineWidth', 2);
subplot(2, 2, 1), xline(intervalo_f1(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 1), xline(intervalo_f1(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
legend('f1(x)', 'a', 'b');
title('f1(x) = x^6 - 7*x^3 + 7');
xlim([0, 3]);
ylim([-30, 30]);
xlabel('x');
ylabel('f(x)');
hold on;
grid on;

subplot(2, 2, 2), plot(x2_values, y2_values, 'b-', 'LineWidth', 2);
subplot(2, 2, 2), xline(intervalo_f2(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 2), xline(intervalo_f2(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
legend('f2(x)', 'a', 'b');
title('f2(x) = ln(x) - sin(x)');
xlim([6, 9]);
xlabel('x');
ylabel('f(x)');
hold on;
grid on;

subplot(2, 2, 3), plot(x3_values, y3_values, 'b-', 'LineWidth', 2);
subplot(2, 2, 3), xline(intervalo_f3(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 3), xline(intervalo_f3(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
legend('f3(x)', 'a', 'b');
title('f3(x) = e^{cos(x)} - x', 'Interpreter', 'tex');
xlim([3.25, 6.25]);
xlabel('x');
ylabel('f(x)');
hold on;
grid on;
```

```
subplot(2, 2, 4), plot(x4_values, y4_values, 'b-', 'LineWidth', 2);
subplot(2, 2, 4), xline(intervalo_f4(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 4), xline(intervalo_f4(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
legend('f4(x)', 'a', 'b');
title('f4(x) = funcionrr');
xlim([1, 4]);
xlabel('x');
ylabel('f(x)');
hold on;
grid on;

sgtitle('Selección de intervalos');
```



2. Métodos de búsqueda

Una vez los intervalos de búsqueda están seleccionados, se van a desarrollar y probar los diferentes métodos de búsqueda objetivo de la actividad.

2.1 Método de búsqueda dicotómica

La búsqueda dicotómica es un método simple y eficiente diseñado para encontrar el mínimo de una función unimodal dentro de un intervalo dado. Este tipo de función tiene una sola tendencia descendente seguida de una ascendente (o viceversa), garantizando que exista un único mínimo local dentro del intervalo considerado. El algoritmo es iterativo y trabaja reduciendo progresivamente el tamaño del intervalo donde se encuentra el

mínimo, basándose en evaluaciones estratégicas de la función en puntos seleccionados dentro del intervalo. El código implementado ha sido el siguiente:

```
function [x_min, iter] = budi(f, a, b, delta)
    % Método de búsqueda dicotómica
    % Entradas:
    %   f: función objetivo
    %   a, b: extremos iniciales del intervalo unimodal
    %   delta: tolerancia para la convergencia
    %
    % Salidas:
    %   x_min: punto donde se alcanza el mínimo
    %   iter: número de iteraciones realizadas

    % Implementación del método

    % valores iniciales
    eps = delta/10; % Siempre más pequeño que la tolerancia.
    % Distancia entre los puntos (epsilon < delta)

    % Inicialización de las variables
    iter = 0;  % Contador de iteraciones
    max_iter = 100;  % Número máximo de iteraciones

    % mientras (b - a) >= tol
    while (b - a) > delta && iter < max_iter
        iter = iter + 1;

        % Calcular xa y xb
        x = (a + b) / 2;
        xa = x - eps / 2;
        xb = x + eps / 2;

        % Evaluar la función en los extremos del nuevo intervalo
        if f(xa) < f(xb)
            b = xb;
        else
            a = xa;
        end
    end
    % Cálculo del mínimo
    x_min = (a + b) / 2;
end
```

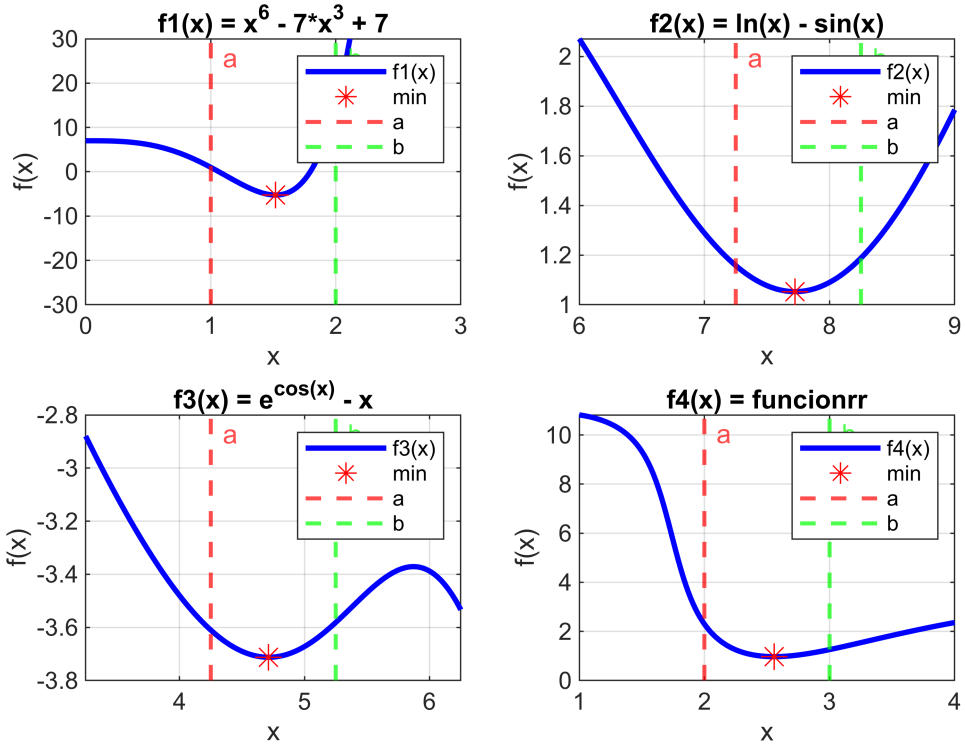
Los resultados obtenidos una vez se pasa el algoritmo a todas las funciones son los siguientes:

```
% Parámetros de la búsqueda dicotómica
delta = 1e-6;  % Tolerancia (delta)

% Ejecución de las búsquedas para cada función
```

```
tic;
[x1_budi, iter1_budi] = budi(f1, intervalo_f1(1), intervalo_f1(2), delta);
tiempo1_budi = toc * 1000;

tic;
[x2_budi, iter2_budi] = budi(f2, intervalo_f2(1), intervalo_f2(2), delta);
tiempo2_budi = toc * 1000;

tic;
[x3_budi, iter3_budi] = budi(f3, intervalo_f3(1), intervalo_f3(2), delta);
tiempo3_budi = toc * 1000;

tic;
[x4_budi, iter4_budi] = budi(f4, intervalo_f4(1), intervalo_f4(2), delta);
tiempo4_budi = toc * 1000;

figure (3);
f1_budi = f1(x1_budi);
subplot(2, 2, 1), plot(x1_values, y1_values, 'b-', 'LineWidth', 2);
hold on;
subplot(2, 2, 1), xline(intervalo_f1(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 1), xline(intervalo_f1(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 1), plot(x1_budi, f1_budi, 'r*', 'MarkerSize', 10, 'MarkerFaceColor',
'r');  % Punto óptimo
legend('f1(x)', 'a', 'b', 'min');
title('f1(x) = x^6 - 7*x^3 + 7');
xlim([0, 3]);
ylim([-30, 30]);
xlabel('x');
ylabel('f(x)');
grid on;

f2_budi = f2(x2_budi);
subplot(2, 2, 2), plot(x2_values, y2_values, 'b-', 'LineWidth', 2);
hold on;
subplot(2, 2, 2), xline(intervalo_f2(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 2), xline(intervalo_f2(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 2), plot(x2_budi, f2_budi, 'r*', 'MarkerSize', 10, 'MarkerFaceColor',
'r');  % Punto óptimo
legend('f2(x)', 'a', 'b',  'min');
title('f2(x) = ln(x) - sin(x)');
xlim([6, 9]);
xlabel('x');
ylabel('f(x)');
grid on;
```

```
f3_budi = f3(x3_budi);
subplot(2, 2, 3), plot(x3_values, y3_values, 'b-', 'LineWidth', 2);
hold on;
subplot(2, 2, 3), xline(intervalo_f3(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 3), xline(intervalo_f3(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 3), plot(x3_budi, f3_budi, 'r*', 'MarkerSize', 10, 'MarkerFaceColor',
'r');  % Punto óptimo
legend('f3(x)', 'a', 'b', 'min');
title('f3(x) = e^{cos(x)} - x', 'Interpreter', 'tex');
xlim([3.25, 6.25]);
xlabel('x');
ylabel('f(x)');
grid on;

f4_budi = f4(x4_budi);
subplot(2, 2, 4), plot(x4_values, y4_values, 'b-', 'LineWidth', 2);
hold on;
subplot(2, 2, 4), xline(intervalo_f4(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 4), xline(intervalo_f4(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 4), plot(x4_budi, f4_budi, 'r*', 'MarkerSize', 10, 'MarkerFaceColor',
'r');  % Punto óptimo
legend('f4(x)', 'a', 'b', 'min');
title('f4(x) = funcionrr');
xlim([1, 4]);
xlabel('x');
ylabel('f(x)');
grid on;

sgtitle('Búsqueda dicotómica: Mínimos encontrados.');
```

Búsqueda dicotómica: Mínimos encontrados.



Se aprecia a simple vista cómo se encuentra el punto mínimo en cada una de las funciones de manera precisa.

En la siguiente tabla, se detallan los valores obtenidos para el valor x_min, para el valor que cobra la función, las iteraciones necesarias, así como el tiempo de ejecución:

```
% Definir datos para las columnas
funciones = {'f1(x)'; 'f2(x)'; 'f3(x)'; 'f4(x)'};
x_min = [x1_budi; x2_budi; x3_budi; x4_budi];
f_x_min = [f1(x1_budi); f2(x2_budi); f3(x3_budi); f4(x4_budi)];
iter = [iter1_budi; iter2_budi; iter3_budi; iter4_budi];
times = [tiempo1_budi; tiempo2_budi; tiempo3_budi; tiempo4_budi];

% Crear una tabla
T = table(funciones, x_min, f_x_min, iter, times, 'VariableNames', {'Funciones', 'x
min', 'f(x min)', 'Iteraciones', 'Tiempos (msec)'});

% Mostrar la tabla en el Live Editor
disp(T);
```

Funciones	x min	f(x min)	Iteraciones	Tiempos (msec)
{'f1(x)'}	1.5183	-5.25	21	1.9008
{'f2(x)'}	7.7242	1.0528	21	1.8718
{'f3(x)'}	4.7124	-3.7124	21	1.1966
{'f4(x)'}	2.5574	0.97258	21	200.2

Será interesante comparar a continuación con los siguientes métodos los resultados de los valores x mínimos y los valores que toma la función en esos mínimos.

En relación con las iteraciones, es interesante darse cuenta cómo para todas las funciones se consume el mismo número de ellas, 21 en este caso. Podemos calcular las iteraciones mínimas necesarias por este algoritmo, en base a la tolerancia escogida:

```
% Cálculo de las iteraciones mínimas de Budi
iter_min_budi = log(delta)/log(.5);
disp(['Número de iteraciones mínimas para búsqueda dicotómica: iter_min <= ',
num2str(ceil(iter_min_budi))])
```

```
Número de iteraciones mínimas para búsqueda dicotómica: iter_min <= 20
```

Se aprecia cómom de acuerdo al parámetro delta escogido, son 20 iteraciones como mínimo. Como se comentaba, en los 4 casos se han usado 21 iteraciones, lo cuál es lógico.

En cuanto al tiempo de ejecución, se observa cómo para las tres primeras funciones el valor ronda entre 1-2 milisegundos. Mientras que para la función 4, el tiempo se dispara hasta casi los 200 milisegundos. Esto se debe a que las evaluciones que necesita hacer la función son en esta última función mucho más costosas por ser una función numérica un tanto compleja.

2.2 Método de interpolación cuadrática (*incu.m*)

La interpolación cuadrática es un método iterativo y eficiente para encontrar el mínimo de una función suave dentro de un intervalo dado. Este método asume que la función puede ser aproximada localmente por una parábola, y utiliza tres puntos del intervalo para construir esta aproximación cuadrática. A partir de esta parábola, se estima el punto donde ocurre el mínimo, lo que permite ajustar el intervalo para acercarse progresivamente al mínimo real. El proceso se repite iterativamente hasta alcanzar una precisión deseada, refinando el intervalo en cada paso y mejorando la aproximación del mínimo. El código implementado ha sido el siguiente:

```
function [x_min, iter] = incu(f, x1, x2, x3, delta)
    % Método de interpolación cuadrática
    % Entradas:
    %   f: función objetivo
    %   x1, x2, x3: puntos iniciales en los que x2<x1 y x2<3
    %   delta: tolerancia para la convergencia
    %
    % Salidas:
    %   x_min: punto donde se alcanza el mínimo
    %   iter: número de iteraciones realizadas

    % Implementación del método
    % Inicialización de las variables
    iter = 0;  % Contador de iteraciones
    max_iter = 100;  % Número máximo de iteraciones

    while iter < max_iter
        iter = iter + 1;
```

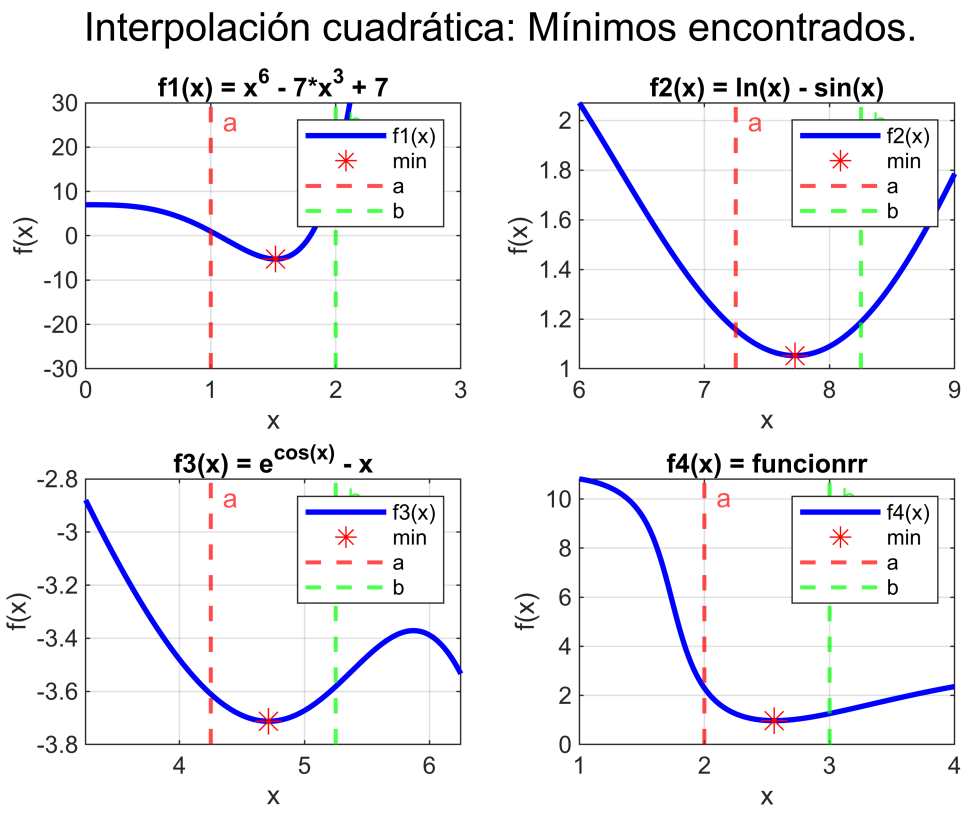
```
        % Calcular el valor mínimo del polinomio interpolador
        f1 = f(x1); f2 = f(x2); f3 = f(x3);
        num = ((x2^2-x3^2)*f1 + (x3^2-x1^2)*f2 + (x1^2-x2^2)*f3);
        den = 2*((x3-x2)*f1 + (x1-x3)*f2 + (x2-x1)*f3);
        x = -num/den;

        % Se cumple el criterio de parada?
        if abs(x2 - x) < delta
            x_min = x;
            return
        else
            % Se reduce el intervalo
            if x2 <= x
                x1 = x2; x2 = x;
            else
                x3 = x2; x2 = x;
            end
        end
    end
    x_min = 0;

end
```

Los resultados obtenidos una vez se pasa el algoritmo a todas las funciones son los siguientes:

```
% Parámetros de la interpolación cuadrática
delta = 1e-6;  % Tolerancia (delta)

% Ejecución de las interpolaciones para cada función
tic;
[x1_incu, iter1_incu] = incu(f1, intervalo_f1(1),
(intervalo_f1(1)+intervalo_f1(2))/2, intervalo_f1(2), delta);
tiempo1_incu = toc * 1000;

tic;
[x2_incu, iter2_incu] = incu(f2, intervalo_f2(1),
(intervalo_f2(1)+intervalo_f2(2))/2, intervalo_f2(2), delta);
tiempo2_incu = toc * 1000;

tic;
[x3_incu, iter3_incu] = incu(f3, intervalo_f3(1),
(intervalo_f3(1)+intervalo_f3(2))/2, intervalo_f3(2), delta);
tiempo3_incu = toc * 1000;

tic;
[x4_incu, iter4_incu] = incu(f4, intervalo_f4(1),
(intervalo_f4(1)+intervalo_f4(2))/2, intervalo_f4(2), delta);
tiempo4_incu = toc * 1000;
```

```
figure (4);
f1_incu = f1(x1_incu);
subplot(2, 2, 1), plot(x1_values, y1_values, 'b-', 'LineWidth', 2);
hold on;
subplot(2, 2, 1), xline(intervalo_f1(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 1), xline(intervalo_f1(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 1), plot(x1_incu, f1_incu, 'r*', 'MarkerSize', 10, 'MarkerFaceColor',
'r');  % Punto óptimo
legend('f1(x)', 'a', 'b', 'min');
title('f1(x) = x^6 - 7*x^3 + 7');
xlim([0, 3]);
ylim([-30, 30]);
xlabel('x');
ylabel('f(x)');
grid on;

f2_incu = f2(x2_incu);
subplot(2, 2, 2), plot(x2_values, y2_values, 'b-', 'LineWidth', 2);
hold on;
subplot(2, 2, 2), xline(intervalo_f2(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 2), xline(intervalo_f2(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 2), plot(x2_incu, f2_incu, 'r*', 'MarkerSize', 10, 'MarkerFaceColor',
'r');  % Punto óptimo
legend('f2(x)', 'a', 'b',  'min');
title('f2(x) = ln(x) - sin(x)');
xlim([6, 9]);
xlabel('x');
ylabel('f(x)');
grid on;

f3_incu = f3(x3_incu);
subplot(2, 2, 3), plot(x3_values, y3_values, 'b-', 'LineWidth', 2);
hold on;
subplot(2, 2, 3), xline(intervalo_f3(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 3), xline(intervalo_f3(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 3), plot(x3_incu, f3_incu, 'r*', 'MarkerSize', 10, 'MarkerFaceColor',
'r');  % Punto óptimo
legend('f3(x)', 'a', 'b', 'min');
title('f3(x) = e^{cos(x)} - x', 'Interpreter', 'tex');
xlim([3.25, 6.25]);
xlabel('x');
ylabel('f(x)');
grid on;
```

```
f4_incu = f4(x4_incu);
subplot(2, 2, 4), plot(x4_values, y4_values, 'b-', 'LineWidth', 2);
hold on;
subplot(2, 2, 4), xline(intervalo_f4(1), '--r', 'a', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 4), xline(intervalo_f4(2), '--g', 'b', 'LabelOrientation',
'horizontal', 'LineWidth', 1.5);
subplot(2, 2, 4), plot(x4_incu, f4_incu, 'r*', 'MarkerSize', 10, 'MarkerFaceColor',
'r');  % Punto óptimo
legend('f4(x)', 'a', 'b', 'min');
title('f4(x) = funcionrr');
xlim([1, 4]);
xlabel('x');
ylabel('f(x)');
grid on;

sgtitle('Interpolación cuadrática: Mínimos encontrados.');
```



Se aprecia a simple vista cómo se encuentra el punto mínimo en cada una de las funciones de manera precisa.

En la siguiente tabla, se detallan los valores obtenidos para el valor x_min, para el valor que cobra la función, las iteraciones necesarias, así como el tiempo de ejecución:

```
% Definir datos para las columnas
funciones = {'f1(x)'; 'f2(x)'; 'f3(x)'; 'f4(x)'};
x_min = [x1_incu; x2_incu; x3_incu; x4_incu];
```

```
f_x_min = [f1(x1_incu); f2(x2_incu); f3(x3_incu); f4(x4_incu)];
iter = [iter1_incu; iter2_incu; iter3_incu; iter4_incu];
times = [tiempo1_incu; tiempo2_incu; tiempo3_incu; tiempo4_incu];

% Crear una tabla
T = table(funciones, x_min, f_x_min, iter, times, 'VariableNames', {'Funciones', 'x
min', 'f(x min)', 'Iteraciones', 'Tiempos (msec)'});

% Mostrar la tabla en el Live Editor
disp(T);
```

Funciones	x min	f(x min)	Iteraciones	Tiempos (msec)
{'f1(x)'}	1.5183	-5.25	9	2.281
{'f2(x)'}	7.7242	1.0528	5	2.1798
{'f3(x)'}	4.7124	-3.7124	5	1.6106
{'f4(x)'}	2.5574	0.97258	7	96.802

Ya se puede comprobar cómo los valores mínimos obtenidos coinciden con el método de búsqueda dicotómica. Al final de la Actividad se presenta una tabla comparativa entre todos ellos.

En relación con las iteraciones, es interesante notar como ahora hay dos funciones que ocupan 5 iteraciones mientras la función 4 ocupa 7 y la función 1 ocupa 9. De manera intuitiva se puede identificar que en aquellas funciones donde el mínimo tiene menos pendiente, se hace más complicado para el método cerrar el intervalo y por tanto tarda más iteraciones.

En cuanto al tiempo de ejecución, se observa cómo para las tres primeras funciones el valor ronda entre 1-2 milisegundos. Mientras que para la función 4, el tiempo se dispara hasta casi los 100 milisegundos. Esto se debe de nuevo a que las evaluciones que necesita hacer la función son en esta última función mucho más costosas por ser una función numérica un tanto compleja.

2.3 Método de Newton (new.*m*)

La interpolación cuadrática es un método iterativo y eficiente para encontrar el mínimo de una función suave dentro de un intervalo dado. Este método asume que la función puede ser aproximada localmente por una parábola, y utiliza tres puntos del intervalo para construir esta aproximación cuadrática. A partir de esta parábola, se estima el punto donde ocurre el mínimo, lo que permite ajustar el intervalo para acercarse progresivamente al mínimo real. El proceso se repite iterativamente hasta alcanzar una precisión deseada, re

```
function [x_min, iter] = new(f, x, delta)
    % Método de interpolación cuadrática
    % Entradas:
    %   f: función objetivo
    %   x: punto inicial
    %   delta: tolerancia para la convergencia
    %
    % Salidas:
    %   x_min: punto donde se alcanza el mínimo
    %   iter: número de iteraciones realizadas

    % Implementación del método
```

```
    % Inicialización de las variables
    iter = 0;  % Contador de iteraciones
    max_iter = 100;  % Número máximo de iteraciones

    while iter < max_iter
        iter = iter + 1;

        % Calcular el valor mínimo del polinomio interpolador
        f1 = f(x1); f2 = f(x2); f3 = f(x3);
        num = ((x2^2-x3^2)*f1 + (x3^2-x1^2)*f2 + (x1^2-x2^2)*f3);
        den = 2*((x3-x2)*f1 + (x1-x3)*f2 + (x2-x1)*f3);
        x = -num/den;

        % Se cumple el criterio de parada?
        if abs(x2 - x) < delta
            x_min = x;
            return
        else
            % Se reduce el intervalo
            if x2 <= x
                x1 = x2; x2 = x;
            else
                x3 = x2; x2 = x;
            end
        end
    end
    x_min = 0;

end
```

finando el intervalo en cada paso y mejorando la aproximación del mínimo. El código implementado ha sido el siguiente:

X. Anexo. Función numérica

La función numérica

```
function f = funcionrr(x_vec)
    % Inicializa un vector de salida
    f = zeros(size(x_vec));

    % Recorre cada valor del vector de entrada
```

```
    for i = 1:length(x_vec)
        x = x_vec(i); % Toma el valor actual
        xini=[x 0];
        t=[0 2*pi];

        options=odeset('RelTol',1e-13,'AbsTol',1e-13);

        [ts, xs]=ode45(@campo_pendulo,t,xini,options);

        f(i)=xs(end,1);
    end

end


function px = campo_pendulo(t,x)

    px(1) = x(2);
    px(2) = sin(x(1)) + 0.5*sin(t);
    px=px';
end
```