Taller 1 - Análisis Numérico y Computación Científica

Santiago Hoyos Ortiz, Winston Rafael Pernett González

Escuela de Ingeniería, Ciencia y Tecnología, Universidad del Rosario

Agosto 23 de 2023

1. Punto 1

Emplear los métodos de solución de ecuaciones de una variable para dar solución a las siguientes ecuaciones:

- $\epsilon^x - 4 + x = 0$
- $x - 0{,}2sin(x) - 0{,}5 = 0$
- $\epsilon^{\frac{x}{2}} - x^2 - 3x = 0$
- $\epsilon^x cos(x) - x^2 + 3x = 0$
- $0{,}5x^3 + x^2 - 2x - 5 = 0$
- $\epsilon^x - 4x^2 - 8x = 0$

Evidenciar el proceso, comparar la velocidad de convergencia entre los diferentes métodos para cada uno de los ejercicios y emplear un criterio de parada de $\epsilon < 10^{-4}$.

```
% Criterio de parada
epsilon = 1e-4;

% Definicion de las ecuaciones y funciones
equations = {
    @(x) exp(x) - 4 + x,
    @(x) x - 0.2*sin(x) - 0.5,
    @(x) exp(x*0.5) - x^2 - 3*x,
    @(x) exp(x)*cos(x) - x^2 + 3*x,
    @(x) 0.5*x^3 + x^2 - 2*x - 5,
    @(x) exp(x) - 4*x^2 - 8*x
};

% Intervalo
interval = [1, 2];
```

1.1. Método de Bisección

```
methods = {'Biseccion'};

for eq_num = 1:length(equations)
    f = equations{eq_num};
    a = interval(1);
    b = interval(2);

    for method = methods
        fprintf('=================\n');
        fprintf('%s: Ecuacion %d\n', char(method), eq_num);

        tic;
        [solution, iterations] = bisection(f, a, b, epsilon);
        time = toc;

        fprintf('Solucion: %f, Iteraciones: %d, Tiempo: %f segundos\n\n',
            solution, iterations, time);
    end
end

function [solution, iterations] = bisection(f, a, b, epsilon) % Funcion de
    biseccion
    iterations = 0;

    while abs((a - b)/a) > epsilon
        c = (a + b)/2;
        if f(c) == 0
            break;
        elseif f(c)*f(a) < 0
            b = c;
        else
            a = c;
        end
        iterations = iterations + 1;
    end

    solution = (a + b)/2;
end
```

```
=================
Bisección: Ecuación 1
Solución: 1.073700, Iteraciones: 14, Tiempo: 0.000981 segundos
=================
Bisección: Ecuación 2
Solución: 1.999939, Iteraciones: 13, Tiempo: 0.000976 segundos
=================
Bisección: Ecuación 3
Solución: 1.999939, Iteraciones: 13, Tiempo: 0.001010 segundos
=================
Bisección: Ecuación 4
Solución: 1.892273, Iteraciones: 13, Tiempo: 0.001653 segundos
=================
Bisección: Ecuación 5
Solución: 1.999939, Iteraciones: 13, Tiempo: 0.000941 segundos
=================
Bisección: Ecuación 6
Solución: 1.999939, Iteraciones: 13, Tiempo: 0.001220 segundos
```

1.2. Método de falsa posición

```
methods = {'Falsa_posicion'};

function [solution, iterations] = false_position(f, a, b, epsilon)
    iterations = 0;
    f_a = f(a);
    f_b = f(b);

    while abs((a - b)/a) > epsilon
        c = b - (f_b * (b - a)) / (f_b - f_a);
        f_c = f(c);

        if abs(f_c) < epsilon
            break;
        end

        if f_c * f_a < 0
            b = c;
            f_b = f_c;
        else
            a = c;
            f_a = f_c;
        end

        iterations = iterations + 1;
    end

    solution = c;
end
```

```
=================
Falsa posición: Ecuación 1
Solución: 1.073720, Iteraciones: 7, Tiempo: 0.000018 segundos

=================
Falsa posición: Ecuación 2
Solución: 0.615555, Iteraciones: 3, Tiempo: 0.000005 segundos

=================
Falsa posición: Ecuación 3
Solución: 0.356038, Iteraciones: 8, Tiempo: 0.000003 segundos

=================
Falsa posición: Ecuación 4
Solución: 1.892251, Iteraciones: 4, Tiempo: 0.000685 segundos

=================
Falsa posición: Ecuación 5
Solución: 2.117934, Iteraciones: 4, Tiempo: 0.000086 segundos

=================
Falsa posición: Ecuación 6
Solución: 0.133955, Iteraciones: 11, Tiempo: 0.000006 segundos
```

1.3. Punto Fijo

```
max_iterations = 1000;

tic;
% Evaluar la funcion en el punto medio del intervalo como valor inicial
x = (a + b) / 2;

for iterations = 1:max_iterations
    x_next = g(x);
    if abs(x_next - x) < epsilon
        root = x_next;
        time = toc;
        fprintf("Raiz aproximada encontrada: x = %.6f\n", root);
        fprintf("Numero de iteraciones: %d\n", iterations);
        fprintf("Tiempo: %.6f segundos\n", time);
        break;
    end
    x = x_next;
end

if iterations == max_iterations
    fprintf("No se alcanzo convergencia en el numero maximo de iteraciones.\n
        ");
end
```

1.3.1. Ecuación 1

```
% Definicion de la funcion y su funcion g(x) para iteracion de punto fijo
f = @(x) exp(x) - 4 + x;
g = @(x) log(4 - x);

% Intervalo inicial
a = 0;
b = 2;
```

```
Raíz aproximada encontrada: x = 1.073715
Número de iteraciones: 8
Tiempo: 0.002671 segundos
```

1.3.2. Ecuación 2

```
% Definicion de la funcion y su funcion g(x) para iteracion de punto fijo
f = @(x) x - 0.2*sin(x) - 0.5;
g = @(x) 0.2*sin(x) + 0.5;

% Intervalo inicial
a = 0;
b = 1;
```

```
Raíz aproximada encontrada: x = 0.615474
Número de iteraciones: 6
Tiempo: 0.002188 segundos
```

1.3.3. Ecuación 3

```
% Definicion de la funcion y su funcion g(x) para iteracion de punto fijo
f = @(x) exp(x/2) - x^2 - 3*x;
g = @(x) (exp(x/2) - x^2)/3;

% Intervalo inicial
a = 0;
b = 1;
```

```
Raíz aproximada encontrada: x = 0.356029
Número de iteraciones: 4
Tiempo: 0.000158 segundos
```

1.3.4. Ecuación 4

```
% Definicion de la funcion y su funcion g(x) para iteracion de punto fijo
f = @(x) exp(x)*cos(x) - x^2 + 3*x;
g = @(x) acos((x^2 - 3*x)/exp(x));

% Intervalo inicial
a = 0;
b = 2;
```

```
Raíz aproximada encontrada: x = 1.892240
Número de iteraciones: 14
Tiempo: 0.000209 segundos
```

1.3.5. Ecuación 5

```
% Definicion de la funcion y su funcion g(x) para iteracion de punto fijo
f = @(x) 0.5*x^3 + x^2 - 2*x - 5;
g = @(x) sqrt(2*x + 5 - 0.5*x^2);

% Intervalo inicial
a = 1;
b = 2;
```

```
Raíz aproximada encontrada: x = 2.610320
Número de iteraciones: 5
Tiempo: 0.000162 segundos
```

1.3.6. Ecuación 6

```
% Definicion de la funcion y su funcion g(x) para iteracion de punto fijo
f = @(x) exp(x) - 4*x^2 - 8*x;
g = @(x) log(4*x^2 + 8*x);

% Intervalo inicial
a = 1;
b = 2;
```

```
Raíz aproximada encontrada: x = 4.910802
Número de iteraciones: 12
Tiempo: 0.000157 segundos
```

1.4. Método de Newton - Raphson

```
 derivatives = {
@(x) exp(x) + 1,
@(x) 1 - 0.2*cos(x),
@(x) 0.5*exp(0.5*x) - 2*x - 3,
@(x) exp(x)*cos(x) - 2*x + 3,
@(x) 1.5*x^2 + 2*x - 2,
@(x) exp(x) - 8*x };

methods = {'Newton-Raphson'};

for eq_num = 1:length(equations)
    f = equations{eq_num};
    df = derivatives{eq_num};

    for method = methods
        fprintf('=================\n');
        fprintf('%s: Ecuacion %d\n', char(method), eq_num);

        tic;
        [solution, iterations] = newton_raphson(f, df, epsilon);
        time = toc;

        fprintf('Solucion: %f, Iteraciones: %d, Tiempo: %f segundos\n\n',
            solution, iterations, time);
    end
end

function [solution, iterations] = newton_raphson(f, df, epsilon)
    p0 = 1.5; % Valor inicial
    iterations = 0;

    while true
        p1 = p0 - f(p0) / df(p0);

        if abs((p1 - p0)/p1)/ < epsilon
            break;
        end
        p0 = p1;
        iterations = iterations + 1;
    end

    solution = p1;
end
```

```
=================
Newton-Raphson: Ecuación 1
Solución: 1.073729, Iteraciones: 3, Tiempo: 0.001286 segundos

=================
Newton-Raphson: Ecuación 2
Solución: 0.615468, Iteraciones: 3, Tiempo: 0.002568 segundos

=================
Newton-Raphson: Ecuación 3
Solución: 0.356029, Iteraciones: 3, Tiempo: 0.004847 segundos

=================
Newton-Raphson: Ecuación 4
Solución: -0.236946, Iteraciones: 6, Tiempo: 0.001605 segundos

=================
Newton-Raphson: Ecuación 5
Solución: 2.117942, Iteraciones: 4, Tiempo: 0.000091 segundos

=================
Newton-Raphson: Ecuación 6
Solución: -2.016377, Iteraciones: 13, Tiempo: 0.000013 segundos
```

1.5. Método de Secante

```
function [solution, iterations] = secant_method(f, epsilon)
    p0 = 1; % Primer punto inicial
    p1 = 2; % Segundo punto inicial

    iterations = 0;

    while true
        f_p0 = f(p0);
        f_p1 = f(p1);

        p2 = p1 - (f_p1 * (p1 - p0)) / (f_p1 - f_p0);

        if abs(p2 - p1) < epsilon
            break;
        end

        p0 = p1;
        p1 = p2;
        iterations = iterations + 1;
    end

    solution = p2;
end
```

```
=================
Secante: Ecuación 1
Solución: 1.073729, Iteraciones: 3, Tiempo: 0.005051 segundos

=================
Secante: Ecuación 2
Solución: 0.615468, Iteraciones: 3, Tiempo: 0.004974 segundos

=================
Secante: Ecuación 3
Solución: 0.356029, Iteraciones: 4, Tiempo: 0.001317 segundos

=================
Secante: Ecuación 4
Solución: 1.892257, Iteraciones: 4, Tiempo: 0.003178 segundos

=================
Secante: Ecuación 5
Solución: 2.117942, Iteraciones: 4, Tiempo: 0.002672 segundos

=================
Secante: Ecuación 6
Solución: 0.133946, Iteraciones: 5, Tiempo: 0.001636 segundos
```

2. Punto 2

La velocidad de una paracaidista se define como:

$$v = \frac{gm}{c} \cdot \left(1 - e^{-\left(\frac{c}{m}\right)t}\right)$$

Teniendo presente que el valor aproximado de la gravedad es de 9.81$\frac{m}{s^2}$, emplee el método de bisección y de falsa posición, con un error inferior a $\epsilon_{rr} \leq 0{,}02\,\%$ para:

- **Calcular el valor de la masa que hace que el paracaidista tenga una velocidad de $v = 36\frac{m}{s}$, con un coeficiente de resistencia $c = 15\frac{kg}{s}$ en un tiempo $t = 10s$**

```
% Datos del problema
v = 36; % m/s
c = 15; % kg/s
t = 10; % s
g = 9.81; % m/s^2

% Funcion de velocidad en funcion de la masa
f = @(m) (g*m/c) * (1 - exp(-c*t/m)) - v;

% Intervalo inicial para el metodo de biseccion
a = 0.01; % Valor minimo posible para la masa
b = 100; % Valor maximo posible para la masa

% Criterio de parada
epsilon = 0.0002; % 0.02%

% Aplicar metodo de biseccion
m_bisection = bisection_method(f, a, b, epsilon);
disp(['Masa (metodo de Biseccion): ', num2str(m_bisection), ' kg']);

% Aplicar metodo de falsa posicion
m_false_position = false_position_method(f, a, b, epsilon);
disp(['Masa (Metodo de Falsa Posicion): ', num2str(m_false_position), ' kg']);
```

```
>> Punto2
Masa (Método de Bisección): 59.9619 kg
Masa (Método de Falsa Posición): 59.9596 kg
```

- **Calcular el valor del coeficiente de resistencia para que un paracaidista de 8kg tenga una velocidad de $36\frac{m}{s}$, después de 4s de caída libre.**

```
% Datos del problema
m = 8; % kg
v = 36; % m/s
t = 4; % s
g = 9.81; % m/s^2

% Funcion en funcion del coeficiente de resistencia c
f = @(c) (g*m/c) * (1 - exp(-c*t/m)) - v;

% Intervalo inicial para el metodo de biseccion
a = 0.01; % Valor minimo posible para c
b = 100; % Valor maximo posible para c

% Criterio de parada
epsilon = 0.0002; % 0.02%

% Aplicar metodo de biseccion
c_bisection = bisection_method(f, a, b, epsilon);
disp(['Coeficiente de Resistencia (Metodo de Biseccion): ', num2str(
    c_bisection), ' kg/s']);

% Aplicar metodo de falsa posicion
c_false_position = false_position_method(f, a, b, epsilon);
disp(['Coeficiente de Resistencia (Metodo de Falsa Posicion): ', num2str(
    c_false_position), ' kg/s']);
```

```
>> Punto2b
Coeficiente de Resistencia (Método de Bisección): 0.34983 kg/s
Coeficiente de Resistencia (Método de Falsa Posición): 0.34981 kg/s
```

3. Punto 3

Encuentre el máximo de la siguiente función con un error inferior al $\varepsilon_{rr} \leq 0{,}05\,\%$:

$$f(x) = -2x^6 - 1{,}5x^4 + 10x + 2$$

Para encontrar el máximo tenemos que encontrar el valor de x cuando $f'(x) = 0$, así encontramos la derivada de $f(x)$:

$$f'(x) = -12x^5 - 6x^3 + 10$$

$$-12x^5 - 6x^3 + 10 = 0$$

a) **Emplee el método de iteración de punto fijo.**

Transformando la función tenemos: $x = \sqrt[5]{\frac{-6x^3+10}{12}}$

```
Iteración: 1, Xn: 1.000000, Xn+1: 0.802742, Error = 0.197258
Iteración: 2, Xn: 0.802742, Xn+1: 0.895132, Error = 0.115094
Iteración: 3, Xn: 0.895132, Xn+1: 0.861562, Error = 0.037502
```

Así tenemos que un punto critico sera $x = 0{,}861562$. Para evaluar si es máximo o mínimo absoluto en el intervalo $[0{,}5, 1{,}5]$ tenemos que evaluar en los extremos y en el valor de x encontrado, que en este caso se encuentra en el intervalo.

$$f(0{,}5) = 6{,}875000$$

$$f(0{,}861562) = 8{,}971138$$

$$f(1{,}5) = -13{,}375000$$

Por lo tanto $x = 0{,}861562$ será un maximo de la función: $f(x) = -2x^6 - 1{,}5x^4 + 10x + 2$.

b) **Emplee el método de Newton - Raphson iniciando en $x_i = 1$.**

```
Iteración: 1, Xn: 1.000000, Xn+1: 0.897436, Error = 0.102564
Iteración: 2, Xn: 0.897436, Xn+1: 0.872682, Error = 0.027582
```

Para este caso obtenemos $x = 0{,}872682$. Ahora si evaluamos de nuevo este valor en $f(x)$ tenemos:

$$f(0{,}872682) = 8{,}973410$$

Por lo tanto $x = 0{,}872682$ corresponde a un máximo.

c) **Emplee el método de la secante a partir de $x_{i-1} = 0$ y $x_i = 1$.**

```
Iteración: 1, a: 0.000000, b: 1.000000, c: 0.555556 ,  Error = 0.800000
Iteración: 2, a: 1.000000, b: 0.555556, c: 0.782350 ,  Error = 0.289889
Iteración: 3, a: 0.555556, b: 0.782350, c: 0.955564 ,  Error = 0.181269
Iteración: 4, a: 0.782350, b: 0.955564, c: 0.856738 ,  Error = 0.115351
Iteración: 5, a: 0.955564, b: 0.856738, c: 0.869138 ,  Error = 0.014267
```

Para este caso obtenemos $x = 0{,}869138$. Ahora si evaluamos de nuevo este valor en $f(x)$ tenemos:

$$f(0{,}869138) = 8{,}973325$$

Por lo tanto $x = 0{,}869138$ corresponde a un máximo en f(x).

d) **Independiente de la convergencia, seleccione la técnica más adecuada para este problema. Justifique su respuesta.**

Para este problema la más adecuada sería el metodo secante, pues con este metodo no es necesario conocer donde está ubicada la raiz.

4. Punto 4

Para la siguiente tabla de datos:

x	1.6	2	2.5	3.2	4	4.5
$f(x)$	2	8	14	15	8	2

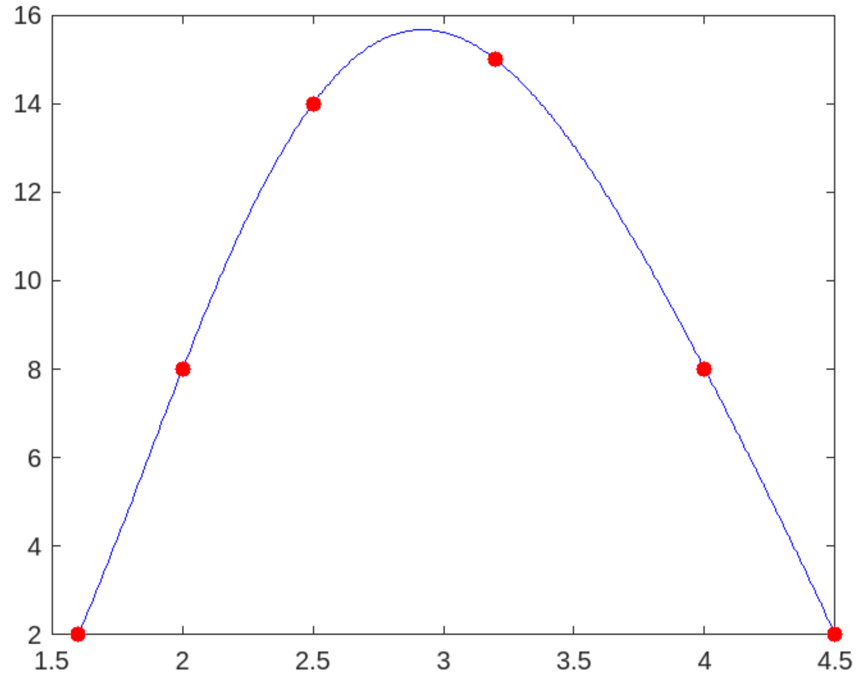
a) **Encuentre el polinomio interpolador de Lagrange que pasa por los puntos de la tabla de datos.**

```
syms x

puntos = [1.6,2,2.5,3.2,4,4.5];
imagen = [2, 8, 14, 15, 8, 2];
Pn = 0;
for i=1:length(puntos)
    L = 1;

    if i ==1
        copia = puntos(i+1:end);
    else
        copia = union(puntos(1:i-1),puntos(i+1:end));
    end

    for j = 1:length(copia)
        L = L*(x-copia(j))/(puntos(i)-copia(j));
    end

    Pn = Pn + (imagen(i)*L);
end


x_i = linspace(puntos(1),puntos(end),500);
resultado = [];
for i=1:length(x_i)
    resultado(i) = round(double(subs(Pn,x,x_i(i))),6);

end

plot(x_i,resultado,'blue')
hold on
scatter(puntos,imagen,'filled','red')

fprintf("f(2.8) = %f", double(subs(Pn,x,2.8)))
```

Utilizando el codigo mostrado, encontramos que el polinomio de lagrange para estos puntos es el siguiente:

$$P_5(x) = -\frac{485\,x^5}{1008} + \frac{2771\,x^4}{336} - \frac{1083949\,x^3}{20160} + \frac{537049\,x^2}{3360} - \frac{515399\,x}{2520} + \frac{639}{7}$$

b) Grafique la tabla de datos y el polinomio interpolador obtenido.



c) Calcule el valor de $f(2{,}8)$.

Con el codigo previamente utilizado, obtenemos que $f(2{,}8) = 15{,}534914$

5. Punto 5

Para las siguientes tabla de datos:

1. **Teniendo en cuenta la siguiente tabla de datos:**

x	3.0	4.5	7.0	9.0
$f(x)$	2.5	1.0	2.5	0.5

a) Encuentre el spline cúbico que pase por los puntos de la tabla de datos.

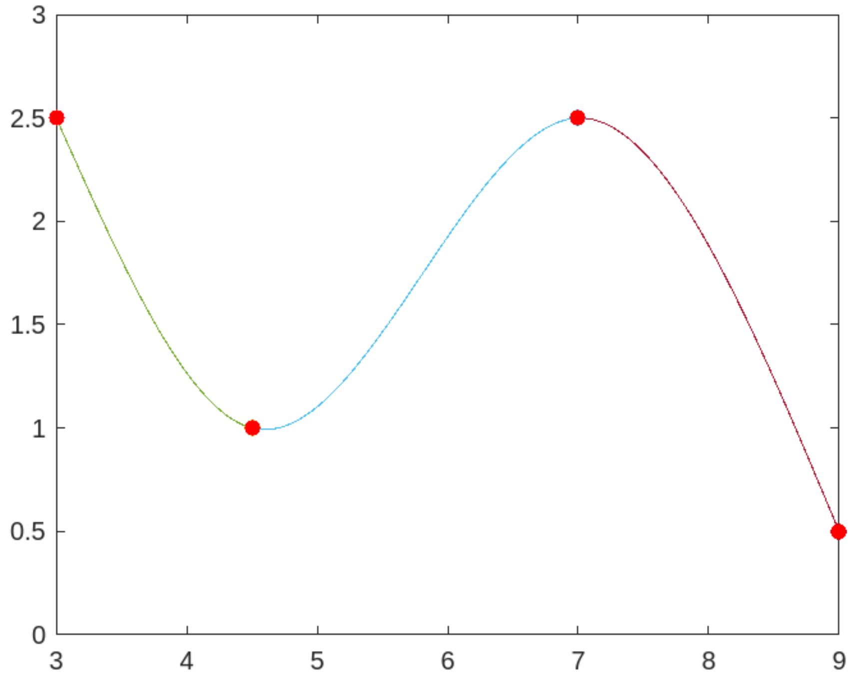
```
clear
syms x
x_i = [3, 4.5, 7, 9];
y_i = [2.5, 1.0, 2.5, 0.5];

n = length(x_i);
s = sym('s',[1 n-1]);
df = sym('d',[1 n-1]);
dd= sym('dd',[1 n-1]);
eqn = sym('eqn',[1 4*(n-1)]);
iter = 1;
variables = sym('a',[4*(n-1) 1]);
for i=1:n-1
    a = sym("a"+i);
```

```
        b = sym("b"+i);
        c = sym("c"+i);
        d = sym("d"+i);
        variables(i) = a;
        variables(i+(n-1)) = b;
        variables(i+2*(n-1)) = c;
        variables(i+3*(n-1)) = d;
        s(i) = a + b*(x-x_i(i)) + c*(x-x_i(i))^2 + d*(x-x_i(i))^3;
        df(i) = b + 2*c*(x-x_i(i)) + 3*d*(x-x_i(i))^2;
        dd(i) = 2*c + 6*d*(x-x_i(i));

        if i == n-1
            eqn(iter) = subs(s(i),x,x_i(i)) == y_i(i);
            iter = iter+1;
            eqn(iter) = subs(s(i),x,x_i(i+1)) == y_i(i+1);
        else
            eqn(iter) = subs(s(i),x,x_i(i)) == y_i(i);
        end
        iter = iter+1;
    end


    for i = 1:n-2
        eqn(iter) = subs(s(i),x,x_i(i+1)) == subs(s(i+1),x,x_i(i+1));
        iter = iter+1;
        eqn(iter) = subs(df(i),x,x_i(i+1)) == subs(df(i+1),x,x_i(i+1));
        iter = iter+1;
        eqn(iter) = subs(dd(i),x,x_i(i+1)) == subs(dd(i+1),x,x_i(i+1));
        iter = iter+1;
    end

    eqn(iter) = subs(dd(1),x,x_i(1)) == 0;
    iter = iter+1;
    eqn(iter) = subs(dd(n-1),x,x_i(n)) == 0;

    [A,b] = equationsToMatrix(eqn);

    X = linsolve(A,b);
    puntos = [];
    Y = [];
    for i = 1:n-1
        s(i) = subs(s(i),variables,X);
        eval = linspace(x_i(i),x_i(i+1),50);
        f_eval = subs(s(i),x,eval);
        plot(eval,f_eval)
        plot(x_i(i),y_i(i),'o')
        plot(x_i(i+1),y_i(i+1),'o')
        ylim([0,3])
        hold on
    end
```

A partir del codigo implementado obtenemos los siguientes splines:

$$S_0(x) = 0{,}18657 * (x-3)^3 - 1{,}4198 * x + 6{,}7593$$
$$S_1(x) = 0{,}83954 * (x-4{,}5)^2 - 0{,}16046 * x - 0{,}21414 * (x-4{,}5)^3 + 1{,}7221$$
$$S_2(x) = 0{,}022053 * x - 0{,}76654 * (x-7{,}0)^2 + 0{,}12776 * (x-7{,}0)^3 + 2{,}3456$$

b) **Grafique la tabla de datos y el spline obtenido.**



c) **Utilice los resultados para estimar el valor en $x = 5$.**
En este caso el valor se encuentre en el intervalo $[4{,}5, 7]$, por lo cual evaluamos $x = 5$ en S_1. Así tenemos:

$$S_1(5) = 0{,}83954 * (5 - 4{,}5)^2 - 0{,}16046 * 5 - 0{,}21414 * (5 - 4{,}5)^3 + 1{,}7221$$
$$S_1(5) = 1{,}1029$$

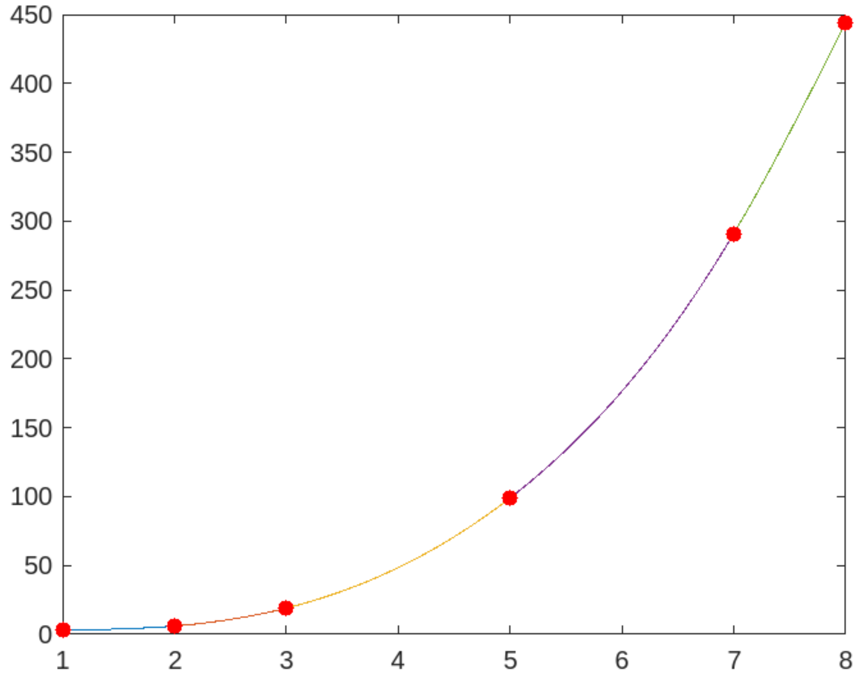
2. **Teniendo en cuenta la siguiente tabla de datos:**

x	1	2	3	5	7	8
$f(x)$	3	6	19	99	291	444

a) **Encuentre el spline cúbico que pase por los puntos de la tabla de datos.**
Para encontrar el spline, utilizamos el codigo previamente creado y le cambiamos los puntos. Tenemos como resultado:

$$S_0(x) = 1{,}1921 * x + 1{,}8079 * (x - 1{,}0)^3 + 1{,}8079$$
$$S_1(x) = 6{,}6157 * x + 5{,}4236 * (x - 2{,}0)^2 + 0{,}9607 * (x - 2{,}0)^3 - 7{,}2314$$
$$S_2(x) = 20{,}345 * x + 8{,}3057 * (x - 3{,}0)^2 + 0{,}76092 * (x - 3{,}0)^3 - 42{,}035$$
$$S_3(x) = 62{,}699 * x + 12{,}871 * (x - 5{,}0)^2 + 1{,}8897 * (x - 5{,}0)^3 - 214{,}49$$
$$S_4(x) = 136{,}86 * x + 24{,}21 * (x - 7{,}0)^2 - 8{,}0699 * (x - 7{,}0)^3 - 667{,}02$$

b) **Grafique la tabla de datos y el spline obtenido.**



c) **Utilice los resultados para estimar el valor en $x = 4$ y $x = 2{,}25$.**
Para $x = 2{,}25$ está en el rango $[2, 3]$ por lo cual utilizamos $S_1(x)$.

$$S_1(2{,}25) = 6{,}6157 * 2{,}25 + 5{,}4236 * (2{,}25 - 2{,}0)^2 + 0{,}9607 * (2{,}25 - 2{,}0)^3 - 7{,}2314$$

$$S_1(2{,}25) = 8{,}0079$$

Para $x = 4$ está en el rango $[3, 5]$ por lo cual utilizamos $S_2(x)$.

$$S_2(4) = 20{,}345 * 4 + 8{,}3057 * (4 - 3{,}0)^2 + 0{,}76092 * (4 - 3{,}0)^3 - 42{,}035$$

$$S_2(x) = 48{,}412$$

6. Punto 6

Desarrolle un código que permita calcular el valor de intermedio en una tabla de datos a partir de Polinomios de Lagrange. El código debe recibir dos arreglos unidimensionales que representan x y $f(x)$ y un valor que se desee estimar a partir de la información contenida en dichos arreglos. El código debe encontrar el intervalo en donde se localiza el valor que se desea estimar y luego aproximarlo mediante polinomios cúbicos de Lagrange. Para los intervalos primero y último emplee polinomios cuadráticos y para valores fuera del rango de datos indique la presencia de un error en la información suministrada. Una vez realizado el código puede probarlo con $f(x) = \ln x$ siendo $x = 1, 2, 3, \ldots, 10$.

```
function estimacionY = LagrangeIntervalos(u,v,z)
syms x
if z < u(1) || z > u(length(u))
    error("El dato ingresado no esta en el intervalo.");
end

intervalo = find(u > z, 1');

if intervalo == 2
```

```
        Pn = Polinomios_lagrange([u(1),u(2),u(3)],[v(1),v(2),v(3)]);
        estimacionY = vpa(subs(Pn,x,z),6);

    elseif intervalo == length(u)
        Pn = Polinomios_lagrange([u(length(v)-2),u(length(v)-1),u(length(v))
            ],[v(length(u)-2),v(length(u)-1),v(length(u))]);
        estimacionY = vpa(subs(Pn,x,z),6);
    else
        Pn = Polinomios_lagrange([u(intervalo-2),u(intervalo-1),u(intervalo),
            u(intervalo+1)],[v(intervalo-2),v(intervalo-1),v(intervalo),v(
            intervalo+1)]);
        estimacionY = vpa(subs(Pn,x,z),6);
    end
```

Probamos el codigo con la función $f(x) = lnx$.

```
x = [1 2 3 4 5 6 7 8 9 10];
f_x = [log(1) log(2) log(3) log(4) log(5) log(6) log(7) log(8) log(9) log
    (10)];

y = LagrangeIntervalos(x,f_x,9.2);

fprintf("P(9.2)= %f",y);
```

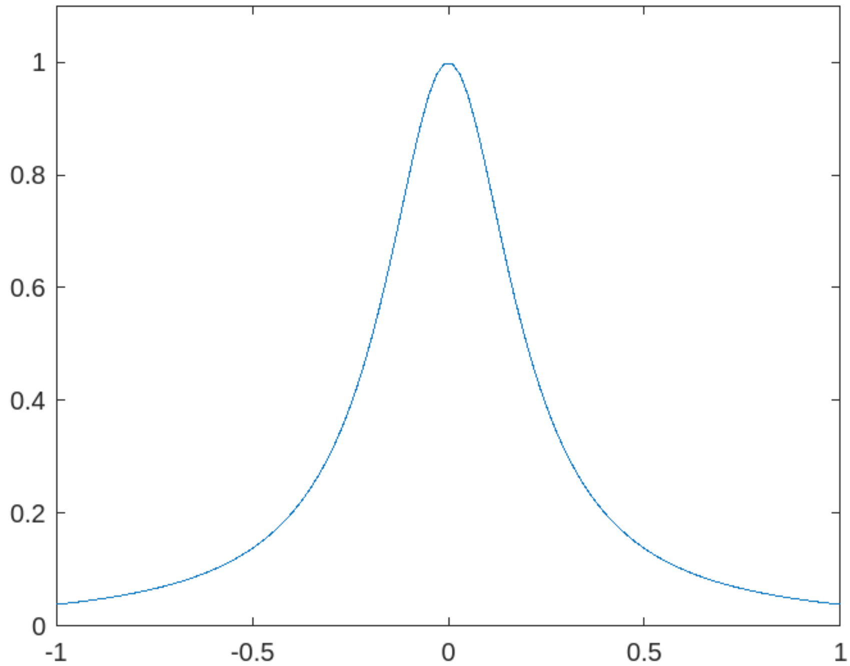
Tenemos como resultado $P(9{,}2) = 2{,}219290$.

7. Punto 7

$$f(x) = \frac{1}{1 + 25x^2}$$

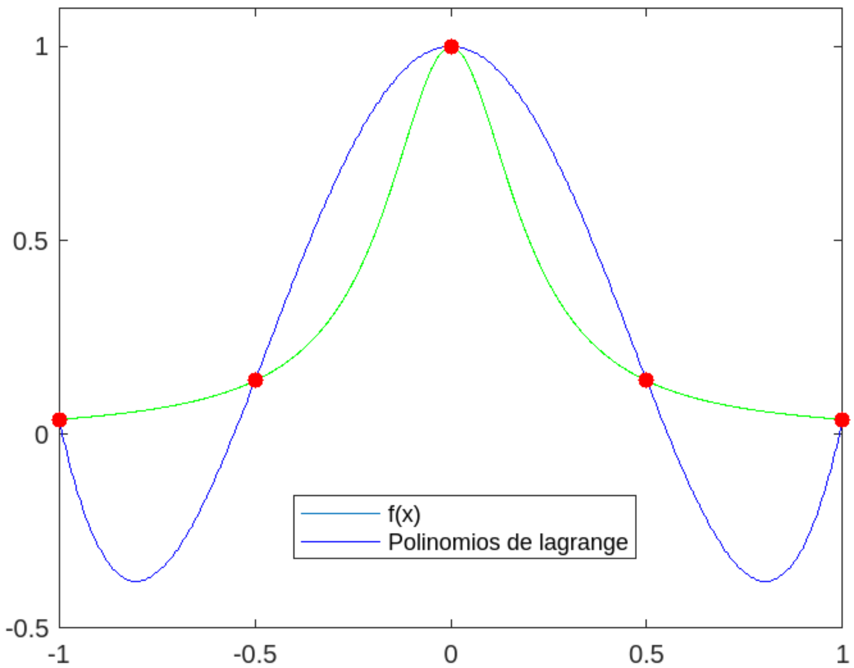
a) **Grafique la función en el intervalo de $x = -1$ a 1.**

```
syms x
f = 1/(1+25*x^2);
intervalo = linspace(-1,1,100);
imagen = subs(f,x,intervalo);

plot(intervalo,imagen)
ylim([0,1.1])
```



b) **Obtenga y grafique el polinomio de Lagrange usando los valores de la función equiespaciados** $x = \begin{bmatrix} -1 & -0{,}5 & 0 & 0{,}5 & 1 \end{bmatrix}$

$$P_4(x) = \frac{(1250 * x^4)}{377} - \frac{3225 * x^2}{754} + 1$$



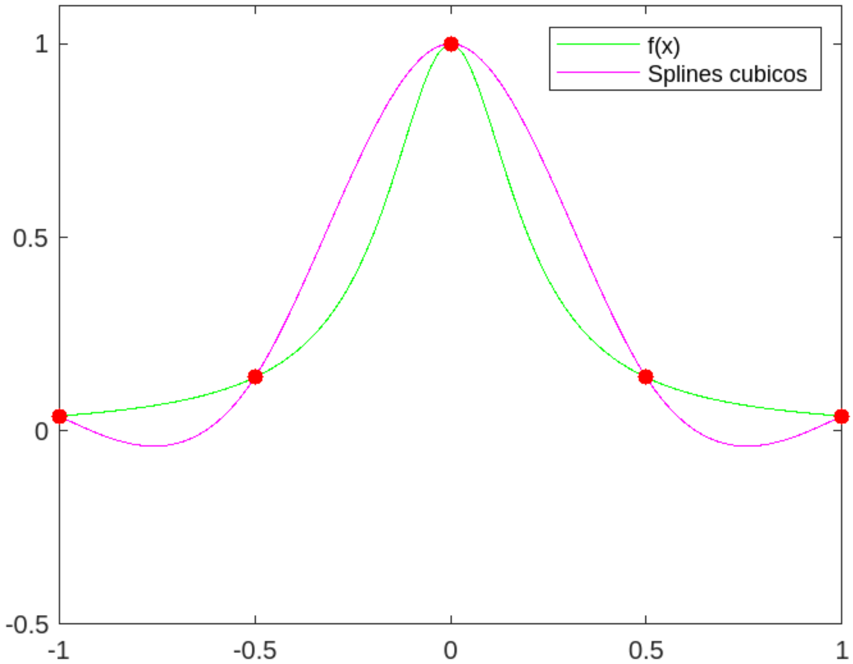
c) **Repita el numeral** b **empleando spline cúbicos.**

$$S_0(x) = 2{,}7283 * (x + 1{,}0)^3 - 0{,}48314 * x - 0{,}44468$$

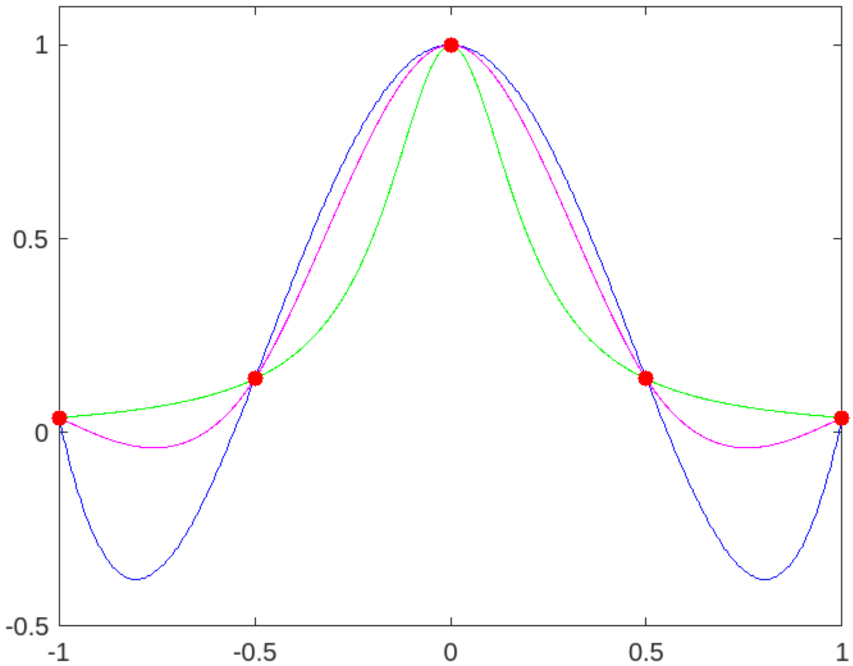
$$S_1(x) = 1{,}5631 * x + 4{,}0925 * (x + 0{,}5)^2 - 7{,}5407 * (x + 0{,}5)^3 + 0{,}91948$$

$$S_2(x) = 7{,}5407 * x^3 - 7{,}2186 * x^2 + 1{,}0$$

$$S_3(x) = 4{,}0925 * (x - 0{,}5)^2 - 1{,}5631 * x - 2{,}7283 * (x - 0{,}5)^3 + 0{,}91948$$



d) **Explique sus resultados.**



Podemos evidenciar al graficar ambas soluciones juntas, que el splines cubico hace que nuestras predicciones tengan menos oscilaciones con respecto a los polinomios de lagrange, lo anterior debido a que con los splines limitamos a funciones de tercer grado mientras que con lagrange obtengo polinomios del grado $n-1$ en este caso para lagrange obtengo un polinomio de grado 4, por lo cual su oscilación sera mayor con respecto a los splines cubicos.