Table of Contents

Part 1: Comparison of Equation Solving Methods

Compares solving x = 3^-x through fixed-point iteration, Aitken's method, and Steffenson's method

```
fprintf('Part 1: Comparison of Equation Solving Methods\n')

tolerance = 10^-8;

% Set starting value here
p0 = 0.5;

f = @(x) 3^-x;

currentIteration = 0;
solutionFound = false;

% Fixed-point iteration
while ~solutionFound
    currentIteration = currentIteration + 1;
    P = f(p0);
    if abs((P-p0)/P) < tolerance
        solutionFound = true;
    end
    p0 = P;
end

fprintf('Fixed-point iteration\n')
fprintf('Tolerance:  %e,  Approximation:  %.8f,  Iterations:  %d\n',
 tolerance, P, currentIteration)

p0 = 0.5;
p1 = f(p0);

currentIteration = 0;
solutionFound = false;

% Aitken's Method
while ~solutionFound
    currentIteration = currentIteration + 1;
    p2 = f(p1);
    hatP1 = p0 - (p1-p0)^2/(p2-(2*p1)+p0);
    % verifies that hatP0 exists before comparison. hatP0 dies not
 exist on
```

```
    % first loop because there is only one HatP value so it can not be
    % compared against anything
    if exist('hatP0', 'var') && abs((hatP1-hatP0)/hatP1) < tolerance
        solutionFound = true;
    end
    p0 = p1;
    p1 = p2;
    hatP0 = hatP1;
end


fprintf('Aitken''s Method\n')
fprintf('Tolerance:  %e,  Approximation:  %.8f,  Iterations:  %d\n',
 tolerance, P, currentIteration)

p0 = 0.5;

currentIteration = 0;
solutionFound = false;

% Steffenson's Method
while ~solutionFound
    currentIteration = currentIteration + 1;
    p1 = f(p0);
    p2 = f(p1);
    P = p0 - (p1 - p0)^2/(p2-(2*p1)+p0);
    if abs((P-p0)/P) < tolerance
        solutionFound = true;
    end
    p0 = P;
end

fprintf('Steffenson''s Method\n')
fprintf('Tolerance:  %e,  Approximation:  %.8f,  Iterations:  %d\n',
 tolerance, P, currentIteration)
```

```
Part 1: Comparison of Equation Solving Methods
Fixed-point iteration
Tolerance:  1.000000e-08,  Approximation:  0.54780862,  Iterations:
 34
Aitken's Method
Tolerance:  1.000000e-08,  Approximation:  0.54780862,  Iterations:
 13
Steffenson's Method
Tolerance:  1.000000e-08,  Approximation:  0.54780862,  Iterations:  4
```

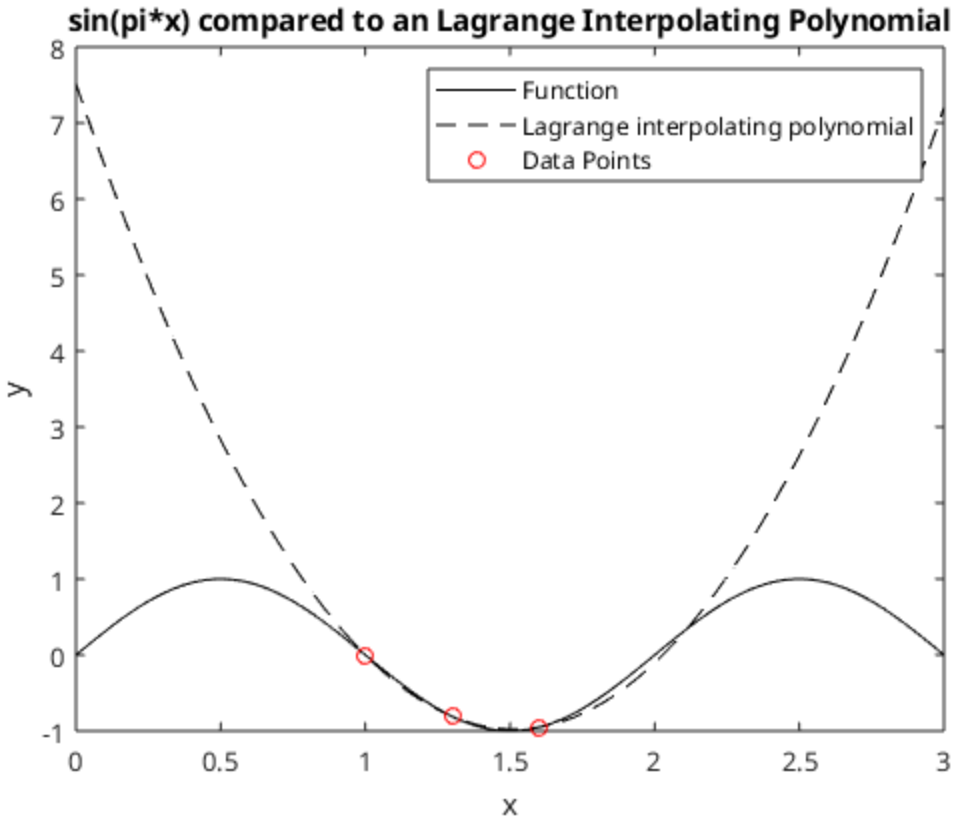
Part 2: Lagrange Interpolating Polynomial

Graphs the Lagrange Interpolating Polynomial for sin(pi*x) with points x = 1, 1.3, 1.6 against the graph of the actual function

```
fprintf('\nPart 2: Lagrange Interpolating Polynomial\n')
```

```
f = @(x) sin(pi*x);

x0 = 1;
x1 = 1.3;
x2 = 1.6;

L0 = @(x) ((x-x1).*(x-x2))/((x0-x1)*(x0-x2));
L1 = @(x) ((x-x0).*(x-x2))/((x1-x0)*(x1-x2));
L2 = @(x) ((x-x0).*(x-x1))/((x2-x0)*(x2-x1));
P = @(x) (f(x0)*L0(x)) + (f(x1)*L1(x)) + (f(x2)*L2(x));

x = linspace(0,3,100);
dataPoints = [x0 x1 x2];

figure;
plot(x, f(x), '-k');
hold on;
plot(x, P(x), '--k');
plot(dataPoints, f(dataPoints), 'or');
hold off;
xlabel('x');
ylabel('y');
title('sin(pi*x) compared to an Lagrange Interpolating Polynomial');
legend('Function', 'Lagrange interpolating polynomial', 'Data
 Points');

fprintf('Approximation of f(0.85): %f, Absolute Error: %f\n', P(0.85),
 abs(f(0.85) - P(0.85)));
fprintf('Approximation of f(1.15): %f, Absolute Error: %f\n', P(1.15),
 abs(f(1.15) - P(1.15)));
fprintf('Approximation of f(1.45): %f, Absolute Error: %f\n', P(1.45),
 abs(f(1.45) - P(1.45)));
fprintf('Approximation of f(1.75): %f, Absolute Error: %f\n', P(1.75),
 abs(f(1.75) - P(1.75)));
```

```
Part 2: Lagrange Interpolating Polynomial
Approximation of f(0.85): 0.654625, Absolute Error: 0.200635
Approximation of f(1.15): -0.487881, Absolute Error: 0.033890
Approximation of f(1.45): -0.963409, Absolute Error: 0.024279
Approximation of f(1.75): -0.771960, Absolute Error: 0.064853
```



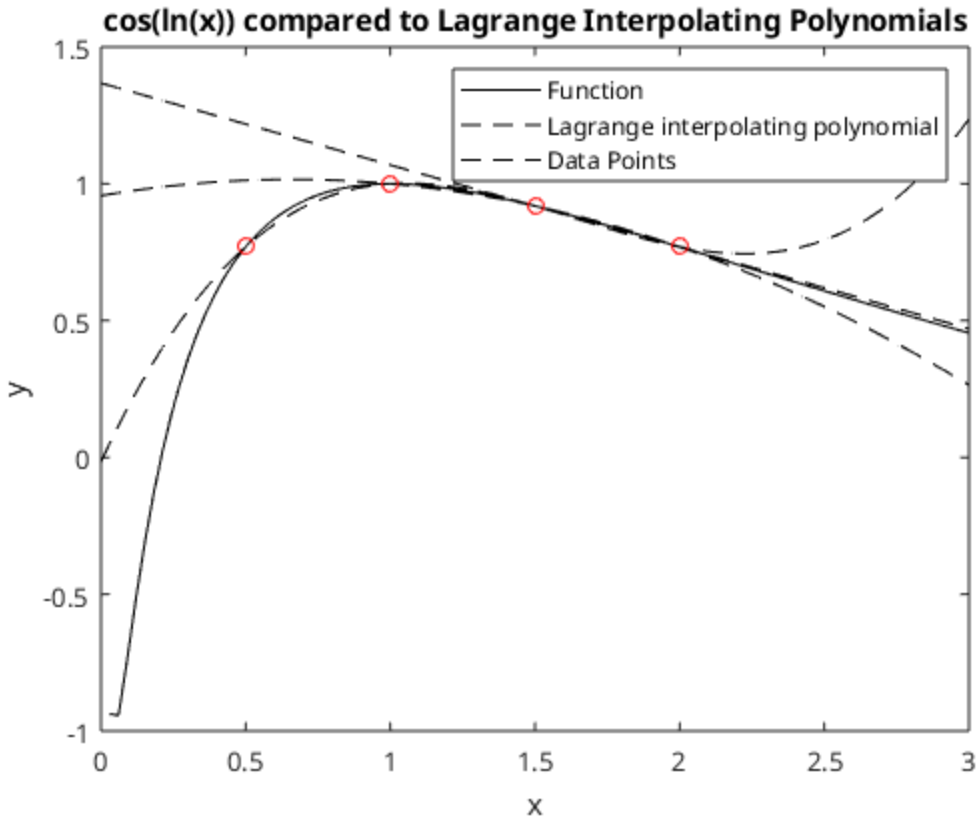
Part 3: Comparing Lagrange Interpolating Polynomials

Graphs a 1st, 2nd, and 3rd degree Lagrange Polynomial

```
fprintf('\nPart 3: Comparing Lagrange Interpolating Polynomials\n')

f = @(x) cos(log(x));

x0 = 0.5;
x1 = 1;
x2 = 1.5;
x3 = 2;

x = linspace(0,3,100);
dataPoints = [x0 x1 x2 x3];

% 1st Degree Langrange Polynomial

L10 = @(x) (x-x3)/(x2-x3);
L11 = @(x) (x-x2)/(x3-x2);
P1 = @(x)  (f(x2)*L10(x)) + (f(x3)*L11(x));

% 2nd Degree Lagrange Polynomial
```

```
L20 = @(x) ((x-x2).*(x-x3))/((x1-x2)*(x1-x3));
L21 = @(x) ((x-x1).*(x-x3))/((x2-x1)*(x2-x3));
L22 = @(x) ((x-x1).*(x-x2))/((x3-x1)*(x3-x2));
P2 = @(x) (f(x1)*L20(x)) + (f(x2)*L21(x)) + (f(x3)*L22(x));

% 3rd Degree Lagrange Polynomial

L30 = @(x) ((x-x1).*(x-x2).*(x-x3))/((x0-x1)*(x0-x2)*(x0-x3));
L31 = @(x) ((x-x0).*(x-x2).*(x-x3))/((x1-x0)*(x1-x2)*(x1-x3));
L32 = @(x) ((x-x0).*(x-x1).*(x-x3))/((x2-x0)*(x2-x1)*(x2-x3));
L33 = @(x) ((x-x0).*(x-x1).*(x-x2))/((x3-x0)*(x3-x1)*(x3-x2));
P3 = @(x) (f(x0)*L30(x)) + (f(x1)*L31(x)) + (f(x2)*L32(x)) +
 (f(x3)*L33(x));

figure;
plot(x, f(x), '-k');
hold on;
plot(x, P1(x), '--k');
plot(x, P2(x), '--k');
plot(x, P3(x), '--k');
plot(dataPoints, f(dataPoints), 'or');
hold off;
xlabel('x');
ylabel('y');
title('cos(ln(x)) compared to Lagrange Interpolating Polynomials');
legend('Function', 'Lagrange interpolating polynomial', 'Data
 Points');

fprintf('Approximation using 1st degree: %f, Absolute Error: %f\n',
 P1(1.75), abs(f(1.75) - P1(1.75)));
fprintf('Approximation using 2nd degree: %f, Absolute Error: %f\n',
 P2(1.75), abs(f(1.75) - P2(1.75)));
fprintf('Approximation using 3rd degree: %f, Absolute Error: %f\n',
 P3(1.75), abs(f(1.75) - P3(1.75)));

fprintf('The 1st degree interpolating polynomial function had the best
 approximation\n');
```

```
Part 3: Comparing Lagrange Interpolating Polynomials
Approximation using 1st degree: 0.844079, Absolute Error: 0.003380
Approximation using 2nd degree: 0.852654, Absolute Error: 0.005195
Approximation using 3rd degree: 0.837451, Absolute Error: 0.010008
The 1st degree interpolating polynomial function had the best
 approximation
```



Published with MATLAB® R2020a