Table of Contents

Part 1

Uses Bisection method to solve x-2^-x=0

```
fprintf('Part 1\n')

solutionFound = false;
currentIteration = 0;

% Set maximum number of iterations here
maxIterations = 30;

% Set acceptable error tolerance here
tolerance = 10^-4;

% Set range here
a = 0;
b = 1;

fa = a-2^-a;

while currentIteration < maxIterations && ~solutionFound
    currentIteration = currentIteration + 1;
    p = (a + b) / 2;
    fp = p-(2^-p);
    if fp == 0 || (b-a)/2 < tolerance
        solutionFound = true;
    end
    if sign(fa)*sign(fp) > 0
        a = p;
        fa = fp;
    else
        b = p;
    end
end
if ~solutionFound
    fprintf('Max iterations of %d reached\n', maxIterations)
end
fprintf('Tolerance:  %e,  Approximation:  %.4f,  Iterations:  %d\n',
 tolerance, p, currentIteration)
```

```
Part 1
```

```
Tolerance:  1.000000e-04,  Approximation:  0.6412,  Iterations:  14
```

Part 2

Uses Bisection method to solve x-2^-x=0

```
fprintf('Part 2\n')

solutionFound = false;
currentIteration = 0;

% Set maximum number of iterations here
maxIterations = 30;

% Set acceptable error tolerance here
tolerance = 10^-12;

% Set range here
a = 0;
b = 1;

fa = a-2^-a;

while currentIteration < maxIterations && ~solutionFound
    currentIteration = currentIteration + 1;
    p = (a + b) / 2;
    fp = p-(2^-p);
    if fp == 0 || (b-a)/2 < tolerance
        solutionFound = true;
    end
    if sign(fa)*sign(fp) > 0
        a = p;
        fa = fp;
    else
        b = p;
    end
end
if ~solutionFound
    fprintf('Max iterations of %d reached\n', maxIterations)
end
fprintf('Tolerance:  %e,  Approximation:  %.12f,  Iterations:  %d\n',
 tolerance, p, currentIteration)
```

```
Part 2
Max iterations of 30 reached
Tolerance:  1.000000e-12,  Approximation:  0.641185744666,
 Iterations:  30
```

Part 3

Uses Bisection method to solve x^3+x-4=0

```
fprintf('Part 3\n')
```

```
solutionFound = false;
currentIteration = 0;

% Set acceptable error tolerance here
tolerance = 10^-4;

% Set range here
a = 1;
b = 4;

fa = (a^3)+a-4;

while ~solutionFound
    currentIteration = currentIteration + 1;
    p = (a + b) / 2;
    fp = (p^3)+p-4;
    if fp == 0 || (b-a)/2 < tolerance
        solutionFound = true;
    end
    if sign(fa)*sign(fp) > 0
        a = p;
        fa = fp;
    else
        b = p;
    end
end
fprintf('Tolerance:  %e,  Approximation:  %.4f,  Iterations:  %d\n',
 tolerance, p, currentIteration)
```

```
Part 3
Tolerance:  1.000000e-04,  Approximation:  1.3788,  Iterations:  15
```

Part 4

Uses Bisection method to find the value of 3^.5

```
fprintf('Part 4\n')

solutionFound = false;
currentIteration = 0;

% Set acceptable error tolerance here
tolerance = 10^-8;

% Set range here
a = 1;
b = 2;

fa = (a^2)-3;

while ~solutionFound
    currentIteration = currentIteration + 1;
```

```
    p = (a + b) / 2;
    fp = (p^2)-3;
    % checks to make sure previousp variable exists first since the
 first
    % time the loop is run there is no previous value for p
    if fp == 0 || (exist('previousp', 'var') && abs((p-previousp)/p) <
 tolerance)
        solutionFound = true;
    end
    if sign(fa)*sign(fp) > 0
        a = p;
        fa = fp;
    else
        b = p;
    end
    previousp = p;
end
fprintf('Tolerance:  %e,  Approximation:  %.8f,  Iterations:  %d\n',
 tolerance, p, currentIteration)
```

```
Part 4
Tolerance:  1.000000e-08,  Approximation:  1.73205082,  Iterations:
 26
```

Part 5

Runs four fixed-point iteration methods to compute 21^1/3 and ranks their speed of convergence

```
fprintf('Part 5\n')

% Set acceptable error tolerance here
tolerance = 10^-10;

% set initial approximation here
p0 = 1;

%Part A

currentIteration = 0;
solutionFound = false;
previousp = p0;

while ~solutionFound
    currentIteration = currentIteration + 1;
    p = ((20*previousp) + (21/(previousp^2)))/21;
    if abs(p - previousp) < tolerance
        fprintf('Part A: Tolerance:  %e,  Approximation:  %.10f,
 Iterations:  %d\n', tolerance, p, currentIteration)
        solutionFound = true;
    end
    previousp = p;
end
```

```
%Part B

currentIteration = 0;
solutionFound = false;
previousp = p0;

while ~solutionFound
    currentIteration = currentIteration + 1;
    p = previousp - ((previousp^3-21)/(3*previousp^2));
    if abs(p - previousp) < tolerance
        fprintf('Part B: Tolerance:  %e,  Approximation:  %.10f,
 Iterations:  %d\n', tolerance, p, currentIteration)
        solutionFound = true;
    end
    previousp = p;
end

%Part C

currentIteration = 0;
solutionFound = false;
previousp = p0;

while ~solutionFound
    currentIteration = currentIteration + 1;
    p = previousp - (((previousp^4)-(21*previousp))/
((previousp^2)-21));
    if abs(p - previousp) < tolerance
        fprintf('Part C: Tolerance:  %e,  Approximation:  %.10f,
 Iterations:  %d\n', tolerance, p, currentIteration)
        solutionFound = true;
    end
    previousp = p;
end

%Part D

currentIteration = 0;
solutionFound = false;
previousp = p0;

while ~solutionFound
    currentIteration = currentIteration + 1;
    p = (21/previousp)^.5;
    if abs(p - previousp) < tolerance
        fprintf('Part D: Tolerance:  %e,  Approximation:  %.10f,
 Iterations:  %d\n', tolerance, p, currentIteration)
        solutionFound = true;
    end
    previousp = p;
end

fprintf('From greatest to least speed of convergence, it goes b, d, a.
 C does not converge.\n')
```

```
Part 5
Part A: Tolerance:  1.000000e-10,  Approximation:  2.7589241758,
 Iterations:  135
Part B: Tolerance:  1.000000e-10,  Approximation:  2.7589241764,
 Iterations:  9
Part C: Tolerance:  1.000000e-10,  Approximation:  0.0000000000,
 Iterations:  2
Part D: Tolerance:  1.000000e-10,  Approximation:  2.7589241764,
 Iterations:  37
From greatest to least speed of convergence, it goes b, d, a. C does
 not converge.
```

Part 6

Uses fixed-point iteration to find an approximation to the fixed point of g(x) = 2^-x

```
fprintf('Part 6\n')

% Set acceptable error tolerance here
tolerance = 10^-4;

% set initial approximation here
p0 = 1;

currentIteration = 0;
solutionFound = false;
previousp = p0;

while ~solutionFound
    currentIteration = currentIteration + 1;
    p = 2^(-previousp);
    if abs((p - previousp)/p) < tolerance
        fprintf('Tolerance:  %e,  Approximation:  %.4f,  Iterations:
 %d\n', tolerance, p, currentIteration)
        solutionFound = true;
    end
    previousp = p;
end
```

```
Part 6
Tolerance:  1.000000e-04,  Approximation:  0.6412,  Iterations:  12
```

Published with MATLAB® R2020a