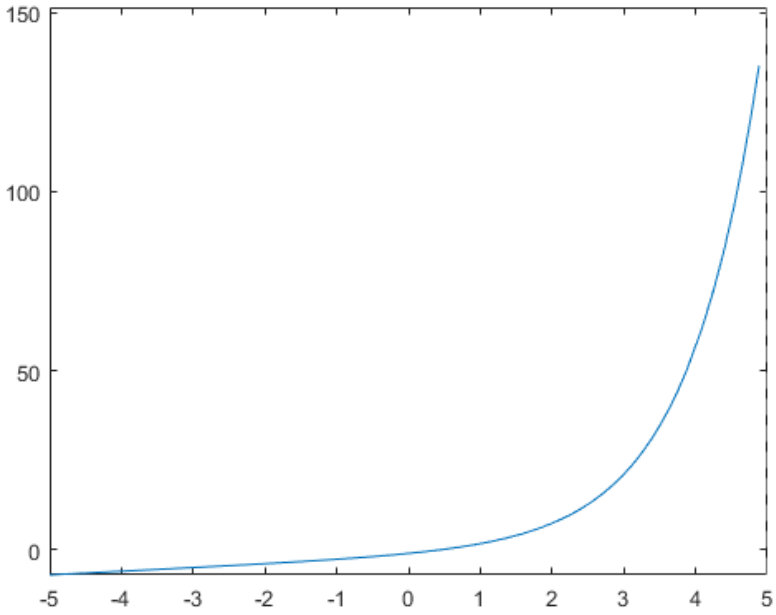
Numerical Analysis

Solving exp(x) + x - 2 = 0

Aristotle University of Thessaloniki

Computational Physics



Papadopoulou Martha

22/11/2023

Contents

1 Introduction

1.1 The Problem Statement

This assignment explores the process of finding the root of the following equation:

$$e^x + x - 2 = 0 \tag{1}$$

To achieve that, three numerical methods are employed: the bisection method, the false position method and the fixed-point iteration. The methodology of each one is described, and in the conclusion, the three methods are compared to see which one converges the fastest to the root. All methods are implemented using MATLAB.

As analyzed below, all the methods require starting points to initiate the iterations. Therefore, it's wise to have a graphical representation of the $f(x) = e^x + x - 2$ function to obtain a better understanding of the range of x-values within which the root lies. This can be easily accomplished using the "fplot" command in MATLAB to quickly visualize the nature of the function. The plot of $f(x)$ is depicted in Figure 1. With a quick observation, one can deduce that the root lies within the range $(0, 1)$. After this brief evaluation, we can commence the application of the methods.

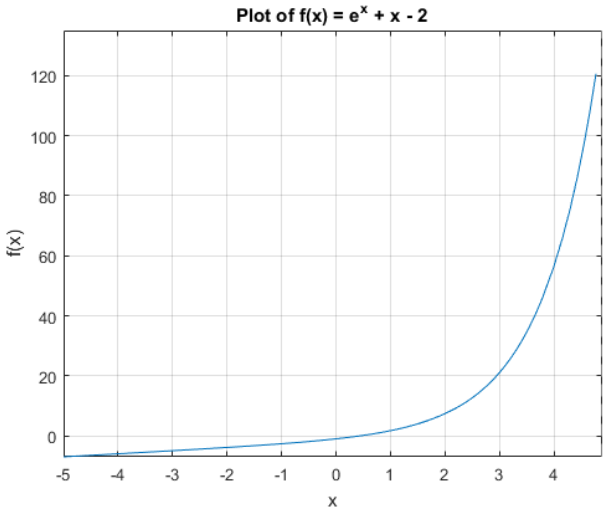


Figure 1: Plot of f(x)

1.2 Error Assessment

Prior to calculating the function's root, it's necessary to discuss about errors. Since the value of root is unknown, the only computable error is the approximate percent relative error.

$$\epsilon_a = \frac{current\ approximation - previous\ approximation}{current\ approximation} \times 100\% \tag{2}$$

In order to have a trustworthy result, up to a few significant figures, the ϵ_a error must be sufficiently small. To achieve this, it's obligatory to establish a threshold that ensures the iterations continue, until a specified condition is met. This condition is that the ϵ_a must be less than a prespecified percent tolerance ϵ_s, $|\epsilon_a| < \epsilon_s$. The number of significant figures that's needed for the root has to be established in order to calculate the value of ϵ_s.

$$\epsilon_s = (0.5 \times 10^{2-n})\% \tag{3}$$

For this assignment, the minimum significant figures that are required are six. Thus, to achieve a root accurate to six significant figures, the absolute value of the approximate error must be $|\epsilon_a| < 0.0005$. This threshold serves as the stopping criterion for all the methods employed.

2 Bisection Method

2.1 Theory of the Bisection Method

The bisection method is one of the simplest numerical methods used to calculate the root of a continuous function, but tends to have a longer computation time compared to other methods.

To start the iterations, two initial values are required, the lower x_l guess and the upper x_u, between which the root is located. This method is called bracketing. To ensure that the range of the starting values includes the root, the following condition must be met.

$$f(x_l)\, f(x_u) < 0 \tag{4}$$

The estimate root is:

$$x_r = \frac{x_l + x_u}{2} \tag{5}$$

To assess the acceptability of this value, the previous condition is tested again using the new x-value. If $f(x_l)\, f(x_r) < 0$, then the root lies within the lower range and the new upper x-value is $x_u = x_r$. If $f(x_l)\, f(x_r) > 0$, then the root is located in the upper range and the new lower x-value is $x_l = x_r$. Subsequently, we calculate the new x_r that lies within the new boundaries. The computation concludes when $f(x_l)\, f(x_r) = 0$. Numerically obtaining a zero value is impossible, even after a large number of iterations.

That is the reason we use the approximate error to control the loop. For the bisection method, the approximate percent error is represented as follows.

$$\epsilon_a = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right| 100\% = \left| \frac{x_u - x_l}{x_u + x_l} \right| 100\% \tag{6}$$

2.2 Implementation of the Bisection Method in Code

It's mentioned in the Introduction that the root of the function is within the range $(0, 1)$. For that reason, it's wise to begin the iterations with these values, $x_u = 1$ and $x_l = 0$. The value of ϵ_a is assigned a large number to enable entry into the loop, but it is not accounted for. All the ϵ_a values that will be computed later, are appended to an $\epsilon_a\, values$ list. Additionally, a "count" variable is created to track the number of iterations required to locate the root. All variables in the bisection method section of the code are named with an underscore "b" at the end, so that they are discrete for the same variables of other methods.

After establishing the initial values, the loop in which the computation occurs is created. The employed loop is a "while" loop, running until the absolute value of the approximate error is less than 0.0005. In case of a error within the code, the loop is terminated when the number of iterations exceeds 1000. While in the loop, the estimate root is calculated using the formula in equation (5). Depending on the condition met, which pertains to the function values discussed in the theoretical section, one of the interval edges assumes the value x_r. Afterward, the approximate percent error is calculated, according to the equation (6), and is appended to the $\epsilon_a\, values$ list. The count is increased by one, each time the loop is iterated.

When the "while" conditions are met and the loop terminates, the estimated root may have more decimal places than necessary, likely exceeding the correct precision after a certain point. The approximate percent error was set to ensure accuracy up to six significant figures, so the MATLAB function "chop" is used to round the root value for six significant figures.

If the computation is successful and the count is below 1000, the program plots the graph of ϵ_a values against the number of repetitions, which is shown in Figure 2.

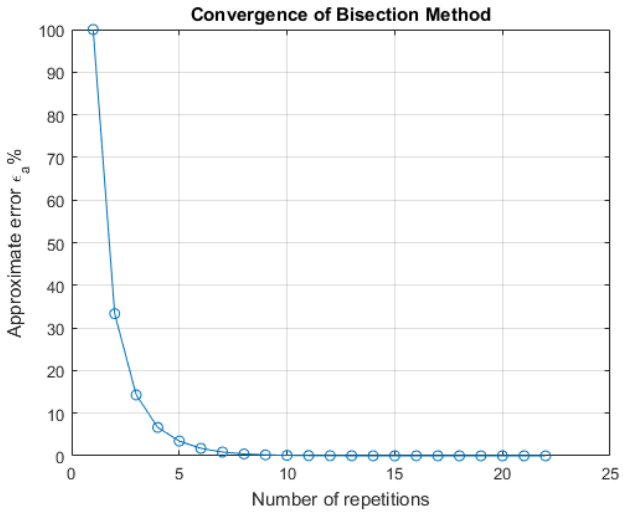


Figure 2: Plot of approximate error against number of repetitions for the Bisection Method

The program also showcases the calculated value of the root along with the number of repetitions that were needed. It is displayed in the output that the root is 0.442854 and that the required iterations are 22. As depicted in the Figure 2, the approximate error initially decreases rapidly. However, as the x-value approaches the root, a greater number of repetitions is necessary to achieve the de-

sired low error value. As expected, the bisection method converges to the root after many repetitions.

3 False Position Method

3.1 Theory of the False Position Method

The false position method is very similar to the bisection method, with a few adjustments that improve its speed. This is also a bracketing method, requiring that the initial guesses of the x-value encompass the root. This method considers that if the function value at one interval edge is closer to zero, then that particular x-value is more likely to be closer to the root.

The conditions used in the bisection method remain consistent with the relational expression (4), but the formula to calculate the estimate root is now:

$$x_r = x_u - \frac{f(x_u)\,(x_l - x_u)}{f(x_l) - f(x_u)} \tag{7}$$

Since the computation still requires an upper and a lower guess, the formula of the approximation error remains the same as expressed in equation (6).

However, if the function has a steep enough slope, only one of the interval edges will undergo change. This poses a challenge to the computation, as the number of iterations increases significantly without achieving a desirable value for the approximate error. To overcome this problem the false position method is modified. In cases where one of the edges changes twice in a row, the function value of the other edge is halved. This simple modification significantly reduces the computation time.

3.2 Implementation of the False Position Method in Code

As mentioned in the theoretical section on the false position method, this approach closely resembles the bisection method. Consequently, the majority of the code remains identical, undergoing only a few modifications. The variables used for the false position method section of the code are named with an underscore "r" in the end, in alignment with the alternative name of the method, Regula Falsi.

Same as before, the starting values are set as $x_u = 1$ and $x_l = 0$. The ϵ_a starts at a high value to initiate the loop, and the additional errors are stored in the $\epsilon_a\, values$ list. Because the code employs the modified version of the method, in addition to the "count" variable that represents the number of iterations, two more "count" variables are needed. These extra "counts" represent the number of consecutive changes in either x_u or x_l variables. It's empirically known that when these "counts" equal to two, the function value of the corresponding x-value has to be halved. But since in our code the $f(x)$ is defined as an one-line function, it cannot be halved. That is the reason why two more variables are defined, f_l and f_u that hold the corresponding values of $f(x_l)$ and $f(x_u)$.

The loop, again a "while" loop with the same conditions as before, runs until the approximate error reaches a satisfactory value or the number of repetitions exceeds 1000. Inside the loop the estimate

root is calculated using the equation (7). Once again, as analyzed in the theory of the bisection method, according to the multiplication of the function values, one of the interval edges assumes the value of x_r. Additionally, the function value of one of the edges takes on the value of $f(x_r)$. Afterward, the error ϵ_a is calculated and is appended to the ϵ_a *values* list. The count of repetitions and the count of the changing edge are both increased by one.

When the loop concludes, the root for six significant figures is determined using the "chop" function. If the computation is successful, the program plots the ϵ_a values as a function of the number of repetitions, as shown in Figure 3.

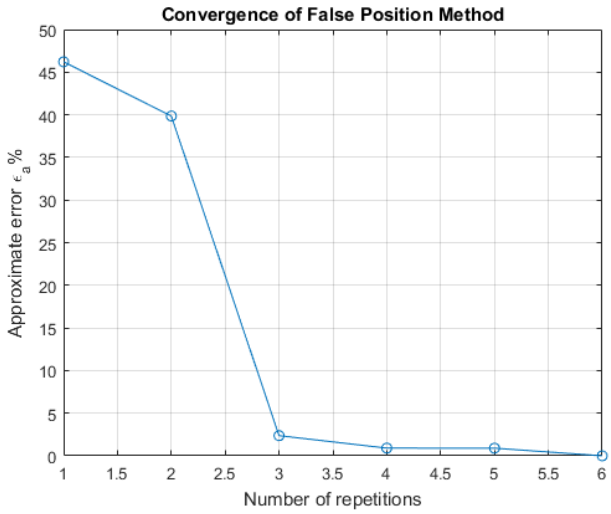


Figure 3: Plot of approximate error against number of repetitions for the False Position Method

In addition to that, the output reveals that the calculated root for six significant figures is 0.442854 and is achieved in 6 iterations. This method, because of its modification, converges to the root with very few repetitions.

4 Fixed-Point Iteration Method

4.1 Theory of the Fixed-Point Iteration

The fixed-point iteration is an open method, meaning that, unlike the two previous methods, it requires only one starting x-value. As the name suggests, this method uses a fixed-point repeatedly to determine the solution of an equation. To achieve this, we have to convert the original function $f(x)$, so that $x = g(x)$. Nevertheless, the $g(x)$ function cannot be arbitrary. In order for the method to converge to the root, the $g(x)$ function must satisfy the following condition.

$$|g'(x)| < 1 \tag{8}$$

If this condition is not met, the method will deviate further from the root, with each iteration. Therefore, we alter the original $f(x)$ function as follows.

$$g(x) = ln(2 - x) \tag{9}$$

In that case, for the initial value $x = 0$, the condition above is satisfied, because $|g'(0)| = 1/2$, so the method will converge. Even so, the value is not sufficiently small, so the convergence will be relatively slow.

Due to the fact that this method necessitates only one initial x-value, the approximate percent error takes the following form.

$$\epsilon_a = \left| \frac{x_{i+1} - x_i}{x_{i+1}} \right| 100\% \tag{10}$$

4.2 Implementation of the Fixed-Point Iteration in Code

The code for the fixed-point iteration differs significantly from the previous two. Firstly, the $g(x)$ of equation (9) is defined with a one-line function. The value of the starting point is set to $x = 0$, as it satisfies the condition (8) and is close to the root. The definition of the approximate error remains the same, the ϵ_a is set to a high value to initiate the loop and the following values are appended to the ϵ_a *values* list. A "count" is also established to track the number of repetitions. For this section of the code, the names of the variables end with an underscore "f".

The computation occurs within a "while" loop, that once more, runs until the ϵ_a reaches the desired value, or the number of iterations exceeds 1000. The code in the loop is very simple, the new estimate root equals the value of the $g(x)$ function for the old root. The approximation error is calculated using the equation (10), and is then added to the ϵ_a *values* list. The count increases by one after each repetition.

Once the loop terminates, the root is calculated using the "chop" function. If the count of repetitions is less than 1000, the same plot as in the other methods is presented, as shown in Figure 4.

Additionally, the program displays the value of the root, which is 0.442854, and the number of iterations, which is 35. From Figure 4, it's observable that the approximate error increases once, before it starts to decrease as the method converges to the root. Also, the number of iterations that were needed for the computation is excessively high. Both of these happen because the $|g'(0)| = 1/2$ is not much smaller than 1, as was suspected in the theoretical section.

5 Discussion

The final part of the code verifies if the roots obtained by the three different methods are equal. Since the results of each method were presented, it's evident they are identical. As an additional verification step, the function $f(x)$ is evaluated for that value of the root. The returned value is approximately 10^{-6}, which is considered sufficiently close to zero for the chosen significant figures.

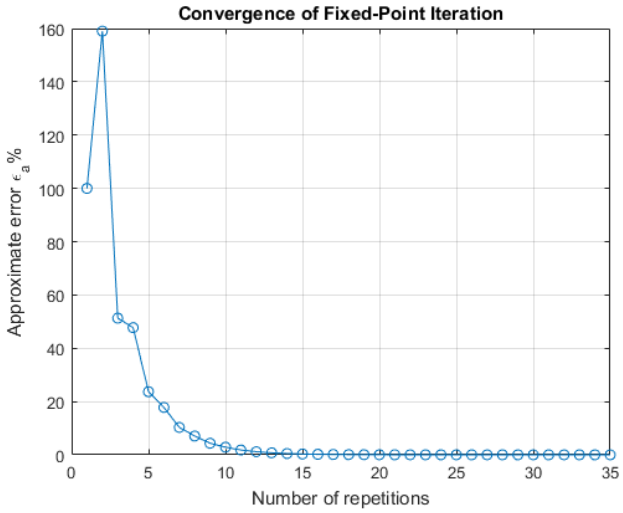


Figure 4: Plot of approximate error against number of repetitions for the Fixed-Point Iteration Method

Given all the data from the plots, the false position method has the lowest number of repetitions, followed by the bisection method, and the fixed-point iteration has the highest number of iterations. So the fasted method to converge to the root is the false position, then the bisection and the slowest method is the fixed-point iteration. This result is logical, considering that the false position method is modified for improved performance. Also, the fixed-point iteration was anticipated to be slow, since the convergence condition was not very satisfactory.

The ease of implementation for the methods follows the opposite order. The fixed-point iteration method is easy to compute, requiring only a few lines of code, without "if" statements. On the contrary, the false position method is rather complicated to compute, involving numerous "if" statements and calculations for each iteration. The bisection method falls between these two in terms of complexity. While it's not as simple as the fixed-point iteration, it includes only one "if" statement.

6 Conclusion

In conclusion for solving $e^x + x - 2 = 0$, all three methods yield the same root $x = 0.442854$, with the fastest being the false position method, followed by the bisection method and the slowest being the fixed-point iteration.

The bisection method is a simple and reliable technique. While the converge might be slow, as observed in this case, this method is guaranteed to converge. In this instance for starting values $x_u = 1$ and $x_l = 0$, 22 repetitions were required to obtain the root value with six significant figures.

The false position method is similar to the bisection method but more efficient. However, this method is prone to problems when the slope of the function is steep, as it is for this analysis. Hence, a slight modification is necessary, which enhances its speed. For the given function and starting values $x_u = 1$ and $x_l = 0$, this method converges to the root for six significant figures in just 6 iterations.

For the fixed-point iteration, the equation takes a fixed-point form $x = g(x)$, in this case $g(x) = ln(2 - x)$. The reason this function was chosen is that the condition $|g'(x)| < 1$ has to be met near the starting point. For this analysis, while the condition is fulfilled for initial value $x = 0$, the value of $|g'(x)|$ is not considerably less than 1, so the convergence is very slow. In order to reach the root value for six significant figures, the needed amount of repetitions is 35.

In summary, the selection of a numerical method depends on several factors. Firstly, the nature of the function and whether it satisfies the convergence conditions of the method. Additionally, one must consider the desired trade off between method efficiency and simplicity. As demonstrated in this assignment, the modified false position method is preferred for functions with steep slopes, as its modification accounts for this challenging scenario.

7 Code Listing

```
% Ypologistika Mathhmatika I
% Project 1
% Martha Papadopoulou

clear
clc

figure(1);
fplot(@(x) exp(x)+x-2);
xlabel('x');
ylabel('f(x)');
title('Plot of f(x) = e^{x} + x - 2');
grid on;
hold on;

%From the plot above we can see that the root is between 0<x<1.
%So for simplicity we will use those as are starting points.

n = 6; %significant figures
es = 0.5*10^(2-n); %prespecified percent tolerance
f = @(x) exp(x)+x-2;

%Bisection Method
xl_b = 0; %lower guess
xu_b = 1; %upper guess
ea_b = 100; %starting value to enter the loop
ea_values_b = []; %approximation error table
count_b = 0;

while ea_b>es && count_b <= 1000

    xr_b = (xu_b+xl_b)/2;

    if f(xl_b)*f(xr_b)<0
        xu_b = xr_b;
    elseif f(xl_b)*f(xr_b)>0
        xl_b = xr_b;
    end

    ea_b = abs((xu_b-xl_b)/(xu_b+xl_b))*100;
    ea_values_b = [ea_values_b, ea_b];
    count_b = count_b + 1;

end

root_b = chop(xr_b, n);

if count_b == 1001
    disp('Not able to find a root within 1000 iterations.')
else
    figure(2);
    plot(1:count_b,ea_values_b, '-o');
    xlabel('Number of repetitions')
    ylabel('Approximate error \epsilon_{a}%')
    title('Convergence of Bisection Method')
    grid on;

    fprintf('The root using the Bisection Method for %d significant figures is: %.6f\
```

```
    n', n, root_b);
    disp(['Number of repetitions using the Bisection Method: ' num2str(count_b)]);
end

%False Position Method (Regula Falsi)
xl_r = 0; %lower guess
xu_r = 1; %upper guess
ea_r = 100; %starting value to enter the loop
ea_values_r = []; %approximation error table
count_r = 0;
xur_count = 0; %counts how many times xu_r changes in a row
xlr_count = 0; %counts how many times xl_r changes in a row
fl_r = f(xl_r);
fu_r = f(xu_r);

while ea_r>es && count_r <= 1000

    xr_r = xu_r - (fu_r*(xl_r - xu_r))/(fl_r - fu_r);

    if fl_r*f(xr_r)<0
        xu_r = xr_r;
        fu_r = f(xr_r);
        xur_count = xur_count + 1;

    elseif fl_r*f(xr_r)>0
        xl_r = xr_r;
        fl_r = f(xr_r);
        xlr_count = xlr_count + 1;
    end

    ea_r = abs((xu_r - xl_r)/(xu_r + xl_r))*100;
    ea_values_r = [ea_values_r, ea_r];
    count_r = count_r + 1;

    if xur_count == 2
        fl_r = fl_r/2;
        xur_count = 0;
    end

    if xlr_count == 2
        fu_r = fu_r/2;
        xlr_count = 0;
    end

end

root_r = chop(xr_r, n);

if count_r == 1001
    disp('Not able to find a root within 1000 iterations.')
else
    figure(3);
    plot(1:count_r, ea_values_r, '-o');
    xlabel('Number of repetitions')
    ylabel('Approximate error \epsilon_{a}%')
    title('Convergence of False Position Method')
    grid on;

    fprintf('The root using the False Position Method for %d significant figures is:
    %.6f\n', n, root_r);
    disp(['Number of repetitions using the False Position Method: ' num2str(count_r)
```

```
    ]);
end

%Fixed-Point Iteration
g = @(x) log(2-x); %so that near the starting point |g'(x)|<1
x_f = 0; %starting point
ea_f = 100; %starting value to enter the loop
ea_values_f = []; %approximation error table
count_f = 0;

while ea_f>es && count_f <= 1000

    ea_f = abs((g(x_f)-x_f)/g(x_f))*100;
    ea_values_f = [ea_values_f, ea_f];
    x_f = g(x_f);
    count_f = count_f + 1;

end

root_f = chop(x_f, n);

if count_f == 1001
    disp('Not able to find a root within 1000 iterations.')
else
    figure(4);
    plot(1:count_f, ea_values_f, '-o');
    xlabel('Number of repetitions')
    ylabel('Approximate error \epsilon_{a}%')
    title('Convergence of Fixed-Point Iteration')
    grid on;

    fprintf('The root using the Fixed-Point Iteration for %d significant figures is:
    %.6f\n', n, root_f);
    disp(['Number of repetitions using the Fixed-Point Iteration: ' num2str(count_f)
    ]);
end

hold off;

result = (root_b==root_r) && (root_r==root_f);

if result == 1
    disp('All methods converge to a consensus value of the root.')
    disp(['The value of the f(x) = exp(x)+x-2 for x = ' num2str(root_b) ' is f(x) = '
     num2str(f(root_b))])
end
```