Alexandria University, Faculty of Engineering, SSP

Non-Linear Equations Analysis

Methods for root finding

Course: Numerical Analysis

Lecturer: Dr. Zeinab Eid

Team members

Name	ID
Mariam Atef Hassan	6892
Marwan Khaled Radwan	7020
Sarah Sherien Abdel Fattah	6818
Yasmine Emad	7107

Project Description:

The objective of the project is to compare and analyze the behavior of different numerical methods for calculating the roots of non-linear equations. The studied methods are Bisection, False-position, Fixed point, Newton-Raphson, and Secant.

The project was implemented using MATLAB. The user is expected to enter an equation (either by typing or reading from file), needed parameters and select a method of the mentioned five to process the computation. Also, single-step mode is available to display each iteration step-by-step and provide a graphical representation for the relation between the change of the approximate root values and the iteration count.

1st: Bisection:

A) Pseudo-code:

```
1. read function f(x) from user
2. receive input parameters:
   a. xa & xb
   b. Maximum number of iterations
   c. precision (tolerable error)
3. If f(xa)*f(xb) > 0
        "Guesses don't bracket (wrong)"
        return
   End If
4. arr ← []  //to return all iterations results
5. x_old ← xa
6. for i = 1 to max_iterations
        xr ← (xa+xb)/2
        ea ← abs((xr-x_old)/x_old)
        arr.append([i-1 xr ea])
        If f(xa)*f(xr) < 0
             xb = xr
        Else If f(xa)*f(xr) > 0
             xa = xr
        Else
             ea ← 0
        End If
        If ea < tolerance
             Break
        End If
        X_old ← xr
7. print last xr value as the result
```

B) MATLAB code:

```
function root = bisection(obj, xl, xu, es, imax, equ)
        syms x

        equFH = matlabFunction(equ);
        fxl = equFH(xl);
        fxu = equFH(xu);
        if (fxl * fxu > 0 ) %guesses do not bracket
            root = -1;
            return
        end
        arr = [];
        xo = xl;
        for i =1:1:imax
            xr= (xu+xl)/2;
            ea = abs((xr-xo)/xo);
            test = equFH(xl)*equFH(xr);
            arr = [arr ; i-1 xr ea];
            if(test < 0)
                xu=xr;
            else
                xl= xr;
            end
            if (test == 0)
                ea=0;
            end
            if(ea < es )
                break ;
            end
            xo = xr;
        end
        root = arr;
    end
```

C) Used Data Structures:

There are no special used data structures, except for MATLAB matrices which were used to store the details of all iterations to display them all for the user.

D) Problematic Functions & Cases:

- Passed values for xa and xb may be invalid if they don't bracket. Therefore, user will have to try again.
- Generally, for bracketing methods, some equations' roots cannot be found as it never satisfies bracketing condition for any initial values of xa & xb. Example: $f(x) = x^2$ or $f(x) = e^x$. Also, in case of intervals that contain discontinuities or only return function values of same sign, bracketing methods won't work.
- Bisection is relatively slow in comparison to other methods.

2nd: False Position:

A) Pseudo-code:

```
1. read function f(x) from user
2. receive input parameters:
   a. xa & xb
   b. Maximum number of iterations
   c. precision (tolerable error)
3. If f(xa)*f(xb) > 0
        "Guesses don't bracket (wrong)"
        return
   End If
4. arr ← []  //to store all iterations results
5. x_old ← xa
6. for i = 1 to max_iterations
        xr ← (xa*f(xb) – xb*f(xa))/(f(xb)-f(xa))
        ea ← abs((xr-x_old)/x_old)
        arr.append([i-1 xr ea])
        If f(xa)*f(xr) < 0
             xb = xr
        Else If f(xa)*f(xr) > 0
             xa = xr
        Else
             ea ← 0
        End If
        If ea < tolerance
             Break
        End If
        X_old ← xr
7. print last xr value as the result
```

B) MATLAB code:

```
function root = false_position(obj,xl, xu, es, maxit,equ)
           syms x
           equFH = matlabFunction(equ);
           gFH = matlabFunction(equ);
           fxl = gFH(xl);
           fxu = gFH(xu);
           if (fxl * fxu > 0 ) %guesses do not bracket
               root = -1;
               return
           end
           arr = [];
           xo = xl;
           for i =1:1:maxit
               fxl = equFH(xl);
               fxu = equFH(xu);
               xr = (xl*fxu - xu*fxl)/(fxu-fxl);
               fxr = gFH(xr);
               ea = abs((xr-xo)/xo);
               test= fxl*fxr;
               arr = [arr ; i-1 xr ea];
               if(test < 0)
                   xu=xr;
               else
                   xl= xr;
               end
               if (test == 0)
                   ea=0;
               end
               if(ea < es )
                   break ;
               end
               xo = xr;
           end
           root = arr;
       end
```

C) Used Data Structures:

There are no special used data structures, except for MATLAB matrices which were used to store the details of all iterations to display them all for the user.

D) Problematic Functions & Cases:

- Passed values for xa and xb may be invalid if they don't bracket. Therefore, user will have to try again.
- Generally, for bracketing methods, some equations' roots cannot be found as for any initial values of xa and xb it won't achieve bracketing conditions such as ; $f(x) = x^2$ or $f(x) = e^x$. Also, in case of intervals that contain discontinuities or return function values of same sign, bracketing methods won't work.
- False Position is relatively slow in comparison to other methods.

3rd: Fixed Point:

A) Pseudo-code:

```
1. read function g(x) from user
2. receive input parameters:
   a. xa
   b. Maximum number of iterations
   c. precision (tolerable error)
3. If g(xa) > 1
         "Method diverges (wrong)"
         return
   End If
4. arr ← []  //to store all iterations results
5. x_old ← xa
6. for i = 1 to max_iterations
         xr ← g(x_old)
         ea ← abs((xr-x_old)/x_old)
         arr.append([i-1 xr ea])
         If ea < tolerance
               Break
         End If
         X_old ← xr
7. print last xr value as the result
```

B) MATLAB code:

```
function root = fixed_point(obj,xi, es, maxit,equ)
           syms x
           equFH = matlabFunction(equ);
           df = diff(equ,x);
           dfFH = matlabFunction(df);
           fdash = dfFH(xi);
           if (abs(fdash) > 1) %diverges
               root = -1;
               return
           end

           arr = [];
           xo = xi;
           for i =1:1:maxit
               xr = equFH(xo);
               ea = abs((xr-xo)/xo);
               arr = [arr ; i xr ea];
               if(ea < es )
                   break ;
               end
               xo = xr;
           end
           root = arr;
       end
```

C) Used Data Structures:

There are no special used data structures, except for MATLAB matrices which were used to store the details of all iterations to display them all for the user.

D) Problematic Functions & Cases:

- Passed function may diverge. Therefore, user will have to try again.
- In case of functions that are insolvable by MATLAB, theoretical bound may not be calculated (as MATLAB won't be able to estimate exact values). This is a very rare case, resulted usually due to invalid equations.

4th: Newton-Raphson:

A) Pseudo-code:

```
1. read function f(x) from user
2. receive input parameters:
   a. xa
   b. Maximum number of iterations
   c. precision (tolerable error)
3. get f`(x)
4. arr ← []  //to store all iterations results
5. x_old ← xa
6. for i = 1 to max_iterations
        xr ← x_old – (f(x_old)/f`(x_old))
        ea ← abs((xr-x_old)/x_old)
        arr.append([i-1 xr ea])
        If ea < tolerance
             Break
        End If
        x_old ← xr
7. print last xr value as the result
```

B) MATLAB code:

```
function root = newton(obj,xi, es, maxit,equ)
    syms x
    equFH = matlabFunction(equ);
    df = diff(equ,x);
    dfFH = matlabFunction(df);
    arr = [];
    xo = xi;
    for i =1:1:maxit
        f = equFH(xo);
        fdash = dfFH(xo);
        xr = xo - (f/fdash);
        ea = abs((xr-xo)/xo);
        arr = [arr ; i xr ea];
        if(ea < es )
            break ;
        end
        xo = xr;
    end
    root = arr;
end
```

C) Used Data Structures:

There are no special used data structures, except for MATLAB matrices which were used to store the details of all iterations to display them all for the user.

D) Problematic Functions & Cases:

- The general Newton-Raphson problem may arise, such as: division by zero, root jumping and inflection point problems.
- It requires derivative calculation. Therefore, more time and complexity and function should be differentiable of constant value !=0 for xa.

5th: Secant:

A) Pseudo-code:

```
1. read function f(x) from user
2. receive input parameters:
   a. xi and xj
   b. Maximum number of iterations
   c. precision (tolerable error)
3. arr ← []  //to store all iterations results
4. for i = 1 to max_iterations
        xr ← xi – (f(xi)*(xj-xi)/(f(xj)-f(xi)))
        ea ← abs((xr-x_old)/x_old)
        arr.append([i-1 xr ea])
        If ea < tolerance
             Break
        End If
        xj ← xi
        xi ← xr
5. print last xr value as the result
```

B) MATLAB code:

```
function root = secant(obj,xj,xi, es, maxit,equ)
    syms x
    equFH = matlabFunction(equ);
    arr = [];
    for i =1:1:maxit
        fxi = equFH(xi);
        fxj = equFH(xj);
        xr = xi - fxi*(xj-xi)/(fxj-fxi);
        ea = abs((xr-xi)/xi);
        arr = [arr ; i xr ea];
        if(ea < es )
            break ;
        end
        xj = xi;
        xi = xr;
    end
    root = arr;
end
```

C) Used Data Structures:

There are no special used data structures, except for MATLAB matrices which were used to store the details of all iterations to display them all for the user.

D) Problematic Functions & Cases:

- No common problems

Behavior Analysis for the Five Methods

Example 1:

$f(x) = x^2 - 3$ *****Function diverges in case of fixed point***

$x_a = 0, \quad x_b = 2, \quad \max_{iterations} = 50, \quad precision = 0.00001$

	Bisection	False position	Fixed point	Newton-Raphson	Secant
Execution time	0.091806	0.072283	-	0.0875589	0.0241187
Iterations	17	6	-	4	5

Example 2:

$f(x) = \cos(x) - 3 * x + 1, \qquad g(x) = (\cos(x) + 1)/3$

$x_a = 0, \quad x_b = 1, \qquad \underset{iterations}{\max} = 50, \qquad precision = 0.00001$

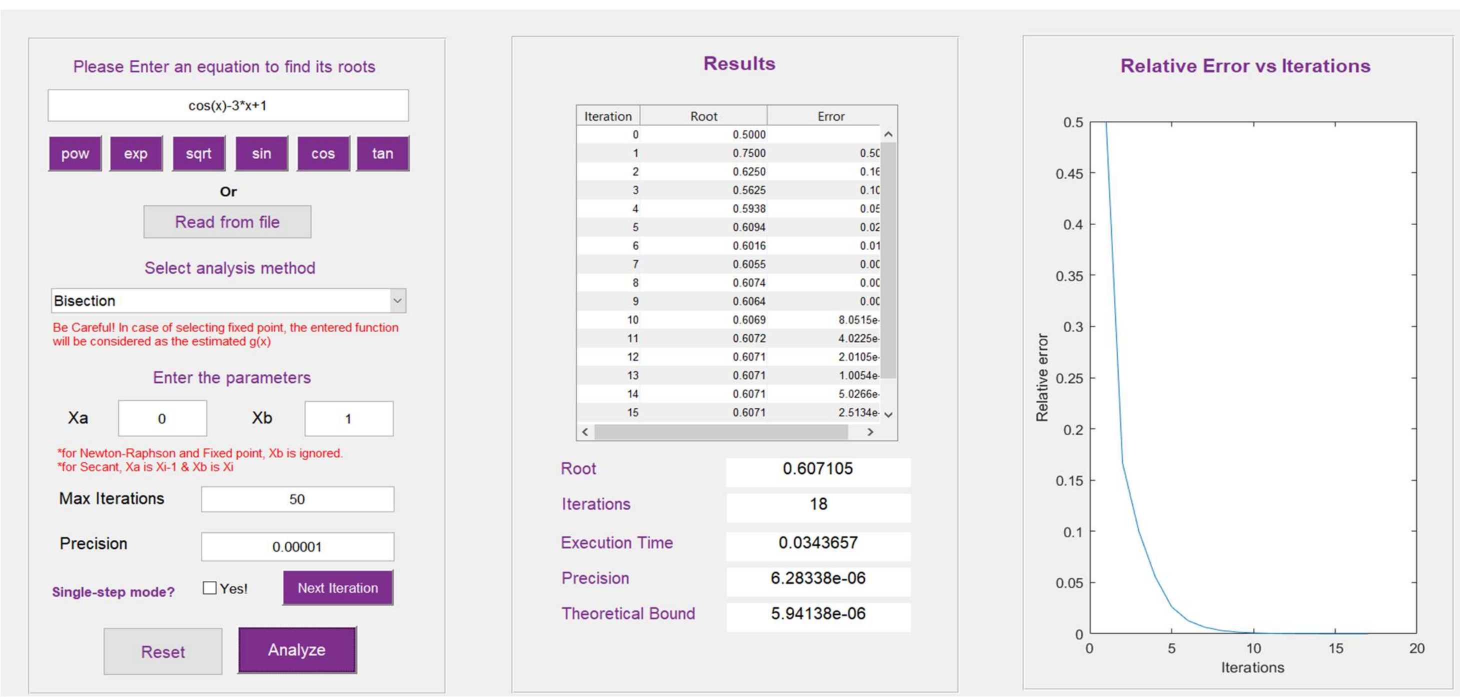
	Bisection	False position	Fixed point	Newton-Raphson	Secant
Execution time	0.0343657	0.0680798	0.0559184	0.0544175	0.0431804
Iterations	18	5	7	4	4

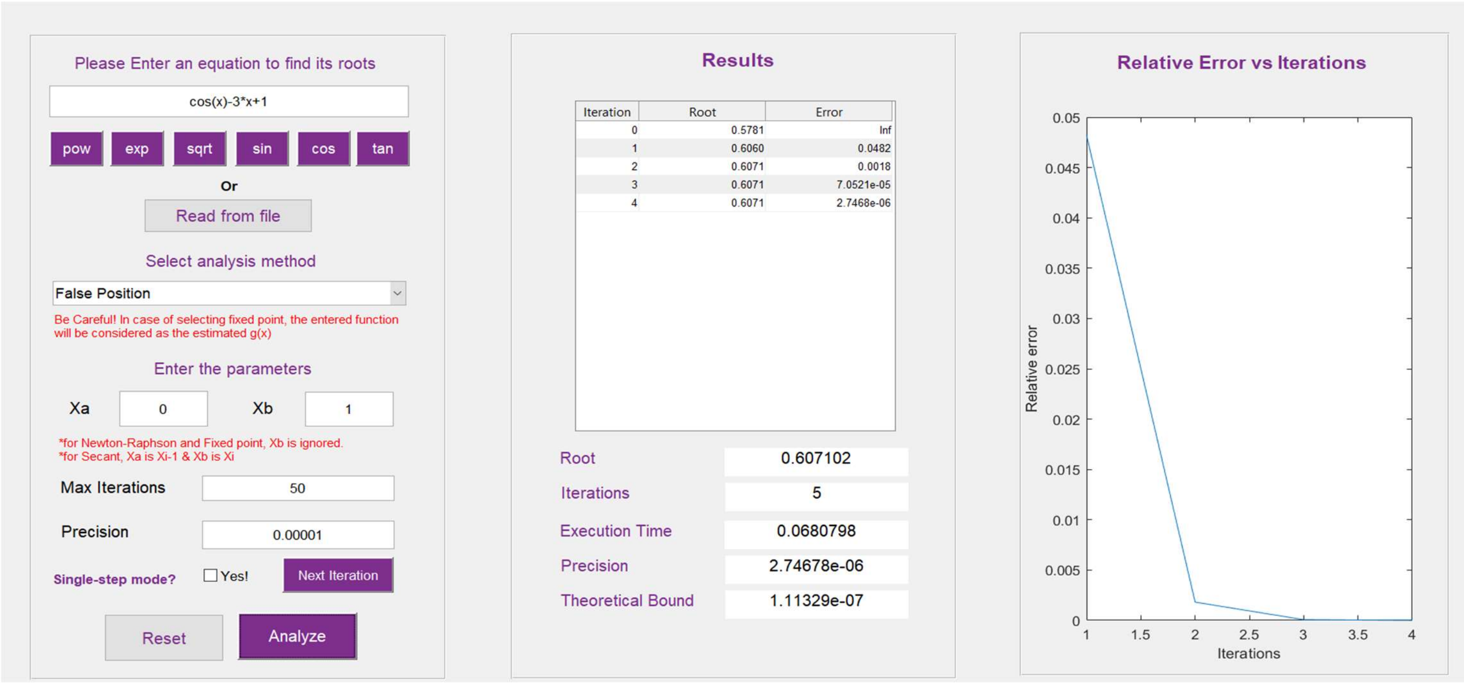
Sample Runs

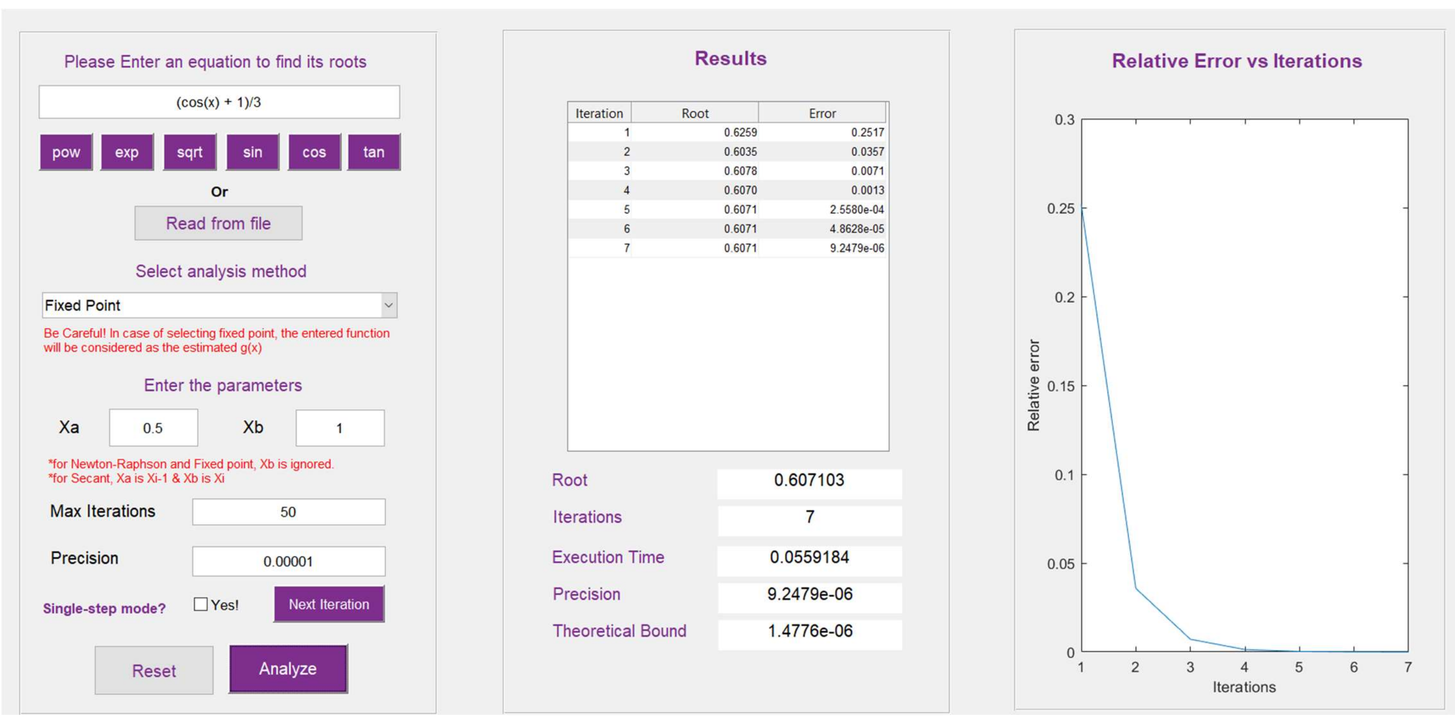
Example 1:

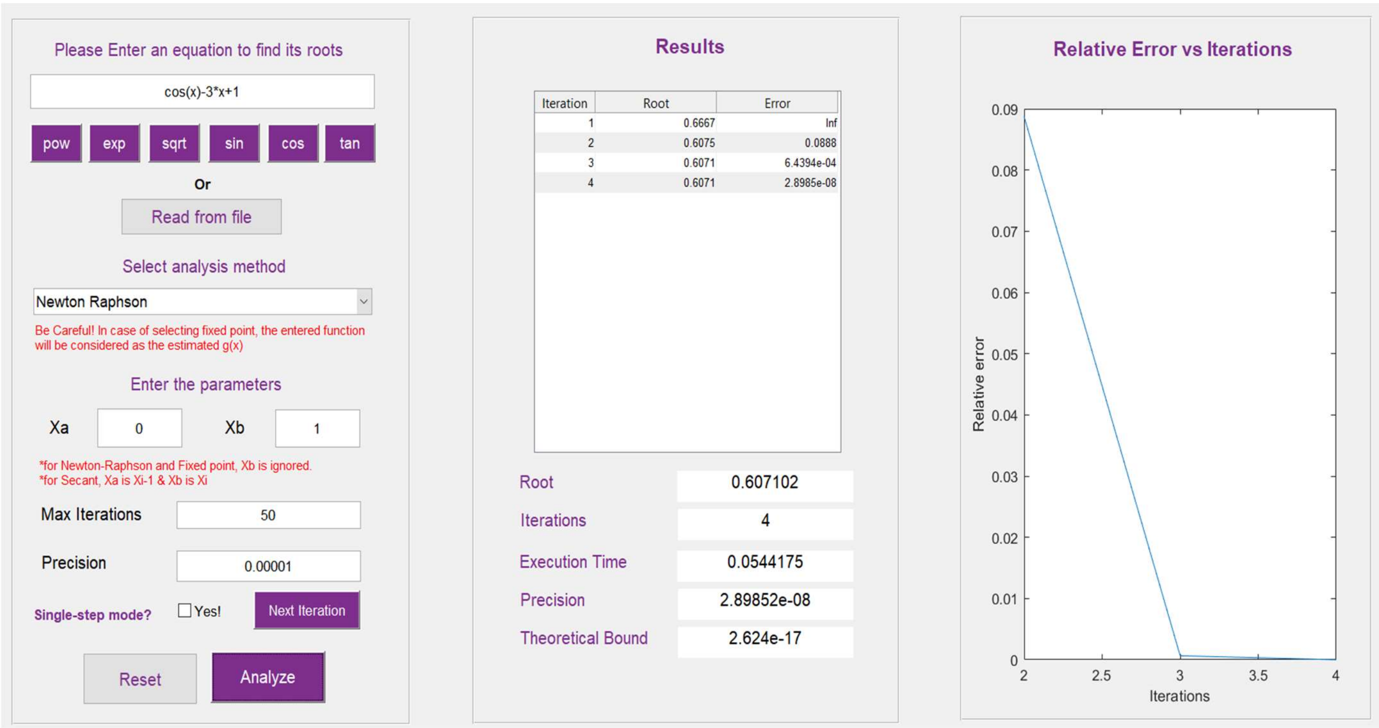
$f(x) = \cos(x) - 3 * x + 1, \qquad g(x) = (\cos(x) + 1)/3$

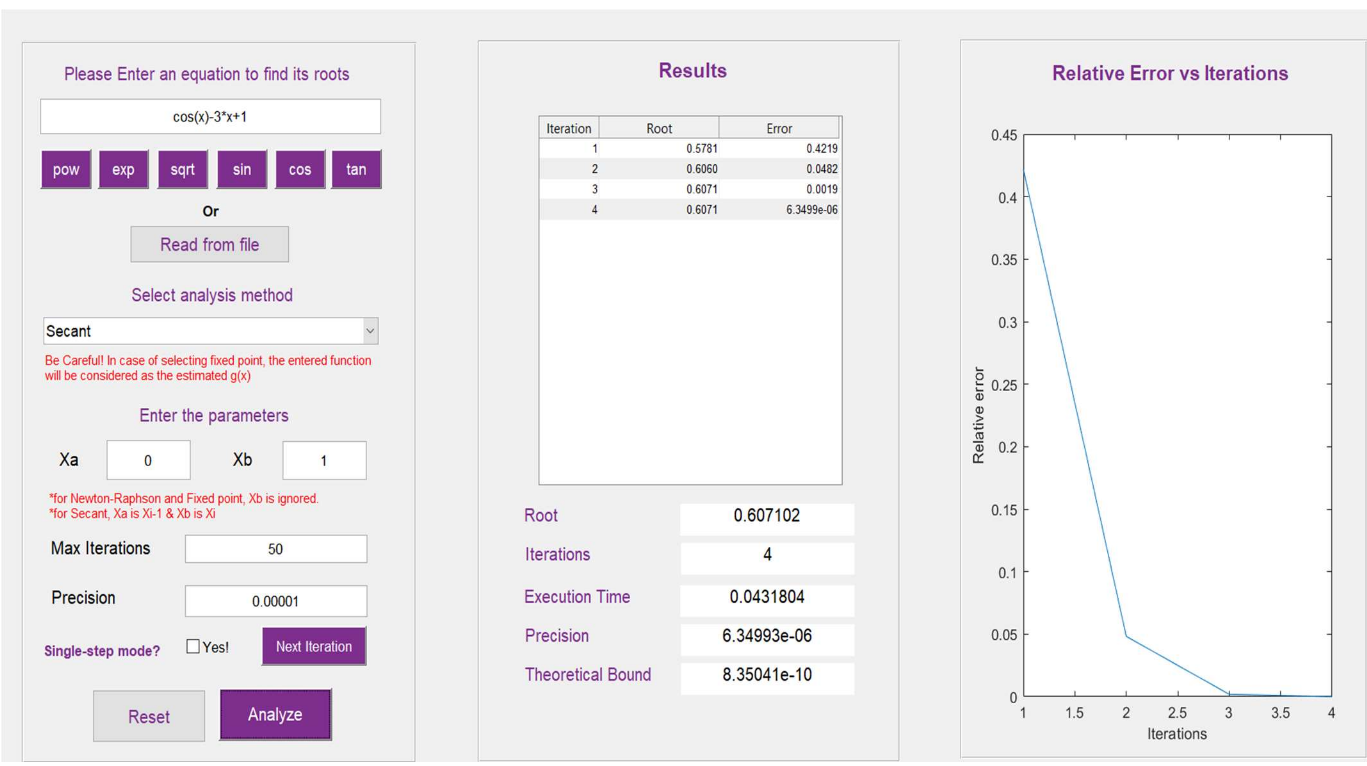
$x_a = 0, \quad x_b = 1, \qquad \underset{iterations}{\max} = 50, \qquad precision = 0.00001$




Please Enter an equation to find its roots
cos(x)-3*x+1
pow
exp
sqrt
sin
cos
tan
Or
Read from file
Select analysis method
False Position
Be Careful! In case of selecting fixed point, the entered function will be considered as the estimated g(x)
Enter the parameters
Xa
0
Xb
1
*for Newton-Raphson and Fixed point, Xb is ignored.
*for Secant, Xa is Xi-1 & Xb is Xi
Max Iterations
50
Precision
0.00001
Single-step mode?
Yes!
Next Iteration
Reset
Analyze
Results
Iteration Root Error
0 0.5781 Inf
1 0.6060 0.0482
2 0.6071 0.0018
3 0.6071 7.0521e-05
4 0.6071 2.7468e-06
Root
0.607102
Iterations
5
Execution Time
0.0680798
Precision
2.74678e-06
Theoretical Bound
1.11329e-07
Relative Error vs Iterations
Relative error
Iterations

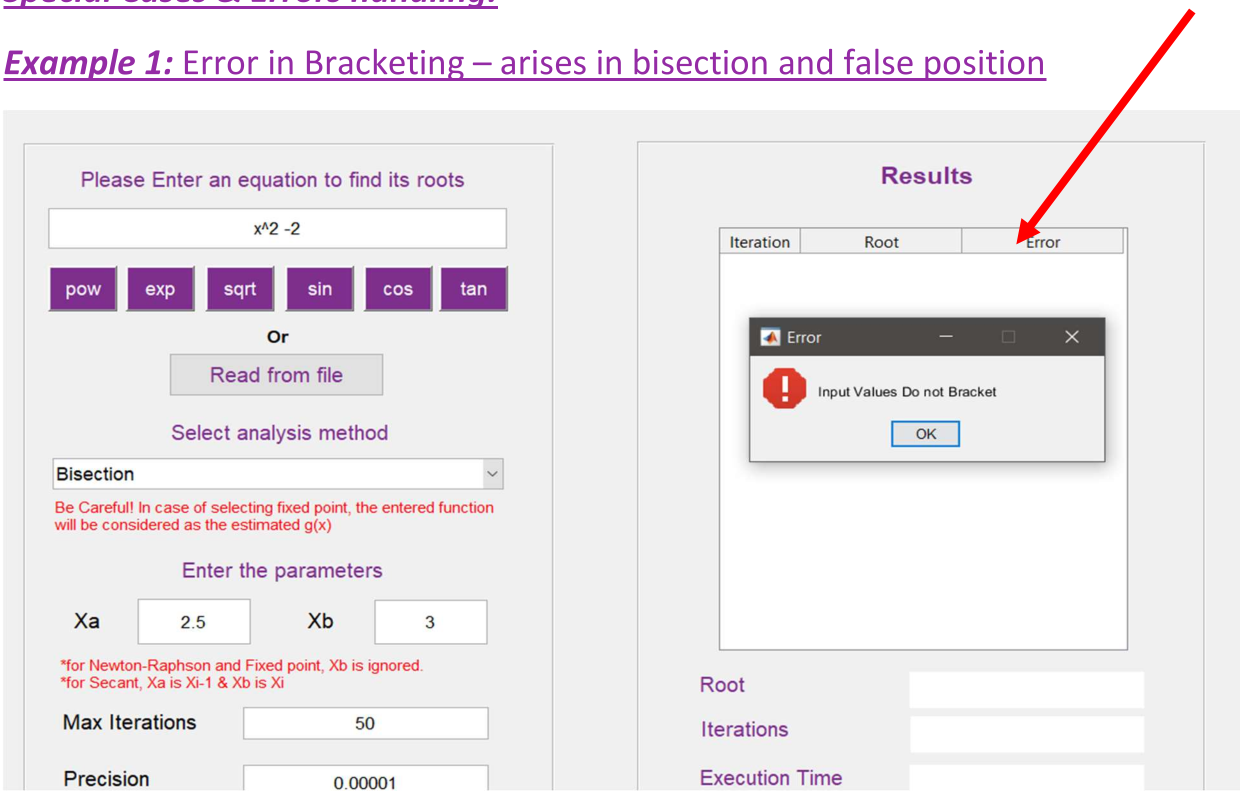

Please Enter an equation to find its roots
(cos(x) + 1)/3
pow
exp
sqrt
sin
cos
tan
Or
Read from file
Select analysis method
Fixed Point
Be Careful! In case of selecting fixed point, the entered function will be considered as the estimated g(x)
Enter the parameters
Xa
0.5
Xb
1
*for Newton-Raphson and Fixed point, Xb is ignored.
*for Secant, Xa is Xi-1 & Xb is Xi
Max Iterations
50
Precision
0.00001
Single-step mode?
Yes!
Next Iteration
Reset
Analyze
Results
Iteration Root Error
1 0.6259 0.2517
2 0.6035 0.0357
3 0.6078 0.0071
4 0.6070 0.0013
5 0.6071 2.5580e-04
6 0.6071 4.8628e-05
7 0.6071 9.2479e-06
Root
0.607103
Iterations
7
Execution Time
0.0559184
Precision
9.2479e-06
Theoretical Bound
1.4776e-06
Relative Error vs Iterations
Relative error
Iterations


Please Enter an equation to find its roots
cos(x)-3*x+1
pow
exp
sqrt
sin
cos
tan
Or
Read from file
Select analysis method
Newton Raphson
Be Careful! In case of selecting fixed point, the entered function will be considered as the estimated g(x)
Enter the parameters
Xa
0
Xb
1
*for Newton-Raphson and Fixed point, Xb is ignored.
*for Secant, Xa is Xi-1 & Xb is Xi
Max Iterations
50
Precision
0.00001
Single-step mode?
Yes!
Next Iteration
Reset
Analyze
Results
Iteration Root Error
1 0.6667 Inf
2 0.6075 0.0888
3 0.6071 6.4394e-04
4 0.6071 2.8985e-08
Root
0.607102
Iterations
4
Execution Time
0.0544175
Precision
2.89852e-08
Theoretical Bound
2.624e-17
Relative Error vs Iterations
Relative error
Iterations
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
2
2.5
3
3.5
4

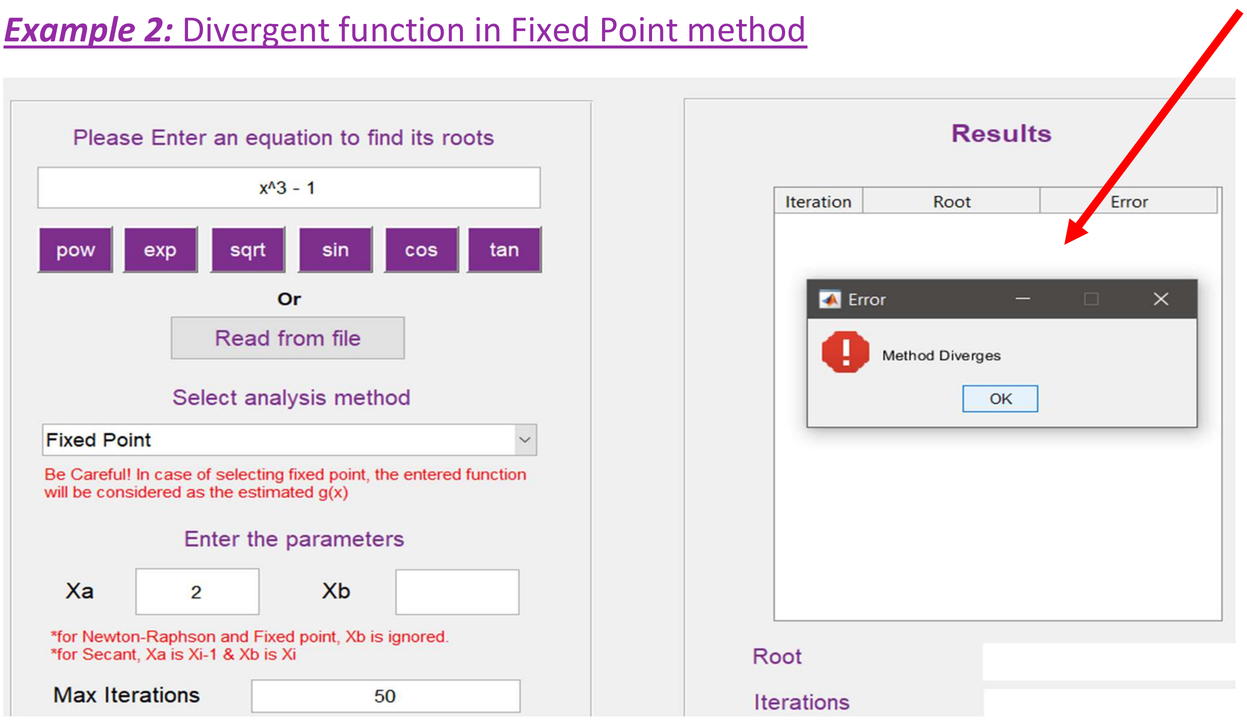

Please Enter an equation to find its roots
cos(x)-3*x+1
pow
exp
sqrt
sin
cos
tan
Or
Read from file
Select analysis method
Secant
Be Careful! In case of selecting fixed point, the entered function will be considered as the estimated g(x)
Enter the parameters
Xa
0
Xb
1
*for Newton-Raphson and Fixed point, Xb is ignored.
*for Secant, Xa is Xi-1 & Xb is Xi
Max Iterations
50
Precision
0.00001
Single-step mode?
Yes!
Next Iteration
Reset
Analyze
Results
Iteration Root Error
1 0.5781 0.4219
2 0.6060 0.0482
3 0.6071 0.0019
4 0.6071 6.3499e-06
Root
0.607102
Iterations
4
Execution Time
0.0431804
Precision
6.34993e-06
Theoretical Bound
8.35041e-10
Relative Error vs Iterations
Relative error
Iterations
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
1
1.5
2
2.5
3
3.5
4

Special Cases & Errors handling:

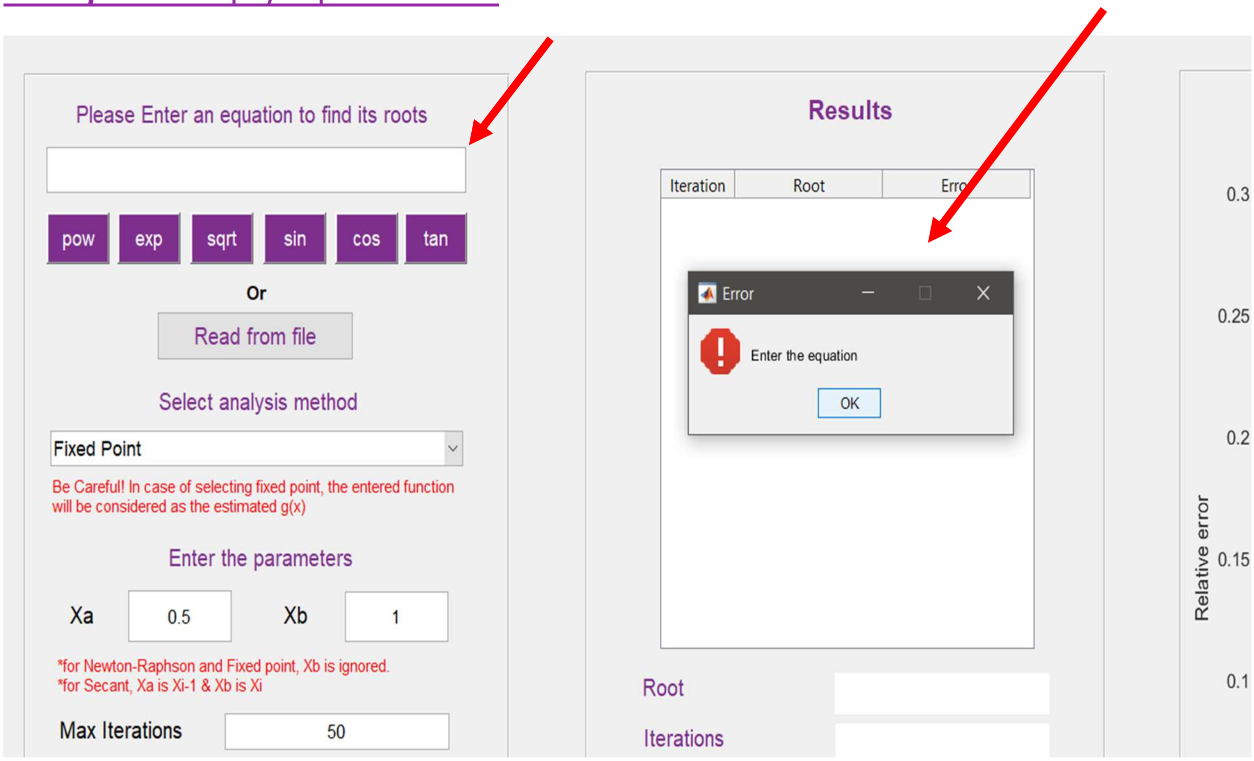
Example 1: Error in Bracketing – arises in bisection and false position



Example 2: Divergent function in Fixed Point method

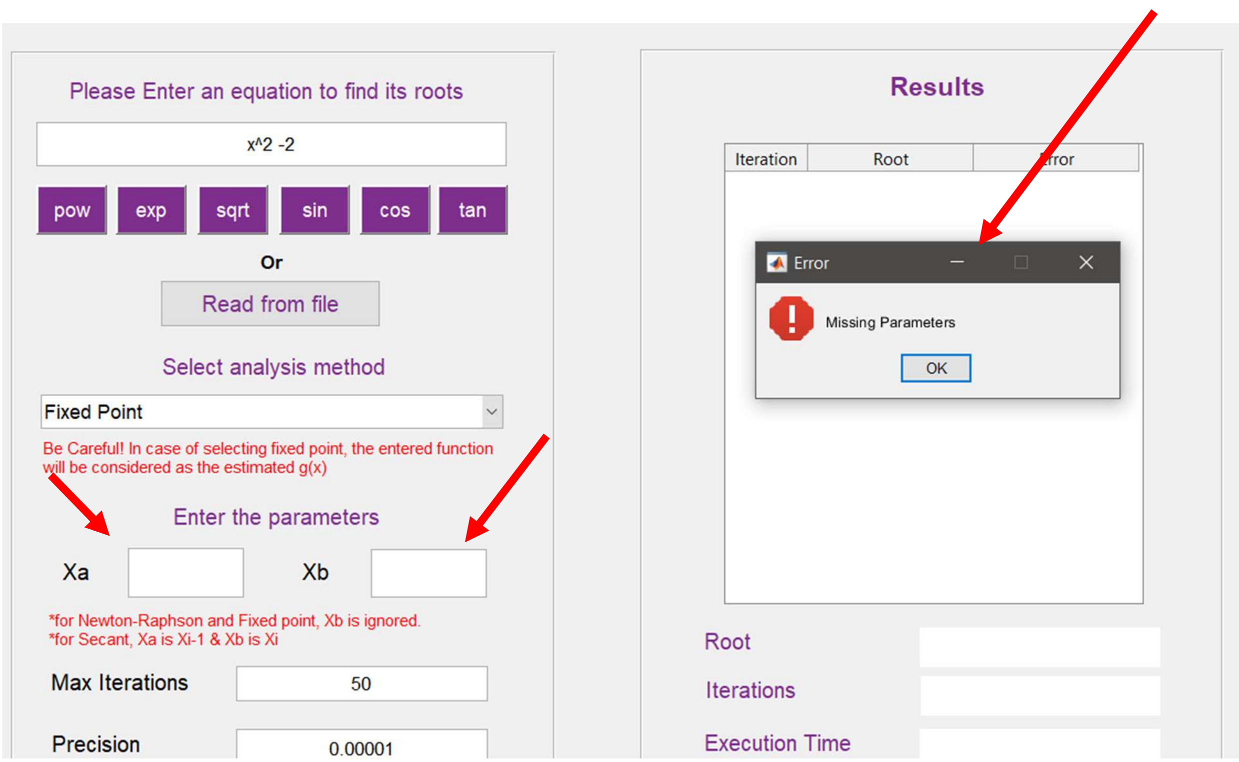


Example 3: Empty equation field

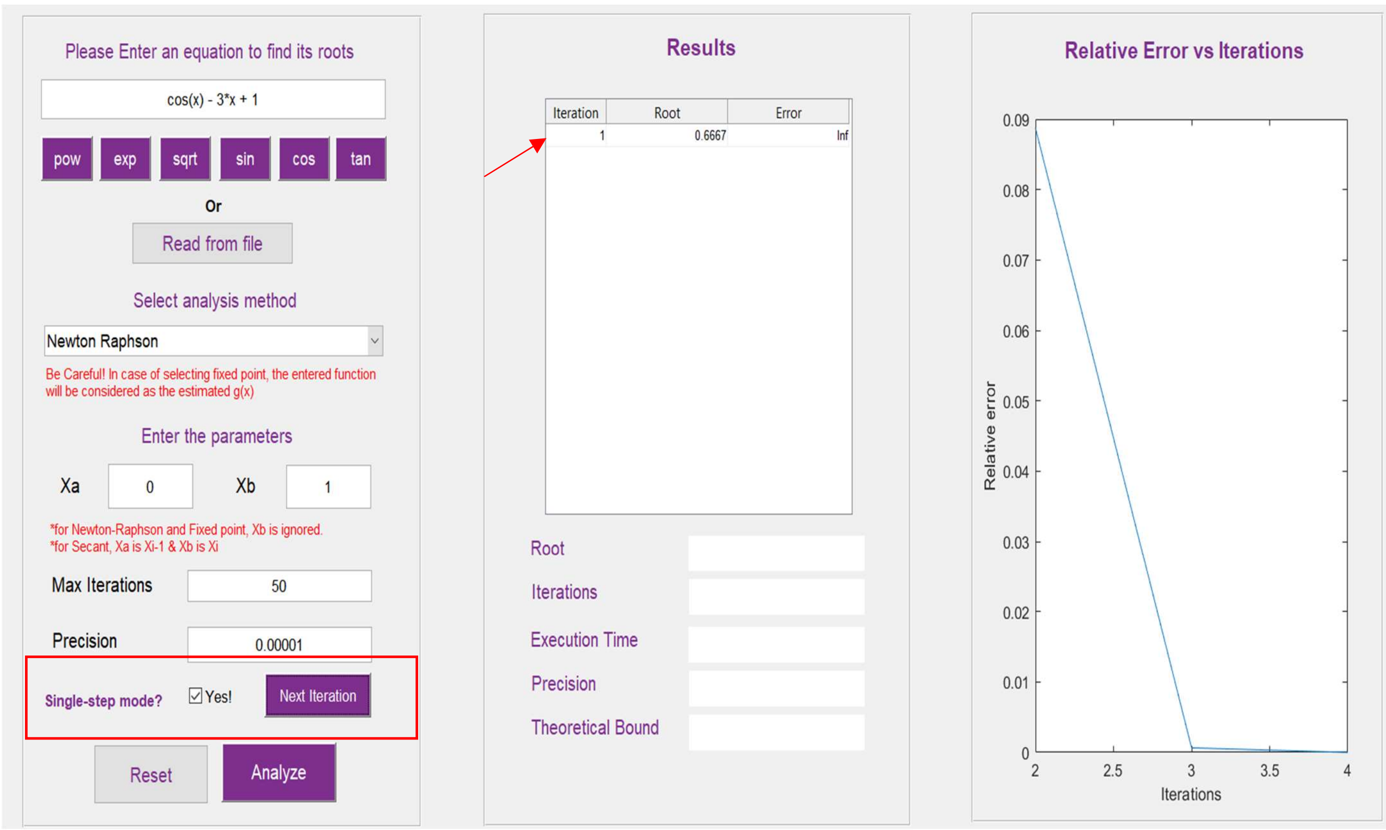


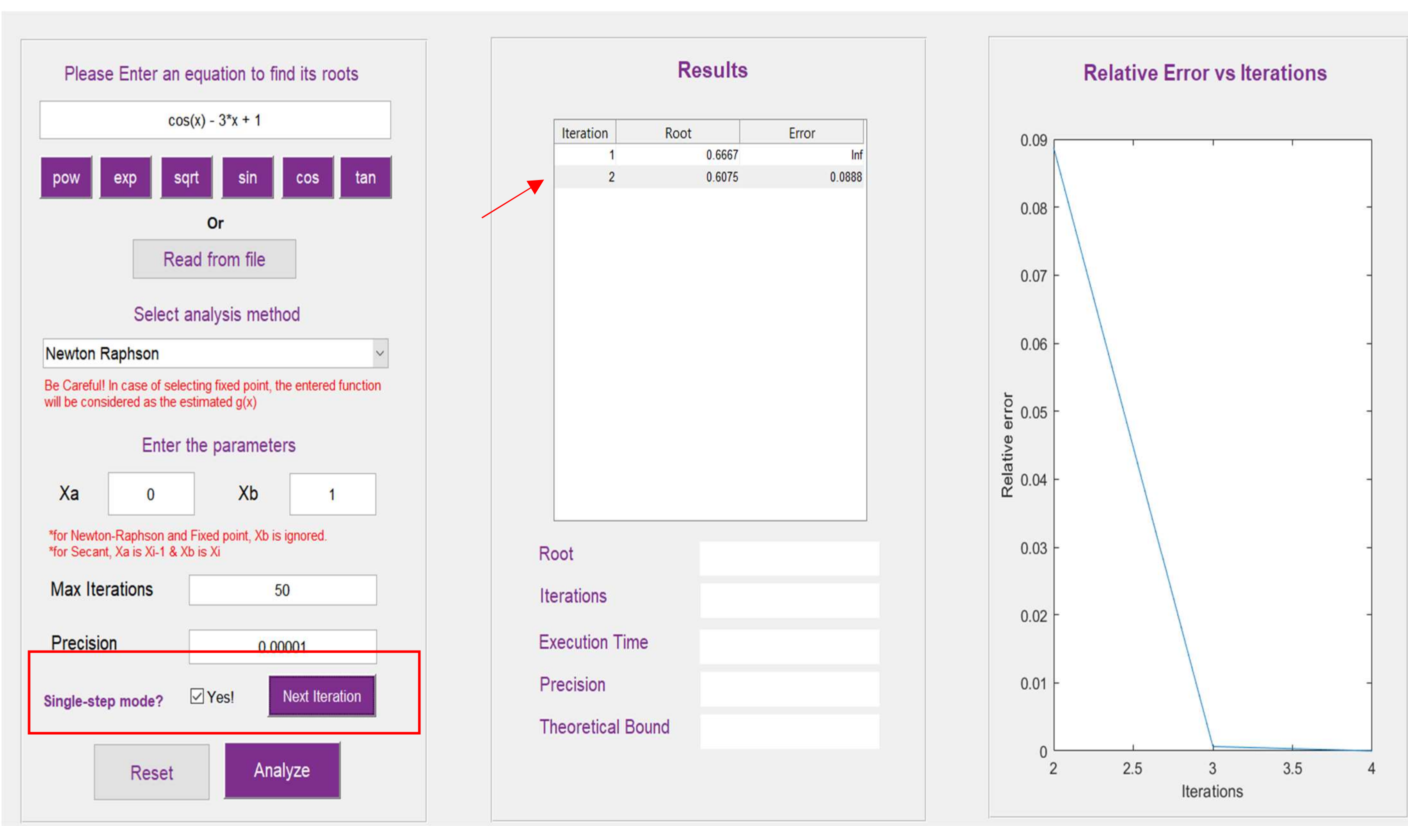
Example 4: missing Parameters

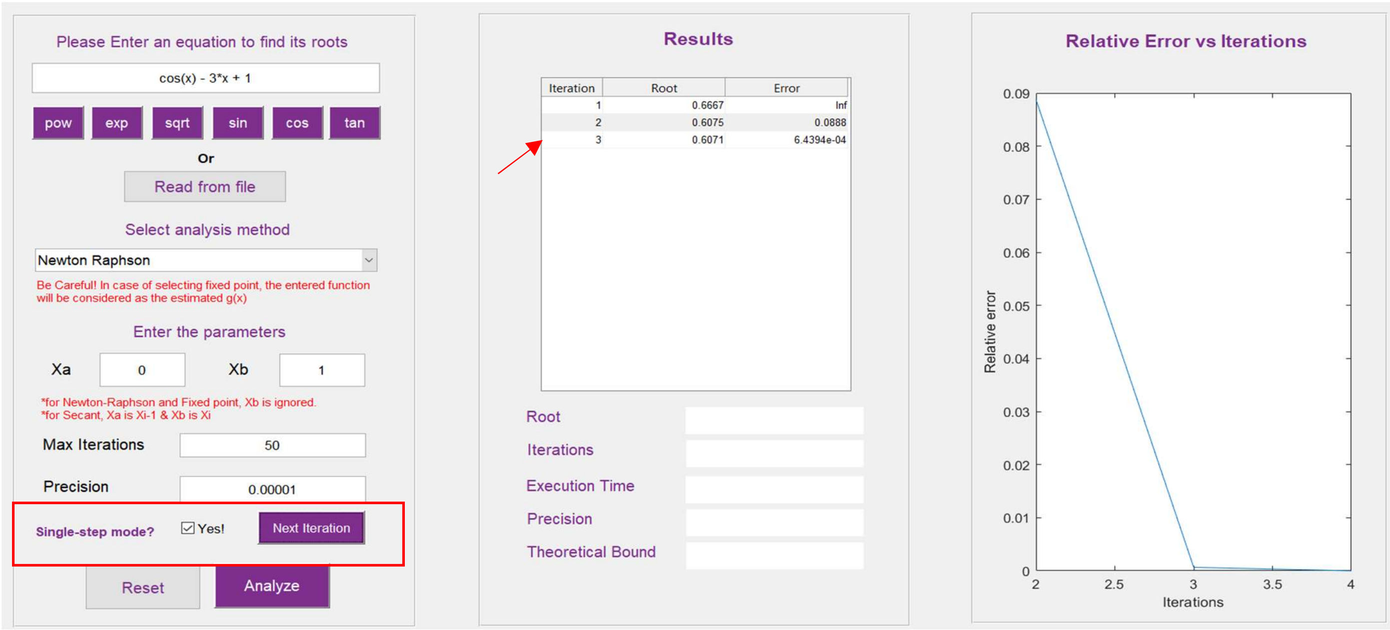
If `max iterations` or `precision` is missing, the program sets it explicitly by the defaults.
If `xb` is missing in case of Newton-Raphson and Fixed point, no error.

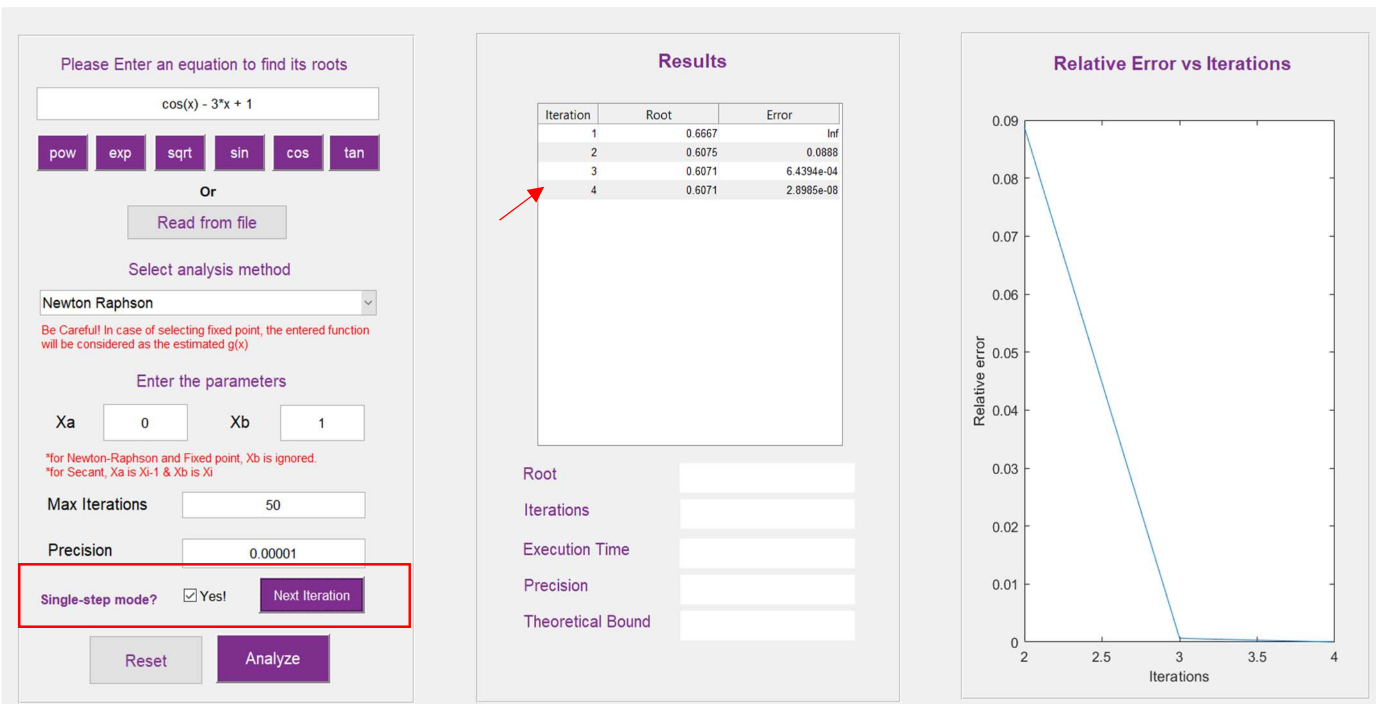


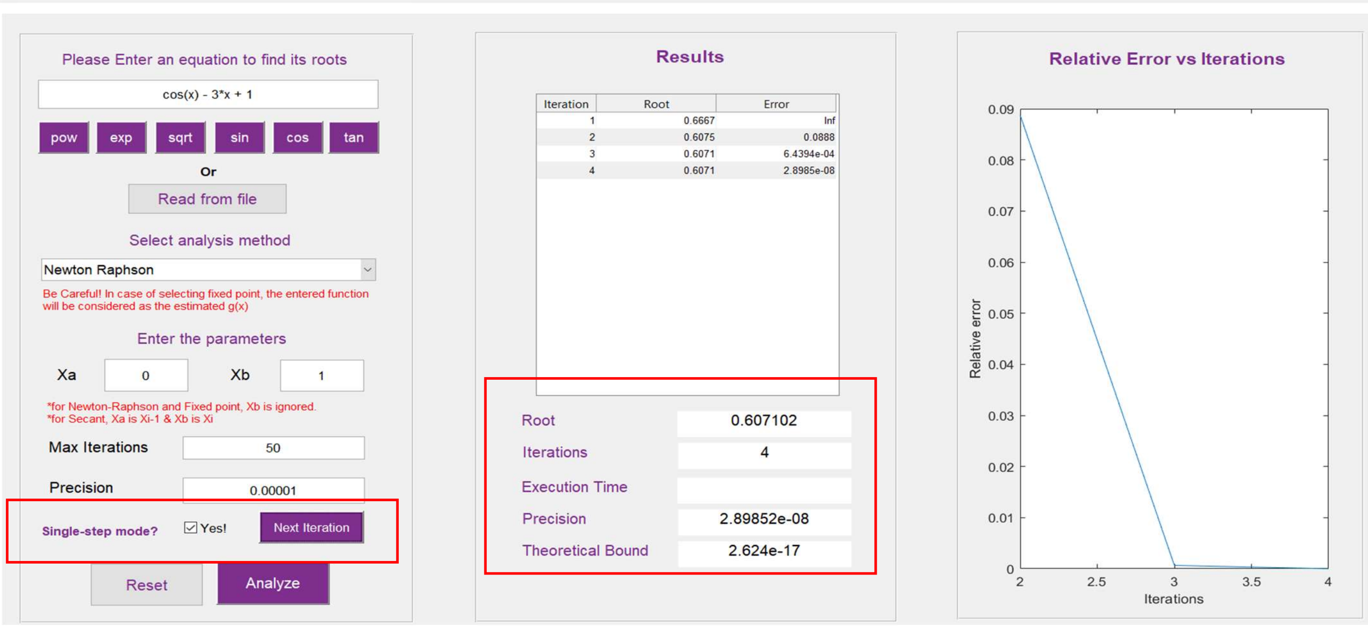
Single-step mode simulation






Please Enter an equation to find its roots
cos(x) - 3*x + 1
pow
exp
sqrt
sin
cos
tan
Or
Read from file
Select analysis method
Newton Raphson
Be Careful! In case of selecting fixed point, the entered function will be considered as the estimated g(x)
Enter the parameters
Xa
0
Xb
1
*for Newton-Raphson and Fixed point, Xb is ignored.
*for Secant, Xa is Xi-1 & Xb is Xi
Max Iterations
50
Precision
0.00001
Single-step mode?
Yes!
Next Iteration
Reset
Analyze
Results
Iteration Root Error
1 0.6667 Inf
2 0.6075 0.0888
3 0.6071 6.4394e-04
Root
Iterations
Execution Time
Precision
Theoretical Bound
Relative Error vs Iterations
Relative error
Iterations


Please Enter an equation to find its roots
cos(x) - 3*x + 1
pow
exp
sqrt
sin
cos
tan
Or
Read from file
Select analysis method
Newton Raphson
Be Careful! In case of selecting fixed point, the entered function will be considered as the estimated g(x)
Enter the parameters
Xa
0
Xb
1
*for Newton-Raphson and Fixed point, Xb is ignored.
*for Secant, Xa is Xi-1 & Xb is Xi
Max Iterations
50
Precision
0.00001
Single-step mode?
Yes!
Next Iteration
Reset
Analyze
Results
Iteration Root Error
1 0.6667 Inf
2 0.6075 0.0888
3 0.6071 6.4394e-04
4 0.6071 2.8985e-08
Root
Iterations
Execution Time
Precision
Theoretical Bound
Relative Error vs Iterations
Relative error
Iterations


Please Enter an equation to find its roots
cos(x) - 3*x + 1
pow
exp
sqrt
sin
cos
tan
Or
Read from file
Select analysis method
Newton Raphson
Be Careful! In case of selecting fixed point, the entered function will be considered as the estimated g(x)
Enter the parameters
Xa
0
Xb
1
*for Newton-Raphson and Fixed point, Xb is ignored.
*for Secant, Xa is Xi-1 & Xb is Xi
Max Iterations
50
Precision
0.00001
Single-step mode?
Yes!
Next Iteration
Reset
Analyze
Results
Iteration Root Error
1 0.6667 Inf
2 0.6075 0.0888
3 0.6071 6.4394e-04
4 0.6071 2.8985e-08
Root
0.607102
Iterations
4
Execution Time
Precision
2.89852e-08
Theoretical Bound
2.624e-17
Relative Error vs Iterations
Relative error
Iterations