# Numerical Project


| | |
|---|---|
| **Fares Mohamed Elsayed Ramadan** | **6537** |
| **Aly ahmed Aly Aboelnasr** | **6511** |
| **Maged Magdy Mohamed Zearban** | **6395** |

# Bisection:

```
1. Start

2. Define function f(x)

3. Input
        a. Lower and Upper guesses x0 and x1
        b. tolerable error e

4. If f(x0)*f(x1) > 0
        print "Incorrect initial guesses"
        goto 3
    End If

5. Do
        x2 = (x0+x1)/2

        If f(x0)*f(x2) < 0
                x1 = x2
        Else
                x0 = x2
        End If

    while abs(f(x2) > e

6. Print root as x2

7. Stop
```

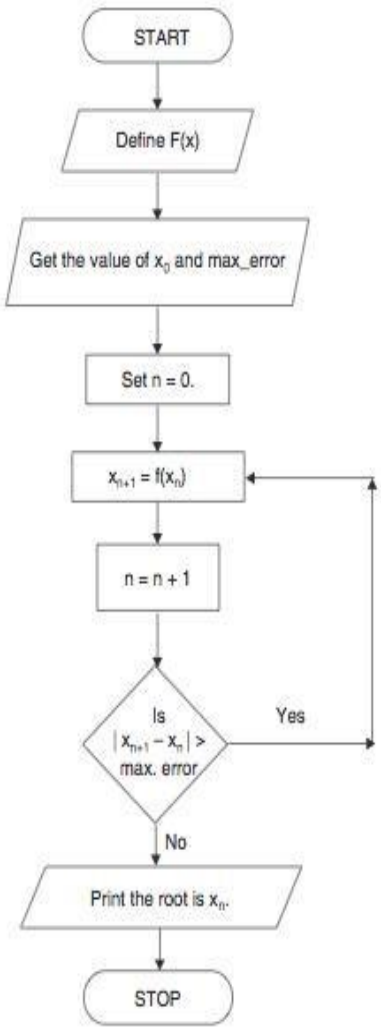
```python
def Bisection(user_input, a, b, Iterations, Epsilon, singlemod):
    x = var('x')
    e = var('e')
    expr = sympify(user_input)
    expr = expr.subs(e, math.e)
    try:
        t = symbols('x')
        expr = sympify(expr)
        if singlemod == 1:
            lam_x = lambdify(t, expr, modules=['numpy'])
            x_vals = linspace(-10, 10, 10000)
            y_vals = lam_x(x_vals)
            axis = mpl.gca()
            axis.set_ylim([min(y_vals), max(y_vals)])
            mpl.plot(x_vals, y_vals)

        expr = sympify(expr)
        fa = expr.subs(x, a)
        fb = expr.subs(x, b)
        Allist = []
        err = "INF"
        oldmid = 0
        start_time = time.time()
        count = 0
        FinalP = 0
        if fa * fb > 0:
            return "No zero in that interval"
        for i in range(Iterations):
            if not isinstance(err, str) and err < Epsilon:
                break
            count = count + 1
            mid = a + (b - a) / 2
            if i > 0:
                err = abs(mid - oldmid)
            fa = expr.subs(x, a)
            fb = expr.subs(x, b)
            fmid = expr.subs(x, mid)
            FinalP = fmid
            templist = [a, b, fa, fb, mid, fmid, err]
            Allist.append(templist)
```

## False Position :

```
1. Start

2. Define function f(x)

3. Input
        a. Lower and Upper guesses a and b
        b. tolerable error e

4. If f(a)*f(b) > 0
        print "Incorrect initial guesses"
        goto 3
   End If

5. Do
        c = b -(f(b)*(b-a))/(f(b)-f(a))

        If f(a)*f(c) < 0
                b = c
        Else
                a = c
        End If

   while (fabs(f(c)) > e)   // fabs -> returns absolute value

6. Print root as c

7. Stop
```

```
def FalseReguli(user_input, xl, xu, Iterations, Epsilon, singlemod):
    x = var('x')
    e = var('e')
    expr = sympify(user_input)
    expr = expr.subs(e, math.e)
    try:
        t = symbols('x')
        expr = sympify(expr)
        if singlemod == 1:
            lam_x = lambdify(t, expr, modules=['numpy'])
            x_vals = linspace(-10, 10, 10000)
            y_vals = lam_x(x_vals)
            axis = mpl.gca()
            axis.set_ylim([min(y_vals), max(y_vals)])
            mpl.plot(x_vals, y_vals)

        expr = sympify(expr)

        fxl = expr.subs(x, xl)
        fxu = expr.subs(x, xu)
        Allist = []
        err = "INF"
        oldmid = 0
        start_time = time.time()
        count = 0
        Fp = 0

        if fxl * fxu > 0:
            return ("No zero in that interval")
        for i in range(Iterations):
            if not isinstance(err, str) and err < Epsilon:
                break

            xr = xl - (xu - xl) * fxl / (fxu - fxl) * 1.0
            if i > 0:
                err = abs(xr - oldmid)
            fxl = expr.subs(x, xl)
            fxu = expr.subs(x, xu)
            fxr = expr.subs(x, xr)
            templist = [xl, xu, fxl, fxu, xr, fxr, err]
            Allist.append(templist)
            oldmid = xr
            Fp = xr
            count = count + 1
            if fxl * fxr > 0:
                xl = xr
            else:
                xu = xr
            if singlemod == 1:
                mpl.scatter(xr, fxr)
                mpl.pause(0.5)

        executiontime = time.time() - start_time
        show(Allist, executiontime, count, Fp)
        mpl.show()

        return ""
    except:
        return "Wrong function"
```

# Fixed Point Flow chart:




```
def FixedPoint(user_input, xr, iterations, epsilon):
    x = var('x')
    e = var('e')
    expr = sympify(user_input)
    expr = expr.subs(e, math.e)
    try:
        t = symbols('x')
        expr = sympify(expr)
        gxdiff = diff(expr, x)
        if gxdiff.subs(x, xr) >= 1:
            return "No Convergance"

        xr = expr.subs(x, xr)
        xrold = 0
        Alllist = []
        err = "INF"
        start_time = time.time()
        FP = 0
        count = 0

        for i in range(iterations):
            if not isinstance(err, str) and err < epsilon:
                break
            count = count + 1
            xr = expr.subs(x, xr)
            if i > 0:
                err = abs(xr - xrold)
            xrold = xr
            FP = xr
            templist = [xr, err]
            Alllist.append(templist)

        execution = time.time() - start_time
        show(Alllist, execution, count, FP)
        return ""
    except:
        return "Wrong function"
```

# Newton Raphson:

1. Start
2. Read x, e, n, d

   *x is the initial guess

   e is the absolute error i.e the desired degree of accuracy

   n is for operating loop

   d is for checking slope*
3. Do for i =1 to n in step of 2
4. f = f(x)
5. f1 = f'(x)
6. If ( [f1] < d), then display too small slope and goto 11.

   *[ ] is used as modulus sign*
7. x1 = x - f/f1
8. If ( [(x1 - x)/x1] < e ), the display the root as x1 and goto 11.

   *[ ] is used as modulus sign*
9. x = x1 and end loop
10. Display method does not converge due to oscillation.
11. Stop

```
def NewtonRaphson(user_input, xi, iterations, epsilon, singlemod):
    x = var('x')
    e = var('e')
    expr = sympify(user_input)
    fxdiff = diff(expr, x)
    expr = expr.subs(e, math.e)
    fxdiff = fxdiff.subs(e, math.e)
    try:
        t = symbols('x')
        expr = sympify(expr)
        if singlemod == 1:
            lam_x = lambdify(t, expr, modules=['numpy'])
            x_vals = linspace(-10, 10, 10000)
            y_vals = lam_x(x_vals)
            axis = mpl.gca()
            axis.set_ylim([min(y_vals), max(y_vals)])
            mpl.plot(x_vals, y_vals)

        expr = sympify(expr)
        fxi = expr.subs(x, xi)
        fxidiff = fxdiff.subs(x, xi)
        Alllist = []
        err = "INF"
        start_time = time.time()
        count = 0
        FP = 0
        for i in range(iterations):
            if not isinstance(err, str) and err < epsilon:
                break
            count = count + 1
            if fxdiff == 0:
                return "Change Xi value"
            xnext = xi - (fxi / fxidiff)
            if i > 0:
                err = abs(xnext - xi)
            xi = xnext
            FP = xnext
            fxi = expr.subs(x, xi)
            fxidiff = fxdiff.subs(x, xi)
            templist = [xi, xnext, fxi, fxidiff, err]
            Alllist.append(templist)
            if singlemod == 1:
                mpl.scatter(xi, fxi)
                mpl.pause(0.5)

        executionTime = time.time() - start_time
        show(Alllist, executionTime, count, FP)
        mpl.show()
        return ""
    except:
        return "Wrong function"
```

# Secant Method:

1. Start
2. Get values of x0, x1 and e

   *Here x0 and x1 are the two initial guesses

   e is the stopping criteria, absolute error or the desired degree of accuracy*
3. Compute f(x0) and f(x1)
4. Compute x2 = [x0*f(x1) - x1*f(x0)] / [f(x1) - f(x0)]
5. Test for accuracy of x2

   If [ (x2 - x1)/x2 ] > e, *Here [ ] is used as modulus sign*

   then assign x0 = x1 and x1 = x2

   goto step 4

   Else,

   goto step 6
6. Display the required root as x2.
7. Stop

```
def Secant(user_input, xold, xi, Iterations, epsilon, singlemod):
    x = var('x')
    e = var('e')
    expr = sympify(user_input)
    expr = expr.subs(e, math.e)
    try:
        t = symbols('x')
        expr = sympify(expr)
        if singlemod == 1:
            lam_x = lambdify(t, expr, modules=['numpy'])
            x_vals = linspace(-10, 10, 10000)
            y_vals = lam_x(x_vals)
            axis = mpl.gca()
            axis.set_ylim([min(y_vals), max(y_vals)])
            mpl.plot(x_vals, y_vals)

        expr = sympify(expr)

        fxi = expr.subs(x, xi)
        fxold = expr.subs(x, xold)
        Alllist = []
        err = "INF"
        start_time = time.time()
        count = 0
        FP = 0
        for i in range(Iterations):
            if not isinstance(err, str) and err < epsilon:
                break
            count = count + 1
            xnext = xi - fxi * (xold - xi) / (fxold - fxi)
            FP = xnext
            if i > 0:
                err = abs(xnext - xi)

            templist = [xold, xi, xnext, fxi, fxold, err]
            Alllist.append(templist)
            xold = xi
            xi = xnext
            fxi = expr.subs(x, xi)
            fxold = expr.subs(x, xold)
            if singlemod == 1:
                mpl.scatter(xi, fxi)
                mpl.pause(0.5)

        executionTime = time.time() - start_time
        show(Alllist, executionTime, count, FP)
        mpl.show()
        return ""
    except:
        return "Wrong function"
```

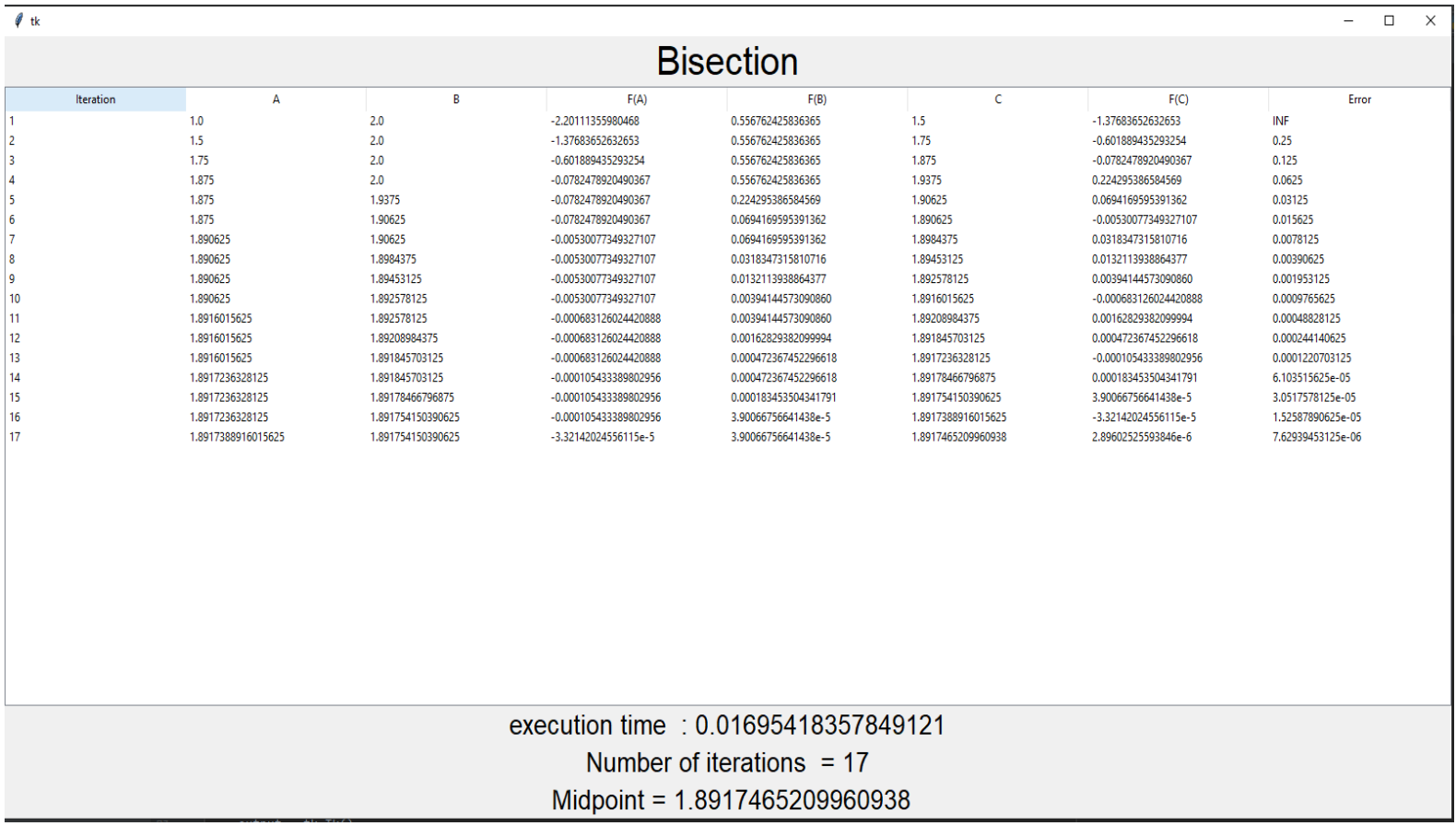
**Data Structure Used :**

We used list of lists data structure in our project to save the value of each variable in all iterations and then show them to the user in a table.

**Analysis :**

Function used (e^x+2*cos(x)-6) with epsilon (10^(-5))

**Bisection Method :**




| Iteration | A | B | F(A) | F(B) | C | F(C) | Error |
|---|---|---|---|---|---|---|---|
| 1 | 1.0 | 2.0 | -2.20111355980468 | 0.556762425836365 | 1.5 | -1.37683652632653 | INF |
| 2 | 1.5 | 2.0 | -1.37683652632653 | 0.556762425836365 | 1.75 | -0.601889435293254 | 0.25 |
| 3 | 1.75 | 2.0 | -0.601889435293254 | 0.556762425836365 | 1.875 | -0.0782478920490367 | 0.125 |
| 4 | 1.875 | 2.0 | -0.0782478920490367 | 0.556762425836365 | 1.9375 | 0.224295386584569 | 0.0625 |
| 5 | 1.875 | 1.9375 | -0.0782478920490367 | 0.224295386584569 | 1.90625 | 0.0694169595391362 | 0.03125 |
| 6 | 1.875 | 1.90625 | -0.0782478920490367 | 0.0694169595391362 | 1.890625 | -0.00530077349327107 | 0.015625 |
| 7 | 1.890625 | 1.90625 | -0.00530077349327107 | 0.0694169595391362 | 1.8984375 | 0.0318347315810716 | 0.0078125 |
| 8 | 1.890625 | 1.8984375 | -0.00530077349327107 | 0.0318347315810716 | 1.89453125 | 0.0132113938864377 | 0.00390625 |
| 9 | 1.890625 | 1.89453125 | -0.00530077349327107 | 0.0132113938864377 | 1.892578125 | 0.00394144573090860 | 0.001953125 |
| 10 | 1.890625 | 1.892578125 | -0.00530077349327107 | 0.00394144573090860 | 1.8916015625 | -0.000683126024420888 | 0.0009765625 |
| 11 | 1.8916015625 | 1.892578125 | -0.000683126024420888 | 0.00394144573090860 | 1.89208984375 | 0.00162829382099994 | 0.00048828125 |
| 12 | 1.8916015625 | 1.89208984375 | -0.000683126024420888 | 0.00162829382099994 | 1.891845703125 | 0.000472367452296618 | 0.000244140625 |
| 13 | 1.8916015625 | 1.891845703125 | -0.000683126024420888 | 0.000472367452296618 | 1.8917236328125 | -0.000105433389802956 | 0.0001220703125 |
| 14 | 1.8917236328125 | 1.891845703125 | -0.000105433389802956 | 0.000472367452296618 | 1.89178466796875 | 0.000183453504341791 | 6.103515625e-05 |
| 15 | 1.8917236328125 | 1.89178466796875 | -0.000105433389802956 | 0.000183453504341791 | 1.8917541503906 25 | 3.90066756641438e-5 | 3.0517578125e-05 |
| 16 | 1.8917236328125 | 1.8917541503906 25 | -0.000105433389802956 | 3.90066756641438e-5 | 1.8917388916015625 | -3.32142024556115e-5 | 1.52587890625e-05 |
| 17 | 1.8917388916015625 | 1.8917541503906 25 | -3.32142024556115e-5 | 3.90066756641438e-5 | 1.8917465209960938 | 2.89602525593846e-6 | 7.62939453125e-06 |

execution time : 0.01695418357849121

Number of iterations = 17

Midpoint = 1.8917465209960938

# False Position Method :

tk

## False Position

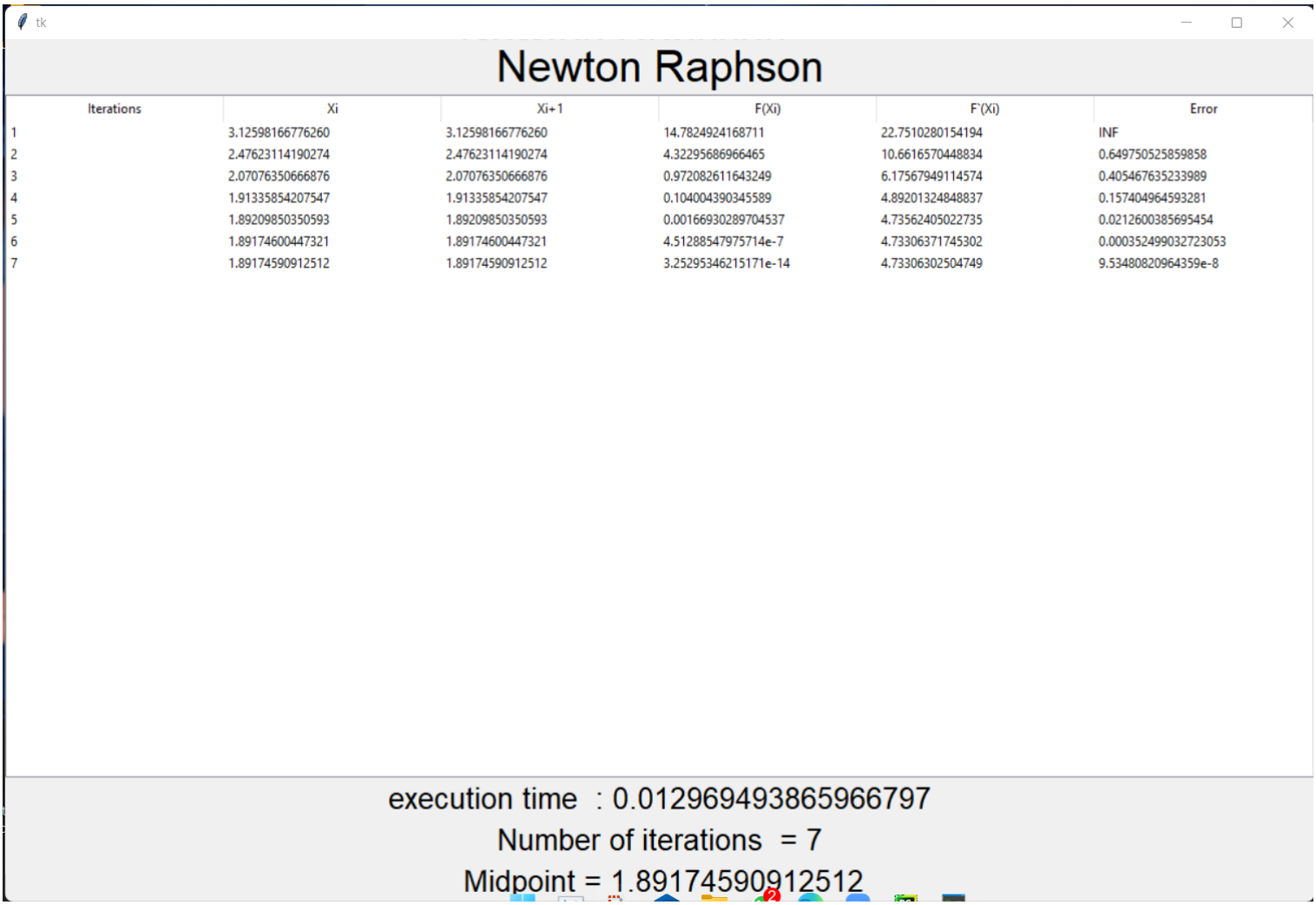
| Iterations | Xl | Xu | F(Xl) | F(Xu) | Xr | F(Xr) | Error |
|---|---|---|---|---|---|---|---|
| 1 | 1.0 | 2.0 | -2.20111355980468 | 0.556762425836365 | 1.79811912184044 | -0.412460501448479 | INF |
| 2 | 1.79811912184044 | 2.0 | -0.412460501448479 | 0.556762425836365 | 1.95924411103353 | 0.336458191269945 | 0.161124989193083 |
| 3 | 1.79811912184044 | 1.95924411103353 | -0.412460501448479 | 0.336458191269945 | 1.86668713851400 | -0.116346992985858 | 0.0925569725195270 |
| 4 | 1.86668713851400 | 1.95924411103353 | -0.116346992985858 | 0.336458191269945 | 1.91766209169535 | 0.125126494104946 | 0.0509749531813468 |
| 5 | 1.86668713851400 | 1.91766209169535 | -0.116346992985858 | 0.125126494104946 | 1.87978500648561 | -0.0560946800527781 | 0.0378770852097334 |
| 6 | 1.87978500648561 | 1.91766209169535 | -0.0560946800527781 | 0.125126494104946 | 1.89803498092510 | 0.0299105394816658 | 0.0182499744394895 |
| 7 | 1.87978500648561 | 1.89803498092510 | -0.0560946800527781 | 0.0299105394816658 | 1.88543405098559 | -0.0297301245762123 | 0.0126009299395153 |
| 8 | 1.88543405098559 | 1.89803498092510 | -0.0297301245762123 | 0.0299105394816658 | 1.89365268163557 | 0.00903808561231711 | 0.00821863064998629 |
| 9 | 1.88543405098559 | 1.89365268163557 | -0.0297301245762123 | 0.00903808561231711 | 1.88953093550104 | -0.0104658114397982 | 0.00412174613453464 |
| 10 | 1.88953093550104 | 1.89365268163557 | -0.0104658114397982 | 0.00903808561231711 | 1.89269177334873 | 0.00448008463626692 | 0.00316083784769350 |
| 11 | 1.88953093550104 | 1.89269177334873 | -0.0104658114397982 | 0.00448008463626692 | 1.89122704437699 | -0.00245484223014891 | 0.00146472897174199 |
| 12 | 1.89122704437699 | 1.89269177334873 | -0.00245484223014891 | 0.00448008463626692 | 1.89225271571517 | 0.00239968033370219 | 0.00102567133818154 |
| 13 | 1.89122704437699 | 1.89225271571517 | -0.00245484223014891 | 0.00239968033370219 | 1.89159011400465 | -0.000737299999048147 | 0.000662601710524013 |
| 14 | 1.89159011400465 | 1.89225271571517 | -0.000737299999048147 | 0.00239968033370219 | 1.89192517942856 | 0.000848614343352261 | 0.000335065423915371 |
| 15 | 1.89159011400465 | 1.89192517942856 | -0.000737299999048147 | 0.000848614343352261 | 1.89166886608803 | -0.000364627998934330 | 0.000256313340532888 |
| 16 | 1.89166886608803 | 1.89192517942856 | -0.000364627998934330 | 0.000848614343352261 | 1.89178802752108 | 0.000199355463832762 | 0.000119161433047354 |
| 17 | 1.89166886608803 | 1.89178802752108 | -0.000364627998934330 | 0.000199355463832762 | 1.89170467887940 | -0.000195139179280490 | 8.33486416800255e-5 |
| 18 | 1.89170467887940 | 1.89178802752108 | -0.000195139179280490 | 0.000199355463832762 | 1.89175856565168 | 5.99047195383173e-5 | 5.38867722832403e-5 |
| 19 | 1.89170467887940 | 1.89175856565168 | -0.000195139179280490 | 5.99047195383173e-5 | 1.89173133429970 | -6.89827960003164e-5 | 2.72313519853462e-5 |
| 20 | 1.89173133429970 | 1.89175856565168 | -6.89827960003164e-5 | 5.99047195383173e-5 | 1.89175216955081 | 2.96311316758668e-5 | 2.08352511146259e-5 |
| 21 | 1.89173133429970 | 1.89175216955081 | -6.89827960003164e-5 | 2.96311316758668e-5 | 1.89174248568136 | -1.62033325220579e-5 | 9.68386945254807e-6 |

execution time: 0.050863027572631836

Number of iterations = 21

Midpoint = 1.89174248568136

## Newton-Raphson Method :



## Secant Method :

tk

Secant

| Iterations | Xold | Xi | Xi+1 | F(Xi) | F(Xold) | Error |
|---|---|---|---|---|---|---|
| 1 | 1.0 | 2.0 | 1.79811912184044 | 0.556762425836365 | -2.20111355980468 | INF |
| 2 | 2.0 | 1.79811912184044 | 1.88403113020762 | -0.412460501448479 | 0.556762425836365 | 0.0859120083671714 |
| 3 | 1.79811912184044 | 1.88403113020762 | 1.89232152579108 | -0.0362990812989470 | -0.412460501448479 | 0.00829039558346012 |
| 4 | 1.88403113020762 | 1.89232152579108 | 1.89174249328873 | 0.00272563328377695 | -0.0362990812989470 | 0.000579032502340482 |
| 5 | 1.89232152579108 | 1.89174249328873 | 1.89174590761708 | -1.61673265142159e-5 | 0.00272563328377695 | 3.41432834050082e-6 |

execution time : 0.008005142211914062

Number of iterations = 5

Midpoint = 1.89174590761708

# Secant is the best in this guess.

## Disadvantages of Bisection-Method :

1-Slow convergence

2-If f(x) touches the x-axis (ex: x^2), It's unable to find the upper and lower guesses.

## Disadvantages of False Position :

1-It may take large time span.

2-It is used to calculate a single unknown in the equation.

## Disadvantages of Fixed-Point :

If g'(x)>=1 the method will converge.

# Disadvantages of Newton-Raphson :

1-Divition by zero problem can occurs.

2-Convergence is not guaranteed.

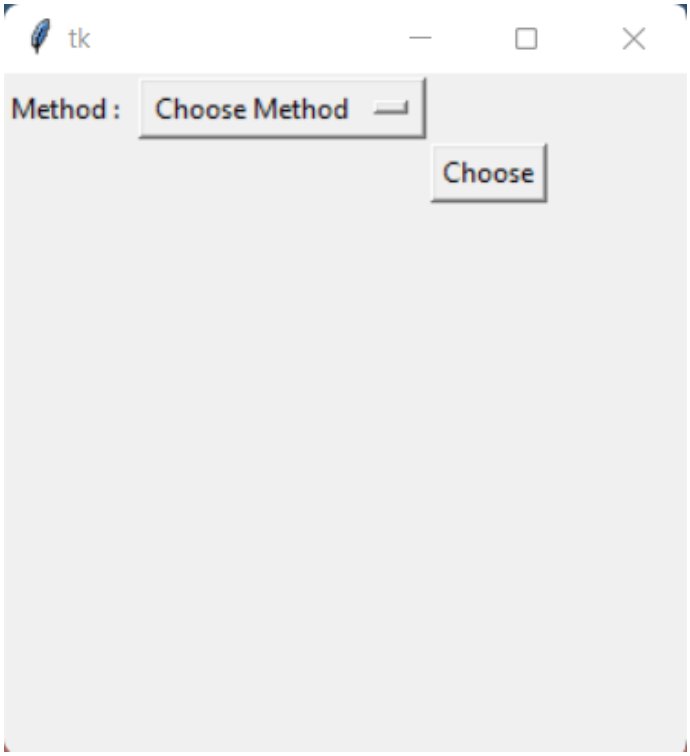
3-In case of multiple roots the method converges slowly.

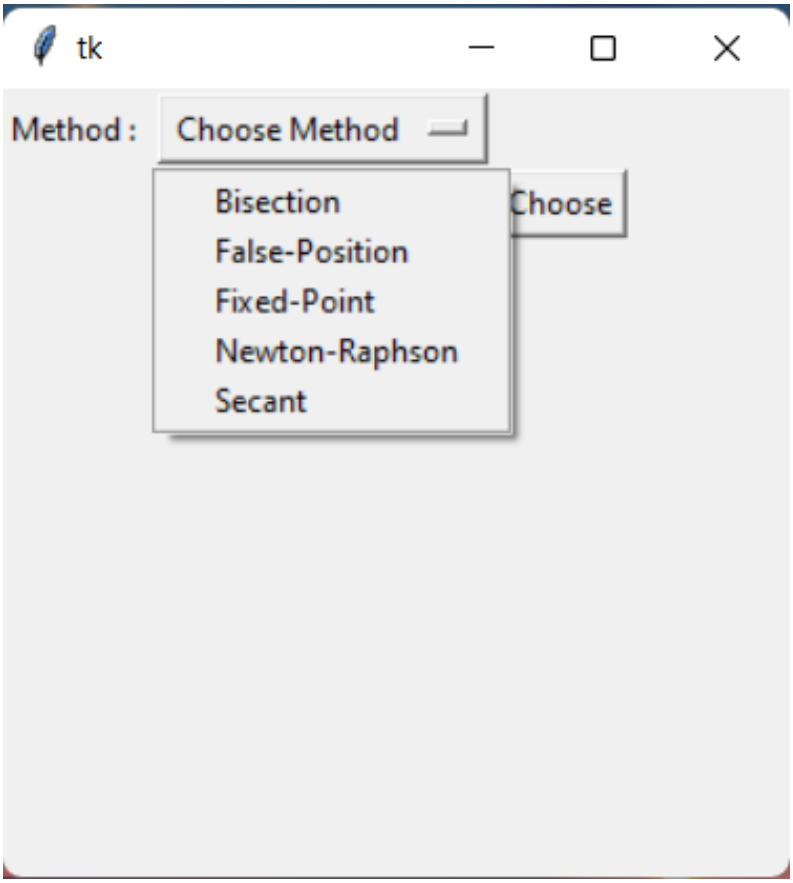
# Disadvantages of Secant :

1-It may not converge.

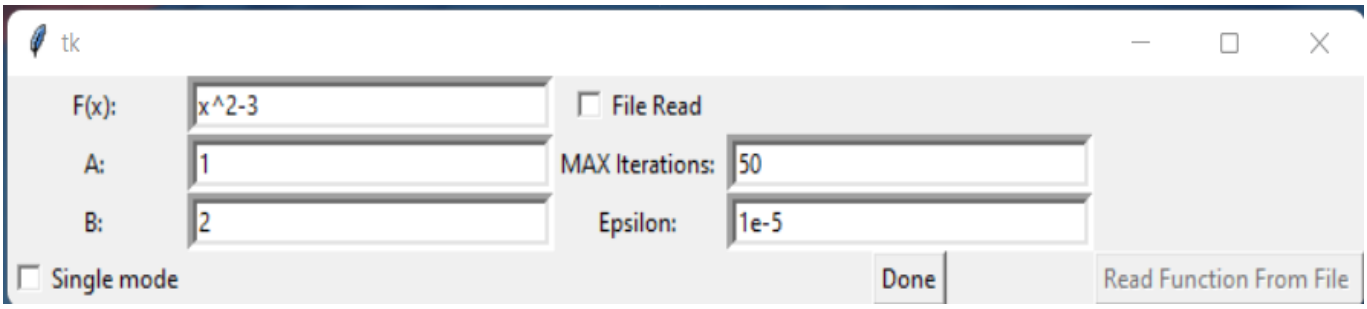
2-There is no guaranteed error bound for the computed iterates.

Graphical User Interface to choose your method for finding the root.


tk
Method :
Choose Method
Choose


tk
Method :
Choose Method
Bisection
False-Position
Fixed-Point
Newton-Raphson
Secant
Choose

The user can type the function or read it from the file (input.txt) and the guess (a, b), the max iterations (default set to 50), the epsilon (default set to 10^(-5)) and choose single mode if the user wants a simulation showing the iterations on the draw function for one method of choice.




**Bonus: Single step mode simulation showing the iterations on the draw function for one method of choice.**

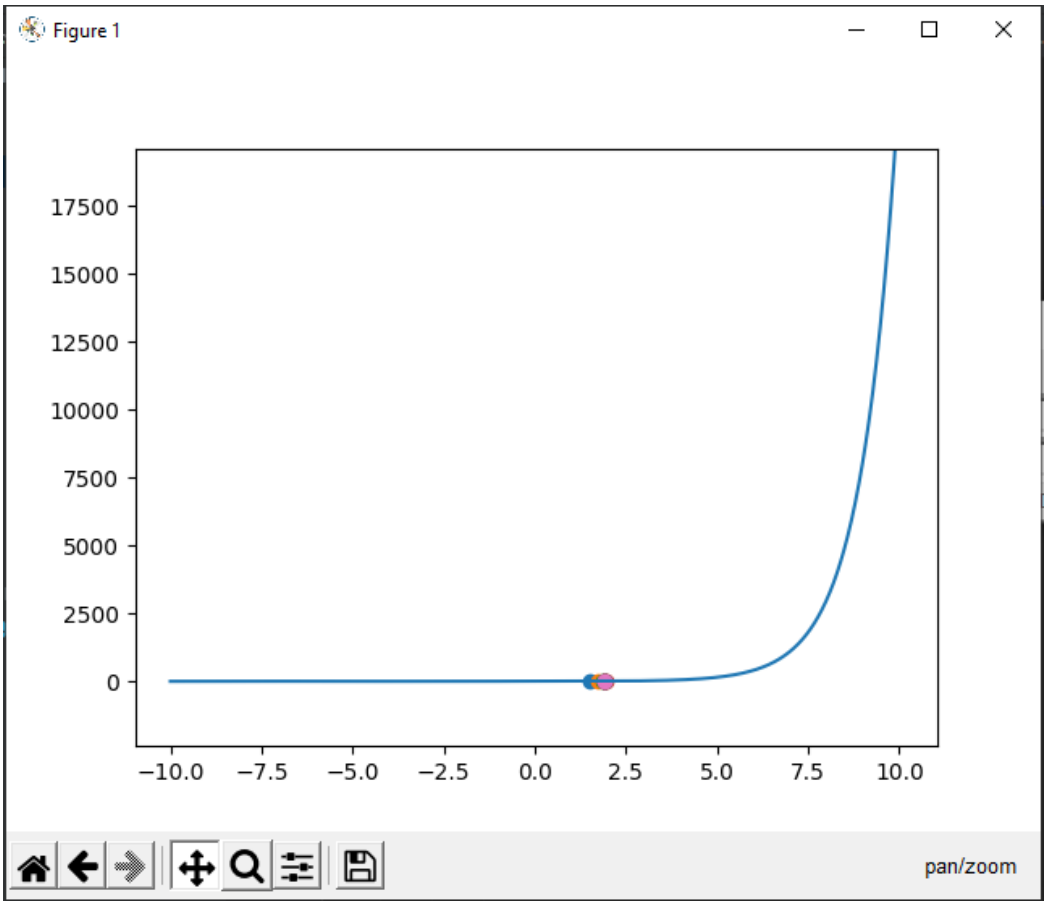
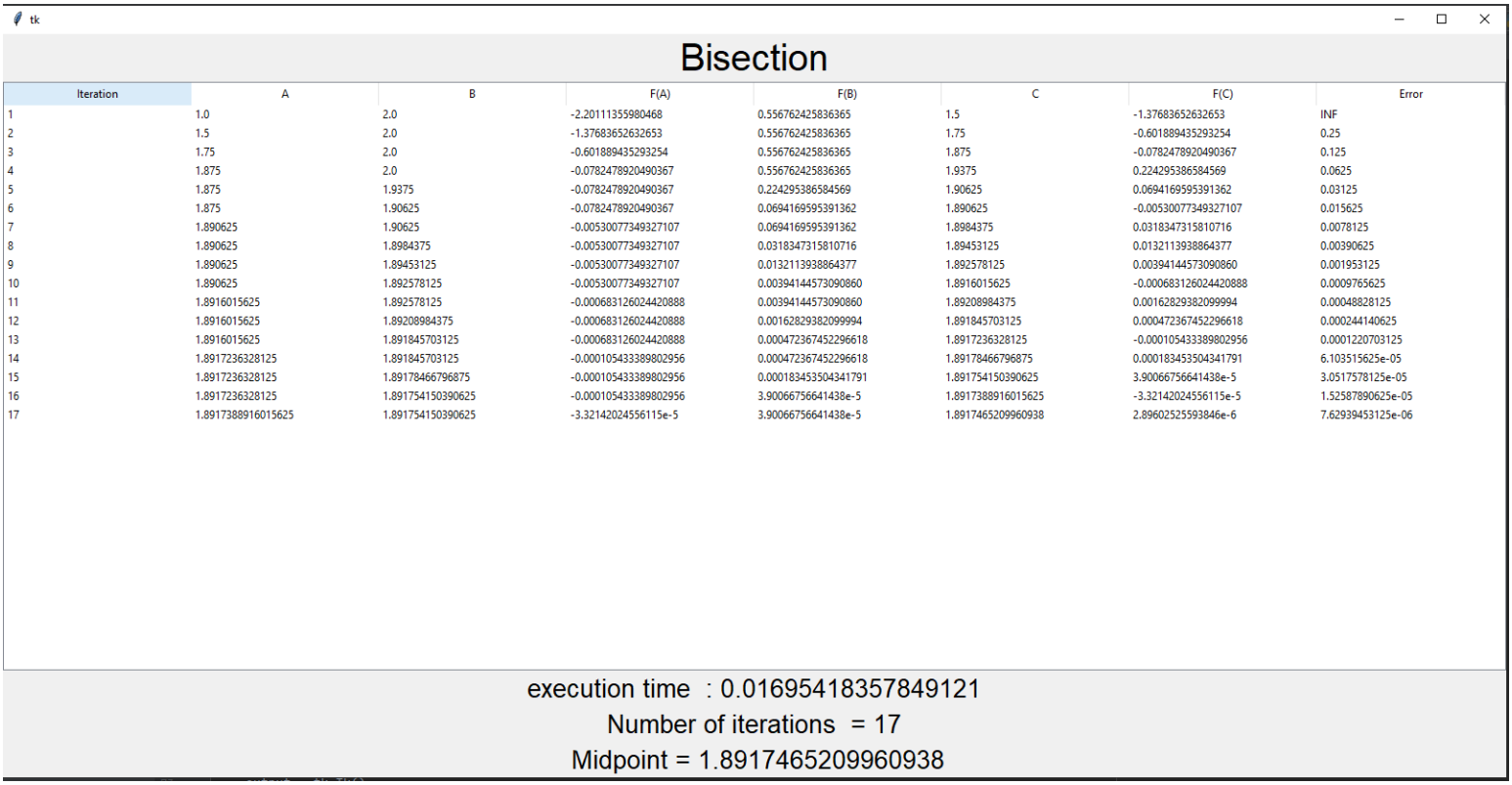


**Table :**

**Shows the iterations step by step (xl, xu, xr, f(xl), f(xu), f(xr), err) and execution time.**




| Iteration | A | B | F(A) | F(B) | C | F(C) | Error |
|---|---|---|---|---|---|---|---|
| 1 | 1.0 | 2.0 | -2.20111355980468 | 0.556762425836365 | 1.5 | -1.37683652632653 | INF |
| 2 | 1.5 | 2.0 | -1.37683652632653 | 0.556762425836365 | 1.75 | -0.601889435293254 | 0.25 |
| 3 | 1.75 | 2.0 | -0.601889435293254 | 0.556762425836365 | 1.875 | -0.0782478920490367 | 0.125 |
| 4 | 1.875 | 2.0 | -0.0782478920490367 | 0.556762425836365 | 1.9375 | 0.224295386584569 | 0.0625 |
| 5 | 1.875 | 1.9375 | -0.0782478920490367 | 0.224295386584569 | 1.90625 | 0.0694169595391362 | 0.03125 |
| 6 | 1.875 | 1.90625 | -0.0782478920490367 | 0.0694169595391362 | 1.890625 | -0.00530077349327107 | 0.015625 |
| 7 | 1.890625 | 1.90625 | -0.00530077349327107 | 0.0694169595391362 | 1.8984375 | 0.0318347315810716 | 0.0078125 |
| 8 | 1.890625 | 1.8984375 | -0.00530077349327107 | 0.0318347315810716 | 1.89453125 | 0.0132113938864377 | 0.00390625 |
| 9 | 1.890625 | 1.89453125 | -0.00530077349327107 | 0.0132113938864377 | 1.892578125 | 0.00394144573090860 | 0.001953125 |
| 10 | 1.890625 | 1.892578125 | -0.00530077349327107 | 0.00394144573090860 | 1.8916015625 | -0.000683126024420888 | 0.0009765625 |
| 11 | 1.8916015625 | 1.892578125 | -0.000683126024420888 | 0.00394144573090860 | 1.89208984375 | 0.00162829382099994 | 0.00048828125 |
| 12 | 1.8916015625 | 1.89208984375 | -0.000683126024420888 | 0.00162829382099994 | 1.891845703125 | 0.000472367452296618 | 0.000244140625 |
| 13 | 1.8916015625 | 1.891845703125 | -0.000683126024420888 | 0.000472367452296618 | 1.8917236328125 | -0.000105433389802956 | 0.0001220703125 |
| 14 | 1.8917236328125 | 1.891845703125 | -0.000105433389802956 | 0.000472367452296618 | 1.89178466796875 | 0.000183453504341791 | 6.103515625e-05 |
| 15 | 1.8917236328125 | 1.89178466796875 | -0.000105433389802956 | 0.000183453504341791 | 1.891754150390625 | 3.90066756641438e-5 | 3.0517578125e-05 |
| 16 | 1.8917236328125 | 1.891754150390625 | -0.000105433389802956 | 3.90066756641438e-5 | 1.8917388916015625 | -3.32142024556115e-5 | 1.5258789062 5e-05 |
| 17 | 1.8917388916015625 | 1.891754150390625 | -3.32142024556115e-5 | 3.90066756641438e-5 | 1.8917465209960938 | 2.89602525593846e-6 | 7.62939453125e-06 |

execution time : 0.01695418357849121
Number of iterations = 17
Midpoint = 1.8917465209960938