Question: Write a program to implement Bisection Method.

Theory

Bisection Method is one of the simplest, reliable, easy to implement & convergence method for finding real roots of linear equation. Also known as Binary Search or Half Interval Method.

Bisection Method is Based on the fact that if $f(x)$ is real & continous function, & for two initial guesses x_0 & x_1 brackets the root such that: $f(x_0) f(x_1) < 0$ then there exists at least one root between x_0 & x_1.

Algorithm

1. Start.
2. Define function $f(x)$.
3. Choose Initial guesses x_1 & x_2 such that $f(x_1) \cdot f(x_2) < 0$.
4. Choose tolerable error as ϵ.
5. Calculate new approximate root as $x_3 = \frac{x_1 + x_2}{2}$.
6. Calculate $f(x_1) \cdot f(x_2)$
   - a) If $f(x_1) \cdot f(x_3) < 0$ then $x_1 = x_1$ & $x_2 = x_3$
   - b) If $f(x_1) \cdot f(x_3) > 0$ then $x_1 = x_3$ & $x_2 = x_2$
   - c) If $f(x_1) \cdot f(x_3) = 0$ then goto step 8.
7. if $|f(x_3)| > e$ then goto step 5 otherwise goto step 8.
8. Display x_3 as root.
9. Stop.

Example:

① $x^3 - 2x - 5$

Here, $f(x) = x^3 - 2x - 5$, error = 0.0001.

Lets initial guesses be,

$x_1 = 2$

$x_2 = 3$

$\therefore f(x_1) \cdot f(x_2) = f(2) \cdot f(3) = -1 \times 16 = -16 < 0$, so the required root lies between 2 & 3.

Using The Bisection Table,

I	x_1	$f(x_1)$	x_2	$f(x_2)$	$x_3 = \frac{x_1+x_2}{2}$	$f(x_3)$
1	2	-1	3	16	2.5	5.62500
2	2	-1	2.5	5.625	2.25	1.890625
3	2	-1	2.25	1.890625	2.125	0.345703
4	2	-1	2.125	0.345703	2.0625	-0.351318
5	2.0625	-0.351318	2.125	0.345703	2.09375	-0.008942
6	2.09375	-0.008942	2.125	0.345703	2.109375	0.166836
7	2.09375	-0.008942	2.109375	0.166836	2.101563	0.078562
8	2.09375	-0.008942	2.101563	0.078562	2.097656	0.034714
9	2.09375	-0.008942	2.097656	0.34714	2.095703	0.012862
10	2.09375	-0.008942	2.095703	0.012862	2.094727	0.001954
11	2.09375	-0.008942	2.094727	0.001954	2.094238	-0.003495
12	2.094238	-0.003495	2.094727	0.001954	2.094482	-0.00771
13	2.094482	-0.00771	2.094727	0.001954	2.094604	0.000592
14	2.094482	-0.00771	2.094604	0.000592	2.094543	-0.000090
15	2.09543	-0.000090	2.94604	0.00592	2.094559	0.000251

Here, The two consecutive roots are similar till the 4th position after decimal point so the required root is 2.0945.

Iteration counter: step = step + 1.

Question: Write a program in C to implement Secant Method.

Theory

Secant Method is open method and starts with two initial guesses for finding real root of non-linear equations. Let initial guesses be x_0 and x_1. Here, x_2 which is the approximate root is obtained by.

$$x_2 = x_1 - (x_1 - x_0) * f(x_1) / (f(x_1) - f(x_0))$$

~~Add an~~

Algorithm

1. Start
2. Define function as f(x)
3. Input initial guesses (x_0 & x_1), error(e) & Iteration(N).
4. Initialize iteration counter i = 1.
5. If $f(x_0) = f(x_1)$ then print "Mathematical Error" & goto (10) Otherwise goto (6).
6. Calculate $x_2 = x_1 - (x_1 - x_0) * f(x_1) / (f(x_1) - f(x_0))$.
7. Increment iteration counter i = i+1.
8. If ~~i >= N~~ $|f(x_2)| > e$ then set $x_0 = x_1$, $x_1 = x_2$ and goto (5) Otherwise goto (9)
9. Print root as x_2.
10. Stop.

Example

Q) $x^3 - 2x - 5$

Here,

$f(x) = x^3 - 2x - 5$ error = 0.00001

Let us consider Initial guesses,

$x_1 = 2$

$x_2 = 3$

I	x_0	$f(x_0)$	x_1	$f(x_1)$	x_2	$f(x_2)$
1	2	-1	3	16	2.058824	-0.390799
2	3	16	2.058824	-0.390799	2.081264	-0.147203
3	2.058824	-0.390799	2.081264	-0.147203	2.094824	0.003043
4	2.081264	-0.147203	2.094824	0.003043	2.094549	-0.000023
5	2.094824	0.003043	2.094549	-0.000023	2.094552	0.000001

Here, $|f(x_2)| <$ error, so the required root of the equation is 2.094552.

Question: Write a program in C to implement Fixed point Iteration Method.

Theory

Fixed point iteration method is open and simple method for finding real root of non-linear equation by successive approximation. It requires only one initial guess to start. Since it is open method its convergence is not guaranteed.

To find the root of nonlinear equation $f(x)=0$ by fixed point iteration method, we write given equation $f(x)=0$ in form of

$$x = g(x)$$

if x_0 is initial guess then next approximated root in this meth is obtained by.

$$x_1 = g(x_0)$$

Similarly, next to next approximated root is obtained by using the value of x_1 i.e,

$$x_2 = g(x_1)$$

And the process is repeated until until we get root within desired accuracy.

Algorithm for fixed point Iteration Method

1. Start
2. Define function $f(x)$
3. Define function $g(x)$ which is obtained from $f(x)=0$ such that $x=g(x)$ and $|g'(x)<1|$
4. Choose Initial guess x_0, Tolerable Error e and Maximum Iteration N
5. Initialize iteration counter: step = ~~step~~ 1
6. Calculate $x_1 = g(x_0)$

(7) Increment iteration counter: step = step + 1

(8) If step > N then print "Not Convergent" and goto (12) otherwise goto(10)

(9) Set $x_0 = x_1$ for next iteration

(10) If $|f(x_1)| > e$ then goto step(6) otherwise goto step(11)

(11) Display x_1 as root.

(12) stop

Example

- $f(x) = \cos x - 3x + 1$, Error: 0.0001

  It can also be written as,

  $x = (1 + \cos x)/3$

  Let initial guess(x_0) = 2

I	x_0	$f(x_0)$	x_1	$f(x_1)$
1	2	-5.416147	0.194618	1.397269
2	0.194618	1.39 7269	0.660374	-0.191359
3	0.660374	-0.191359	0.596588	0.037495
4	0.596588	0.037498	0.609086	-0.007086
5	0.609086	-0.007086	0.606724	0.001349
6	0.606724	0.001349	0.607174	-0.000257
7	0.607174	-0.000257	0.607088	0.000049

Hence, the root is 0.607088.

Question: Write a C program to implement Newton Raphson Method.

Theory

Newton Raphson method is an open method & starts with one initial guess for finding real root of non-linear equations.

In Newton Raphson method if x_0 is initial guess then next approximated root x_1 is obtained by following formula:

$$x_1 = x_0 - f(x_0)/g(x_0)$$

Algorithm

1. Start
2. Define function as f(x).
3. Define derivative of f(x) as g(x).
4. Input initial guess x_0, error(e) & max iteration(N).
5. Iteration Initialization of counter i=1.
6. If $g(x_0) = 0$ then print "Mathematical Error" & goto (12) step, otherwise goto step(7).
7. Calculate $x_1 = x_0 - f(x_0)/g(x_0)$
8. Increment iteration counter $i = i+1$
9. If $i = N$ then print "Not Convergent" & goto (12) otherwise goto (10).
10. If $|f(x_1)| > e$ then set $x_0 = x_1$ & goto (6) otherwise goto (11).
11. Print root as x_1.
12. Stop.

Example

Q) $3x - \cos x - 1$, Error = 0.00001

Here,

$f(x) = 3x - \cos x - 1$

$f'(x) = 3 + \sin x$

Let Initial guess $(x_0) = 2$

Iteration1

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 2 - \frac{f(2)}{f'(2)} = 2 - \frac{5.41614683 65}{3.9092974268} = 0.6145472$$

Iteration2

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 0.6145472 - \frac{f(0.614547)}{f'(0.614547)} = 0.614457 - \frac{0.026607}{3.576588} = 0.6071077$$

Iteration3

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 0.6071 - \frac{f(0.6071)}{f'(0.6071)} = 0.6071 - \frac{0.000023}{0.574049} = 0.60710$$

Here, Two coscutive roots are similar till the 4th position after decimal point so the required root is 0.60710.

Question: Write a program in c to implement Newton's divided difference formula.

Theory

Newton's Divided Difference formula was put forward to overcome few limitations of lagrane's formula. In lagrane's formula, if another interpolation value were to be inserted, then the interpolation coefficients were to be calculated again. This is not the case in divided difference.

If $(x_0, y_0), (x_1, y_1), (x_2, y_2)$ ..... be given points, then the first divided difference for the arguments x_0, x_1 is defined as: $[x_0, x_1] = (y_1 - y_0)/(x_1 - x_0)$ and, similarly, $[x_1, x_2] = (y_2 - y_1)/(x_2 - x_1)$ & after that $[x_2, x_3] = (y_3 - y_2)/(x_3 - x_2)$ & so on.

$$F(x) = f(x_0) + (x - x_0) F[x_0, x_1] + (x - x_0)(x - x_1) F(x_0, x_1, x_2] + \cdots$$
$$+ (x - x_0)(x - x_1) \cdots (x - x_{k-1}) F[x_0, x_1, \cdots, x_k],$$

The second divided difference is defined as: $[x_0, x_1, x_2] = ([x_1, x_2] - [x_0, x_1]) / (x_2 - x_0)$. This goes on similar fashion for the third, fourth .... & nth divided differences. Based on these formulas, two properties of this method/formula can be outlined as given below.

① The divided differences are symmetrical in their arguments i.e. independent of the order of the arguments.

② The nth divided differences of a polynomial of the nth degree are constant.

Example

Time(t)	Velocity(v)
0	0
10	227.04
15	362.78
20	517.35
22.5	602.97
30	901.67

The upward velocity of a rocket is given as a function of time in table above. Determine the value of the velocity at $t=16$ seconds with third order polynomial using Newton's divided difference Polynomial method.

Solution

For a third order polynomial, the velocity is given by

$$p(x) = a_0 + a_1(x-x_0) + a_2(x-x_0)(x-x_1) + a_3(x-x_0)(x-x_1)(x-x_2)$$

Since we want to find the velocity at $t=16$, & we are using a third order polynomial, we need to choose the four data points that are closest to $x=16$ that also bracket $t=16$ to evaluate it. The four data points are $x_0=10$, $x_1=15$, $x_2=20$ & $x_3=22.5$

Then,

$x_0=10$, $f(x_0)=227.04$

$x_1=15$, $f(x_1)=362.78$

$x_2=20$, $f(x_2)=517.35$

$x_3=22.5$, $f(x_3)=602.97$

Now, calculate values of divided differences as below.

$a_0 = F[x_0] = f(x_0) = 227.04$

$a_1 = F[x_1, x_0] = \dfrac{f(x_1) - f(x_0)}{x_1 - x_0} = \dfrac{362.78 - 227.04}{15 - 10} = 27.148$

$a_2 = F[x_2, x_1, x_0] = \dfrac{f[x_2, x_1] - f[x_1, x_0]}{x_2 - x_0}$

$a_3 = F[x_3, x_2, x_1, x_0] = \dfrac{F[x_3, x_2, x_1] - f(x_2, x_1, x_0)}{x_3 - x_0}$

We know,

$f[x_2, x_1] = \dfrac{f(x_2) - f(x_1)}{x_2 - x_1} = \dfrac{517.35 - 362.78}{20 - 15} = 30.914$

&

$f[x_1, x_0] = 27.148$

$\Rightarrow a_2 = \dfrac{F[x_2, x_1] - F[x_1, x_0]}{x_2 - x_0} = \dfrac{30.914 - 27.148}{20 - 10} = 0.37660$

Again,

$F(x_3, x_2, x_1) = \dfrac{F[x_3, x_2] - f[x_2, x_1]}{x_3 - x_1}$

We know that,

$f[x_3, x_2] = \dfrac{f(x_3) - f(x_2)}{x_3 - x_2} = \dfrac{602.97 - 517.35}{22.5 - 20} = 34.248$

$\Rightarrow f[x_3, x_2, x_1] = \dfrac{f[x_3, x_2] - f[x_2, x_1]}{x_3 - x_1} = \dfrac{34.248 - 30.914}{22.5 - 15}$

$= 0.44453$

Therefore,

$a_3 = \dfrac{f[x_3, x_2, x_1] - f(x_2, x_1, x_0]}{x_3 - x_0} = \dfrac{0.44453 - 0.37660}{22.5 - 10} = 5.4347 \times 10$

Question: Write a C program to implement False position Method.

Theory

False position method is one of bracketing and convergence guaranteed method for finding real root of non-linear equations. It starts with initial guesses say x_0 and x_1 such that x_0 & x_1 brackets the root i.e, $f(x_0) \cdot f(x_1) < 0$

If x_0 and x_1 are two initial guesses, then we compute new approximated root as:

$$x_2 = x_0 - ((x_0 - x_1) * f(x_0)) / (f(x_0) - f(x_1))$$

Here, we have different cases,

1. If $f(x_2) = 0$ then the root is x_2.
2. If $f(x_0) \cdot f(x_2) < 0$ then root lies between x_0 & x_2.
3. If $f(x_0) \cdot f(x_2) > 0$ then root lie between x_1 & x_2.

And then process is repeated until we find root within desired accuracy.

Algorithm

1. Start
2. Define function $f(x)$.
3. Choose initial guesses x_0 & x_1 such that $f(x_0) \cdot f(x_1) < 0$.
4. Choose pre-specified tolerable error e.
5. Calculate new approximated root as:

$$x_2 = x_0 - ((x_0 - x_1) * f(x_0)) / (f(x_0) - f(x_1))$$

6. Calculate $f(x_0) \cdot f(x_2)$
   - a. If $f(x_0) \cdot f(x_2) < 0$ then $x_0 = x_0$ and $x_1 = x_2$
   - b. If $f(x_0) \cdot f(x_2) > 0$ then $x_0 = x_2$ and $x_1 = x_1$
   - c. If $f(x_0) \cdot f(x_2) = 0$ then goto (8).
7. If $|f(x_2)| > e$ then goto (5) otherwise goto (8).
8. Display x_2 as root.
9. Stop

Example: $x^3 - 2x - 5$, Tolerable error$(\varepsilon) = 0.00001$

I	x_0	$f(x_0)$	x_1	$f(x_1)$	x_2	$f(x_2)$
1	2	-1	3	16	2.058825	-0.390799
2	2.058825	-0.390799	3	16	2.081264	-0.147203
3	2.081264	-0.147203	3	16	2.089639	-0.054677
4	2.089639	-0.054677	3	16	2.092740	-0.020203
5	2.092740	-0.020203	3	16	2.093884	-0.007450
6	2.093884	-0.007450	3	16	2.094306	-0.002745
7	2.094306	-0.002745	3	16	2.094461	-0.001010
8	2.094461	-0.001010	3	16	2.094518	-0.000372
9	2.094518	-0.000372	3	16	2.094539	-0.000137
10	2.094537	-0.000137	3	16	2.094547	-0.000050
11	2.094547	-0.000050	3	16	2.094550	-0.000018
12	2.094550	-0.000018	3	16	2.094551	-0.000007