Bisection Method

Objective: To find the roots of a polynomial equation

Overall Process: It separates the interval and subdivides the interval in which the root of the equation lies.

Algorithm:

Step 1: Find two points, `a` and `b`, where `a` is smaller than `b`, and the product of `f(a) f(b)` is negative.

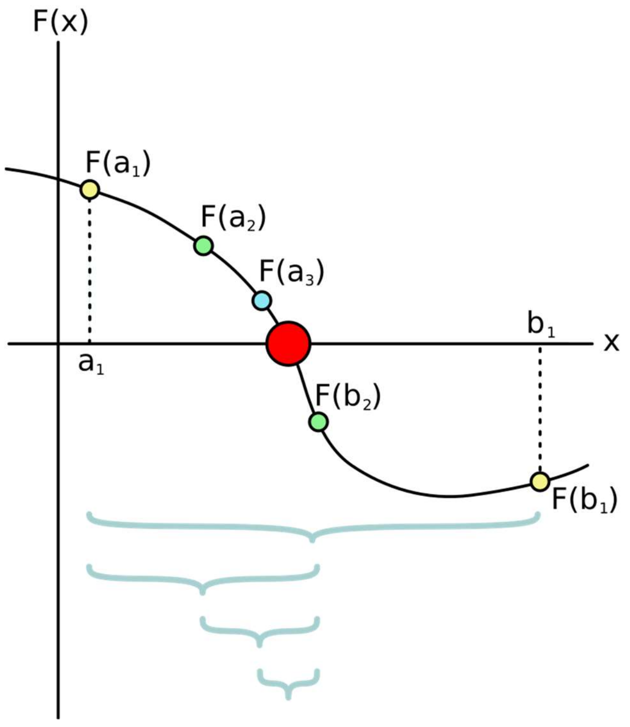
Step 2: Calculate the midpoint, `c`, between `a` and `b`.

Step 3: If `f(c)` equals 00, then `c` is the root of the function. If not, proceed to the next step.

Step 4: Divide the interval:

- If the product of `f(c) f(a)` is negative, assign `c` to `b`.
- Else assign `c` to `a`.

Step 5: Repeat the above three steps until `f(c)` equals 00.




```
#include<stdio.h>
#define EPSILON_ACCURACY 0.01
double function(double x)
      {return x*x*x - x*x + 2;}
void bisection(double a, double b)
{
      /* Are the initial assumptions correct? */
      if (function(a) * function(b) >= 0)
      {     printf("Your initial assumptions are not correct\n");
            return;
      }
      double c;
      while ((b-a) >= EPSILON_ACCURACY)
      {
            c = (a+b)/2;     /*Compute the middle point*/
            if (function(c) == 0.0)break; /*Have we found the root?*/
            /* Continue process*/
            else if (function(c)*function(a) < 0) b = c;
            else a = c;
      }
      printf("The value of root is : %f",c);
}

int main()
{     double a =-200, b = 300;
      bisection(a, b);
      return 0;
}
```

False Position or Regula Falsi Method

Objective: To find the roots of a non-linear equation

Overall Process: It is a trial and error method of using "false" values of variable and then altering the false value according to the result.

Algorithm:

Step 1: Find two points, `a` and `b`, where `a` is smaller than `b`, and the product of `f(a)f(b)` is negative.

Step 2: Calculate `c = a - f(a) x (b-a) / [f(b) - f(a)]`

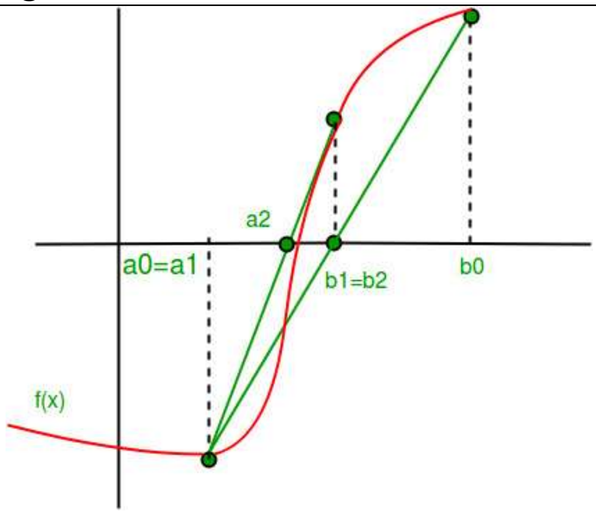
Step 3: If `f(c) == 0`, then `c` is the root of the solution. STOP.

Step 4: if `f(c) != 0`

If value `f(a)*f(c) < 0` then `c = a;`

Else If `f(b)*f(c) < 0` then `c = b.`

Step 5: Repeat the steps




```
#include<stdio.h>
#define ITERATION 100000
double function(double x)
      {return x*x*x - x*x + 2; }

void falsePosition(double a, double b)
{
      int i;
      /* Initial assumptions correct? */
      if (function(a) * function(b) >= 0)
      {     printf("Your initial assumptions are not correct\n");
            return;
      }
      double c;
      for(i=1; i<=ITERATION; ++i)
      {
            /*Compute the middle point*/
c = (a*function(b) - b*function(a))/ (function(b) - function(a));
            if (function(c) == 0.0)break; /*Have we found the root?*/
            /* if not found the root, continue the process*/
            else if (function(c)*function(a) < 0)b = c;
            else a = c;
      }
      printf("The value of root is : %f",c);
}
int main()
{
      double a =-200, b = 300;
      falsePosition(a, b);
      return 0;
}
```

Secant Method

Objective: To find the roots of a non-linear equation.

Overall Process: Here, we don't need to check f(a)f(b)<0 again and again. Neighbourhoods roots are approximated by secant line or chord to the function f(x)

Algorithm:

Step 1: Find two points, `a` and `b`, such that f(a)<0 and f(b)>0.

Step 2: Calculate `c = a - f(a) x (b-a) / [f(b) - f(a)]`

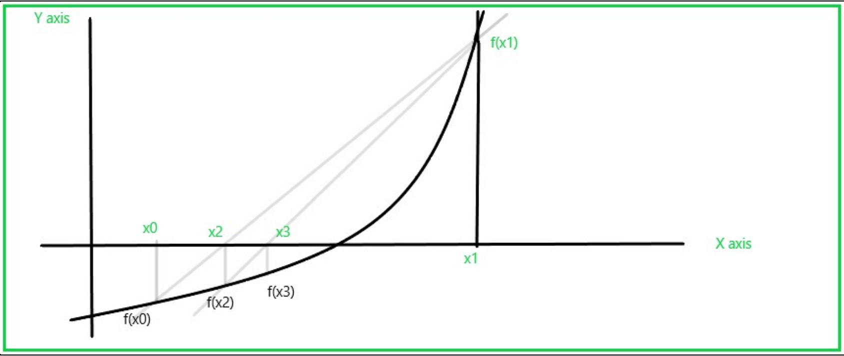
Step 3: If `f(c) == 0`, then `c` is the root of the solution. STOP.

Step 4: if `f(c) != 0`

If value `f(a)*f(c) < 0` then `c = a;`

Else If `f(b)*f(c) < 0` then `c = b.`

Step 5: Repeat the steps

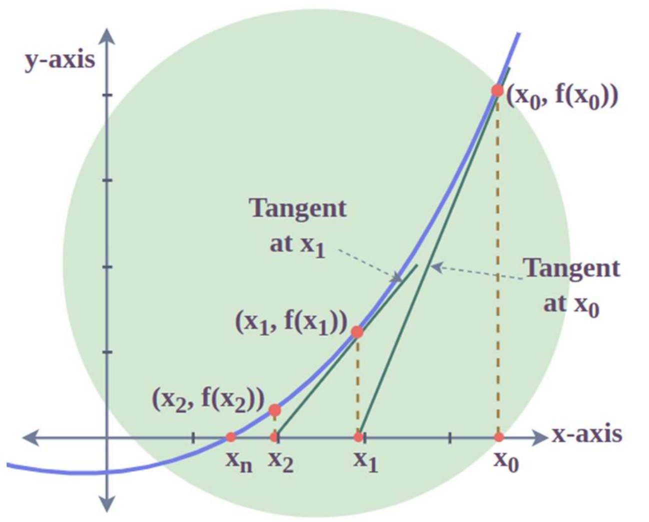



```
#include<stdio.h>
#define ACCURACY 0.001
double function(double x)
      {return x*x*x - x*x + 2; }
void secant(double a, double b)
{
      double c1, c2, isRoot;
      /* Are the initial assumptions correct? */
      if (function(a) * function(b) >= 0)
      {     printf("Your initial assumptions are not correct\n");
            return;
      }
      else
      {     do
            {     c1 = (a * function(b) - b * function(a)) / (function(b) - function(a));
                  isRoot = function(a) * function(c1);
                  a = b;
                  b = c1;
                  if(isRoot==0.00)break;
                  c2 = (a * function(b) - b * function(a)) / (function(b) - function(a));
            }while(abs(c2-c1)>=ACCURACY);
      }
      printf("The value of root is : %f",c1);
}
int main()
{     double a =-200, b = 300;
      secant(a, b);
      return 0;
}
```

Newton Raphson Method

Objective: To find the roots of a non-linear equation.
Overall Process: It is an iterative numerical method used to find the roots of a real-valued function
Algorithm:
`b = a – f(a)/f'(a)`
`a` is the initial value,
`f(a)` is the value of the equation at initial value, and
`f'(a)` is the value of the first order derivative of the equation.

Step 1: Draw a graph of `f(x)` for different values of x.
Step 2: A tangent is drawn to `f(x)` at `a`. This is the initial value.
Step 3: This tangent will intersect the X- axis at some fixed point `(a,0)` if the first derivative of `f(a)` is not zero i.e. `f'(a) ≠ 0`.
Step 4: As this method assumes iteration of roots, this `b` is considered to be the next approximation of the root.
Step 5: Now steps 2 to 4 are repeated until we reach the actual root.




```
#include<stdio.h>
#define ACCURACY 0.001

double function(double x)
{
      return x*x*x - x*x + 2;
}
double derivativeFunction(double x)
{
      return 3*x*x - 2*x;
}

void NewtonRaphson(double a)
{
      double b;
      /* Are the initial assumptions correct? */
      b = function(a) / derivativeFunction(a);
      while(fabs(b)>=ACCURACY)
      {
            b = function(a) / derivativeFunction(a);
            a = a - b;
            printf("a = %f b = %f\n",a,b);
      }
      printf("The value of root is : %f",a);
}

int main()
{
      double a =-20;
      NewtonRaphson(a);
      return 0;
}
```

Trapezoidal Method

Objective: To find the approximation of a definite integral.

Overall Process: The basic idea is to assume the region under the graph of the given function to be a trapezoid and calculate its area.

Algorithm:

`b = a − f(a)/f'(a)`

`a` is the initial value,

`f(a)` is the value of the equation at initial value, and

`f'(a)` is the value of the first order derivative of the equation.

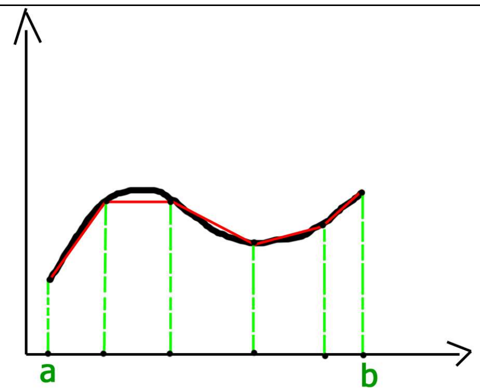
Step 1: Draw a graph of `f(x)` for different values of x.

Step 2: A tangent is drawn to `f(x)` at `a`. This is the initial value.

Step 3: This tangent will intersect the X- axis at some fixed point `(a,0)` if the first derivative of `f(a)` is not zero i.e. `f'(a) ≠ 0`.

Step 4: As this method assumes iteration of roots, this `b` is considered to be the next approximation of the root.

Step 5: Now steps 2 to 4 are repeated until we reach the actual root.




```
#include<stdio.h>

float function(float x)
      {return 1/(1+x*x);}

float trapezoidal(float a, float b, float n)
{
      int i;
     float h = (b-a)/n;   /* Grid spacing */

     float s = function(a)+function(b);

     /* Adding middle strips */
     for (i = 1; i < n; i++)s += 2*function(a+i*h);

     return (h/2)*s;
}

int main()
{
     float a = 0; /* Initial Value */
     float b = 1; /* Final Value */
      int n = 6;   /* Number of grid */

     printf("Value of integral is %f\n", trapezoidal(a, b, n));
     return 0;
}
```