Department of Physics
University of Dhaka
Computational Physics

If the function is polynomial then one can find the root of the function analytically or even numerically. But if the function is transcendental, the analytical solution is not possible one must need numerical solution.
Transcendental Function: A transcendental function is an analytic function that does not satisfy a polynomial equation, in contrast to an algebraic function. In other words, a transcendental function "transcends" algebra in that it cannot be expressed in terms of a finite sequence of the algebraic operations of addition, multiplication, and root extraction.
Examples of transcendental functions include the exponential function, the logarithm, and the trigonometric functions.
Example: $f(x) = x^{\pi}$, $f(x) = e^x$, $f(x) = x^x$, $f(x) = sin(x)$, $f(x) = log_e(x)$ etc.
We will discuss about the **Bisection method, Newton Raphson method and the Secant method.**

Bisection Method

- Given a function f(x) on floating number x and two numbers a and b such that $f(a)*f(b) < 0$ and f(x) is continuous in [a, b]. Here f(x) represents algebraic or transcendental equation. Find root of function in interval [a, b] (Or find a value of x such that f(x) is 0).
- Algebraic function are the one which can be represented in the form of polynomials like $f(x) = a_1x^3 + a_2x^2 + ... + e$ where $a_0, a_1, a_2, ...$ are constants and x is a variable. Transcendental function are non algebraic functions, for example $f(x) = sin(x) * x - 3$ or $f(x) = e^x + x^2$ or $f(x) = ln(x) + x....$
- The method is also called the interval halving method, the binary search method or the dichotomy method. This method is used to find root of an equation in a given interval that is value of 'x' for which $f(x) = 0$.

  The method is based on The Intermediate Value Theorem which states that if f(x) is a continuous function and there are two real numbers a and b such that $f(a) * f(b) < 0$. Then f(x) has at least one zero between a and b. If for a function $f(a) < 0$ and $f(b) > 0$) or $f(a) > 0$ and $f(b) < 0$, then it is guaranteed that it has at least one root between them.
  Assumption:

  1. f(x) is a continuous function in interval [a, b].
  2. $f(a) * f(b) < 0$

  Steps:

  1. Find middle point $c = (a + b)/2$.
  2. If $f(c) == 0$, then c is the root of the solution.
  3. Else $f(c)! = 0$

- ⋆ **If** value $f(a) * f(c) < 0$ then root lies between a and c. So we recur for a and c.
- ⋆ **Else If** $f(b) * f(c) < 0$ then root lies between b and c. So we recur b and c.
- ⋆ **Else** given function doesn't follow one of assumptions.

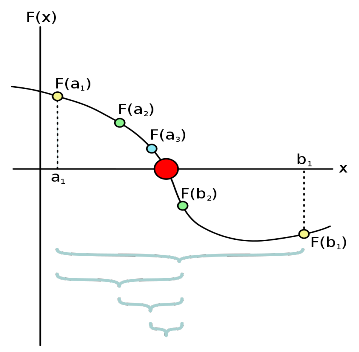

Figure 1: The bisection method.

- **Advantage of the Bisection method:**
  - ⋆ The bisection method is always convergent. Since the method brackets the root, the method is guaranteed to converge.
  - ⋆ As iterations are conducted, the interval gets halved. So one can guarantee the decrease in the error in the solution of the equation.
- **Drawbacks:**
  - ⋆ The convergence of bisection method is slow as it is simply based on halving the interval..
  - ⋆ If one of the initial guesses is closer to the root, it will take larger number of iterations to reach the root.
  - ⋆ If a function $f(x)$ is such that it just touches the x-axis such as $f(x) = x^2$ it will be unable to find the lower guess, x_l and upper guess, x_u , such that $f(x_l) * f(x_u) < 0$.
  - ⋆ d) For functions $f(x)$ where there is a singularity and it reverses sign at the singularity, bisection method may converge on the singularity.An example include $f(x) = 1/x$

```
// C++ program for implementation of Bisection Method for // solving equations
#include< iostream >
using namespace std;
#define EPSILON 0.01

    // An example function whose solution is determined using
// Bisection Method. The function is x^3 − x^2 + 2
double func(double x)
{ return x*x*x - x*x + 2; } // Prints root of func(x) with error of EPSILON
void bisection(double a, double b)
{ if (func(a) * func(b) >= 0)
{ cout << ”You have not assumed right a and b”;
return;
}
```

```
    double c = a;
while ((b − a) >= EPSILON)
{ // Find middle point
c = (a+b)/2;

    // Check if middle point is root
if (func(c) == 0.0)
break;

    // Decide the side to repeat the steps
else if (func(c) ∗ func(a) < 0)
b = c;
else
a = c;
}
cout << "The value of root is : " << c; }
// Driver program to test above function
int main()
{
// Initial values assumed
double a =-200, b = 300;
bisection(a, b);
return 0;
}
```

Newton-Raphson Method

This is probably the most well known (also known as Newtons) method for finding function roots. The method can produce faster convergence by cleverly implementing some information about the function f. Unlike the bisection method, Newtons method requires only one starting value and does not need to satisfy any other serious conditions (except maybe one!). This is a one-point method. Newton-Raphson method is based on the principle that if the initial guess of the root of f(x)=0 is at x_i, then if one draws the tangent to the curve at $f(x_i)$, the point x_{i+1} where the tangent crosses the x-axis is an improved estimate of the root (Figure 2).

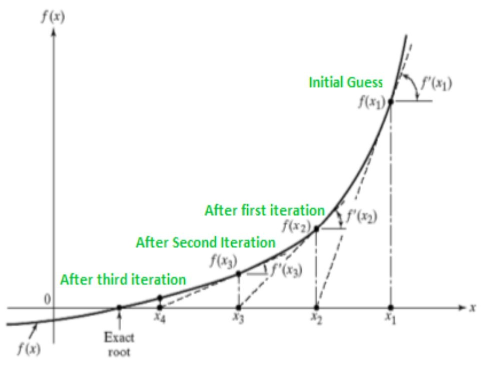

Figure 2: Geometrical illustration of Newton-Raphson method.

Let us start by presenting one of the ways that we can actually obtain such a numerical root finding scheme. Let a be an approximate root of $f(x) = 0$ and let $b = a + h$ be the corrected root so that $f(b) = 0$ By Taylor series exapnasion we have,
$f(b) = 0$
$f(a + h) = 0$
$f(a) + hf'(a) + \frac{h^2}{2!}f''(a) + ... = 0$
Neglecting the second and higher order derivative, we have

$$f(a) + hf'(a) = 0$$

whice give

$$h = -\frac{f(a)}{f'(a)}$$

Therefore,

$$b = a - \frac{f(a)}{f'(a)}$$

The next point,

$$c = b - \frac{f(b)}{f'(b)}$$

$$d = c - \frac{f(c)}{f'(c)}$$

In general

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

That is if we choose the initial guess x_0 close enough to the root of the function f(x) we are essentially guaranteed that we will find the true root!
In short the method is implemented as follows:

- Obtain a starting guess x_0 and find a formula for the derivative f' of f.
- Produce the next iterate x_n through the following equation

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

- Stop if any of the following occurs: required tolerance is reached, maximum iterates exceeded, $f'(x_{n-1}) = 0$.

Advantage: Fast converging.
Disadvantages:

- Calculating the required derivative itself $f'(x_n)$ for every iteration may be a costly task for some functions f.
- May not produce a root unless the starting value x_0 is close to the actual root of the function.
- May not produce a root if for instance the iterations get to a point x_{n-1} such that $f'(x_{n-1}) = 0$. Then the method fails!

Find the root of the equation $f(x) = x^3 - x^2 + 2$ by using Newton-Rapson Method.

```
#include< iostream >
#define EPSILON 0.001
using namespace std;

    double func(double x)
{
return x*x*x - x*x + 2;
}

    double derivFunc(double x)
{
return 3*x*x - 2*x;
}

    void newtonRaphson(double x)
{
double h = func(x) / derivFunc(x);
while (abs(h) >= EPSILON)
{
h = func(x)/derivFunc(x);
x = x - h;
}
cout << ”The value of the root is : ” << x;
}
int main()
{
double x0 = -20; // Initial values assumed
newtonRaphson(x0);
return 0;
}
```

The Secant Method

One of the main drawbacks of the Newton-Raphson method is that you have to evaluate the derivative of the function. With availability of symbolic manipulators such as Maple, Mathcad, Mathematica and Matlab, this process has become more convenient. However, it is still can be a laborious process. To overcome this drawback, the derivative, $f'(x)$ of the function, f(x) is approximated as

$$f'(x_n) = \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

Substituting $f'(x_n)$ in the equation of the N-R namely in

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

We have,

$$x_{n+1} = x_n - \frac{f(x_n) * (x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}$$

The above equation is called the Secant method. This method now requires two initial guesses, but unlike the bisection method, the two initial guesses do not need to bracket the root of the equation. The Secant method may or may not converge, but when it converges, it converges faster than the bisection method. However, since the derivative is approximated, it converges slower then Newton-Raphson method. Let us examine one example, if we let the first guess $x_{n-1} = x1$, and $x_n = x2$ and $x_{n+1} = x0$ then the above formula can be written as:

$$x0 = \frac{f(x2) * (x1) - x2 * f(x1)}{f(x2) - f(x1)}$$

Now, Let us solve the equation $f(x) = x^3 + x - 1$ by using secant method.

```
#include < iostream >
#include < cmath >
using namespace std;
double f(double x)
{
double f = pow(x, 3) + x - 1;
return f;
}
void secant(double x1, double x2, double E)
{
double n = 0, xm, x0, c;
if (f(x1) * f(x2) ¡ 0) {
do {
x0 = (x1 * f(x2) - x2 * f(x1)) / (f(x2) - f(x1));
c = f(x1) * f(x0);
x1 = x2;
x2 = x0;
n++;
if (c == 0)
break;
xm = (x1 * f(x2) - x2 * f(x1)) / (f(x2) - f(x1));
} while (fabs(xm - x0) >= E);
cout << ”Root of the given equation=” << x0 << endl;
cout << ”No. of iterations = ” << n << endl; } else
cout << ”Can not find a root in the given inteval”;
}
int main()
{

    double x1 = 0.0, x2 = 1.0, E = 0.0001;
secant(x1, x2, E);
return 0;
}
```

Exercise:

1. Find the root of the $f(x) = xe^x - 1 = 0$ using all the three methods, corrected up to three decimal places.
2. Find the root of the $cos(x) - xe^x = 0$ using all the three methods, corrected up to three decimal places.

3. Determine the smallest positive root of $x - e^{-x}$ up to three significant figure.
4.Find a root of $xsinx + cosx = 0$
5. Find the square root of 18 using Newton-Raphson method (Hints: you need to solve $x^2 - n = 0$ where n is the number and x is its square root.)