EXAMPLES OF NUMERICAL ANALYSIS

1. Taylor Series
2. Square Root
3. Integration
4. Root Finding
5. Line Fitting

Example 1:
Taylor Series

Example 1: Taylor Series

- Consider the Taylor series for computing sin(x)

$$sin(x) = \frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k+1}}{(2k+1)!}$$

- For a small x value, only a few terms are needed to get a good approximation result of sin(x).

- The rest of the terms (...) are truncated.

Truncation error = $f_{actual} - f_{sum}$

- The size of the truncation error depends on x and the number of terms included in f_{sum}.

- Write a C program to read X and N from user, then calculate the sin(X).

- Test your program for X=150 degree and following N values.
  - First run: N = 3
  - Second run: N = 7
  - Third run: N = 50

- Also using the built-in sin(x) function, calculate the f_{actual}, then compare with your results on right side.

- Which N value gives the most correct result?

Program

```c
#include <stdio.h>
#include <math.h>
#define PI 3.14

float factorial(int M)
{
  float result=1;
  int i;

  for (i=1; i<=M; i++)
      result *= i;
  return result;
}
```

```c
int main()
{
  int N;              // Number of terms
  int i;              // Loop counter
  int x = 150;        // Angle in degrees
  float toplam = 0;   // Sum of Taylor series
  float actual;       // Actual sinus
  int t;              // Term

  printf("Terim sayisini (N) veriniz :");
  scanf("%d", &N);

  for (i=0; i<= N-1; i++) {
      t = 2*i+1;
      toplam = toplam +
               ( pow(-1, i) * pow(x*PI/180 ,t) )
               / factorial(t);
  }

  printf("Calculated sum = %f\n", toplam);
  actual = sin(x*PI/180);
  // Angle is converted to radian

  printf("Actual sinus = %f\n", actual);
  printf("Truncation error = %f\n", toplam- actual);

} // end main
```

Screen outputs of test cases

- The result will be more accurate for bigger N values.

Program Output 1

```
Terim sayisini (N) veriniz :3
Calculated sum = 0.652897
Actual sinus = 0.501149
Truncation error = 0.151748
```

Program Output 2

```
Terim sayisini (N) veriniz :7
Calculated sum = 0.501150
Actual sinus = 0.501149
Truncation error = 0.000001
```

Program Output 3

```
Terim sayisini (N) veriniz :20
Calculated sum = 0.501149
Actual sinus = 0.501149
Truncation error = -0.000000
```

Example 2:
Square Root

Example 2: Square Root Computing with Newton Method

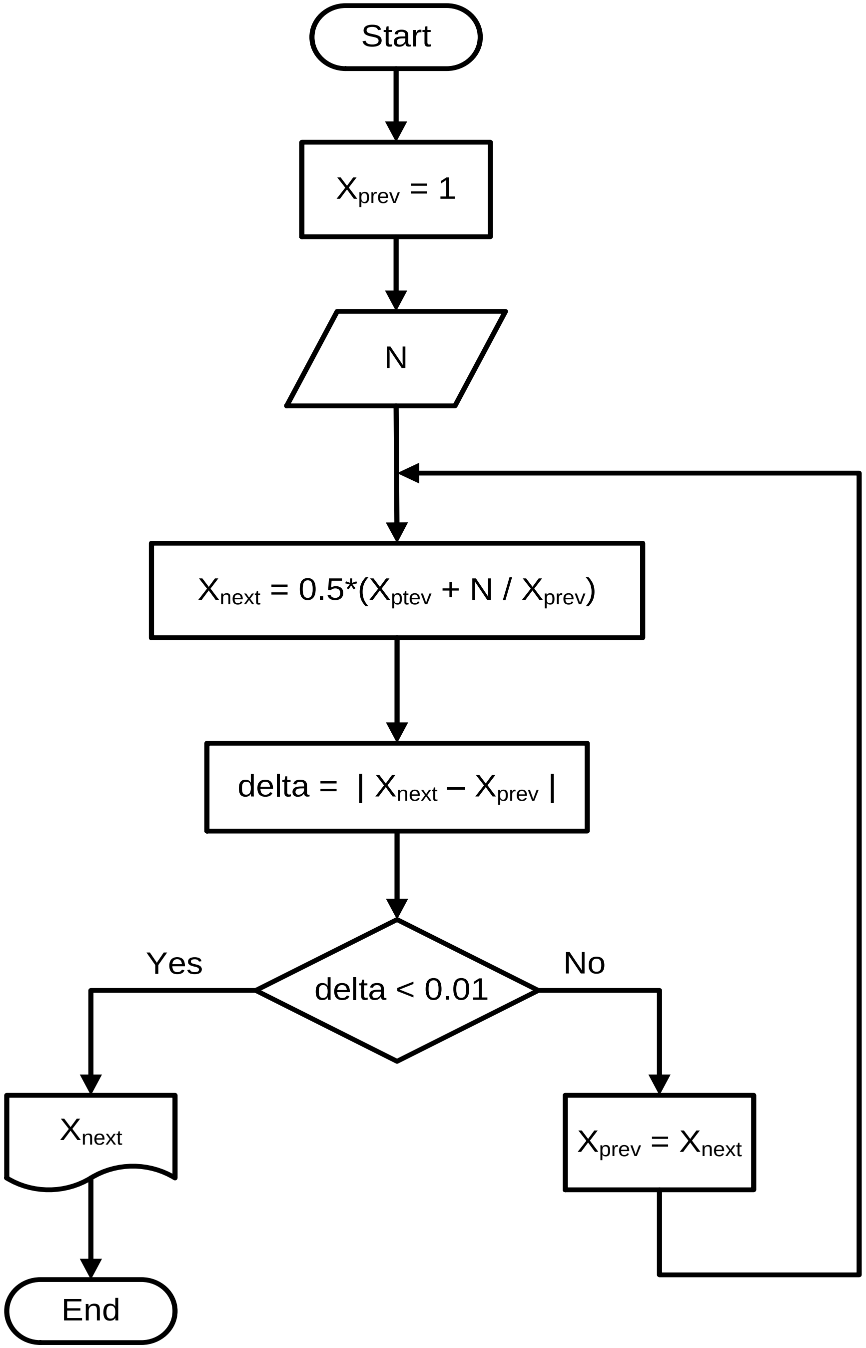
- Write a C program to compute the square root of N entered by user.

- The square root $\sqrt{N}$ of a positive integer number N can be calculated by the following Newton iterative equation, where $X_0 = 1$.

$$X_{k+1} = \frac{1}{2}\left(X_k + \frac{N}{X_k}\right) \qquad \Delta = \left| X_{k+1} - X_k \right|$$

- When delta (i.e. tolerance) < 0.01 , then the iterations (i.e. repetitions) must stop.
- The final value of X_{k+1} will be the answer.

- You should not use the built-in **sqrt()** function, but you can use the **fabs()** function.



Program

```
#include <stdio.h>
#include <math.h>

int main()
{
  int N;
  float XPrev, XNext, delta;
  printf("Enter N :");
  scanf("%d", &N);

  XPrev = 1;
  while (1)
  {
    XNext = 0.5*(XPrev + N / XPrev);
    delta = fabs(XNext - XPrev);
    if (delta < 0.01) break;
    XPrev = XNext;
  } // end while

  printf("Square Root = %f \n", XNext);

} // end main
```

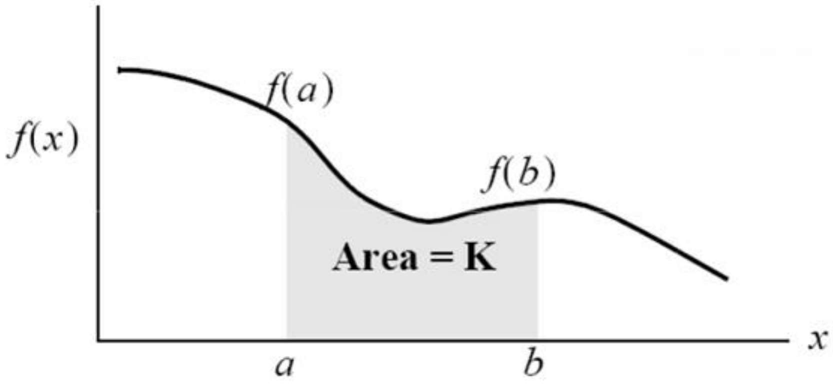
```
Enter N :2
Square Root = 1.414216
```

Example 3: Integration

Example 3: Integration Computing

- **<u>Integration :</u>** Common mathematical operation in science and engineering.
- Calculating area, volume, velocity from acceleration, work from force and displacement are just few examples where integration is used.
- Integration of simple functions can be done analytically.

- Consider an arbitrary mathematical function $f(x)$ in the interval $a \leq x \leq b$.
- The definite integral of this function is equal to the area under the curve.
- For a simple function, we evaluate the integral in closed form.
- If the integral exists in closed form, the solution will be of the form
  K = F(b) - F(a) where F'(x) = f(x)



$$K = \int_a^b f(x)dx$$

$$K = F(x)\Big|_a^b = F(b) - F(a)$$

Analytical Integration Example

- Let's compute the following definite integral.

$$K = \int_0^2 x^2 dx = ?$$

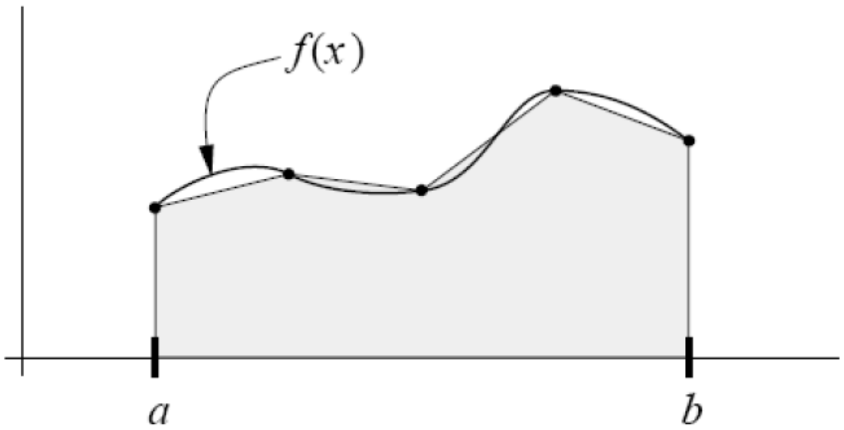
- Analytical Solution:

$$f(x) = x^2 \quad \Rightarrow \quad F(x) = \frac{1}{3}x^3$$

$$K = \int_0^2 x^2 dx \quad = \quad \frac{1}{3}x^3 \Big|_0^2 \quad = F(2) - F(0) = \frac{8}{3} - \frac{0}{3} = \quad 2.667$$

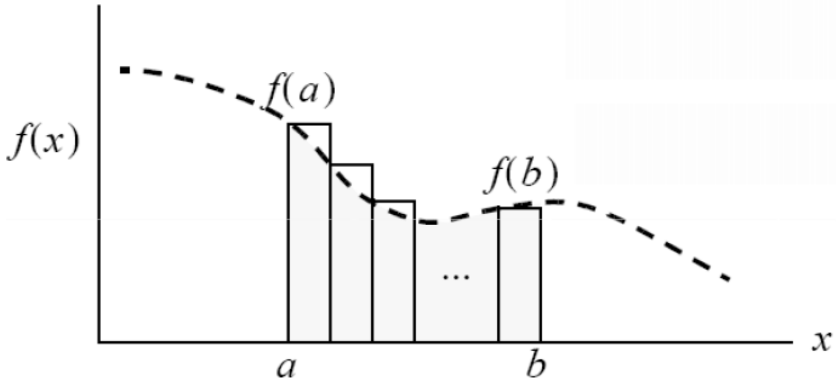
Approximation of Numerical Integration

- Numerical solutions resort to finding the area under the f(x) curve through some approximation technique.
- The value of $\int_a^b f(x)dx$ is approximated by the shaded area under the piecewise-linear interpolation of f(x).



Rectangular Rule

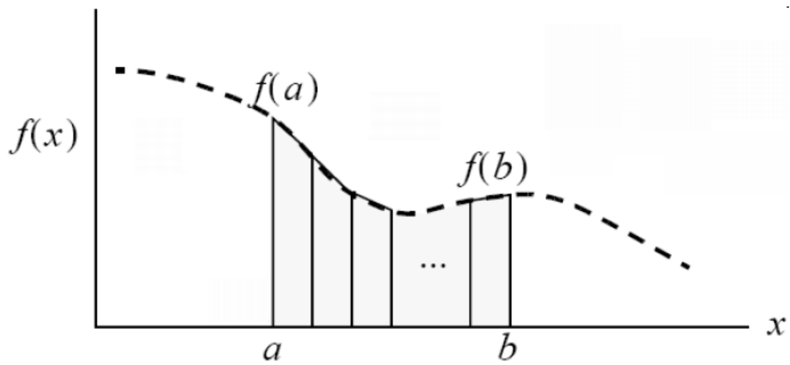
- Area under curve is approximated by sum of the areas of small rectangles.



$$K \approx \sum Rectangular\ Areas$$

Trapezoidal Rule

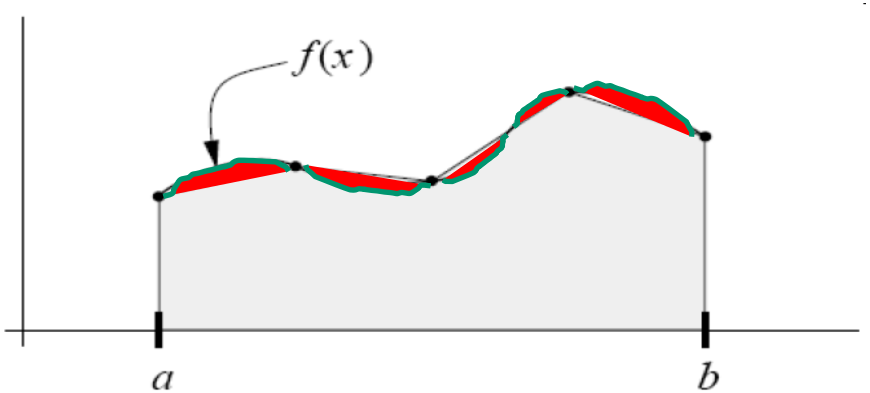
- Area under curve is approximated by sum of the areas of small trapezoidals.



$$K \approx \sum Trapezoidal\ Areas$$

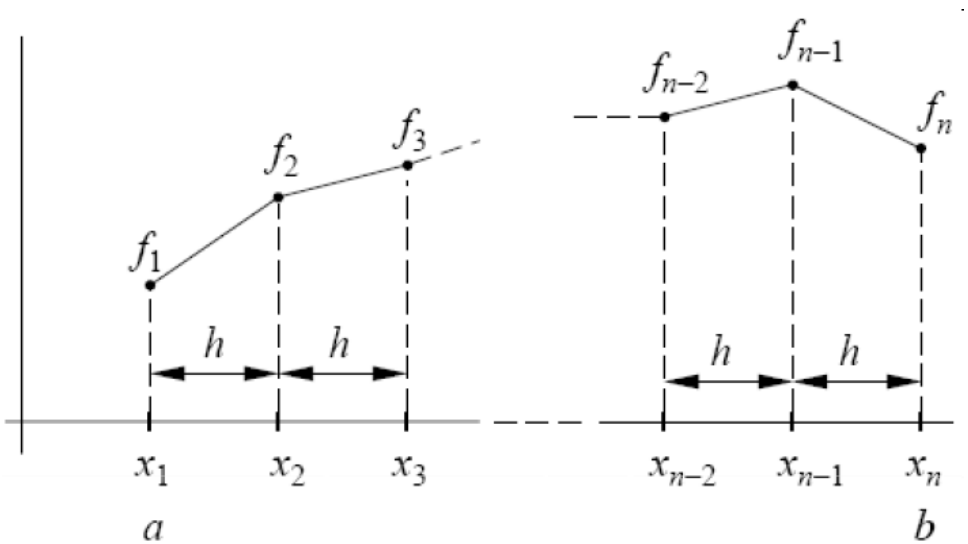
Truncation Errors

- All numerical integral approximation methods may contain some small truncation errors, due to the uncalculated tiny pieces of areas in function curve boundaries.



Numerical Integration with Composite Trapezoidal Rule

- The trapezoidal method divides the region under a curve into a set of panels. It then adds up the areas of the individual panels (trapezoids) to get the integral.
- The composite trapezoid rule is obtained summing the areas of all panels.
- The area under f (x) is divided into N vertical panels each of width h, called the step-length.
- K is the estimate approximation to the integral, where $x_i = a + ih$



$$K = \frac{h}{2}\left[f(a) + f(b) + 2\sum_{i=2}^{n-1} f(x_i)\right]$$

$$h = \frac{b-a}{n}$$

Program

```c
#include <stdio.h>

float func(float x)
{  // Curve function f(x)
     return x*x ;
}

int main() {
  float a, b;  // Lower and upper limits of integral
  int N;       // Number of panels
  float h;     // Step size
  float x;     // Loop values
  float sum=0;
  float K;     // Resulting integral
  printf("Enter N :"); scanf("%d", &N);
  printf("Enter a and b :"); scanf("%f%f", &a, &b);

  h = (b-a)/N;
  for (x=a+h; x<=b-h; x+=h)
      sum += func(x);

  K = 0.5*h * (func(a) + 2*sum + func(b) );
  printf("Integration = %f \n", K);
} // end main
```

Screen output

- Expected result $$\int_0^2 x^2 dx = 2.667$$

- The result will be more accurate for bigger N values.

Program Output 1

```
Enter N :10
Enter a and b : 0  2
Integration = 2.032000
```

Program Output 2

```
Enter N :30
Enter a and b : 0  2
Integration = 2.418964
```

Program Output 3

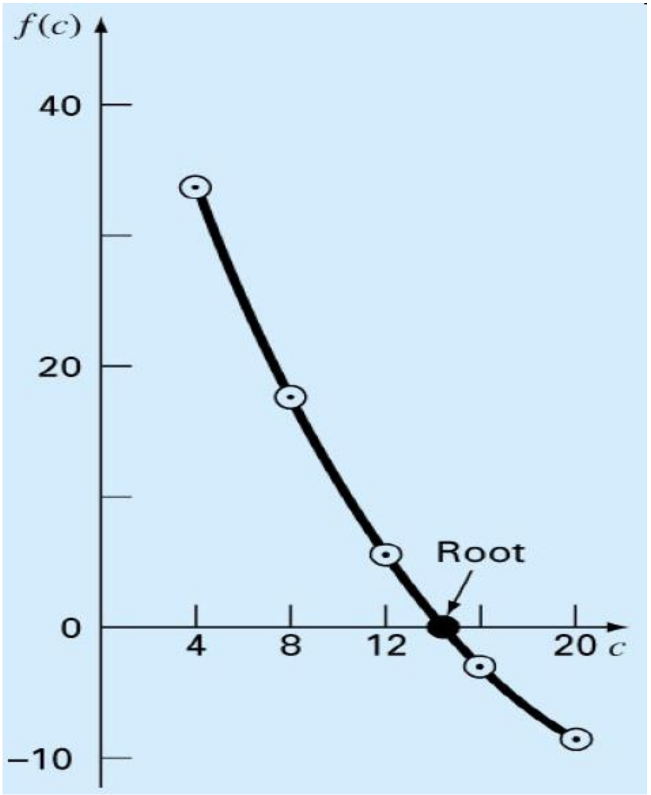
```
Enter N :50
Enter a and b : 0  2
Integration = 2.667199
```

Example 4:
Root Finding

Example 4: Root Finding for Nonlinear equations

- Nonlinear equations can be written as f(x) = 0

- Finding the roots of a nonlinear equation is equivalent to finding the values of x for which f(x) is zero.



Successive Substitution (Iteration)

- A fundamental principle in computer science is iteration.
- As the name suggests, a process is repeated until an answer is achieved.
- Iterative techniques are used to find roots of equations, solutions of linear and nonlinear systems of equations, and solutions of differential equations.
- A rule or function for computing successive terms is needed, together with a starting value .
- Then a sequence of values is obtained using the iterative rule $V_{k+1}=g(V_k)$

Roots of $f(x) = 0$

- Any function of one variable can be put in the form $f(x) = 0$.

- Example: To find the x that satisfies

  $\cos(x) = x$

- Find the zero crossing of

  $f(x) = \cos(x) - x = 0$

The basic strategy for root-finding procedure

1. First, plot the function to see a rough outline.

   The plot provides an initial guess, and an indication of potential problems.

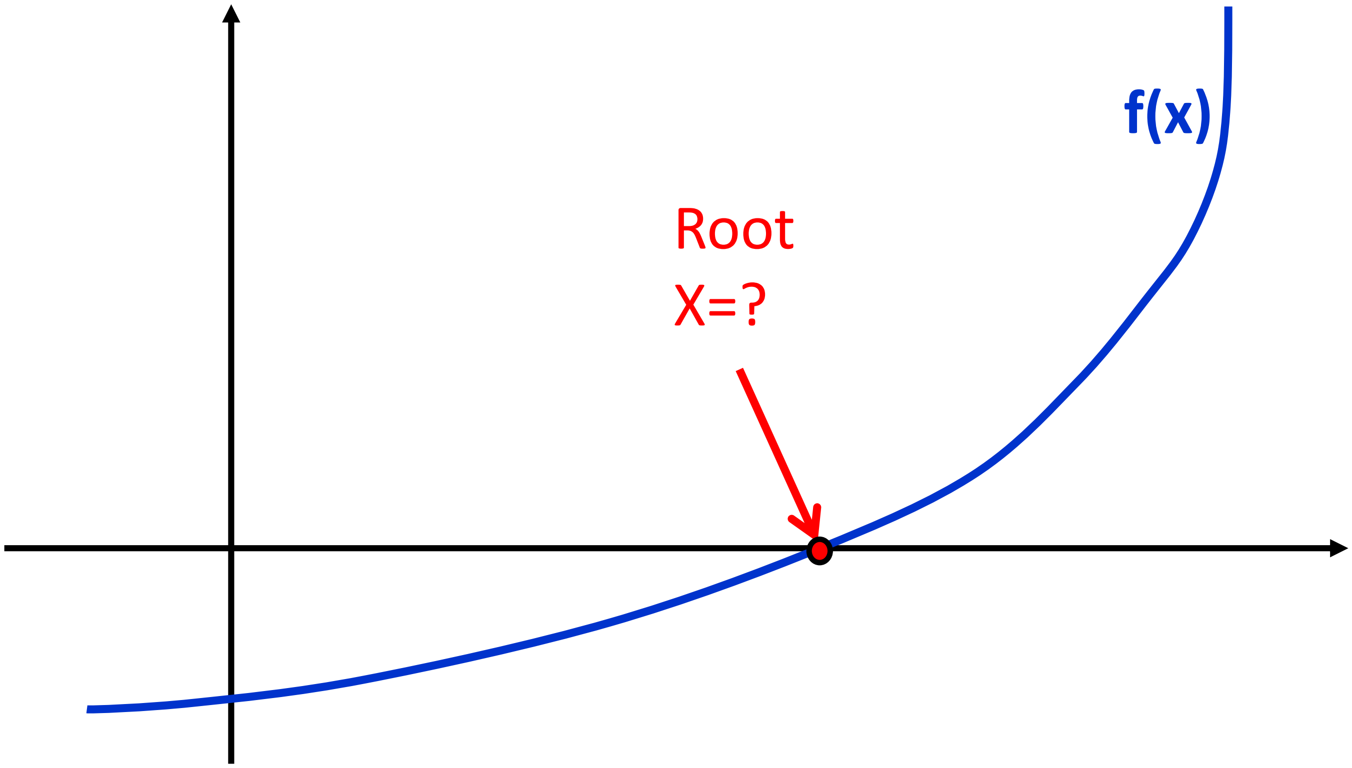
2. Then, select an **initial guess**.

3. Iteratively refine the initial guess with a root finding algorithm.

   If x_k is the estimate to the root on the k^{th} iteration, then the iterations **converge**

Example Function

- Assume the following is the grahics of f(x) function.
- We want to find the exact root where f(x) = 0



Newton-Raphson Method Algorithm

Step 1: Initially, user gives an X_1 value as an estimated root.
Plug X1 in the function equation and calculate $f(X_1)$.
Start at the point $(X_1, f(X_1))$

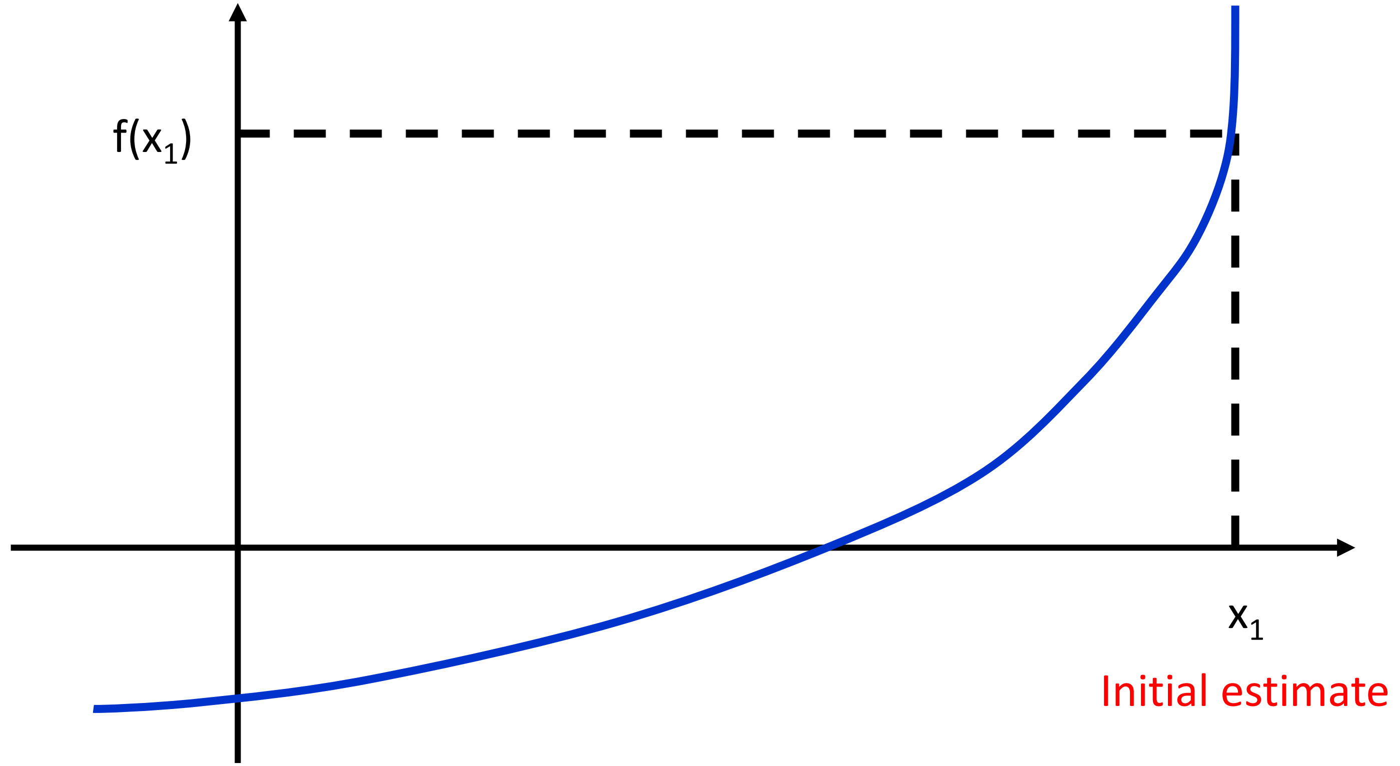
Step 2: The intersection of the tangent of f(X) function at this point and the X-axis can be calculated by following formula:
$X_2 = X_1 – f(X_1) / f'(X_1)$

Step 3: Examine if $f(X_2) = 0$
or $abs(X_2 – X_1) < Tolerance$

Step 4: If yes, solution $X_{root} = X_2$
If not, assign X_2 to X_1 ($X_1 \leftarrow X_2$), then repeat the iteration above.

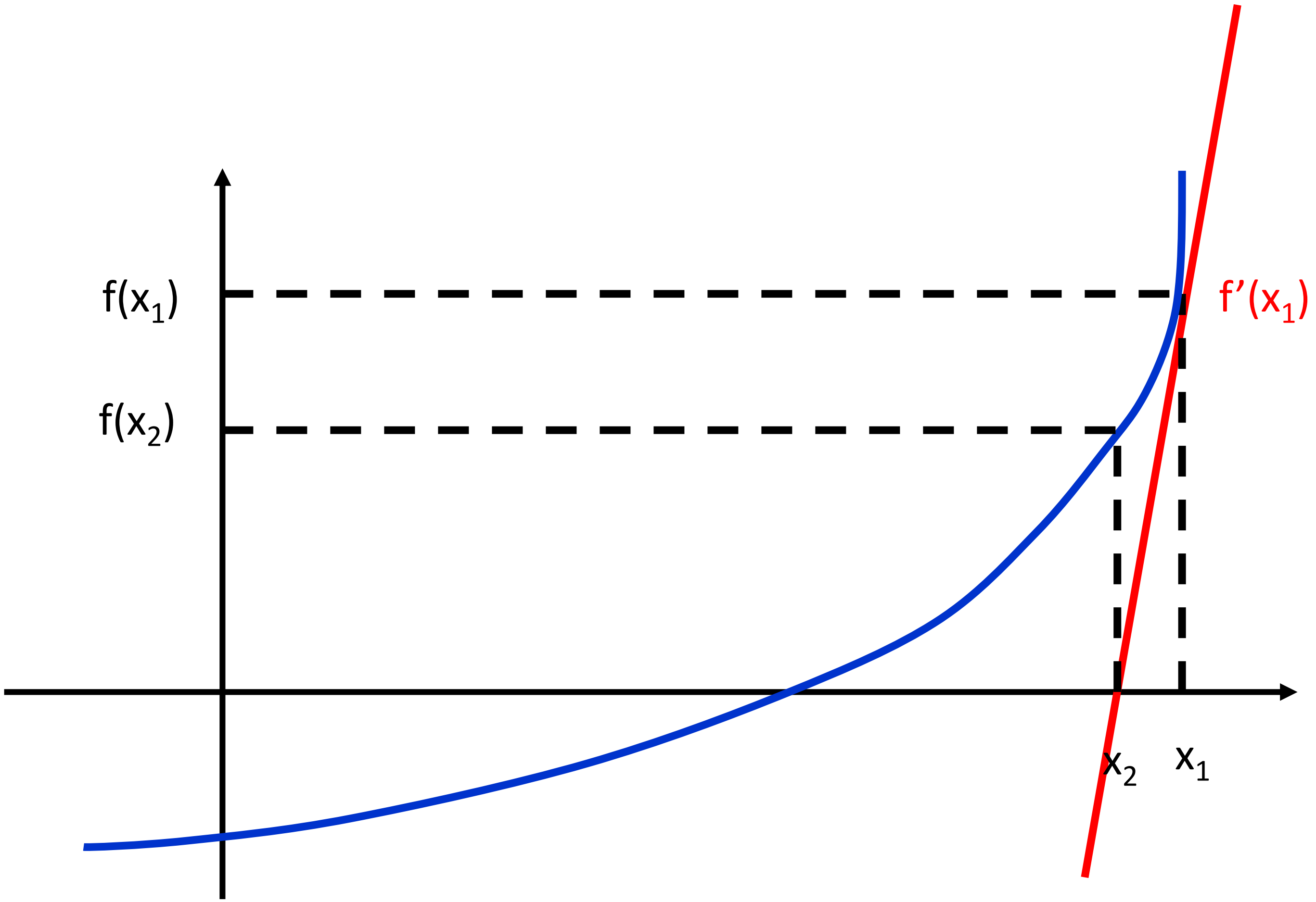
Initial Root Estimation

- Initially, user gives an X_1 value as an estimated root.
- Plug X_1 in the function equation and calculate $f(X_1)$.



Iteration-1

- Calculate X_2 by using the tangent (teğet) formula.
- Plug X_2 in the function equation and calculate $f(X_2)$.
- Test if $f(X_2)$ is zero OR Tolerance > abs(X_2-X_1) for stopping.



Finding X_2 by using the Slope Formula

- The slope (tangent) of function *f* at point X_1 is the drivative of *f.*

$$\boxed{Slope = f'(x_1) = \frac{f(x_1)}{x_1 - x_2}}$$

$$f'(x_1).(x_1 - x_2) = f(x_1)$$

$$f'(x_1).x_1 - f'(x_1).x_2 = f(x_1)$$

$$f'(x_1).x_2 = f'(x_1).x_1 - f(x_1)$$

$$x_2 = \frac{f'(x_1).x_1 - f(x_1)}{f'(x_1)}$$

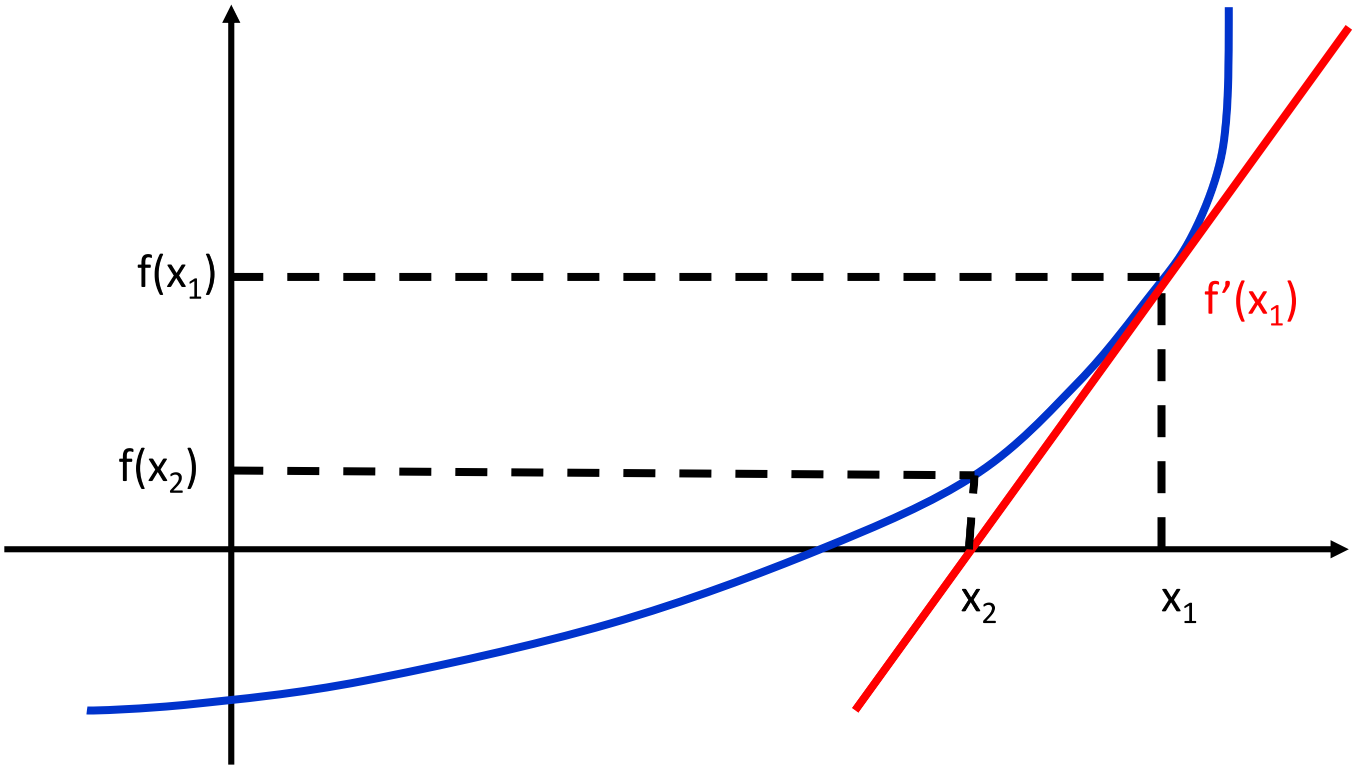
$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \quad \Longrightarrow$$

General iteration formula:

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}}$$

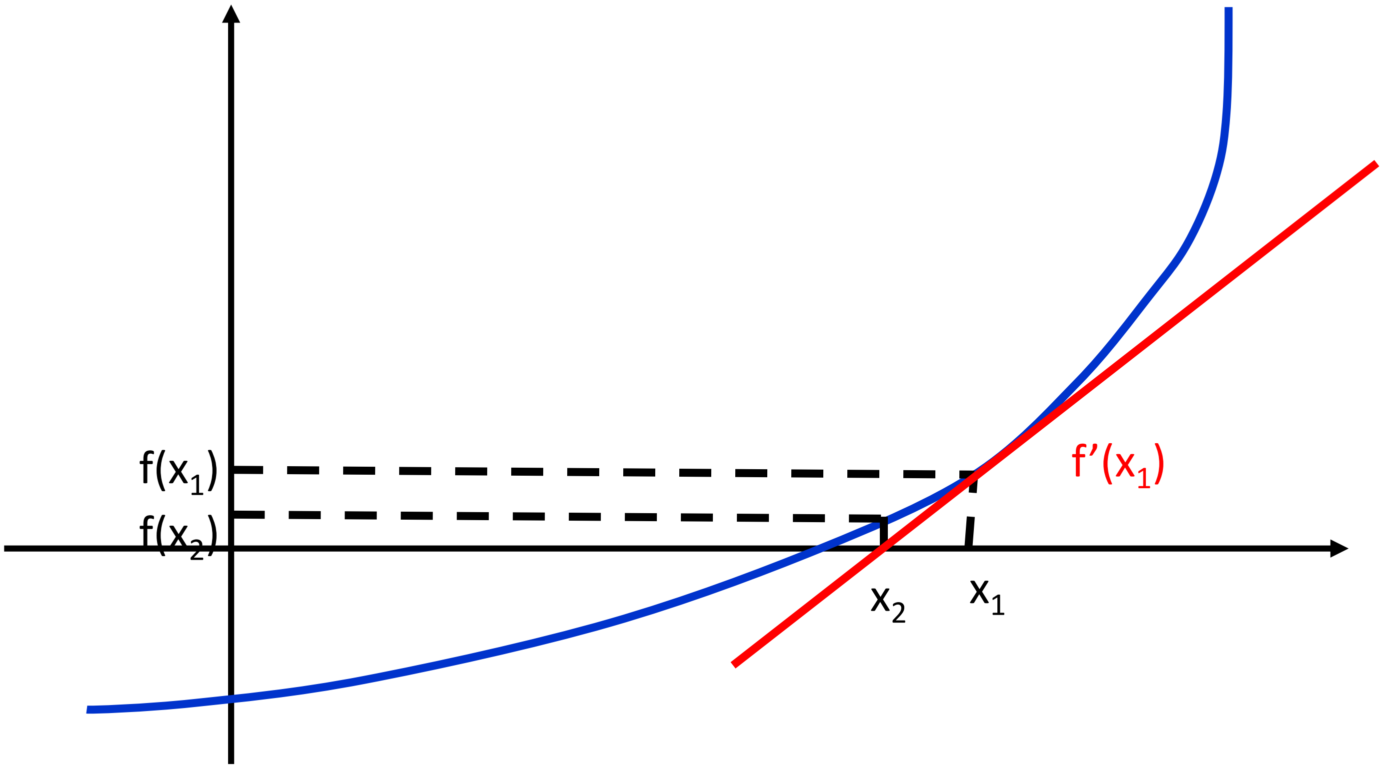
Iteration-2

- Previous X_2 has now become the new X_1 .
- Calculate new X_2 by using tangent formula again.
- Plug X_2 in function equation and calculate $f(X_2)$.
- Test for iteration stopping again.



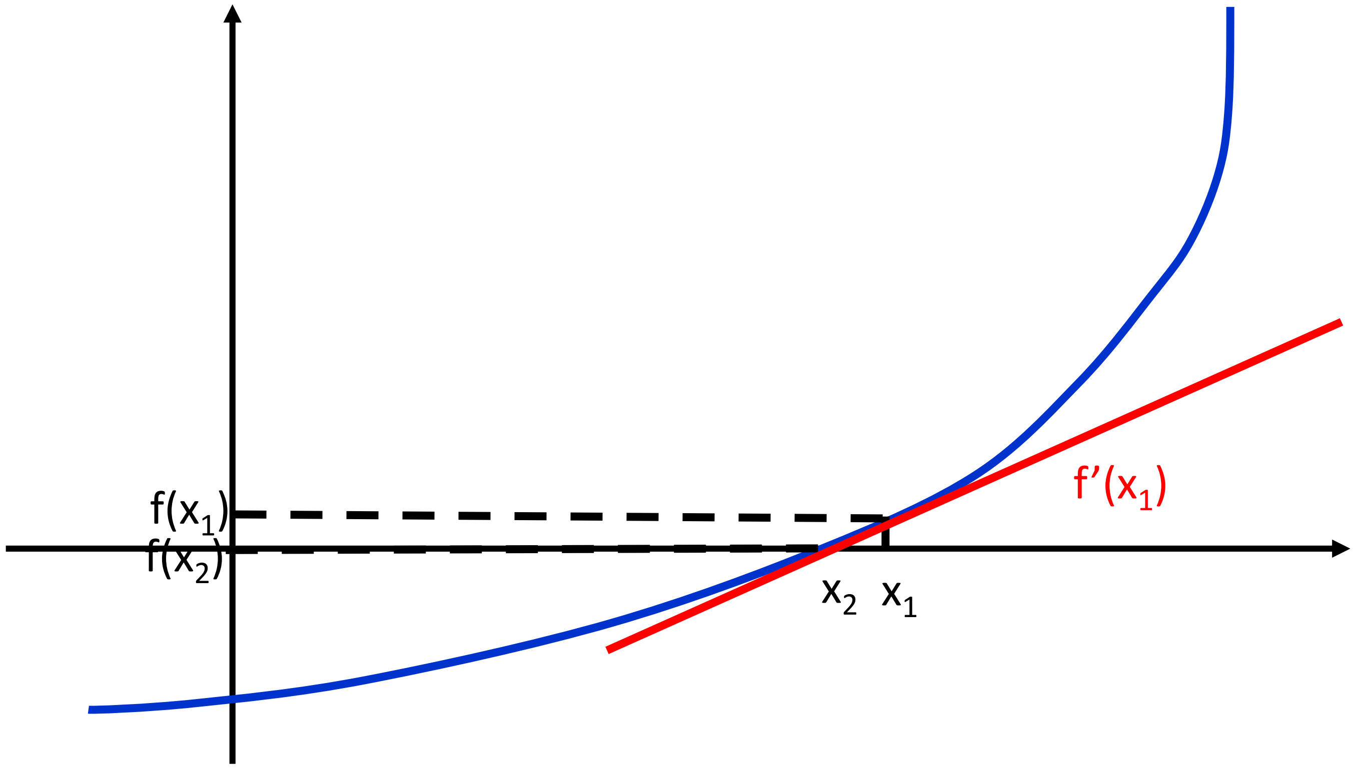
Iteration-3

- Calculate new X_2 by using tangent formula.
- Plug X_2 in function equation and calculate $f(X_2)$.
- Test for iteration stopping.



Iteration-4

- Calculate new X_2 by using tangent formula.
- Plug X_2 in function equation and calculate $f(X_2)$.
- Stopping condition succeeds, iterations stop. The root is the last X_2.



Example function

- Find the root of the following function.

$$\boxed{f(x) = x - \sqrt[3]{x} - 2 \quad = \quad x - x^{\frac{1}{3}} - 2 = 0}$$

- First derivative is:

$$\boxed{f'(x) = 1 - \frac{1}{3} x^{-\frac{2}{3}} = 0}$$

- The iteration formula is:

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}}$$

Program

```
// Root finding for a nonlinear equation using Newton-Raphson method.
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

float fonk(float x) { // Nonlinear function
  return x - pow(x, 1.0/3) - 2 ;
}

float dfonk(float x) { // Derivative of function
  return 1 - (1.0/3) * pow(x, -2.0/3) ;
}


int main() {
  int n=5; // Set as default limit.
  int k;
  float x0,x1,x2,f,dfdx;

  printf("Enter initial guess of root : ");
  scanf("%f", &x0);
  x1 = x0;
```

(continued)

```
printf("  k  f(x)          f'(x)         x(k+1) \n");

for (k=1; k<=n; k++)
{
  f = fonk(x1);
  dfdx = dfonk(x1);
  x2 = x1 -f/dfdx;

  printf("%3d %12.3e %12.3e %18.15f \n", k-1, f, dfdx, x2);

  if (fabs(x2-x1) <= 1.0E-3) { // Tolerance (Convergance testing)
    printf("\nRoot found = %.2f\n", x2);
    return 0; // Stop program
  }
  else
    x1 = x2; // Update x1 (Copy x2 to x1)

} // end for

printf("WARNING: No convergence on root after %d iterations! \n", n);
} // end main
```

Screen output

```
Enter initial guess of root : 3.0


  k  f(x)              f'(x)             x(k+1)
  0  -4.422e-001    8.398e-001  3.526644229888916
  1   4.507e-003    8.561e-001  3.521380186080933
  2   4.103e-007    8.560e-001  3.521379709243774



Root found = 3.5214
```

Example 5:
Line Fitting

Example 5: Line Fitting

- A data file contains set of values
  (x_1,y_1) , (x_2,y_2) , … , (x_N,y_N)

- X is independent variable, Y is dependent variable

- Write a program to do the followings:
  - Read data into X and Y arrays, and display them on screen.
  - Perform a Linear Line Fitting
    (also known as **Regression Analysis**)

- This means calculating the followings for
  y = mx + c equation.
  Slope = m
  Intercept = c

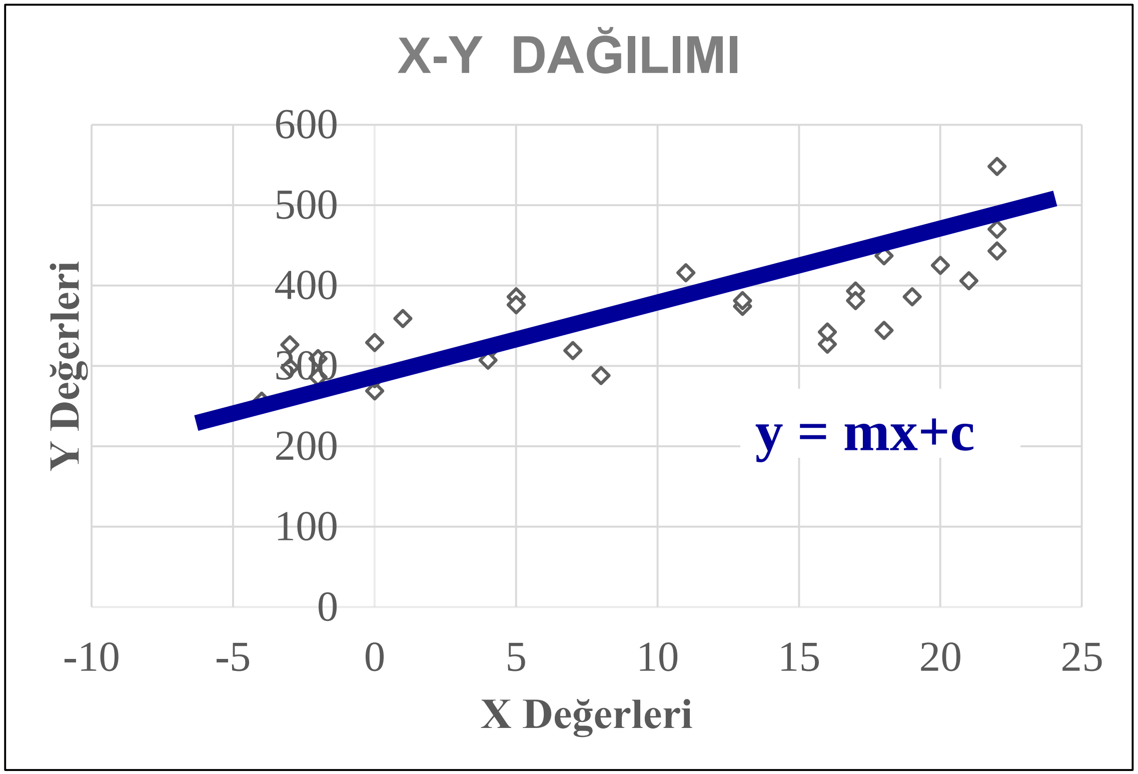
istatistik_veri.txt File

5	386
13	374
17	393
20	425
21	406
18	344
16	327
….	
….	
….	
-2	309
1	359
5	376
11	416
18	437
22	548

Line Fitting formulas

- Calculate and display the m and c values.

$$\bar{x} = \frac{\sum x}{N}$$



$$\bar{y} = \frac{\sum y}{N}$$

$$m = \frac{\frac{\sum xy}{N} - \bar{x}\,\bar{y}}{\frac{\sum x^2}{N} - \bar{x}^2}$$

$$c = \bar{y} - m\bar{x}$$

Program

```
// Line Fitting (Regression Analysis)
#include <stdio.h>
#include <stdlib.h>

int main()
{
int X[100], Y[100];
int N, I=0;               // Count of numbers
float M, C;               // Line slope and intercept
float XBAR, YBAR;         // Mean x and y values
float XSUM=0, YSUM=0, XYSUM=0, XXSUM; // Sum values

FILE * dosya = fopen("istatistik_veri.txt", "r");
if (!dosya) {
  printf("File can not be opened!\n");
  return 0;
}
```

(continued)

```
while (!feof(dosya))
{
   fscanf(dosya, "%d %d", &X[I], &Y[I] );
   //printf("%d %d %d \n", I, X[I], Y[I]);
   XSUM += X[I];
   YSUM += Y[I];
   XXSUM += X[I]*X[I];
   XYSUM += X[I]*Y[I];
   I++;
}

fclose(dosya);
N = I-1;

// Calculate best-fit straight line
XBAR = XSUM / N;
YBAR = YSUM / N;
M = ( XYSUM / N - XBAR * YBAR ) / ( XXSUM / N - XBAR * XBAR);
C = YBAR - M * XBAR;
printf("Line Equation : Y = MX + C \n");
printf("Slope = M = %.2f \n", M);
printf("Intercept = C = %.2f \n", C);

} // end main
```

Screen output

```
Line Equation : Y = MX + C

Slope = M = 4.35

Intercept = C = 329.04
```