

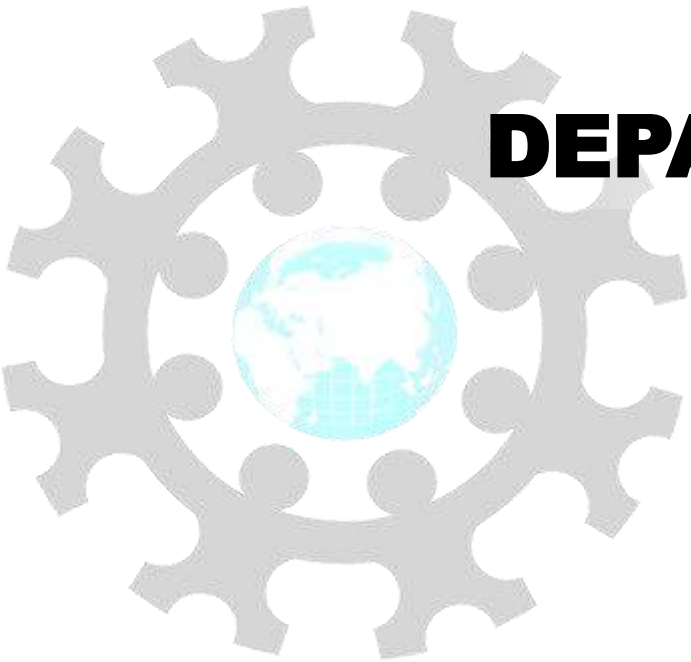


**CHANDIGARH
UNIVERSITY**

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



**NUMERICAL METHODS AND OPTIMIZATION
USING PYTHON**

COURSE CODE- 22CSH-259/22ITH-259

DR. HARDEEP KAUR (15828)

DISCOVER . LEARN . EMPOWER

CHAPTER 1.3

Overview of Numerical Methods

Numerical methods play a crucial role in scientific computing and engineering, and Python has become a popular language for implementing and utilizing these methods. Here are some reasons for the importance of numerical methods in Python and their applications:

Importance of Numerical Methods in Python:

Complex Problem Solving:

Numerical methods are essential for solving complex mathematical problems that may lack analytical solutions. Many real-world problems, especially in science and engineering, require numerical approaches.

Approximation of Solutions:

In many cases, exact solutions are either impossible or impractical to obtain. Numerical methods allow for the approximation of solutions, enabling the analysis of systems and processes.

Computational Efficiency:

Numerical methods often provide efficient algorithms for solving mathematical problems on computers. These methods are designed to handle large datasets and complex computations, making them well-suited for modern computational environments.

Implementation of Mathematical Models:

Numerical methods are crucial for implementing and simulating mathematical models that describe physical, biological, and engineering systems. Python's flexibility and ease of use make it an excellent choice for such implementations.

Optimization and Minimization:

Numerical optimization methods help find the maximum or minimum of a function, which is essential in various fields like finance, engineering design, and machine learning.

Data Analysis and Signal Processing:

Numerical methods are used extensively in data analysis, signal processing, and image processing. Python libraries such as NumPy and SciPy provide powerful tools for handling numerical data and performing operations like Fourier transforms.

Applications of Numerical Methods in Python:

Numerical Integration and Differentiation:

Techniques such as the trapezoidal rule and Simpson's rule are used to approximate definite integrals. Numerical differentiation methods help estimate derivatives.

Root Finding:

Methods like the Newton-Raphson method or the bisection method are employed for finding roots of equations.

Linear Algebra Operations:

Applications in solving linear systems, eigenvalue problems, and matrix factorizations. NumPy is a powerful Python library for linear algebra operations.

Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs):

Solving ODEs and PDEs is critical in physics, engineering, and other fields. Python has libraries like SciPy with functions for solving differential equations.

Monte Carlo Simulation:

Used in finance, risk analysis, and various scientific fields for probabilistic modeling and simulation.

Finite Element Analysis (FEA):

Numerical methods are widely employed in FEA for solving problems in structural mechanics, heat transfer, fluid dynamics, and other engineering disciplines.

Machine Learning and Optimization:

Numerical optimization methods are fundamental in training machine learning models. Techniques like gradient descent are used to minimize cost functions.

Signal Processing:

Fourier transforms and other numerical techniques are essential in signal processing for tasks like filtering and spectral analysis.

The combination of Python's readability, extensive libraries (NumPy, SciPy, SymPy), and a vibrant scientific computing community makes it an excellent platform for implementing and applying numerical methods in various scientific and engineering domains.

THANK YOU

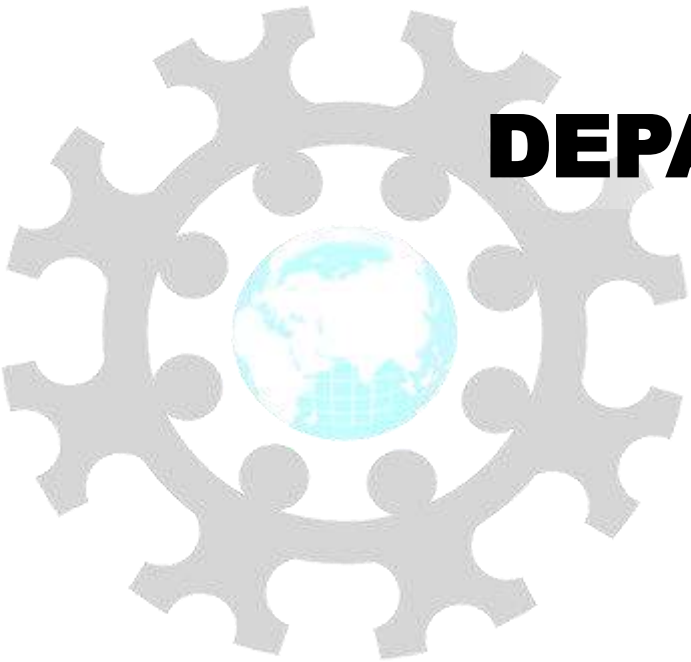


**CHANDIGARH
UNIVERSITY**

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



**NUMERICAL METHODS AND OPTIMIZATION
USING PYTHON**

COURSE CODE- 22CSH-259/22ITH-259

DR. HARDEEP KAUR (15828)

DISCOVER . LEARN . EMPOWER

NumPy and SciPy libraries for numerical computing

NumPy:

Purpose: NumPy (Numerical Python) is the fundamental package for numerical computing in Python.

Key Features:

- Multidimensional arrays: Efficient data structures for representing vectors, matrices, and tensors.
- Mathematical functions: A wide range of mathematical operations that can be applied element-wise to arrays.
- Broadcasting: A powerful mechanism for performing operations on arrays of different shapes.

- Linear algebra, random number generation, and Fourier analysis functions.

Example:

python

```
import numpy as np

# Create a NumPy array
arr = np.array([1, 2, 3, 4, 5])

# Perform operations on arrays
mean_value = np.mean(arr)
```

Overview:

Purpose: SciPy builds on NumPy and provides additional functionality for scientific computing.

Key Features:

- Integration, interpolation, and optimization tools.
- Special functions: Bessel functions, gamma functions, and more.
- Signal and image processing functions.

- Sparse matrix operations and solvers.
- Statistical functions.

Example:

```
python

from scipy.optimize import minimize

# Define an optimization problem
def objective_function(x):
    return x**2 + 4*x + 4

# Minimize the objective function
result = minimize(objective_function, x0=0)
print(result)
```


Installation:

- Both NumPy and SciPy can be installed using the following command:

```
bash
```

```
pip install numpy scipy
```

THANK YOU

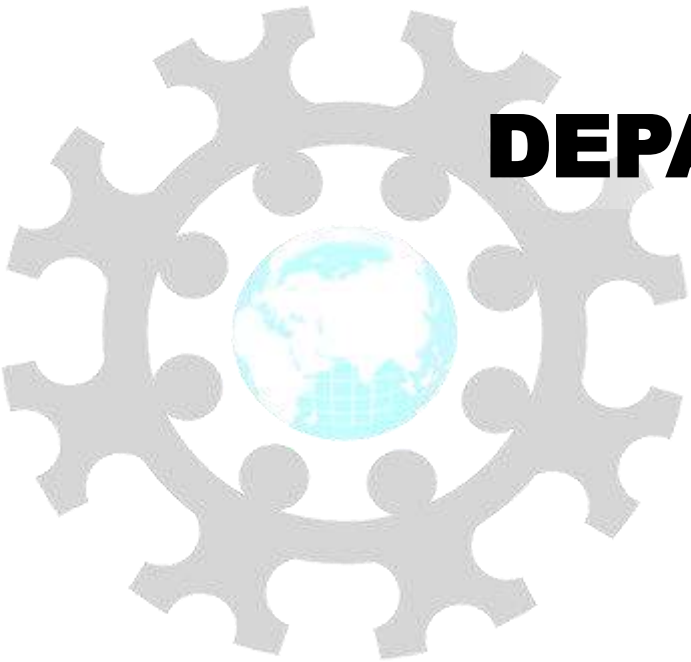


CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



NUMERICAL METHODS AND OPTIMIZATION
USING PYTHON

COURSE CODE- 22CSH-259/22ITH-259

DR. HARDEEP KAUR (15828)

DISCOVER . **LEARN** . EMPOWER

Root-finding Methods : Bisection method

- **Introduction**

- The Bisection Method is a fundamental numerical technique used for finding roots of a continuous function. It is a bracketing method, meaning it works by narrowing down the interval in which a root lies until the interval is sufficiently small. The Bisection Method is particularly useful due to its simplicity and guaranteed convergence for continuous functions where the Intermediate Value Theorem applies.

Basic Principle

- The Bisection Method relies on the Intermediate Value Theorem, which states that if a continuous function $f(x)$ changes sign over an interval $[a,b]$, then there exists at least one root in that interval. The method repeatedly bisects the interval and selects the subinterval in which the function changes sign, thereby narrowing down the interval that contains the root.

Steps of the Bisection Method

1. **Initial Interval Selection:** Choose an initial interval $[a, b]$ such that $f(a)$ and $f(b)$ have opposite signs (i.e., $f(a) \cdot f(b) < 0$).
2. **Midpoint Calculation:** Compute the midpoint c of the interval:

$$c = \frac{a + b}{2}$$

3. **Function Evaluation:** Evaluate the function at the midpoint $f(c)$.
4. **Interval Update:**
 - If $f(c) = 0$, then c is the root, and the process terminates.
 - If $f(c)$ has the same sign as $f(a)$, then the root lies in the interval $[c, b]$. Update a to c .
 - If $f(c)$ has the same sign as $f(b)$, then the root lies in the interval $[a, c]$. Update b to c .
5. **Convergence Check:** Repeat steps 2-4 until the interval $[a, b]$ is sufficiently small, or until $|f(c)|$ is below a predefined tolerance level.

Pseudocode

- Here's a simple pseudocode representation of the Bisection Method:

```
SCSS Copy code  
  
function bisection(f, a, b, tol)  
  if f(a) * f(b) >= 0  
    error "f(a) and f(b) must have opposite signs"  
  end if  
  
  while (b - a) / 2 > tol  
    c = (a + b) / 2  
    if f(c) == 0  
      return c  
    else if f(c) * f(a) < 0  
      b = c  
    else  
      a = c  
    end if  
  end while  
  
  return (a + b) / 2  
end function
```

Example:

Consider finding the root of the function $f(x) = x^2 - 4$ in the interval $[1, 3]$.

1. Initial Interval: $[a, b] = [1, 3]$, $f(1) = -3$, $f(3) = 5$.
2. First Midpoint: $c = \frac{1+3}{2} = 2$, $f(2) = 0$.

Since $f(2) = 0$, $c = 2$ is the root.

Example-1

Find a root of an equation $f(x) = x^3 - x - 1$ using Bisection method

Solution:

Here $x^3 - x - 1 = 0$

Let $f(x) = x^3 - x - 1$

Here

x	0	1	2
$f(x)$	-1	-1	5

1st iteration :

Here $f(1) = -1 < 0$ and $f(2) = 5 > 0$

∴ Now, Root lies between 1 and 2

$$x_0 = \frac{1 + 2}{2} = 1.5$$

$$f(x_0) = f(1.5) = 0.875 > 0$$

2nd iteration :

Here $f(1) = -1 < 0$ and $f(1.5) = 0.875 > 0$

∴ Now, Root lies between 1 and 1.5

$$x_1 = \frac{1 + 1.5}{2} = 1.25$$

$$f(x_1) = f(1.25) = -0.29688 < 0$$

3rd iteration :

Here $f(1.25) = -0.29688 < 0$ and $f(1.5) = 0.875 > 0$

∴ Now, Root lies between 1.25 and 1.5

$$x_2 = \frac{1.25 + 1.5}{2} = 1.375$$

$$f(x_2) = f(1.375) = 0.22461 > 0$$

4th iteration :

Here $f(1.25) = -0.29688 < 0$ and $f(1.375) = 0.22461 > 0$

∴ Now, Root lies between 1.25 and 1.375

$$x_3 = \frac{1.25 + 1.375}{2} = 1.3125$$

$$f(x_3) = f(1.3125) = -0.05151 < 0$$

5th iteration :

Here $f(1.3125) = -0.05151 < 0$ and $f(1.375) = 0.22461 > 0$

∴ Now, Root lies between 1.3125 and 1.375

$$x_4 = \frac{1.3125 + 1.375}{2} = 1.34375$$

$$f(x_4) = f(1.34375) = 0.08261 > 0$$

6th iteration :

Here $f(1.3125) = -0.05151 < 0$ and $f(1.34375) = 0.08261 > 0$

∴ Now, Root lies between 1.3125 and 1.34375

$$x_5 = \frac{1.3125 + 1.34375}{2} = 1.32812$$

$$f(x_5) = f(1.32812) = 0.01458 > 0$$

7th iteration :

Here $f(1.3125) = -0.05151 < 0$ and $f(1.32812) = 0.01458 > 0$

∴ Now, Root lies between 1.3125 and 1.32812

$$x_6 = \frac{1.3125 + 1.32812}{2} = 1.32031$$

$$f(x_6) = f(1.32031) = -0.01871 < 0$$

Approximate root of the equation $x^3 - x - 1 = 0$ using Bisection method is 1.32471

n	a	$f(a)$	b	$f(b)$	$c = \frac{a+b}{2}$	$f(c)$	Update
1	1	-1	2	5	1.5	0.875	$b = c$
2	1	-1	1.5	0.875	1.25	-0.29688	$a = c$
3	1.25	-0.29688	1.5	0.875	1.375	0.22461	$b = c$
4	1.25	-0.29688	1.375	0.22461	1.3125	-0.05151	$a = c$
5	1.3125	-0.05151	1.375	0.22461	1.34375	0.08261	$b = c$
6	1.3125	-0.05151	1.34375	0.08261	1.32812	0.01458	$b = c$
7	1.3125	-0.05151	1.32812	0.01458	1.32031	-0.01871	$a = c$
8	1.32031	-0.01871	1.32812	0.01458	1.32422	-0.00213	$a = c$
9	1.32422	-0.00213	1.32812	0.01458	1.32617	0.00621	$b = c$
10	1.32422	-0.00213	1.32617	0.00621	1.3252	0.00204	$b = c$
11	1.32422	-0.00213	1.3252	0.00204	1.32471	-0.00005	$a = c$

Advantages and Disadvantages

- **Advantages:**

- **Simplicity:** The method is easy to understand and implement.
- **Guaranteed Convergence:** It always converges to a root if the initial interval is chosen correctly.
- **Robustness:** Works well for a wide range of functions.

- **Disadvantages:**

- **Slow Convergence:** The method converges linearly, which can be slow compared to other methods like Newton's method.
- **Initial Interval Requirement:** Requires an initial interval where the function changes sign.

Applications

- The Bisection Method is used in various fields such as engineering, physics, economics, and any area that requires finding roots of equations. It is particularly useful in scenarios where the function is not well-behaved or where other root-finding methods may fail.

Conclusion

- The Bisection Method is a powerful tool in numerical analysis for finding roots of continuous functions. Its simplicity and guaranteed convergence make it a reliable choice, especially when dealing with functions that exhibit problematic behavior. Despite its slow convergence, its robustness ensures that it remains a valuable method in the numerical analyst's toolkit.

THANK YOU

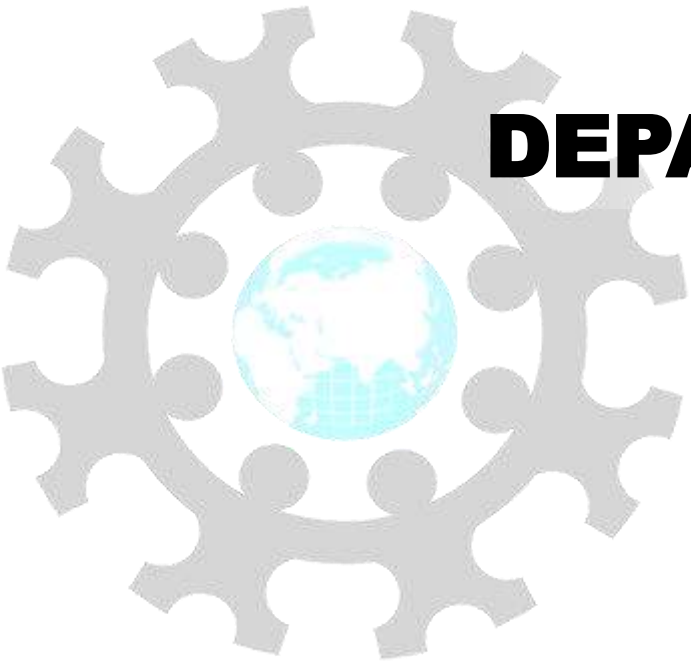


CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



NUMERICAL METHODS AND OPTIMIZATION
USING PYTHON

COURSE CODE- 22CSH-259/22ITH-259

DR. HARDEEP KAUR (15828)

DISCOVER . **LEARN** . EMPOWER

Newton-Raphson Method

- **Introduction**

- The Newton-Raphson method, also known simply as Newton's method, is a powerful and widely used technique for finding successively better approximations to the roots (or zeroes) of a real-valued function. It is an iterative method that uses the derivative of the function to find its roots. This method is particularly useful because of its rapid convergence, especially when close to the root.

- **Basic Principle**

- The Newton-Raphson method is based on the idea of linear approximation. Given a function $f(x)$ and its derivative $f'(x)$, we start with an initial guess x_0 for a root of $f(x)=0$. The function is approximated by its tangent line at x_0 and the root of this tangent line is taken as the next approximation x_1 . This process is repeated until convergence.

Mathematical Formulation

- The iterative formula for the Newton-Raphson method is:

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

Where:

- x_n is the current approximation.
- x_{n+1} is the next approximation.
- $f(x_n)$ is the function value at x_n .
- $f'(x_n)$ is the derivative of the function at x_n .

Steps of the Newton-Raphson Method

1. **Initial Guess:** Start with an initial guess x_0 .
2. **Iterative Process:** Apply the Newton-Raphson formula to compute the next approximation:


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

3. **Convergence Check:** Check if $|x_{n+1} - x_n|$ or $|f(x_{n+1})|$ is below a predefined tolerance level.
4. **Repeat:** Repeat steps 2 and 3 until convergence.

Pseudocode

- Here's a simple pseudocode representation of the Newton-Raphson method:

lua

 Copy code

```
function newtonRaphson(f, df, x0, tol, maxIter)
  x = x0
  for i = 1 to maxIter
    x_new = x - f(x) / df(x)
    if abs(x_new - x) < tol
      return x_new
    end if
    x = x_new
  end for
  error "Method did not converge"
end function
```

Example

Consider finding the root of the function $f(x) = x^2 - 2$.

1. Function and Derivative:

$$f(x) = x^2 - 2$$

$$f'(x) = 2x$$

2. Initial Guess: $x_0 = 1$.

3. First Iteration:

$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)} = 1 - \frac{1^2 - 2}{2 \cdot 1} = 1 + 0.5 = 1.5$$

4. Second Iteration:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} = 1.5 - \frac{1.5^2 - 2}{2 \cdot 1.5} = 1.5 - \frac{0.25}{3} = 1.4167$$

5. Third Iteration:

$$x_3 = x_2 - \frac{f(x_2)}{f'(x_2)} = 1.4167 - \frac{1.4167^2 - 2}{2 \cdot 1.4167} \approx 1.4142$$

Continue this process until the change in x is smaller than the predefined tolerance.

Advantages and Disadvantages

- **Advantages:**

- **Rapid Convergence:** The method typically converges very quickly when near the root.
- **Simplicity:** The formula is straightforward to implement and understand.
- **Wide Applicability:** Useful for a variety of functions, provided the derivative can be computed.

- **Disadvantages:**

- **Derivative Requirement:** The method requires the function to be differentiable and the derivative to be known.
- **Initial Guess Sensitivity:** Poor initial guesses can lead to divergence or slow convergence.
- **Multiple Roots and Singularities:** The method may fail or behave unpredictably near multiple roots or singularities.

Applications

- The Newton-Raphson method is widely used in scientific computing, engineering, and mathematics. It is particularly useful in solving nonlinear equations, optimizing functions, and in various numerical simulations where root-finding is necessary.

Conclusion

- The Newton-Raphson method is a robust and efficient tool for finding the roots of functions. Its fast convergence and simplicity make it a popular choice for numerical analysts and engineers. However, care must be taken in choosing the initial guess and ensuring the function and its derivative are well-behaved to guarantee the method's success.

THANK YOU

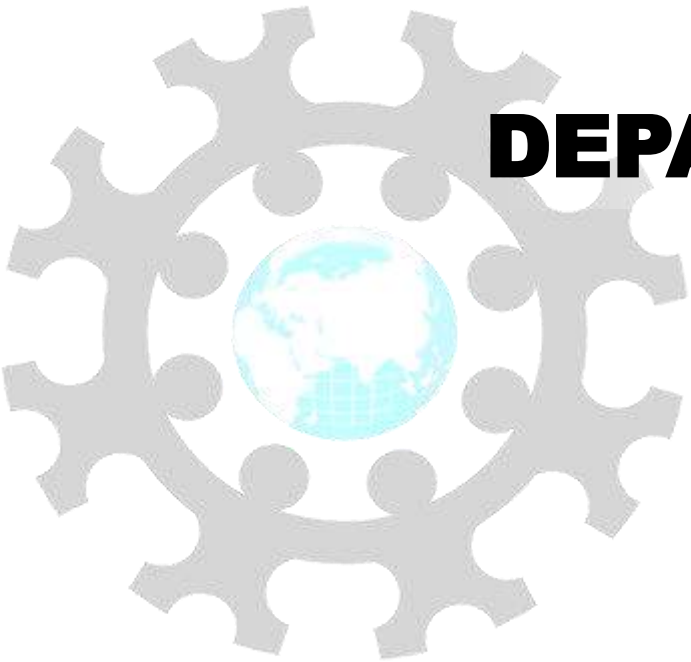


CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

UNIVERSITY INSTITUTE OF ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING



NUMERICAL METHODS AND OPTIMIZATION
USING PYTHON

COURSE CODE- 22CSH-259/22ITH-259

DR. HARDEEP KAUR (15828)

DISCOVER . **LEARN** . EMPOWER

Secant Method

- **Introduction**
- The Secant Method is a numerical technique used to find the roots of a real-valued function. It is similar to the Newton-Raphson method but does not require the computation of the derivative of the function. Instead, it uses a sequence of secant lines, which are lines that intersect the graph of the function at two points, to approximate the root. This method is especially useful when the derivative of the function is difficult to compute.

Basic Principle

The Secant Method approximates the root of a function $f(x)$ by using two initial approximations, x_0 and x_1 . The method iteratively refines these approximations by constructing secant lines between these points and using the x-intercepts of these lines as new approximations.

Mathematical Formulation

The iterative formula for the Secant Method is:

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

Where:

- x_n is the current approximation.
- x_{n-1} is the previous approximation.
- $f(x_n)$ and $f(x_{n-1})$ are the function values at x_n and x_{n-1} , respectively.

Steps of the Secant Method

1. **Initial Guesses:** Start with two initial approximations x_0 and x_1 .
2. **Iterative Process:** Apply the Secant Method formula to compute the next approximation:


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

3. **Convergence Check:** Check if $|x_{n+1} - x_n|$ or $|f(x_{n+1})|$ is below a predefined tolerance level.
4. **Repeat:** Repeat steps 2 and 3 until convergence.

Pseudocode

Here's a simple pseudocode representation of the Secant Method:

lua

 Copy code

```
function secantMethod(f, x0, x1, tol, maxIter)
  for i = 1 to maxIter
    x2 = x1 - f(x1) * (x1 - x0) / (f(x1) - f(x0))
    if abs(x2 - x1) < tol
      return x2
    end if
    x0 = x1
    x1 = x2
  end for
  error "Method did not converge"
end function
```

Example

Consider finding the root of the function $f(x) = x^2 - 4$.

1. Initial Guesses: $x_0 = 1$ and $x_1 = 3$.
2. First Iteration:

$$x_2 = x_1 - \frac{f(x_1)(x_1 - x_0)}{f(x_1) - f(x_0)} = 3 - \frac{(3^2 - 4)(3 - 1)}{(3^2 - 4) - (1^2 - 4)} = 3 - \frac{5 \cdot 2}{5 - (-3)}$$

3. Second Iteration:

$$x_3 = x_2 - \frac{f(x_2)(x_2 - x_1)}{f(x_2) - f(x_1)} = 1.75 - \frac{(1.75^2 - 4)(1.75 - 3)}{(1.75^2 - 4) - (3^2 - 4)} = 1.75 -$$

4. Third Iteration:

$$x_4 = x_3 - \frac{f(x_3)(x_3 - x_2)}{f(x_3) - f(x_2)} = 1.9474 - \frac{(1.9474^2 - 4)(1.9474 - 1.75)}{(1.9474^2 - 4) - (1.75^2 - 4)}$$

Continue this process until the change in x is smaller than the predefined tolerance.

Advantages and Disadvantages

- **Advantages:**

- **No Derivative Required:** The method does not require the computation of the derivative, making it suitable for functions where the derivative is difficult or impossible to compute.
- **Simplicity:** The formula is straightforward and easy to implement.
- **Faster Convergence:** Generally converges faster than the Bisection Method.

- **Disadvantages:**

- **Initial Guess Sensitivity:** The method requires two initial guesses that should be reasonably close to the actual root for faster convergence.
- **Convergence Issues:** The method may not converge if the function is not well-behaved between the initial guesses or if the secant line becomes nearly horizontal.
- **Multiple Roots:** The method can sometimes be unreliable near multiple roots.

Applications

- The Secant Method is widely used in various scientific and engineering fields for solving nonlinear equations where the derivative of the function is not readily available. It is particularly useful in root-finding problems and numerical simulations.

Conclusion

- The Secant Method is an effective and efficient root-finding technique that does not require the computation of derivatives. Its simplicity and relatively fast convergence make it a valuable tool in numerical analysis, especially when dealing with complex functions. However, careful selection of initial guesses is crucial for the method's success and reliability.

THANK YOU