CECC421: Numerical Analysis
Lecture 4: Finding the Roots of Functions

Eng. Aya Kherbek
Eng. Baher Kherbek
Faculty of Engineering
Department of Informatics
Manara University

Purpose of the Session:

Studying algorithms for finding function roots using numerical methods and converting them into Python programming code.

Bisection Method Algorithm

The **Bisection Method**, also known as **interval halving**, is one of the numerical techniques used to find the root of a function. It works by iteratively halving the interval where the root lies and selecting a subinterval that also contains the root to refine the computation. Although it is very simple and robust, the bisection method is relatively slow.

If the function **f(x) = 0** is continuous and defined in the interval **[a, b]**, where **f(a) * f(b) < 0**, meaning they have opposite signs, this indicates that at least one root of the function **f(x)** lies within the interval **[a, b]**. In this case, we follow the following algorithm to solve for the root:

1.Check if the interval contains a root:

1.1. Compute the midpoint of the values **a** and **b**, denoted as x_1, where $x_1 = (a + b) / 2$.

1.2. If $f(x_1) < tol$, then x_1 is a root of the function **f(x)** and a solution to the equation.

1.3. Otherwise, if $f(x_1) * f(b) < 0$, set $a = x_1$ to refine the interval and approach the solution.

1.4. Otherwise, if $f(x_1) * f(a) < 0$, set $b = x_1$ to refine the interval and approach the solution.

2.Repeat steps 1.1, 1.2, 1.3, and 1.4 until reaching a value where $f(x_i) = 0$ or $f(x_i) < tol$, where **tol** represents the desired accuracy level of the solution.

3.Otherwise, there is no solution in this interval.

Use **Python** to:

1.Write a function to find the approximate root of a given function using the **Bisection Method** algorithm.

2.Use the previously written function to find the approximate roots of the equation $f(x) = x^3 - 9x + 1$ within the interval **[2,4]** with different accuracies:

1. With accuracy $\varepsilon = 0.5$, print the value of the function **f** at this root.
2. Then, with accuracy $\varepsilon = 0.05$, print the value of the function **f** at this root.
3. Finally, with accuracy $\varepsilon = 0.0005$, print the value of the function **f** at this root.

1. **Writing a function that can be called each time for a new function** (as we learned earlier).
2. **Conditional statements in Python** (such as if, elif, and else for decision-making).
3. **Using Lambda expressions** to define the function that needs to be studied.

In Python, there are three forms of the if...else statement.

1. if statement
2. if...else statement
3. if...elif...else statement

Python Nested if statements

We can also use an if statement inside of an if statement. This is known as a nested if statement.

The syntax of nested if statement is:

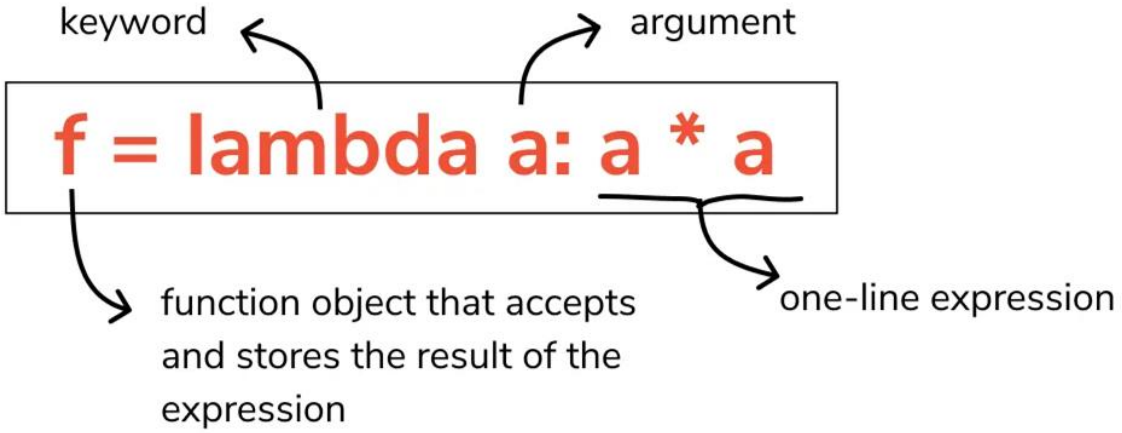
```
# outer if statement
if condition1:
    # statement(s)
    # inner if statement
    if condition2:
     # statement(s)
```

Notes:

- We can add `else` and `elif` statements to the inner `if` statement as required.
- We can also insert inner `if` statement inside the outer `else` or `elif` statements(if they exist)
- We can nest multiple layers of `if` statements.

The necessary tools to solve the exercise:

In Python, a Lambda function is an anonymous function, meaning it is a function without a name.
It can have any number of parameters but only a single expression, which is evaluated and returned. It is not necessary for it to have a return value.



```python
import numpy as np
def my_bisection(f, a, b, tol):
        # check if a and b bound a root
    if f(a)* f(b)<0:
       # get midpoint
        mid = (a + b)/2
        if np.abs(f(mid)) < tol:
         # stopping condition, report m as root
              return mid
        elif f(a)*f(mid)<0:
         # case where mid is an improvement on a.
         # Make recursive call with a = m
            return my_bisection(f, a, mid, tol)
        elif f(b)*f(mid)<0:
         # case where mid is an improvement on b.
         # Make recursive call with b = m
            return my_bisection(f, mid, b, tol)
    else:
       print("No Solution")
```

The programming code related to writing the function

Note that this function is a recursive function that calls itself from within its body.

Using my_bisection function

```
f = lambda x: x**3 - 9*x + 1


r1 = my_bisection(f, 2, 4, 0.5)
print("r1 =", r1)
print("f(r1) =", f(r1),"\n")


r2 = my_bisection(f, 2, 4, 0.05)
print("r2 =", r2)
print("f(r2) =", f(r2),"\n")


r3 = my_bisection(f, 2, 4, 0.0005)
print("r3 =", r3)
print("f(r3) =", f(r3))
```

The programming code related to using the previous function to determine the root of the function:

$$f(x) = x^3 - 9x + 1$$

within the range **[2,4]**, with precision

ε = 0.5

ε = 0.05

and accuracy ε=0.0005.

OUTPUT

```
r1 = 2.9375
f(r1) = -0.090087890625

r2 = 2.9453125
f(r2) = 0.04237794876098633

r3 = 2.94281005859375
f(r3) = -0.0001697912650797661 6
```

The output of the programming code related to using the previous function to determine the root of the function:

$$f(x) = x^3 - 9x + 1$$

within the range $[2,4]$, with precision:

$\varepsilon = 0.5$

$\varepsilon = 0.05$

and $\varepsilon = 0.0005$.

Notice that the tolerance or precision value affects both the root r and the function value f.

The programming code related to using the previous function to determine the root of the function:

$$f(x) = x^3 - 9x + 1$$

within the range **[3,4]**, with precision

ε=0.5.

```
f = lambda x: x**3 - 9*x + 1


r1 = my_bisection(f, 3, 4, 0.5)
print("r1 =", r1)
print("f(r1) =", f(r1),"\n")
```

No Solution
r1 = None

TypeError Traceback (most recent call last)

<ipython-input-11-7ae2f9b8cd4b> in <cell line: 5>()
3 r1 = my_bisection(f, 3, 4, 0.5)
4 print("r1 =", r1)
----> 5 print("f(r1) =", f(r1),"\n")

<ipython-input-11-7ae2f9b8cd4b> in <lambda>(x)
----> 1 f = lambda x: x**3 - 9*x + 1
2
3 r1 = my_bisection(f, 3, 4, 0.5)
4 print("r1 =", r1)
5 print("f(r1) =", f(r1),"\n")

TypeError: unsupported operand type(s) for ** or pow(): 'NoneType' and 'int'

The output of the programming code related to using the previous function to determine the root of the function:

$$f(x) = x^3 - 9x + 1$$

within the range $[3,4]$, with precision:

$$\varepsilon = 0.5$$

Newton_Raphson Algorithm

An iterative algorithm that starts from an initial point considered an approximate solution to the function and refines it to approach the actual solution.

If the function f(x)=0 is defined, continuous, and differentiable in the interval [a,b], and we have an initial value x_0 close to the solution, we follow the following algorithm to find the solution:

1.Check if f(x_0)<tol:

1.1 - Return x_0 as the root of the function.

2. Otherwise, calculate the new value:

$$x_0 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

3.Repeat steps 1 and 2 until the solution is found.

Since the algorithm is iterative, the exercise can be solved using loops or recursive functions. Note that the problem statement requires the solution to be implemented as a function.

Use Python:

To write a function to find the approximate root of a given function using the **Newton-Raphson algorithm**.

Use the previous function to find the approximate root of the equation:

$$f(x) = x^3 - 9x + 1$$

starting from an initial guess of **1.5**, with a precision of **1e-6**.

Print the function value f at this root on the screen.

Note: The value **1e-6** represents **0.000001**

The programming code related to writing the function:

```
import numpy as np
from mpmath import *   #importing differntiation

def my_newton(f, df, x0, tol):
  # output is an estimation of the root of f
  # using the Newton Raphson method
  # recursive implementation
  if abs(f(x0)) < tol:
     return x0
  else:
     return my_newton(f, df, x0 - (f(x0)/df(x0)), tol)
```

The programming code related to using the previous function to determine the root of the function:

$$f(x) = x^3 - 9x + 1$$

within the range [3,4], with precision ε=0.5.

```
f = lambda x: x**3-x - 1
df=lambda x:diff(f,x, 1)

newton_raphson = my_newton(f,df,1.5, 1e-6)
print("newton_raphson =", newton_raphson)
print(f(newton_raphson))
```

```
newton_raphson = 1.32471817399905
9.24377759670136e-7
```

جَامعة
المَنارة
MANARA UNIVERSITY

Thanks for Listening