CECC421: Numerical Analysis
Lecture 5+6: Finding the Roots of Functions-2

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.

Fixed Point Algorithm

The fixed-point method differs from previous methods in that it starts from the function $f(x)=0$ to obtain the form $g(x)-x=0$.
Then, this equation is used to derive the iterative formula $x_{i+1}=g(x_i)$ to find the sequence $x_0, x_1, x_2, \ldots$, starting from an initial approximation.

If we have the function f(x) = 0 expressed as x = g(x), then:

1. Define the necessary inputs for the algorithm:
    1. The initial point x_0
    2. The error tolerance tol
    3. The function $f(x)$
    4. The initial value for the iteration count
2. Repeat the following steps:
    1. Compute $x_1 = g(x_0)$
    2. If |x1−x0|<tol, then x_1 is a root of $f(x)$. Print the root value and stop the loop.
    3. Set $x_0 = x_1$ and increase the iteration count.
3. Print the statement **"No solution found."**

In Python:

1- Use loops to find the approximate root of a given function $f(x) = x - e^{-x} = 0$ using the fixed-point algorithm.

- For the starting point, $x_0 = 0$, and for 10 iterations, with accuracy tol=1e−6, print the value of the root and the value of the function f at this root if found.
- If no solution exists, print the statement **"No solution found."**

2- Output your results in a table.

Since the algorithm is iterative, the exercise can be solved using loops, recursive functions, or by using a function inside a loop. Note that the exercise text specifically requests a solution using loops.

"The tools needed to solve the exercise:"

1. Conditional statements in Python.
2. Using Lambda (to define the required function).
3. Using a while loop.
4. Using the tabulate library.

Python while Loop

```
while condition:
    # body of while loop
```

Python While loop with else

```
while condition:
    # body of while loop
else:
    # body of else
```

In **Python**, a while loop can include an optional else block.

- The else block is executed **only** after the loop condition evaluates to False.
- **Note:** The else block will **not** execute if the while loop is terminated by a break statement.

Python break and continue

```
while condition:
    # code
    if condition:
        break

    # code
```

```
while condition:
    # code
    if condition:
        continue

    # code
```

The necessary tools to solve the exercise:

```
from tabulate import tabulate

print(tabulate(all_data,headers='firstrow',tablefmt='grid'))
```

A list representing the rows of a table, where each element in this list is itself a list.

Table Head

TABLE SHAPE

```
import numpy as np
import math
from tabulate import tabulate

tol=1e-6
x0=0
i=1
n=10
f=lambda x:x-math.exp(-x)
header=["i","xi","f(xi)","|epsilon|,%"]
# epsilon=np.abs((x1-x0)/x1)*100
mydata=[[0,0,f(0),     ]]
```

```
while i<=n:
  x1=math.exp(-x0)
  if np.abs(x1-x0)<tol:
    print("x1=",x1)
    k=f(x1)
    print(k)
    break
  epsilon=np.abs((x1-x0)/x1)*100
  mydata.append([i,x1,f(x1),epsilon])
  x0=x1
  i=i+1
  else :
  print("There Is No Solution")
print(tabulate(mydata, headers=header, tablefmt="grid"))
```

```
There Is No Solution
+-----+----------+-------------+---------------+
|   i |       xi |       f(xi) |   |epsilon|,% |
+=====+==========+=============+===============+
|   0 | 0        | -1          |               |
+-----+----------+-------------+---------------+
|   1 | 1        |  0.632121   |     100       |
+-----+----------+-------------+---------------+
|   2 | 0.367879 | -0.324321   |     171.828   |
+-----+----------+-------------+---------------+
|   3 | 0.692201 |  0.191727   |      46.8536  |
+-----+----------+-------------+---------------+
|   4 | 0.500474 | -0.10577    |      38.3091  |
+-----+----------+-------------+---------------+
|   5 | 0.606244 |  0.0608477  |      17.4468  |
+-----+----------+-------------+---------------+
|   6 | 0.545396 | -0.0342165  |      11.1566  |
+-----+----------+-------------+---------------+
|   7 | 0.579612 |  0.0194969  |       5.90335 |
+-----+----------+-------------+---------------+
|   8 | 0.560115 | -0.0110277  |       3.48087 |
+-----+----------+-------------+---------------+
|   9 | 0.571143 |  0.00626377 |       1.9308  |
+-----+----------+-------------+---------------+
|  10 | 0.564879 | -0.00354938 |       1.10887 |
+-----+----------+-------------+---------------+
```

OUTPUT

Use Python:

1. Write a function to find the approximate root of the given function $f(x) = x - e^{-x} = 0$ using the fixed-point algorithm.
   - For the initial point $x_0 = 0$, perform 10 iterations with a precision of tol = 1e−6.
   - Print the root value and the function value f at this root if found.
   - If no solution exists, print the statement **"No solution found."**
2- Output your results in a table.

```
import numpy as np
import math
from tabulate import tabulate
def My_FixedPoint(x0,tol,f,n):
 i=1
 while i<=n:
  x1=math.exp(-x0)
  if np.abs(x1-x0)<tol:
   print("x1=",x1)
   k=f(x1)
   print(k)
   p=0
   break
  epsilon=np.abs((x1-x0)/x1)*100
  mydata.append([i,x1,f(x1),epsilon])
  x0=x1
  p=1
  i=i+1
 if p==1:
  print("There Is No Solution")
```

```
tol=1e-6
x0=0
n=10
f=lambda x:x-math.exp(-x)
header=["i","xi","f(xi)","|epsilon|,%"]# epsilon=np.abs((x1-x0)/x1)*100
mydata=[[0,0,f(0),    ]]
My_FixedPoint(x0,tol,f,n)
print(tabulate(mydata, headers=header,tablefmt="grid"))
```

There Is No Solution

i	xi	f(xi)	|epsilon|,%
0	0	-1	
1	1	0.632121	100
2	0.367879	-0.324321	171.828
3	0.692201	0.191727	46.8536
4	0.500474	-0.10577	38.3091
5	0.606244	0.0608477	17.4468
6	0.545396	-0.0342165	11.1566
7	0.579612	0.0194969	5.90335
8	0.560115	-0.0110277	3.48087
9	0.571143	0.00626377	1.9308
10	0.564879	-0.00354938	1.10887

OUTPUT

Secant Algorithm

The secant method is an open method that starts with two initial guesses to find the real root of nonlinear equations.

In the **Secant method**, if x_0 and x_1 are initial guesses, the next approximate root x_2 is obtained using the following formula:

$$x_2 = x_1 - (x_1 - x_0) * \frac{f(x_1)}{f(x_1) - f(x_0)}$$

The **Secant algorithm** involves **repeating** the above process, meaning we use x_1 and x_2 to find x_3 , and so on, until we locate the root within the required accuracy.

1. Start
2. Define function as f(x)
3. Input initial guesses (x0 and x1),tolerable error (e) and maximum iteration (N)
4. Calculate x2 = x1 - (x1-x0) * f(x1) / (f(x1) - f(x0))
5. Increment iteration counter i = i + 1
6. If i >= N then print "Not Convergent" and goto (11) otherwise goto (9)
7. If |f(x2)| > e then set x0 = x1, x1 = x2 and goto (5) otherwise goto (10)
8. Print root as x2
9. Stop

Use Python:

1. Find the approximate root of a given function using the **Secant method**.
   - Use this method to find the approximate root of the equation
   $$f(x) = e^x + 3x \ .$$
   - Start with initial guesses $x_0 = -1,\ x_1 = 1.$
   - Perform **10 iterations** with an accuracy of 1e−4.
   - Print the function value f at the root.

2- Display the values in a table.

```
import numpy as np
from tabulate import tabulate

f=lambda x: np.exp(x)+3*x
x0=-1;x1=1
i=0
n=10

header=["i","x0","x1","f(x1)","f(x1
)<Epsilon"]
my_secant_result=[[0,-
1,1,f(1),f(1)<1E-4]]
```

```
while i<=n :
  i=i+1
  x2=x1-(f(x1)*(x1-x0)/(f(x1)-f(x0)))
  if np.abs(f(x2))<=1e-4:
    print(x2)
    k=f(x2)
    print(k)
    break
  x0=x1
  x1=x2

my_secant_result.append([i,x0,x1,f(x1)
,np.abs(f(x1))<1E-4])
else:
  print("No Solution For This
Function")
print(tabulate(my_secant_result,
headers=header, tablefmt="grid"))
```

جَامعة
المَنارة
MANARA UNIVERSITY

Thanks for Listening