BASIC TECHNIQUES FOR COMPUTER SIMULATIONS

ROOT FINDING METHODS

INTRODUCTION TO WPO SESSIONS

- Each WPO presentation contains an overview of the numerical techniques discussed during the theoretical sessions, with some useful advice on how to apply them using **Python**

- Our Python scripts will be developed and executed in **Spyder**
  (For installation instructions, visit: https://docs.spyder-ide.org/current/installation.html)

- The pdf file with the exercises, which can be downloaded from CANVAS, also includes details and instructions for the submission

- After the presentation, I will assign the homework, which must be submitted within 10 days from today. You will start in class and continue at home.

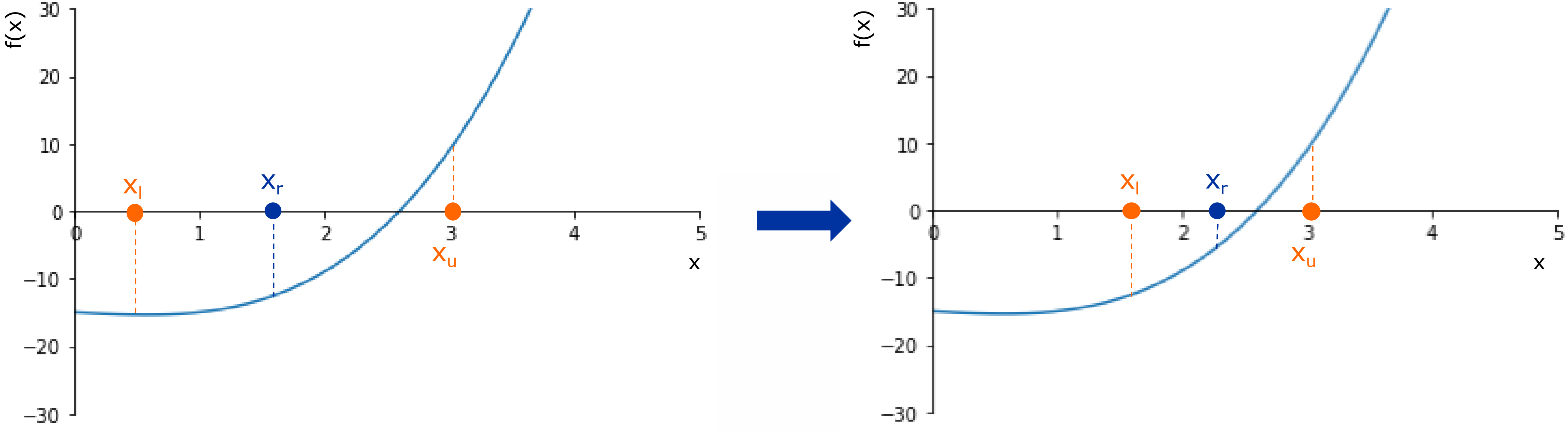
Contact:
matteo.gravili@vub.be

ROOT-FINDING METHODS

- By hand: mathematical formulas → **exact solution** $f(x) = x^2 - 4 = 0$
- Computer: numerical methods → **approximation** $f(x) = e^x - 5\sin(x) = 0$

1. BISECTION

A BRACKETING (CLOSED) METHOD



Choose $[x_l, x_u]$, so that $f(x_l)*f(x_u) < 0$

Approximation: $x_r = (x_l + x_u) / 2$

Depending on the sign of $f(x_r)$, x_r becomes either the new lower or new upper limit for the next iteration

New approximation: $x_r = (x_l + x_u) / 2$

VRIJE UNIVERSITEIT BRUSSEL

HOW TO MAKE GOOD INITIAL GUESSES FOR X_U AND X_L?

- In the bracketing methods, we initially want to choose x_l and x_u, so that $f(x_l)*f(x_u) < 0$
- We can always **plot the function** that we have to solve, to make sure that the initial range $[x_l,x_u]$ that we chose bounds a root
- Let's see a Python script which is able to plot any function

1. BISECTION

PYTHON

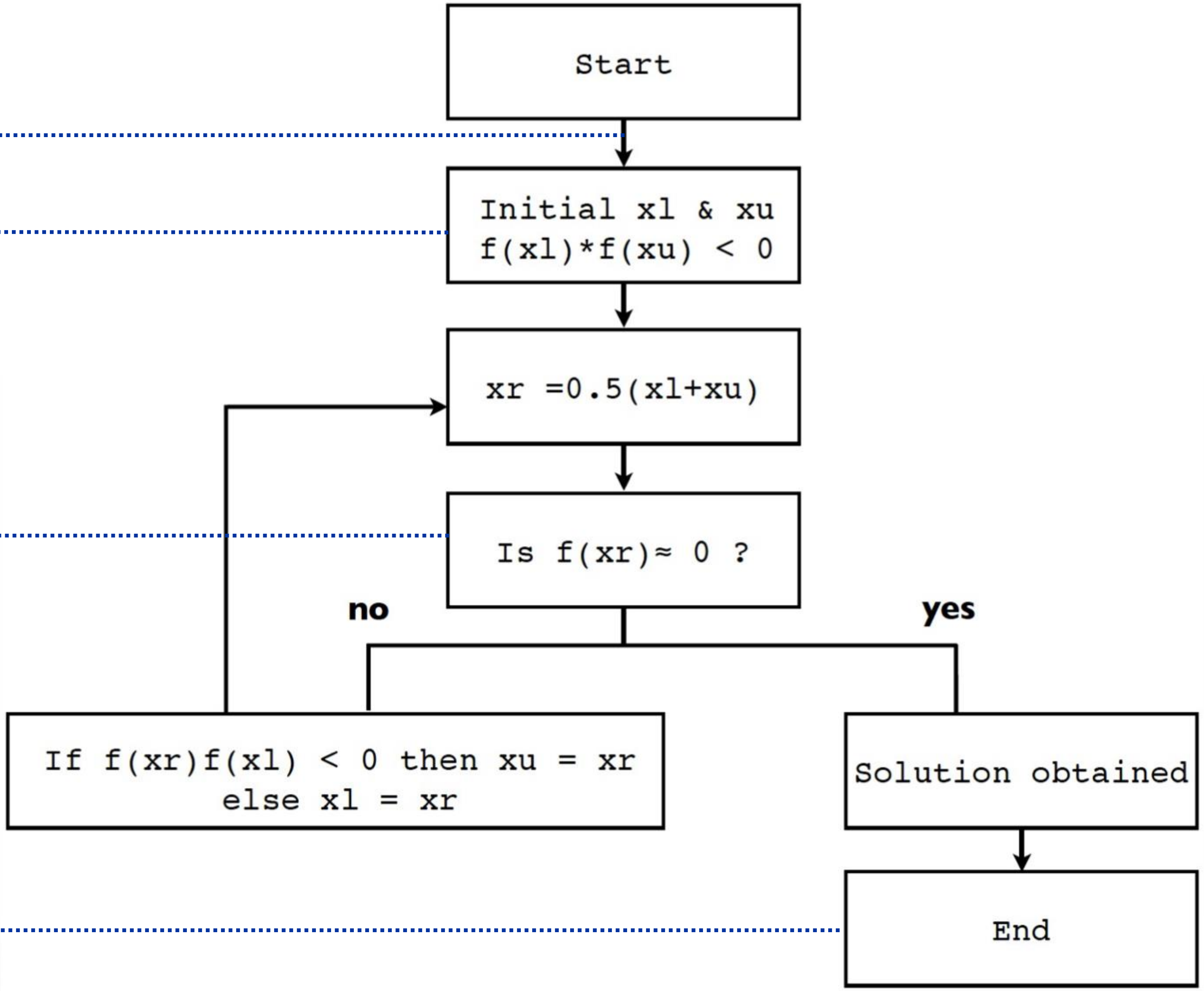
FLOW CHART

In **def main()** define function f to be solved

call **def bisection(…)**

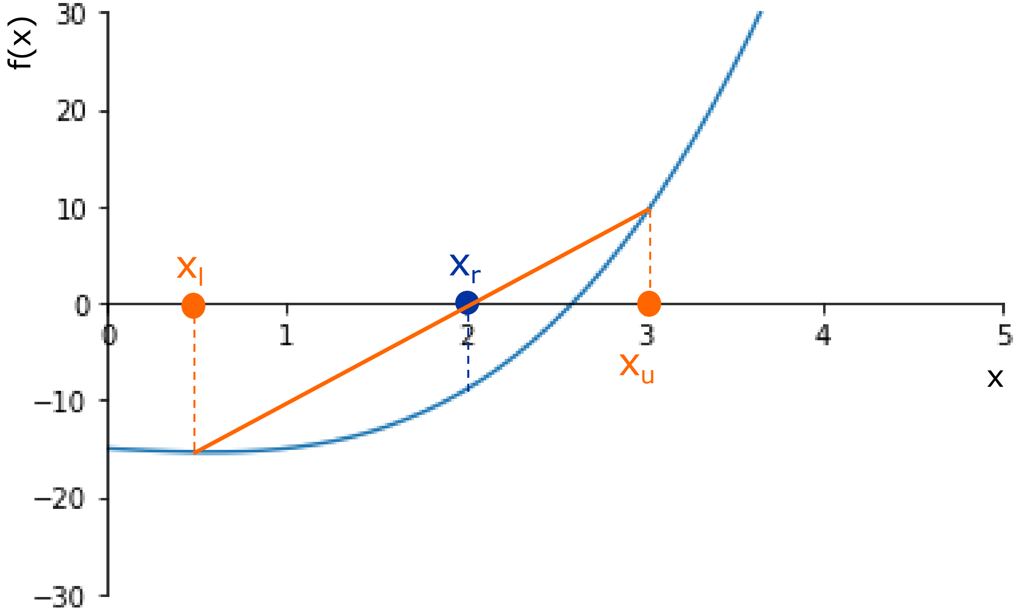
is |f(xr)| < chosen_tolerance ?

return solution to **def main()**



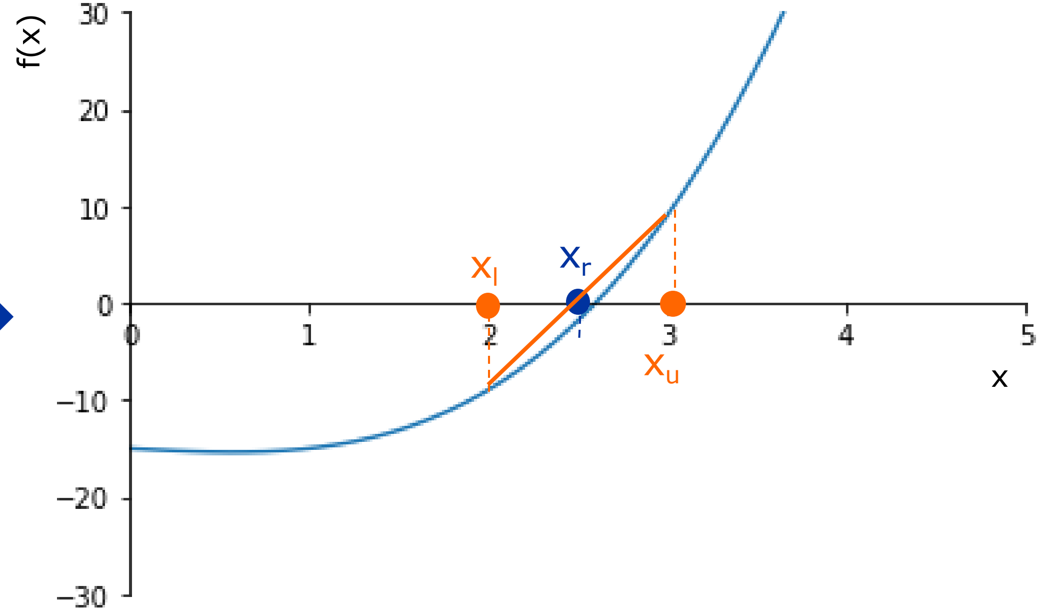
2. REGULA FALSI (FALSE POSITION)

ANOTHER BRACKETING (CLOSED) METHOD



Choose $[x_l,x_u]$, so that $f(x_l)*f(x_u) < 0$

Approximation: $x_r = x_u - \frac{f(x_u)(x_u - x_l)}{f(x_u) - f(x_l)}$



Depending on the sign of $f(x_r)$, x_r becomes either the new lower or new upper limit for the next iteration

New approximation: $x_r = x_u - \frac{f(x_u)(x_u - x_l)}{f(xu) - f(x_l)}$

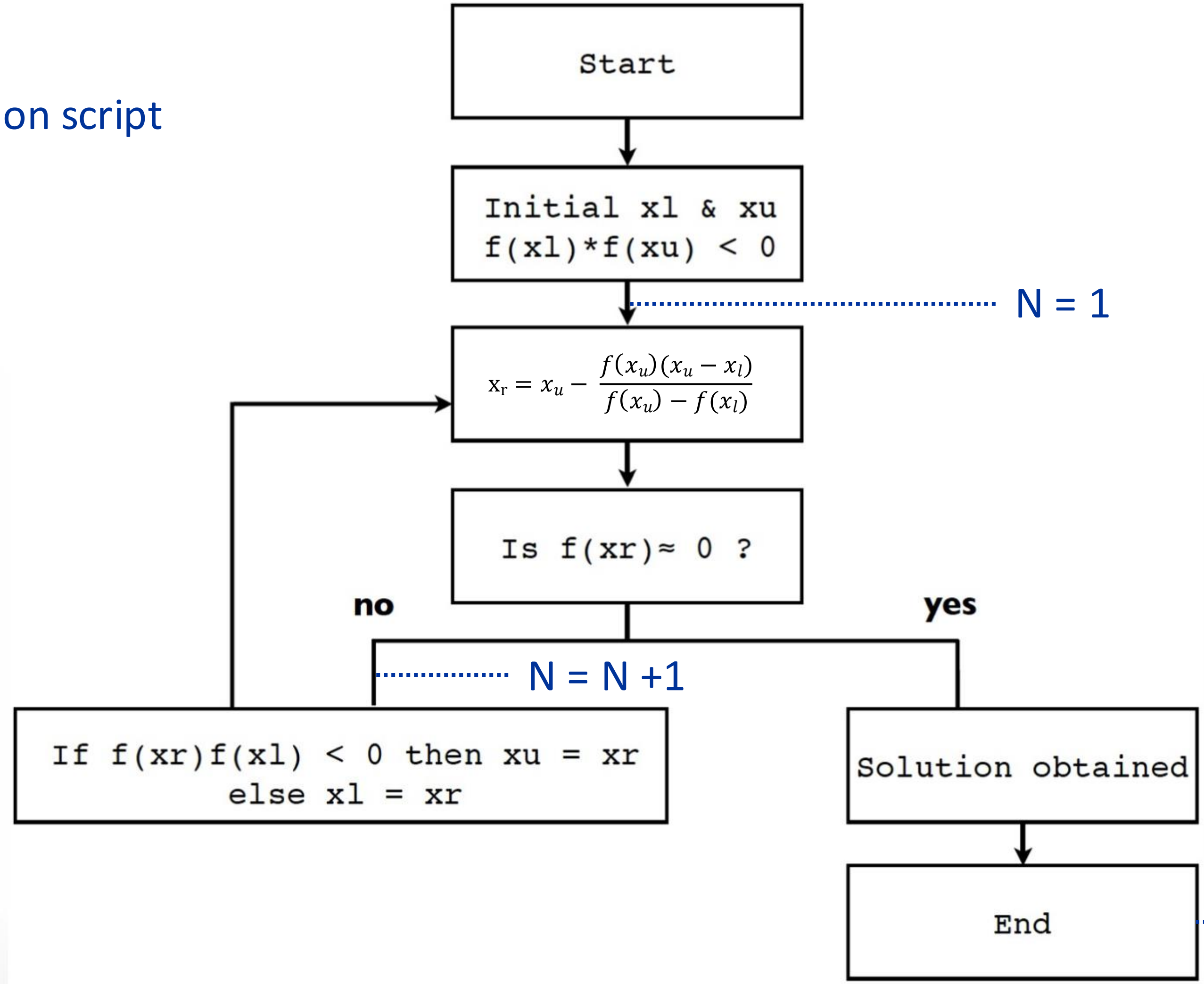
2. REGULA FALSI

PYTHON

- Very similar to the bisection script

FLOW CHART

How to keep track of how many iterations N it takes for the method to converge?



return solution and current N to def main()

3. NEWTON RAPHSON

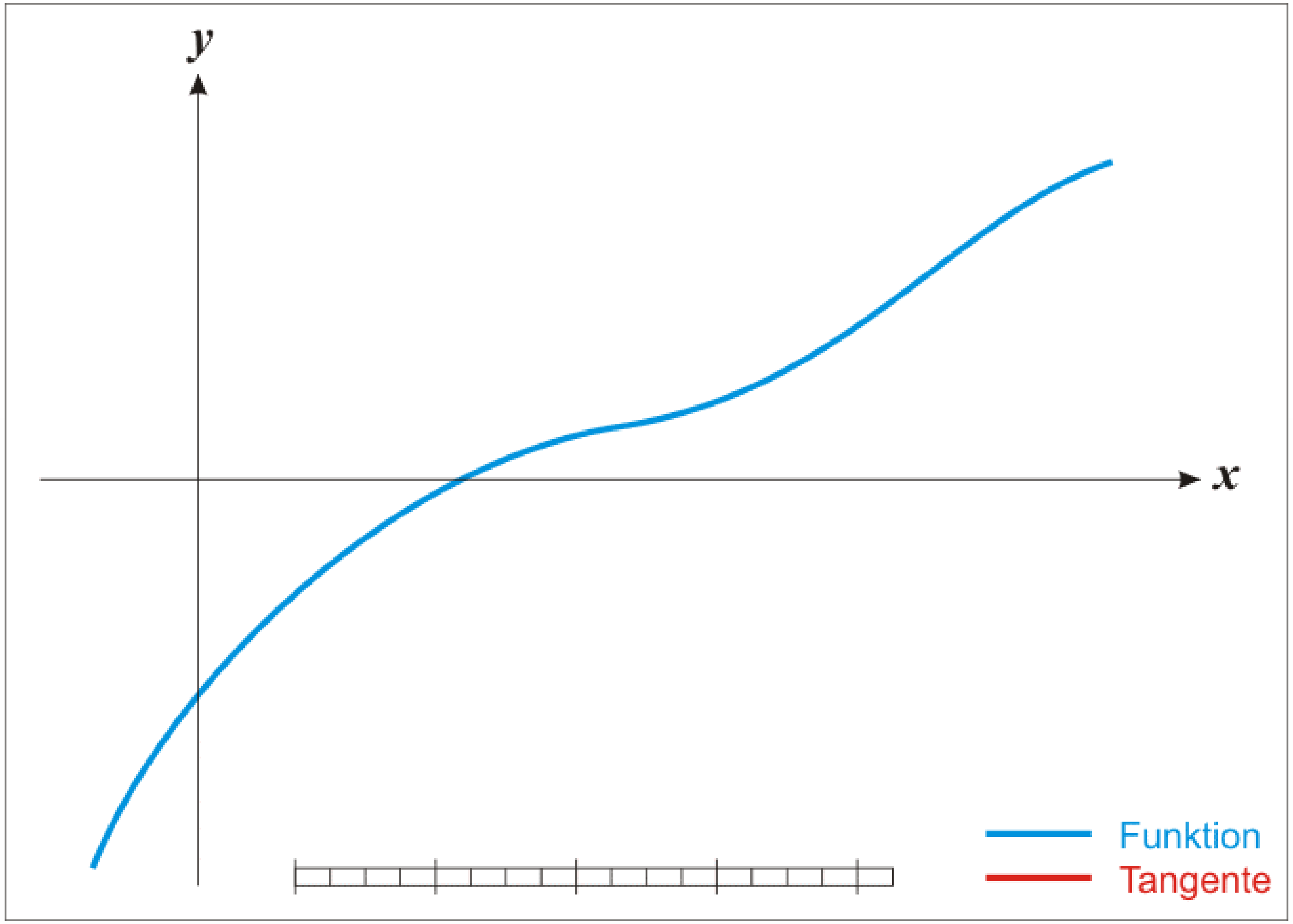
AN OPEN METHOD

- One initial guess needed (x_n), no interval
- Draw tangent line to the graph of f(x) at the point $x = x_n$
- Tangent line equation:

$$y = f'(x_n)(x - x_n) + f(x_n)$$

- The root of the tangent line gives us our new approximation:

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



source: https://nl.wikipedia.org/wiki/Methode_van_Newton-Raphson

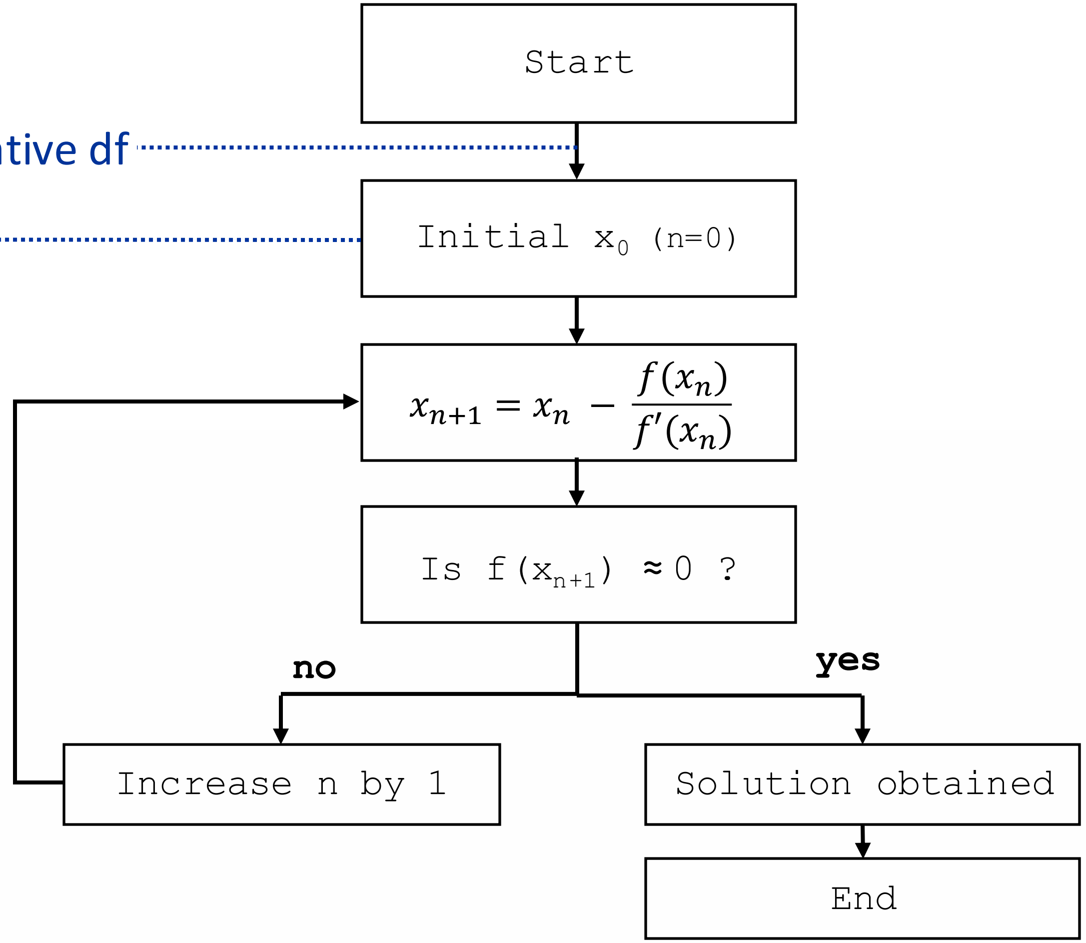
3. NEWTON RAPHSON

PYTHON

In **def main()** define function f to be solved and its derivative df

call **def newton(...)**

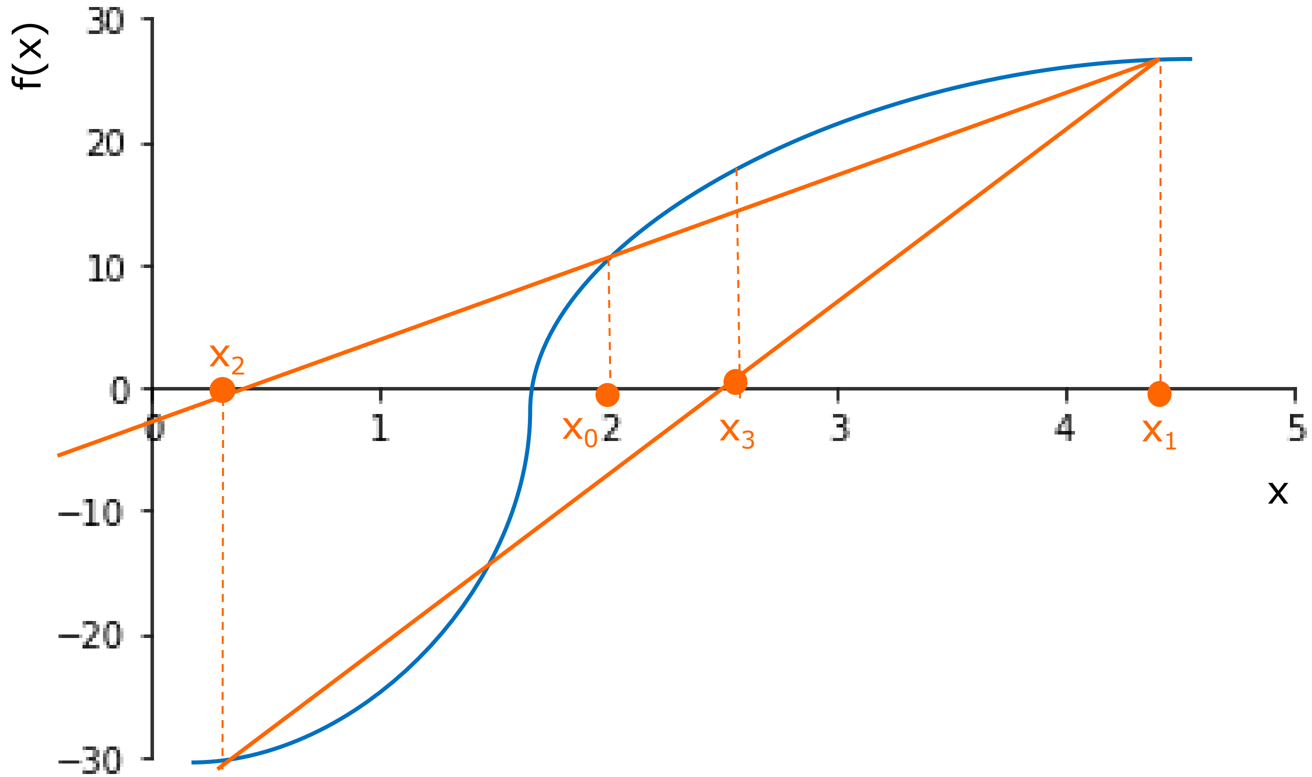
FLOW CHART



4. SECANT

ANOTHER OPEN METHOD

- Similar to regula falsi method but:
  1. The two initial values (x_0, x_1) do not need to bound a root
  2. Keeps the two most recent root approximations
  3. No need to check the sign of the function



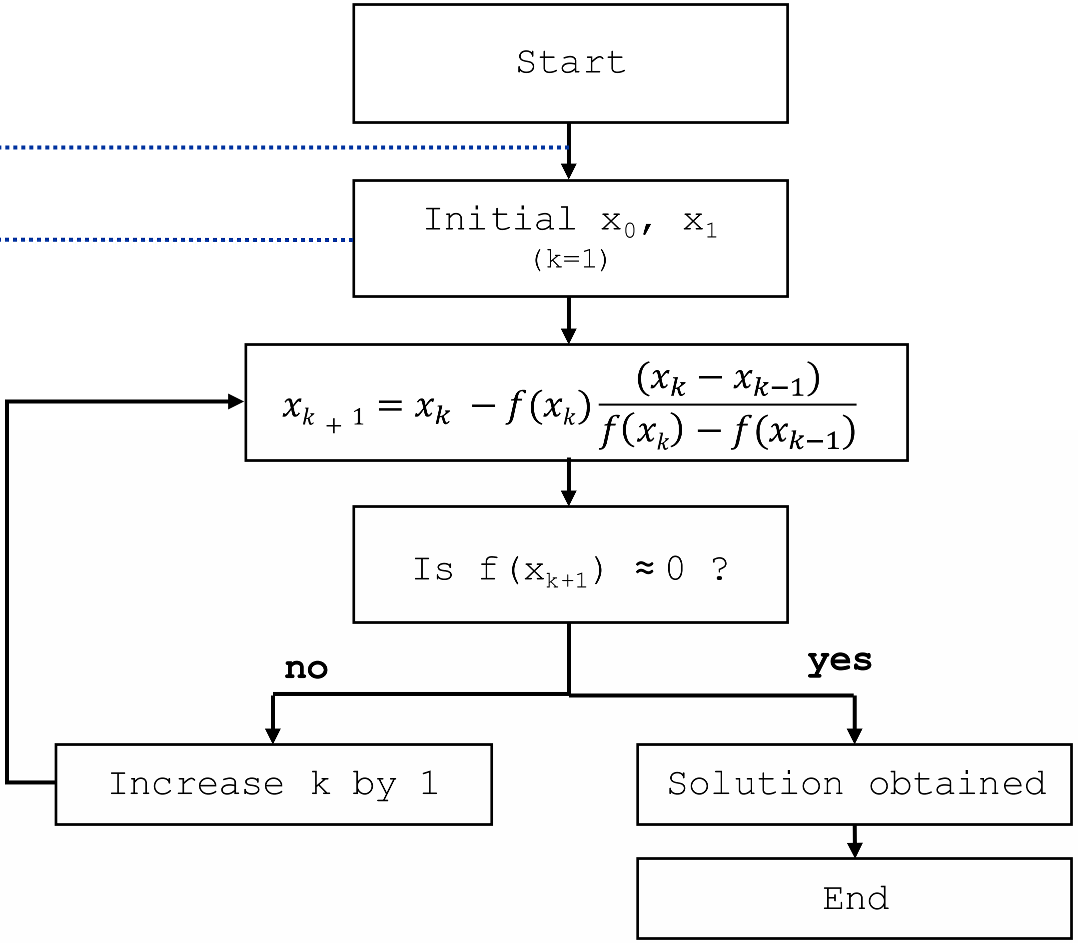
4. SECANT

PYTHON

In **def main()** define function f to be solved

call **def secant(…)**

FLOW CHART



EXTRA STOPPING CRITERIA

- Critical to avoid infinite loops.
  Stop the code when:

  - **Maximum iterations** have been reached

GENERAL FLOW CHART FOR ROOT-FINDING METHODS