032CM - 2026

PROGRAMMING FOR COMPUTATIONAL CHEMISTRY

Root Finding

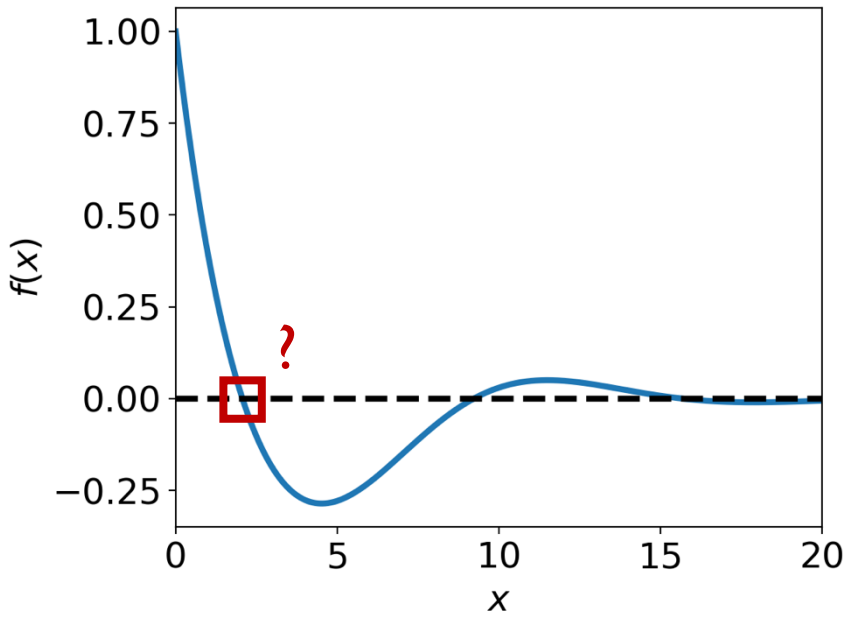
Gianluca Levi

gianluca.levi@units.it, giale@hi.is

Office: Building C11, 3rd floor, Room 329

Fall 2026

Finding approximate solutions (roots) to nonlinear equations

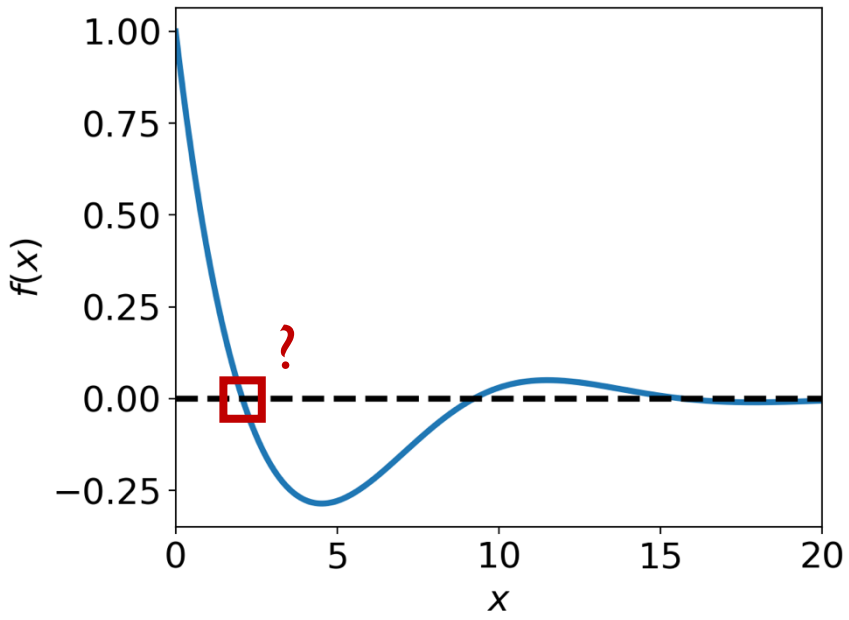


$$e^{-x/4}\cos\left(\frac{x}{2}\right) = e^{-x/2}\sin\left(\frac{x}{2}\right)$$

$$e^{-x/4}\cos\left(\frac{x}{2}\right) - e^{-x/2}\sin\left(\frac{x}{2}\right) = 0$$

$$f(x) = 0$$

Finding approximate solutions (roots) to nonlinear equations



$$e^{-x/4}\cos\left(\frac{x}{2}\right) = e^{-x/2}\sin\left(\frac{x}{2}\right)$$

$$e^{-x/4}\cos\left(\frac{x}{2}\right) - e^{-x/2}\sin\left(\frac{x}{2}\right) = 0$$

$$f(x) = 0$$

Minimization and maximization of a function (**function optimization**)

$$f'(x) = 0$$

- E.g. optimization of atomic structures, variational method (quantum chemistry)
- **Least-squares fitting** (e.g. fitting spectroscopic data, optimizing force fields for molecular modelling)

Solving **systems of nonlinear equations**

- E.g. chemical equilibrium calculations

Brute force root finding

1. Detect $f(x_i)\, f(x_{i+1}) < 0$
2. Assume linear variation of f

$$f(x) \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}\,(x - x_i) + f(x_i)$$

3. Approximate root as:

$$x = x_i - \frac{x_{i+1} - x_i}{f(x_{i+1}) - f(x_i)}\, f(x_i)$$

Brute force root finding

1. Detect $f(x_i)\, f(x_{i+1}) < 0$
2. Assume linear variation of f

$$f(x) \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}\,(x - x_i) + f(x_i)$$

3. Approximate root as:

$$x = x_i - \frac{x_{i+1} - x_i}{f(x_{i+1}) - f(x_i)}\, f(x_i)$$

Inefficient and slow

- Cost scales with density of points
- Evaluates many points far from any root

Bisection method

1. Identify interval where the function changes sign
2. Evaluate the midpoint of the current interval
3. Decide which half still contains a sign change and update interval
4. Repeat until the error is below a desired tolerance

Solution guaranteed to be found (if it exist in the interval)

Convergence is steady, but can we do better?

Newton-Raphson method

1. Choose an initial guess
2. Approximate function at the current guess by the line tangent to it

$$f(x) \approx f(x_n) + f'(x_n)\,(x - x_n)$$

3. Approximate the root as the intercept of the tangent line with the x-axis

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n = 0, 1, 2, \ldots$$

4. Repeat until the error is below a desired tolerance

https://en.wikipedia.org/wiki/File:NewtonIteration_Ani.gif

Very fast

Does not guarantee that a solution is found

Convergence **dependent on the initial guess**

Rate of convergence

How fast the error decreases from one iteration to the next

$$e_{n+1} = C\, e_n^q$$

q: **Order** (or **rate**) of convergence

- Can be computed from a log-log plot of the error

$$\log e_{n+1} = \log C + q \log e_n$$

Assignment 5

Problem 1

The wave function of the hydrogen-like $3p_z$ atomic orbital in spherical polar coordinates is given by:

$$\psi_{3p_z}(r,\theta,\phi) = R_{31}(r)Y_{10}(\theta,\phi)$$

where $R_{31}(r)$ is the radial part:

$$R_{31}(r) = \frac{1}{27}\left(\frac{2Z}{3a_0}\right)^{3/2} \rho\,(6-\rho)\,e^{-\rho/3}, \qquad \rho = \frac{Zr}{a_0},$$

and $Y_{10}(\theta,\phi)$ is the angular part (spherical harmonic):

$$Y_{10}(\theta,\phi) = \sqrt{\frac{3}{4\pi}}\cos\theta.$$

Here, Z is the atomic number and a_0 is the Bohr radius. Throughout this problem, consider the hydrogen atom ($Z = 1$).

(a) Write a Python function that computes the ψ_{3p_z} wave function for the hydrogen atom for given values of the radial coordinate r and polar angle θ. Use atomic units, where the Bohr radius $a_0 = 1$.

(b) Plot the radial wave function $R_{31}(r)$ for $r \in [0, 25]$ Bohr, and create a 2D contour plot of ψ_{3p_z} in the xz-plane for $x, z \in [-15, 15]$ Bohr. How many radial nodes can you identify? Compare with the number of radial nodes obtained using the formula $n - l - 1$, where n and l are the principal and angular momentum quantum numbers, respectively. *Note*: the point at $r = 0$ is not considered a node.

Hint: For the 2D plot, convert Cartesian to spherical coordinates using

$$r = \sqrt{x^2 + y^2 + z^2}$$
$$\theta = \arccos\left(\frac{z}{r}\right).$$

(c) Find the approximate location of the radial node between $r = 2.5$ and 7.5 Bohr using the bisection and the Newton-Raphson methods. Use $[2.5, 7.5]$ Bohr as the initial interval for the bisection method and $r_0 = 2.5$ Bohr as the initial guess for the Newton method.

(d) For each method, save the error at each iteration. For the bisection method, use the length of the interval as the error. For the Newton-Raphson method, use the absolute value of the radial wave function evaluated at the current approximation, $\epsilon_i = |R_{31}(r_i)|$. Then, plot the error ϵ_i versus the iteration number using a logarithmic scale for the y-axis and create a log-log plot of ϵ_{i+1} versus ϵ_i. Exclude zero error values from the logarithmic plots, if necessary. Roughly estimate the slope for each method from the log-log plot. The slope is the *order of convergence*, which measures the rate of convergence of the algorithm. Comment on the differences between the two methods.

Hint: To set the log scale in Matplotlib use `plt.xscale('log')/plt.yscale('log')`.

(e) Find the approximate location of the maximum of the radial wave function $R_{31}(r)$ using the Newton-Raphson method with an initial guess of $r_0 = 1$ Bohr. What happens if you use $r_0 = 7$ Bohr? Comment.