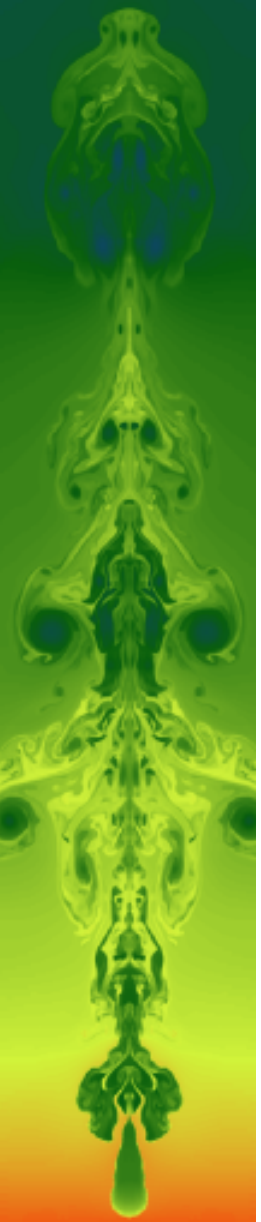


Finding roots (Bracketing methods)

Numerical methods for solving nonlinear equations

Today's topics

- Statement of the problem and the basic numerical methods for finding roots
- Closed (bracketing) methods
  - Bisection method
  - False-position method
- Termination criteria (when to stop)
- Project 3A: Root finding (Bracketing methods)

Mathematical introduction

- There are many situations where we need to solve an equation for the value(s) of one or more independent variables.
  - Examples: Find value(s) of x that satisfy

$$ax^2 + bx + c = 0 \quad \text{or} \quad e^{-x} = x$$

- But there is often no analytical solution!
- Can always write as $f(x) = 0$: $f(x) = e^{-x} - x$
- Consider a skydiver of mass m, subject to air resistance ("linear drag") with drag coefficient C_d:
  - Velocity v at time t given by $v = \frac{gm}{C_d}\left(1 - e^{-(C_d/m)t}\right)$
- But suppose we want to determine C_d :
  - Numerical solution of $f(C_d)=0$ is required where

$$f(C_d) = \frac{gm}{C_d}\left(1 - e^{-(C_d/m)t}\right) - v$$

General concepts

- Two classes of problem:
  - Finding all real & complex roots of polynomials
  - Finding real roots of algebraic or transcendental equations
- Numerical root-finding methods generally proceed in two steps:
  - Make a first guess (or guesses).
  - Refine guess until a solution is found (iteration).
- **Closed (bracketing) methods (Today & this coming week)**
  - Initial guess for upper and lower bounds must bracket root.
  - Successively reduce width of bracket
- **Open methods (The following week!)**
  - Do not require initial guesses to bracket solution
  - Path to solution based on slope, $f'(x)$ of function

Closed (bracketing) methods

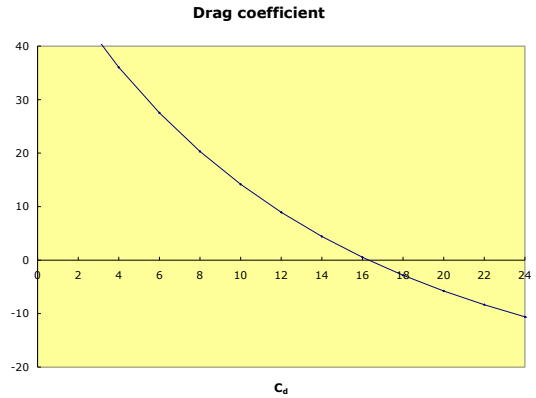
- Exploits the fact that a function usually changes sign in the vicinity of the root
- Recipe:
  - Guess upper and lower bounds x_u & x_l that bracket root
    - Plot $f(x)$ to find good first guesses.
  - Systematically reduce width of bracket to find root

- Example: Skydiver

$$f(C_d) = \frac{gm}{C_d}\left(1 - e^{-(C_d/m)t}\right) - v$$

  - Parameters
    - $g = 9.81$ m/s
    - $m = 68.1$ kg
    - $t = 10$ s
    - $v = 40$ m/s

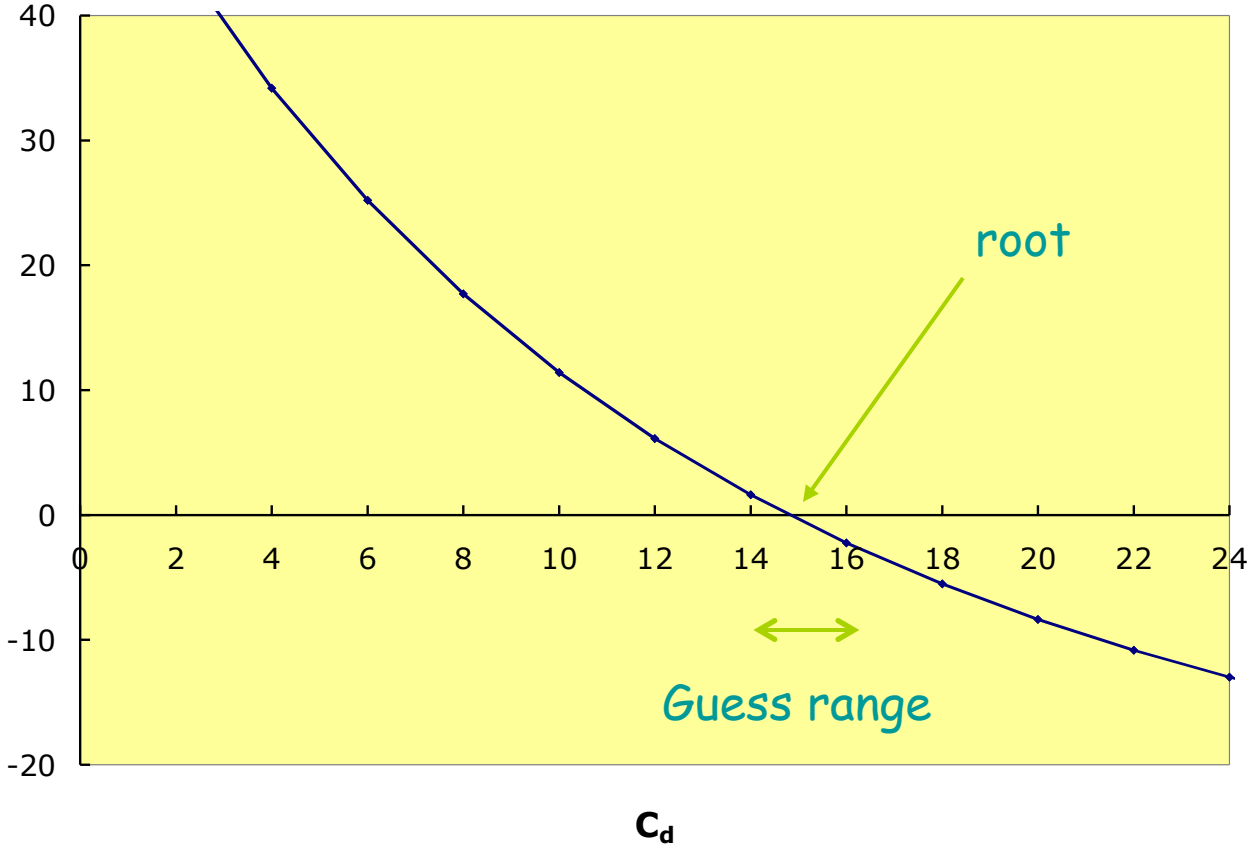
- Plot $f(C_d)$ vs C_d:



- $f(C_d) = 0$ when $C_d \approx 15$ kg/s
  - Set $x_u = 16$; $x_l = 14$

Graphical root estimation

$$f(C_d) = \frac{gm}{C_d}\left(1 - e^{-(C_d/m)t}\right) - v$$

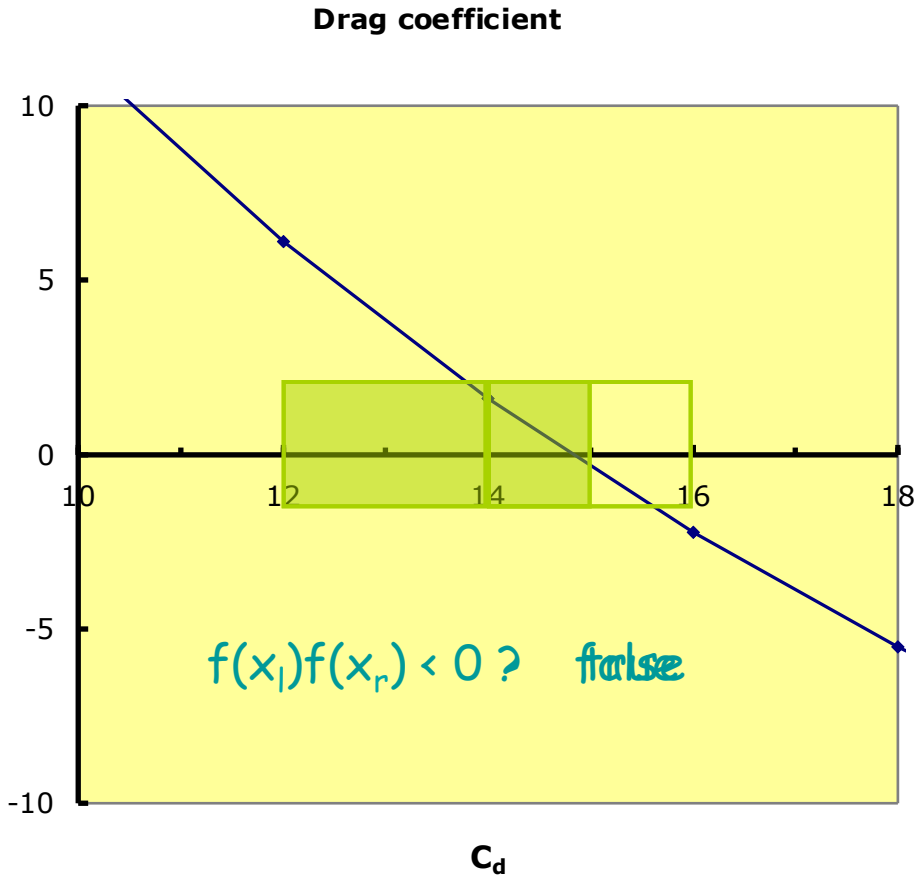


Bisection method algorithm

1. Supply first guesses for x_l & x_U
   (must bracket root: $f(x_l)f(x_u) \le 0$)
2. Bisect to guess root:
   $x_r = (x_l + x_U)/2$
3. If $f(x_l)f(x_r) < 0$
   % Guess too high
   - Set $x_U = x_r$
   - Back to step 2.

   Else if $f(x_l)f(x_r) > 0$
   % Guess too low
   - Set $x_l = x_r$
   - Back to step 2.

   Else if $f(x_l)f(x_r) = 0$
   - Root found
   - End



Termination – when to stop

- Keep track of the relative error
  - Relative difference between previous & current estimates x_r^{old} and x_r^{new}

$$\varepsilon_a = \left| \frac{x_r^{new} - x_r^{old}}{x_r^{new}} \right|$$

  - This is always larger than the true 'error'
- Stop calculation when ε_a is less than a pre-determined target value, ε_s

$$\varepsilon_a < \varepsilon_s$$

- Note also that the number of iterations required to reach a given **absolute** error can be predicted in advance
  - Each succeeding iteration halves error

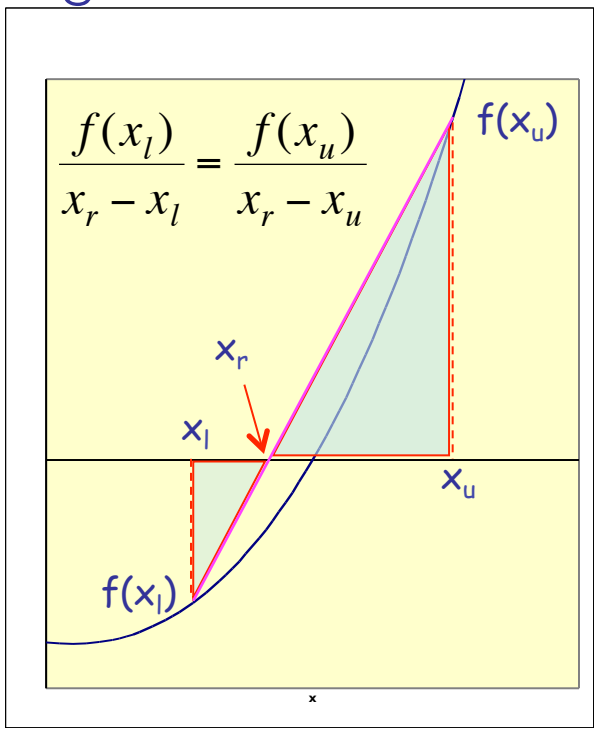
Error at start $E_a^0 = x_u^0 - x_l^0 = \Delta x^0$

Error after *n* iterations $E_a^n = \frac{\Delta x^0}{2^n}$

Solve for *n* $n = \frac{\log\left(\Delta x^0 / E_a^n\right)}{\log 2}$

False position method

- Uses same basic bisection idea but is often more efficient
- Attempts to get closer to the root at each iteration by making a linear interpolation between current guesses x_U & x_l
  - This takes into account relative magnitudes of $f(x_U)$ and $f(x_l)$.
- Intersection of straight line joining x_l and x_U with x-axis gives estimate of root
- x_r found using similar triangles



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

Pros 'n' cons

- Bisection vs. False position
  - False position
    - *Takes fewer iterations in most situations*
    - *Can get 'stuck' (converge slowly) if function has large curvature*
  - Bisection
    - *Usually takes more iterations*
    - *Not affected by function curvature*
- Speed of convergence especially matters if…
  - Evaluating *f*(x) is computationally-intensive

    and/or
  - You have to do the root finding many times

Project 3A

- Write a Python program to find single roots of a mathematical function *f*(x) using the bisection method.
- Part 1
  - Use the function to find real, positive roots in range $0 < x < 5$ for five different functions.
  - Return value of root and number of iterations for the bisection method.
- Part 2
  - Wien's Displacement constant: Use your bisection method program to determine the value of Wien's Displacement constant and hence the surface temperature of the Sun.