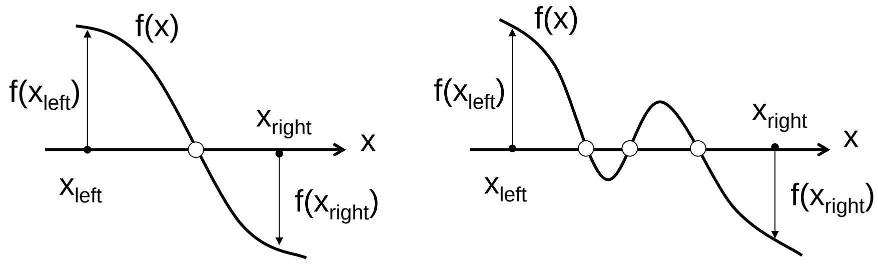
二分勘根法

Bisection Root Finding

丁培毅

Finding the Roots of f(x)

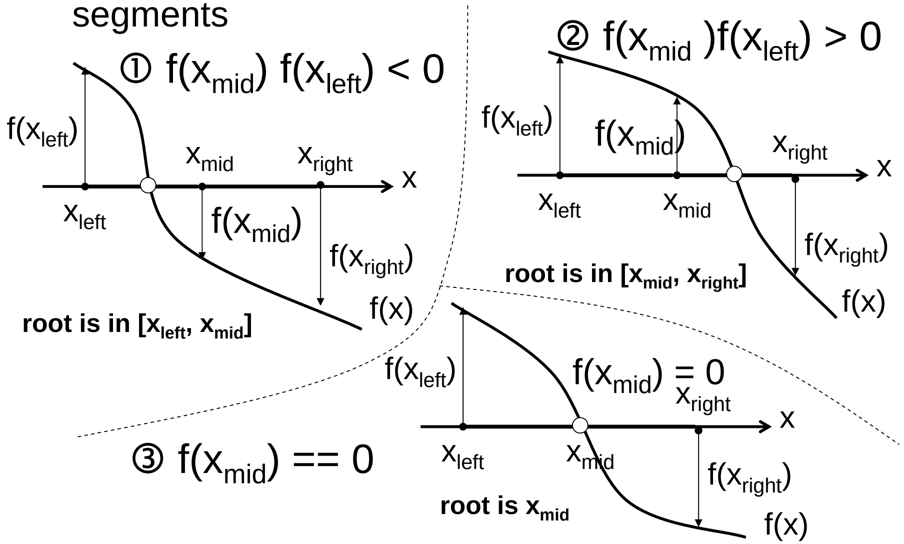


(a) One root **(b) Three roots**

- Change of sign implies an odd number of roots in the segment $[x_{left}, x_{right}]$
- Assume there is only one root in this region, …

Three Possibilities

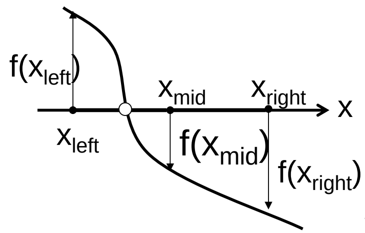
- When the interval $[x_{left}, x_{right}]$ is divided as two equal segments



Use a **while** Loop to divide the interval by 2 each time

1. Given x_{left} and x_{right} , $x_{mid} = (x_{left}+x_{right}) / 2$
2. a. Calculate $f(x_{mid})$
   b. if $(f(x_{left})f(x_{mid}) < 0)$ $x_{right} = x_{mid}$
   c. else if $(f(x_{left})f(x_{mid}) > 0)$ $x_{left} = x_{mid}$
   d. else if $(f(x_{mid}) == 0)$ root is x_{mid}, break

Repeat the above two steps



x_right-x_left > 1.0e-10

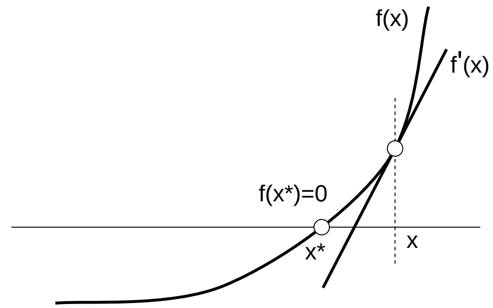
```
while (1)
{
    x_mid = (x_left + x_right) / 2.0;
    if (fabs(f(x_mid)) < 1.0e-10)
        break;
    else if (f(x_left) * f(x_mid) < 0.0)
        x_right = x_mid;
    else // if (f(x_right) * f(x_mid) < 0.0)
        x_left = x_mid;
}
```

Eliminating Redundant Evaluations

- function f() on each point x_{mid} is called 3 times in one iteration, and is called once as x_{left} or x_{right} in the following iteration
  - Use variables to save the function values calculated previously
- $log_2(n)$ evaluations out of $n=(x_1-x_0)/\varepsilon$ segments
  - $(x_1-x_0)/2^k \approx \varepsilon$
  - i.e. $k \approx log_2(n)$

```
f_left = f(x_left);
f_right = f(x_right);
while (x_right-x_left > 1.0e-10) {       // ← ε
    x_mid = (x_left + x_right) / 2.0;
    f_mid = f(x_mid);
    if (fabs(f_mid) < 1.0e-10)
        break;
    else if (f_left * f_mid < 0.0) {
        x_right = x_mid;
        f_right = f_mid;
    }
    else if (f_right * f_mid < 0.0) {
        x_left = x_mid;
        f_left = f_mid;
    }
}
```


Other Related Applications



- Newton's method for finding minima (or root)
- Binary Search: find the specified value from a sorted array of integers

Other Applications (cont'd)

- Find the Duplicate Number (Leetcode 287)
  - Given an array nums[ ] containing n+1 integers where each integer is between 1 and n (inclusive), Pidgin hole principle assures that at least one duplicate number must exist. Assume that there is only one duplicate number, find it. Note: You must not modify the array. You must use only constant, O(1) extra space. Your runtime complexity should be less than $O(n^2)$.
- Find Minimum in Rotated Sorted Array (Leetcode 153)
  - Suppose a sorted array is rotated by you beforehand. (i.e., 0 1 2 4 5 6 7 might become 4 5 6 7 0 1 2). Find the minimum element. You may assume no duplicate exists in the array. Computation $O(log_2 n)$ is demanded.