UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical &
Computer Engineering

ECE 204 *Numerical methods*

The bisection method

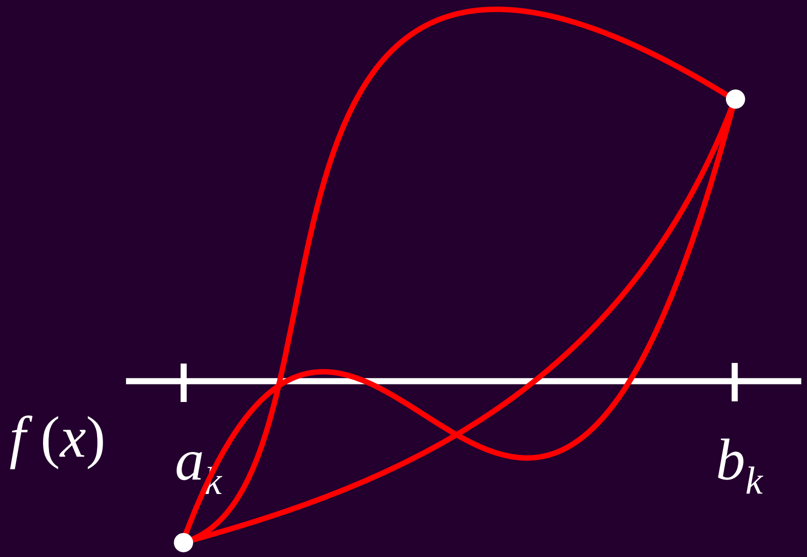
Douglas Wilhelm Harder, LEL, M.Math.
dwharder@uwaterloo.ca
dwharder@gmail.com

Introduction

- In this topic, we will
  - Describe the idea behind the bisection method
  - Determine how fast it converges
    - We will compare it to Newton's method
  - Look at an example
  - Compare the algorithm with Newton's method
  - Look at an implementation

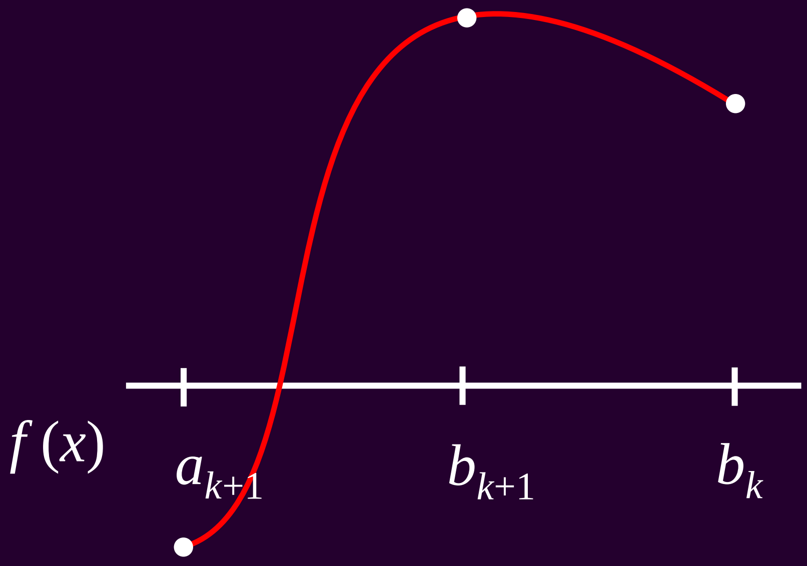
Bracketing a root

- Suppose we have a real-valued function of real variable and suppose that a_k and b_k are two points such that $f(a_k)$ and $f(b_k)$ have opposite signs
  - If f is continuous, the f must have a root on $[a_k, b_k]$



Bracketing a root

- Suppose we have a real-valued function of real variable and suppose that a_k and b_k are two points such that $f(a_k)$ and $f(b_k)$ have opposite signs
  - If f is continuous, the f must have a root on $[a_k, b_k]$



Calculating the mid-point

- Thus, given two bounds a_k and b_k such that $f(a_k)$ and $f(b_k)$ have opposite signs
  - Find the mid-point
    $$m_k \leftarrow \frac{a_k + b_k}{2}$$
  - If $f(m_k) = 0$, we have found a root, so we are done
  - If $f(a_k)$ and $f(m_k)$ have opposite signs,
    let $a_{k+1} \leftarrow a_k$ and $b_{k+1} \leftarrow m_k$
  - Otherwise, $f(m_k)$ and $f(b_k)$ have opposite signs,
    let $a_{k+1} \leftarrow m_k$ and $b_{k+1} \leftarrow b_k$

Error analysis

- Approximate the root by choosing whichever end-point has the smallest absolute value
  - What is the maximum error?
  - The root could be arbitrarily close to the other end-point, so the maximum error is $h = b_k - a_k$
  - With each step, the maximum error is halved, or $\frac{1}{2}h$
    - Thus, if h is the error, this technique is $O(h)$

Example

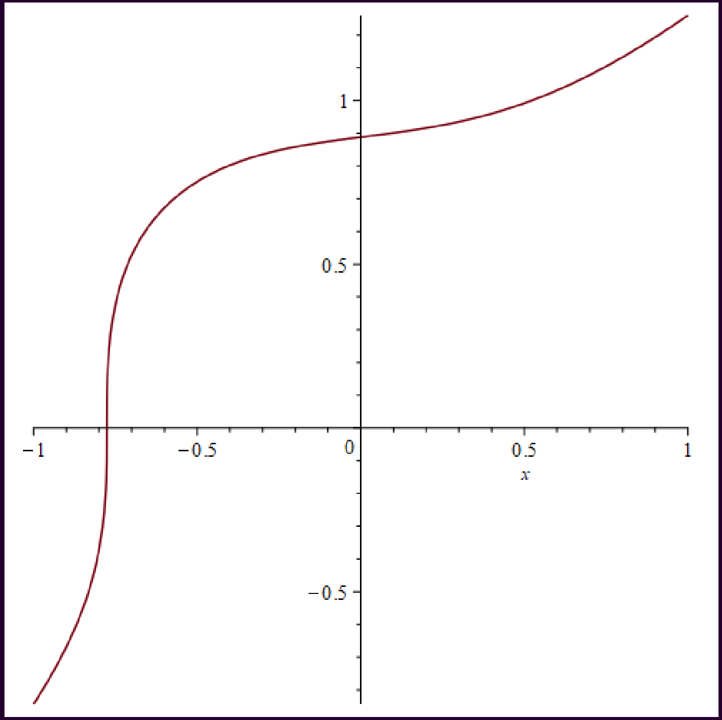
- Find the first root of $2e^{-2x} - e^{-x}$ starting with [0, 1]
  - The solution is ln(2)

k	a_k	b_k	$f(a_k)$	$f(b_k)$	Maximum error	m_k	$f(m_k)$
0	0	1	1.0	–0.09721	1	0.5	0.1292
1	0.5	1	0.1292	–0.09721	0.5	0.75	–0.02611
2	0.5	0.75	0.1292	–0.02611	0.25	0.625	0.03775
3	0.625	0.75	0.03775	–0.02611	0.125	0.6875	0.002848
4	0.6875	0.75	0.00285	–0.02611	0.0625	0.71875	–0.01232
5	0.6875	0.71875	0.00285	–0.01232	0.03125	0.703125	–0.004915
6	0.6875	0.703125	0.00285	–0.004915	0.015625	0.6953125	–0.001079
7	0.6875	0.6953125	0.00285	–0.001079	0.0078125	0.69140625	0.0008727
8	0.69140625	0.6953125	0.0008727	–0.001079	0.00390625	0.693359375	–0.0001061
9	0.69140625	0.693359375	0.0008727	–0.0001061	0.001953125	0.6923828125	0.0003826
10	0.6923828125	0.693359375	0.0003826	–0.0001061	0.0009765625		

Comparison with Newton's method

- The bisection method converges very slowly
  - However, if there is a root and if *f* is continuous on $[a_0, b_0]$, it is very likely to converge
  - It may not converge if the slope at the root is close to infinity
    - For example,

$$\sqrt[3]{x^3 + 0.3x + 0.7}$$



Implementation

```
double bisection( std::function<double( double x )> f,
                  double a, double b,
                  double eps_step, double eps_abs,
                  unsigned int max_iterations ) {
    assert( a < b );

    double fa{ f( a ) };
    double fb{ f( b ) };

    if ( !std::isfinite( fa ) || !std::isfinite( fb ) ) {
        return NAN;
    }

    if ( fa == 0.0 ) {
        return a;
    }

    if ( fb == 0.0 ) {
        return b;
    }
```

C/C++ Code is provided to demonstrate the straight-forward nature of these algorithms and not required for the examination

Implementation

```
for ( unsigned int k{0}; k < max_iterations; ++k ) {
    double m{ a + (b - a)/2.0 };    // This avoids overflow
    double fm{ f( m ) };

    if ( !std::isfinite( fm ) ) {
        return NAN;
    }

    if ( fm == 0.0 ) {
        return m;
    } else if ( std::signbit( fa ) == std::signbit( fm ) ) {
         a =  m;
        fa = fm;
    } else {
         b =  m;
        fb = fm;
    }
```

C/C++ Code is provided to demonstrate the straight-forward nature of these algorithms and not required for the examination

Implementation

```
            if ( (b - a) < eps_step ) {
                if ( std::abs( fa ) <= std::abs( fb ) ) {
                    if ( std::abs( fa ) < eps_abs ) {
                        return a;
                    }
                } else {
                    if ( std::abs( fb ) < eps_abs ) {
                        return b;
                    }
                }
            }
        }

        return NAN;
    }
```

$b_{k+1} - a_{k+1} < \varepsilon_{\text{step}}$ and either

$$\left|f\left(a_{k+1}\right)\right| < \varepsilon_{\text{abs}} \text{ or } \left|f\left(b_{k+1}\right)\right| < \varepsilon_{\text{abs}},$$

returning whichever is smaller in absolute value

C/C++ Code is provided to demonstrate the straight-forward nature of these algorithms and not required for the examination

Summary

- Following this topic, you now
  - Understand the idea behind the bisection method
  - Know that it converges according to $\mathrm{O}(h)$
  - Have seen an example
  - Understand that it will generally converge under normal conditions
  - Considered an implementation

References

[1] https://en.wikipedia.org/wiki/Bisection_method

Acknowledgments

None so far.

Colophon

These slides were prepared using the Cambria typeface. Mathematical equations use Times New Roman, and source code is presented using `Consolas`. Mathematical equations are prepared in MathType by Design Science, Inc.

Examples may be formulated and checked using Maple by Maplesoft, Inc.

The photographs of flowers and a monarch butter appearing on the title slide and accenting the top of each other slide were taken at the Royal Botanical Gardens in October of 2017 by Douglas Wilhelm Harder. Please see

https://www.rbg.ca/

for more information.

Disclaimer