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

ECE 204 *Numerical methods*

Golden-ratio search for finding extrema

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

Introduction

- In this topic, we will
  - Review the idea of using bisection for finding a minimum
  - Describe the idea of trisection
  - Consider increasing efficiency by halving the number of function evaluations
  - Introduce the golden ratio and the reciprocal thereof
  - Look at an implementation
  - Give an example

Bracketed methods

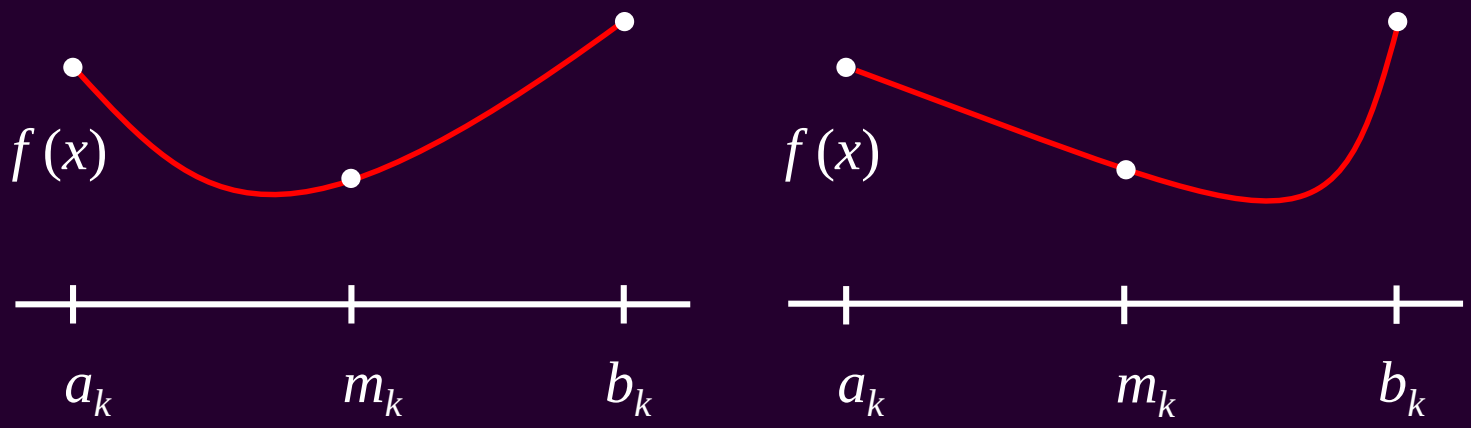
- Suppose you are aware there is a unique local minimum on an interval $[a_0, b_0]$
  - We will develop a bracketed technique that attempts to reduce the width of the interval

Midpoint

- Suppose we know there is a minimum on the interval $[a_k, b_k]$
  - Suppose we mimic the bisection method and calculate the mid-point:

$$m_k \leftarrow \frac{a_k + b_k}{2}$$

  - Does $f(m_k)$ allow us to reduce the width of the interval?

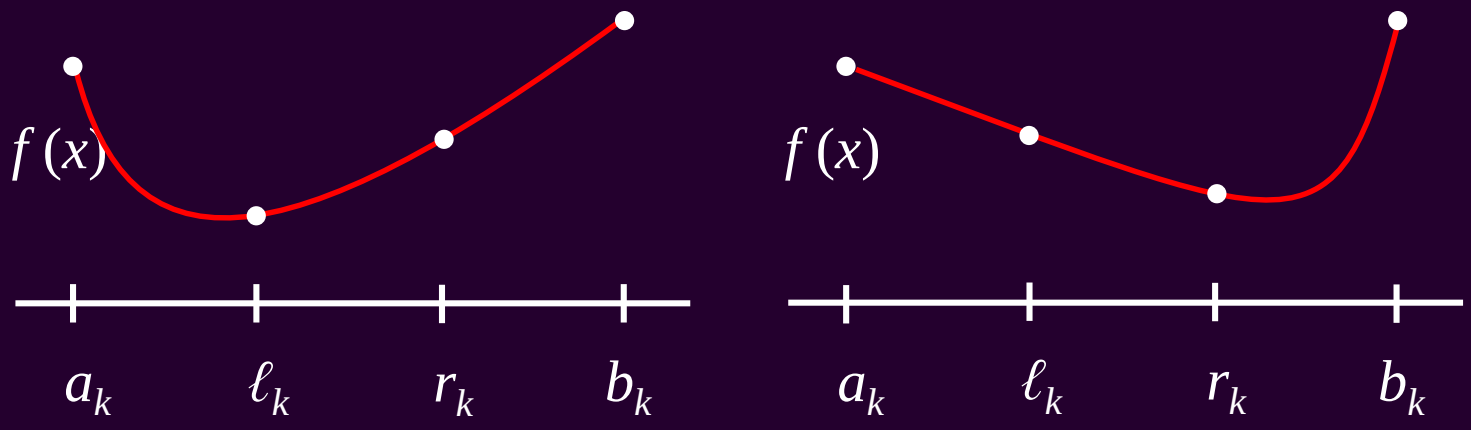


Trisection

- Suppose, instead, we trisect the interval:

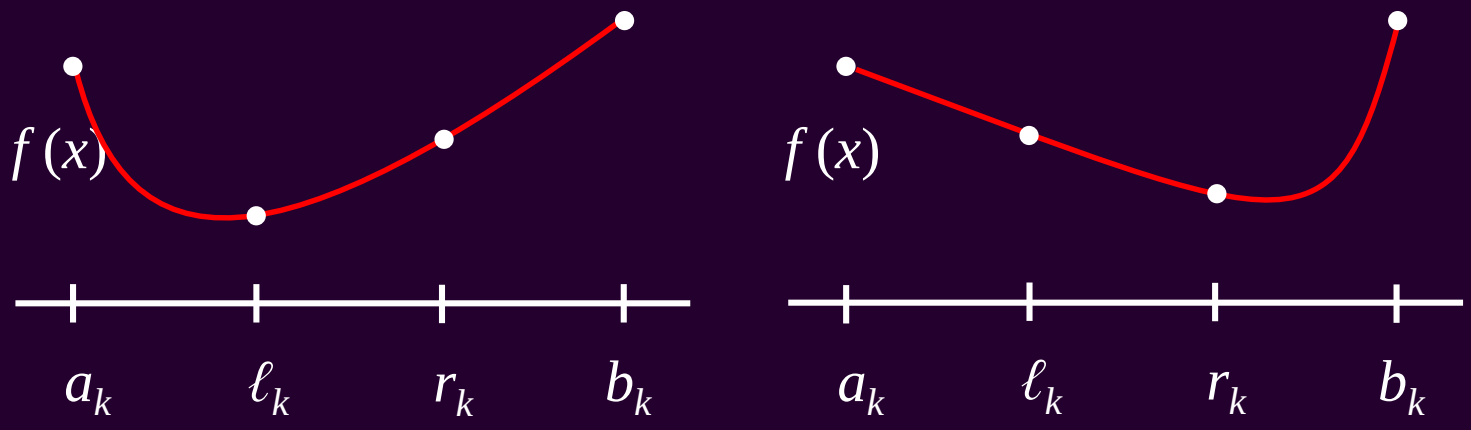
$$\ell_k \leftarrow \frac{2a_k + b_k}{3} = b_k - \frac{2}{3}(b_k - a_k)$$

$$r_k \leftarrow \frac{a_k + 2b_k}{3} = a_k + \frac{2}{3}(b_k - a_k)$$



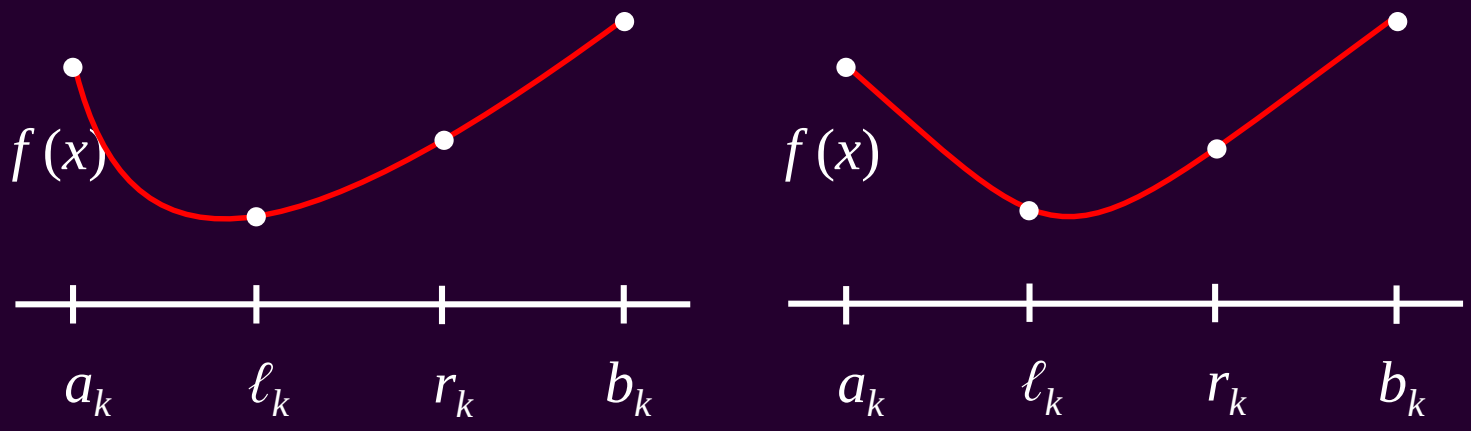
Trisection

- Consequently,
  - If $f(\ell_k) < f(r_k)$, we can restrict the minimum to $[a_k, r_k]$
    - Thus, set $a_{k+1} \leftarrow a_k$ and $b_{k+1} \leftarrow r_k$
  - If $f(\ell_k) > f(r_k)$, we can restrict the minimum to $[\ell_k, b_k]$
    - Thus, set $a_{k+1} \leftarrow \ell_k$ and $b_{k+1} \leftarrow b_k$



Trisection

- Can we do better?
  - In both these images, our next interval is $[a_k, r_k]$



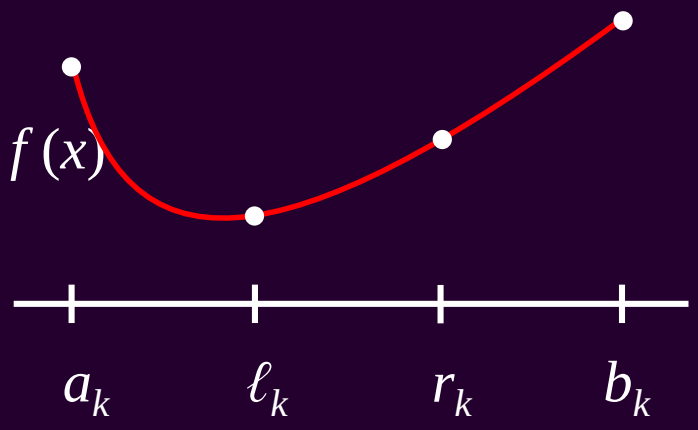
Trisection

- Thus, with each step,
  - We reduce the width of the interval by a factor of 2/3
  - We require two function evaluations

- Function evaluations are likely the most expensive operation
  - Can we reduce the work required?

Trisection

- By trisecting the interval,
  - The point ℓ_k is dead center of the interval $[a_k, r_k]$

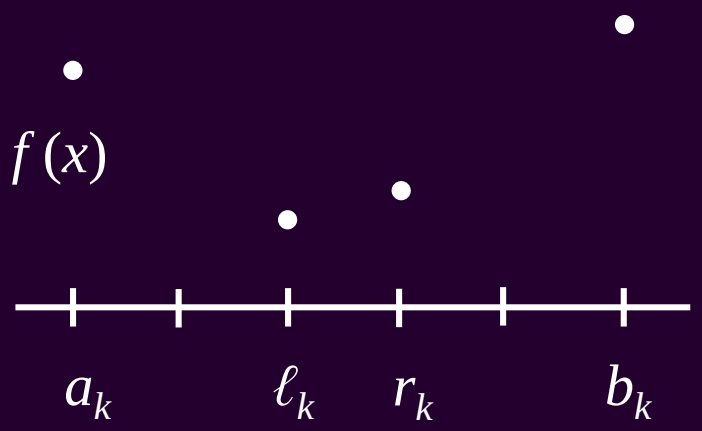


Fifths

- Let's divide the interval into fifths:

$$\ell_k \leftarrow \frac{3a_k + 2b_k}{5} = b_k - \frac{3}{5}(b_k - a_k)$$

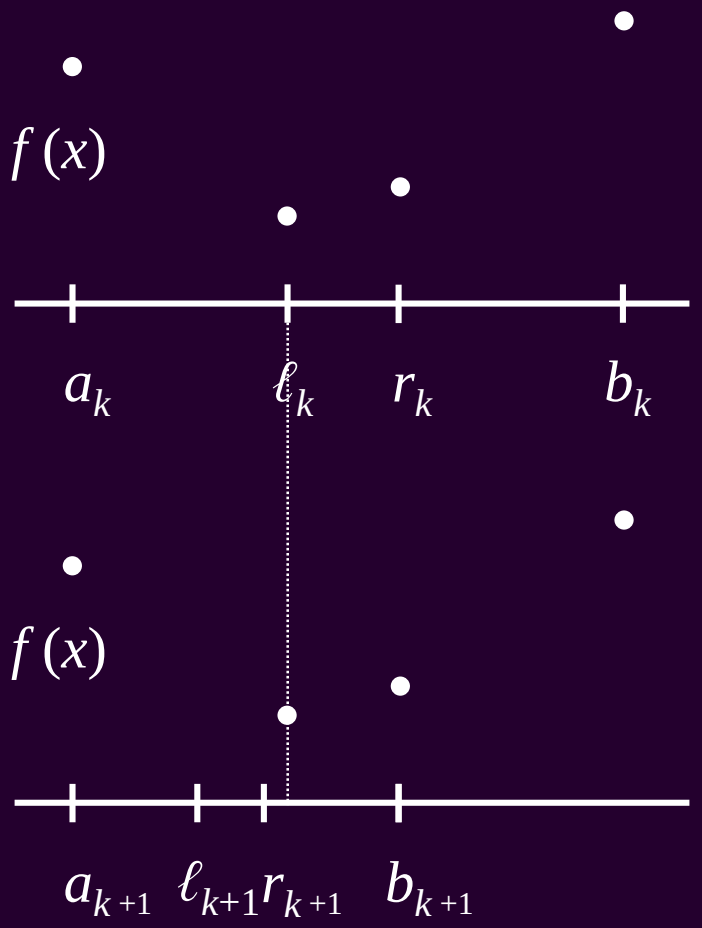
$$r_k \leftarrow \frac{2a_k + 3b_k}{5} = a_k + \frac{3}{5}(b_k - a_k)$$



Fifths

- Having divided the interval into fifths:

$$r_{k+1} \approx \ell_k$$



Golden ratio

- Can we choose a scaling factor so that

$$r_{k+1} = \ell_k \, ?$$

  - Let's choose a scaling factor s :

$$\ell_k \leftarrow b_k - s(b_k - a_k)$$

$$r_k \leftarrow a_k + s(b_k - a_k)$$

  - Suppose $b_{k+1} \leftarrow r_k$

$$r_{k+1} \leftarrow a_{k+1} + s(b_{k+1} - a_{k+1})$$

$$= a_k + s(s(b_k - a_k)) \quad = a_k + s^2(b_k - a_k)$$

$$b_k - s(b_k - a_k) = a_k + s^2(b_k - a_k)$$

Golden ratio

- Thus, if $r_{k+1} = \ell_k$

$$b_k - s(b_k - a_k) = a_k + s^2(b_k - a_k)$$

  - Expand this out:

$$0 = s^2(b_k - a_k) + s(b_k - a_k) + (a_k - b_k)$$
$$= (b_k - a_k)(s^2 + s - 1)$$

  - Thus, we have:

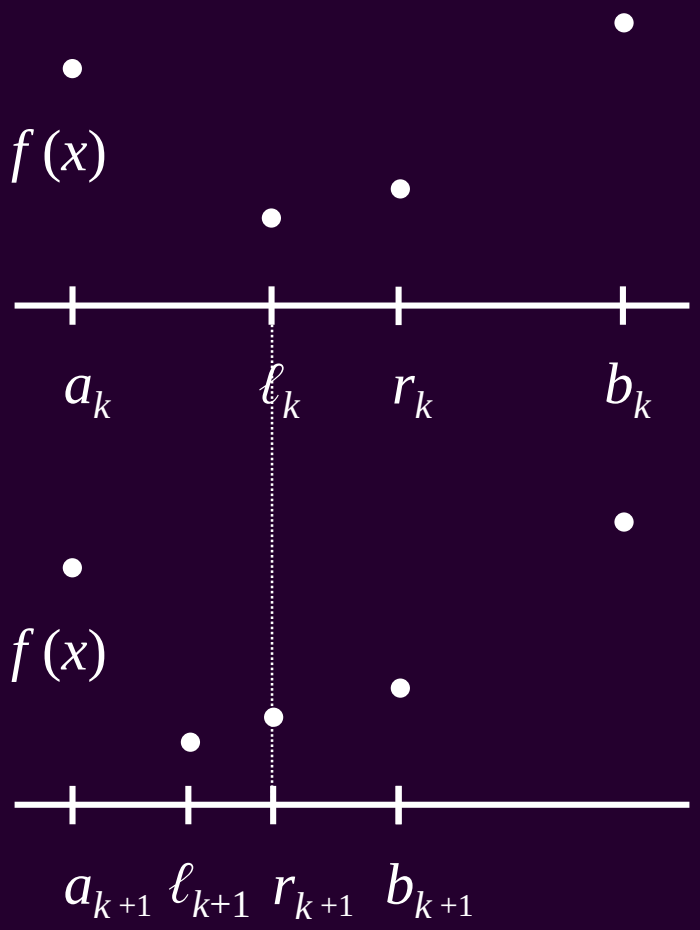
$$s = \frac{-1 \pm \sqrt{1^2 - 4 \cdot 1 \cdot (-1)}}{2 \cdot 1} = \frac{-1 \pm \sqrt{5}}{2}$$

  - Only one of these is positive:

$$s = \frac{-1 + \sqrt{5}}{2} = \frac{\sqrt{5} - 1}{2} \approx 0.618034$$

Golden ratio

- Now we have that $r_{k+1} = \ell_k$:



Golden ratio

- This is called a *golden-ratio search* because

$$\phi = \frac{\sqrt{5}+1}{2} \approx 1.618034$$

$$\frac{1}{\phi} = \frac{\sqrt{5}-1}{2} \approx 0.618034$$

$$\frac{1}{\phi} = \phi - 1$$

$$\phi^2 = \phi + 1$$

$$\phi^3 = 2\phi + 1$$

Golden ratio

- Thus, with each step,
  - We reduce the width of the interval by a factor of $1/\phi$
  - We require only one function evaluation

- Note that with two function evaluations,
  we reduce the interval width to $1/\phi^2 \approx 0.381966$
- Recall when we did trisection, two function evaluations were required with each step
  we reduce the interval width by $2/3 \approx 0.666667$
- Consequently, for the same number of function evaluations,
  we can converge must more quickly

Implementation

```
std::pair<double, double> golden(
    double f( double x ), double a, double b,
    double eps_step, double eps_abs,
    unsigned int max_iterations
) {
    assert( a < b );
    double const INV_PHI{ (std::sqrt(5.0) - 1.0)/2.0 };
    double width{ (b - a)*INV_PHI };
    double x1{ b - width };
    double x2{ a + width };

    double f1{ f( x1 ) };
    double f2{ f( x2 ) };
    assert( std::isfinite( f1 ) && std::isfinite( f2 ) );
```


Implementation

```
for ( unsigned int k{0}; k < max_iterations; ++k ) {
    width *= INV_PHI;

    if ( f1 < f2 ) {
        if ( (width < eps_step) && (std::abs( f1 - f2 ) < eps_abs) ) {
            return std::make_pair( x1, f1 );
        }

         b = x2;
        x2 = x1;
        f2 = f1;
        x1 = b - width;
        f1 = f( x1 );
        assert( std::isfinite( f1 ) );
    } else ...
```

Implementation

```
} else if ( f2 < f1 ) {
    if ( (width < eps_step) && (std::abs( f1 - f2 ) < eps_abs) ) {
        return std::make_pair( x2, f2 );
    }

     a = x1;
    x1 = x2;
    f1 = f2;
    x2 = a + width;
    f2 = f( x2 );
    assert( std::isfinite( f2 ) );
} ...
```

Implementation

```
        } else {
            assert( f1 == f2 );

            if ( width < eps_step ) {
                return std::make_pair( (x1 + x2)/2.0, f1 );
            }

            a = x1;
            b = x2;
            width *= INV_PHI*INV_PHI;
            x1 = b - width;
            f1 = f( x1 );
            x2 = a + width;
            f2 = f( x2 );
            assert( std::isfinite( f1 ) && std::isfinite( f2 ) );
        }
    }

    return std::make_pair( NAN, NAN );
}
```

Example

- Find the first minimum of $2e^{-2x} - e^{-x}$ starting with [1, 2]

1	1.3819660113	1.6180339887	2
–0.0972088747	–0.1249976480	–0.1196517697	–0.0987040055
1	1.2360679775	1.3819660113	1.6180339887
–0.0972088747	–0.1217155585	–0.1249976480	–0.1196517697
1.2360679775	1.3819660113	1.4721359550	1.6180339887
–0.1217155585	–0.1249976480	–0.1241541532	–0.1196517697
1.2360679775	1.3262379212	1.3819660113	1.4721359550
–0.1217155585	–0.1245211032	–0.1249976480	–0.1241541532

Summary

- Following this topic, you now
  - Understand a bracketed method for finding a minimum
  - Are aware of the usefulness of using the reciprocal of the golden ratio
  - Have seen an implementation
  - Have seen an example

References

[1] https://en.wikipedia.org/wiki/Golden-section_search

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

These slides are provided for the ECE 204 *Numerical methods* course taught at the University of Waterloo. The material in it reflects the author's best judgment in light of the information available to them at the time of preparation. Any reliance on these course slides by any party for any other purpose are the responsibility of such parties. The authors accept no responsibility for damages, if any, suffered by any party as a result of decisions made or actions based on these course slides for any other purpose than that for which it was intended.