COL100: Introduction to Computer Science

# 12: Numerical methods

# Conditioning and stability

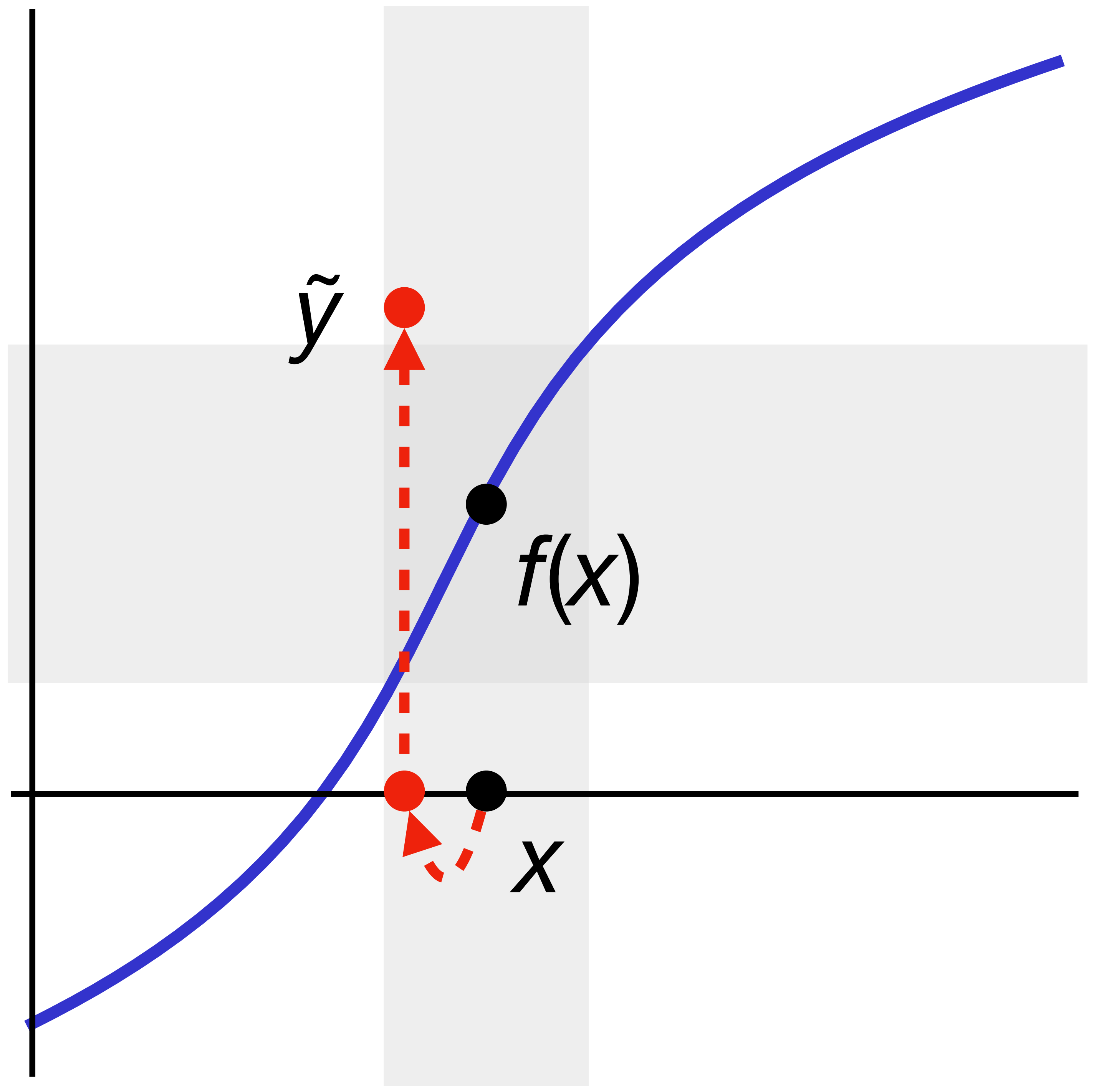
Given inaccurate inputs and inaccurate arithmetic, how accurate of an answer can we expect?

**Conditioning:** how sensitive the *problem* is to errors in *input*

**Stability:** how sensitive an *algorithm* is to errors in *computation*

# Stability

An algorithm is **stable** if error due to computation is not too large, relative to error due to input




$\tilde{y} \approx f(\tilde{x})$ and $\tilde{x} \approx x$

"nearly* the right answer
to nearly* the right question"

*with relative error $O(u)$

# Example: An unstable algorithm

Let $f(x) = \sqrt{(x + 1)} - \sqrt{x}$.

Verify that for large $x$, the condition number $\approx 1/2$.

Compute $f(12345)$ with six decimal digits:

$$
\begin{gathered}
f(12345) \\
= \sqrt{12346} - \sqrt{12345} \\
\approx 111.113 - 111.108 \\
= 5 \times 10^{-3}
\end{gathered}
$$

Differs from correct answer $4.50003\ldots \times 10^{-3}$ by more than 10%!

# Example: An unstable algorithm

Rewrite function as $f(x) = (\sqrt{(x + 1)} + \sqrt{x})^{-1}$.

Compute $f(12345)$ again with six decimal digits:

$$
\begin{aligned}
&f(12345) \\
&= (\sqrt{12346} + \sqrt{12345})^{-1} \\
&\approx (111.113 + 111.108)^{-1} \\
&= 222.221^{-1} \\
&= 4.50002 \times 10^{-3}
\end{aligned}
$$

Now the result is correct to five digits.

# Example: Solving linear systems

$$\underbrace{\begin{bmatrix} 1.00 & 2.01 \\ 1.01 & 2.03 \end{bmatrix}}_{A} \underbrace{\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} 1.01 \\ 1.02 \end{bmatrix}}_{b}$$

**Problem:** Solve $Ax = b$, i.e. compute $x = f(b) = A^{-1}b$. Exact solution: $x = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$

**Naive algorithm:** Compute $A^{-1}$, multiply with $b$. With 3 decimal digits,

$$\mathrm{inv}(A) = \begin{bmatrix} -2.03 \times 10^4 & 2.01 \times 10^4 \\ 1.01 \times 10^4 & -1.00 \times 10^4 \end{bmatrix}, \quad \mathrm{inv}(A) \odot b = \begin{bmatrix} 0.00 \\ 0.00 \end{bmatrix}$$

Instead, do Gaussian elimination on $A$ and $b$, get $\tilde{x} = \begin{bmatrix} 1.01 \\ 0.00 \end{bmatrix}$. Why is this better?

# Exercise

- Given a large set of data $x_1$, $x_2$, …, $x_n$, there are two mathematically equivalent ways to compute the variance,

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \quad \text{where } \mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

$$= \frac{1}{n}\sum_{i=1}^{n} x_i^2 - \left(\frac{1}{n}\sum_{i=1}^{n} x_i\right)^2$$

Which one is more numerically stable? Construct a simple example with known $\mu$ and $\sigma^2$ where the difference in accuracy can be observed.

# Infinite series

How to compute $e^{\pi}$ in the first place?

- $\pi \neq$ `math.pi` $\in \mathbb{F}$ …rounding error
- Cannot sum infinitely many terms …truncation error

$$\mathtt{exp}(x) = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!}$$

**Exercise:** Let $H(n) = 1 + ½ + ⅓ + \cdots + 1/n$. Why doesn't $H(n)$ grow arbitrarily large when summed naively in floating-point arithmetic? How can you fix it?

# Finite differences

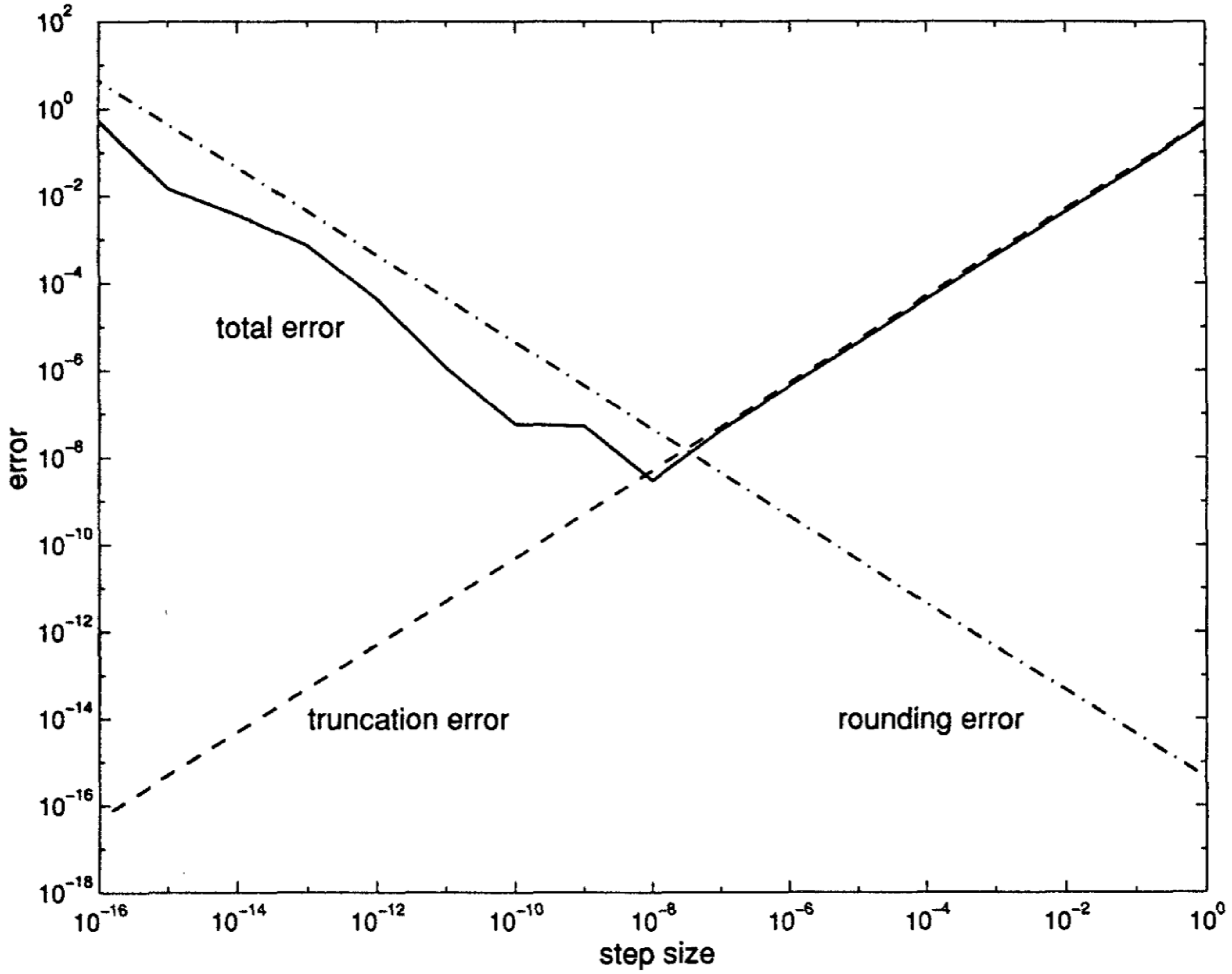
Given an black-box function $f$, how to compute (or approximate) $f'(x)$?

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

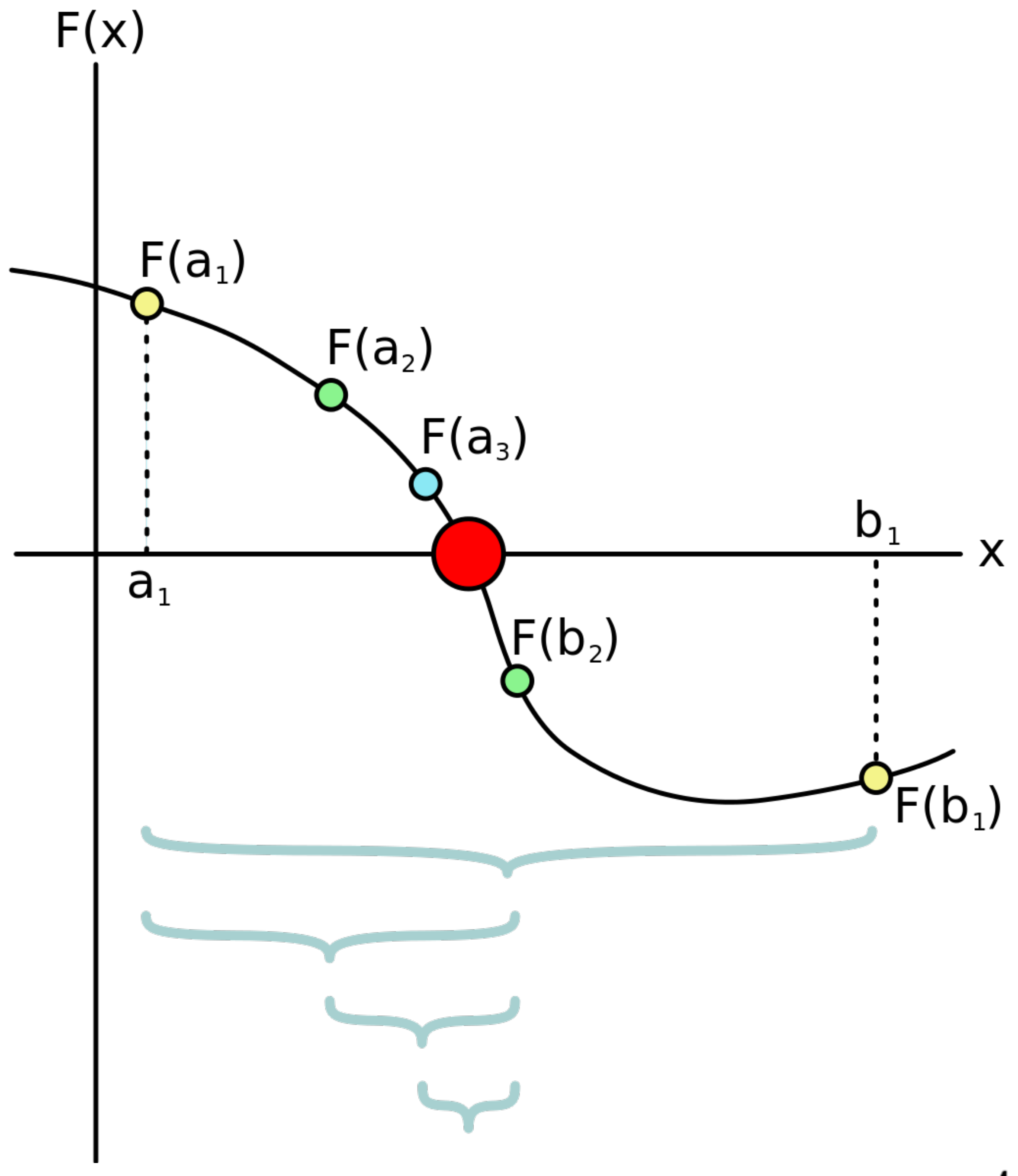
- Taylor series ⇒ truncation error = $O(h)$
- But rounding error = $O(u/h)$

Total error is minimized at $h = O(u^{1/2})$

**Exercise:** Show that $(f(x+h) - f(x-h))/(2h)$ has $O(h^2)$ truncation error, allowing $h = O(u^{2/3})$



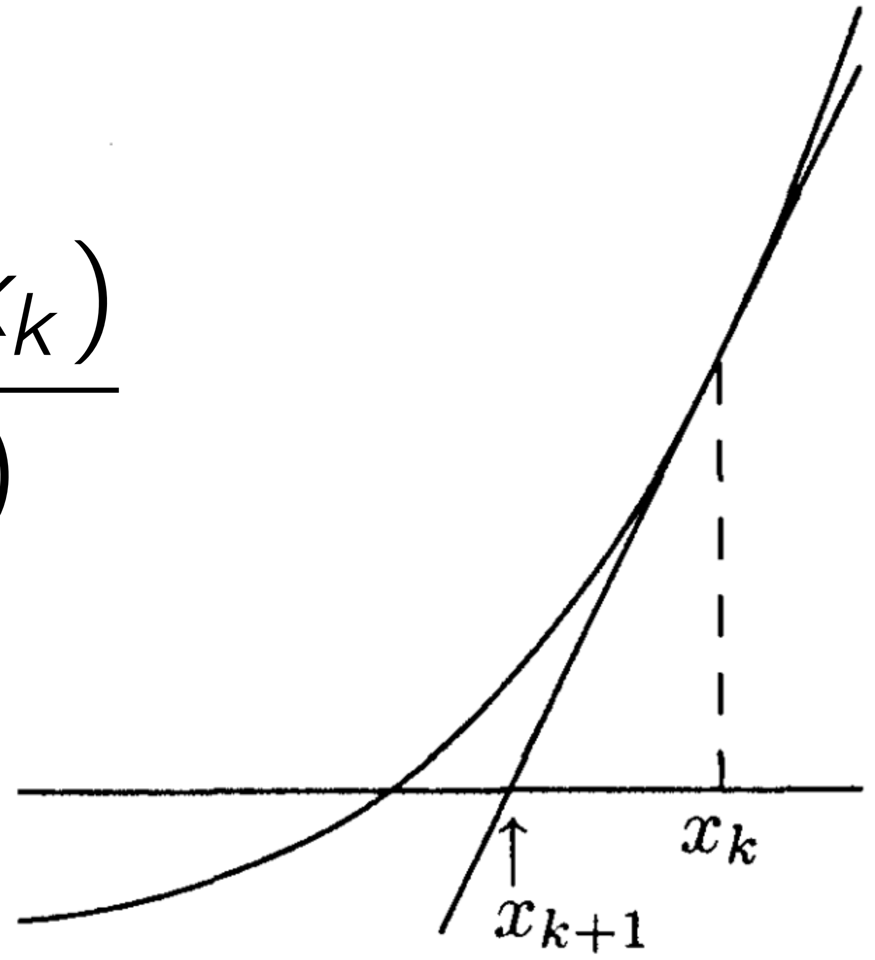
# Root finding




Given $f : \mathbb{R} \to \mathbb{R}$ and $y \in \mathbb{R}$, find $x^* \in \mathbb{R}$ such that $f(x^*) = y$.

- If $f$ continuous and given **bracket** $[a, b]$ s.t. $f(a) < y < f(b)$, apply **bisection** (analogous to binary search in array)
    - Error bound $|b-a|$ decreases by ½ on each iteration
- If $f$ differentiable, **Newton's method**

$$f(x_{k+1}) \approx f(x_k) + (x_{k+1} - x_k) f'(x_k) \implies x_{k+1} = x_k + \frac{y - f(x_k)}{f'(x_k)}$$




- Error decreases as $|x_{k+1} - x^*| = O(|x_k - x^*|^2)$
  …if $f'(x^*) \neq 0$ *and* $x_k$ is close to $x^*$!

# Afterwards

- Read the notes on floating-point numbers and numerical computation, Sec. 1.5.2 onwards
- Derive Newton's method for computing $\sqrt{y}$ by solving $x^2 = y$, and compare with the Babylonian method $x_{k+1} = \frac{1}{2}(x_k + y/x_k)$.