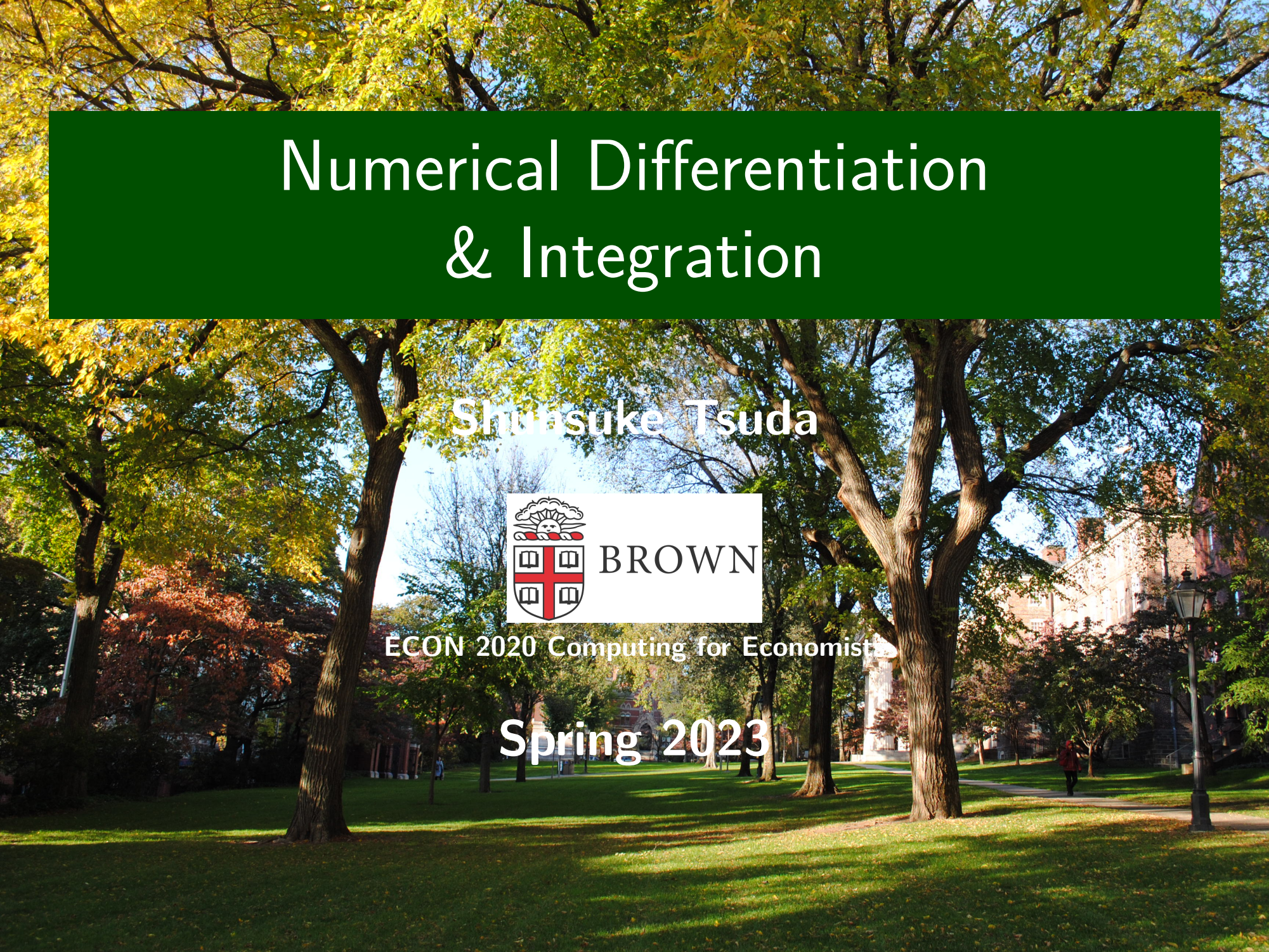


Numerical Differentiation & Integration

Shunsuke Tsuda

ECON 2020 Computing for Economists

Spring 2023

Motivation

Differentiation and integration are two basic operations in scientific computation and a lot of economic applications. For example:

- Optimization and non-linear equation solving (gradients, Hessians, Jacobians)
- Differential equations
- Computing expectations given some probability distributions (integration)
- Computing consumer surplus (integration)
- Statistics and econometrics

Numerical Differentiation

Why Do We Need Numerical Differentiation?

- Any “approximation” involves errors.

Why Do We Need Numerical Differentiation?

- Any "approximation" involves errors.
- From the standpoint of computational accuracy and time (by computer), obtaining derivatives analytically should be the first priority.

Why Do We Need Numerical Differentiation?

- Any "approximation" involves errors.
- From the standpoint of computational accuracy and time (by computer), obtaining derivatives analytically should be the first priority.
- However, we encounter situations where the explicit form of a function, whose derivative is of our interest, is unknown (e.g., simulated method of moments).

Why Do We Need Numerical Differentiation?

- Any "approximation" involves errors.
- From the standpoint of computational accuracy and time (by computer), obtaining derivatives analytically should be the first priority.
- However, we encounter situations where the explicit form of a function, whose derivative is of our interest, is unknown (e.g., simulated method of moments).
- Even if it is possible to obtain derivatives analytically, it might be extremely time consuming in some cases.
- Also, numerical differentiation helps us to check errors of analytical computations by hand (e.g., deriving theoretical predictions for signs).

Overview

- Direct approach (Forward/Backward difference)
- Central difference
- Partial derivatives
- Higher-order and cross derivatives
- Other methods:
    - Three-point approximations
    - Richardson extrapolation

Direct Approach (Forward Difference)

- Derivative of a function:

$$f'(x) = \lim_{\triangle x \to 0} \frac{f(x + \triangle x) - f(x)}{\triangle x}$$

Direct Approach (Forward Difference)

- Derivative of a function:

$$f'(x) = \lim_{\triangle x \to 0} \frac{f(x + \triangle x) - f(x)}{\triangle x}$$

- A natural approximation ("forward difference"):

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

for some small $h > 0$.

Direct Approach (Forward Difference)

- Derivative of a function:

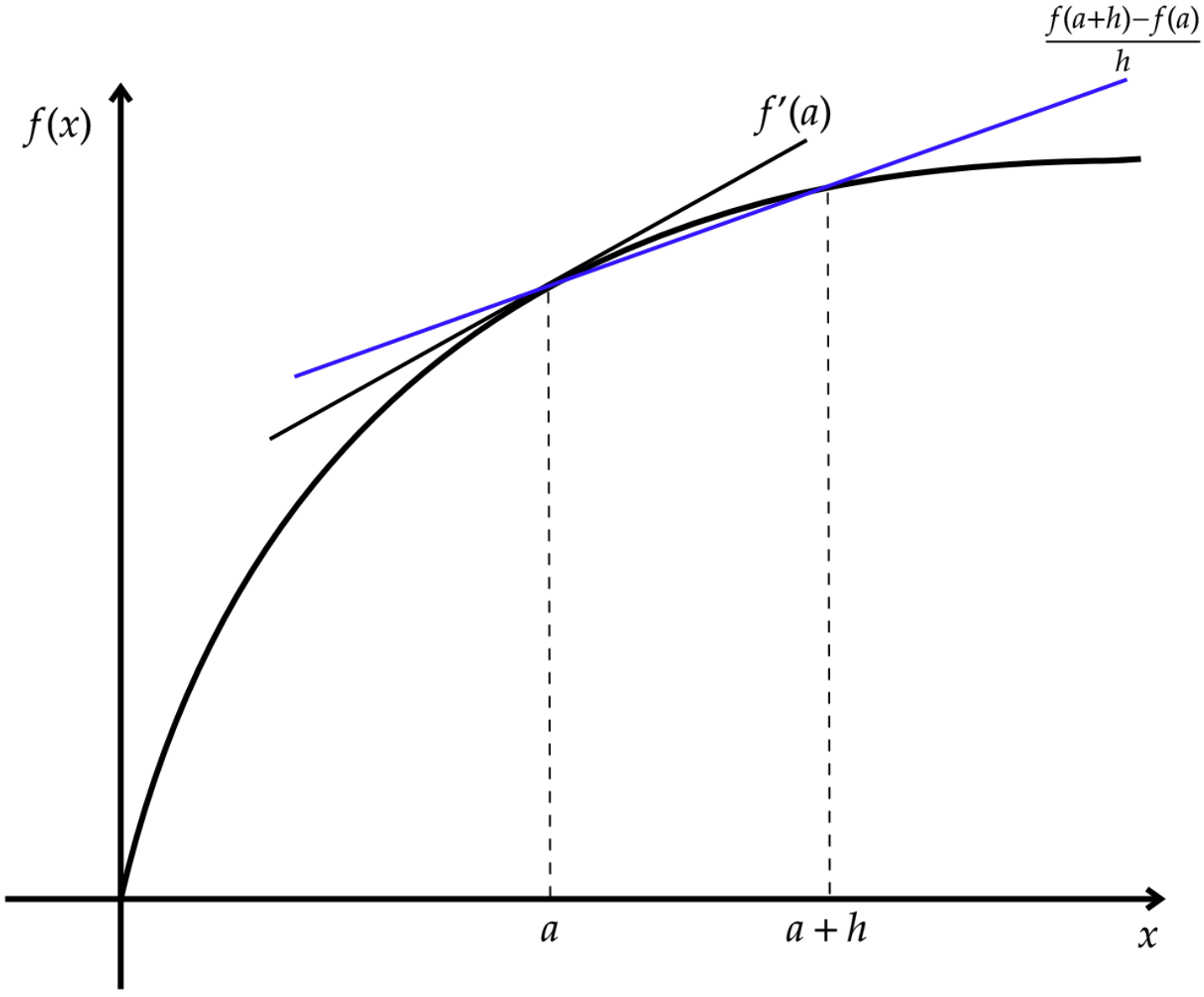
$$f'(x) = \lim_{\triangle x \to 0} \frac{f(x + \triangle x) - f(x)}{\triangle x}$$

- A natural approximation ("forward difference"):

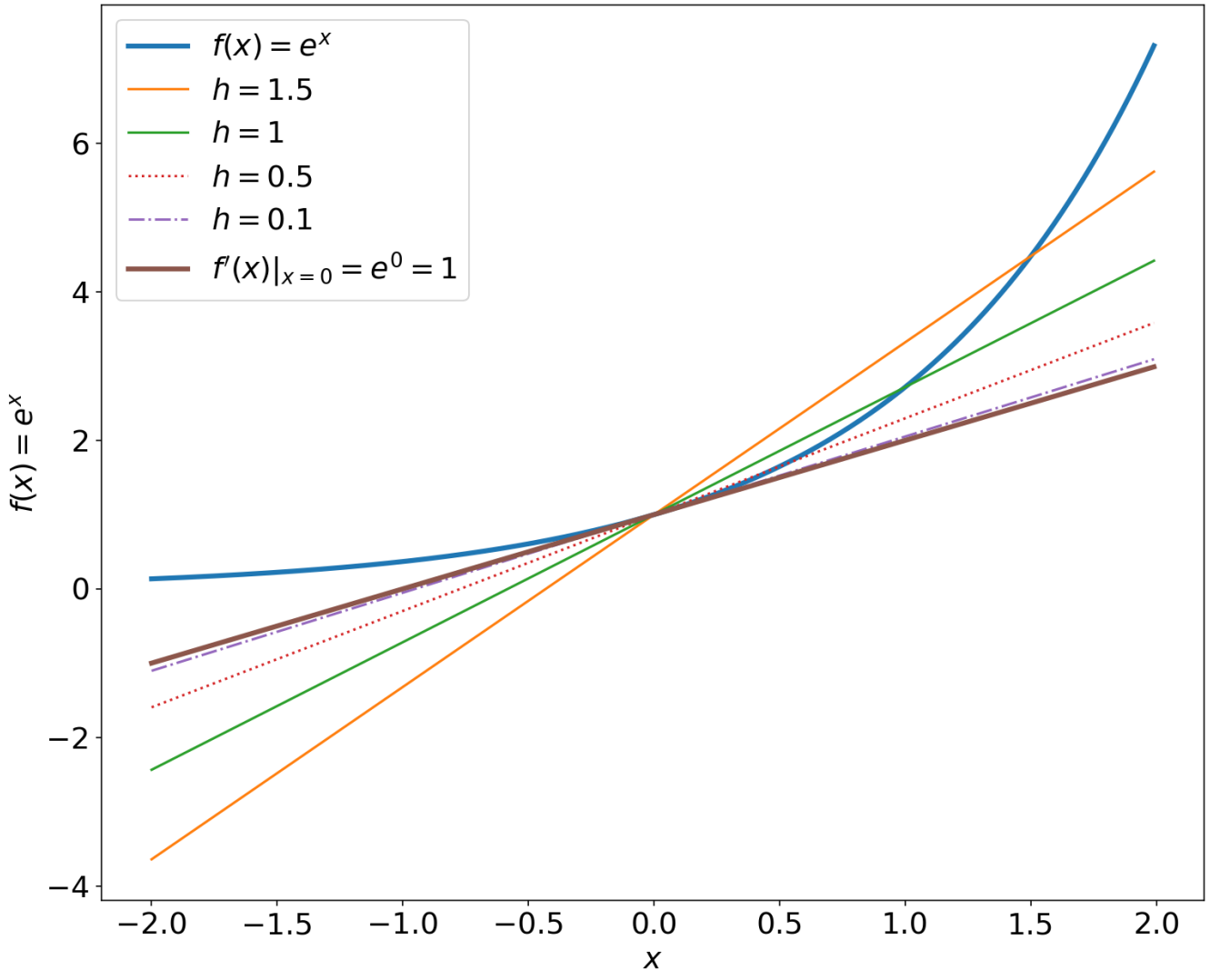
$$f'(x) \approx \frac{f(x + h) - f(x)}{h}$$

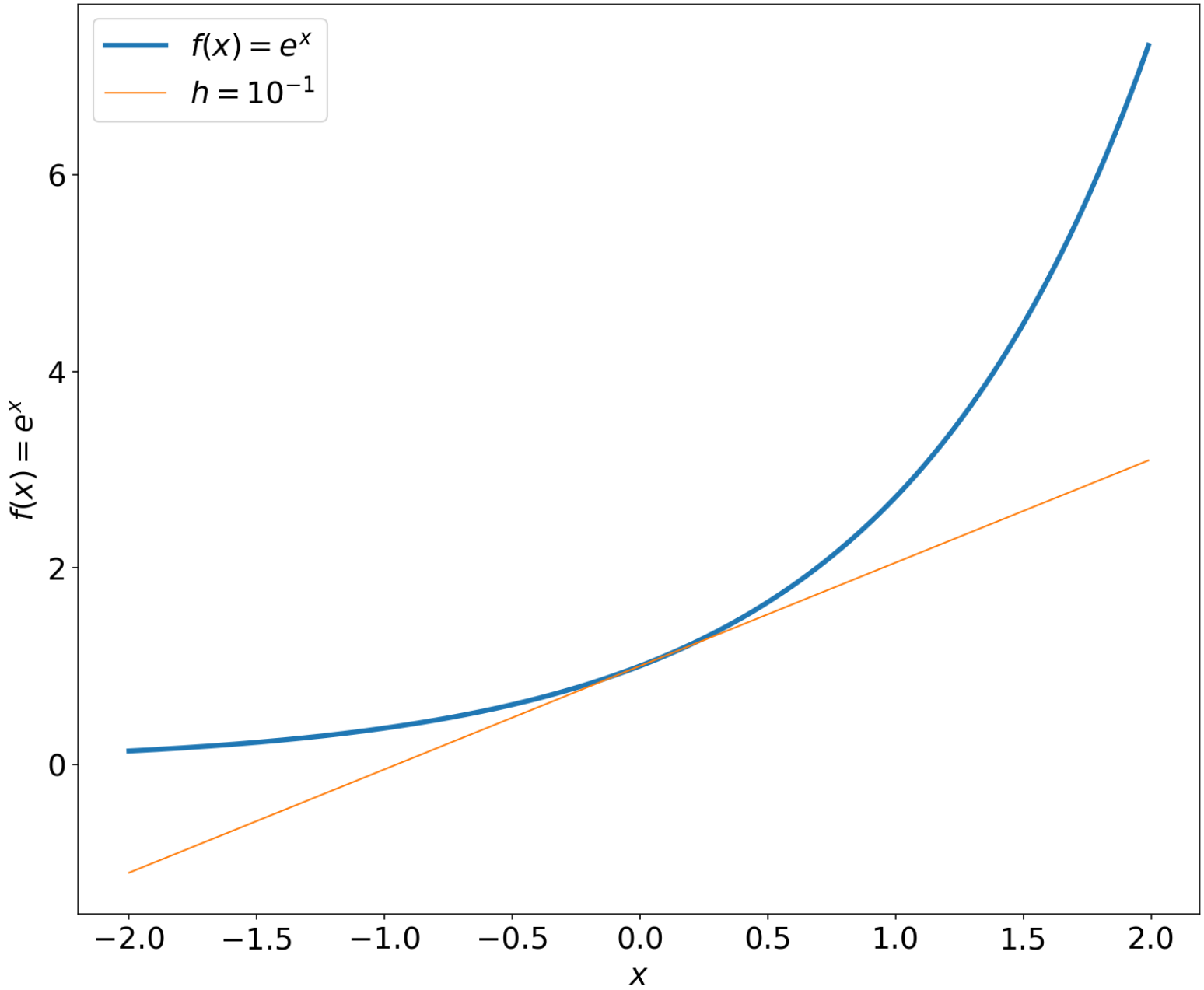
  for some small $h > 0$.
- How small should h be?
  The smaller, the more precise?

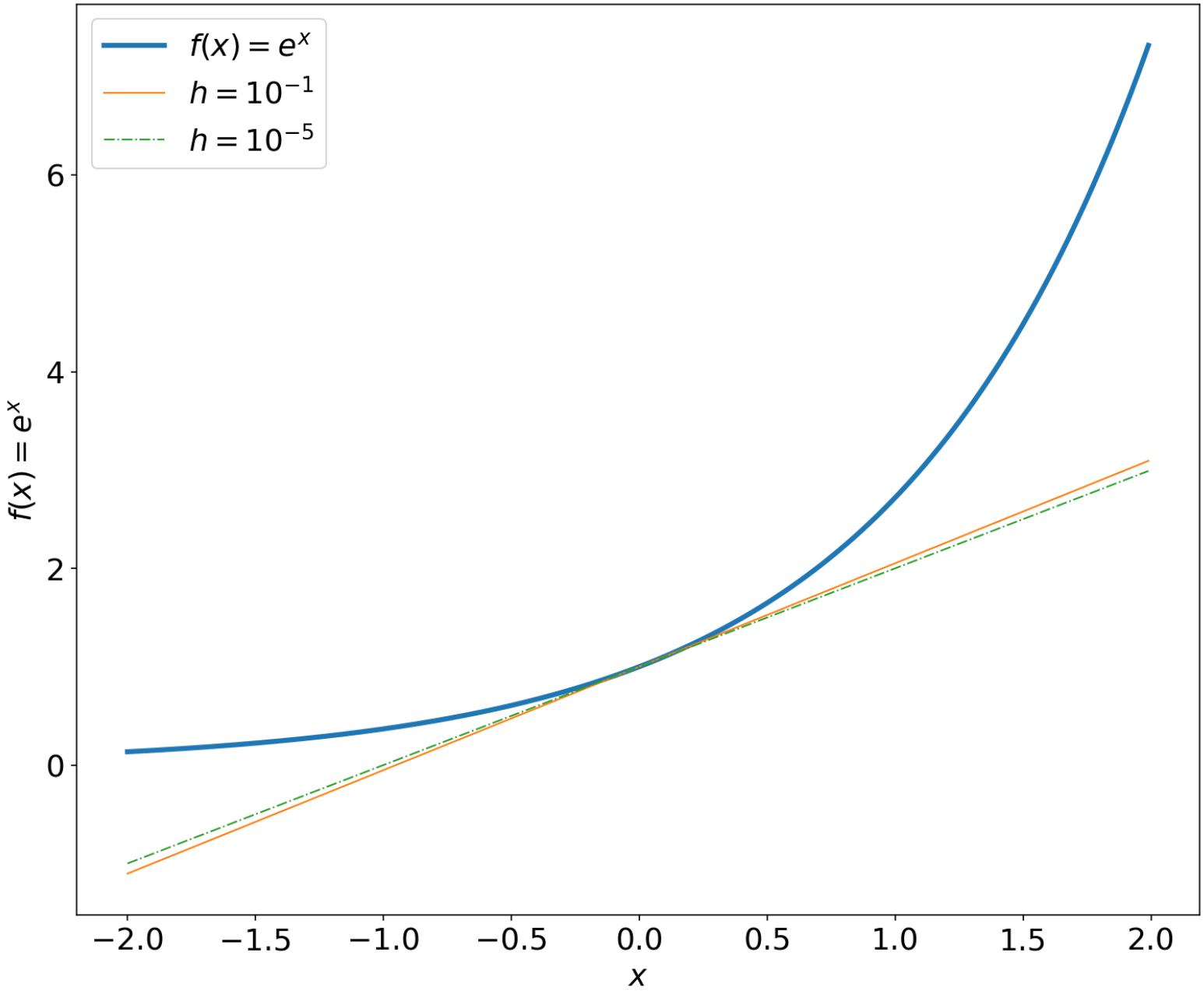
Illustration of the Forward Difference

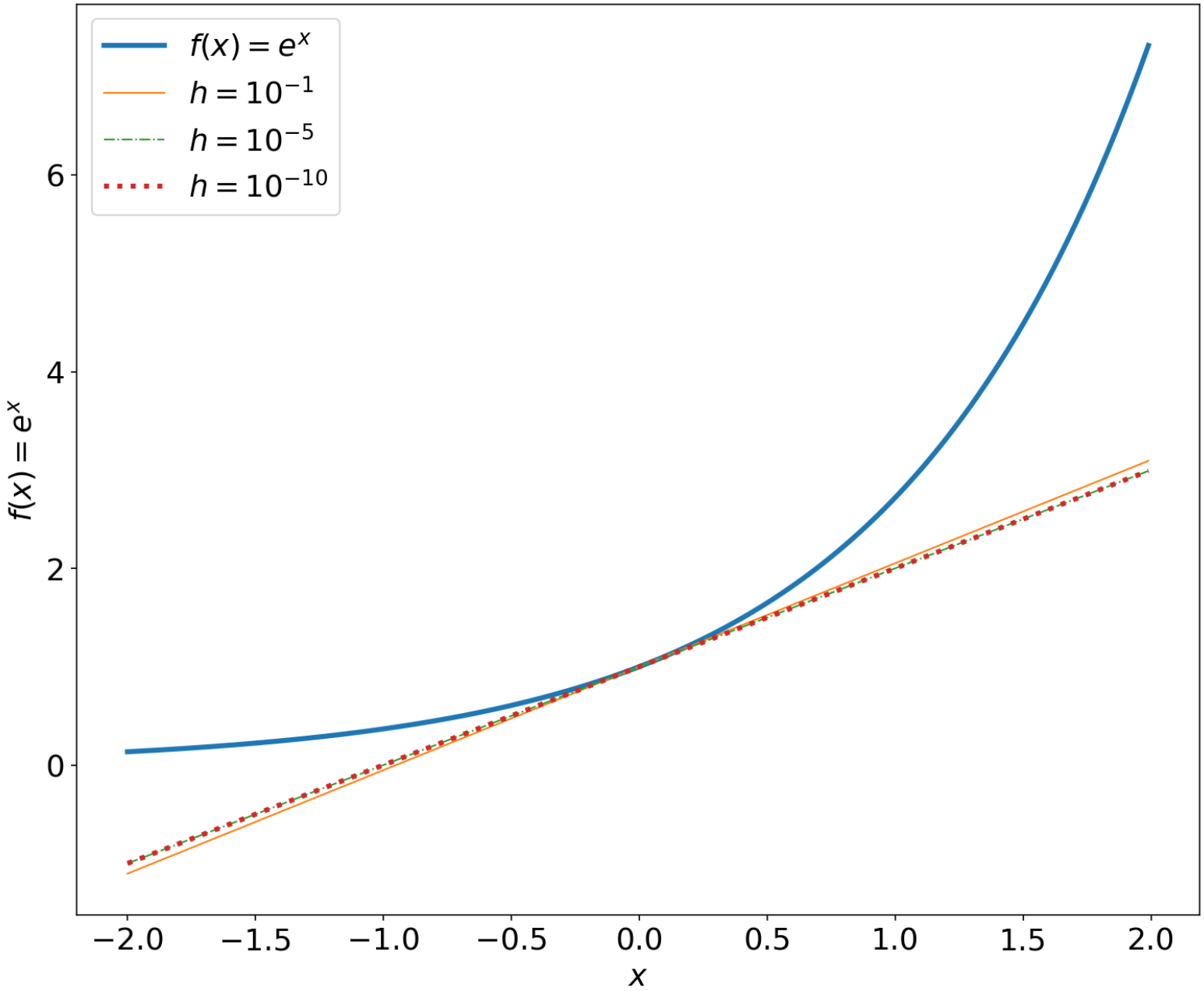


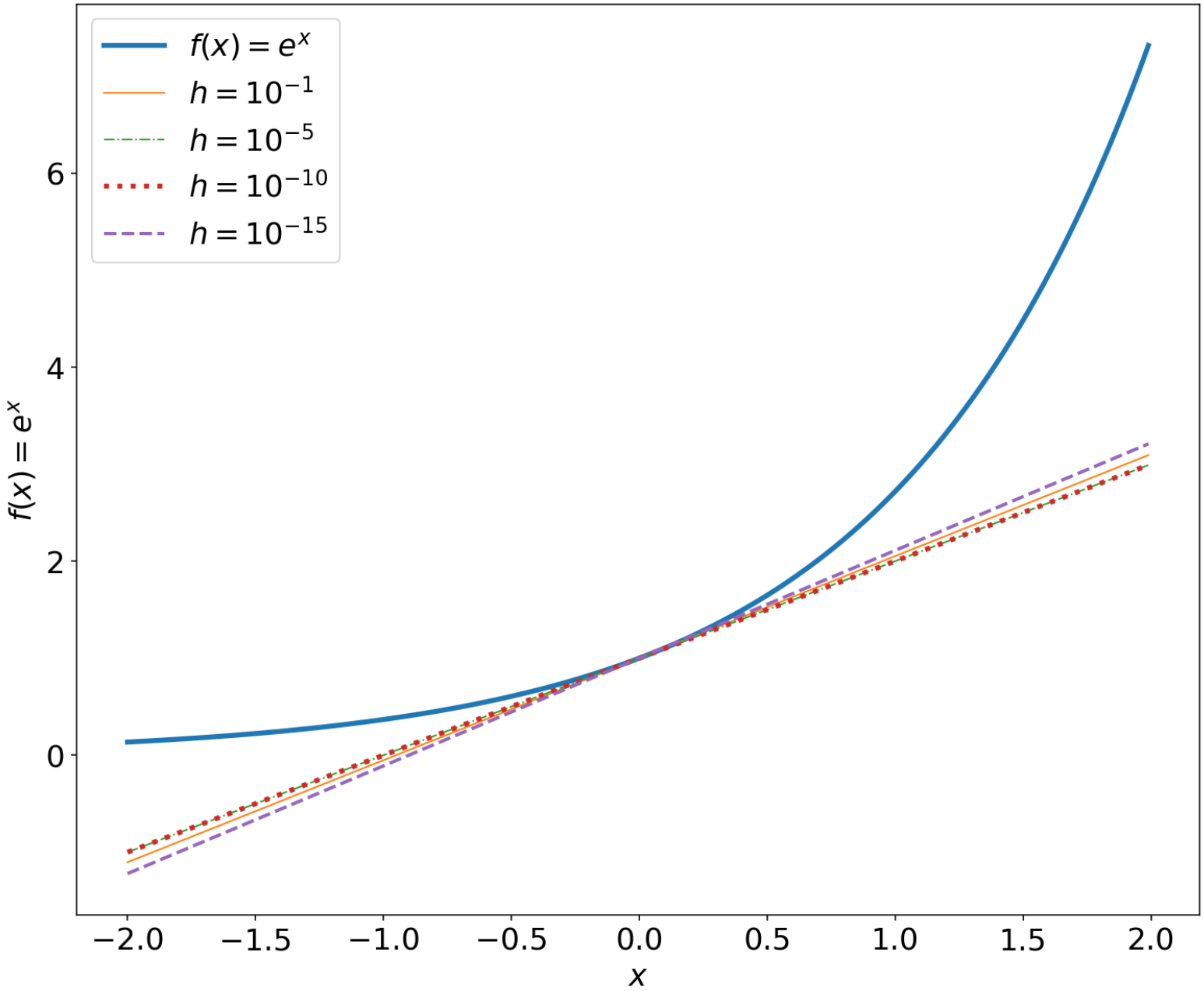
Numerical Illustration of the Forward Difference

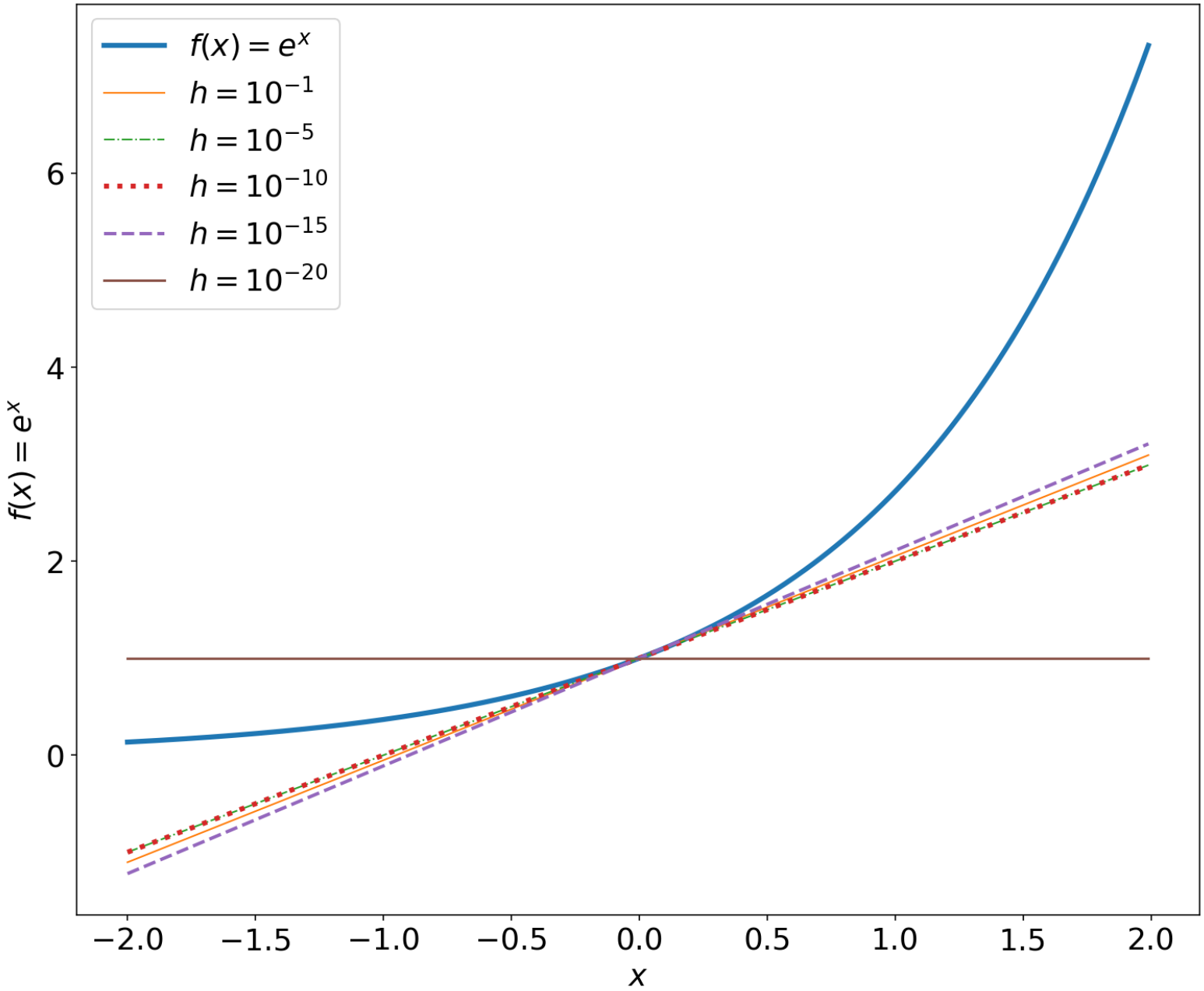



$f(x) = e^x$
$h = 10^{-1}$
6
4
2
0
$f(x) = e^x$
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x

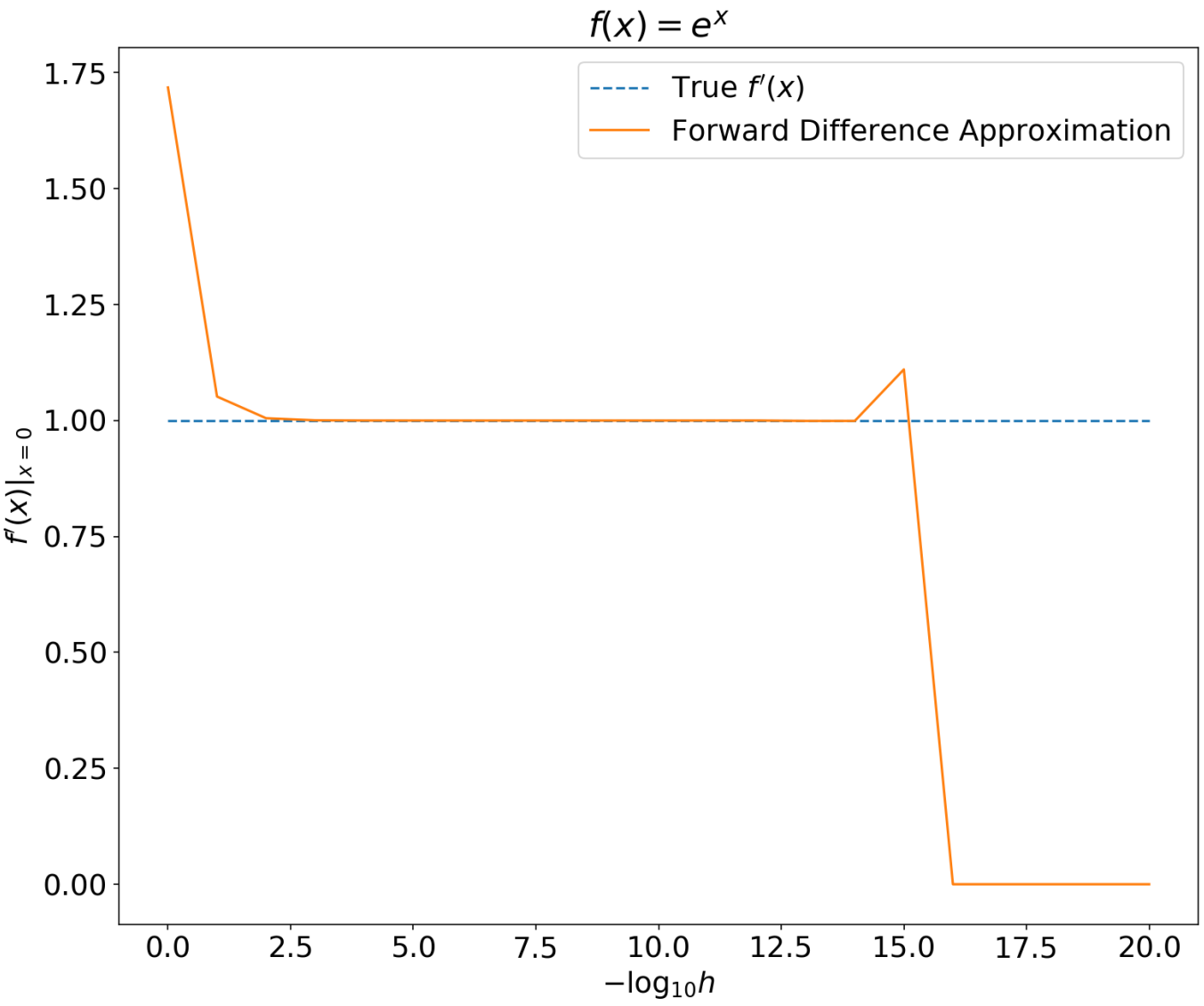

$f(x) = e^x$
$h = 10^{-1}$
$h = 10^{-5}$
6
4
2
0
$f(x) = e^x$
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x


$f(x) = e^x$
$h = 10^{-1}$
$h = 10^{-5}$
$h = 10^{-10}$
$f(x) = e^x$
6
4
2
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x


$f(x) = e^x$
$h = 10^{-1}$
$h = 10^{-5}$
$h = 10^{-10}$
$h = 10^{-15}$
$f(x) = e^x$
6
4
2
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x


$f(x) = e^x$
$h = 10^{-1}$
$h = 10^{-5}$
$h = 10^{-10}$
$h = 10^{-15}$
$h = 10^{-20}$
$f(x) = e^x$
6
4
2
0
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x

Too Small h Lose Precision!



Round-off & Truncation Errors

- Define:
  - $L_f \equiv \max_{\xi \in [x, x+h]} |f(\xi)|$
  - $\hat{f}(x)$: computed value of $f(x)$
  - ϵ: machine precision such that:
    $|f(x) - \hat{f(x)}| \leq \epsilon L_f \quad \forall x$

Round-off & Truncation Errors

- Define:
  - $L_f \equiv \max_{\xi \in [x, x+h]} |f(\xi)|$
  - $\hat{f}(x)$: computed value of $f(x)$
  - ϵ: machine precision such that:
    $|f(x) - \hat{f}(x)| \leq \epsilon L_f \quad \forall x$
- **Round-off error**: $\left| \frac{f(x+h)-f(x)}{h} - \frac{\hat{f}(x+h)-\hat{f}(x)}{h} \right| \leq \frac{2\epsilon L_f}{h}$

Round-off & Truncation Errors

- Define:
  - $L_f \equiv \max_{\xi \in [x, x+h]} |f(\xi)|$
  - $\hat{f}(x)$: computed value of $f(x)$
  - ϵ: machine precision such that:
    $|f(x) - \hat{f(x)}| \leq \epsilon L_f \quad \forall x$
- **Round-off error**: $|\frac{f(x+h)-f(x)}{h} - \frac{\hat{f}(x+h)-\hat{f}(x)}{h}| \leq \frac{2\epsilon L_f}{h}$
- Taylor expansion of $f(x+h)$ around $h = 0$:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(\xi)}{2}h^2, \; \exists\, \xi \in [x, x+h] \Rightarrow$$

Round-off & Truncation Errors

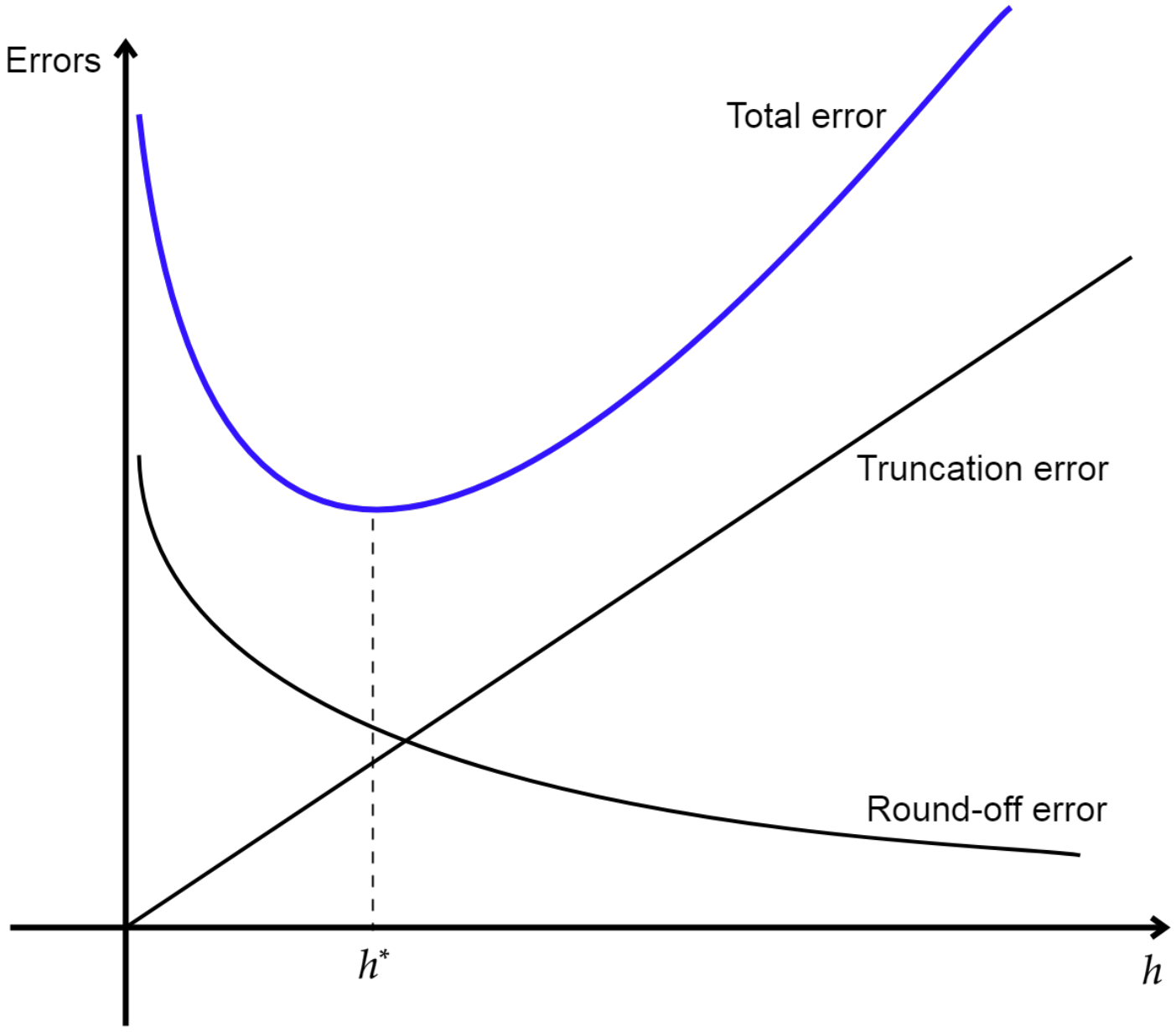
- Define:
  - $L_f \equiv \max_{\xi \in [x, x+h]} |f(\xi)|$
  - $\hat{f}(x)$: computed value of $f(x)$
  - ϵ: machine precision such that:
    $|f(x) - \hat{f(x)}| \leq \epsilon L_f \quad \forall x$
- **Round-off error**: $|\frac{f(x+h)-f(x)}{h} - \frac{\hat{f}(x+h)-\hat{f}(x)}{h}| \leq \frac{2\epsilon L_f}{h}$
- Taylor expansion of $f(x+h)$ around $h = 0$:

$$f(x+h) = f(x) + f'(x)h + \frac{f''(\xi)}{2}h^2, \ \exists\, \xi \in [x, x+h] \Rightarrow$$

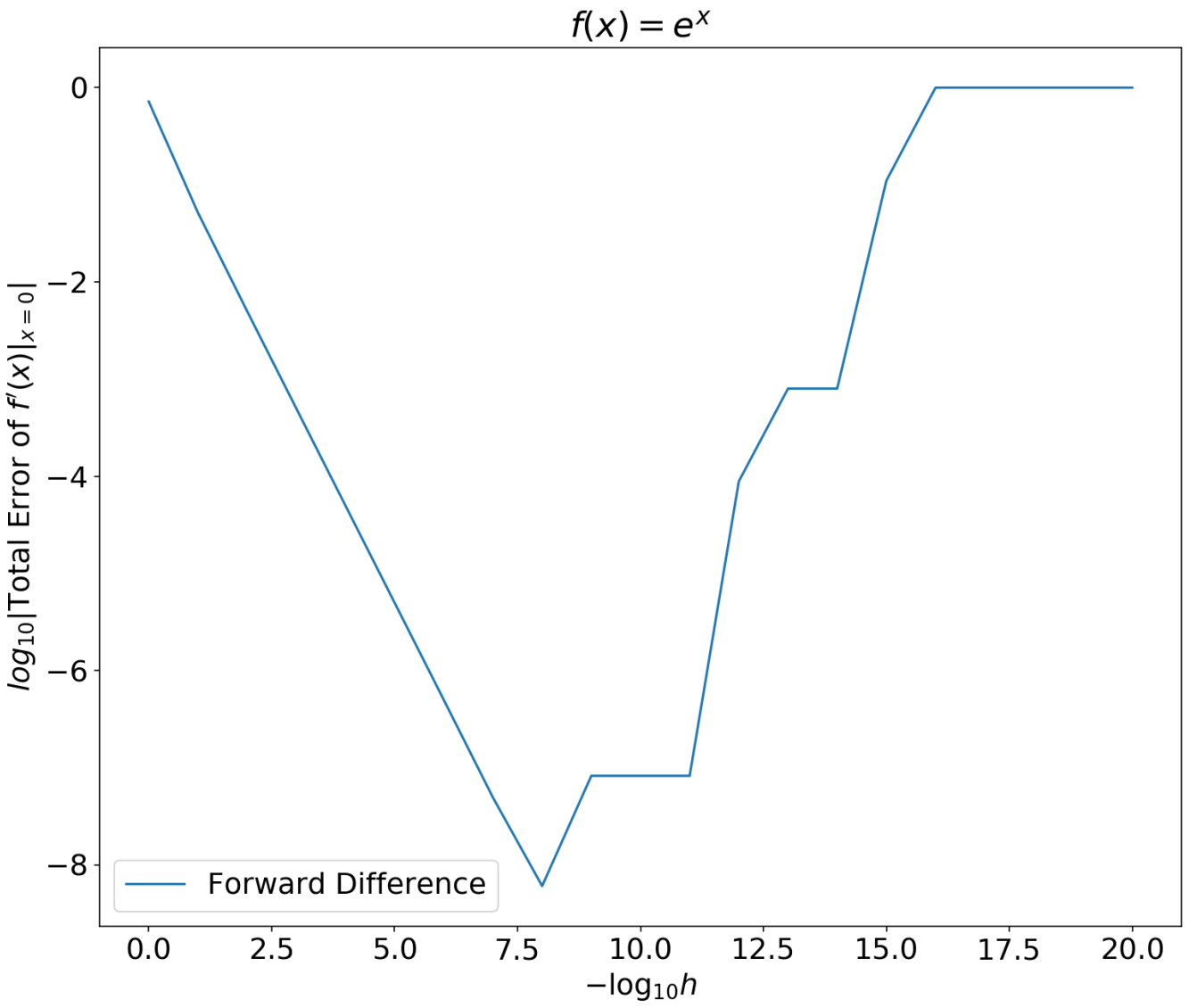
- **Truncation error**: $|f'(x) - \frac{f(x+h)-f(x)}{h}| \leq \frac{hM}{2}$

  where $M \equiv \max_{\xi \in [x, x+h]} |f''(\xi)|$

Total Error is Non-Monotonic in h



Errors from the Previous Example



Total Error is Non-Monotonic in h

- Total error is the sum of round-off and truncation errors.

Total Error is Non-Monotonic in h

- Total error is the sum of round-off and truncation errors.
- **Round-off error**: decreasing in h
  - Take differences and sums of floating-point numbers and divide the result by a small number.
  - Equivalent to multiplying the result by a large number.
  - This multiplication magnifies any round-off errors in numerators, which is larger with a smaller h.

Total Error is Non-Monotonic in h

- Total error is the sum of round-off and truncation errors.
- **Round-off error**: decreasing in h
  - Take differences and sums of floating-point numbers and divide the result by a small number.
  - Equivalent to multiplying the result by a large number.
  - This multiplication magnifies any round-off errors in numerators, which is larger with a smaller h.
- **Truncation error**: increasing in h
  - Mathematical error in the approximation
  - $O(h)$ (the order of error is one)

    $O(h^k)$: the sum of terms with kth and higher powers of h

Optimal h Minimizes the Total Error

- Total error ($=$ truncation $+$ round-off errors):

$$|f'(x) - \frac{\hat{f}(x+h) - \hat{f}(x)}{h}| \leq \frac{2\epsilon L_f}{h} + \frac{hM}{2} \equiv g(h)$$

Recall: $L_f \equiv \max_{\xi \in [x, x+h]} |f(\xi)|$ & $M \equiv \max_{\xi \in [x, x+h]} |f''(\xi)|$

- Optimal h: $h^* = \arg\min_h g(h) = 2\sqrt{\frac{\epsilon L_f}{M}}$

- In practice, often set: $h^* = \max(|x|, 1)\sqrt{\epsilon}$

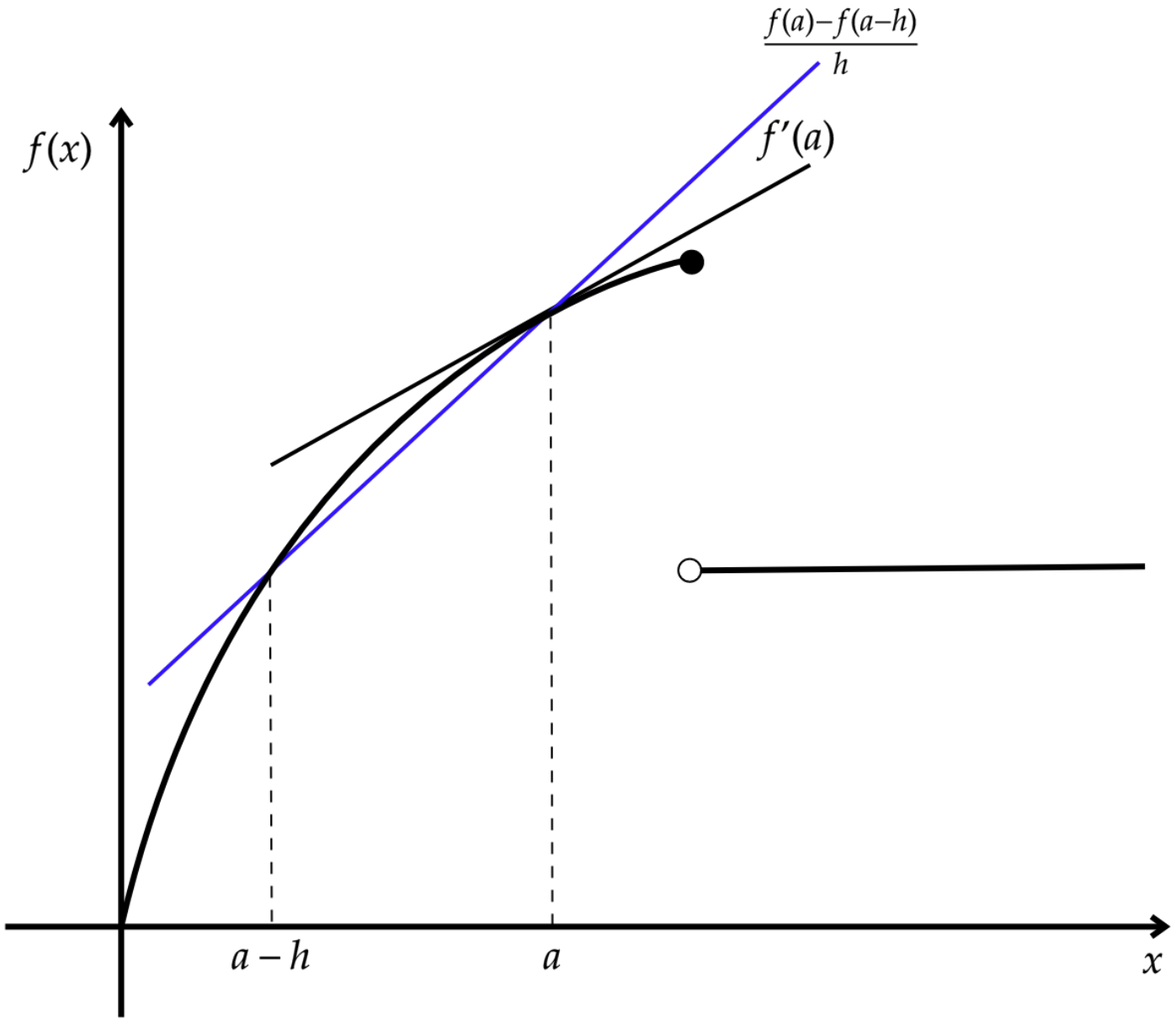
Backward Difference

- Approximate $f'(x)$ by:

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

- Similar properties as the forward difference
- When might we use this backward difference instead of the forward difference?
- Important to have a sense of functional shapes

Prefer Backward Difference When Left Derivative Matters



Central Difference

- Approximate $f'(x)$ by:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

- Manipulate Taylor expansions:

$$f(x+h) - f(x-h) = 2f'(x)h + \frac{f^{(3)}(\xi_1) + f^{(3)}(\xi_2)}{6}h^3 \Rightarrow$$

- Truncation error: $|f'(x) - \frac{f(x+h)-f(x-h)}{2h}| = O(h^2)$

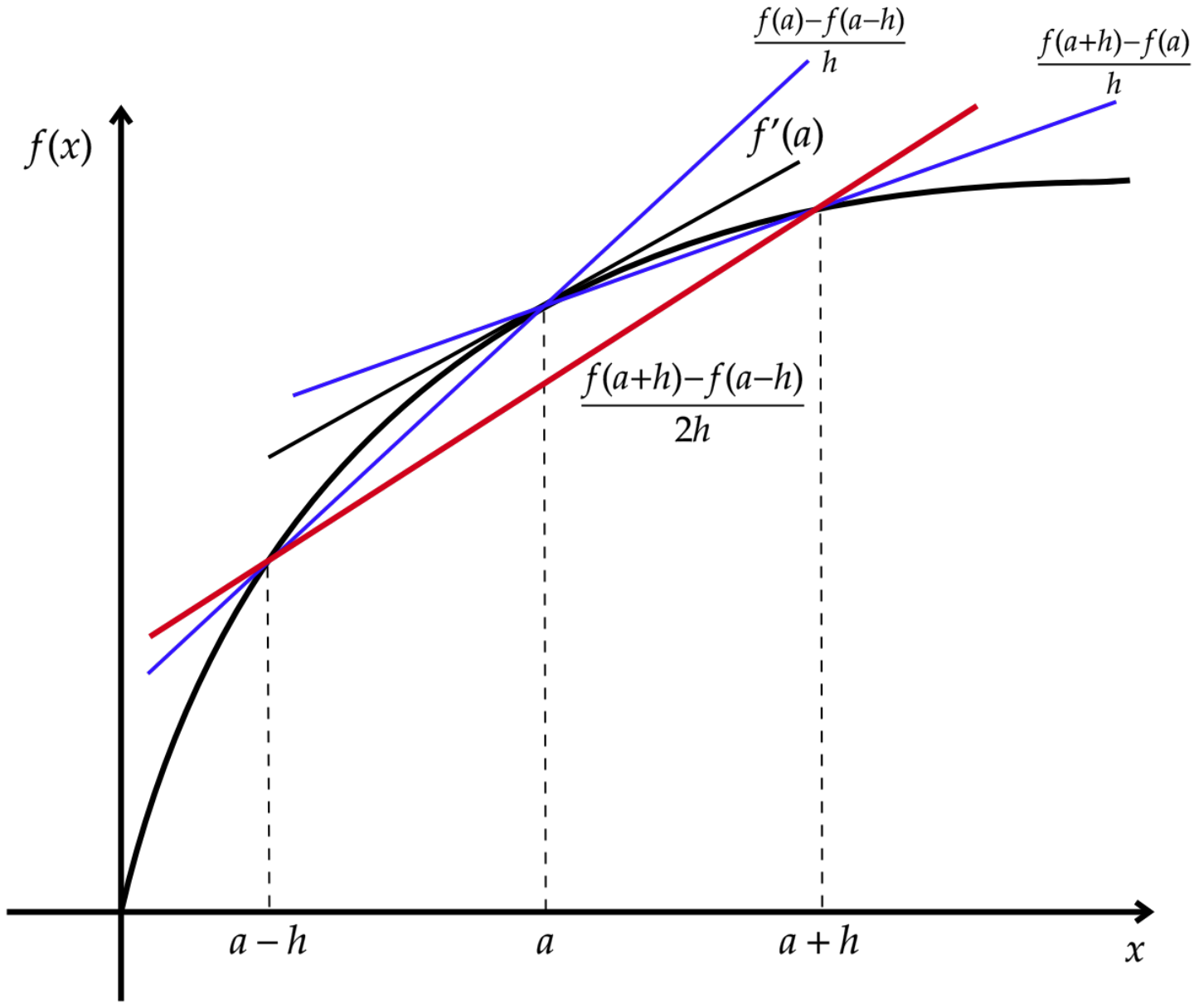


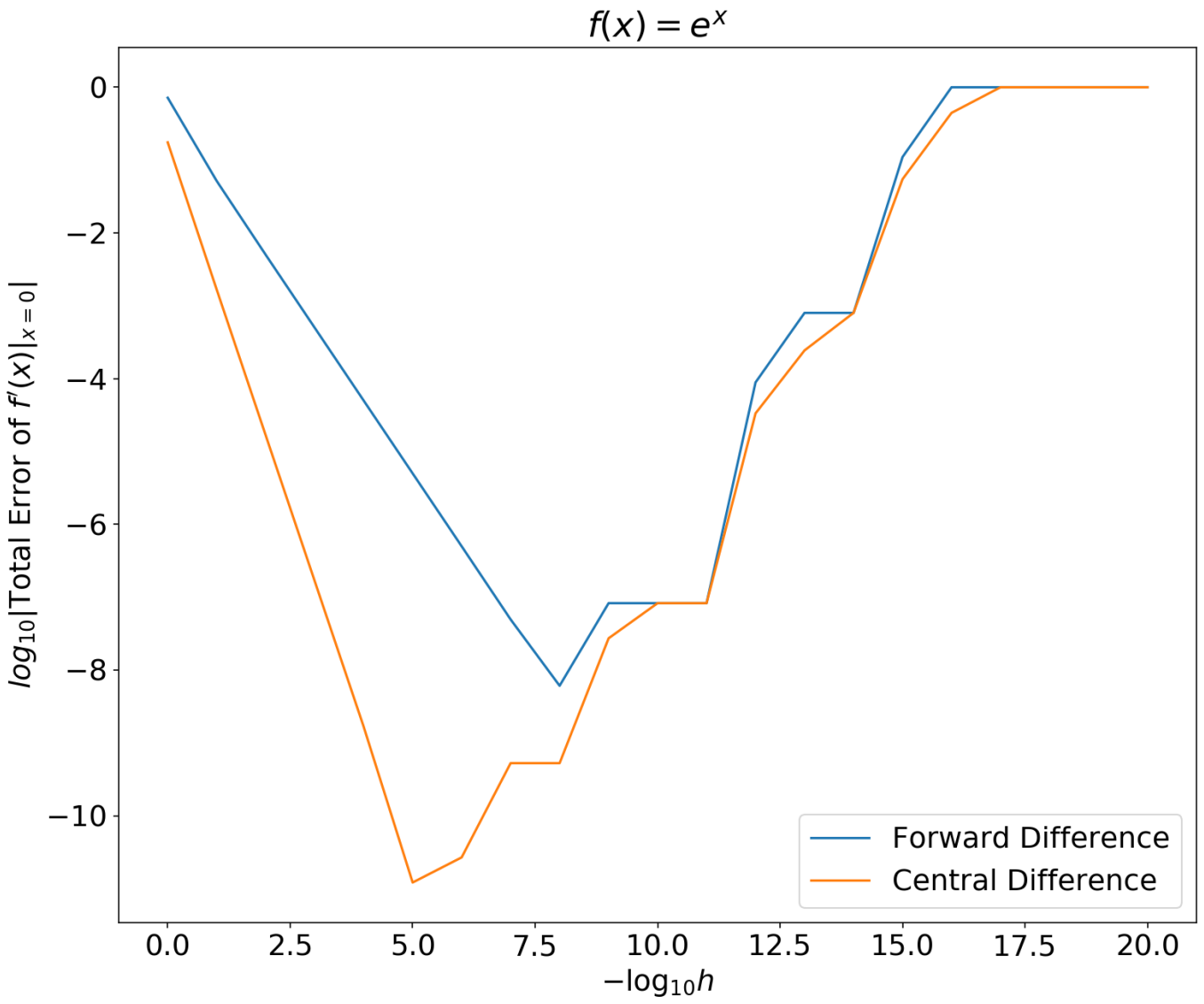
Figure: Higher Accuracy by the Central Difference

Errors and Optimal h

- Round-off error: $\left|\frac{f(x+h)-f(x-h)}{2h} - \frac{\hat{f}(x+h)-\hat{f}(x-h)}{2h}\right| \leq \frac{\epsilon L_f}{h}$
- Truncation error: $\left|f'(x) - \frac{f(x+h)-f(x-h)}{2h}\right| \leq \frac{h^2 M}{6}$

  $(L_f \equiv \max_{\xi \in [x-h,x+h]} |f(\xi)| \ \& \ M \equiv \max_{\xi \in [x-h,x+h]} |f^{(3)}(\xi)|)$
- Total error: $\left|f'(x) - \frac{\hat{f}(x+h)-\hat{f}(x-h)}{2h}\right| \leq \frac{\epsilon L_f}{h} + \frac{h^2 M}{6}$
- $h^* = \frac{\sqrt[3]{3\epsilon L_f}}{\sqrt[3]{M}}$
- In practice, set: $h^* = \max(|x|, 1)\sqrt[3]{\epsilon}$

Central Difference is More Accurate than Forward Difference



Partial Derivatives

- $f : \mathbb{R}^n \rightarrow \mathbb{R}$
- $e_i \equiv \begin{pmatrix} 0 \\ . \\ . \\ 1 \\ . \\ . \\ 0 \end{pmatrix} \leftarrow i$'th element
- In the case of central difference,

$$\frac{\partial f(x)}{\partial x_i} = \frac{f(x + e_i h_i) - f(x - e_i h_i)}{2h_i} + O(h^2)$$

Trade-off b/w Accuracy & Efficiency

- The central difference is one order more accurate than the preceeding one-sided (forward/backward) difference.
- The tradeoff comes into play as we increase the dimensionality of f.
- E.g.) Suppose we compute the Jacobian matrix of $f : \mathbb{R}^n \to \mathbb{R}^m$. How many functional evaluations are required by the one-sided difference and by the central difference?

Trade-off b/w Accuracy & Efficiency

- The central difference is one order more accurate than the preceeding one-sided (forward/backward) difference.
- The tradeoff comes into play as we increase the dimensionality of f.
- E.g.) Suppose we compute the Jacobian matrix of $f : \mathbb{R}^n \to \mathbb{R}^m$. How many functional evaluations are required by the one-sided difference and by the central difference?

  The central difference takes approximately twice as long to compute when n is large.

Higher-Order & Cross Derivatives

- $f : \mathbb{R}^n \longrightarrow \mathbb{R}$
- In the case of central difference,

$$\frac{\partial^2 f(x)}{\partial x_i^2} = \frac{f(x + e_i h_i) - 2f(x) + f(x - e_i h_i)}{h_i^2} + O(h^4)$$

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{f(x + h_i e_i + h_j e_j) - f(x + h_i e_i - h_j e_j) - f(x - h_i e_i + h_j e_j) + f(x - h_i e_i - h_j e_j)}{4 h_i h_j} + O(h^4)$$

where $O(h^4)$ part is proportional to $h_i^2 h_j^2$ for the cross derivative

(Derive them by yourself!)

- In practice, set: $h_i^* = \max(|x|, 1)\sqrt[4]{\epsilon}$
- Numerical differentiation accumulates errors for higher-order derivatives.

Three-point Approximation

- Aproximate the derivative with a weighted sum of evaluated values of the function at 3 points:

$$f'(x) \approx af(x) + bf(x+h) + cf(x+\lambda h)$$

- Use Taylor expansions for $f(x+h)$ and $f(x+\lambda h)$ around x to obtain:

$$\begin{aligned} & af(x) + bf(x+h) + cf(x+\lambda h) \\ &= (a+b+c)f(x) + h(b+c\lambda)f'(x) \\ &+ \tfrac{h^2}{2}(b+c\lambda^2)f''(x) + \tfrac{h^3}{6}[bf^{(3)}(\xi_1) + c\lambda^3 f^{(3)}(\xi_2)] \end{aligned}$$

- Obtain the coefficients by solving:

$$\begin{aligned} a+b+c &= 0 \\ b+c\lambda &= 1/h \\ b+c\lambda^2 &= 0 \end{aligned}$$

- Error: $O(h^2)$

Richardson extrapolation

- In some approximation procedures, one first decides a step size h and then generates an approximation $A(h)$ to some desired quantity A.
- Idea: Generate a better approximation (i.e., with a higher order error) from multiple lower order approximations.
- See *Collard's lecture note 4* for detail.

Further Readings

- *Judd, Chapter 7*
- *Collard, Lecture Notes 4*
- *Miranda & Fackler, Chapter 5*

Lab

Exercise

Aim. Understand the importance of round-off errors arising from floating point numbers. Compare the accuracy between different methods

- Let $f(x) = \sin x$
- Then, $f'(x) = \cos x$ and $f'(0.5) \approx 0.8775825619$

Task.

- Approximate $f'(0.5)$ by forward and central differences
- Try $h = 10^{-k}$ for $k = 1, 2, \ldots, 20$
- Plot $\log_{10}$(total errors) by the two methods in the same figure, with $-\log_{10}(h)$ in the horizontal axis. Submit your plot.

Assignment 5

Setting: Quasi-hyperbolic discounting structure

- An application to a simple structural model
- Time preferences play important roles for various dynamic decisions.
- Not only discount factor, but also present biasness matters.
  e.g.) I do not want to do my homework just now, so I allocate much more study time to tomorrow. The ratio of study time between today and a future day can differ from the planned ratio between two future days with an equal interval.
- An individual at period t maximizes lifetime utility:

$$U(c) = u(c_t) + \beta \sum_{k=1}^{\infty} \delta^k u(c_{t+k})$$

- Read Andreoni and Sprenger (2012 AER), Augenblick et al. (2015 QJE), and Casaburi and Macchiavello (2019 AER) if you are interested, but not necessary for this assignment.

Experiment & Data Generating Process

- A researcher conducts a lab experiment in India for obtaining time preference parameters.
- Subjects are asked to choose two-period intertemporal allocations of money (Rs. 4000) within a convex budget set, with various t (earlier date), k (time interval between th earlier and later dates), and several interest rates $P = (1 + r)$.
- Assume that the subjects solve:

$$U(c_t, c_{t+k}) = c_t^{\alpha} + \beta^{\mathbb{1}_{t=0}} \delta^k c_{t+k}^{\alpha}$$
$$\text{s.t. } P c_t + c_{t+k} = 4000$$

- Solving this,

$$c_t = \frac{4000}{(\beta^{\mathbb{1}_{t=0}} \delta^k P)^{1/(1-\alpha)} + P} \equiv g(\mathbb{1}_{t=0}, k, P; \beta, \delta, \alpha)$$

- Parameters: β: present biasness, δ: discount factor, α: curvature ($IES = 1/(1-\alpha)$).
- Distributed data: $w_{i,q} \equiv \{c_{i,t_q}, c_{i,t_q+k_q}, t_q, k_q, P_q\}$ (i: individual, q: question)

Assignment

Aim. Experience that numerical differentiation might affect a researcher's conclusion significantly.

- In the previous assignment, you have estimated the parameters $\hat{\theta} = (\hat{\beta}, \hat{\delta}, \hat{\alpha})$ by non-linear least squares (NLLS), as an M-estimator, a class of extremum estimators:

$$\max_{\beta,\delta,\alpha} \sum_{i,q} \left\{ - \left[c_{i,t_q} - g(\mathbb{1}_{t_q=0}, k_q, P_q; \beta, \delta, \alpha) \right]^2 \right\} \equiv \max_{\theta} \sum_{i,q} m(w_{i,q}; \theta)$$

Task. Given $w_{i,q}$ & $\hat{\theta}$ (that we estimated), numerically compute standard errors and 95% confidence intervals of $\hat{\theta}$.

- For computing $s(w_{i,q}; \theta)$, the score function, analytically derive it and substitute parameter estimates and data into it.
- Obtain $H(w_{i,q}; \theta)$, Hessian of $m(w_{i,q}; \theta)$, by the following ways:
  1. Directly apply 2nd-order numerical differentiations to $m(w_{i,q}; \theta)$.
  2. Apply 1st-order numerical differentiations to $s(w_{i,q}; \theta)$, i.e., numerically compute the Jacobian of $s(w_{i,q}; \theta)$.
  3. Analytically derive the consistent estimate of $\mathbb{E}[H(w_{i,q}; \theta_0)]$ and substitute parameter estimates and data into it

Hints

- For numerical differentiation, use a hand-made central difference method or numdifftools.
- For analytical derivations, do them by hand or by SymPy.
- Assume that conditions in Propositions 7.3 (or 7.4) and 7.8 in the Hayashi textbook are satisfied.
- Therefore, use the result of asymptotic normality of M-estimators and the consistent asymptotic variance estimation:

$$\widehat{Avar(\hat{\theta})} = \left\{\frac{1}{N}\sum_{i,q} H(w_{i,q};\hat{\theta})\right\}^{-1} \hat{\Sigma} \left\{\frac{1}{N}\sum_{i,q} H(w_{i,q};\hat{\theta})\right\}^{-1}$$

where $\hat{\Sigma} \equiv \frac{1}{N}\sum_{i,q} s(w_{i,q};\hat{\theta}) s(w_{i,q};\hat{\theta})'$ (N: total observation),
$s(w_{i,q};\theta) \equiv \frac{\partial m(w_{i,q};\theta)}{\partial \theta}$, $H(w_{i,q};\theta) \equiv \frac{\partial^2 m(w_{i,q};\theta)}{\partial \theta \partial \theta'}$

Review also Chap 7 of Hayashi's textbook or the handbook chapter Newey & McFadden (1994)

Numerical Integration

Quadrature Problem: Big Picture

- Compute $\int_D f(x)dx$ where $f : \mathbb{R}^n \to \mathbb{R}$ is an integrable function over the domain $D \subset \mathbb{R}$.
- However, most integrals cannot be evaluated analytically.

Quadrature Problem: Big Picture

- Compute $\int_D f(x)dx$ where $f : \mathbb{R}^n \to \mathbb{R}$ is an integrable function over the domain $D \subset \mathbb{R}$.
- However, most integrals cannot be evaluated analytically.
- Use a finite number of evaluations of the integrand f and a weighted sum of those values to approximate: $\int_D f(x)dx \approx \sum_{n \in N} w(n) f(x_n)$.
- This is necessary because it is infeasible to evaluate $f(x)$ for all $x \in D$; $|D| = \infty$.

Quadrature Problem: Big Picture

- Compute $\int_D f(x)dx$ where $f : \mathbb{R}^n \to \mathbb{R}$ is an integrable function over the domain $D \subset \mathbb{R}$.
- However, most integrals cannot be evaluated analytically.
- Use a finite number of evaluations of the integrand f and a weighted sum of those values to approximate: $\int_D f(x)dx \approx \sum_{n \in N} w(n) f(x_n)$.
- This is necessary because it is infeasible to evaluate $f(x)$ for all $x \in D$; $|D| = \infty$.
- Select the efficient method among several alternatives for improving running times and keeping accuracy.
- Methods differ in how to choose:
  - N: nodes at which the integrand is evaluated
  - $w(n)$: weight assigned to each function evaluation

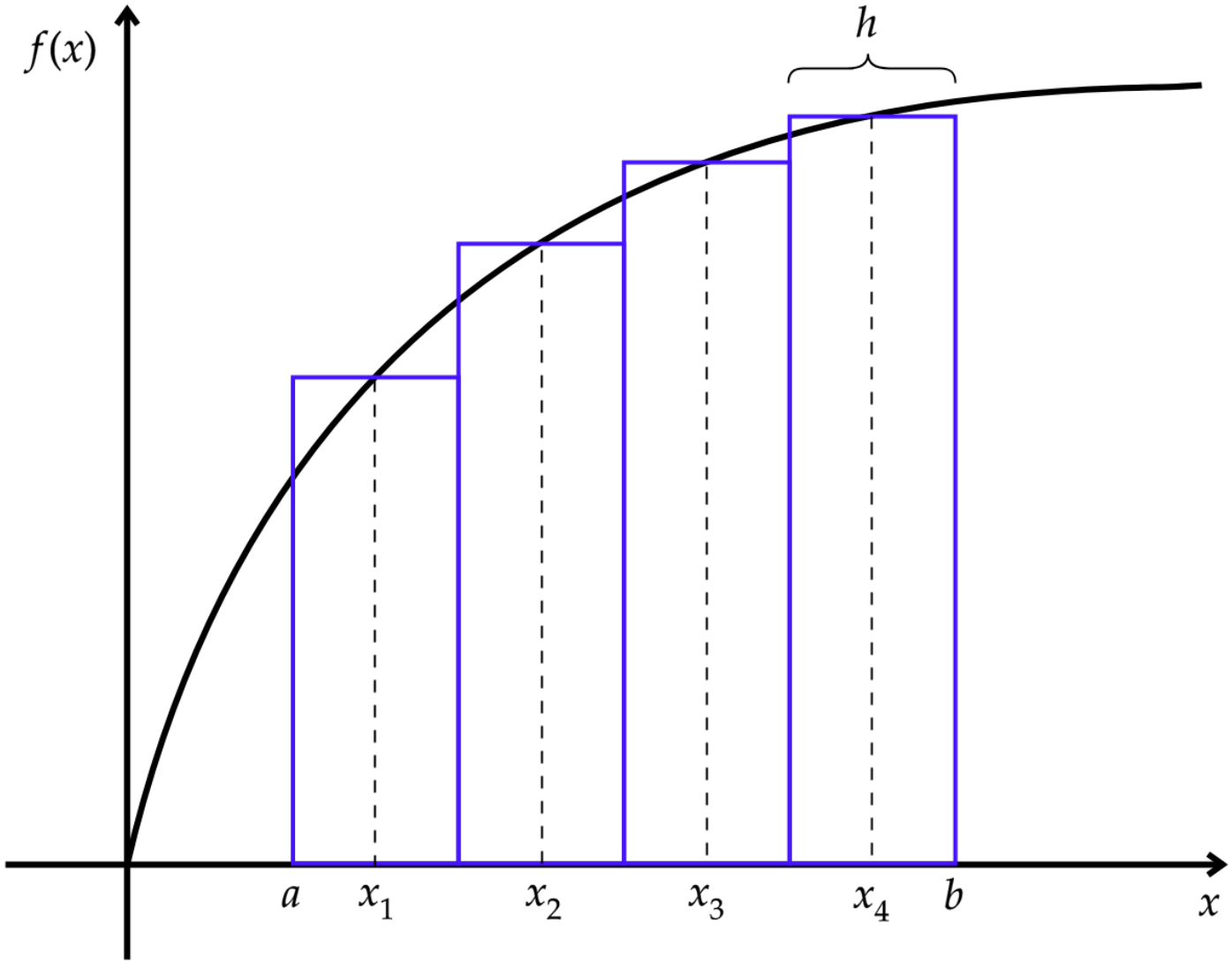
Overview

- Newton-Cotes formulas
  - Mid-point rule
  - Trapezoid rule
  - Simpson rule
- Brief intro to interpolation methods
- Adaptive quadrature
- Infinite integration domain
- Gaussian formulas
- Multidimensional quadrature
- Monte Carlo integration
- SciPy integration package (tutorial)

Newton-Cotes formulas: Overview

- $\{x_i\}_{i=1}^n$: a partition of $[a, b] \subset \mathbb{R}$
- $\int_a^b f(x)dx = \lim_{n\to\infty} \sum_{i=1}^{n-1} f(\xi_i)(x_{i+1} - x_i)$
  where $\xi_i \in [x_i, x_{i+1}]$
- General workflow of Newton-Cotes formulas:
  - Split the interval into small subintervals
  - Approximate f by a polynomial on each subinterval
  - Integrate this polynomial rather than f
  - Add together the contributions from each subinterval

Newton-Cotes Formula 1: Midpoint Rule



Midpoint Rule

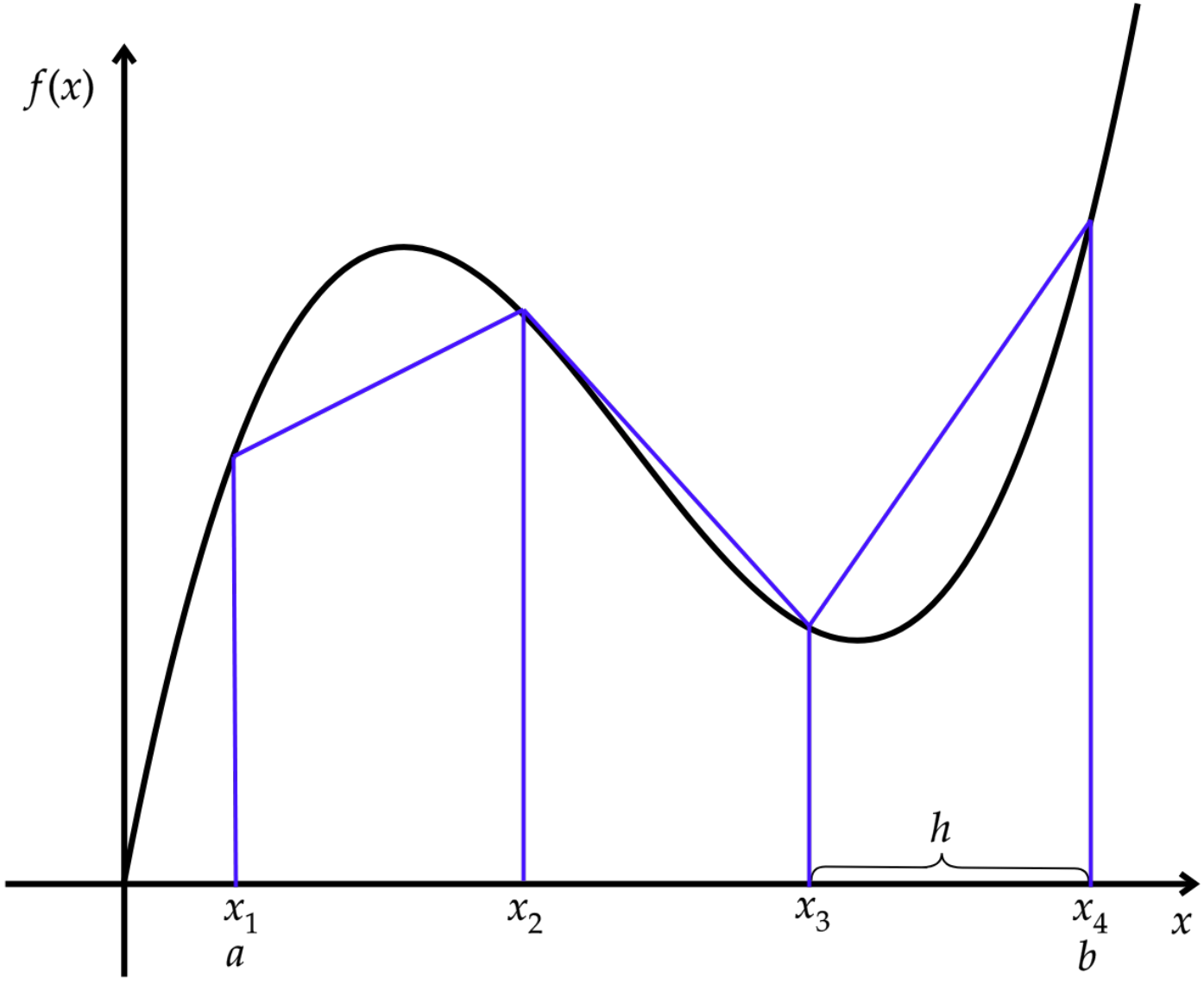
- The simplest method: just interpolate the constant function value at the middle point on each interval

$$\int_a^b f(x)dx = h\sum_{i=1}^{n} f(x_i) + \text{Error}$$

where $h \equiv \frac{b-a}{n}$ & $x_i \equiv a + (i - \frac{1}{2})h$

- $|\text{Error}| \leq (b-a)\frac{h^2}{24}M$ where $M \equiv \max_{x\in[a,b]} |f''(x)|$
- Quadratic convergence for $f \in C^2$:
  Halve the interval width $\Longrightarrow$ Reduce the error by $\approx$75%

Newton-Cotes Formula 2: Trapezoid Rule



Trapezoid Rule

- Approximte f on each interval with the secant that interpolates f at both ends of the interval

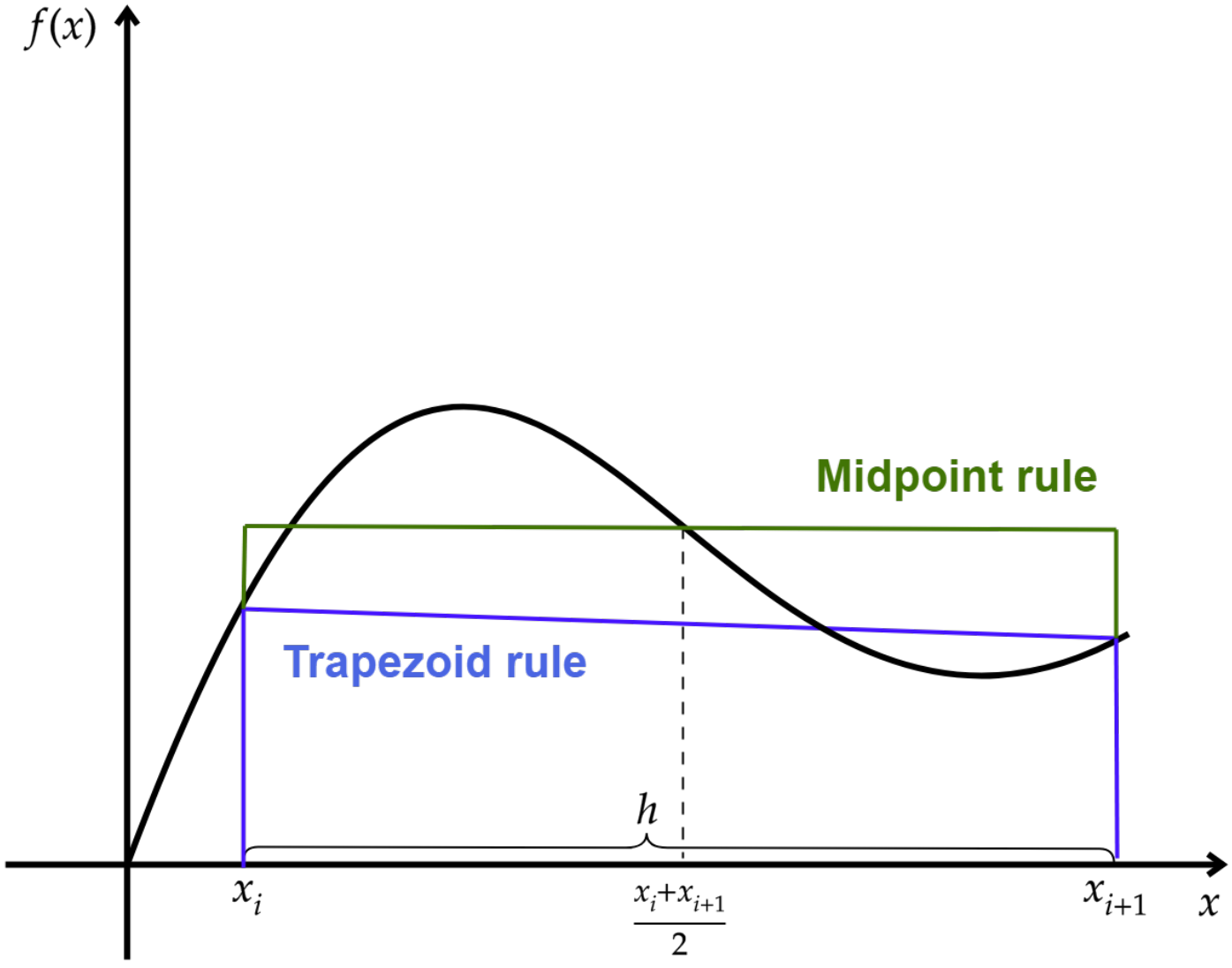
$$\begin{aligned}\int_a^b f(x)dx &= h\sum_{i=1}^{n}\frac{f(x_i)+f(x_{i+1})}{2} + \text{Error} \\ &= h\left(\frac{f(a)+f(b)}{2} + \sum_{i=1}^{n-1} f(x_{i+1})\right) + \text{Error}\end{aligned}$$

  where $h \equiv \frac{b-a}{n}$ & $x_i \equiv a + (i-1)h$

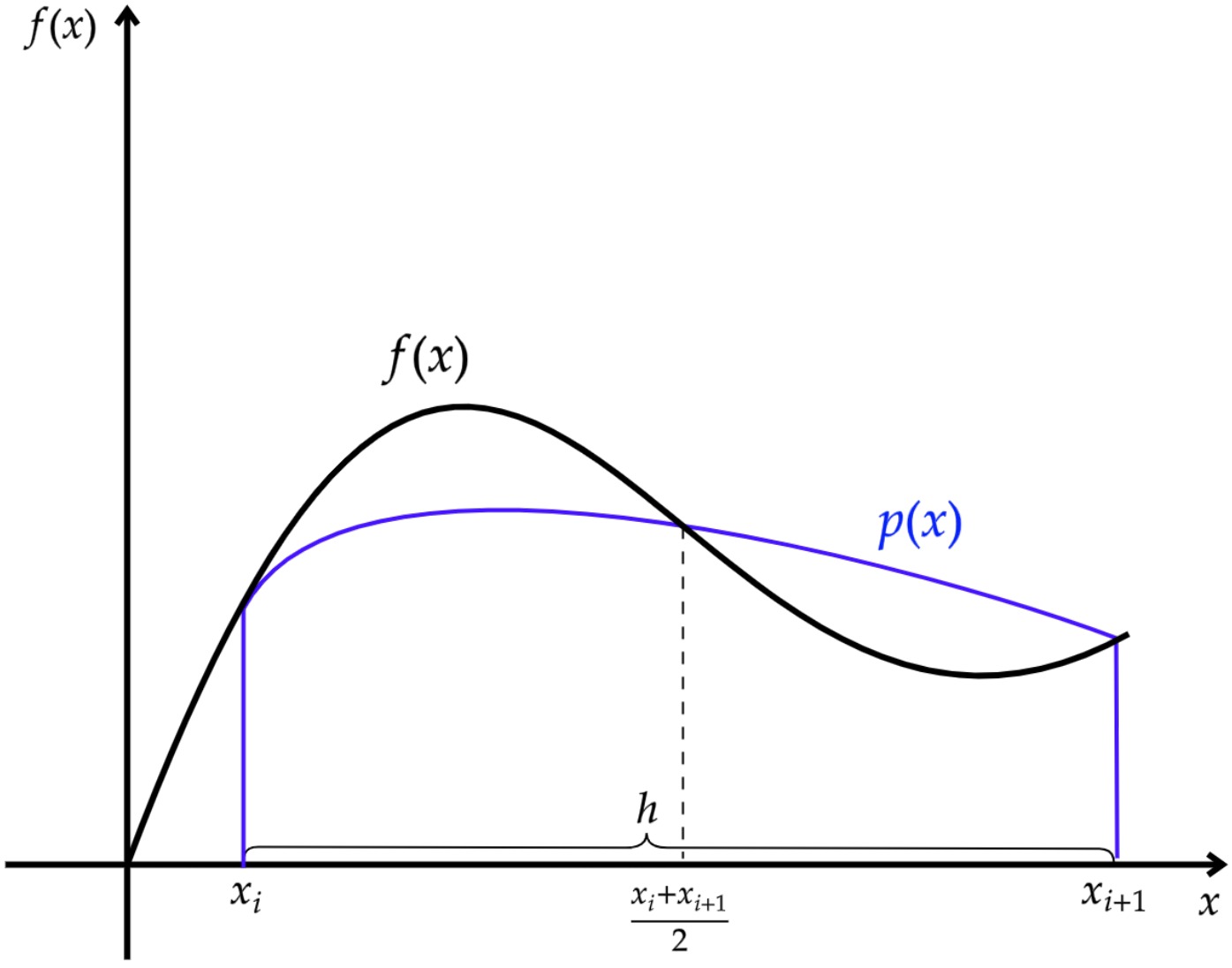
- $|\text{Error}| \leq (b-a)\frac{h^2}{6}M$ where $M \equiv \max_{x\in[a,b]} |f''(x)|$
  (Derive this by yourself!)

- Gained nothing (at the maximum errors) by approximating f by a linear function instead of a constant!
  i.e., Despite more functional evaluations than the midpoint rule, there is no accuracy gain!

Too Coarse with Whichever Rule



Newton-Cotes Formula 3: Simpson's Rule



Simpson's Rule: Motivation

- Circumvent the inefficiencies of the midpoint/trapezoid rules
- Use a piecewise quadratic interpolant of f which uses the values of f at x_i, x_{i+1}, and $\frac{x_i+x_{i+1}}{2}$ for $i = 1, \ldots, n$
- Need **interpolation**, one type of function approximation problems

Function Approximation Methods

- In many situations, we need to approximate functions because in many cases:
  - Computing values of a function at all points is not possible (as we saw just now!)
  - Functional forms are unknown but only a few points are observed
- In both cases, compute or use values of a function at only a few points and guess its values elsewhere
- **Interpolation**: any procedure that finds a "nice" function that goes through a collection of prescribed points
- The simplest one is linear interpolation, which we've already used in the trapezoid rule
- For other approximation methods, read *Judd, Chap 6*

Lagrange Interpolation

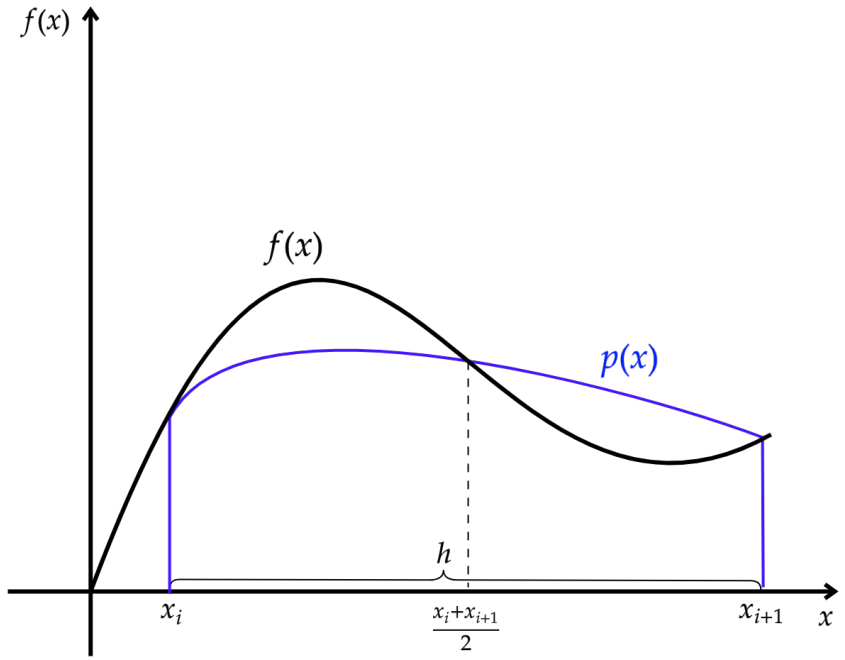
- Take a collection of n points in $\mathbb{R}^2$, $D = \{(x_i, y_i) | i = 1, \ldots, n\}$ (data)
- Then, find a degree $n-1$ polynomial, $p(x)$ s.t. $y_i = p(x_i)$, $i = 1, \ldots, n$
- Define: $l_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$
- Notice: $l_i(x) = 1$ if $x = x_i$ & $l_i(x) = 0$ if $x = x_j$ for $j \neq i$
- Therefore, $p(x) = \sum_{i=1}^{n} y_i l_i(x)$ interpolates the data, i.e., $y_i = p(x_i) \, \forall i$
- In the case of Simpson's rule, $n = 3$ and thus this becomes a quadratic interpolant
- For other interpolation methods, read *Judd, Chap 6*

Simpson's Rule on $[x_i, x_{i+1}]$

- Using the Lagrange interpolation,

$$\int_{x_i}^{x_{i+1}} f(x)dx = \left(\frac{x_{i+1} - x_i}{6}\right)\left[f(x_i) + 4f\left(\frac{x_i + x_{i+1}}{2}\right) + f(x_{i+1})\right] + \text{Error}$$

- $|\text{Error}| \leq \frac{(x_{i+1}-x_i)^5}{2880} \max_{x\in[x_i,x_{i+1}]} f^{(4)}(x)$



Simpson's Rule with Intervals

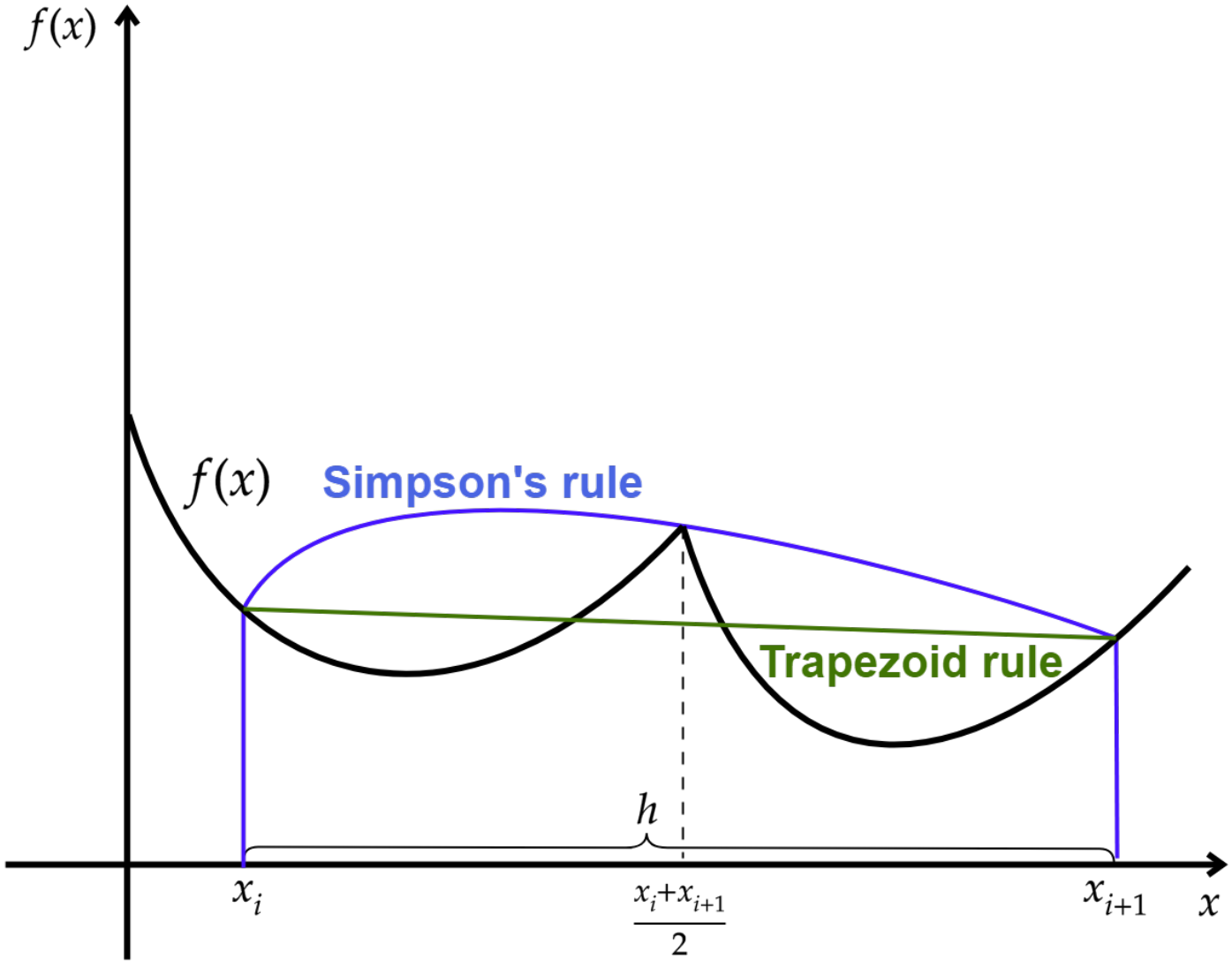
- n: an even number of intervals
- Using the Lagrange interpolation and the previous result,

$$\int_a^b f(x)dx = \frac{h}{3}\left[f(x_1) + 4f(x_2) + 2f(x_3) + 4f(x_4) + \cdots + 4f(x_n) + f(x_{n+1})\right] + \text{Error}$$

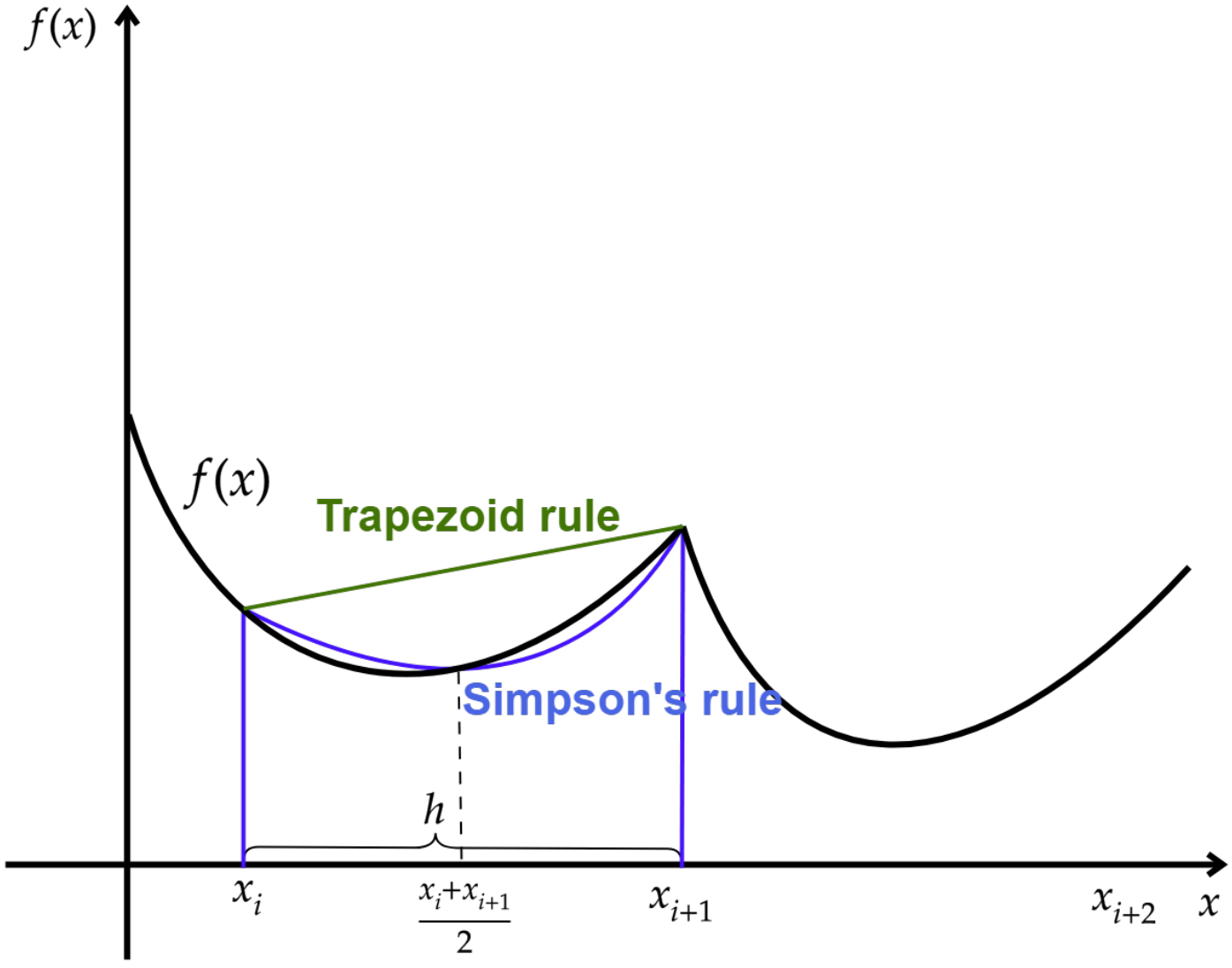
where $h \equiv \frac{b-a}{n}$ & $x_i \equiv a + (i-1)h$

- $|\text{Error}| \leq \frac{h^4(b-a)}{180} \max_{x\in[a,b]} f^{(4)}(x)$ (4th-order convergence)
- Halve the interval width $\Longrightarrow$ Reduce the error by $\approx$93.75%
- With this asymptotically smaller error, Simpson's rule is a very popular method

Choice of h Matters A Lot!



Choice of h Matters A Lot!



Adaptive Quadrature

- Truncation error matters, while numerical integration is very insensitive to round-off errors
- But, too small h has a computational burden

Adaptive Quadrature

- Truncation error matters, while numerical integration is very insensitive to round-off errors
- But, too small h has a computational burden
- Idea: increase the number of nodes until the sequence of estimates of the integral converges.
- Recommended especially when the functional shape is unclear

Adaptive Quadrature

- Truncation error matters, while numerical integration is very insensitive to round-off errors
- But, too small h has a computational burden
- Idea: increase the number of nodes until the sequence of estimates of the integral converges.
- Recommended especially when the functional shape is unclear
- One simple way: double the number of intervals with each operation.

Adaptive Quadrature

- Truncation error matters, while numerical integration is very insensitive to round-off errors
- But, too small h has a computational burden
- Idea: increase the number of nodes until the sequence of estimates of the integral converges.
- Recommended especially when the functional shape is unclear
- One simple way: double the number of intervals with each operation.
- More sophisticated way: concentrate new evaluation points in those areas where the integrand appears to be most irregular.

Infinite Integration Domains

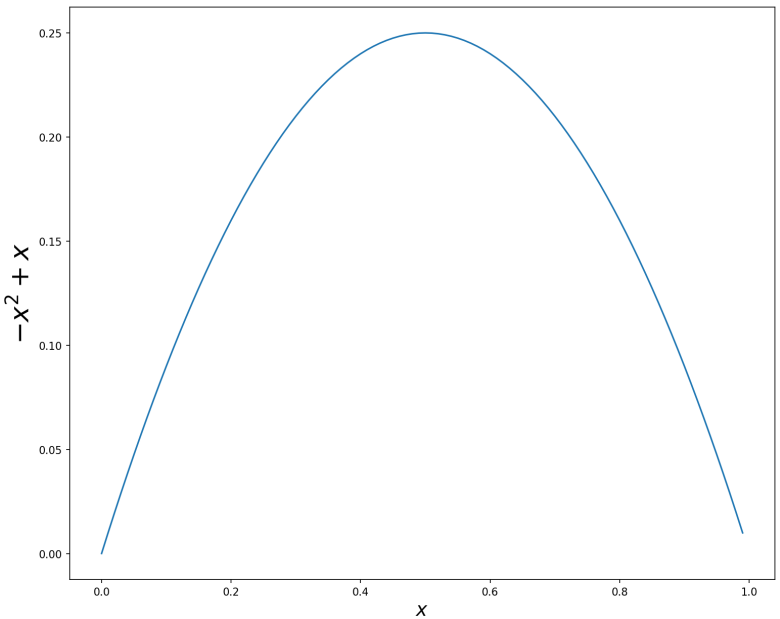
- How to approximate $\int_0^\infty f(x)dx$?
  (Restrict to the cases where $\int_0^\infty f(x)dx$ exists)
- $\int_0^\infty f(x)dx = \lim_{b\to\infty}\int_0^b f(x)dx \approx \int_0^b f(x)dx$ with a very large b is not a good idea: too time consuming
- Transform it to the finite integration domain by using:

$$\int_a^b f(x)dx = \int_{\phi^{-1}(a)}^{\phi^{-1}(b)} f(\phi(z))\phi'(z)dz$$

where $\phi : \mathbb{R} \to \mathbb{R}$, increasing, and C^1 on $[a, b]$

E.g.)
$$\int_0^\infty f(x)dx = \int_0^1 f\left(\frac{z}{1-z}\right)(1-z)^{-2}dz$$
$$\int_{-\infty}^\infty f(x)dx = \int_0^1 f\left(\ln\frac{z}{1-z}\right)[z(1-z)]^{-1}dz$$

Newton-Cotes Formulas: Example



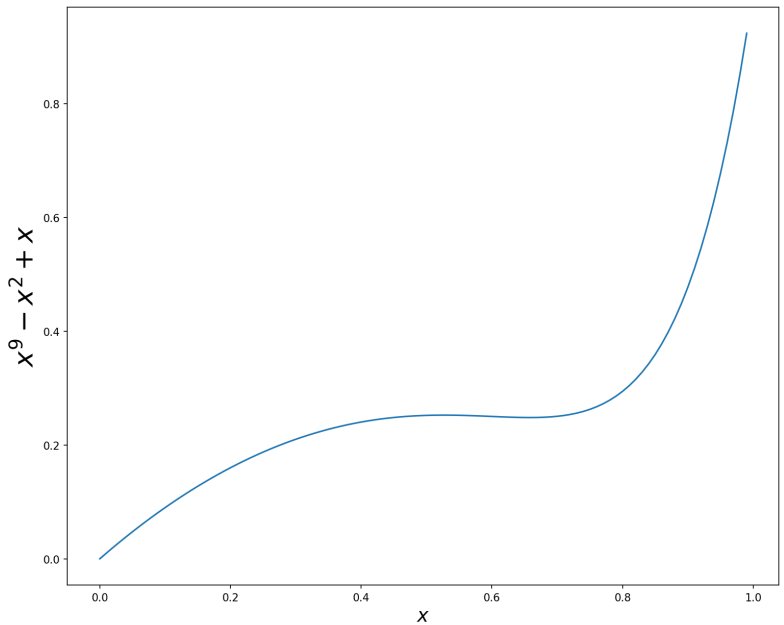


Table: Approximations which achieve errors $<10^{-6}$

Object	$\int_0^1 (-x^2 + x)dx$			$\int_0^1 (x^9 - x^2 + x)dx$		
Method	Midpoint	Trapezoid	Simpson's	Midpoint	Trapezoid	Simpson's
n	289	410	2	541	764	42
Computation time (sec.)	0.000271	6.13E-05	0.000122	0.000684	0.000138	0.00012

Available SciPy functions: `scipy.integrate.trapz` & `scipy.integrate.simps`

Gaussian Quadrature: Motivation

- (Recall) Newton-Cotes formulas: $\int_a^b f(x)dx \approx \sum_{i=1}^n \omega_i f(x_i)$ for some *arbitrary* nodes $\{x_i\} \in [a, b]$ and weights $\{\omega_i\}$

  i.e., N-C formulas attempt to approximate the given function directly on subintervals using polynomials

- Are there more *efficient* choices of nodes and weights?

Gaussian Quadrature: Motivation

- (Recall) Newton-Cotes formulas: $\int_a^b f(x)dx \approx \sum_{i=1}^n \omega_i f(x_i)$ for some *arbitrary* nodes $\{x_i\} \in [a,b]$ and weights $\{\omega_i\}$

  i.e., N-C formulas attempt to approximate the given function directly on subintervals using polynomials

- Are there more *efficient* choices of nodes and weights?
- Idea: Gaussian approach finds nodes $\{x_i\}$ and weights $\{\omega_i\}$ to achieve the *better* approximation
- Given a nonnegative weighting function $w(x)$, Gaussian quadrature computes the following approximation:

$$\int_a^b f(x)w(x)dx \approx \sum_{i=1}^n \omega_i f(x_i)$$

  for some nodes $x_i \in [a,b]$ and positive weights ω_i

Gaussian Quadrature: Intuition

- *Exact integration* for a finite-dimensional collection of functions: Choose weights and nodes such that the approximation is exactly correct if f is a polynomial of the given order
- Gaussian quadrature accomplishes this for spaces of degree $2n-1$ polynomials using n nodes and n weights:

  Given a nonnegative weighting function $w(x)$, we can find n points $\{x_i\}_{i=1}^n \subset [a,b]$ and n nonnegative weights $\{\omega_i\}_{i=1}^n$ s.t.

$$\int_a^b f(x)w(x)dx = \sum_{i=1}^n \omega_i f(x_i) \quad \forall\, f \in \mathcal{F}_{2n-1}$$

Theorem

Suppose

1. $\{\varphi_k(x)\}_{k=0}^{\infty}$: an orthonormal family of polynomials w.r.t. $w(x)$ on [a,b]
2. q_k: s.t. $\varphi_k(x) = q_k x^k + \cdots$
3. $x_i, i = 1, \ldots, n$: n roots of $\varphi_n(x)$ s.t. $x_1 < x_2 \cdots < x_n$
4. $\omega_i = -\frac{q_{n+1}/q_n}{\varphi_n'(x_i)\varphi_{n+1}(x_i)}$

Then

(i) $a < x_1 < x_2 \cdots < x_n < b$

(ii) If f is $C^{(2n)}$ on $[a,b]$, then
$\int_a^b f(x)w(x)dx = \sum_{i=1}^n \omega_i f(x_i) + \frac{f^{(2n)}(\xi)}{q_n^2(2n)!} \quad \exists\, \xi \in [a,b]$

(iii) $\int_a^b f(x)w(x)dx = \sum_{i=1}^n \omega_i f(x_i) \quad \forall\, f \in \mathcal{F}_{2n-1}$

Gaussian Quadrature: Implementation

- The theorem tells us how to compute the necessary nodes $\{x_i\}$ and weights $\{\omega_i\}$

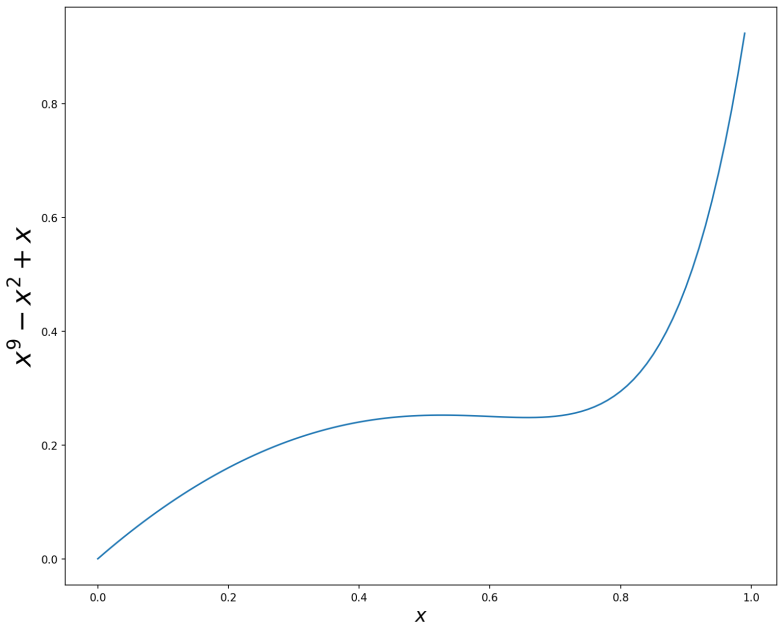
  SciPy integration package contains modules computing them.

- Or, follow some specific Gaussian formulas:

Formula	Domain	Weight
Gauss-Chebyshev	$[-1, 1]$	$w(x) = (1 - x^2)^{-\frac{1}{2}}$
Gauss-Legendre	$[-1, 1]$	$w(x) = 1$
Gauss-Hermite	$[-\infty, \infty]$	$w(x) = e^{-x^2}$
Gauss-Laguerre	$[0, \infty]$	$w(x) = e^{-x}$

Their nodes and weights are available in tables (e.g., in *Judd, Section 7.2*), in online data files (e.g., this), and in standard software packages and libraries (e.g., `scipy.special`).

Gaussian Quadrature: E.g. with `scipy.integrate.fixed_quad`



Again, compute $\int_0^1 (x^{\mathbf{9}} - x^2 + x)dx$

n	2n-1	Approximation	Error	Computation time (sec.)
1	1	0.2519531	0.014714	0.000693
2	3	0.2256944	0.040972	0.00105
3	5	0.2622292	0.004438	0.000316
4	7	0.2665646	0.000102	0.000597
5	**9**	0.2666667	1.11E-16	0.000327

Recall: 541, 764, and 42 draws were needed to achieve errors $< 10^{-6}$ for the same function by midpoint, trapezoid, and Simpson's rules, respectively!

Gauss-Chebyshev

- Formula:

$$\int_{-1}^{1} f(x)(1-x^2)^{-\frac{1}{2}}\,dx = \frac{\pi}{n}\sum_{i=1}^{n} f(x_i) + \frac{\pi}{2^{2n-1}}\frac{f^{(2n)}(\xi)}{(2n)!}$$

  for some $\xi \in [-1,1]$ where $x_i = \cos\left(\frac{2i-1}{2n}\pi\right)$

- Letting $y \equiv \frac{2(x-a)}{b-a} - 1$,

$$\int_{a}^{b} f(x)dx = \frac{b-a}{2}\int_{-1}^{1} g(y)(1-y^2)^{-\frac{1}{2}}\,dy$$

  where $g(y) \equiv f\left(a + \frac{(y+1)(b-a)}{2}\right)(1-y^2)^{\frac{1}{2}}$

Gauss-Legendre

- Formula:

$$\int_{-1}^{1} f(x)dx = \sum_{i=1}^{n} \omega_i f(x_i) + \frac{2^{2n+1}(n!)^4}{(2n+1)!(2n)!} \frac{f^{(2n)}(\xi)}{(2n)!}$$

  for some $\xi \in [-1, 1]$

- Uses a similar way of domain conversion from $[a, b]$
- Exponential convergence to the true value

Gauss-Hermite

- Formula:

$$\int_{-\infty}^{\infty} f(x)e^{-x^2}dx = \sum_{i=1}^{n} \omega_i f(x_i) + \frac{n!\sqrt{\pi}}{2^n}\frac{f^{(2n)}(\xi)}{(2n)!}$$

  for some $\xi \in (-\infty, \infty)$

- Useful for many economics application because normally distributed random variables are often used.

- Let $x \sim N(\mu, \sigma^2)$ & $y \equiv \frac{x-\mu}{\sqrt{2}\sigma}$. Then,

$$\begin{aligned} E[f(x)] &= (2\pi\sigma^2)^{-\frac{1}{2}} \int_{-\infty}^{\infty} f(x)e^{-\frac{(x-\mu)^2}{2\sigma^2}}dx \\ &= \pi^{-\frac{1}{2}} \int_{-\infty}^{\infty} f\left(\sqrt{2}\sigma y + \mu\right)e^{-y^2}dy \end{aligned}$$

- $\ln(x) \sim N(\mu, \sigma^2)$ is also often used. In what situations?

Gauss-Laguerre

- Formula:

$$\int_0^\infty f(x)e^{-x}dx = \sum_{i=1}^{n} \omega_i f(x_i) + \frac{(n!)^2}{(2n+1)!(2n)!} \frac{f^{(2n)}(\xi)}{(2n)!}$$

  for some $\xi \in [0, \infty)$

- Useful for computing the discounted sum of payoffs in an infinite horizon problem

- E.g.)

$$\int_0^\infty e^{-\rho t} u(c(t))dt = \int_0^\infty e^{-y} u\left(c\left(\frac{y}{\rho}\right)\right)\frac{1}{\rho}dy \approx \frac{1}{\rho}\sum_{i=1}^{n} \omega_i f\left(\frac{y_i}{\rho}\right)$$

  where $y \equiv \rho t$ & $f \equiv u \circ c$

Multidimensional Quadrature

- One way: Product rule
- Approximate: $\int_{a_1}^{b_1} \cdots \int_{a_d}^{b_d} f(x_1, \ldots, x_d) dx_1 \cdots dx_d$
- by: $\sum_{i_1=1}^{n} \cdots \sum_{i_d=1}^{n} \omega_{i_1}^1 \cdots \omega_{i_d}^d f(x_{i_1}^1, \ldots, x_{i_d}^d)$
- Apply either Newton-Cotes or Gaussian formulas.
- Curse of dimensionality: with n nodes in each direction for a d-dimensional problem, n^d functional evaluations are needed.
- Alternatives for high dimensional problems:
  - Monte Carlo Integration
  - Sparse grids: Heiss and Winschel 2008 (web)

Monte Carlo (MC) Integration: Overview

- Based on the law of large number and the central limit theorem
- Any result is a random variable
- Put a structure on the error which has a probabilistic distribution
- Therefore, we need to present both the estimate of integral and the estimate of its variance or standard error
- Useful for high-dimensional problems
- Robust and simple

MC Integration: A Crude Way

- Draw a random sample $x_1, x_2, \ldots, x_n$ from the distribution whose density is $f(x)$ and approximate:

$$\mu_g \equiv Eg(x) = \int g(x)f(x)dx \approx \frac{1}{n}\sum_{i=1}^{n} g(x_i) \equiv \hat{\mu}_g$$

- Its variance:

$$\sigma^2_{\hat{\mu}_g} = \frac{1}{n}\int (g(x) - \mu_g)^2 dx = \frac{\sigma^2_g}{n}$$

where σ^2_g is estimated by:

$$\hat{\sigma}^2_g = \frac{1}{n-1}\sum_{i=1}^{n}(g(x_i) - \hat{\mu}_g)^2$$

Pseudo-Random Numbers

- MC methods rely on random numbers
- Random numbers cannot be generated by computers
- Instead, computers generate *pseudo-random* numbers that *look* random numbers
- All these numbers are generated with deterministic algorithms
- Advantage: no need to store the obtained random numbers and replication is easy by setting the same *seed*

Generate Pseudo-Random Numbers

- Most numerical software packages provide pseudo-random number geneators from uniform and normal distributions
- For others, use the inverse CDF method:

  For a CDF F and a $U \sim U(0,1)$, $X = F^{-1}(U)$ has the same CDF F
- To generate a pseudo-random sample $x_1, x_2, \ldots, x_n$ from the distribution F, generate a pseudo-random sample $u_1, u_2, \ldots, u_n$ from $U(0,1)$ and set $x_i = F^{-1}(u_i)$
- Check out `numpy.random`

MC Integration: Practical Techniques

- The crude way that we saw is unbiased
- But, there is a scope for reducing its variance (while retaining its unbiasedness)
- Several such techniques:
  - Stratified sampling
  - Importance sampling
  - Antithetic variates
  - Control variates
  - Quasi-Monte Carlo

e.g.) MC Integration vs. Gaussian

- Now, compute $\mathbb{E}[(x^{\mathbf{9}} - x^2 + x)]$ with $x \sim N(0, 0.01)$
- Gauss-Hermite formula is used: Use `scipy.special.roots_hermite` for obtaining Gauss-Hermite nodes and weghts.

Method	n	2n-1	Approximation	Computation time (sec.)
Gaussian	3	5	-0.0099999999999999998	0.001789
Gaussian	4	7	-0.0100000000000000002	0.002563
Gaussian	**5**	**9**	-0.01	0.001547
MC	10^2		-0.0018134	0.0029
MC	10^3		-0.0057573	0.0035
MC	10^4		-0.0089993	0.0176
MC	10^5		-0.0094415	0.117
MC	10^6		-0.0099246	1.346
MC	10^7		-0.0100042	7.515
MC	10^8		-0.010006	26.481

Heiss and Winschel (2008) Sparse Grids Integration

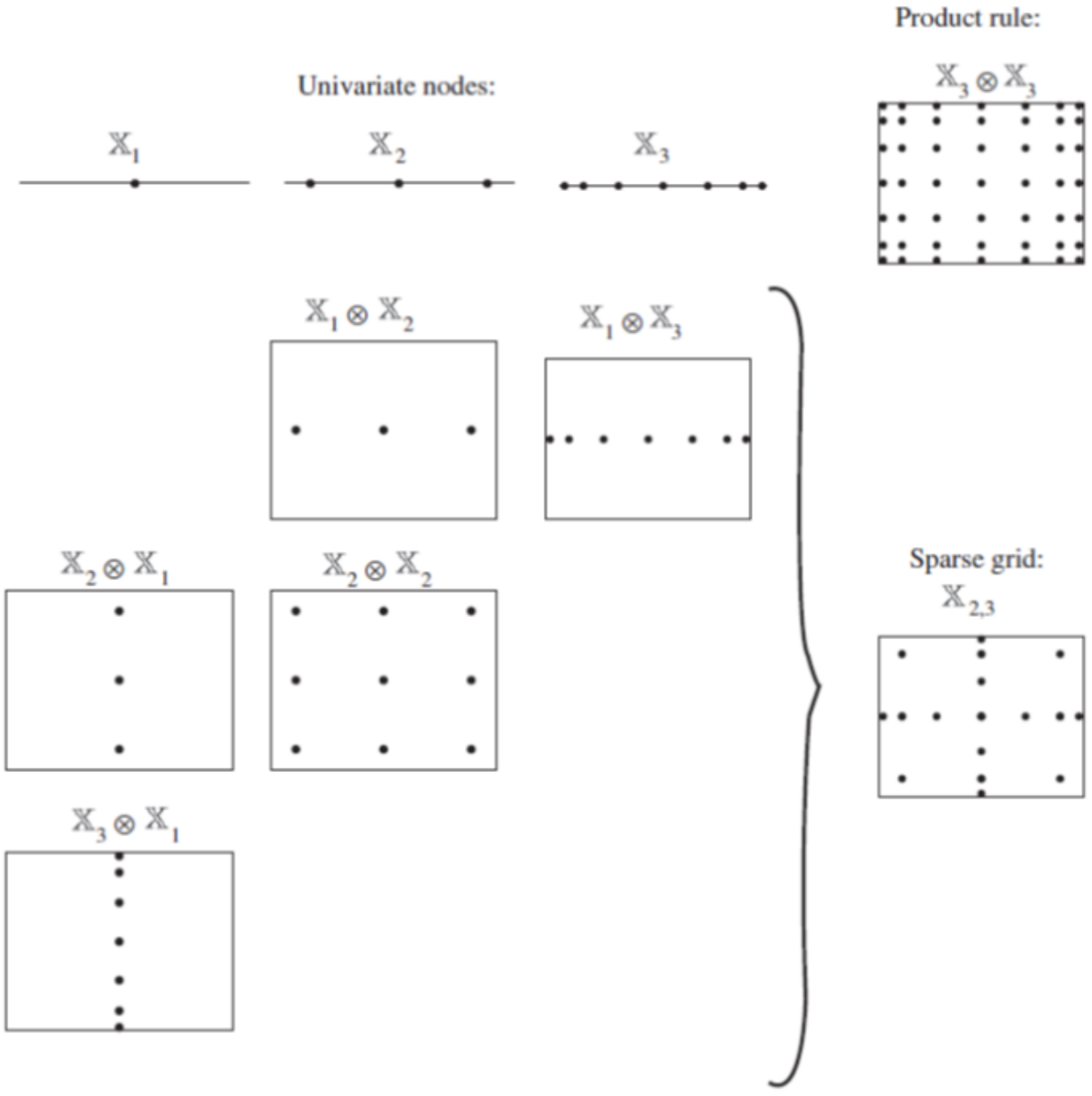


Fig. 1. Construction of the sparse grid in two dimensions.

Further Readings

- *Judd, Chapter 7*
- *Collard, Lecture Notes 4*
- Note by Skrainka and Judd

Lab

Exercise

Task 1.

- Numerically compute $\int_0^2 e^x dx$ by:

  (a) Midpoint, (b) Trapezoid, (c) Simpson's, (d) Gaussian, and (e) Monte Carlo (the crude way) (Set a seed by `np.random.seed(1)`)
- Feel free to use SciPy integration package for some methods
- Write a program for getting how many n we need by (a)–(c) and (e) to achieve the approximation error $< 10^{-6}$
- Compare the computation time of (d) to (a)-(c) and (e) (with the obtained n)

Task 2.

- Compute $\int_0^2 \int_0^2 e^x e^y dxdy$ (with a general-purpose function)
- Use (a) One of Newton-Cotes formulas (b) Monte Carlo (the crude way)
  (Set seeds by `np.random.seed(1)` for x & by `np.random.seed(2)` for y)
- Try $n = 10^k$ with $k = 1, \ldots, 7$
- Check errors and computation times

Assignment 6

Intertemporal Decision under Interest Rate Uncertainty

- Recall the setting and experiment in Assignment 4.
- Assume that the subjects now solve:

$$\begin{aligned} & \mathbb{E}U(c_t, c_{t+k}) = c_t^{\alpha} + \beta^{\mathbb{1}_{t=0}} \delta^k \mathbb{E}[c_{t+k}^{\alpha}] \\ \text{s.t.} \quad & \tilde{P} c_t + c_{t+k} = 4000 \\ & \tilde{P} = P + \epsilon,\ \epsilon \sim N(0, 0.01) \end{aligned}$$

- Now, there is an interest rate uncertainty.
- Euler equation:

$$\beta^{\mathbb{1}_{t=0}} \delta^k \mathbb{E}\left[\left(\frac{4000 - \tilde{P} c_t}{c_t} \right)^{(\alpha - 1)} \tilde{P} \right] - 1 = 0$$

- Parameters: β: present biasness, δ: discount factor, α: curvature ($IES = 1/(1-\alpha)$).
- Data: $w_{i,q} \equiv \{c_{i,t_q}, t_q, k_q, P_q\}$ (i: individual; q: question)

Note. *Use the* **newly distributed data**, *which is different from the data you have used in the previous assignments.*

Assignment

Aim. Get accustomed to several numerical intergation methods and experience computational burdens.

Task. Given $w_{i,q}$, obtain parameter values $\hat{\theta} = (\hat{\beta}, \hat{\delta}, \hat{\alpha})$ which minimize the following function:

$$Q_N(\theta) = \sum_{i,q} \left[\beta^{\mathbb{1}_{t_q=0}} \delta^{k_q} \mathbb{E}\left[\left(\frac{4000 - \tilde{P}_q c_{i,t_q}}{c_{i,t_q}} \right)^{(\alpha-1)} \tilde{P}_q \right] - 1 \right]^2$$

by searching over 18000 grid points:
$\beta \in [0.7, 1] \times \delta \in [0.8, 1] \times \alpha \in [0.5, 0.8]$ with interval size 0.01.

- For computing the expectation part, use the following methods:
  (a) **Monte Carlo** (the crude way): Try 100 and 1000 for the number of draws (with grid search). Also, try > 1000 (with Nelder-Mead).
  (b) **Gauss-Hermite**: Use `scipy.special.roots_hermite` for obtaining Gauss-Hermite nodes and weghts.
- Optional 1 Use **parallel processing** for your grid search process.
- Optional 2 Simulate a data using the obtained parameters and re-estimate the parameters with the simulated data.

Final Project

Final Project

- We have learnt a basic comprehensive set of scientific computation and numerical methods.
- Pick an economic model that interests you from what you have learnt in your first-year courses.
- Develop a library to robustlly do one or more of the following:
  - Solve the model
  - Simulate data from the model
  - Estimate the model
- Optional Develop unit tests to validate your library.
- Group work up to among 2-3 students is allowed. In this case, collaborate together sharing a same private repository.
- Please submit a one-page pdf proposal summarizing the model and other mathematical details (due will be announced in class).
  You should submit a PDF write-up of your summary and your codes via GitHub.