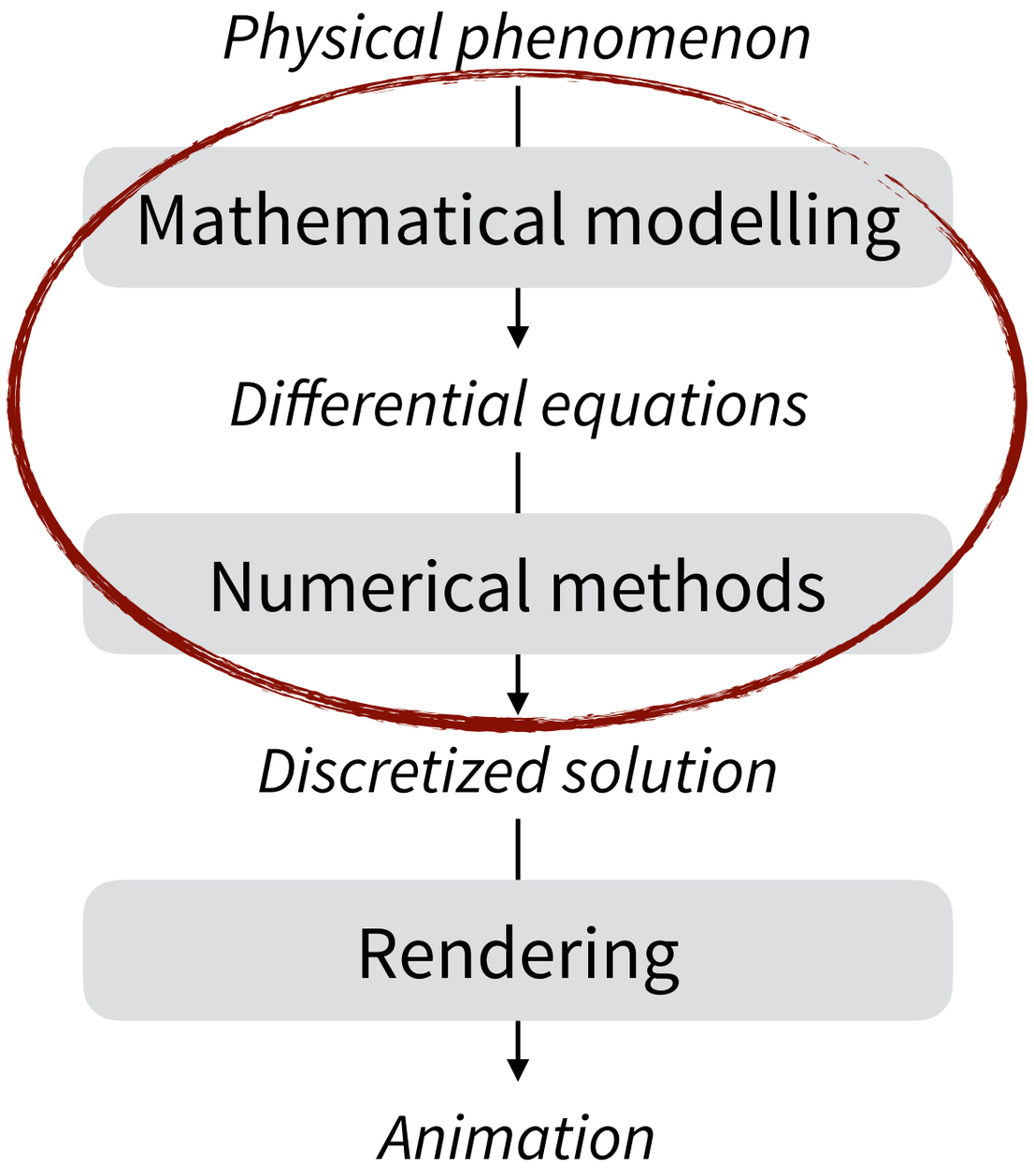
COL865: Special Topics in Computer Applications

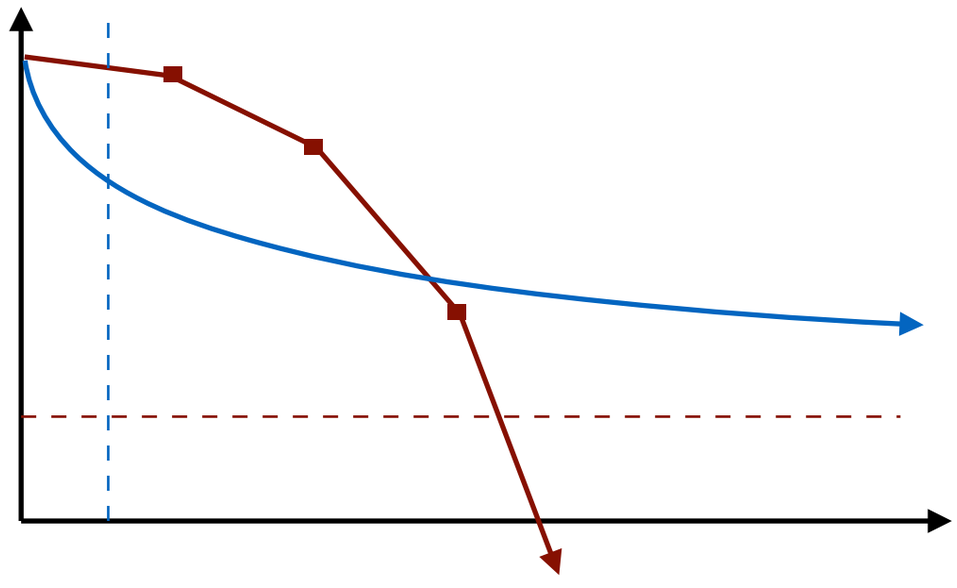
Physics-Based Animation

2 — Numerical analysis

Last class



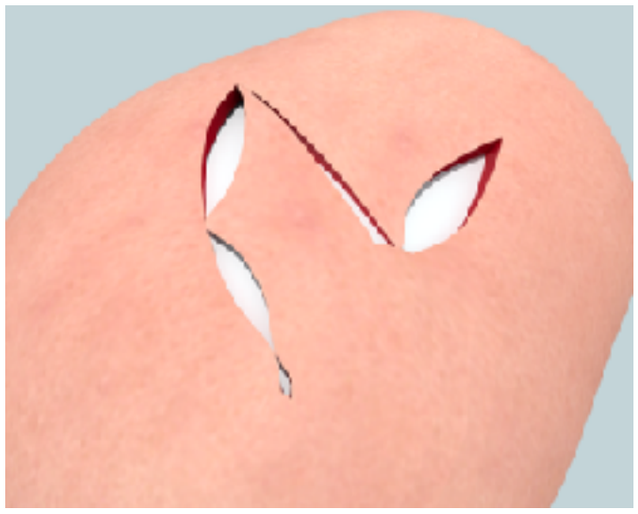
- Mathematical models and numerical methods for animating physical phenomena
- "Eyeball metric": prioritize errors **noticeable** to human viewers
- Need moderate accuracy, high speed, high stability



Emerging applications

- Physics-based robot motion planning
- Real-time virtual surgery
- Virtual clothing for e-commerce and design

[Cusumano-Towner et al. 2011]



[Mitchell et al. 2015]

[Avametric]

Policies

Grading breakdown:

- Presentations & discussions: 35%
  (presentation/discussion breakdown TBD)
- Assignments: 45% (3 × 15%)
- Final project: 20%

Late policy:

- 6 total free late days across 3 assignments. Use them wisely

Audit policy:

- C or better grade **and** at least 60% points in “presentations & discussions” component

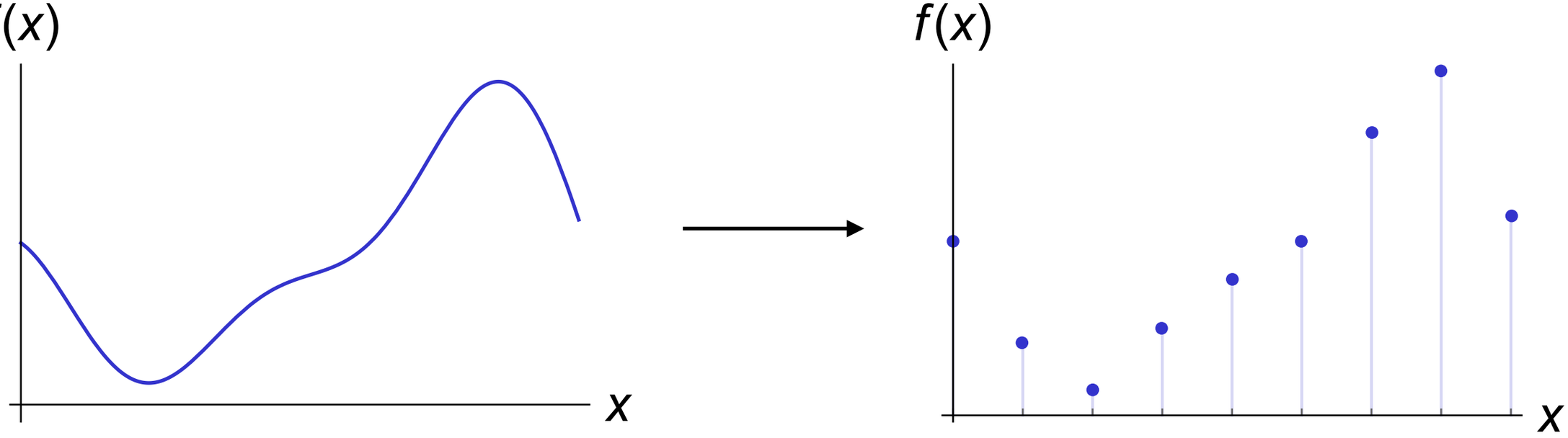
Numerical analysis

Numerical analysis

Mathematical models involve calculus on **continuous** functions

e.g. $$\mathrm{d}^2/\mathrm{d}t^2\, x(t) = f(t)/m$$

…but in computation, we can only work with **discrete** values



Consider a function $f : \mathbb{R} \to \mathbb{R}$, of which we only have finitely many samples $y_i = f(x_i)$. How to estimate $\mathrm{d}f/\mathrm{d}x$? $\int f(x)\,\mathrm{d}x$?

References

- **Burden and Faires, *Numerical Analysis*, Ch. 4.1, 4.3, 4.4**
- Solomon, *Numerical Algorithms*, Ch. 13, 14

Both posted on Moodle

(*Numerical Algorithms* is also free on Justin Solomon's webpage)

Interpolation

Basic approach in many numerical analysis methods:

Given n samples $y_1 = f(x_1), y_2 = f(x_2), \ldots, y_n = f(x_n)$,

- Construct a smooth function f that **interpolates** (passes through) all the samples:

$$\hat{f}(x_i) = y_i$$

- Then you can differentiate or integrate $\hat{f}$ as desired

Interpolation

Construct $\hat{f}$ as a linear combination of a set of **basis functions**:

$$\hat{f}(x) = a_1 b_1(x) + a_2 b_2(x) + \cdots + a_n b_n(x)$$

Solve for unknown coefficients so that $\hat{f}$ interpolates data

- Leads to an $n \times n$ system of linear equations

$$\begin{aligned} a_1 b_1(x_1) + a_2 b_2(x_1) + \cdots + a_n b_n(x_1) &= y_1 \\ a_1 b_1(x_2) + a_2 b_2(x_2) + \cdots + a_n b_n(x_2) &= y_2 \\ &\vdots \\ a_1 b_1(x_n) + a_2 b_2(x_n) + \cdots + a_n b_n(x_n) &= y_n \end{aligned}$$

Interpolation

Construct $\hat{f}$ as a linear combination of a set of **basis functions**:

$$\hat{f}(x) = a_1 b_1(x) + a_2 b_2(x) + \cdots + a_n b_n(x)$$

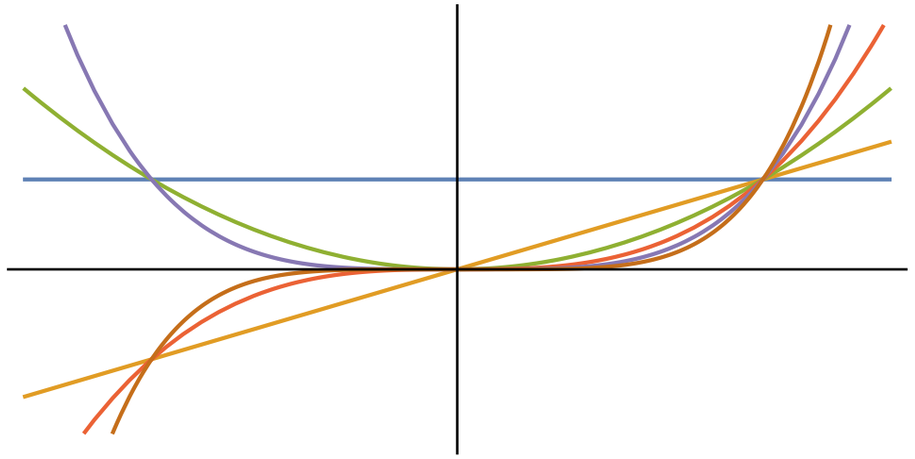
Solve for unknown coefficients so that $\hat{f}$ interpolates data

- Leads to an $n \times n$ system of linear equations

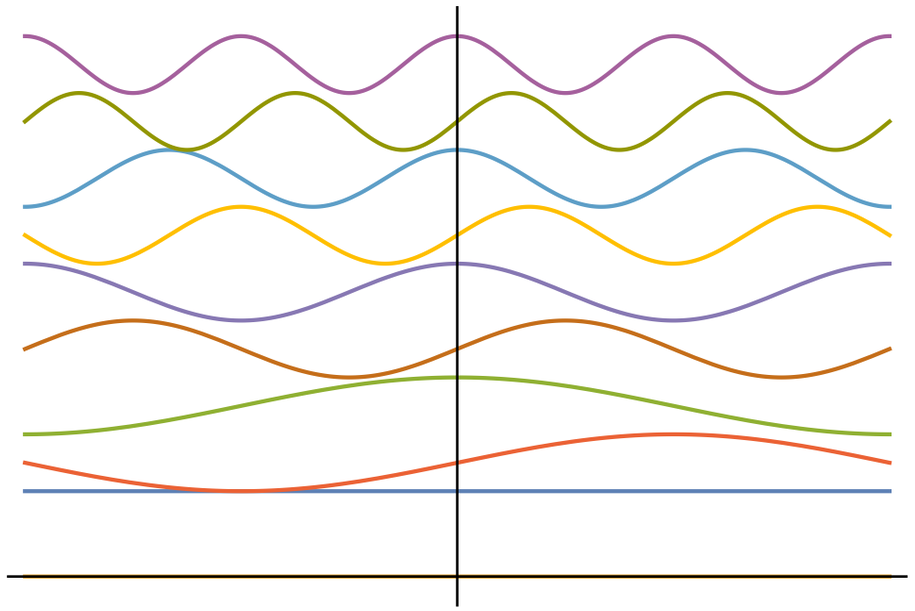
$$\begin{bmatrix} b_1(x_1) & b_2(x_1) & \cdots & b_n(x_1) \\ b_1(x_2) & b_2(x_2) & \cdots & b_n(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ b_1(x_n) & b_2(x_n) & \cdots & b_n(x_n) \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Interpolation

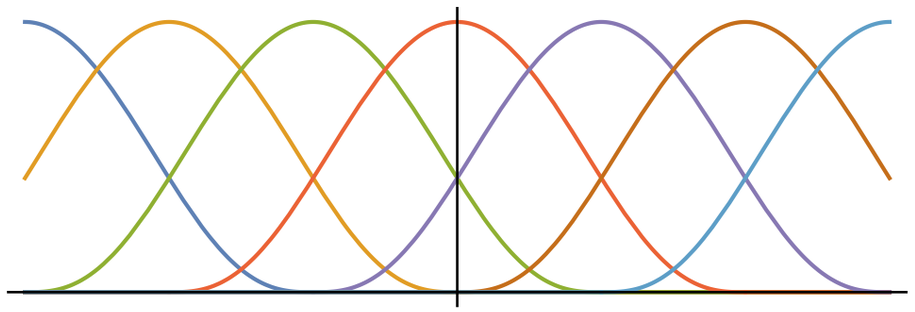
- **Polynomial interpolation**:
  basis functions are $1, x, x^2, \ldots, x^{n-1}$



- **Fourier basis**: sines and cosines
  (this is essentially what the
  discrete Fourier transform gives)

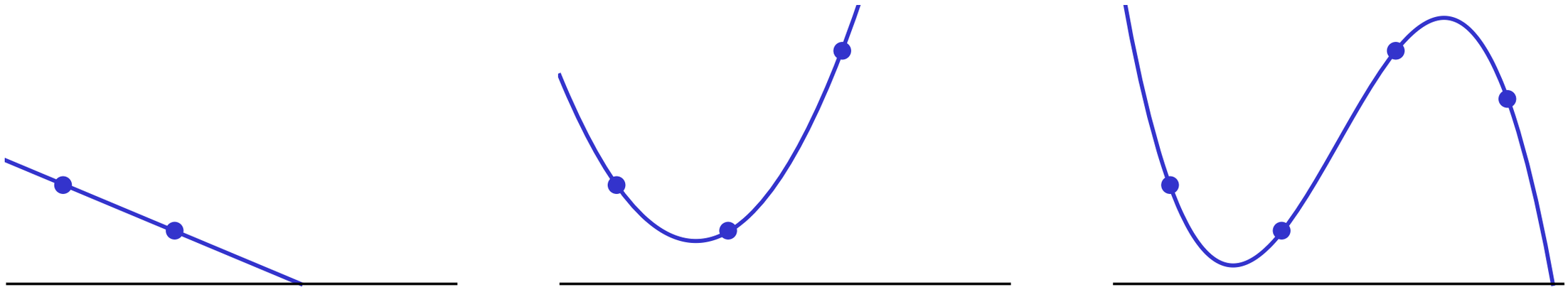


- **Radial basis functions**:
  copies of kernel fn. centered at x_i



Polynomial interpolation

Find a polynomial $p(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1}$ passing through n points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$



2 points = linear

3 points = quadratic

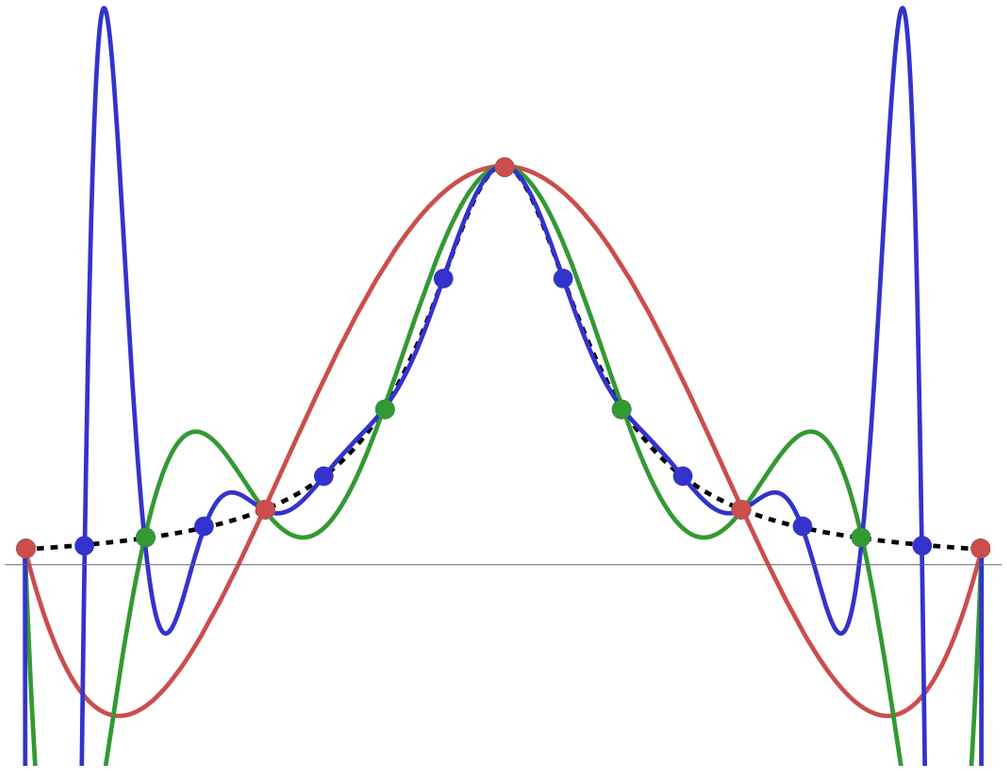
4 points = cubic

$1, x, x^2, \ldots, x^{n-1}$ is the **monomial basis**. Other bases (**Lagrange**, **Newton**) can be easier to solve, yield exactly same polynomial

Polynomial interpolation

When n is large, we have problems:

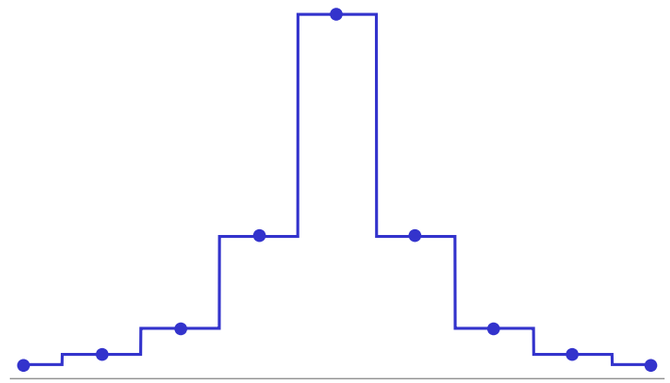
- Solving linear system is expensive, poorly conditioned
- Solution is not useful anyway: too wiggly!
  Runge's phenomenon



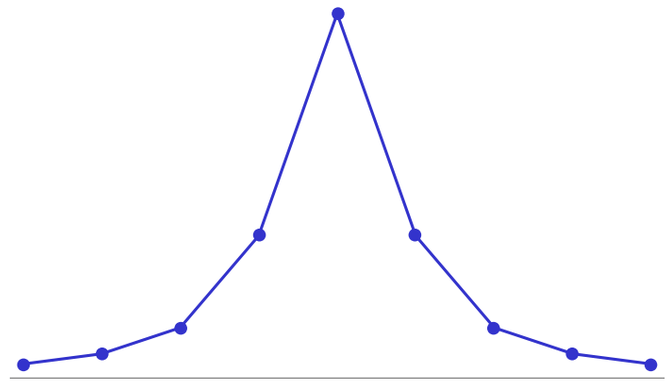
black: $f(x) = 1/(1 + 25x^2)$
red: polynomial through 5 points
green: polynomial through 9 points
blue: polynomial through 17 points

Piecewise polynomial interpolation

Divide domain into pieces, use low-order polynomial interpolation on each piece

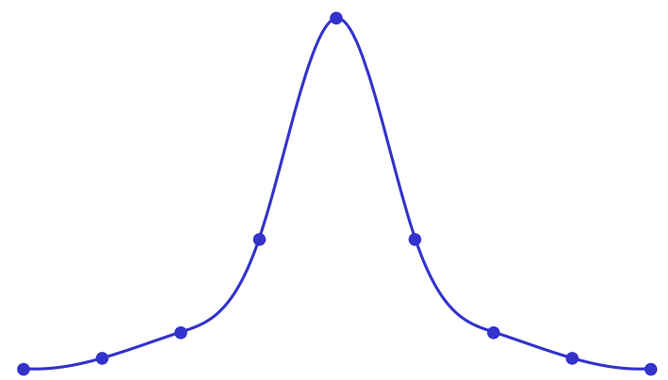


- **Piecewise constant**



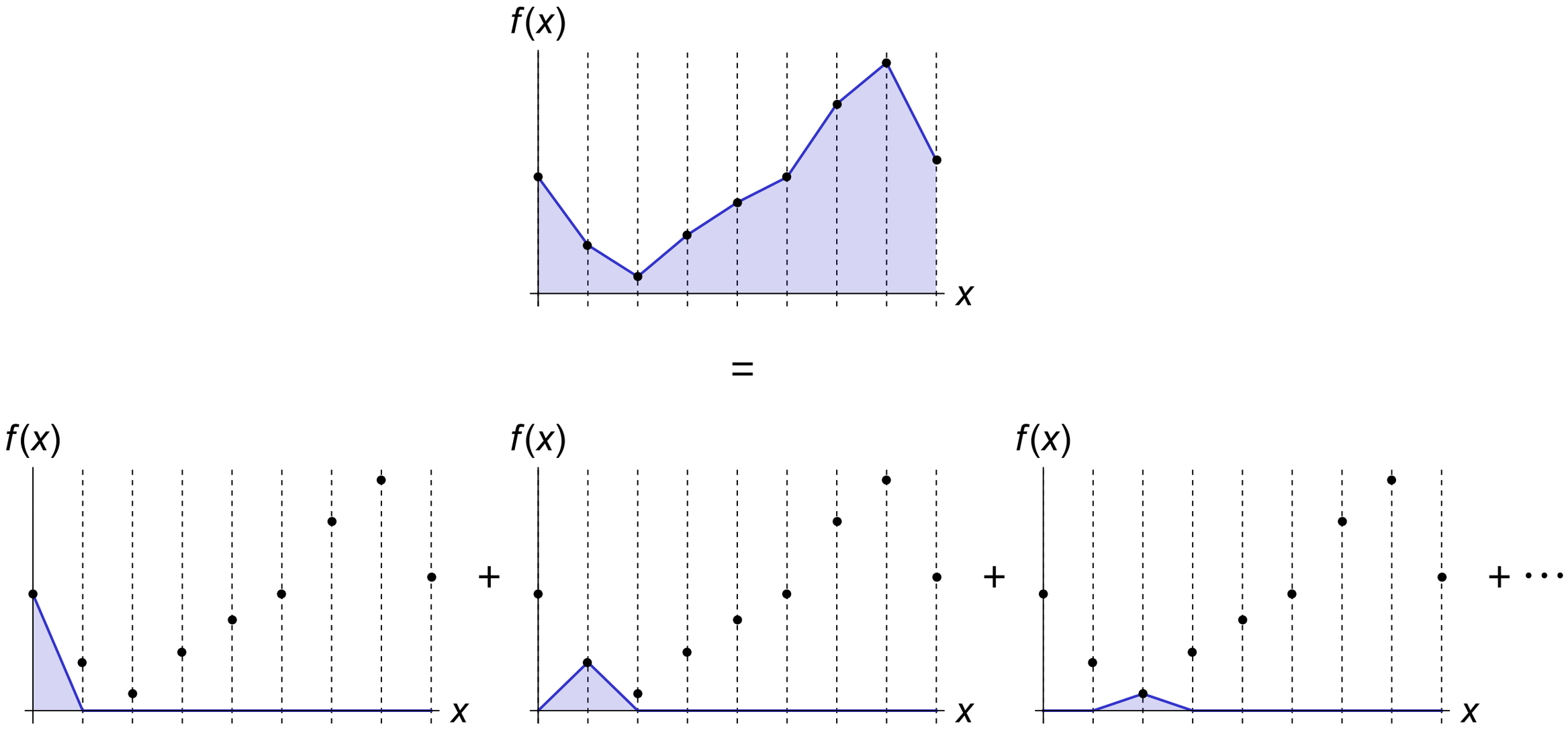
- **Piecewise linear**
- **Piecewise cubic**

Increasing degrees of smoothness, but also increasing oscillations if input is not smooth



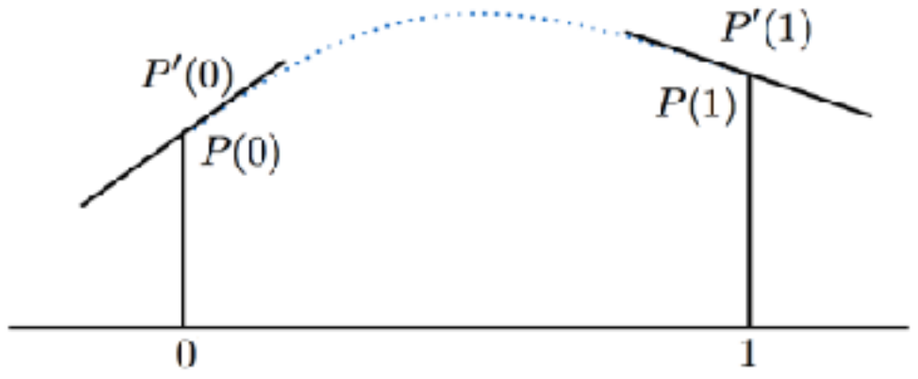
Piecewise polynomial interpolation

Can be viewed as a linear combination of piecewise polynomial basis functions, e.g.



Piecewise cubic interpolation

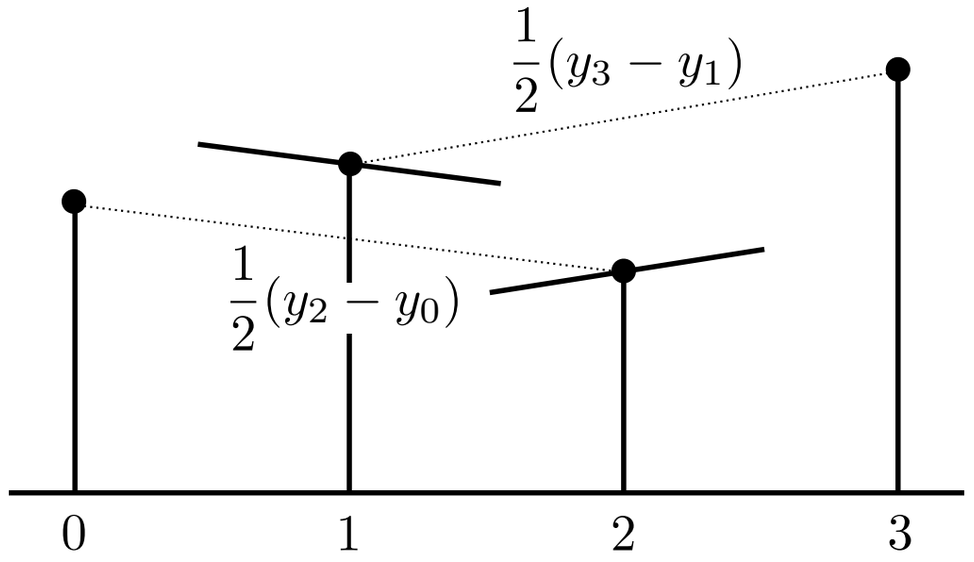
On each interval $[x_i, x_{i+1}]$, fit a cubic
$p(x) = a_0 + a_1x + a_2x^2 + a_3x^3$



Need 4 equations:
$p(x_i) = y_i$, $p(x_{i+1}) = y_{i+1}$, $p'(x_i) = ?$, $p'(x_{i+1}) = ?$

For differentiability, p' must match on adjacent intervals

- **Cubic spline**: make p'' match as well, solve system of equations over all intervals
- Simpler approach (usually what's meant by **piecewise cubic**): take $p'(x_i) = (y_{i+1}-y_{i-1})/(x_{i+1}-x_{i-1})$

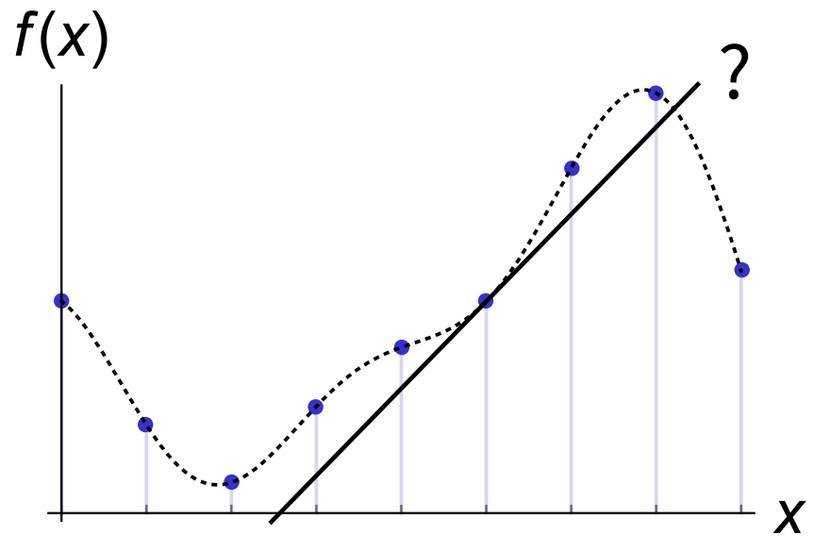


Numerical differentiation

Numerical differentiation

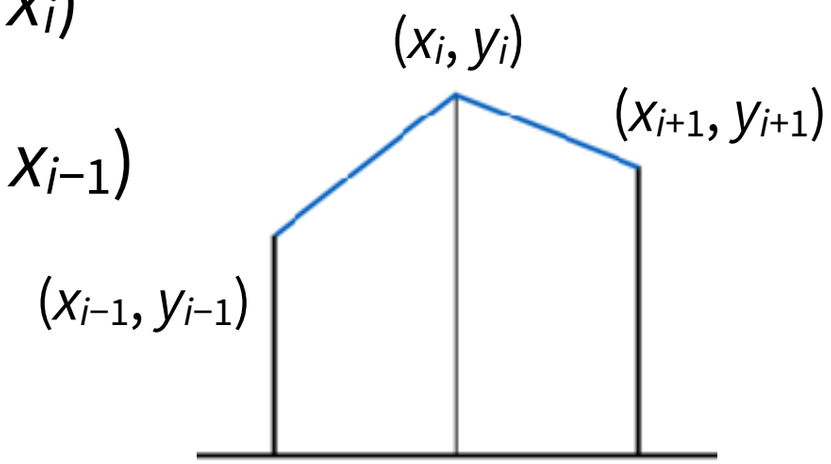
Estimate $f'(x_i)$ knowing only samples $y_i = f(x_i)$

How? Construct local interpolation, differentiate that



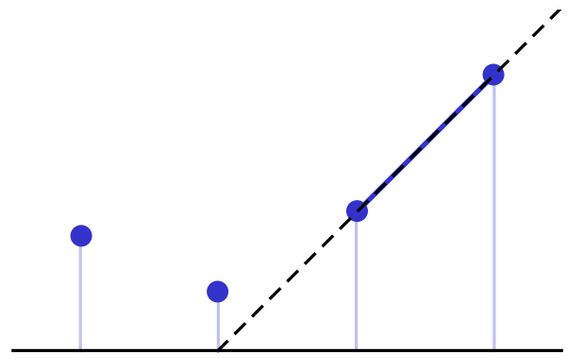
e.g. linear interpolation using one adjacent point

- **Forward difference**: $f'(x_i) \approx (y_{i+1} - y_i)/(x_{i+1} - x_i)$
- **Backward difference**: $f'(x_i) \approx (y_i - y_{i-1})/(x_i - x_{i-1})$

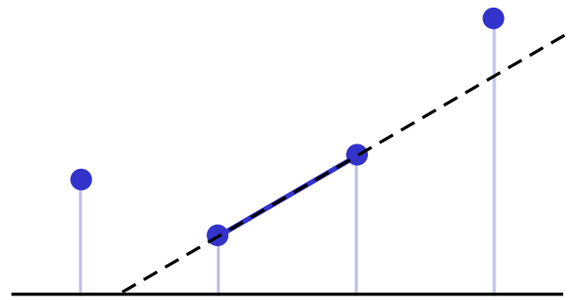


Numerical differentiation

- **Forward difference**:
  $f'(x_i) \approx (y_{i+1} - y_i)/(x_{i+1} - x_i)$



- **Backward difference**:
  $f'(x_i) \approx (y_i - y_{i-1})/(x_i - x_{i-1})$



Compare with definition of derivative:

$$f'(x_i) = \lim_{h \to 0} (f(x+h) - f(x))/h$$

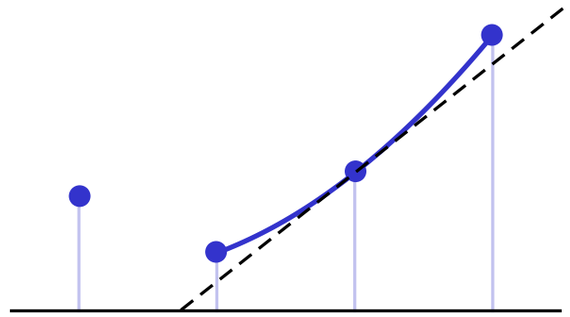
Only change: h does not go to 0, has a finite value $x_{i+1} - x_i$
So these formulas are called **finite differences**

Second-order finite differences

Take two adjacent points, perform quadratic interpolation, differentiate at x_i

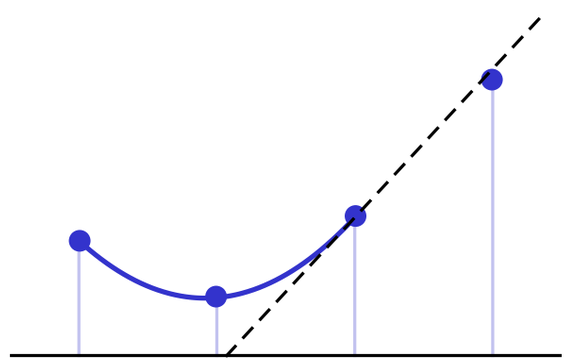
- Using x_{i-1}, x_i, x_{i+1} gives **centered difference**:

$$f'(x_i) \approx (y_{i+1} - y_{i-1})/(2h)$$



- Using x_{i-2}, x_{i-1}, x_i gives **2nd-order backward differentiation formula** (BDF2):

$$f'(x_i) \approx ({}^3/_2\, y_i - 2y_{i-1} + {}^1/_2\, y_{i-2})/h$$



These ought to be more accurate… but how can we prove it?

Accuracy analysis

Taylor series:

$$f(x+h) = f(x) + h\,f'(x) + h^2/2\,f''(x) + \cdots$$

Plug into forward difference formula:

$$(f(x+h) - f(x))/h \quad = \quad f'(x) \quad + \quad h/2\,f''(x) + \cdots$$

Forward difference Desired value Error terms

Error is $O(h)$: forward difference has **first-order accuracy**
Same for backward difference

Centered difference and BDF2 are **second-order**: error is $O(h^2)$

Higher-order formulas possible (but usually not worth it)

Higher-order derivatives

e.g. What if you want to estimate $f''(x_i)$?

Quadratic interpolation yields **three-point stencil**:

$$f''(x_i) \approx (y_{i+1} - 2y_i + y_{i-1})/h^2$$

Error = $O(h^2)$

Same as forward difference of backward difference (or vice versa)!

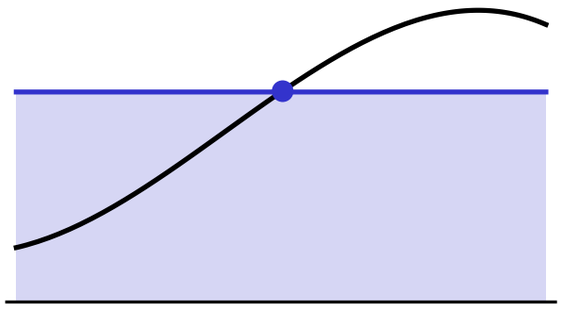
- What if you apply backward difference twice, or forward difference twice?

Numerical integration

Numerical integration

Converse problem of differentiation: Estimate $\int f(x)\,\mathrm{d}x$ over an interval $[a, b]$ using only finitely many evaluations $y_i = f(x_i)$

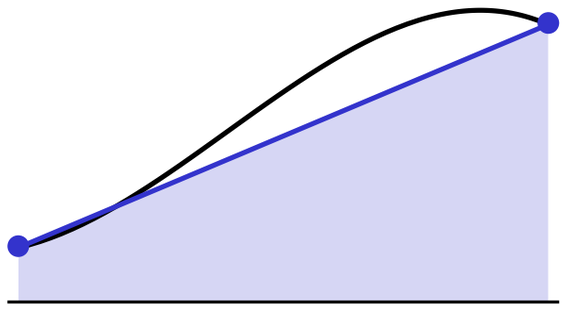
This is also called **quadrature** (= finding area)



Midpoint rule

$$h\,f\left(\frac{a+b}{2}\right)$$

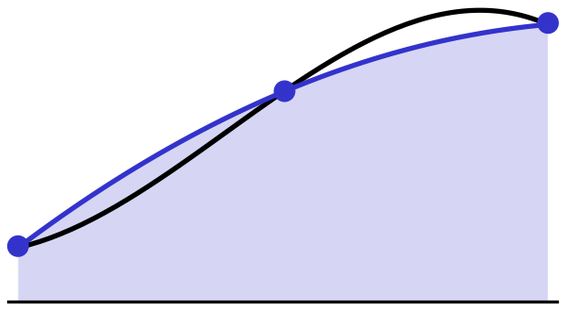
where $h = b - a$



Trapezoidal rule

$$\frac{h}{2}\left(f(a)+f(b)\right)$$

where $h = b - a$



Simpson's rule

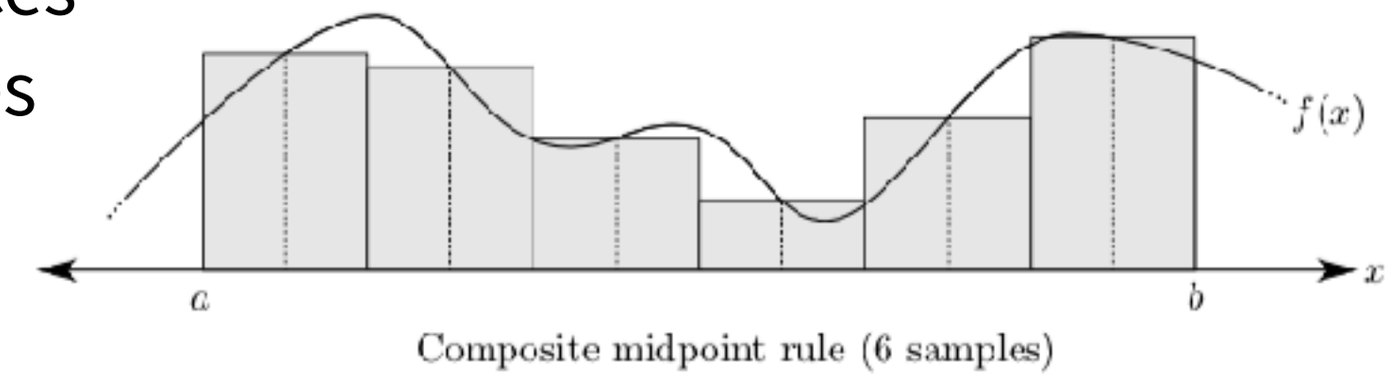
$$\frac{h}{3}\left(f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right)$$

where $h = (b - a)/2$

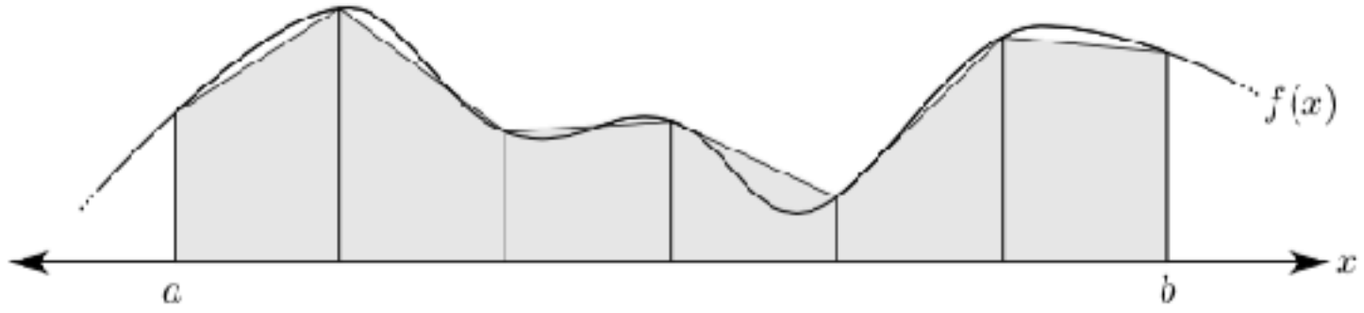
Composite numerical integration

More samples? Don’t use higher-order interpolation (why?)

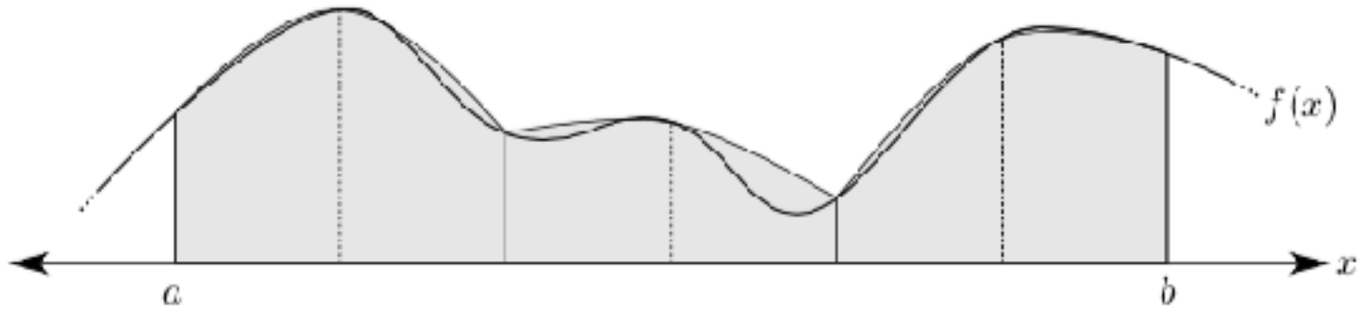
Divide interval into pieces
and apply previous rules



Composite midpoint rule (6 samples)



Composite trapezoidal rule (7 samples)



Composite Simpson’s rule (7 samples)

[Solomon]

Accuracy analysis

Can be evaluated through Taylor series again

- Midpoint rule: Error in each interval is $O(h^3)$, but there are $O(1/h)$ intervals, so **total error** = $O(h^3) \cdot O(1/h) = O(h^2)$
- Trapezoidal rule: also $O(h^2)$
- Simpson's rule: $O(h^4)$

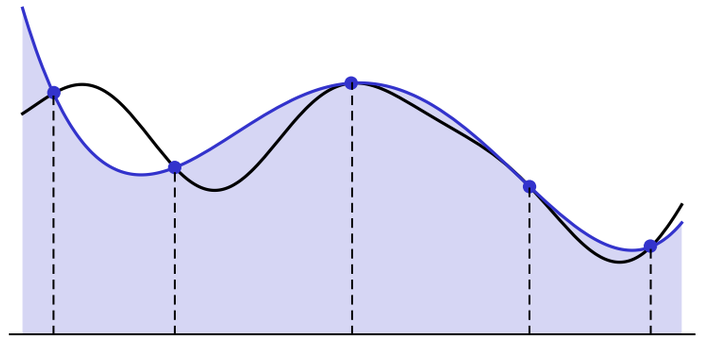
Midpoint and trapezoidal have **linear precision**: give exact result for linear polynomials. Simpson's rule has **cubic precision**

1. How can midpoint have equal accuracy to trapezoidal?
2. Can piecewise cubic do better than Simpson's rule?

Beyond equally spaced samples

Can do better if we place samples carefully

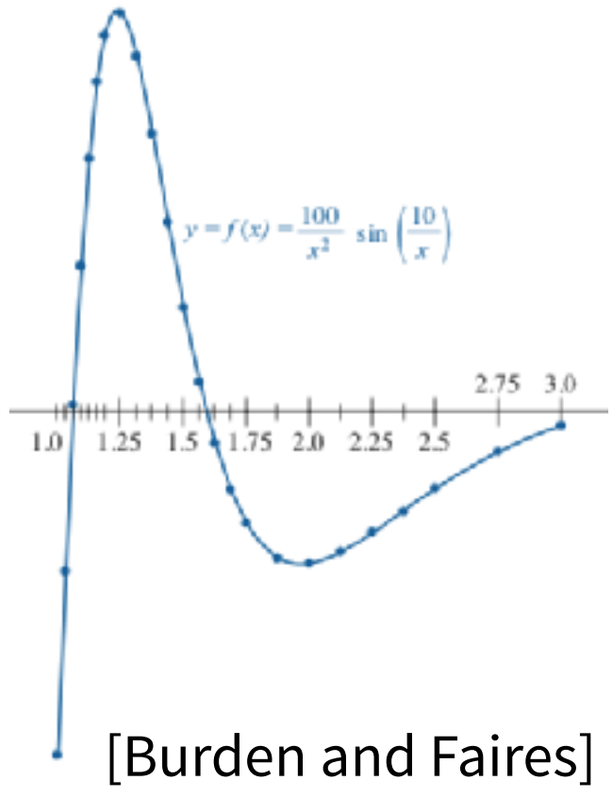
Gaussian quadrature places more points near endpoints to minimize oscillations



- Gives order $2n+1$ precision for n samples

Adaptive quadrature: subdivide intervals where interpolation is inaccurate

- Useful for functions that vary rapidly in only a small part of the domain



[Burden and Faires]