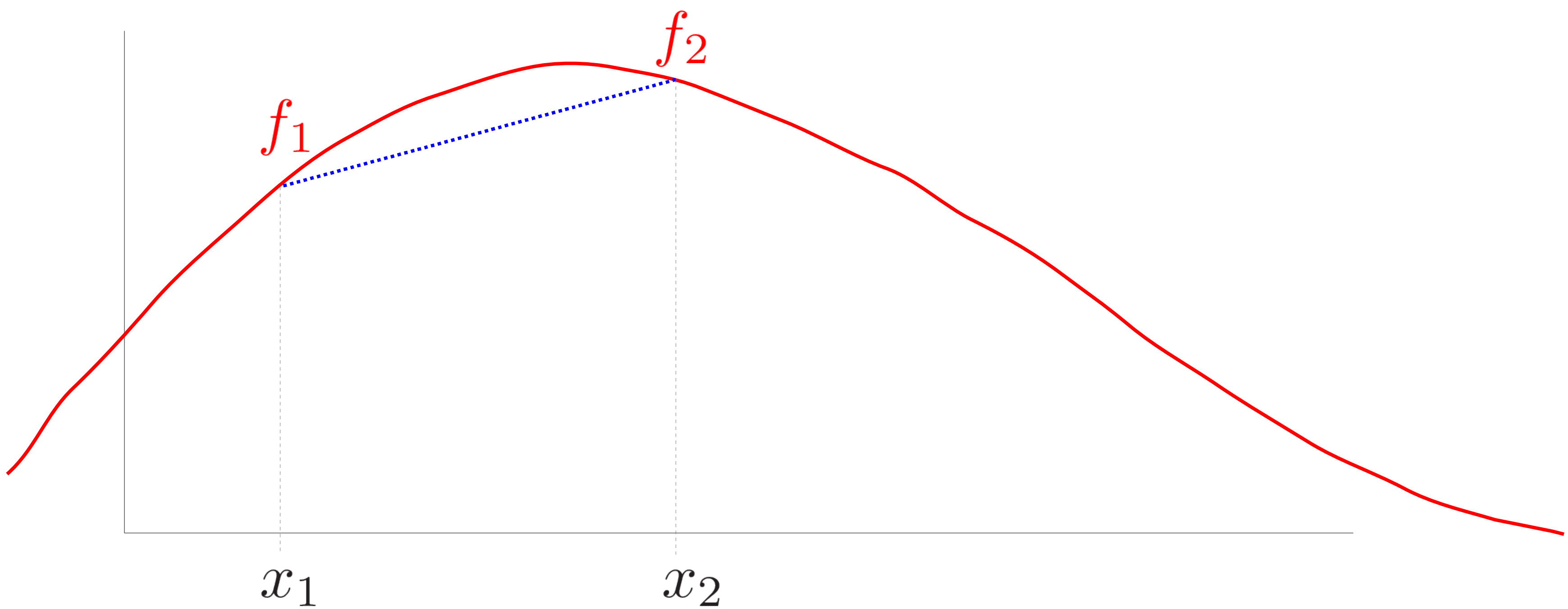
# Interpolation

- As we've seen, we frequent have data only at a discrete number of points
  - Interpolation fills in the gaps by making an assumption about the behavior of the functional form of the data
- Many different types of interpolation exist
  - Some ensure no new extrema are introduced
  - Some match derivatives at end points
  - ...
- Generally speaking: larger number of points used to build the interpolant, the higher the accuracy
  - Pathological cases exist
  - You may want to enforce some other property on the form of the interpolant
- We'll follow the discussion from Pang

# Linear Interpolation

- Simplest idea—draw a line between two points




$$f(x) = \frac{f_2 - f_1}{x_2 - x_1}(x - x_1) + f_1$$

  - Exactly recovers the function values at the end points

# Linear Interpolation

- Actual $f(x)$ is given by

$$f(x) = \underbrace{f_i + \frac{x - x_i}{\Delta x}(f_{i+1} - f_i)}_{\text{Linear interpolant}} + \Delta f(x)$$

  - Want to know the error at a point $x = a$ in $[x_i, x_{i+1}]$
  - We can fit a quadratic to $x_i, a, x_{i+1}$
  - Error can be shown (through lots of algebra...) to be

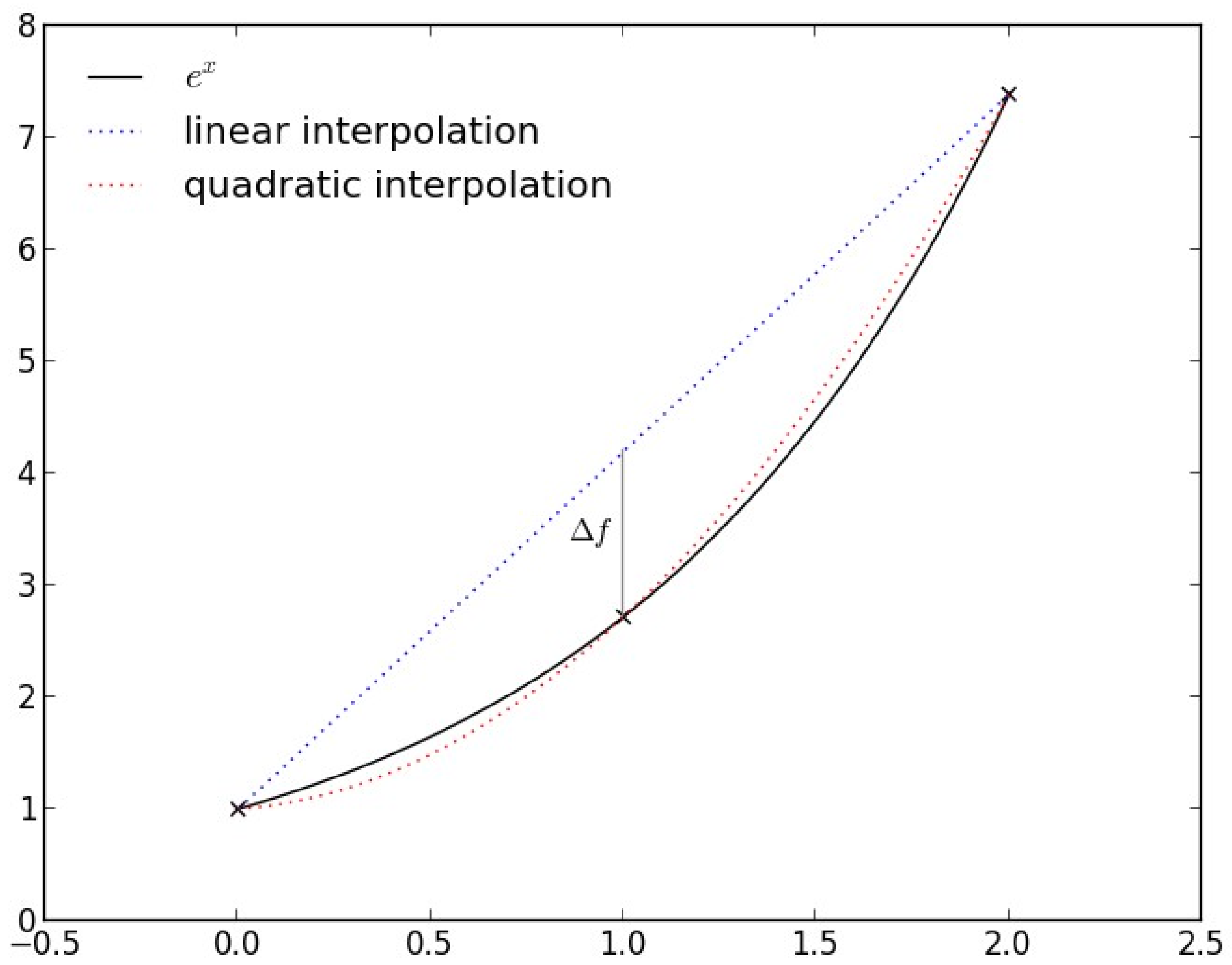
$$\Delta f(x) = \frac{f''(x)}{2}(x - x_i)(x - x_{i+1})\bigg|_{x=a}$$

  (at that point)

- This means error in linear interpolation $\sim O(\Delta x^2)$

# Linear Interpolation

- Error estimate graphically



# Quadratic Interpolation

- Fit a parabola—requires 3 points
  - Already saw this with Simpson's rule
- Note: can fall out of the range of $f_1$ , $f_2$ , or $f_3$

# Example: Mass Fractions

- Higher-order is not always better.
- Practical example
  - In hydrodynamics codes, you often carry around mass fractions, $X_k$ with

$$\sum_k X_k = 1$$

  - If you have these defined at two points: $a$ and $b$ and need them in-between, then:

$$X_k(x) = (X_k)_a + \frac{(X_k)_b - (X_k)_a}{\Delta x}(x - a)$$

    sums to 1 for all $x$
  - Higher-order interpolation can violate this constraint

# Lagrange Interpolation

- General method for building a single polynomial that goes through all the points (alternate formulations exist)
- Given $n$ points: $x_0, x_1, \ldots, x_{n-1}$ , with associated function values: $f_0, f_1, \ldots, f_{n-1}$
  - construct basis functions:

$$l_i(x) = \prod_{j=0, i \neq j}^{n-1} \frac{x - x_j}{x_i - x_j}$$

    - Basis function $l_i$ is 0 at all $x_j$ except for $x_i$ (where it is 1)
  - Function value at $x$ is:

$$f(x) = \sum_{i=0}^{n-1} l_i f_i$$

# Lagrange Interpolation

- Consider a quadratic
  - Three points: $(x_0, f_0), (x_1, f_1), (x_2, f_2)$
  - Three basis functions:

$$l_0 = \frac{x - x_1}{x_0 - x_1} \cdot \frac{x - x_2}{x_0 - x_2} = \frac{(x - x_1)(x - x_2)}{2\Delta x^2}$$

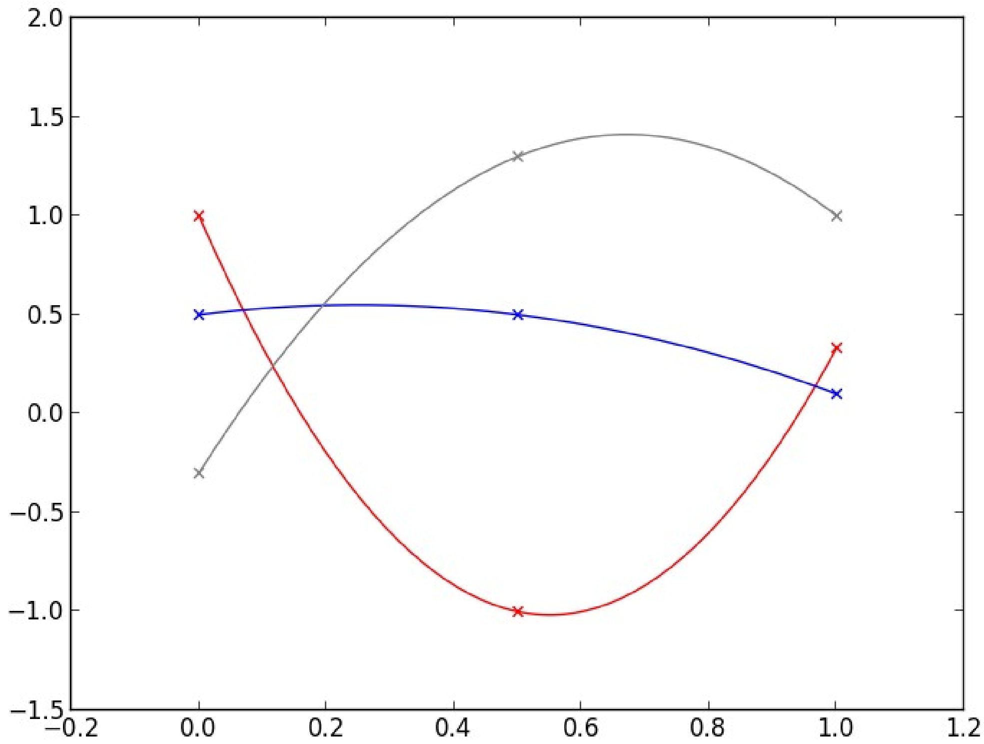
$$l_1 = \frac{x - x_0}{x_1 - x_0} \cdot \frac{x - x_2}{x_1 - x_2} = -\frac{(x - x_0)(x - x_2)}{\Delta x^2}$$

$$l_2 = \frac{x - x_0}{x_2 - x_0} \cdot \frac{x - x_1}{x_2 - x_1} = \frac{(x - x_0)(x - x_1)}{2\Delta x^2}$$

# Lagrange Interpolation

- Quadratic Lagrange polynomial:

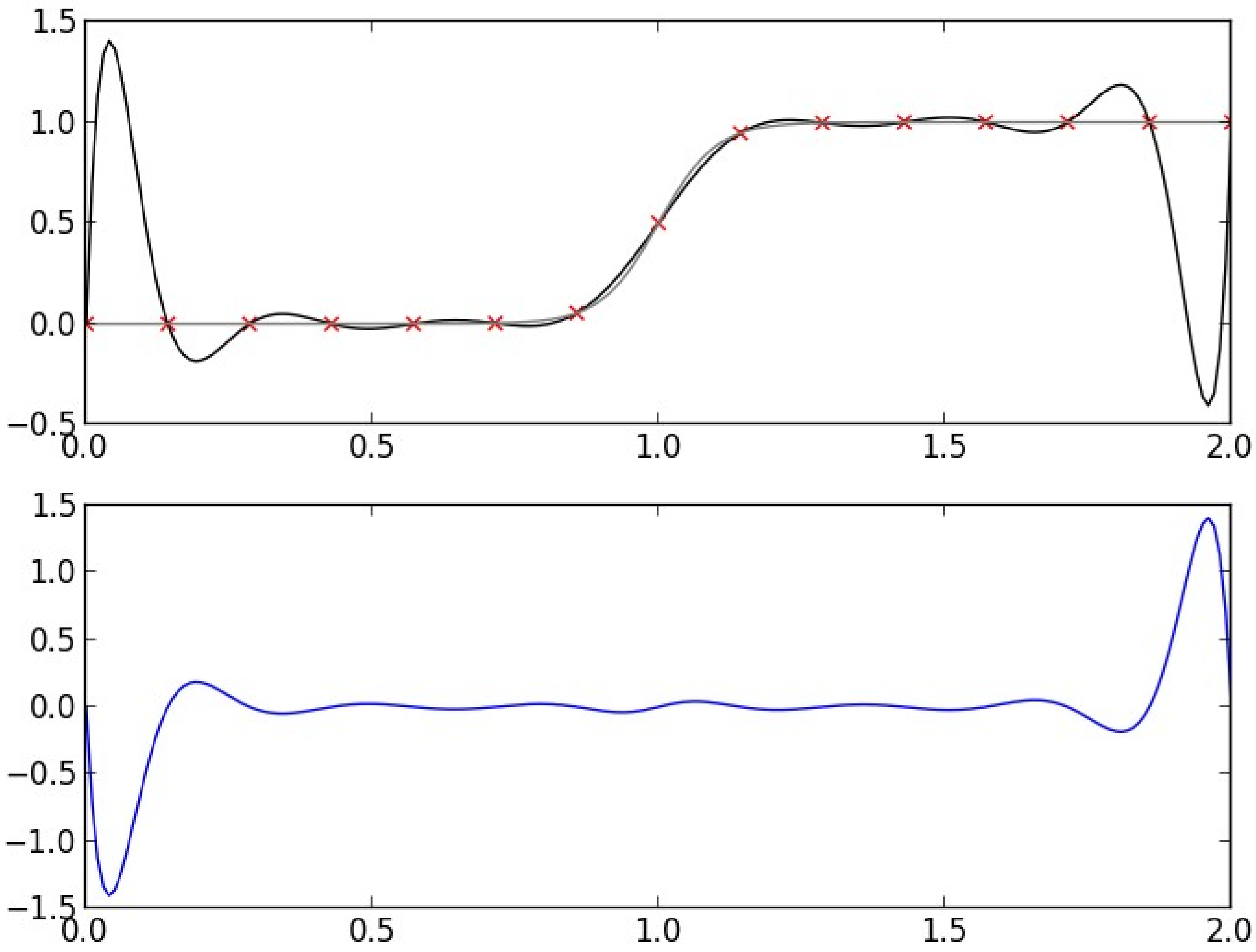
$$f(x) = \frac{(x - x_1)(x - x_2)}{2\Delta x^2} f_0 - \frac{(x - x_0)(x - x_2)}{\Delta x^2} f_1 + \frac{(x - x_0)(x - x_1)}{2\Delta x^2} f_2$$



# Lagrange Interpolation

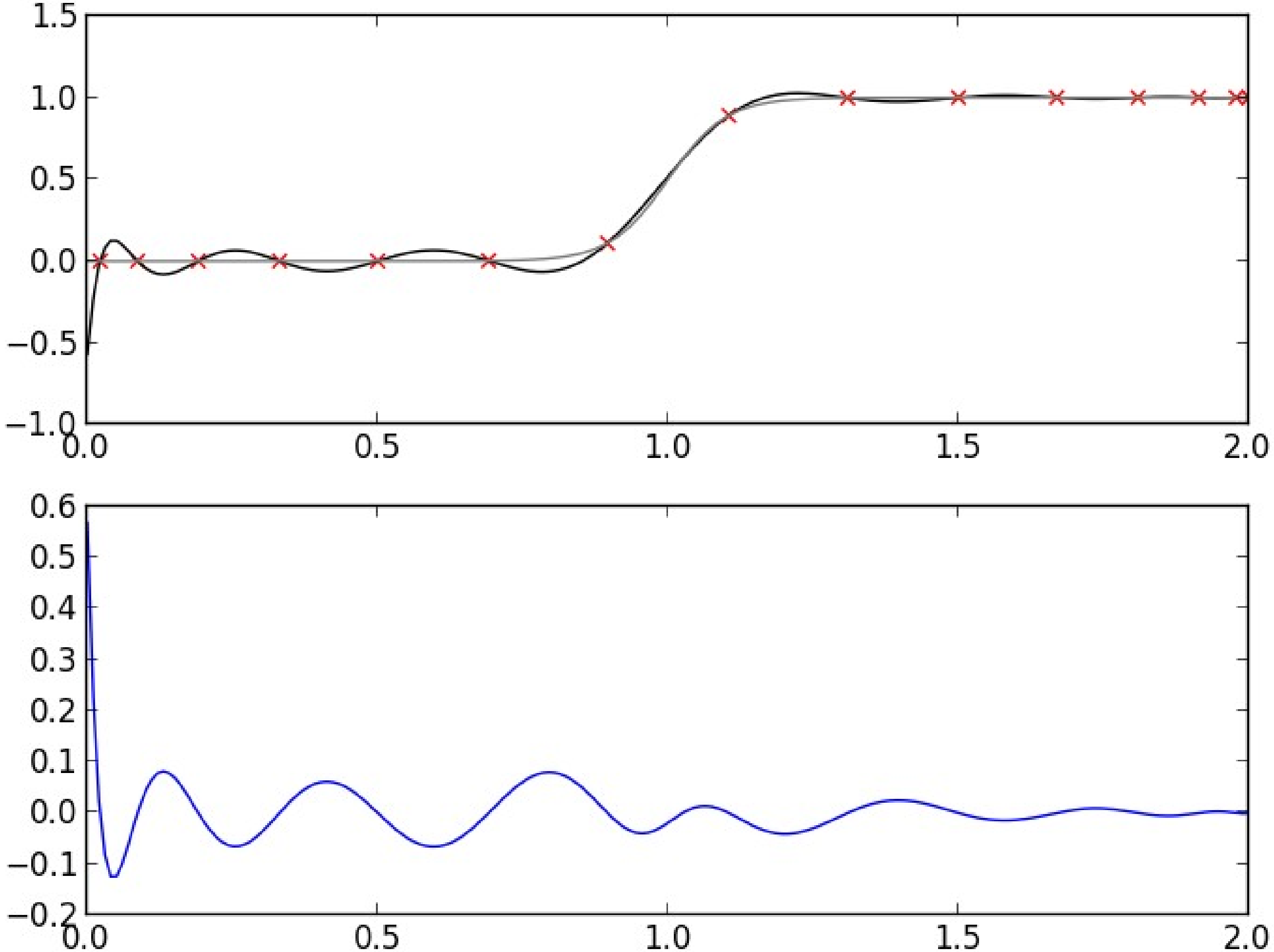
- Form is easy to remember
- Not the most efficient form to compute the polynomial
  - All other forms for $n$-degree polynomials that pass through the specified $n+1$ points are equivalent
  - High-order polynomials can suffer from round-off error

# Lagrange Interpolation




Interpolation through 15 points sampled uniformly from tanh(). The error is shown below.

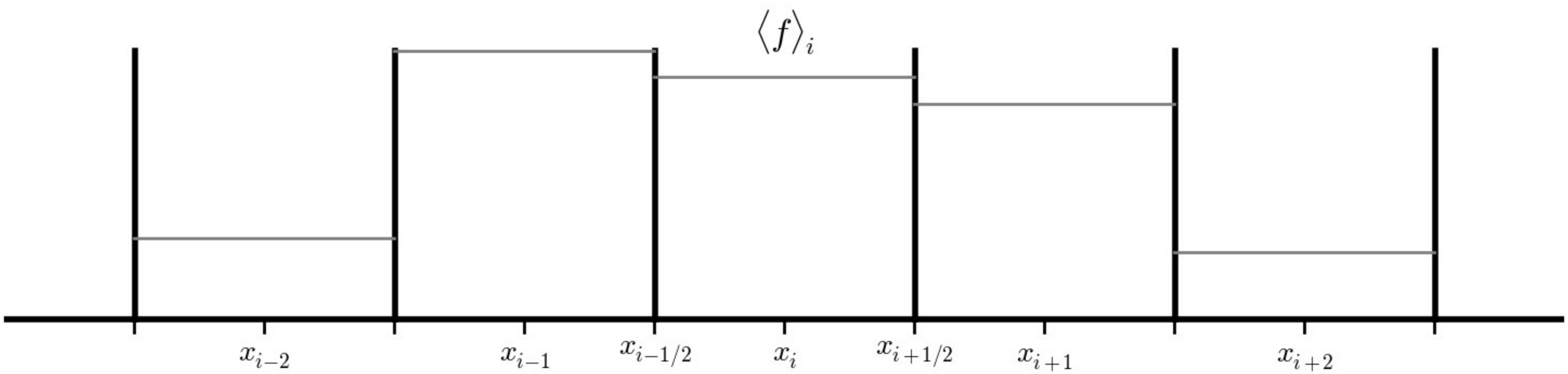
# Lagrange Interpolation




Again: interpolation through 15 points sampled from tanh(), but now with non-uniform spacing (this choice is the Chebyshev nodes). The error is shown below.

# Conservative Interpolation

- Imagine that instead of having f(x) at discrete points, we instead knew the average of f in some interval
  - Finite-volume discretization




$$\langle f \rangle_i = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} f(x) dx$$

- We want a interpolant that respects these averages
  - Conservative interpolant

# Conservative Interpolation

- Quadratic interpolation
  - We are given the average of f in each zone
  - Constraints:

$$\langle f \rangle_{i-1} = \frac{1}{\Delta x} \int_{x_{i-3/2}}^{x_{i-1/2}} f(x) dx$$

$$\langle f \rangle_{i} = \frac{1}{\Delta x} \int_{x_{i-1/2}}^{x_{i+1/2}} f(x) dx$$

$$\langle f \rangle_{i+1} = \frac{1}{\Delta x} \int_{x_{i+1/2}}^{x_{i+3/2}} f(x) dx$$

  - Reconstruction polynomial

$$f(x) = a(x - x_i)^2 + b(x - x_i) + c$$

  - Three equations and three unknowns

# Conservative Interpolation

- Solve for unknowns:

$$f(x) = \frac{\langle f\rangle_{i-1} - 2\langle f\rangle_i + \langle f\rangle_{i+1}}{2\Delta x^2}(x - x_i)^2 + \frac{\langle f\rangle_{i+1} - \langle f\rangle_{i-1}}{2\Delta x}(x - x_i) + \frac{-\langle f\rangle_{i-1} + 26\langle f\rangle_i - \langle f\rangle_{i+1}}{24}$$

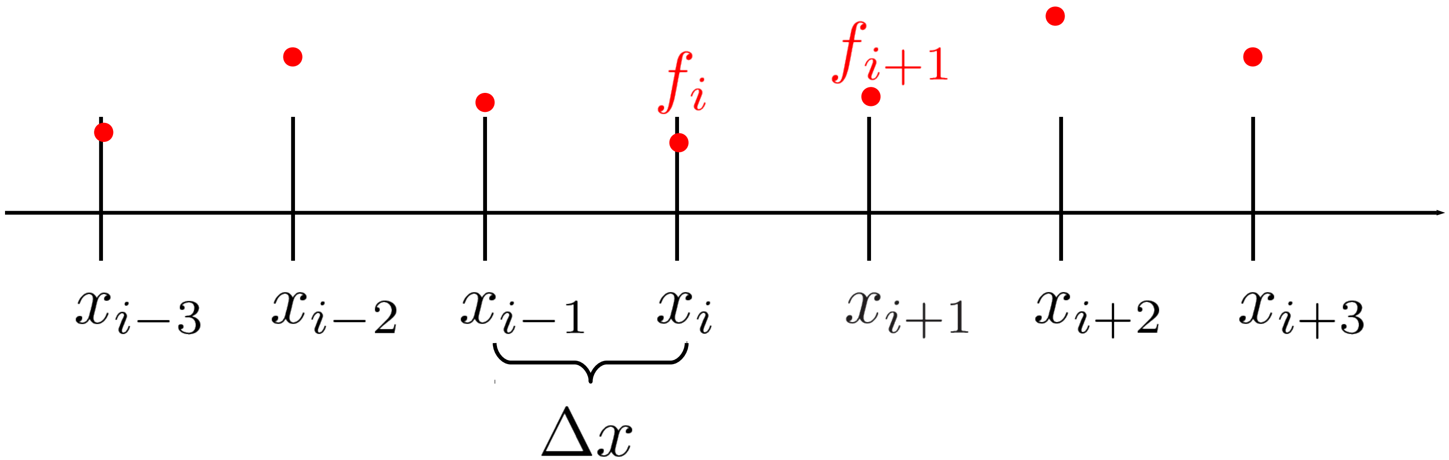
  - This recovers the proper averages in each interval
  - Usually termed: reconstruction
  - We'll see this when we talk about finite-volume methods for advection.

# Splines

- So far, we've only worried about going through the specified points
- Large number of points → two distinct options:
  - Use a single high-order polynomial that passes through them all
  - Fit a (somewhat) high order polynomial to *each interval* and match all derivatives at each point—this is a spline
- Splines match the derivatives at end points of intervals
  - Piecewise splines can give a high-degree of accuracy
- Cubic spline is the most popular
  - Matches first and second derivative at each data point
  - Results in a smooth appearance
  - Avoids severe oscillations of higher-order polynomial

# Splines

- We have a set of discrete data: $f_i = f(x_i)$ at $x_0, x_1, x_2, \ldots, x_n$
  - We'll assume regular spacing here




- m-th order polynomial in $[x_i, x_{i+1}]$

$$p_i(x) = \sum_{k=0}^{m} c_{i,k} x^k$$

- Smoothness requirement: all derivatives match at endpoints

$$p_i^{(l)}(x_{i+1}) = p_{i+1}^{(l)}(x_{i+1}) \qquad l = 0, 1, \ldots, m-1$$

# Splines

- We have $(m+1)\,n$ coefficients
  - Smoothness operates on the $n-1$ interior points
  - End point values provide 2 more constraints
  - Remaining constraints come from imposing conditions on the second derivatives at end points
- You are solving for the coefficients of all piecewise polynomial interpolants together, in a coupled fashion.

# Cubic Splines

- Cubic splines: 3rd order polynomial
  - 1. Start by linearly interpolating second derivatives

$$p_i''(x) = \frac{1}{\Delta x}\left[(x - x_i)p_{i+1}'' - (x - x_{i+1})p_i''\right]$$

  - 2. Integrate twice:

$$p_i''(x) = \frac{1}{6\Delta x}\left[(x - x_i)^3 p_{i+1}'' - (x - x_{i+1})^3 p_i''\right] + A(x - x_i) + B(x - x_{i+1})$$

    (note we wrote the integration constants in a convenient form)

  - 3. Impose constraints: $p(x_i) = f_i$ and $p(x_{i+1}) = f_{i+1}$

Note: different texts use different forms of the cubic—the ideas are all the same though. This form also seems to be what NR chooses.

# Cubic Splines

- Result (after a bunch of algebra):

$$p_i(x) = \alpha_i (x - x_i)^3 + \beta_i (x - x_{i+1})^3 + \gamma_i (x - x_i) + \eta_i (x - x_{i+1})$$

$$\alpha_i = \frac{p''_{i+1}}{6\Delta x} \quad \beta_i = -\frac{p''_i}{6\Delta x} \quad \gamma_i = \frac{-\frac{1}{6} p''_{i+1} \Delta x^2 + f_{i+1}}{\Delta x} \quad \eta_i = \frac{\frac{1}{6} p''_i \Delta x^2 - f_i}{\Delta x}$$

- Note that all the coefficients in the cubic are in terms of the second derivative at the data points.
  - We need to solve for all second derivatives
- Final continuity constraint:

$$p'_{i-1}(x_i) = p'_i(x_i)$$

# Cubic Splines

- After lots of algebra, we arrive at:

$$p''_{i-1}\Delta x + 4p''_i\Delta x + p''_{i+1}\Delta x = \frac{6}{\Delta x}\left(f_{i-1} - 2f_i + f_{i+1}\right)$$

  - This is a linear system
  - Applies to all interior points, $i = 1, \ldots\ , n{-}1$ to give $p''_i$
  - Natural boundary conditions:

$$p''_0 = p''_n = 0$$

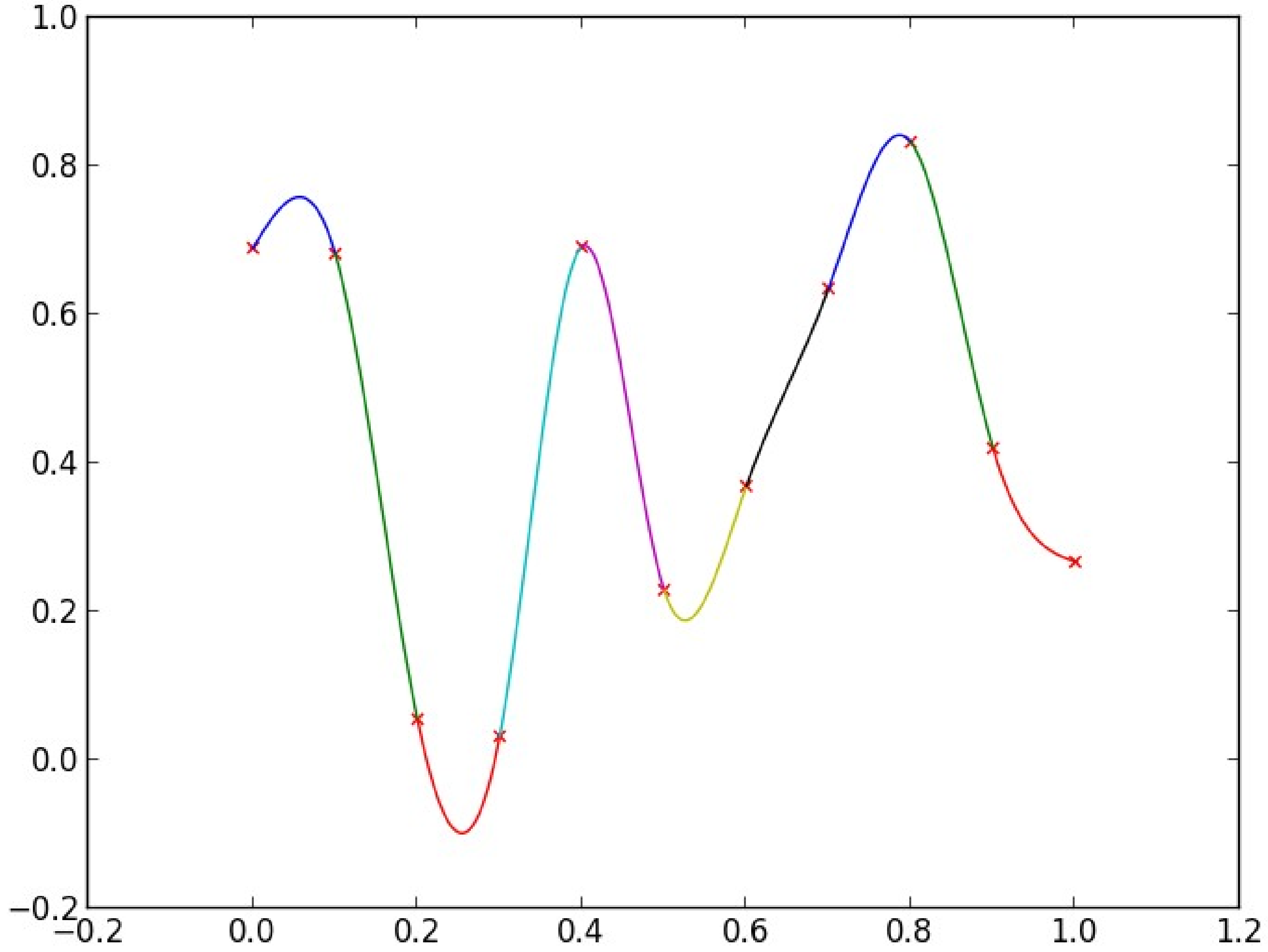
# Cubic Splines

- Matrix form:

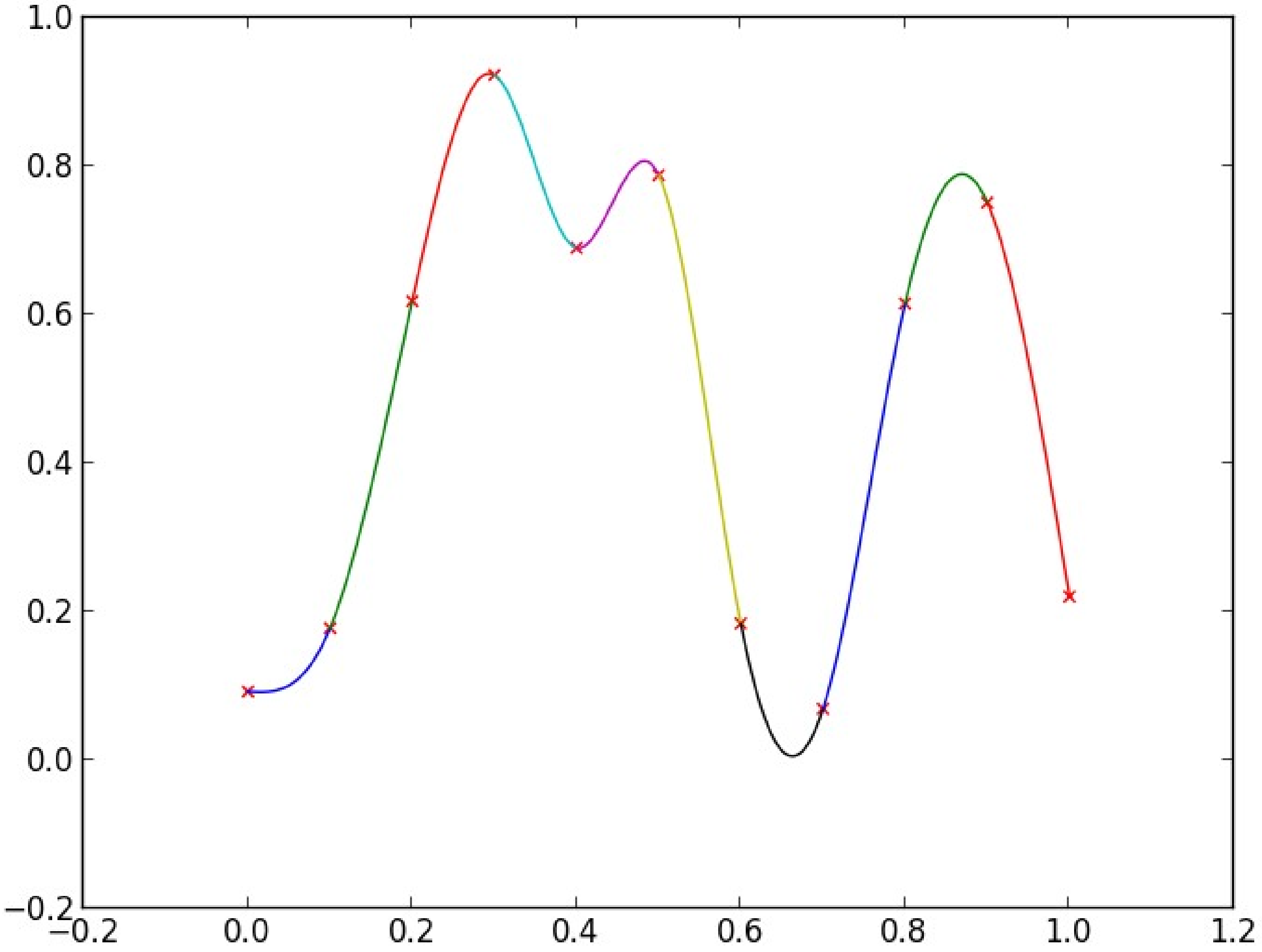
$$
\begin{pmatrix}
4\Delta x & \Delta x & & & & & & \\
\Delta x & 4\Delta x & \Delta x & & & & & \\
 & \Delta x & 4\Delta x & \Delta x & & & & \\
 & & \ddots & \ddots & \ddots & & & \\
 & & & \ddots & \ddots & \ddots & & \\
 & & & & \Delta x & 4\Delta x & \Delta x \\
 & & & & & \Delta x & 4\Delta x
\end{pmatrix}
\begin{pmatrix}
p''_1 \\ p''_2 \\ p''_3 \\ \vdots \\ \vdots \\ p''_{n-2} \\ p''_{n-1}
\end{pmatrix}
= \frac{6}{\Delta x}
\begin{pmatrix}
f_0 - 2f_1 + f_2 \\
f_1 - 2f_2 + f_3 \\
f_2 - 2f_3 + f_4 \\
\vdots \\
\vdots \\
f_{n-3} - 2f_{n-2} + f_{n-1} \\
f_{n-2} - 2f_{n-1} + f_n
\end{pmatrix}
$$

  - This is a tridiagonal matrix
  - We'll look at linear algebra later—for now we can use a “canned” solver
  - Example code...

# Cubic Splines



# Cubic Splines



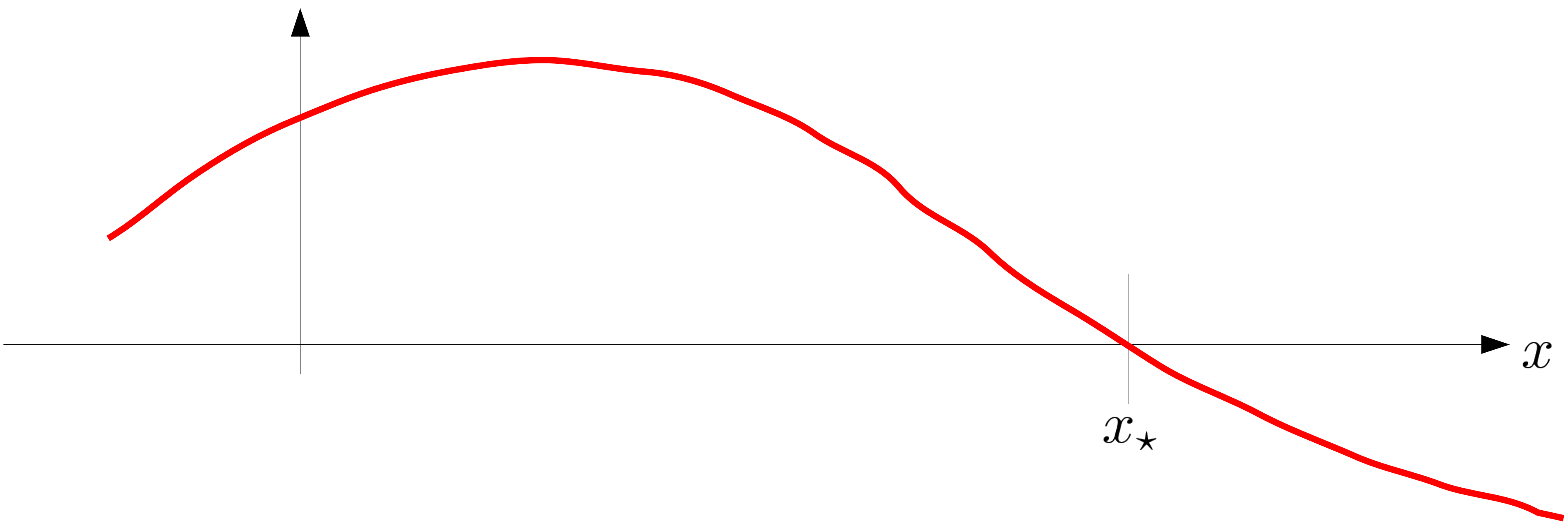

Note that the splines can overshoot the original data values

# Cubic Splines

- Note: cubic splines are not necessarily the most accurate interpolation scheme (and sometimes far from...)
- But, for plotting/graphics applications, they look right

# Root Finding

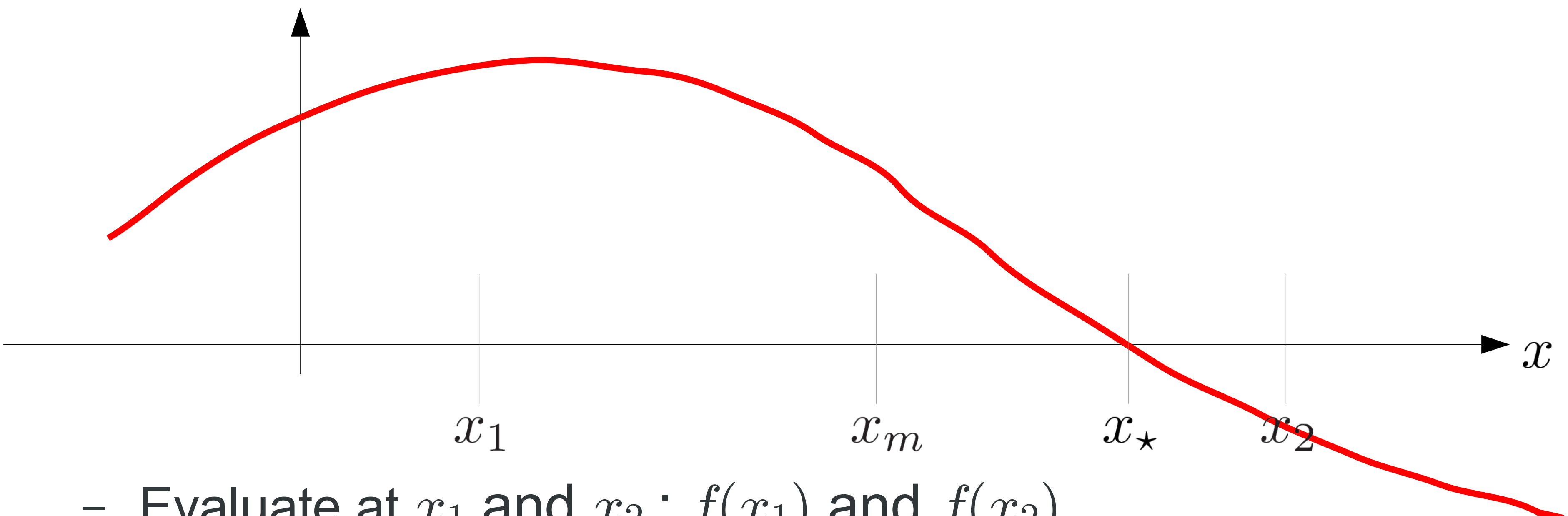
- Basic methods can be understood by looking at the function graphically




  - Function $f(x)$ has a zero at $x_\star$
  - Note the sign of $f(x)$ changes at the root

# Bisection

- Simplest method: bisection

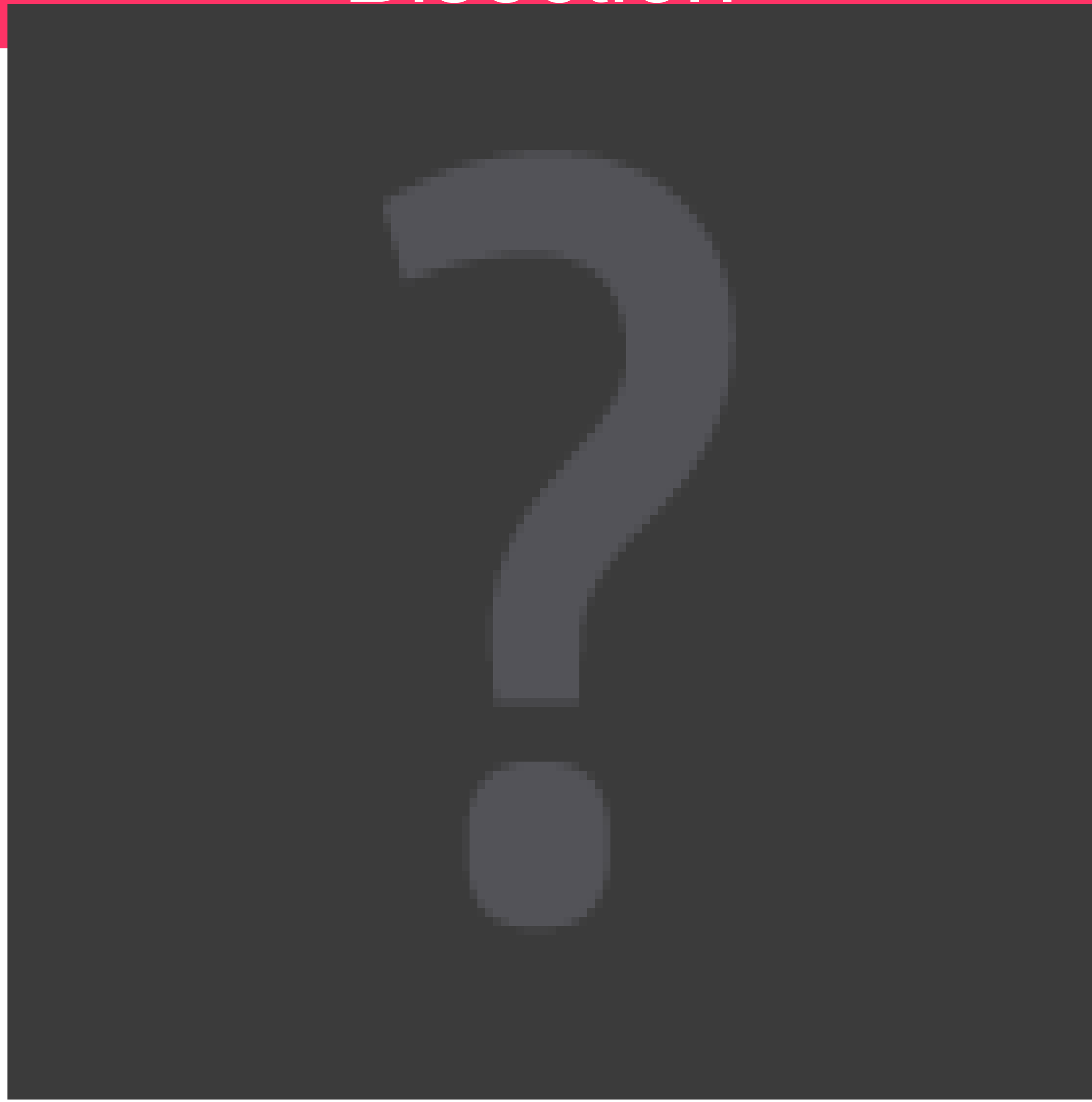



  - Evaluate at $x_1$ and $x_2$ : $f(x_1)$ and $f(x_2)$
  - If these are different signs, then the root lies between them
  - Evaluate at the midpoint: $x_m = (x_1 + x_2)/2$ getting $f(x_m)$
  - The root lies in one of the two intervals—repeat the process

# Bisection

- Need two initial points (guesses) that you believe bound the root
  - If there are two roots in-between, then you are in trouble
  - Some pathological cases: e.g. $f(x) = x^2$
- Convergence can be slow—each iteration reduces the error by a factor of 2

# Bisection



# Newton-Raphson

- If we know $df/dx$ we can do better
  - Start with an initial guess, $x_0$, that is “close” to the root
  - Taylor expansion:

$$f(x_0 + \delta) \approx f(x_0) + f'(x_0)\delta + \ldots$$

  - If we are close, then

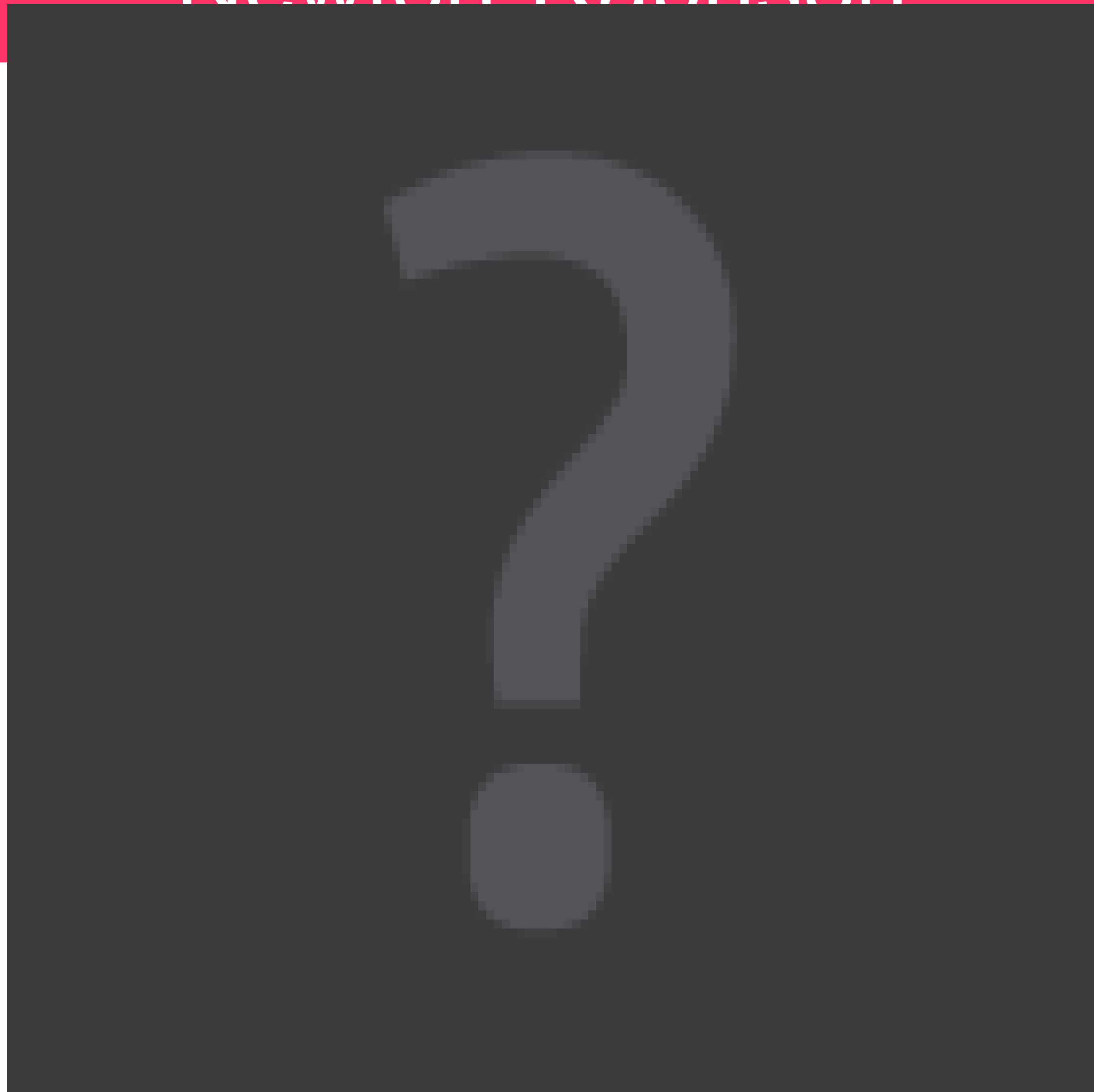
$$f(x_0 + \delta) \approx 0 \longrightarrow \delta = -\frac{f(x_0)}{f'(x_0)}$$

  - Update

$$x_1 = x_0 + \delta$$

  - We can continue, iterating again and again, until the change in the root < ε
- Converges fast: usually only a few iterations are needed

# Newton-Raphson



# Newton-Raphson

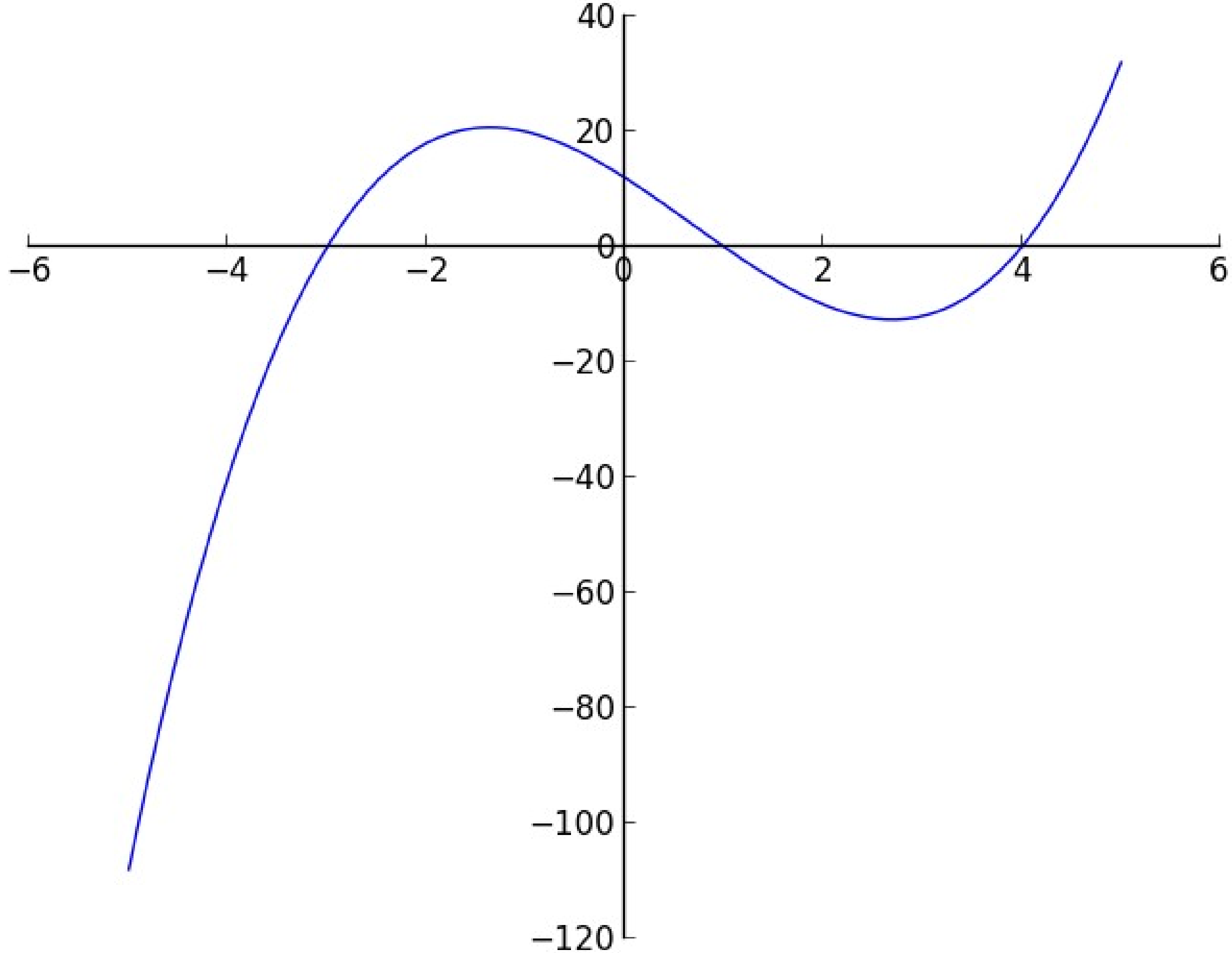
- Requirements for good convergence:
  - Derivative must exists and be non-zero in the interval near the root
  - Second derivative must be finite
  - $x_0$ must be close to the root
- Can be used with systems (we'll see this later)
- Multiple roots?
  - Generally: try to start with a good estimate*

*not a guarantee

# Newton-Raphson

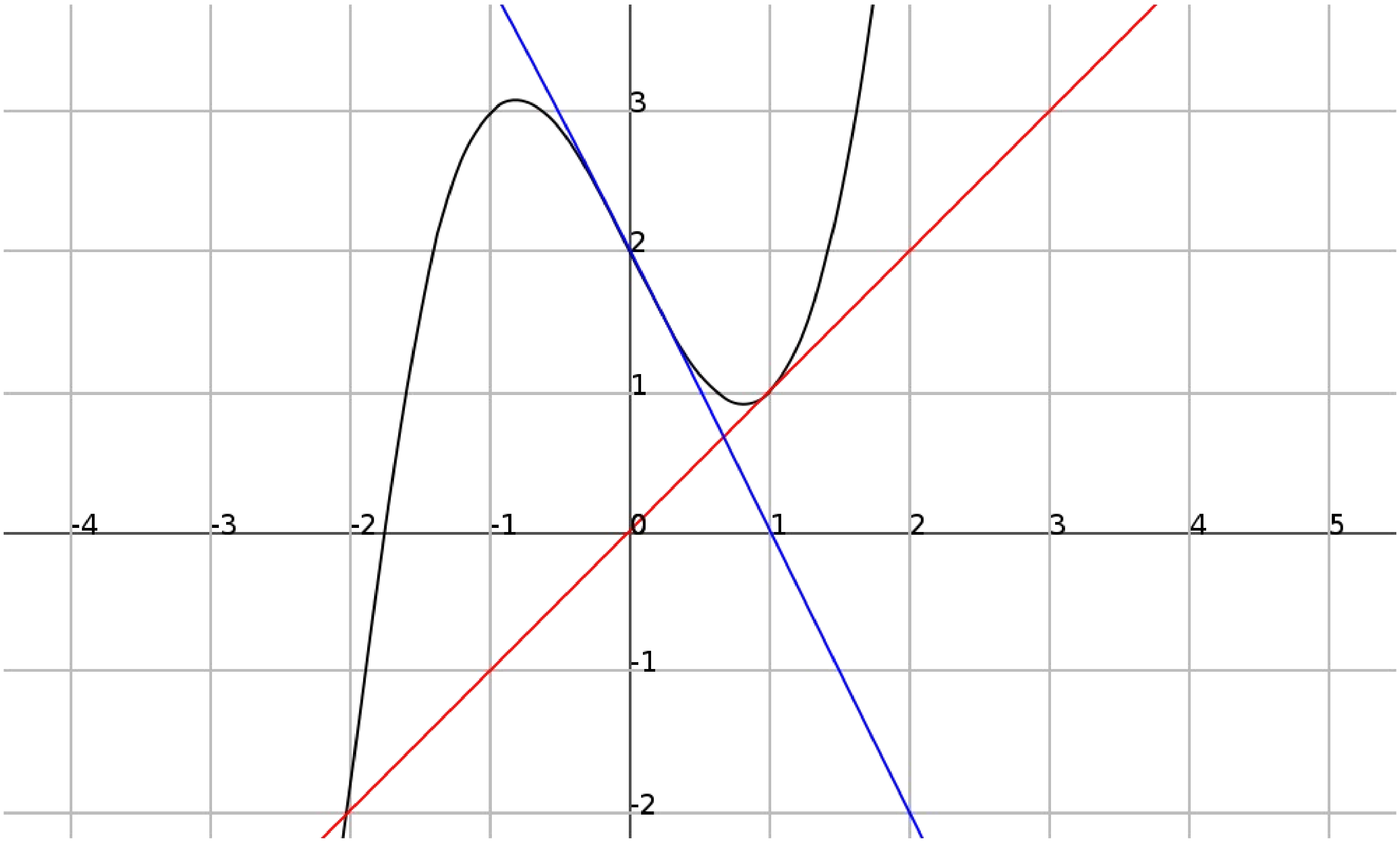
- Basins
  - Consider $q(x) = x^3 - 2x^2 - 11\,x + 12$ (example from Wikipedia / Dence, T. 1997)

| x0 | root |
|---|---|
| 2.35287527 | 4.0 |
| 2.35284172 | -3.0 |
| 2.35283735 | 4.0 |
| 2.352836327 | -3.0 |
| 2.352836323 | 1.0 |



# Newton-Raphson

- Consider $f(x) = x^3 - 2x + 2$
  - Start with 0 → 1 → 0 → 1 → 0 ...
  - Cycle






(wikipedia)

# Secant Method

- If we don't know $df/dx$, we can still use the same ideas
  - We need to initial guesses: $x_{-1}$ and $x_0$
  - Use approximate derivative

$$x_1 = x_0 - \frac{f(x_0)}{[f(x_0) - f(x_{-1})]/(x_0 - x_{-1})}$$

  - Used when an analytic derivative is unavailable, or too expensive to compute (e.g. EOS)

# Practical Notes

- N-R is used successfully with equations of state
  - Function takes $\rho$ , T and we want to come in with P, $\rho$
  - Requires well-behaved derivatives and an initial guess
- Secant method is used, for example, in Riemann solvers in hydrodynamics, where EOS evaluations can be expensive or the derivative may not be known