

# PHY604 Lecture 6

September 19, 2023

Review:

3 point Gauss-Legendre quadrature

$$\int_{-1}^1 f(x) dx \simeq \frac{5}{9} f(-\sqrt{3/5}) + \frac{8}{9} f(0) + \frac{5}{9} f(\sqrt{3/5})$$

# Review: Gauss-Legendre quadrature

- From Newman Sec. 5.6:

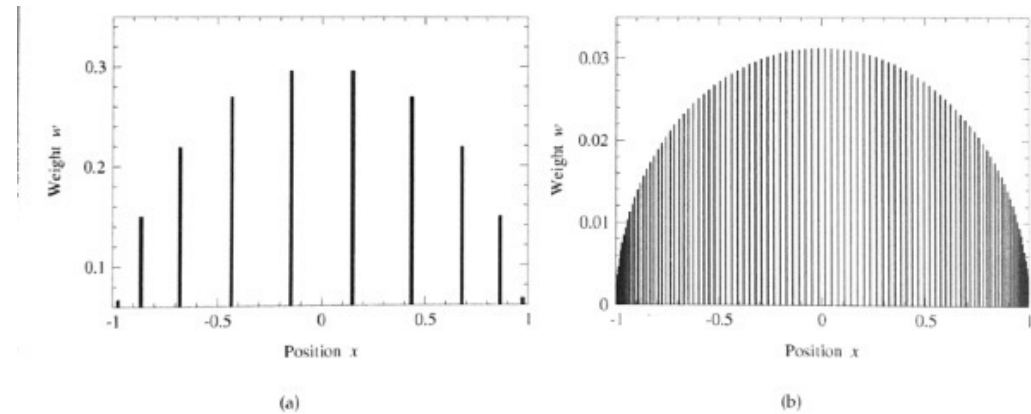


Figure 5.4: Sample points and weights for Gaussian quadrature. The positions and heights of the bars represent the sample points and their associated weights for Gaussian quadrature with (a)  $N = 10$  and (b)  $N = 100$ .

- From Garcia 10.3:

Table 10.7: Grid points and weights for Gauss-Legendre integration.

$\pm x_i$	$w_i$	$\pm x_i$	$w_i$
$N = 2$		$N = 8$	
0.5773502692	1.0000000000	0.1834346425	0.3626837834
$N = 3$		0.5255324099	0.3137066459
0.0000000000	0.8888888889	0.7966664774	0.2223810345
0.7745966692	0.5555555556	0.9602898565	0.1012285363
$N = 4$		$N = 12$	
0.3399810436	0.6521451549	0.1252334085	0.2491470458
0.8611363116	0.3478548451	0.3678314990	0.2334925365
$N = 5$		0.5873179543	0.2031674267
0.0000000000	0.5688888889	0.7699026742	0.1600783285
0.5384693101	0.4786286705	0.9041172564	0.1069393260
0.9061798459	0.2369268850	0.9815606342	0.0471753364

# Review: Choosing an integration method

(Newman Sec. 5.7)

- Trapezoid method:
  - Trivial to program
  - Equally spaced points, often true of experimental data
  - Good choice for poorly behaved data (noisy, singularities)
  - Adaptive method gives guaranteed accuracy level
  - **Not very accurate for given number of points**
- Romberg integration:
  - Equally spaced points, often true of experimental data
  - Guaranteed accuracy level
  - Potentially high accuracy for small number of points
  - **Will not work well for noisy or pathological data/integrands**
- Gaussian Quadrature
  - Potentially high accuracy for small number of points
  - Simple to program (weights and roots tabulated)
  - **Will not work well for noisy or pathological data/integrands**
  - **Need to have data on specific, unequally-spaced grid**

# Review: Lagrange interpolation

- General method for building a single polynomial that goes through all the points (alternate formulations exist)

- Given  $n$  points:  $x_0, x_1, \dots, x_{n-1}$ , with associated function values:  $f_0, f_1, \dots, f_{n-1}$

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

- Note basis function  $l_i$  is 0 at all  $x_j$  except for  $x_i$  (where it is one)

- Function value at  $x$  is: 
$$f(x) = \sum_{i=0}^{n-1} l_i(x) f_i$$

# Review: Quadratic Lagrange polynomial

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

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

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

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

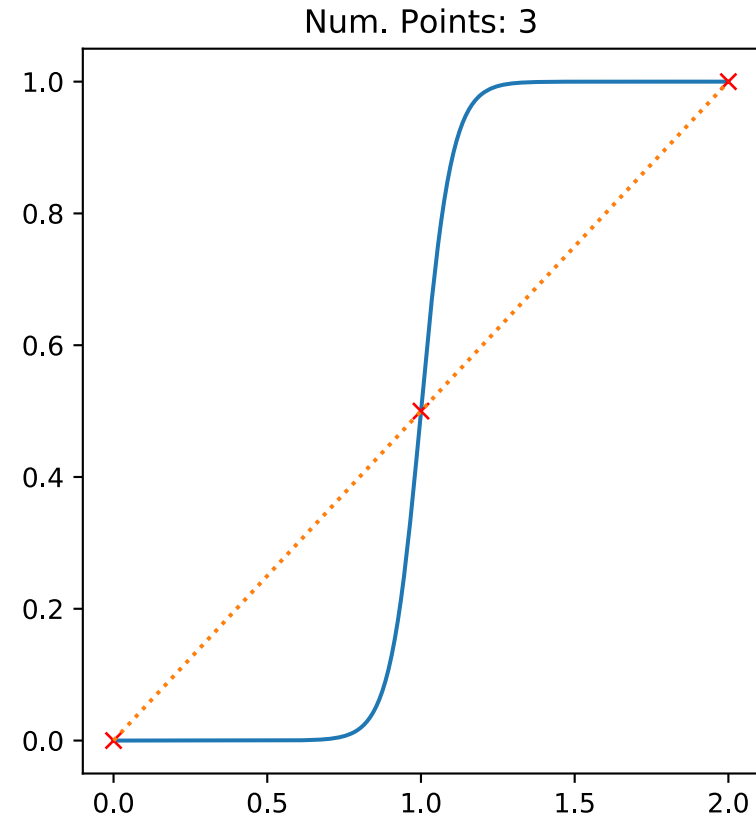
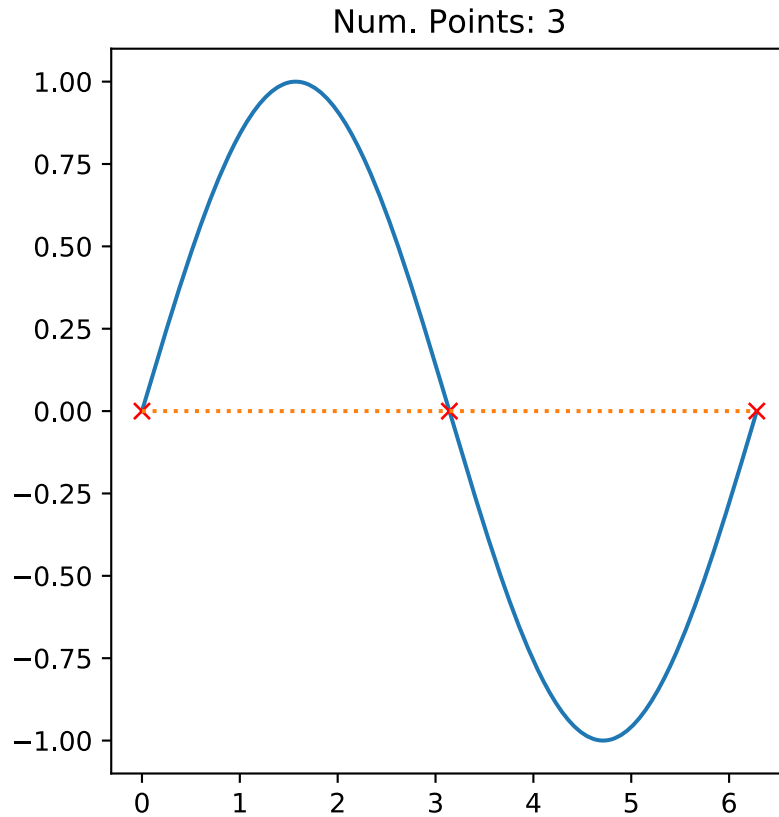
- Polynomial:

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

# Today's lecture:

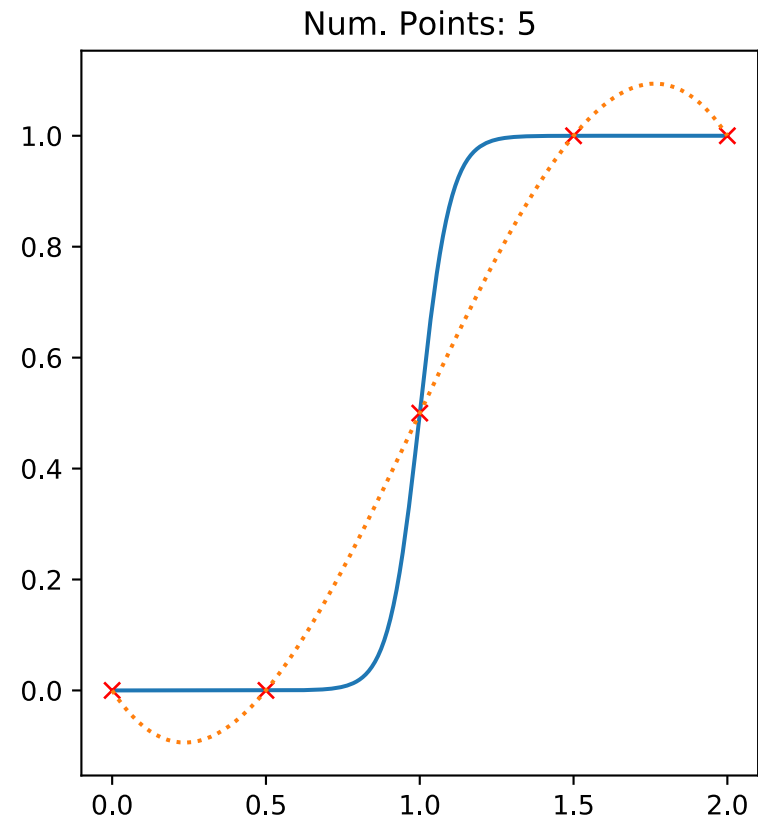
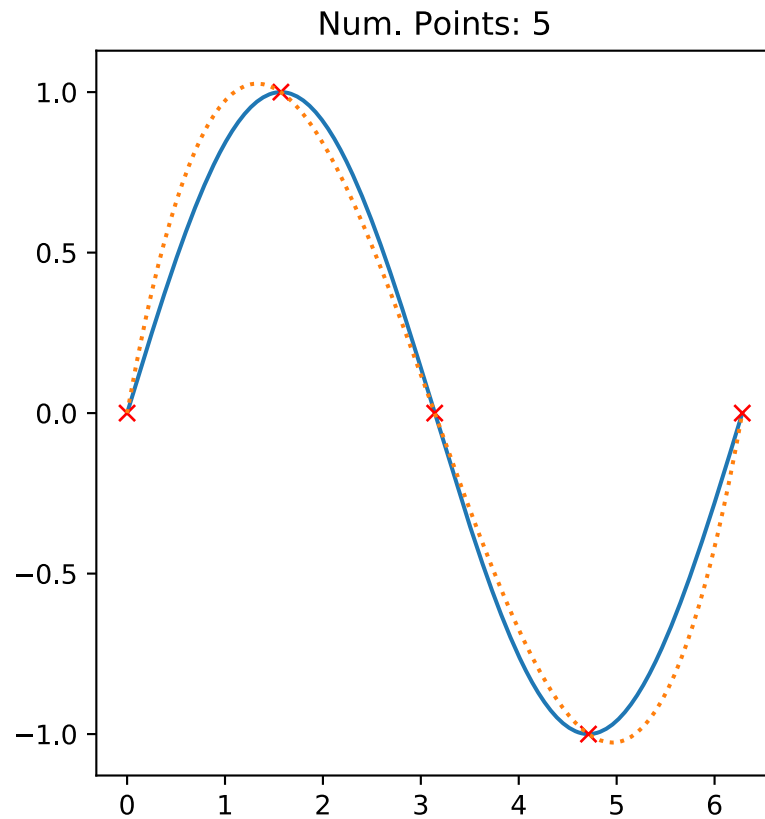
- Continue discussing interpolation
  - Lagrange Interpolation
  - Cubic splines
- Begin discussing finding roots of functions
  - Bisection method
  - Newton Raphson method
  - Secant method

# Example: Lagrange Interpolation of two functions

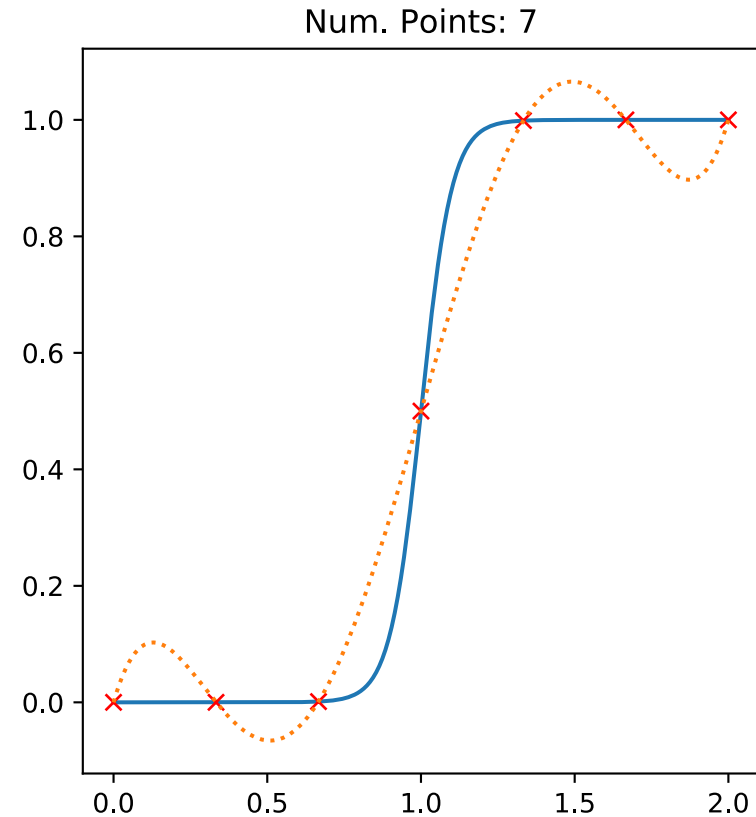
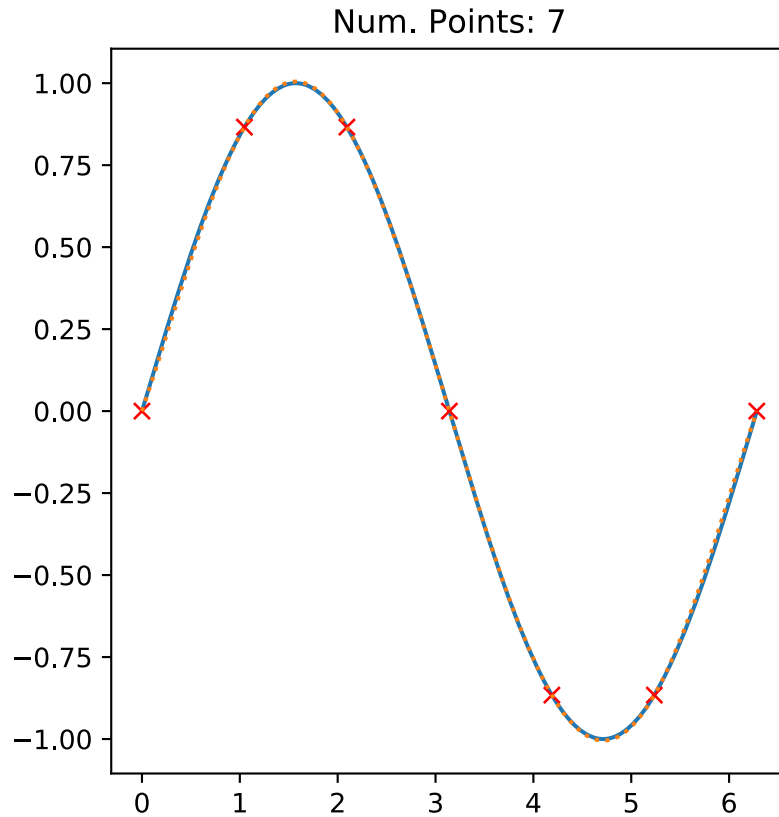




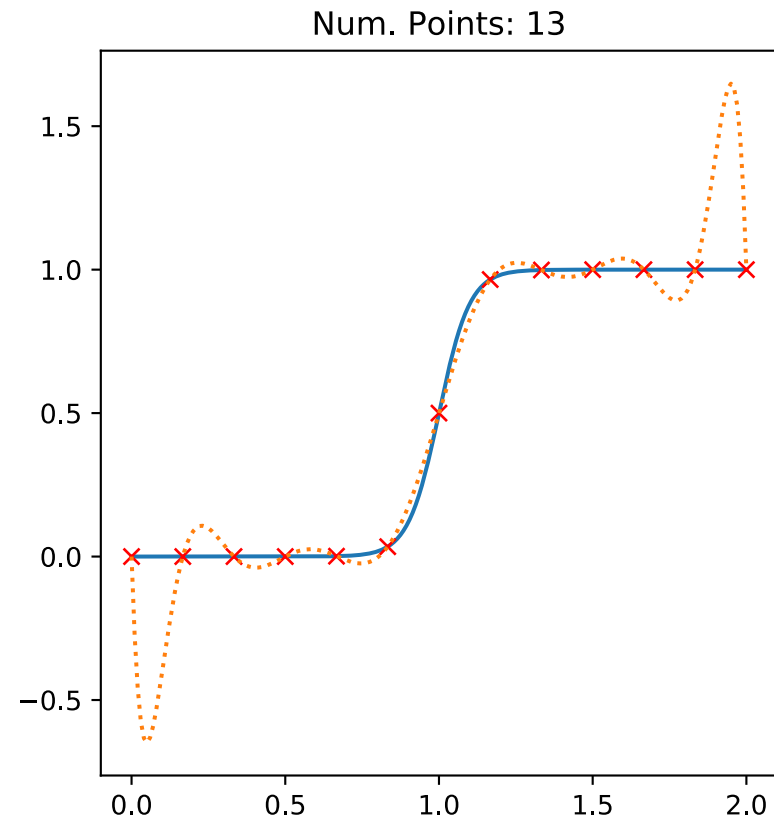
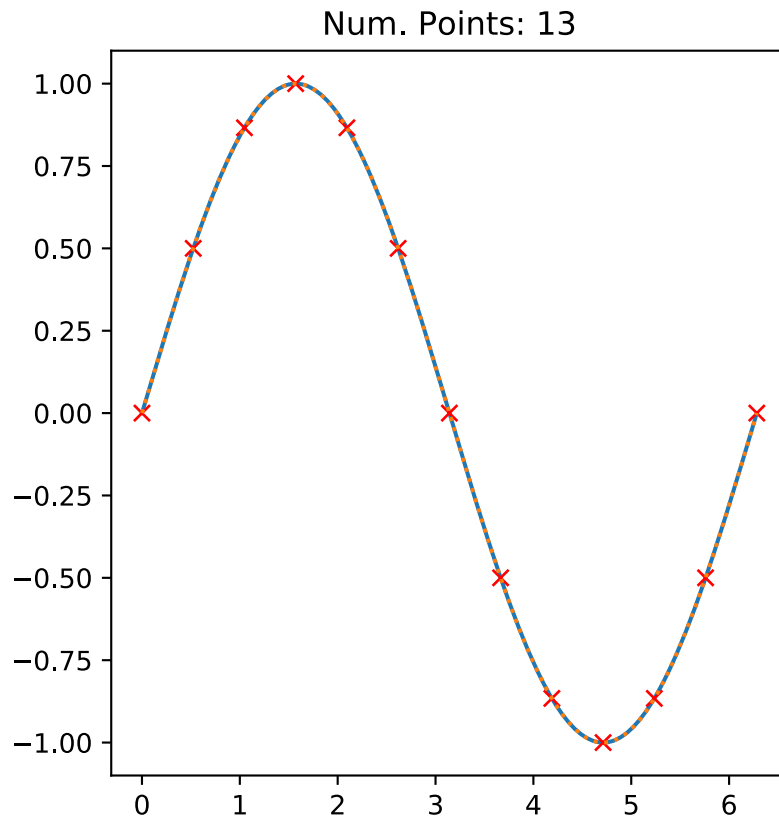
# Example: Lagrange Interpolation of two functions



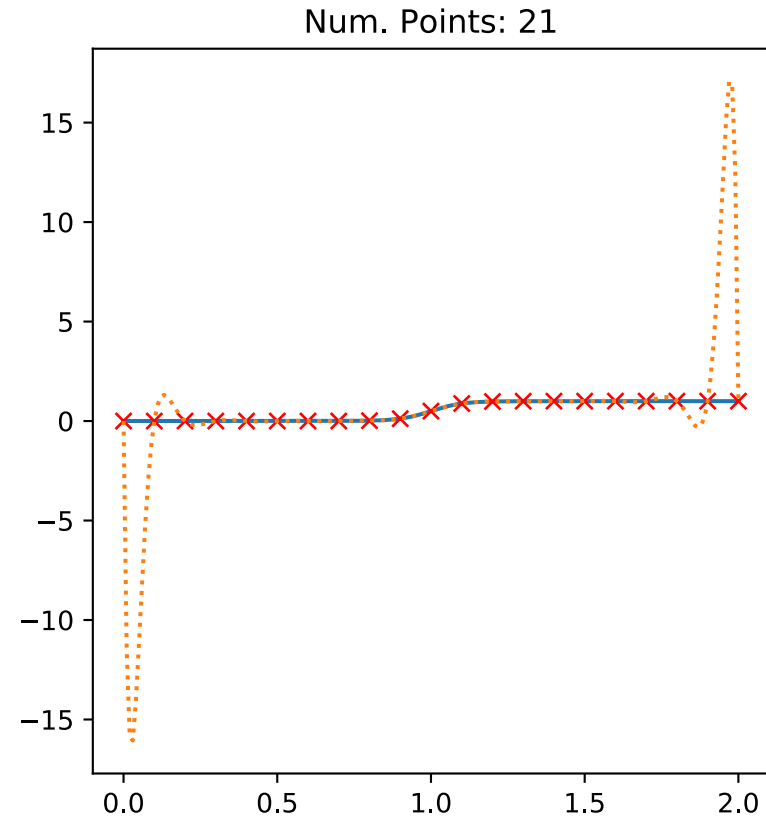
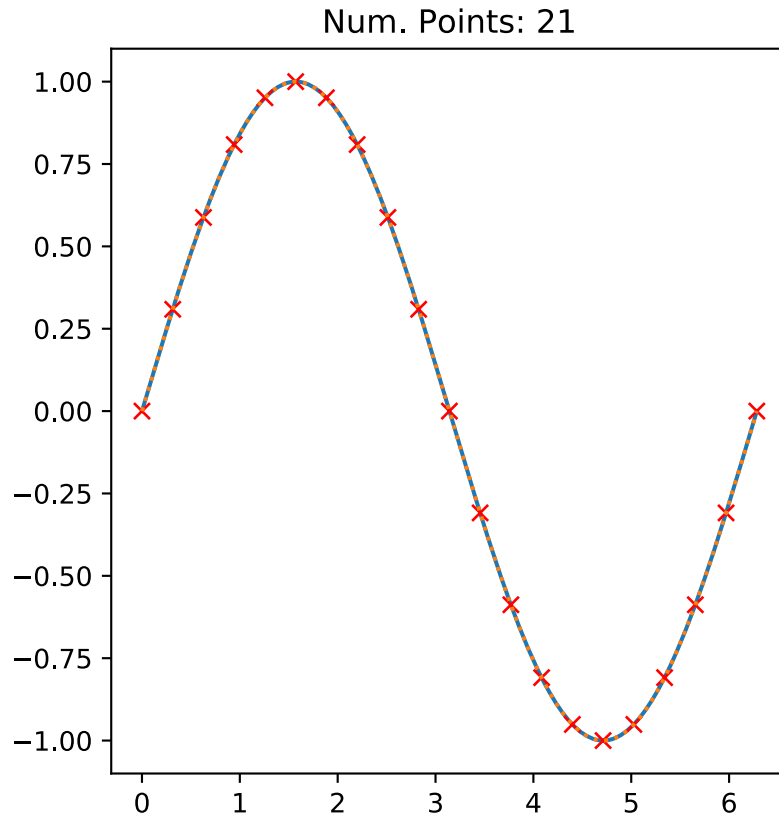
# Example: Lagrange Interpolation of two functions



# Example: Lagrange Interpolation of two functions



# Example: Lagrange Interpolation of two functions

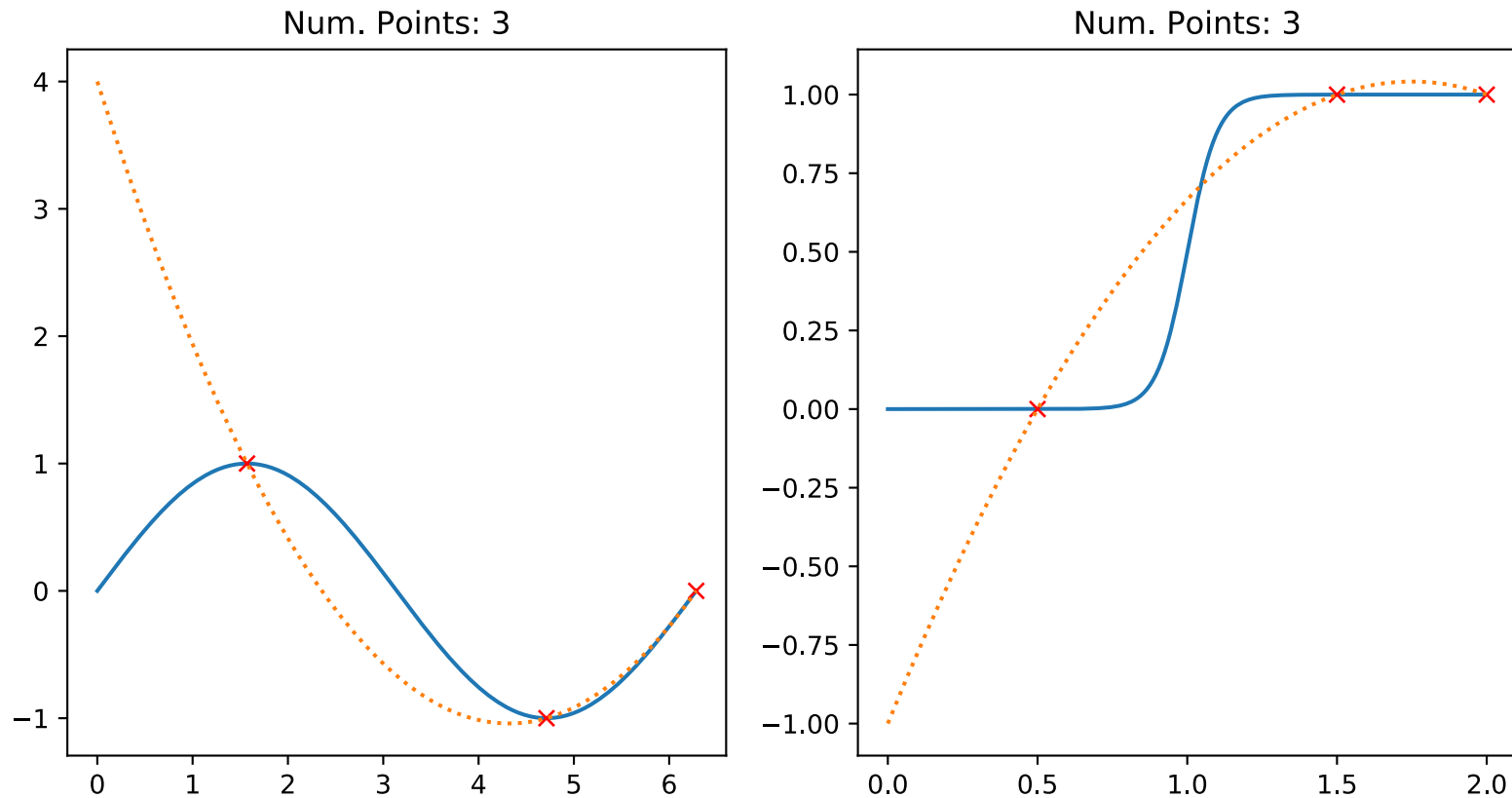


# Example: Lagrange Interpolation of two functions with Chebyshev spacing

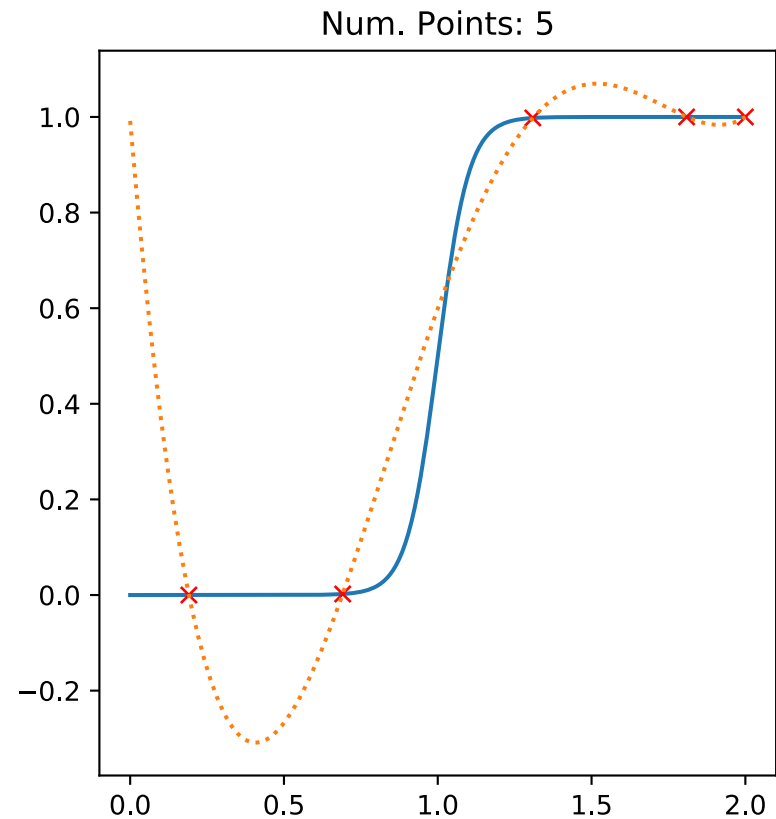
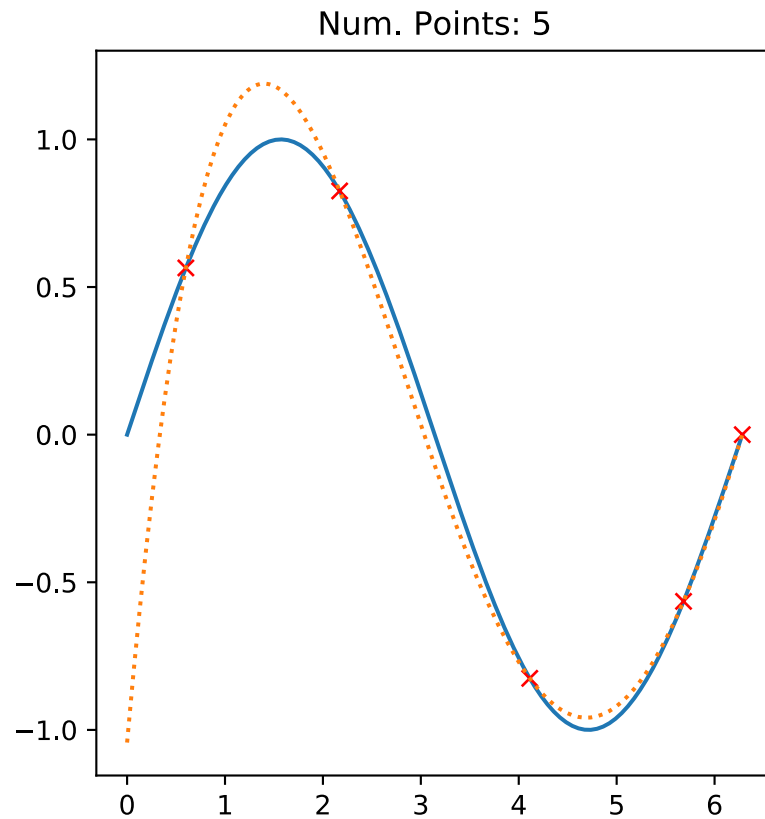
- For the hyperbolic tangent case, increasing the number of points beyond a certain limit increases the error
  - Runge phenomena: Oscillations at the edges of the interval
  - Increasing the number of points causes a divergence in the error
- Can do better by varying the spacing of the interpolating points
  - e.g., Chebyshev polynomial roots are concentrated toward the end of the interval
  - Chebyshev polynomial spacing is usually (almost always) convergent with the number of interpolating points

$$x_k = \frac{1}{2}(a + b) + \frac{1}{2}(b - a) \cos \left( \frac{2k - 1}{2n} \pi \right), \quad k = 1, \dots, n$$

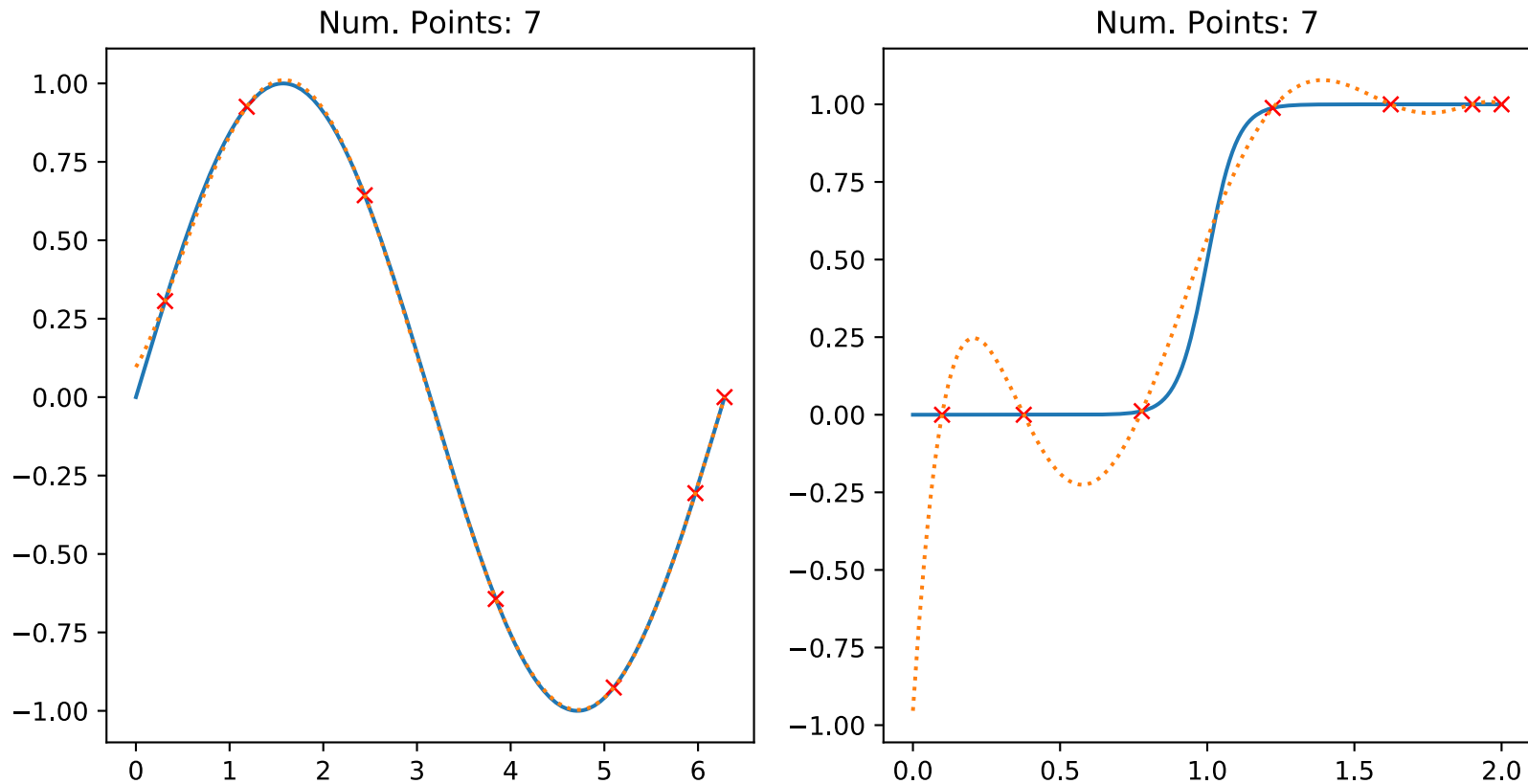
# Example: Lagrange Interpolation of two functions with Chebyshev nodes



# Example: Lagrange Interpolation of two functions with Chebyshev nodes

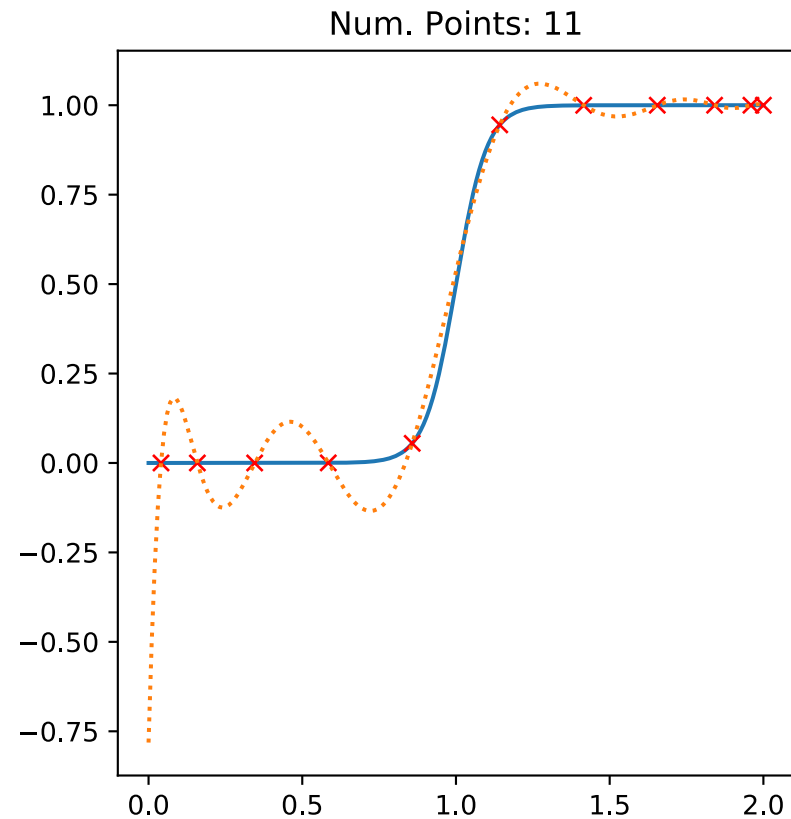
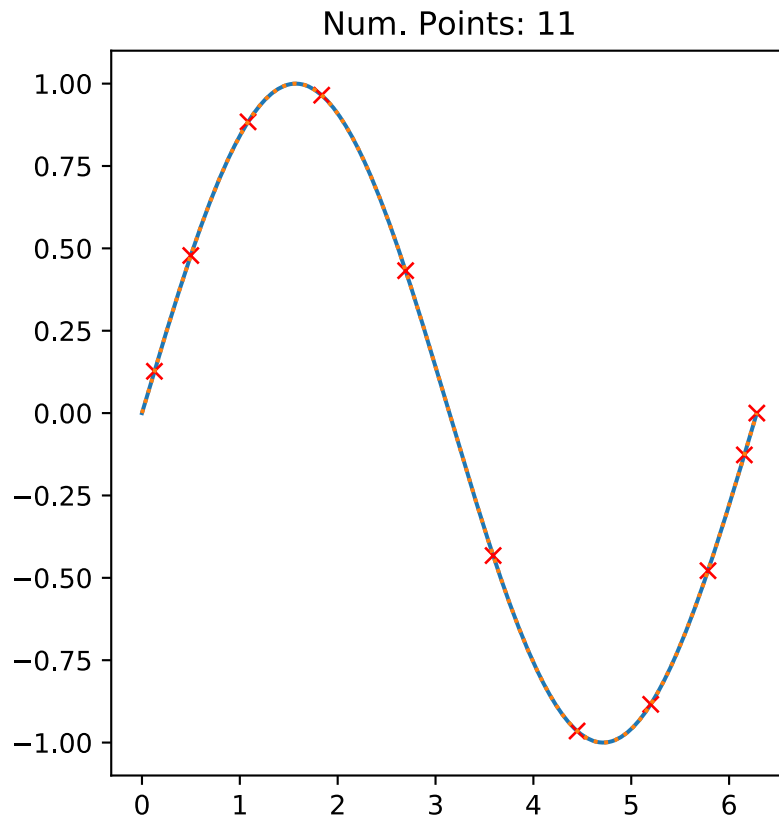


# Example: Lagrange Interpolation of two functions with Chebyshev nodes

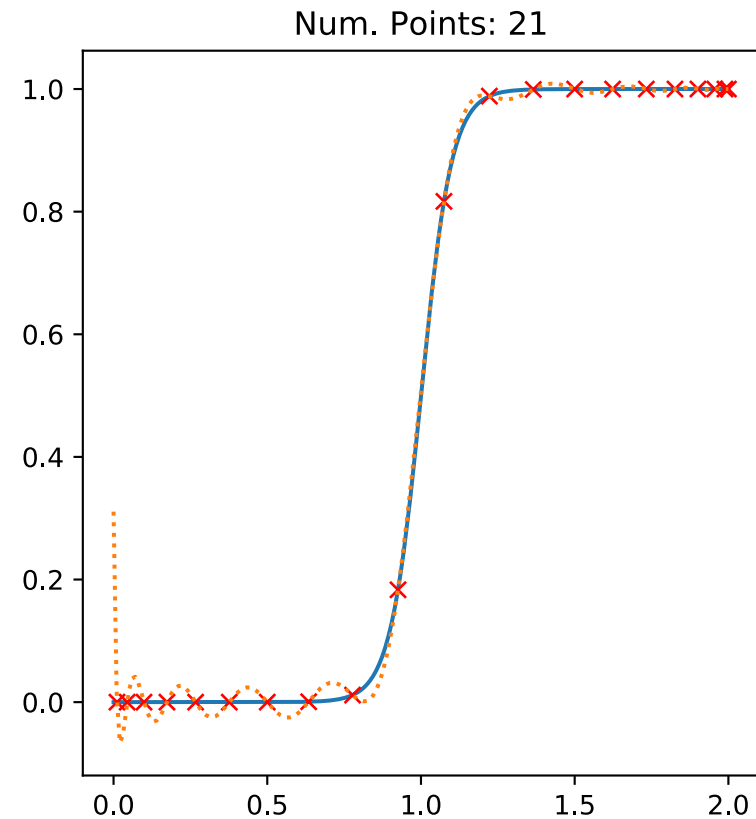
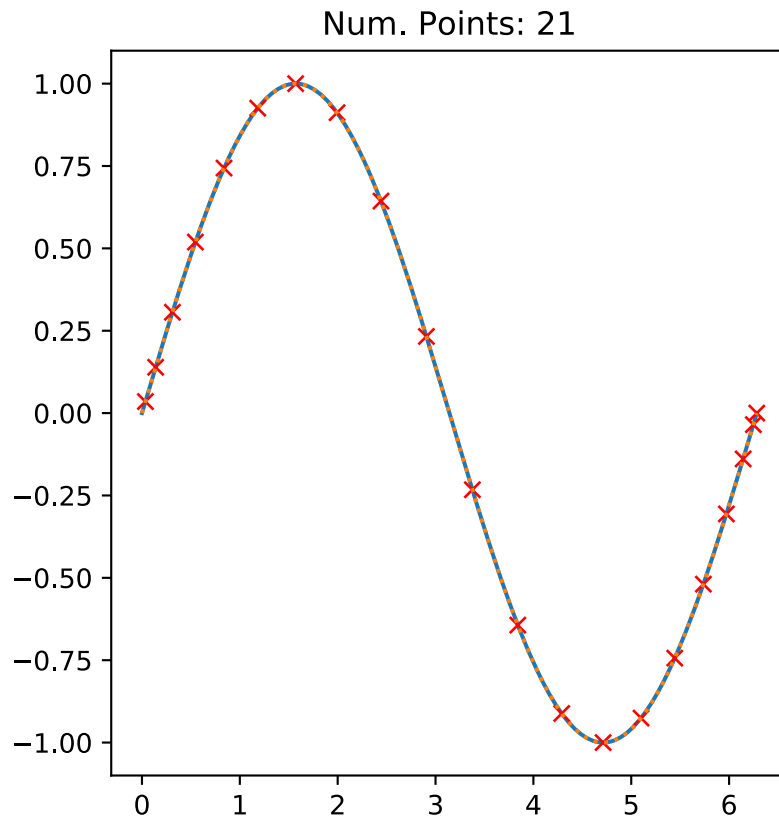




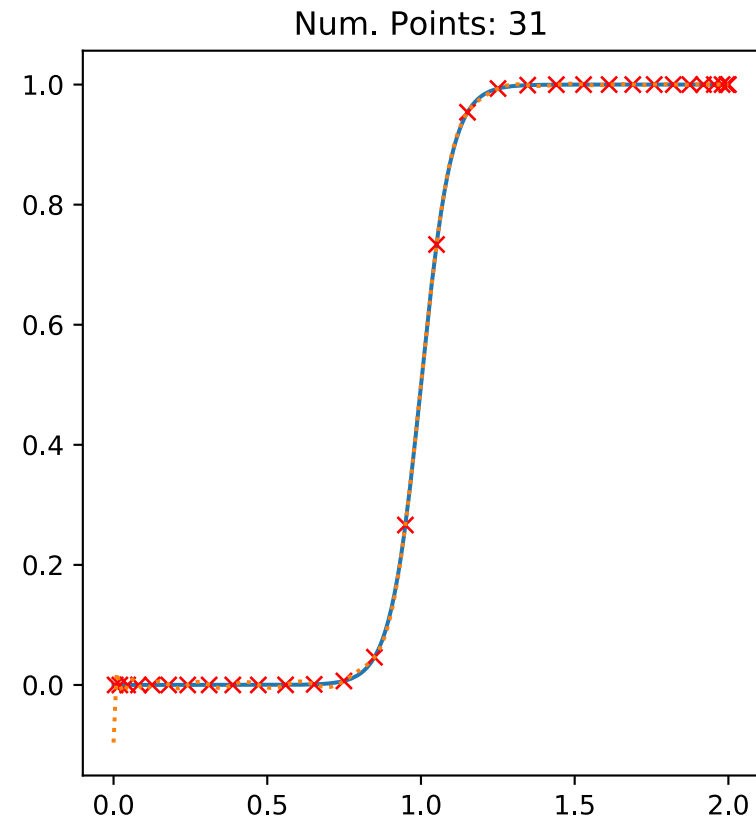
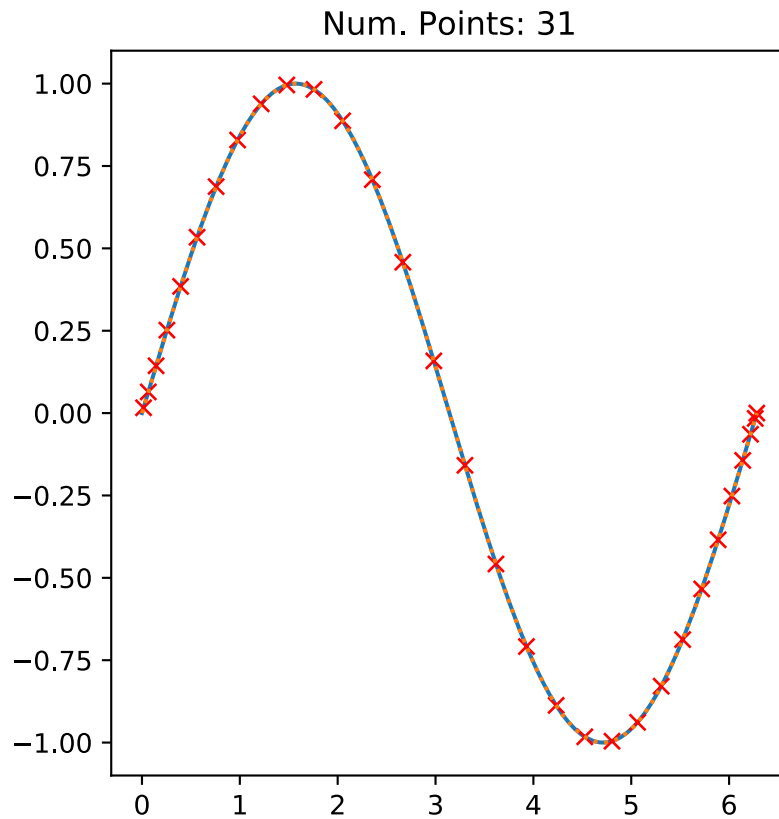
# Example: Lagrange Interpolation of two functions with Chebyshev nodes



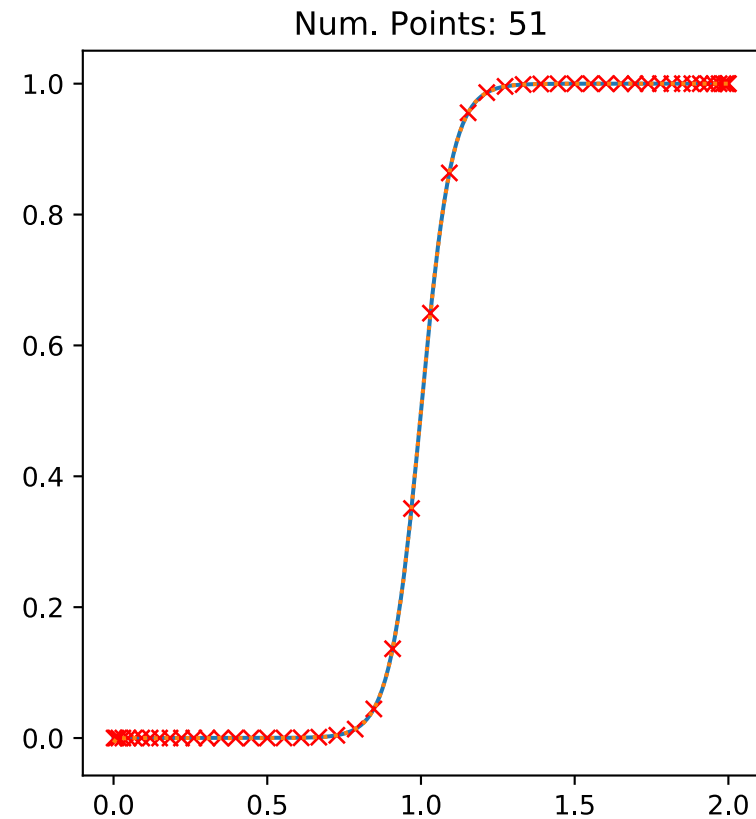
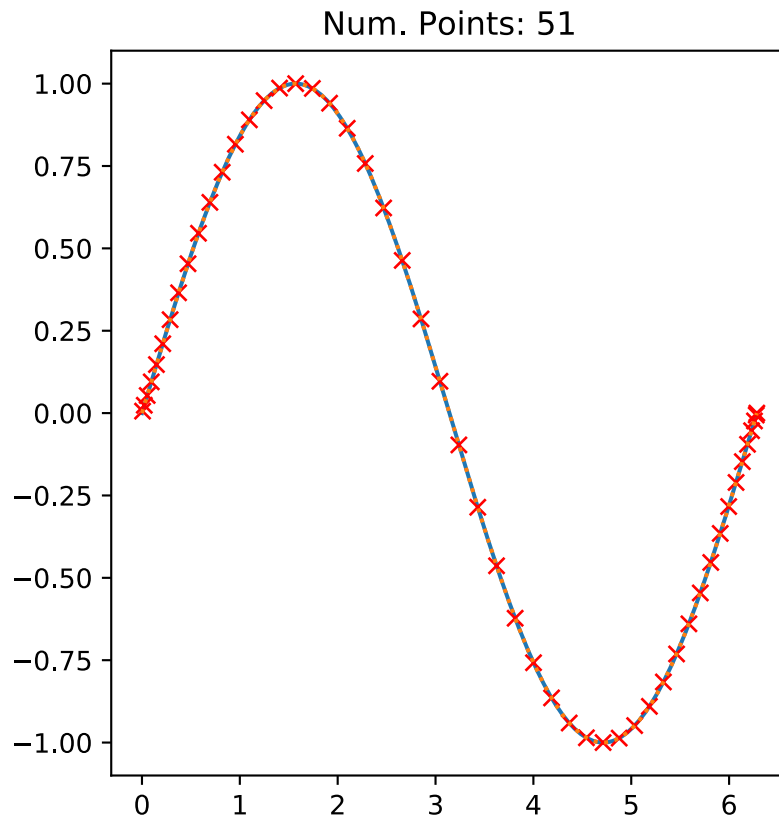
# Example: Lagrange Interpolation of two functions with Chebyshev nodes



# Example: Lagrange Interpolation of two functions with Chebyshev nodes



# Example: Lagrange Interpolation of two functions with Chebyshev nodes



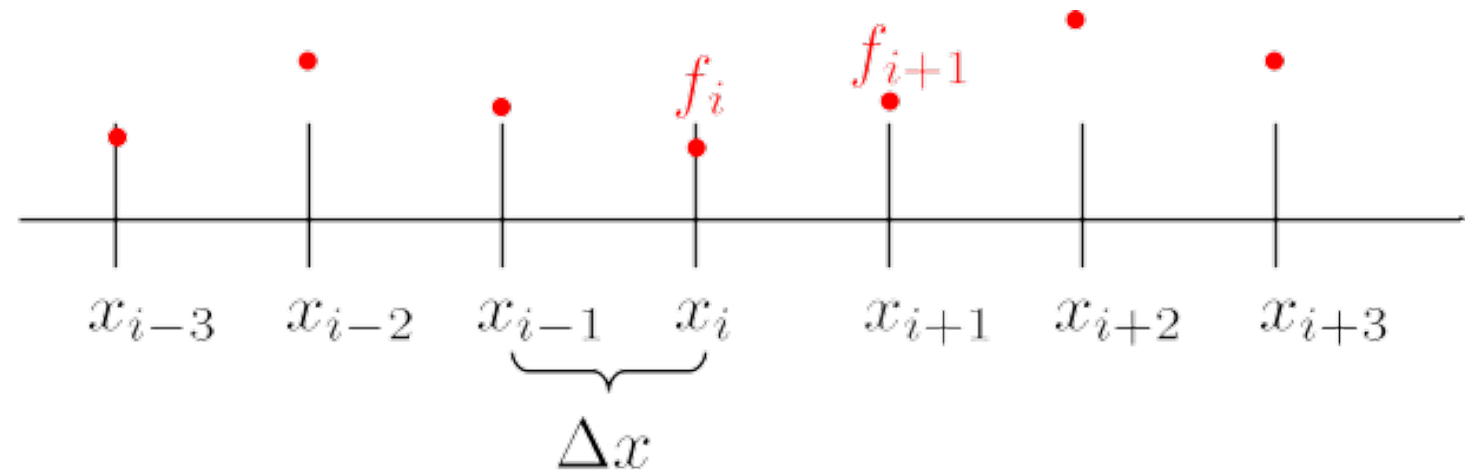
# Today's lecture:

- Continue discussing interpolation
  - Lagrange Interpolation
  - Cubic splines
- Begin discussing finding roots of functions
  - Bisection method
  - Newton Raphson method
  - Secant method

# Splines (Pang Sec. 2.4)

- 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 regular-spaced discrete data:  $f_i = f(x_i)$  at  $x_0, x_1, x_2, \dots, x_n$
- $m$ -th order polynomial to approximate  $f(x)$  for  $x$  in  $[x_i, x_{i+1}]$ :

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

- Coefficients chosen so  $p_i(x_i) = f_i$  and from smoothness condition: all derivatives ( $l$ ) match at the endpoints

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

- Except for points on the boundary of the curve

# Splines: Determining the coefficients

- There are  $n$  intervals; in each interval:  $m+1$  coefficients for the polynomial
- Total:  $(m+1)n$  coefficients:
  - Smoothness condition on interior points:  $(m)(n-1)$  equations
  - Curve passing through interior points:  $(n-1)$  equations
  - Remaining  $m+1$  equations from imposing conditions on derivatives at end points
    - Natural spline: Setting highest-order derivative to zero at both endpoints



# Most popular: Cubic splines, $m = 3$

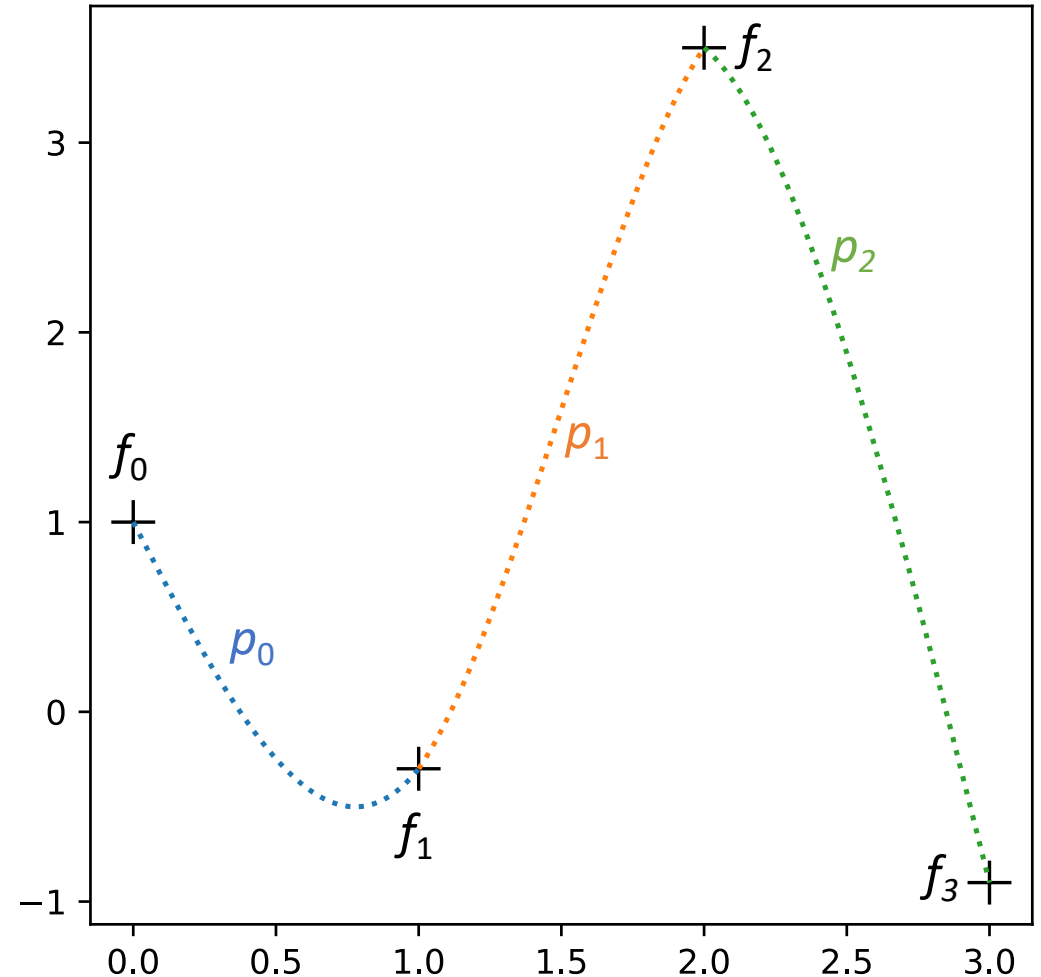
- Easy to implement
- Produce a curve that appears to be seamless
- Avoids distortions near the edges
- Only piecewise continuous, third derivatives are discontinuous

# Cubic spline example: 3 intervals

- Order:  $m=3$ , intervals:  $n=3$ , points:  $x = 0, 1, 2, 3$
- Constraints:  $(m+1)n = 12$

- Interior point 1:  $p_0(x_1) = f_1$   
 $p_1(x_1) = f_1$   
 $p'_0(x_1) = p'_1(x_1)$   
 $p''_0(x_1) = p''_1(x_1)$

- Interior point 2:  $p_1(x_2) = f_2$   
 $p_2(x_2) = f_2$   
 $p'_1(x_2) = p'_2(x_2)$   
 $p''_1(x_2) = p''_2(x_2)$



# Cubic spline example: 3 intervals

- At the boundaries:

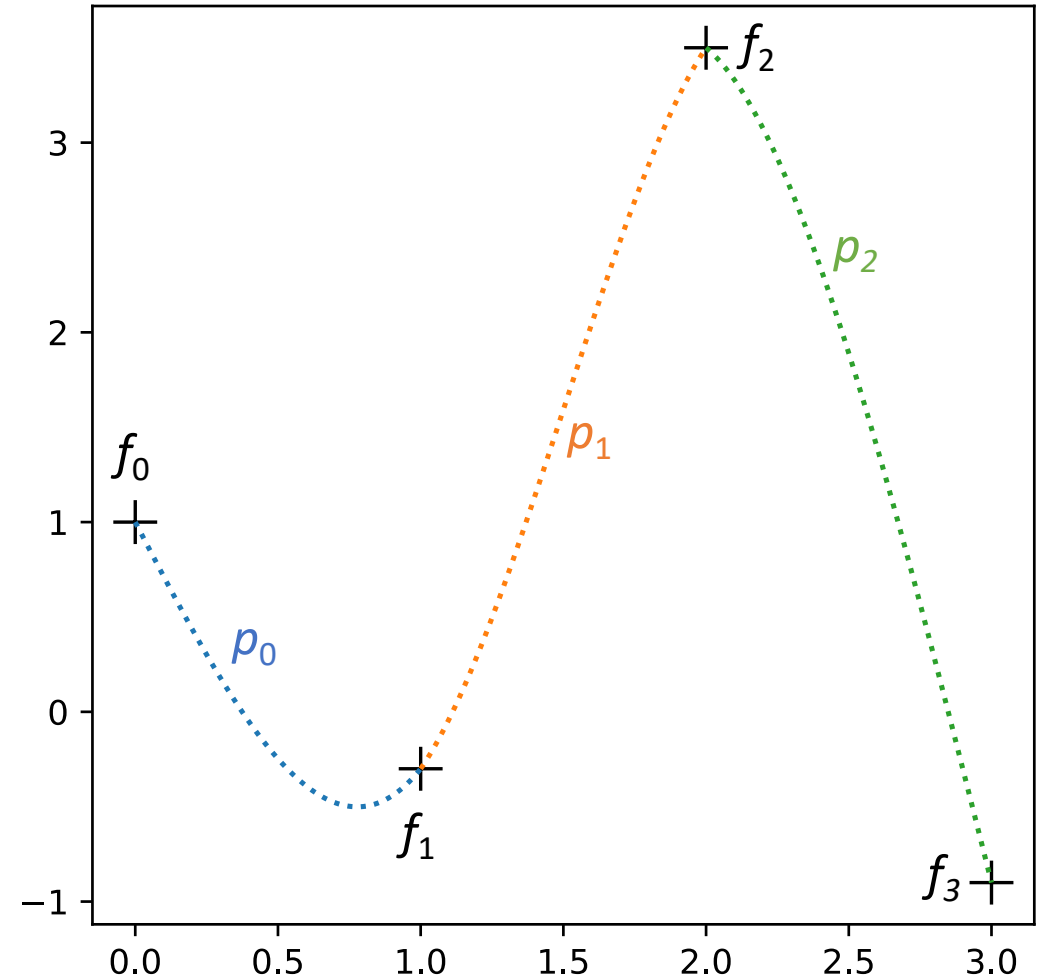
$$p_0(x_0) = f_0$$

$$p_2(x_3) = f_3$$

- Natural spline, second derivatives at the boundary set to zero

$$p_0''(x_0) = 0$$

$$p_2''(x_3) = 0$$



# Now solve for the coefficients:

- Linearly interpolate the second derivative:

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

- Integrate twice:

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

- Impose constraints:  $p_i(x_i) = f_i$ ,  $p(x_{i+1}) = f_{i+1}$

Now solve for the coefficients:

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

• Results:

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

• For now, in terms of second derivative

• To get second derivative, use continuity condition

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

Now solve for the coefficients:

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

- Applies to all interior points
- Natural boundary conditions:

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

- Results in a system of linear equations

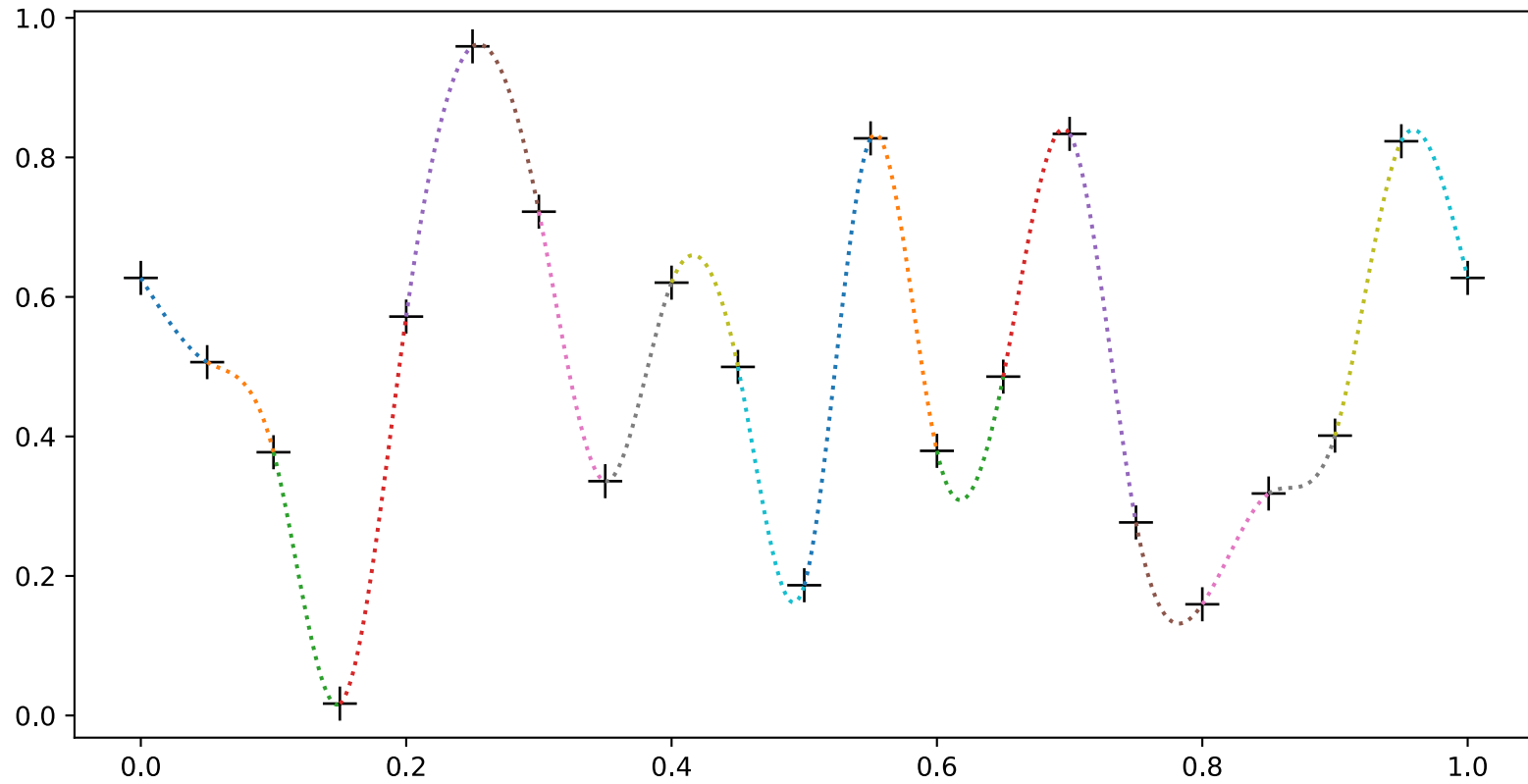
# Results in system of linear equations

- Can be written as a tridiagonal matrix:

$$\begin{pmatrix} 4\Delta x & \Delta x & & & & \\ \Delta x & 4\Delta x & \Delta x & & & \\ & \Delta x & 4\Delta x & \Delta x & & \\ & & \ddots & \ddots & \ddots & \\ & & & \Delta x & 4\Delta x & \Delta x \\ & & & & \Delta x & 4\Delta x \end{pmatrix} \begin{pmatrix} p_1'' \\ p_2'' \\ p_3'' \\ \vdots \\ p_{n-2}'' \\ p_{n-1}'' \end{pmatrix} = \frac{6}{\Delta x} \begin{pmatrix} f_0 - 2f_1 + f_2 \\ f_1 - 2f_2 + f_3 \\ f_2 - 2f_3 + f_4 \\ \vdots \\ f_{n-3} - 2f_{n-2} + f_{n-1} \\ f_{n-2} - 2f_{n-1} + f_n \end{pmatrix}$$

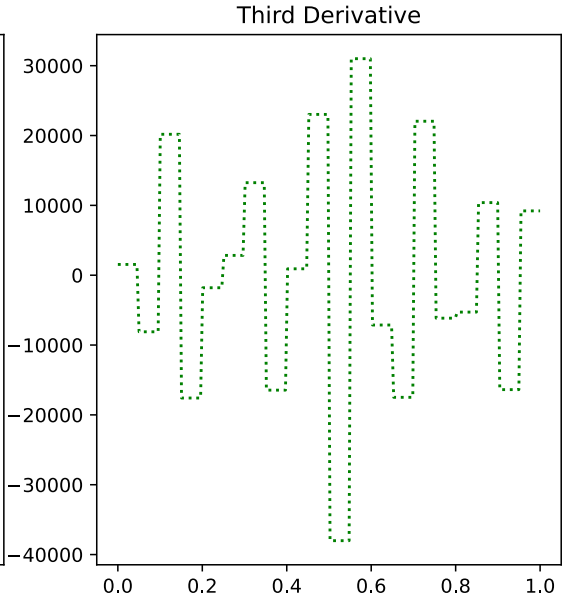
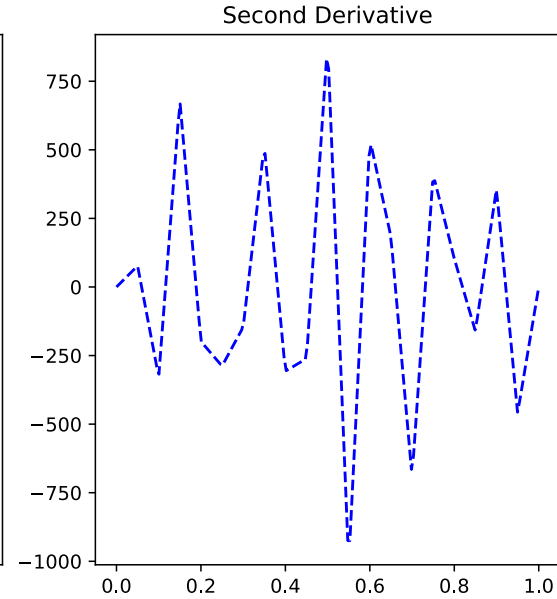
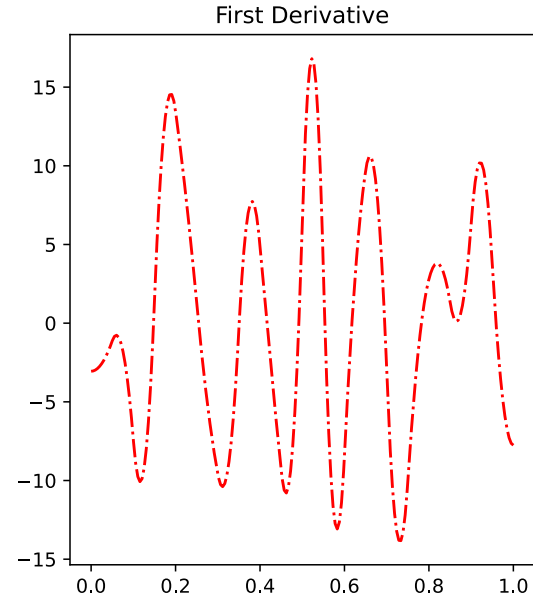
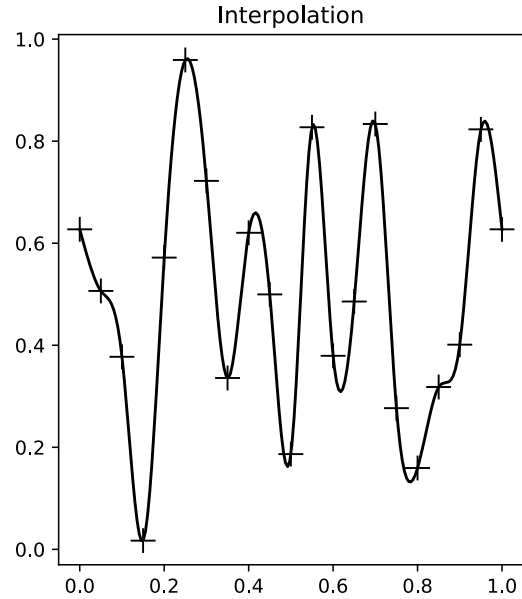
- We will discuss linear algebra in a later class

# Example: Cubic spline for random numbers





# Example: Derivatives of cubic splines



# Today's lecture:

- Continue discussing interpolation
  - Lagrange Interpolation
  - Cubic splines
- Begin discussing finding roots of functions
  - Bisection method
  - Newton Raphson method
  - Secant method

# Purpose: Find the root of a function

- Root of a function  $f(x)$  is  $x_r$  such that:  $f(x_r) = 0$
- Why? We can cast more general solutions in the form of finding roots.

- Example: Suppose I have the following equation for velocity of a free-falling mass  $m$  with a coefficient of drag  $c_d$ :

$$v(t) = \sqrt{\frac{gm}{c_d}} \tanh \left( \sqrt{\frac{gc_d}{m}} t \right)$$

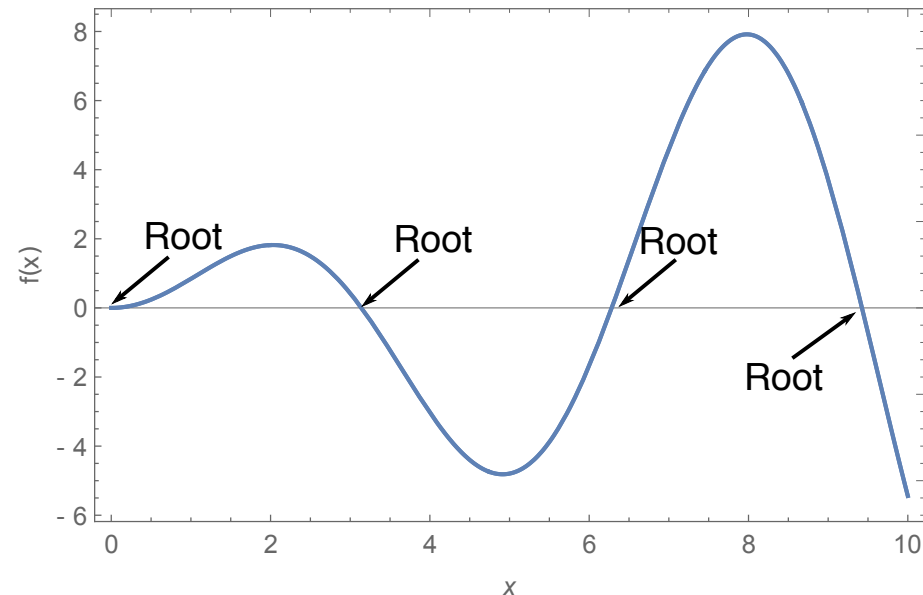
- I would like to find the mass that would give me a velocity of 36 m/s after 4s of free fall. We can do this by rewriting the equation as:

$$f(m) = \sqrt{\frac{gm}{c_d}} \tanh \left( \sqrt{\frac{gc_d}{m}} t \right) - v(t)$$

- And finding the root of  $f(m)$  for  $t = 4s$  and  $v = 36$  m/s

# Purpose: Find the root of a function

- For very simple functions, we can find the root analytically
  - For more complicated functions, we must do this numerically
- First rule of root finding: If possible, plot the function to get an idea of where roots are, how many, etc.:



# Bisection method

- 1. Choose **two initial guesses** for the root, a lower ( $x_l$ ) and upper ( $x_u$ )
  - Chosen such that the function evaluated at  $x_l$  and  $x_u$  have different signs
  - This can be checked by ensuring that:  $f(x_l) f(x_u) < 0$

- 2. An estimate for the root is determined as the **midpoint between the guesses**

$$x_r = \frac{x_l + x_u}{2}$$

- 3. Make the following evaluations to determine in **which subinterval the root lies**, and thus obtain a refined guess:
  - If  $f(x_l) f(x_r) < 0$ , set  $x_u = x_r$ , return to step 2
  - If  $f(x_l) f(x_r) > 0$ , set  $x_l = x_r$ , return to step 2
  - If  $f(x_l) f(x_r) = 0$  to some tolerance,  $x_r$  is the root and the calculation is complete

# Newton-Raphson method

- Let  $x_r$  be a root of  $f(x)$ . Expand  $f(x)$  in a Taylor series about around a **different** point  $x_0$  that is close to  $x_r$ :

$$f(x) \simeq f(x_0) + f'(x_0)(x - x_0)$$

- Then:

$$f(x_r) = 0 \simeq f(x_0) + f'(x_0)(x_r - x_0)$$

- So:

$$x_r \simeq x_0 - \frac{f(x_0)}{f'(x_0)}$$

- Of course, this is only accurate if  $x_0$  is close to  $x_r$ , but we can use this relation to refine the guess for the root

# Newton-Raphson method procedure

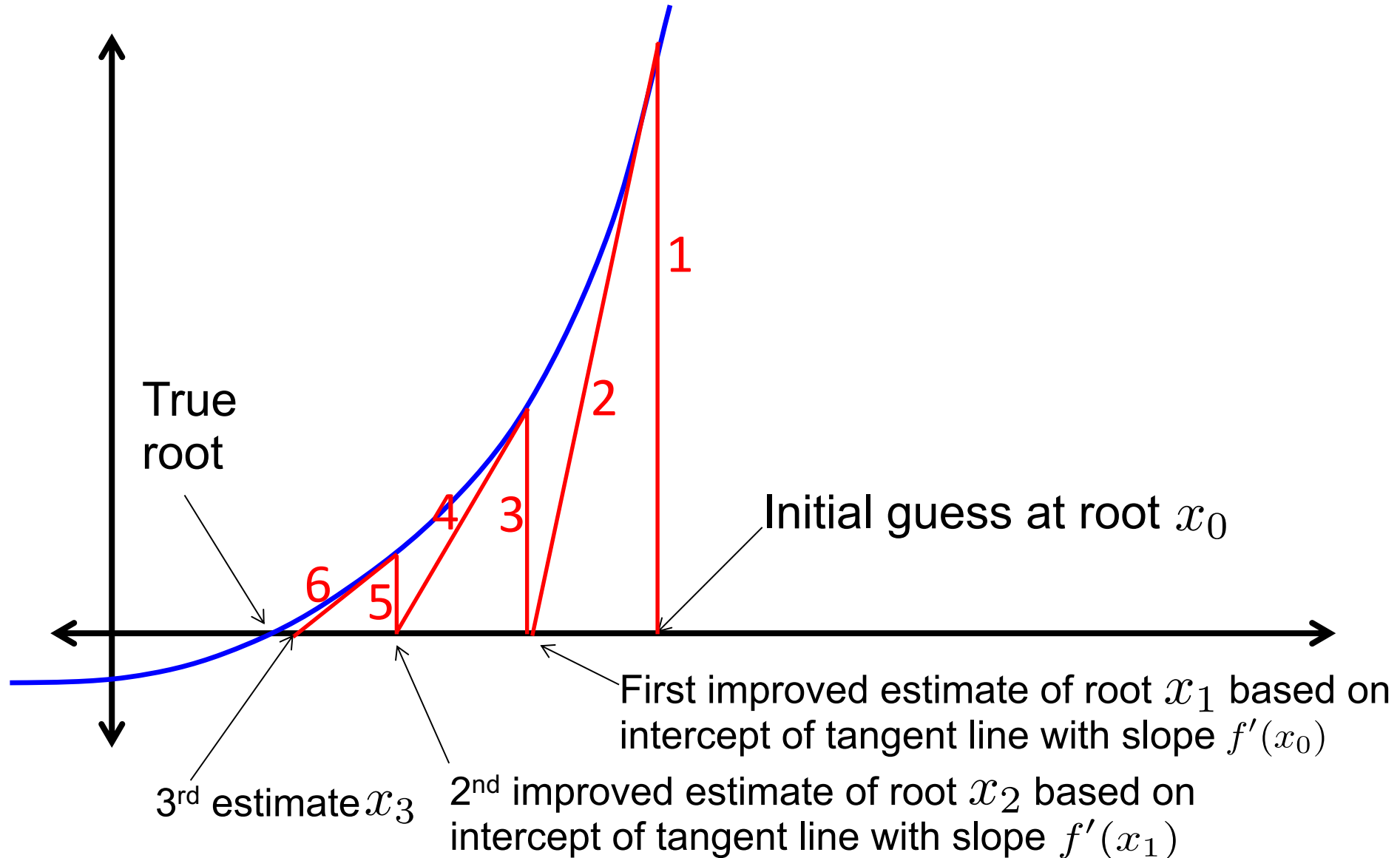
- 1. Make an **initial guess** for the root:  $x_0$
- 2. Use the Taylor series expansion to **find a better estimate** of the root:

$$x_1 \simeq x_0 - \frac{f(x_0)}{f'(x_0)}$$

- 3. Use  $x_1$  as an improved estimate at the root and employ the Taylor series expansion again to get a better estimate  $x_2$
- Repeat process until the answer is accurate enough at the  $n$ th estimate:

$$x_n \simeq x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

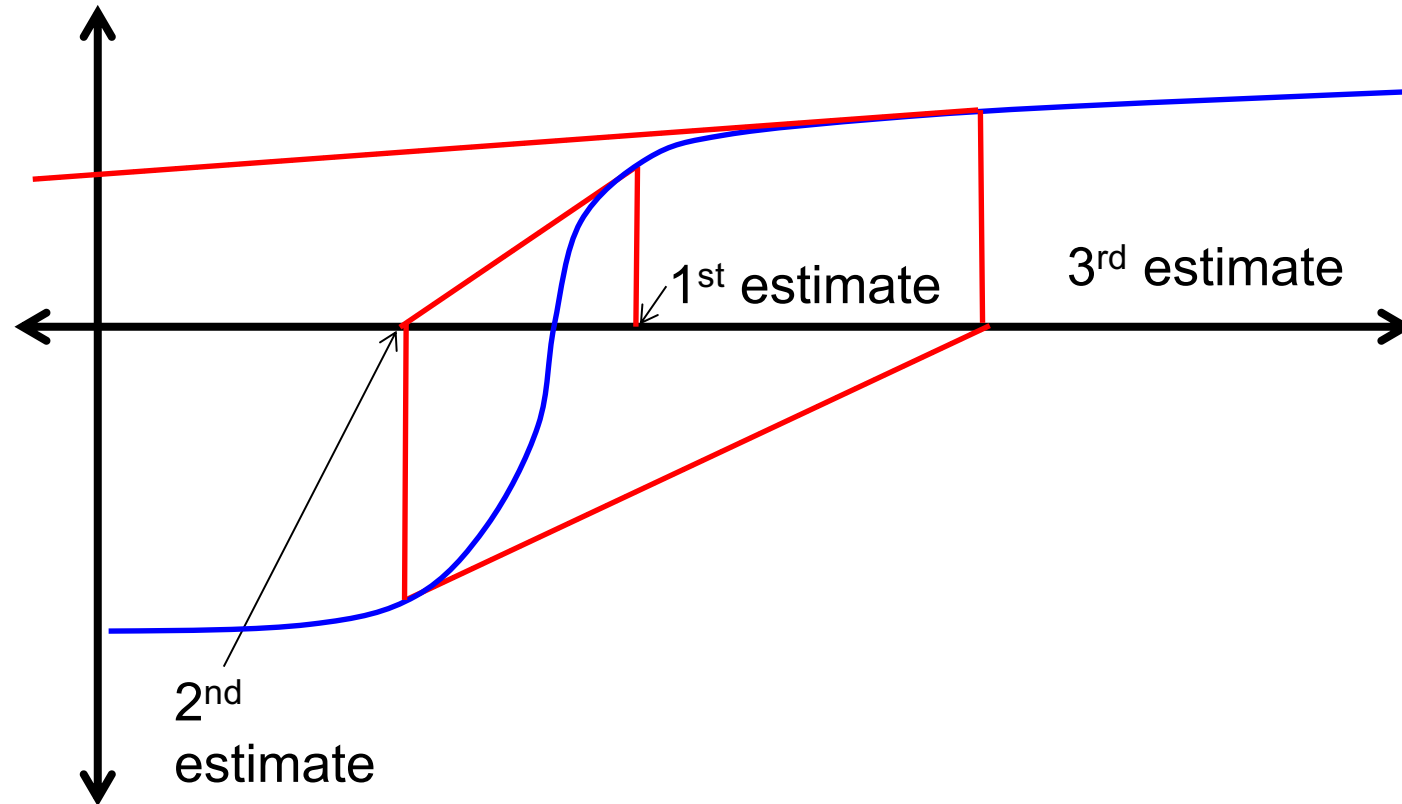
# Geometrical Interpretation of Newton-Raphson Iteration





# Failure of Newton-Raphson

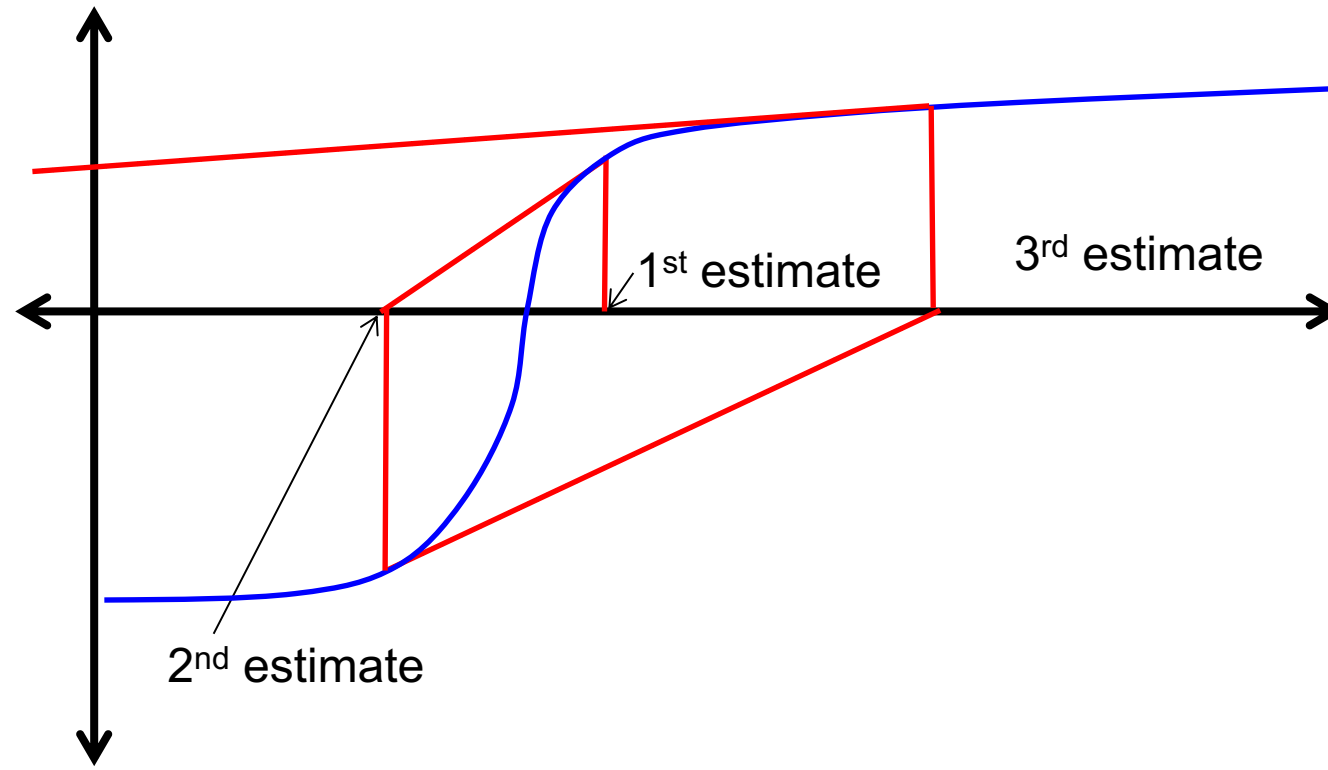
- Example of a simple function that will defeat Newton-Raphson Iteration:



- Each estimate gets further from the true root. Estimates are diverging not converging

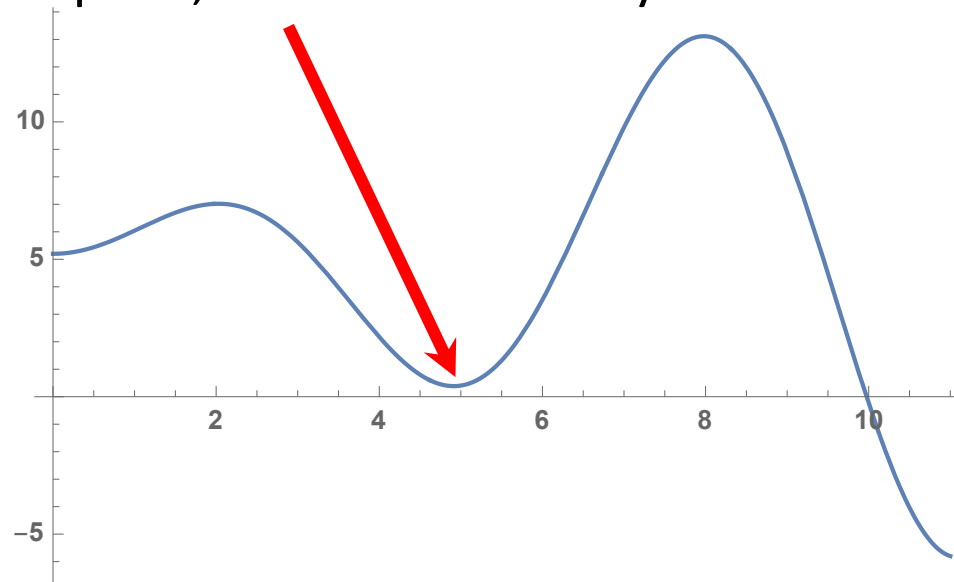
# Stopping criteria for iterations must be chosen carefully

- Could stop when we reach some **maximum number of iterations**
  - Estimate may be no where near the root
  - We can consider this case a failure of the method and warn user about it.



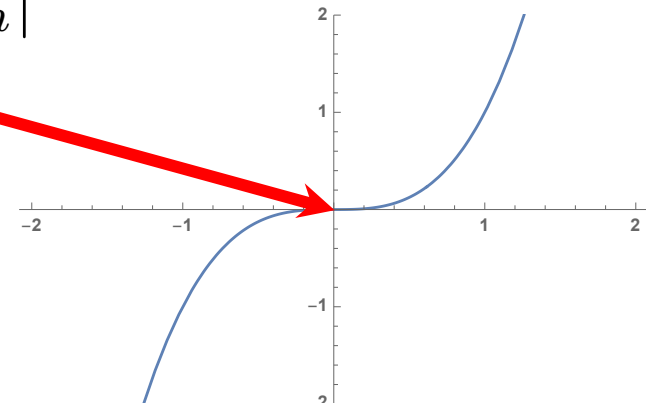
# Stopping criteria for iterations must be chosen carefully

- Could stop when we reach some **maximum number of iterations**
  - Estimate may be no where near the root
  - We can consider this case a failure of the method and warn user about it.
- Could stop when **value of the function** evaluated at the  $n$ th estimate **less than small number** :  $|f(x_n)| < \epsilon$ 
  - But this can be deceptive; final estimate may not be near the root, might just be close to zero



# Stopping criteria for iterations must be chosen carefully

- Could stop when we reach some **maximum number of iterations**
  - Estimate may be nowhere near the root
  - We can consider this case a failure of the method and warn user about it.
- Could stop when **value of the function** evaluated at the  $n$ th estimate **less than small number** :  $|f(x_n)| < \epsilon$ 
  - But this can be deceptive; final estimate may not be near the root, might just be close to zero
- Could stop when **change between estimates becomes small** relative to the current ( $n$ th) estimate:  $|x_{n+1} - x_n| < \epsilon|x_n|$ 
  - Better, but still fails when root is located at zero



# Stopping criteria for iterations must be chosen carefully

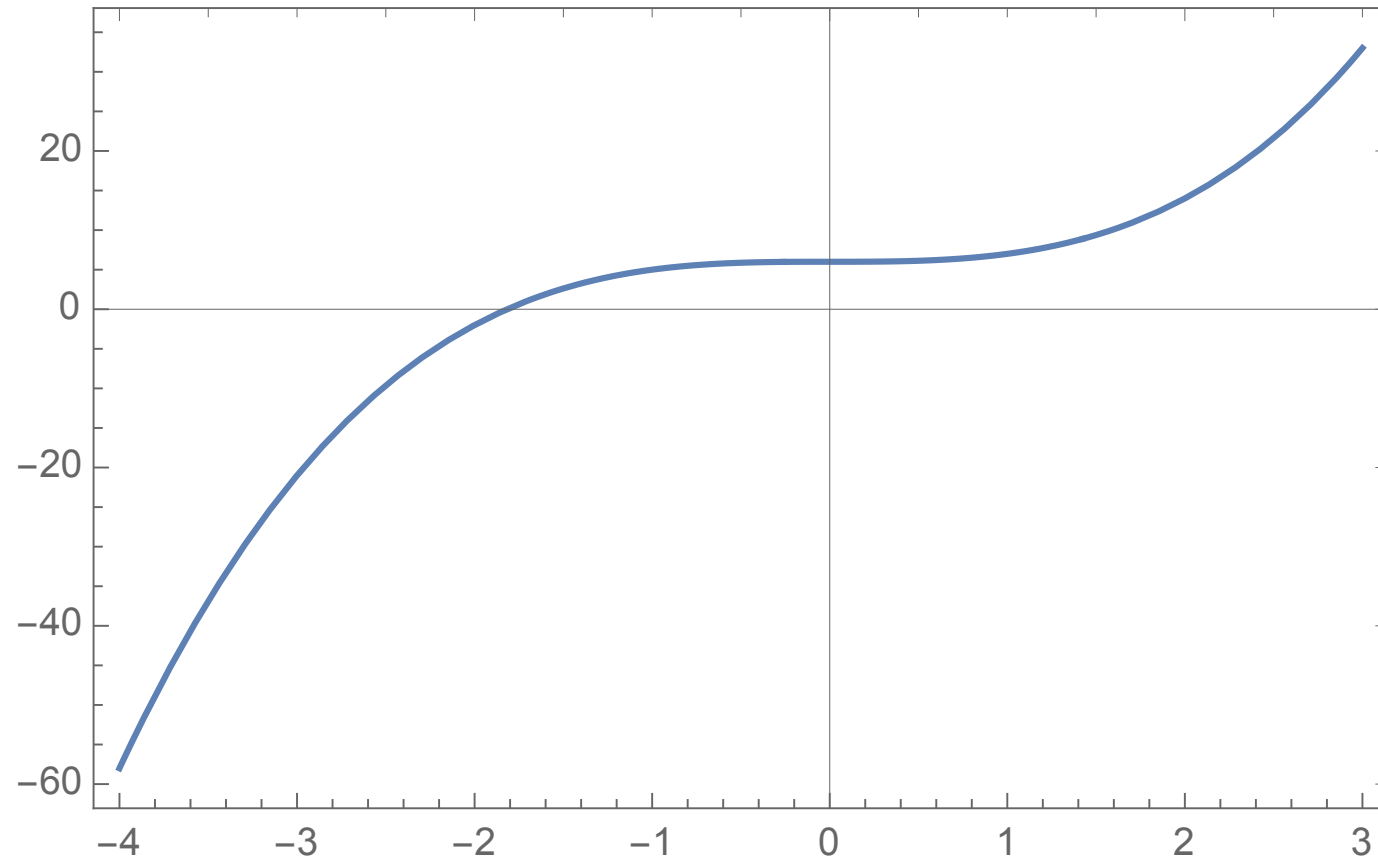
- Could stop when we reach some **maximum number of iterations**
  - Estimate may be nowhere near the root
  - We can consider this case a failure of the method and warn user about it.
- Could stop when **value of the function** evaluated at the  $n$ th estimate **less than small number** :  $|f(x_n)| < \epsilon$ 
  - But this can be deceptive; final estimate may not be near the root, might just be close to zero
- Could stop when **change between estimates becomes small** relative to the current ( $n$ th) estimate:  $|x_{n+1} - x_n| < \epsilon|x_n|$ 
  - Better, but still fails when root is located at zero
- So let's use:
$$|x_{n+1} - x_n| < \begin{cases} \epsilon|x_n|, & \text{when } |x_n| \neq 0 \\ \epsilon, & \text{when } |x_n| = 0 \end{cases}$$

# Pseudocode of Newton-Raphson Algorithm

- 1. Choose initial guess at the root ( $x_0$ ), and the convergence tolerance ( $\epsilon$ ).
- 2. Loop through  $n$  up to a maximum number  $N_{\max}$  (exit and tell the user that the root finding has failed if it reaches  $N_{\max}$ )
- 3. Make sure  $f'(x) \neq 0$
- 4. Compute new estimate of root:  $x_n \simeq x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$
- 5. Check convergence criteria:

$$|x_{n+1} - x_n| < \begin{cases} \epsilon|x_n|, & \text{when } |x_n| \neq 0 \\ \epsilon, & \text{when } |x_n| = 0 \end{cases}$$

Example:  $f(x) = x^3 + 6$



- See [NR\\_root.ipynb](#)

# Secant method

- Similar to the Newton-Raphson method, but does not require calculating the derivative of the function
- Start with two initial guesses,  $x_{i-1}$  and  $x_i$
- Use finite difference derivative to get a new guess  $x_{i+1}$

$$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

- Proceed in the same way as the Newton-Raphson method



# Summary of root-finding methods

- Bisection:
  - Robust (with appropriate initial guesses)
  - Slow, each iteration reduces error by a factor of two
  - Need to make sure root is within initial guesses
- Newton-Raphson:
  - Fast: often only takes a few iterations
  - Need to know derivative of function, and they must exist
  - Can diverge, e.g., in cases with small second derivatives
- Secant method
  - Similar convergence speed as NR method
  - Don't need analytical derivatives
  - Same divergence properties as NR method
  - Numerical derivatives may be noisy

# After class tasks

- Homework 1 has been posted
  - Let me know if you have HW questions or questions/issues on github classroom
  - Office hours: Mondays, 3:00pm to 4:00pm; Thursdays, 11:05am to 1:00pm
    - Feel free to send me an email, and remember, if you push your changes, I should be able to see them
- Readings:
  - Pang Section 2.1 and 3.3
  - [Wikipedia article on Chebyshev nodes](#)
  - [Myths about polynomial interpolation](#)
  - [Wikipedia page on root finding](#)