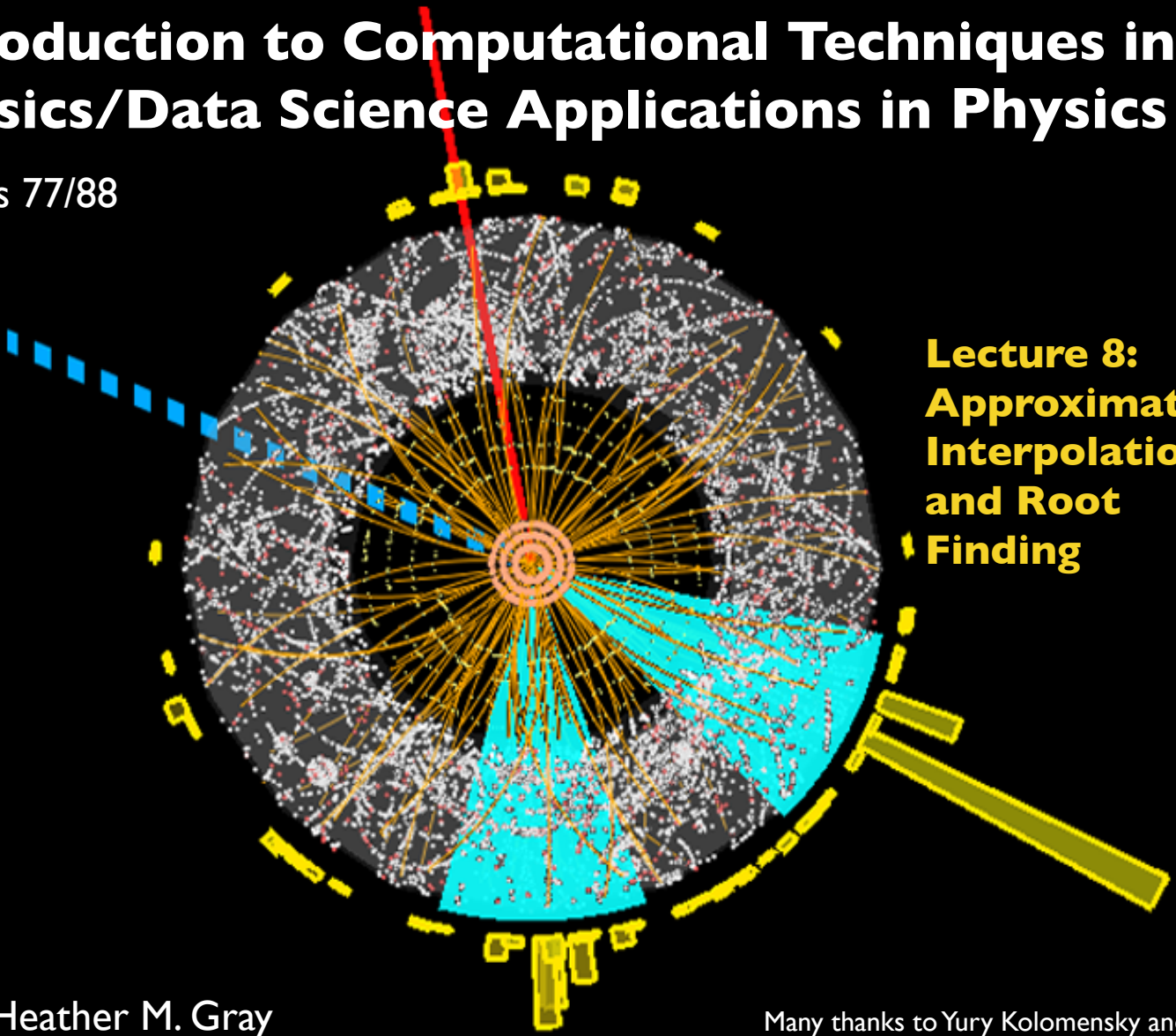
# Introduction to Computational Techniques in Physics/Data Science Applications in Physics

Physics 77/88



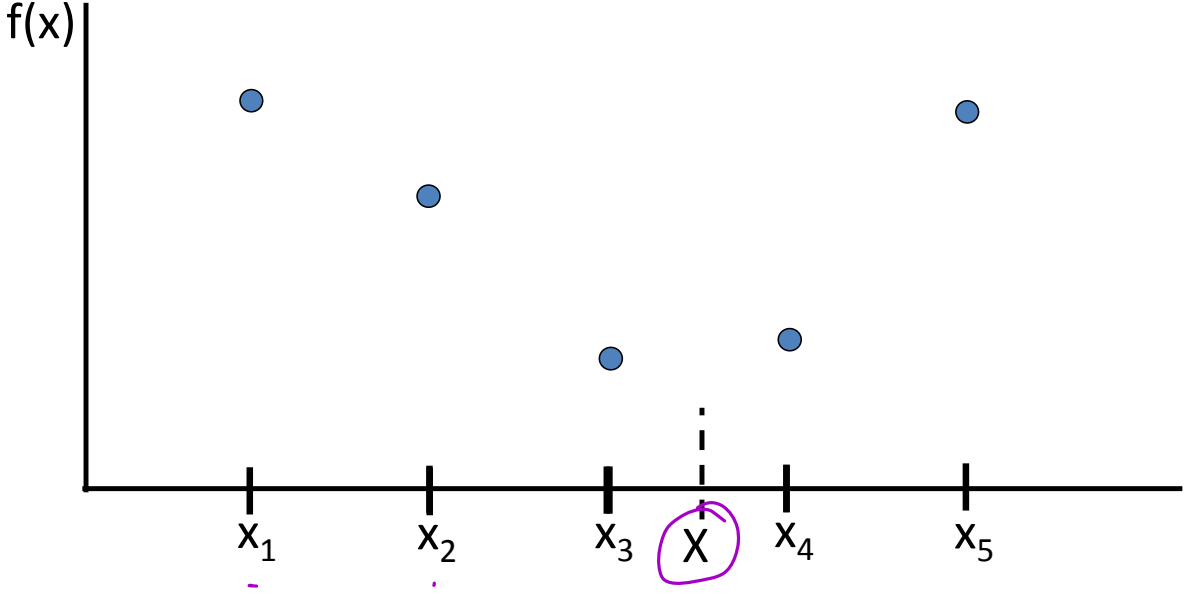
## Lecture 8: Approximation: Interpolation and Root Finding

Prof. Heather M. Gray

Many thanks to Yury Kolomensky and Amin Jazaeri for material used in this course

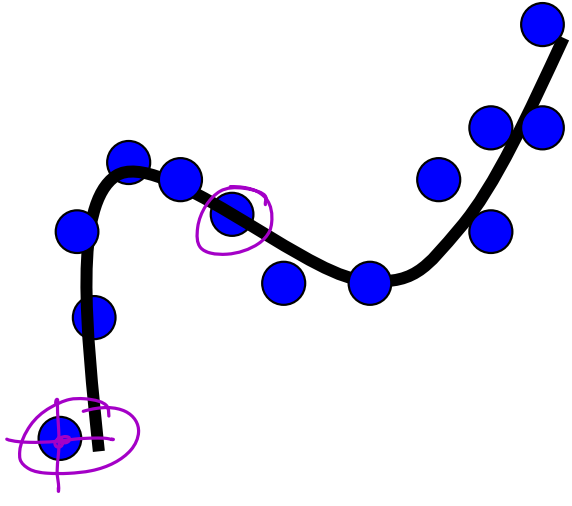
# Approximations

- Suppose we have set of discrete measurements, and we wish to estimate values between the known data points
  - This is a general problem of approximation a function



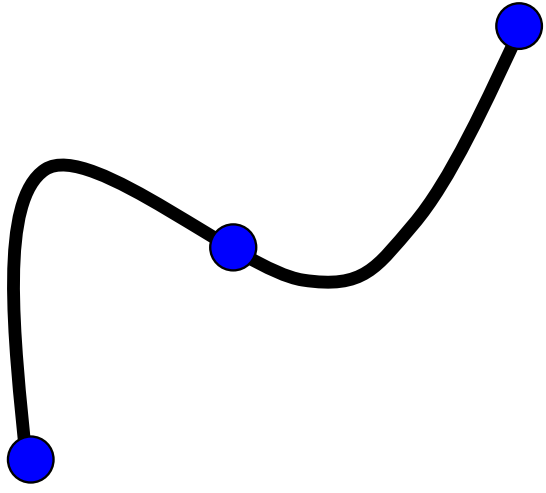
# Interpolation vs Fitting

- We can fit a function to the data points, and evaluate that function between the data points
  - e.g. a least-squares fit
  - Requires an analytical (typically) form of the function
  - Does not guarantee that the curve would go through the known points precisely
  - Appropriate when the data points have some uncertainty



# Interpolation vs Fitting

- Or we can require that the function goes exactly through the data points: this is known as interpolation
  - Appropriate if the data points are known



# Basis Functions

- Many choices of function that can go through the data
  - Called basis functions
- Function families commonly used for interpolation
  - Polynomials
  - Piecewise polynomial
  - Trigonometric functions
  - Exponential functions
  - Rational functions
- We will focus on interpolation using polynomials and piecewise polynomial

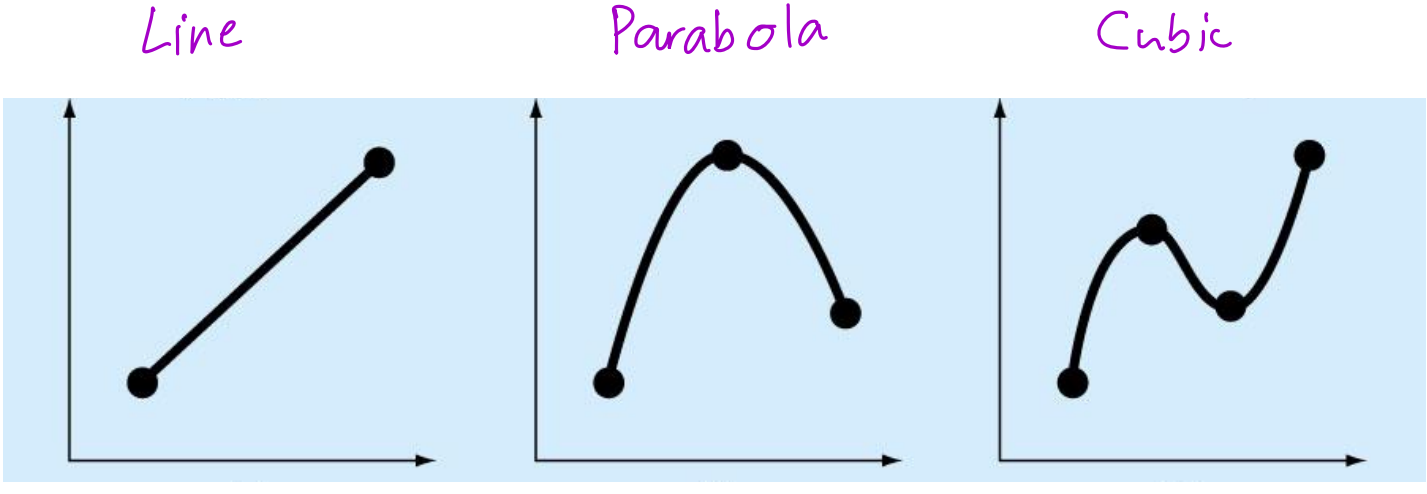
# Existence and Uniqueness

- Existence and uniqueness of interpolant depends on number of datapoints, $m$, and the number of basis functions, $n$
- If $m > n$ interpolant may or may not exist
- If $m < n$ : interpolant is not unique
- If $m = n$ : number of unknown parameters can be found exactly
  - i.e. data can be fit exactly
- A good choice of basis functions is essential

# Polynomial Interpolation

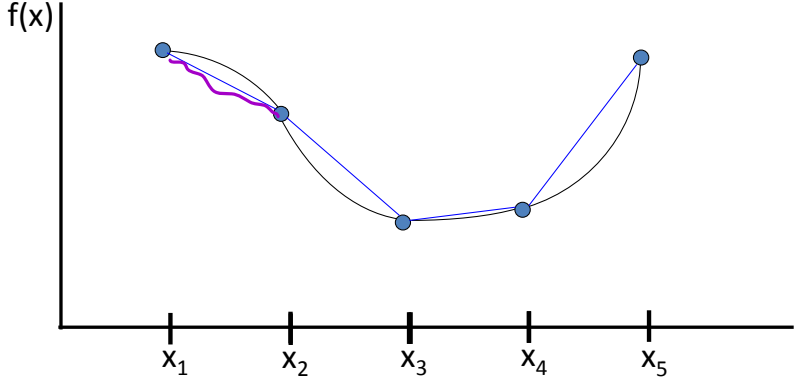
- Simplest and most common type of interpolation uses polynomials
- Polynomial interpolation solves for an $(n-1)$th order polynomial that passes through $n$ data points
  - $f(x) = a_1 + a_2 x + a_3 x^2 + \dots + a_n x^n$
  - $$\begin{bmatrix} 1 & x_1 & \dots & x_1^{n-2} & x_1^{n-1} \\ 1 & x_2 & \dots & x_2^{n-2} & x_2^{n-1} \\ \vdots & \vdots & \vdots & \vdots & \\ 1 & x_{n-1} & \dots & x_{n-1}^{n-2} & x_{n-1}^{n-1} \\ 1 & x_n & \dots & x_n^{n-2} & x_n^{n-1} \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{Bmatrix} = \begin{Bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{Bmatrix}$$

# Choice of Polynomials



# Global vs Piece-wise Interpolation

- An nth order polynomial contains n+1 coefficients, so can go through n+1 points
- Alternatively, use a series of polynomials, or splines, to link the points together
  - May require derivatives to be continuous
  - Splines are usually cubic polynomials, this is an example of a linear spline interpolation



# Linear Lagrange Interpolation

- Perform a Taylor's Series expansion of our function, $f(x)$ at two different points: $x_1$ and $x_2$
  - $f(x_1) = f(x) + (x_1 - x) f'(x) + O(\Delta^2)$
  - $f(x_2) = f(x) + (x_2 - x) f'(x) + O(\Delta^2)$
- Define a first order polynomial $p(x)$ to approximate $f(x)$ by requiring $f(x_i) = p(x_i)$
  - $f(x_1) = p(x) + (x_1 - x) p'(x) + O(\Delta^2)$
  - $f(x_2) = p(x) + (x_2 - x) p'(x) + O(\Delta^2)$
- Note that $p'(x)$ is constant

# Linear Lagrange Interpolation

- Multiply top equation by $(x - x_2)$ and bottom equation by $(x - x_1)$
  - $f(x_1)(x_2 - x) = p(x)(x_2 - x) + (x_1 - x)(x_2 - x)p'(x)$
  - $f(x_2)(x_1 - x) = p(x)(x_1 - x) + (x_1 - x)(x_2 - x)p'(x)$
- Subtract bottom from top

$$
\begin{aligned}
f(x_1)(x_2 - x) - f(x_2)(x_1 - x) &= p(x)(x_2 - x) - p(x)(x_1 - x) \\
&= p(x)[(x_2 - x) - (x_1 - x)] \\
&= p(x)(x_2 - x_1)
\end{aligned}
$$

# Linear Lagrange Interpolation

- Solve for $p(x)$:

$$
\begin{aligned}
p(x) &= \frac{f(x_1)(x_2 - x)}{x_2 - x_1} - \frac{f(x_2)(x_1 - x)}{x_2 - x_1} \\
&= \frac{f(x_1)(x_2 - x)}{x_2 - x_1} + \frac{f(x_2)(x_1 - x)}{x_1 - x_2} \\
&= f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1)
\end{aligned}
$$

# Quadratic Lagrange Interpolation

- Taylor expand again, but assume the function is a second order polynomial, $p(x)$
  - $f(x_1) = p(x) + (x_1 - x)p'(x) + (x_1 - x)^2 p''(x)/2$
  - $f(x_2) = p(x) + (x_2 - x)p'(x) + (x_2 - x)^2 p''(x)/2$
  - $f(x_3) = p(x) + (x_3 - x)p'(x) + (x_3 - x)^2 p''(x)/2$
- After some more (lengthy) algebra to eliminate $p'(x)$ and $p''(x)$ to yield
- $$p(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} f(x_1) + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} f(x_2) + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} f(x_3)$$

# Lagrange Polynomials

- In general

$$P(x) = y_1 \frac{(x - x_2)(x - x_3)\cdots(x - x_n)}{(x_1 - x_2)(x_1 - x_3)\cdots(x_1 - x_n)} + y_2 \frac{(x - x_1)(x - x_3)\cdots(x - x_n)}{(x_2 - x_1)(x_2 - x_3)\cdots(x_2 - x_n)} + \cdots + y_n \frac{(x - x_1)(x - x_2)\cdots(x - x_{n-1})}{(x_n - x_1)(x_n - x_2)\cdots(x_n - x_{n-1})}$$

# Disadvantages

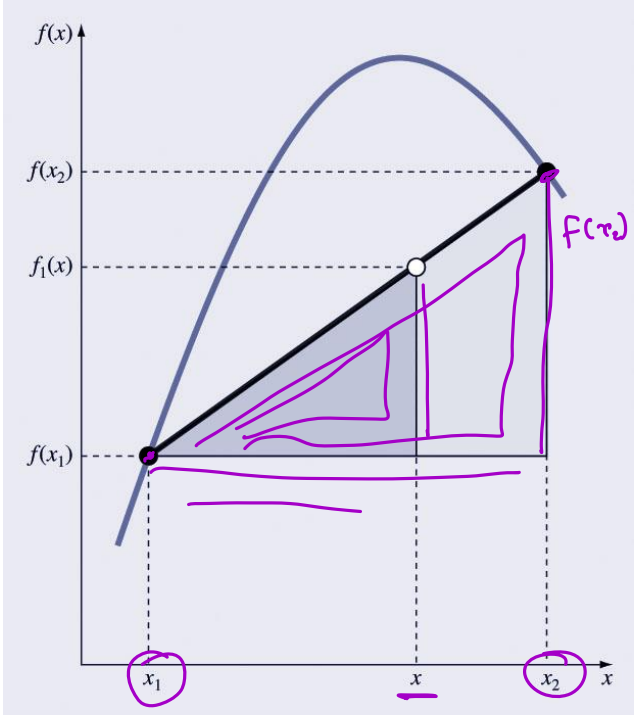
- Although the computation of $P_n(x)$ is simple the method is inefficient for large n
- If n is large and the data ~~for~~ is ordered, the efficiency can be improved by considering only data pairs in the vicinity of the x value for which $P_n(x)$ is sought
- However, this can degrade the approximation of $P_n(x)$

# Newton's Interpolating Polynomials

- In general, n+1 data points can be fitted by an nth order polynomial of the form
- $f_n(x) = b_0 + b_1(x - x_0) + \cdots + b_n (x - x_0)(x - x_1) \cdots (x - x_{n-1})$
- Coefficients are evaluated as follows
  - $b_0 = f(x_0) = f[x_0]$
  - $b_1 = \dfrac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_1, x_0]$ $b_1$
  - $b_2 = \dfrac{\dfrac{f(x_2) - f(x_1)}{x_2 - x_1} - \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0} = f[x_2, x_1, x_0]$
  - $b_n = f[x_n, x_{n-1}, \cdots, x_1, x_0]$

# First Order Newton Interpolating Polynomial

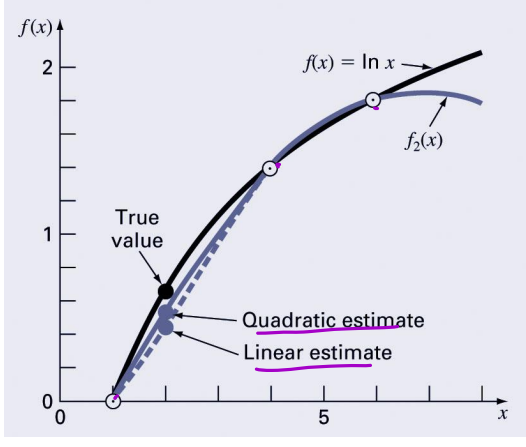
- First order Newton interpolating polynomial is obtained from linear interpolation and similar triangles
- Formula depends on known points $x_1$ and $x_2$ and $f(x_1)$ and $f(x_2)$




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

# 2nd Order Newton Interpolating Polynomial

- Second order Newton interpolating polynomial goes through the points but introduces some curvature
- Formula depends on known points $x_1$ and $x_2$ and $x_3$, $f(x_1)$ $f(x_2)$ and $f(x_3)$




$$f_1(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1) + \frac{\dfrac{f(x_3) - f(x_2)}{x_3 - x_2} - \dfrac{f(x_2) - f(x_1)}{x_2 - x_1}}{x_3 - x_1}(x - x_1)(x - x_2)$$

# Piecewise Polynomials

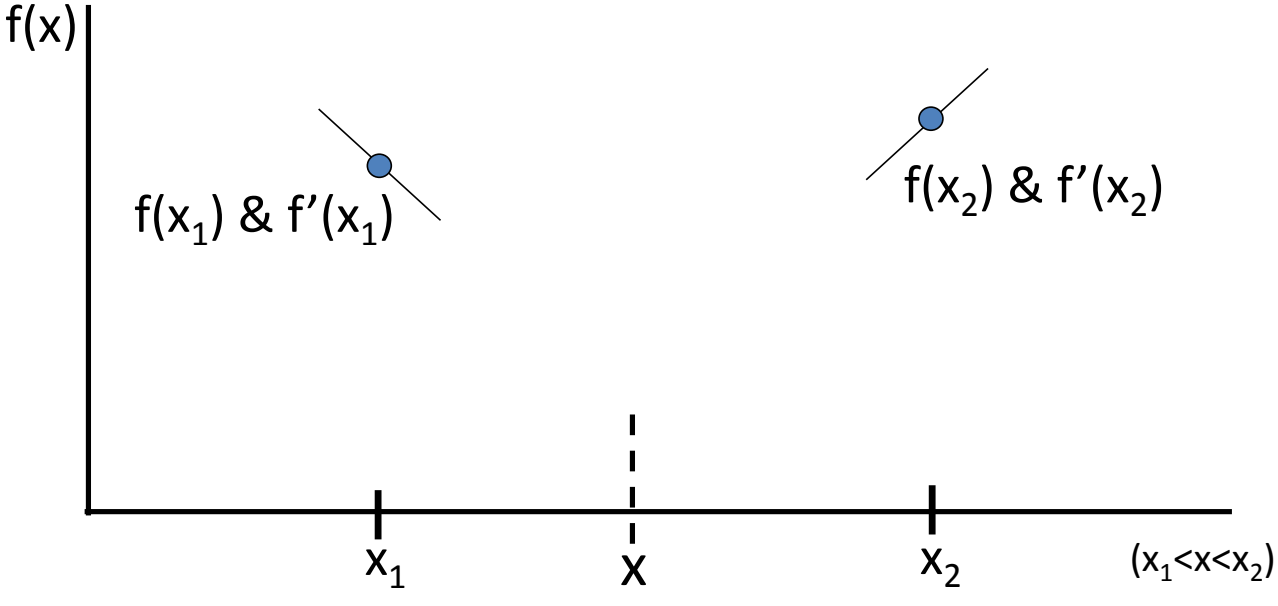
- Fitting a single polynomial to many data points can yield an oscillating interpolant
- Piecewise polynomials are a practical alternative to high degree polynomials
  - Fit may data points with low-degree polynomials
  - Different functions in each subinterval $[x_i, x_{i+1}]$
  - Abscissas $x_i$ are called knots or breakpoints
    - Where interpolant changes from one function to another

# Piecewise Polynomials

- Linear interpolation
  - Connect pairs of data points with straight lines
- Piecewise interpolation eliminates oscillation and non-convergence
  - Interpolating function is not smooth
- Many degrees of freedom in choosing polynomial interpolant
  - Can be used to obtain smooth interpolating function

# Hermite Interpolation

- If we have the function value and derivative at two points we can fully constrain a higher order polynomial



# Hermite Cubic Interpolation

- Given the general form for a cubic polynomial
  - $p(x) = ax^3 + bx^2 + cx + d$
- And its derivative
  - $p'(x) = 3ax^2 + 2bx + c$
- Two points
  - $f(x_1) = ax_1^3 + bx_1^2 + cx_1 + d$
  - $f(x_2) = ax_2^3 + bx_2^2 + cx_2 + d$
- Derivatives
  - $f'(x_1) = 3ax_1^2 + 2bx_1 + c$
  - $f'(x_2) = 3ax_2^2 + 2bx_2 + c$

# Hermite Polynomials

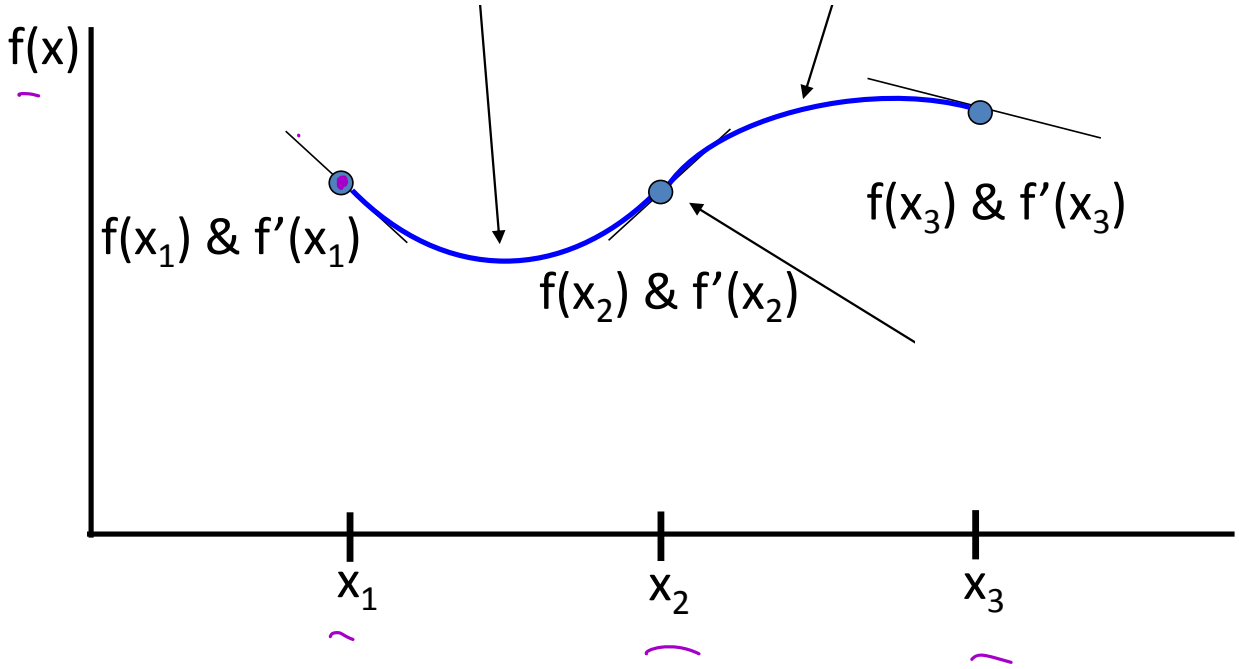
- Can again solve for $p(x)$

$$p_{1,2}(x) = \frac{(3x_1 - 2x - x_2)(x - x_2)^2}{(x_1 - x_2)^3} f(x_1) + \frac{(3x_2 - 2x - x_1)(x - x_1)^2}{(x_2 - x_1)^3} f(x_2)$$

- $$+ \frac{(x - x_1)(x - x_2)^2}{(x_1 - x_2)^2} f'(x_1) + \frac{(x - x_2)(x - x_1)^2}{(x_2 - x_1)^2} f'(x_2)$$

- If we fit a different polynomial $p_{2,3}(x)$ between $x_2$ and $x_3$ it will have the same derivative at $x_2$ as $p_{1,2}(x)$
  - $p'_{1,2}(x_2) = p'_{2,3}(x_2)$
- Interpolation between pairs of points is continuous
- Derivatives are also continuous
  - Hermite interpolation is smooth

# Hermite Polynomials at Boundaries



# Hermite vs Lagrange Interpolation

- A series of Hermite interpolating polynomials and their derivatives are continuous at the points
  - i.e. smooth
- A series of Lagrange interpolating polynomials is continuous at the points but their derivatives are not
- Hermite interpolating polynomials can fit cubic to local data: $x_1 < x < x_2$
- Lagrange interpolating polynomials require four points to fit a cubic: $x_1 < x_2 < x < x_3 < x_4$

# Cubic Spline Interpolation

- If derivatives at each point are unknown, can still require that the derivatives are continuous, up to a certain order
  - i.e. require that the derivatives of the polynomial to the left and right of each data point are the same
  - Cubic spline: for each pair of points, a 3rd degree polynomial has 4 unknown coefficients
    - $4n$ unknowns
  - Constraints:
    - $2n$ function values
    - $2(n-2)$ equations for 1st and 2nd derivative
    - Fix derivatives at the ends

# Hermite vs Cubic Spline

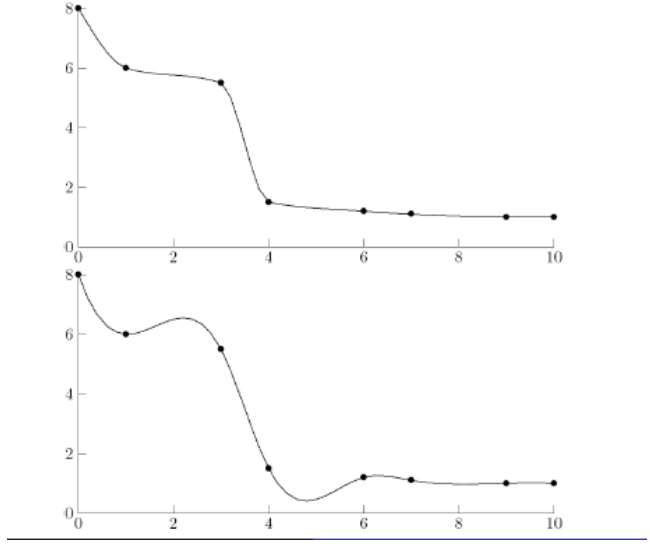
- Hermite cubic interpolant is a piecewise cubic polynomial with a continuous first derivative
  - Piecewise cubic polynomial with n knots has 4(n-1) parameters to be determined
  - Interpolates given data: 2(n-1) equations
  - One continuous derivative: (n-2) additional equations
  - Total of 3n-4 equations and n free parameters
- Hermite cubic interpolant is not unique and remaining free parameters can be chosen to satisfy additional constraints

# Hermite vs Cubic Spline

- Spline is a piecewise polynomial of degree $k$ that is $k-1$ times continuously differentiable
  - Linear spline is of degree $1$ and has $0$ continuous derivatives
    - Continuous but not smooth : broken line
  - Cubic spline is a piecewise cubic polynomial that is twice continuously differentiable
    - Like Hermite cubic, interpolating given data and requiring one continuous derivatives imposes $3n-4$ constraints
    - Requiring continuous second derivative imposes $n-2$ additional constraints: $2$ free parameters

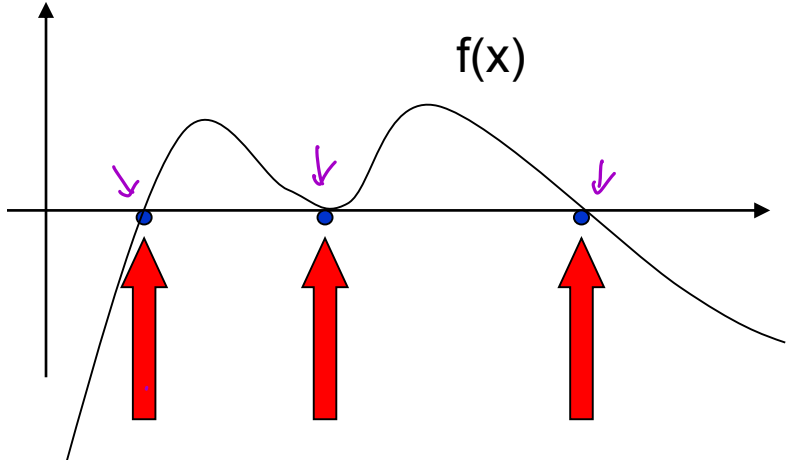
# Hermite vs Spline

- Choice depends on data to be fit and purpose of the interpolation
- Smoothness : spline interpolation
- Hermite polynomial is more appealing visually and preserves monotonicity of data
- Rule of thumb: plot interpolant and data to assess quality of interpolating function



# Root Finding

- Many problems in physics require finding roots of a function: $f(x) = 0$
- Zeros of any function $f(x)$ are the values of $x$ where the graph of the function crosses (or touches) the $x$-axis

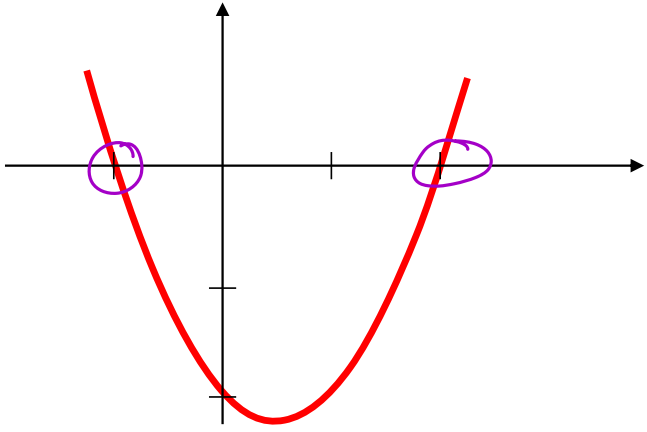


# Types of Roots

- Simple zeros

- $f(x) = (x+1)(x-2)$
- $= x^2 - x - 2$

- simple zeros
  - $x =$ 2
  - $x =$ -1

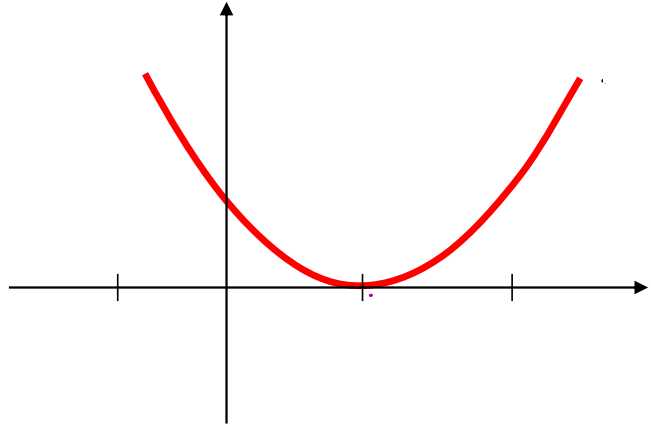


# Types of Roots

- Multiple zeros

$f(x) = (x-1)^2$

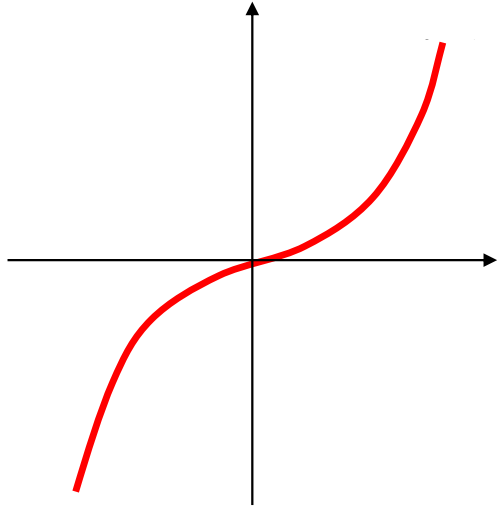
- $= x^2 - 2x + 1$



- Double zeros at
  - $x = 1$
- Or zero with multiplicity 2

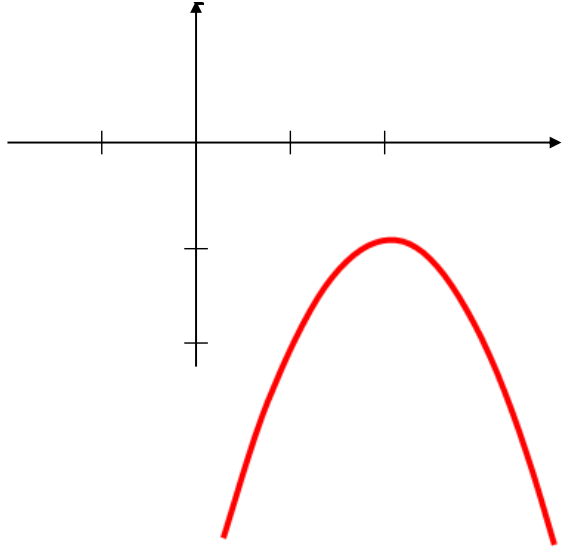
# Type of Roots

- Multiple zeros
- $f(x) = x^3$
- Zero with multiplicity 3 at $x = 0$



# Types of Roots

- No zeros
- $f(x) = -x^2 + 4x - 5$
- No real zeros



# Root of a Polynomial

- Consider a general polynomial
  - $f_n(x) = \Sigma_{i=0}^{n} a_i x^i = 0$ with $a_n = 1$
- $n = 1$
  - $f_1(x) = a_0 + a_1 x \; ; \; x = \frac{-a_0}{a_1}$ is the only root
- $n = 2$
  - $f_2(x) = a_0 + a_1 x + x^2; x = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_0}}{2}$
- $n = 3$
  - $f_3(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3; x =$

See http://mathworld.wolfram.com/CubicFormula.html

# Polynomials

- Factored form
  - $P(x) = a_n (x - x_1)(x - x_2) \ldots (x - x_n)$
- N roots
- N + 1 parameters
- Both real and complex roots
- No analytical solutions for polynomials of degree 5 or hight

# Facts about Polynomials

- Any nth order polynomial has exactly n zeros
  - Includes both real and complex zeros and their multiplicities
- Any polynomial with an odd order has at least one real zero
- If a function has a zero at $x=r$ with multiplicity $m$
  - Function and its first $m-1$ derivatives are zero at $x=r$ and the $m$th derivative at $r$ is not zero

# Root Finding

- In general, if an analytical solution to the equation does not exist, can find the solution approximately using numerical techniques
  - Iterative (Relaxation) method
  - Bracketing methods
  - Open method

# Iterative (Relaxation) Method

- Example:
  - Take your (engineering) calculator (or app), switch angles to radians, type in any number, press "cos", and keep pressing until the number on the screen stops changing.
- What did you find ?

# Iterative (Relaxation) Method

- If the function is of the form
  - $f(x) =$ $g(x) - x$
- Finding root $f(x) = 0$ can be done iteratively:
  - Pick $x_0$
  - Compute $x_1 =$ $g(x_0)$
  - $x_n =$ $g(x_{n-1})$
- Converges to the solution quickly
  - Or diverges / oscillates

# Bracketing Methods

- In bracketing methods, start with an interval that contains the root
- Procedure is applied to obtain a small interval containing the root
- Examples of bracketing methods
  - Bisection method
  - False position method

# Open methods

- Start with one or more initial guesses
- Obtain a new guess of the root for each iteration
- Typically more efficient than bracketing methods
- May not converge to the root

# Iterative Root Finding Methods

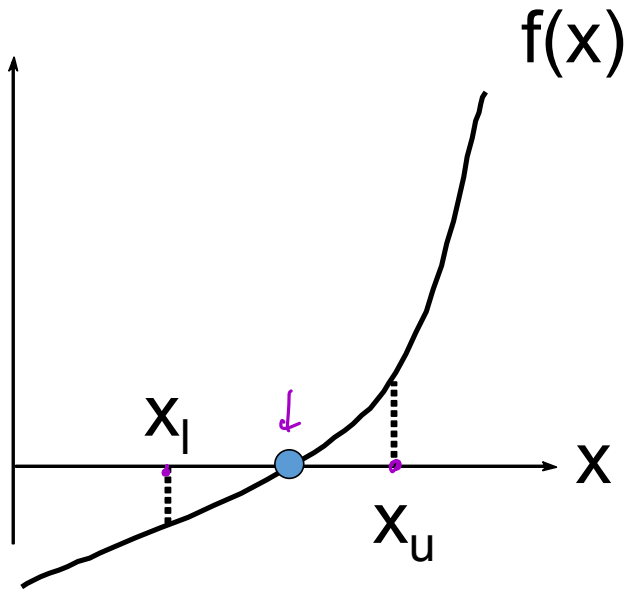
- Numerical algorithms take any input function $f(x)$ and search for solutions
  - No perfect algorithm for all functions $f(x)$
  - Some converge fast for certain $f(x)$ but slowly for others
  - Certain $f(x)$ may not yield numerical solutions
- Warning: don't use algorithms as black boxes
  - Know the limitations of each algorithm

# Iterative Root Finding Methods

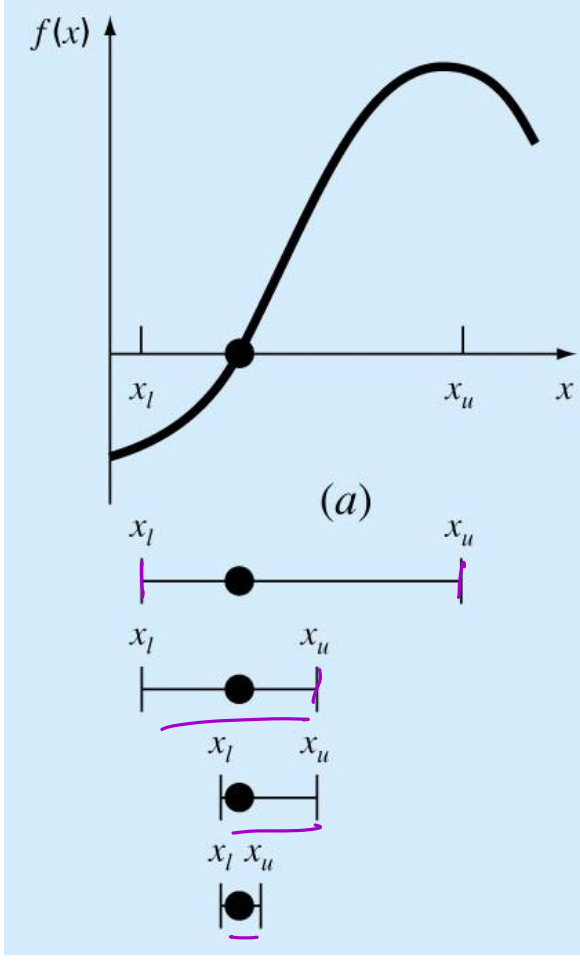
- Bisection
- Newton-Raphson
- Secant Method
- Müller-Brent Method

# Bracketing and Bisection

- Given a region bounded by $x_l$ and $x_u$ such that the sign of the function $f(x)$ between $f(x_l)$ and $f(x_u)$ there must be at least one root
- Assumes function is continuous and doesn't have infinities
- Root is said to be bracketed by the interval $(x_l, x_u)$

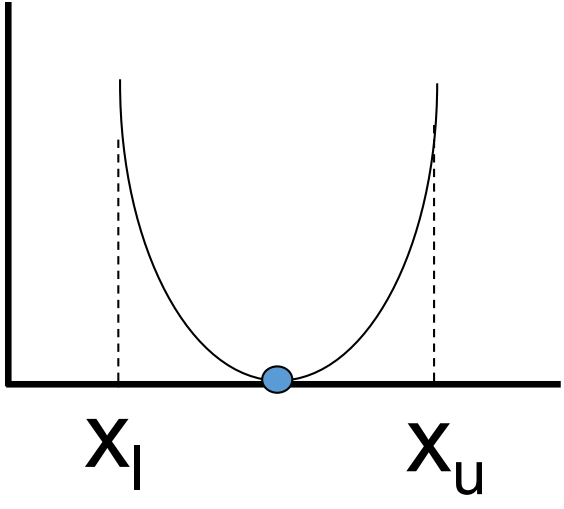


# Bracketing Method

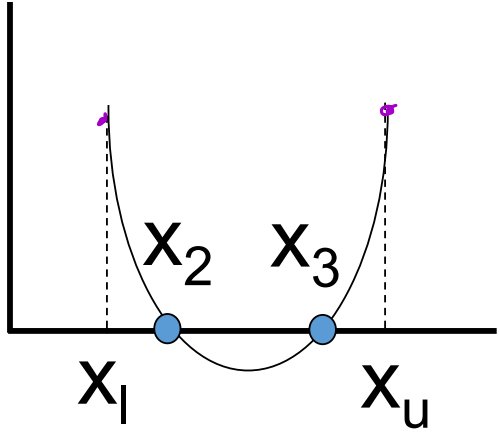


# Other Possibilities

- Even if a region is bracketed by two positive (negative) values, a root may still exist

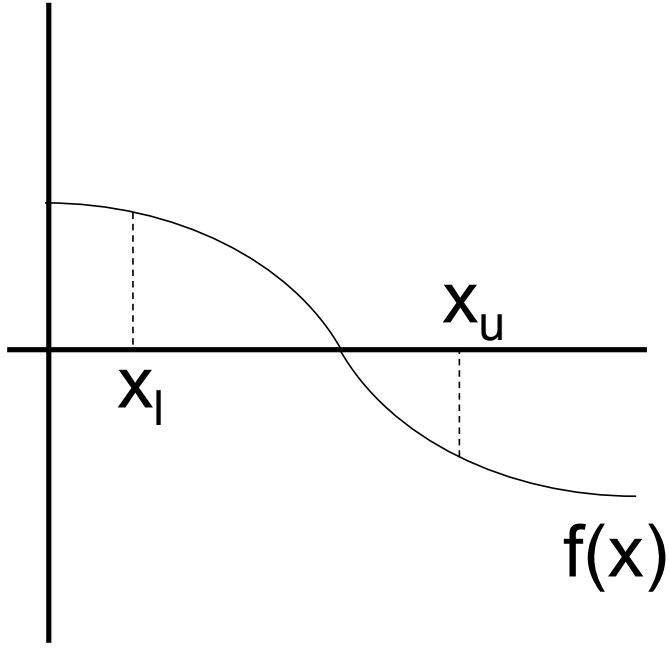



- There might even be 2 or more roots in this region for functions with nearby root



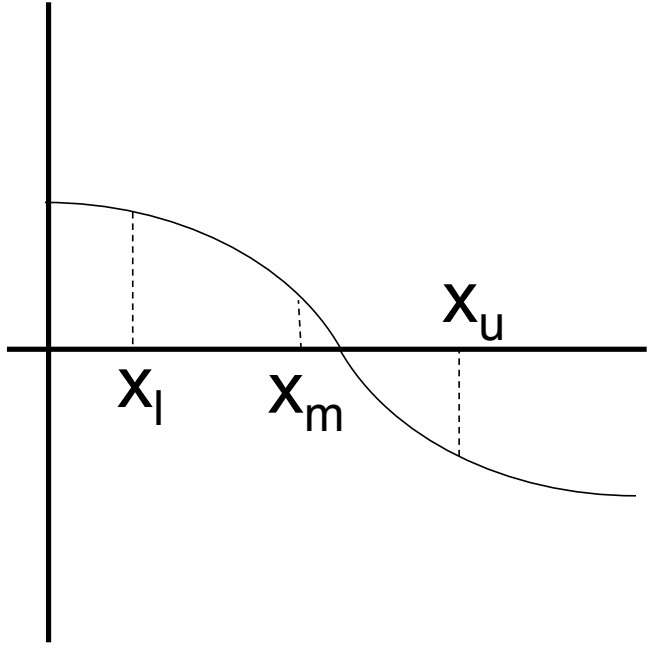
# Root Finding by Bisection

- Step 1: Given that we know that there exists an $f(x_0) = 0$ solution, choose $x_l$ and $x_u$ as the brackets for the root
  - Because $f(x_l)$ and $f(x_u)$ must have opposite signs, we have: $f(x_l) \times f(x_u) < 0$
- e.g. $f(x) =$ $\cos(x) - x$
- $\Rightarrow 0 < x_0 < \pi/2$



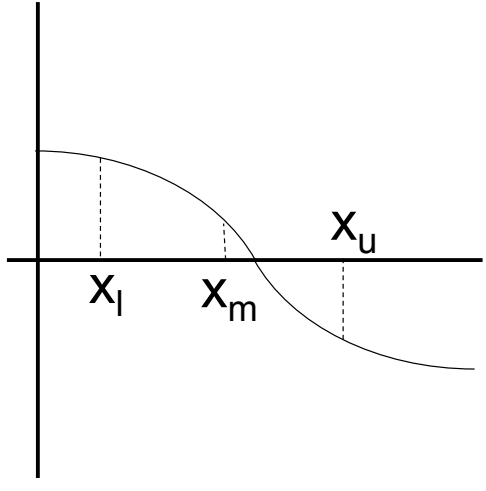
# Bisection: Find the Mid-Point

- Step 2: Let
  $x_m =$ 0.5 ($x_l + x_u$)
- $x_m$ is the midpoint between the two brackets
- It must be closer to the root than one of $x_l$ and $x_u$
- Next step is to select the new bracket from $x_l, x_m$ and $x_u$
- Particular case: new $x_l$ is the same as old $x_m$



# Bisection: Find New Bracket

- Step 3
  - If $f(x_l)f(x_m) < 0$
    - Root lies between $x_l$ and $x_m$
    - $x_l = x_l; x_u = x_m$
  - If $f(x_l)f(x_m) > 0$
    - Root lies between $x_m$ and $x_u$
    - $x_l = x_m ; x_u = x_u$
  - If $f(x_l)f(x_m) = 0$
    - Root is $x_m$ ; stop



# Final Step

- Step 4: Test accuracy
  - $x_m$ is the best guess at this stage
  - Absolute error is $|x_m - x_0|$ but we don't know $x_0$
  - Error estimate: $\varepsilon = |x_m^n - x_n^{n-1}|$ where $n$ corresponds to the nth iteration
  - Check if $\varepsilon < \delta$ where $\delta$ is the tolerance that you decided at the start of root finding
    - If $\varepsilon < \delta$ stop
    - If $\varepsilon > \delta$ return to step 1

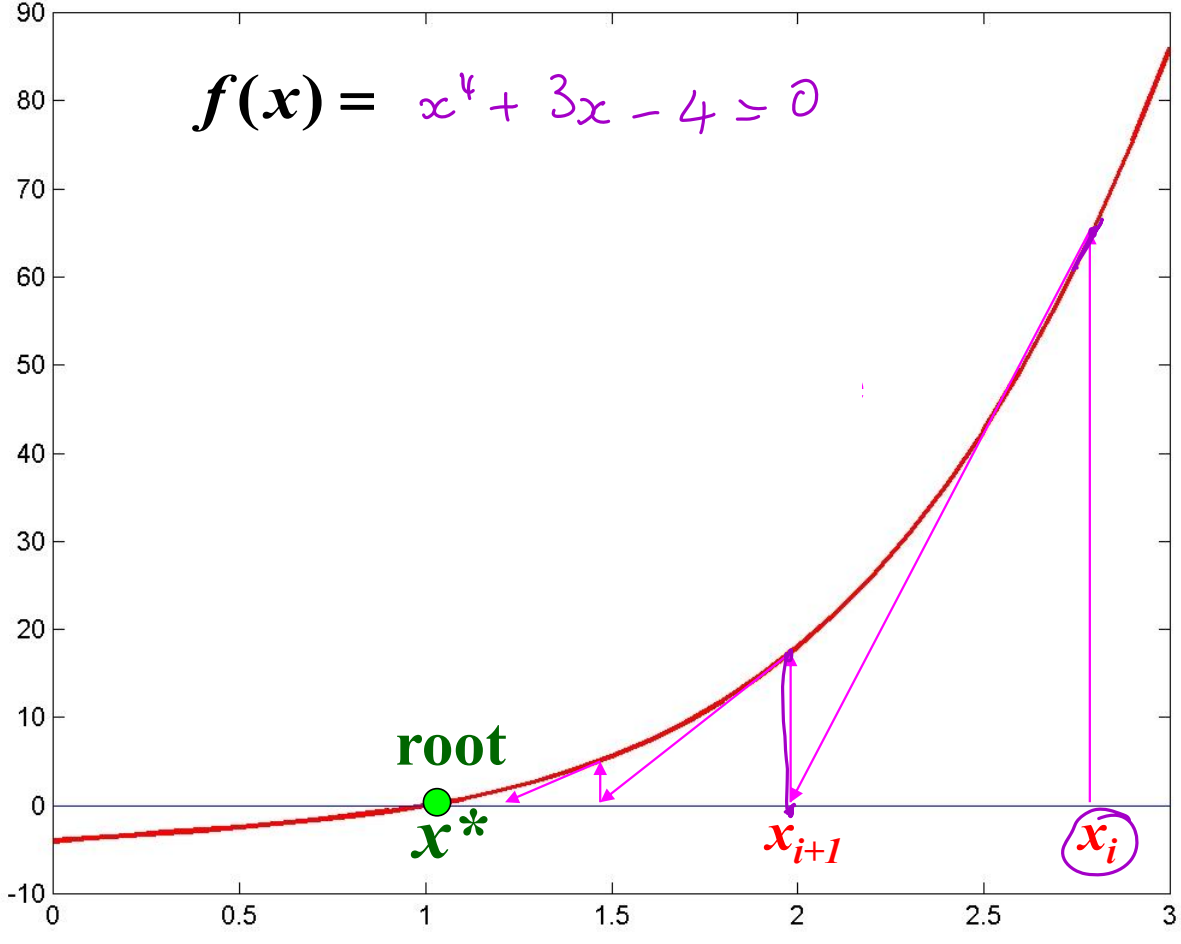
# Pros and Cons of Bisection

- Pros:
  - Bisection is robust
    - Will always find a root
  - Can be programmed straightforward
- Cons
  - Bisection converges very slowly (linear)
    - Width of bracket: $x_u^n - x_l^n = w^n = w^{n-1}/2$
      - Width halves at each stage
      - Know in advance that the width of the bracket after $n$ steps will have width $\omega^0/2^n$
      - Will achieve tolerance when $n = \log_2(\omega^0/\delta)$

# Newton-Raphson Method

- For many numerical methods, underlying algorithms begin with a Taylor expansion
- Taylor expansion of $f(x)$ around a point $x_0$
  - $f(x) \approx f(x_0) + (x - x_0)f'(x_0) + \cdots$
- If $x$ is a root
  - $0 \approx f(x_0) + (x - x_0)f'(x_0)$
  - $\Rightarrow x \approx x_0 - \frac{f(x_0)}{f'(x_0)}$
- Formula applies as long as $f'(x_0)$ is not an extremum
  - $f'(x_0) \neq 0$

# Newton-Raphson Method

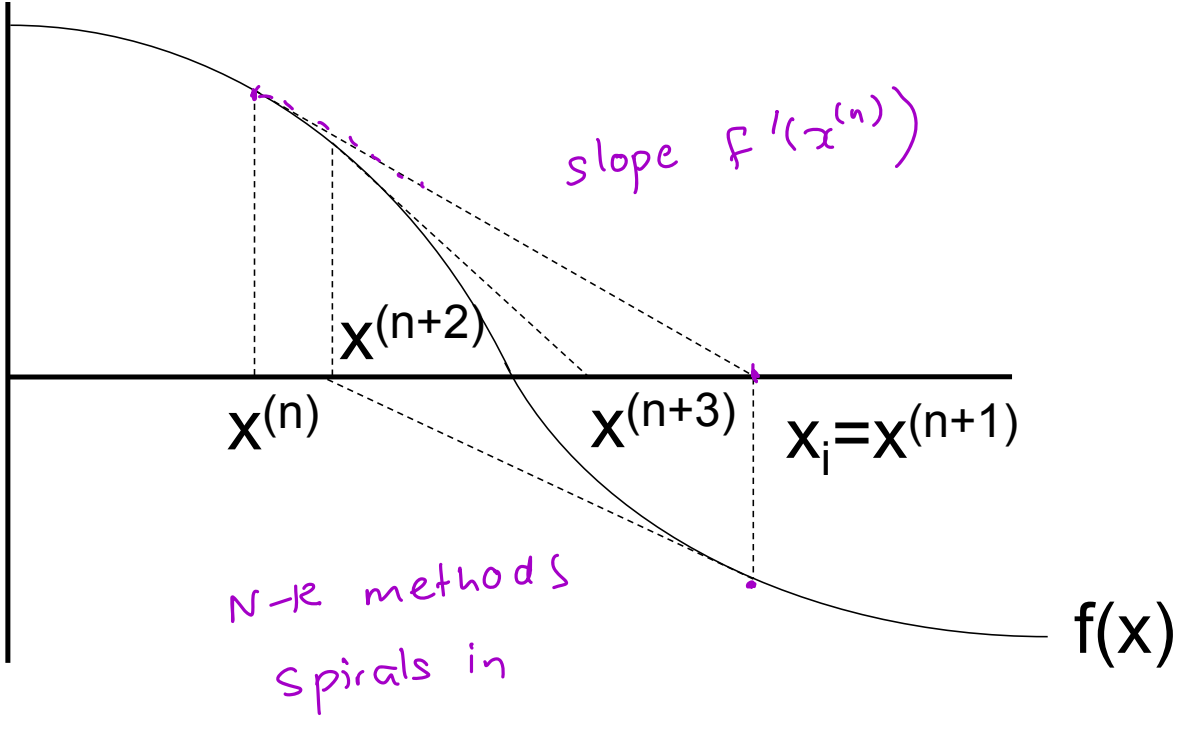


# Newton Raphson Method

- Step 1: Start at the point $(x_1, f(x_1))$
- Step 2: Calculate ~~interaction~~ intersection of tangent $f(x)$ at this point and the $x$-axis
  - $x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$
- Step 3: Check if $f(x_2) = 0$ or $|x_2 - x_1| < \delta$ (tolerance)
- Step 4:
  - If yes, solution $x_r = x_2$
  - If no, repeat the iteration $x_2$

# Geometric Interpretation

- General step in the iteration is
  - $x^{(n+1)} \approx x^{(n)} - \frac{f(x)^{(n)}}{f'(x)^{n}}$



# Newton-Raphson's Method

- Note than an evaluation of the derivative (slope) is required
- You may have to do this numerically
- Open method: convergence depends on the initial guess and it not guaranteed
- However, Newton's method can converge very quickly
  - Quadratic convergence

# Convergence

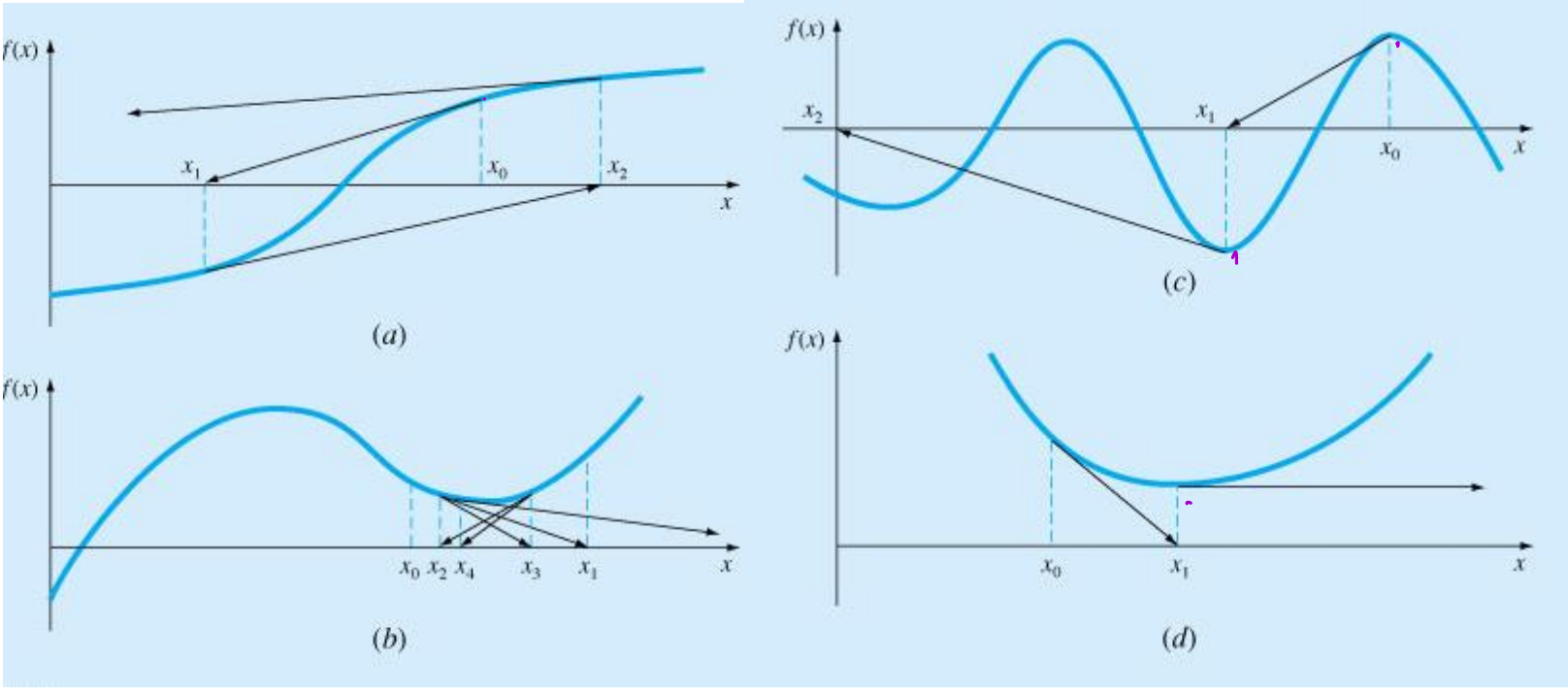
- Necessary and sufficient condition
  - $\left| \frac{F(x)\,F''(x)}{[F'(x)]^2} \right| < 1$
- Error term of:
  - $E(x) = K(\Delta x)^n$
- Doesn't converge if $F(x)$ has multiple roots

# Newton-Raphson Method

- Although Newton-Raphson converges very rapidly it may diverge and fail to find roots if
  - There are multiple roots
  - An inflection point, $f''=0$, is near the root
  - There is a local minimum or maximum: $f'=0$
  - A zero slope is reached

Convergence is not guaranteed for open methods

# Poor convergence for Newton-Raphson

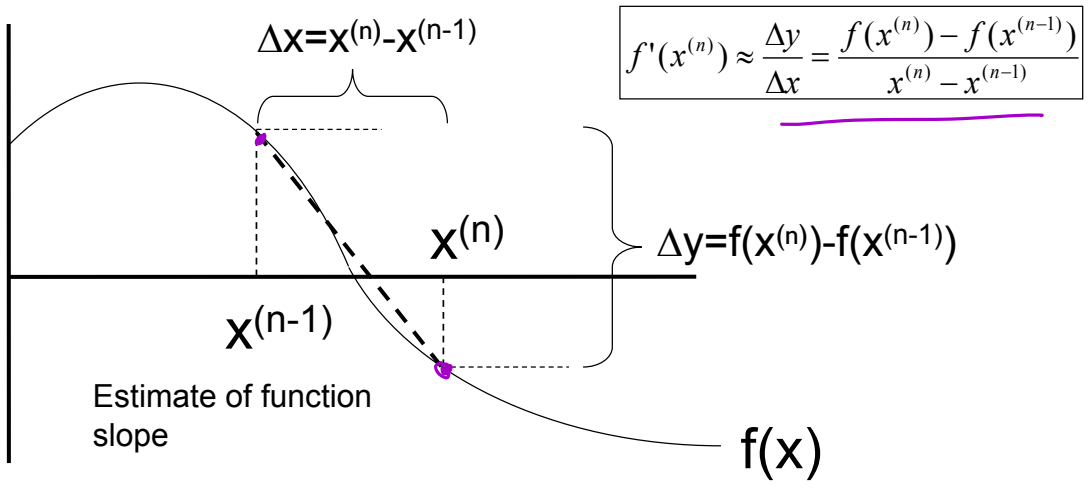


# Newton-Raphson-Bisection Hybrid

- Combine the two methods
  - Good convergence from Newton-Raphson
  - Guaranteed convergence from bisection
- Start with $x_l < x^{(n)} < x_u$ as current best guess within the bracket
- Let $x_{NR}^{(n+1)} \approx x^n - \frac{f(x^n)}{f'(x^n)}$
  - If $x_l < x_{NR}^{n+1} < x_u$ then take an NR step
  - Else set $x_{NR}^{(n+1)} = x_m^{(n+1)} = 0.5(x_e + x_u)$
    - Check and update $x_l$ and $x_u$ values for new bracket
- Throw away bad NR steps: do a bisection step instead

# Secant Method

- A key issue in NR is that you need the derivative
  - May be unknown if its defined numerically
  - Could be very hard to calculate
- Can estimate the local slope using $x^{(n)}$, $x^{(n-1)}$, $f(x^{(n-1)})$ and $f(x^{(n)})$




$$f'(x^{(n)}) \approx \frac{\Delta y}{\Delta x} = \frac{f(x^{(n)}) - f(x^{(n-1)})}{x^{(n)} - x^{(n-1)}}$$

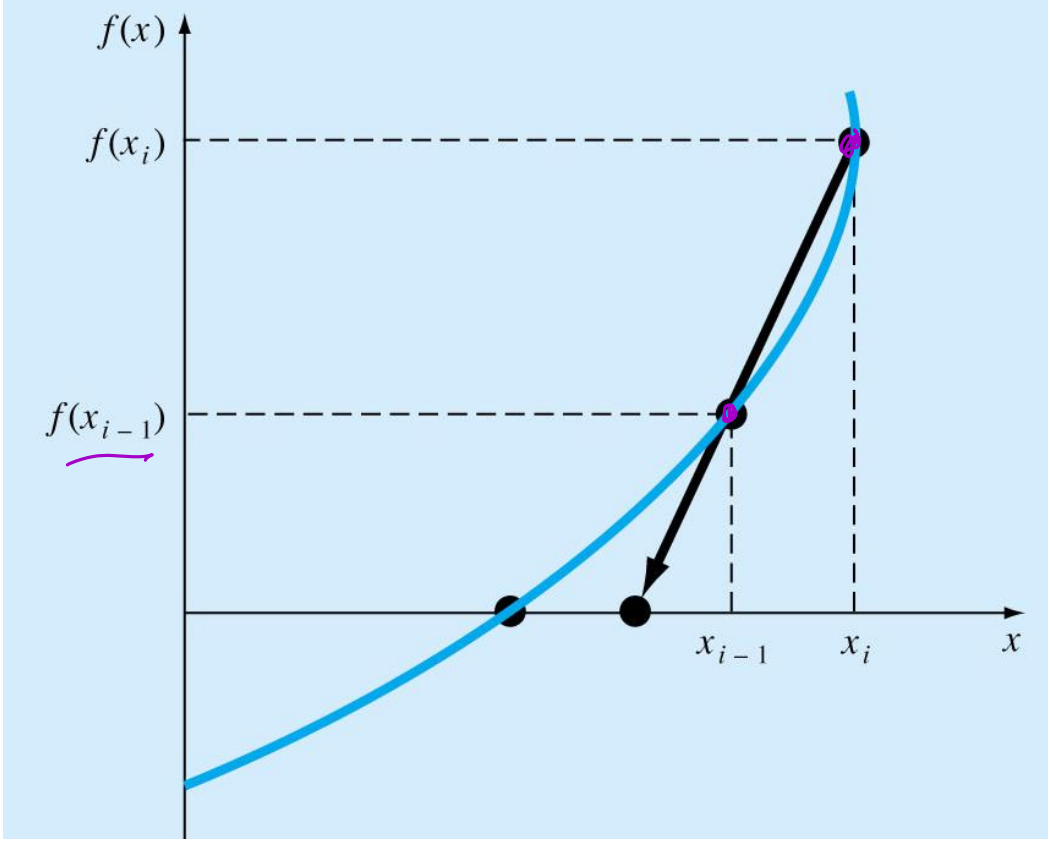
# Secant Method

- After initially selecting a pair of points bracketing the root $(x^{(n-1)}, x^{(n)})$, $x^{(n+1)}$ is given by

$$x^{(n+1)} = x^{(n)} - f(x^{(n)}) \frac{x^{(n)} - x^{(n-1)}}{f(x^{(n)}) - f(x^{(n-1)})}$$

- Iterate to a specific tolerance as in NR
- Converges almost as quickly as NR

# Secant Method




Use secant line instead of tangent line at $f(x_i)$

# Algorithm for Secant Method

- Open Method
- Step 1: Begin with any two endpoints $[a, b] = [x_0, x_1]$
- Step 2: Calculate $x_2$ using the secant method formula
  - $x_{i+1} = x_i - \dfrac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$ ←
- Step 3: Replace $x_0$ with $x_1$, $x_1$ with $x_2$ and repeat from step 2 until convergence is reached
- Use the two most recently generated points in subsequent iterations (not a bracket method)

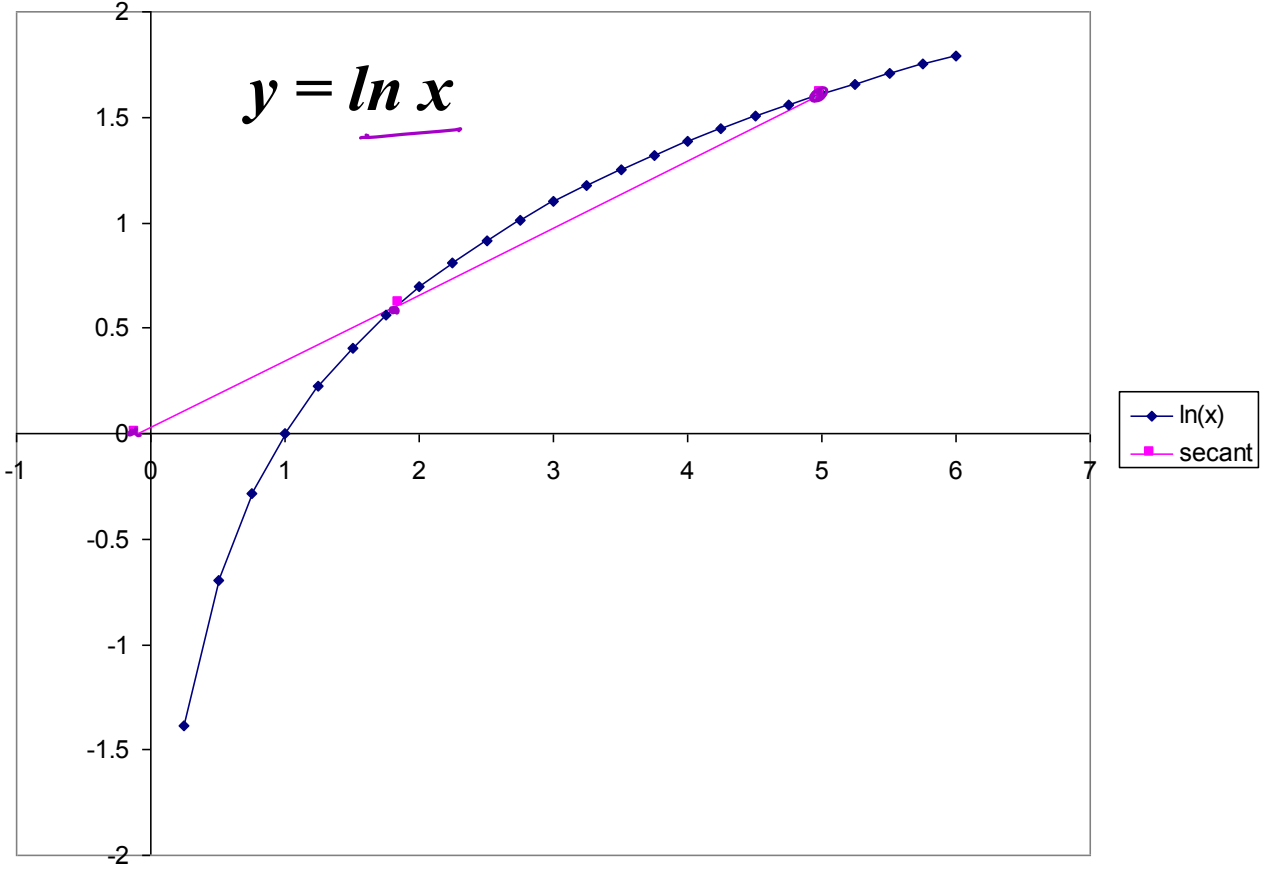
# Secant Method

- Pros
  - It can converge fast and doesn't to bracket the root
- Cons
  - It is not guaranteed to converge
  - It may diverge and fail to yield an answer

# Issues with the Secant Method

- Given the initial values $x_l = x^{(n-1)}$ and $x_u = x^{(n)}$ that bracket the root, the calculation of $x^{(n+1)}$ is local
  - Easy to program, all values are easily
- Difficulty is with finding suitable $x_l$ and $x_u$ since this is a global problem
  - Determined by $f(x)$ over its entire range
  - Consequently difficult to program

# Example: Lack of Convergence



# Müller-Brent Method

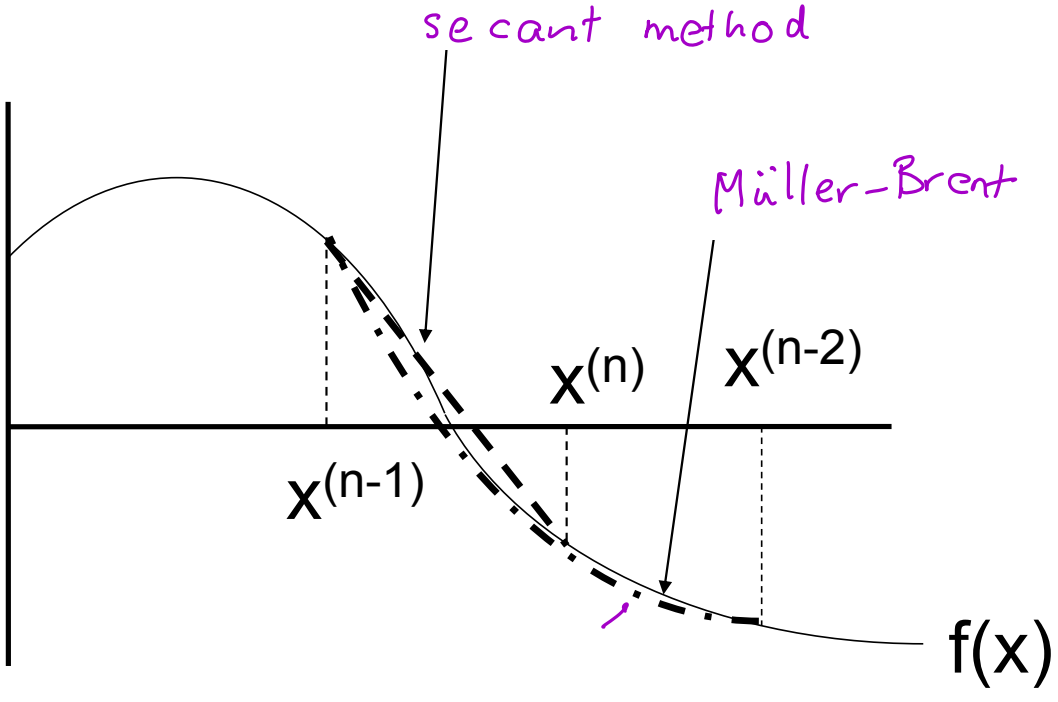
- Most expensive part of root finding algorithms is frequently the function evaluation
- In Secant method the n+1 guess is based upon function evaluations from n and n-1 steps
  - Throw away the n-2 step
  - With only two function evaluations, the fitting method must be linear
- Could use the n-2 step to give us three points and use those for quadratic interpolation

# Müller Method

- Method consists of deriving coefficients of a parabola that goes through the three points
- Let's write the equation in a convenient form
  - $f_2(x) = a(x - x_2)^2 + b(x - x_2) + c$

# Secant Method compared to Müller-Brent

- Secant method fits a straight line between $(x^{(n)}, f(x^{(n)}))$ and $(x^{(n-1)}, f(x^{(n-1)}))$
  - Corresponds to dashed line



# Fitting a Quadratic

- Three points is enough data to fit a function of the form
  - $q(x) = a(x - x^{(n)})^2 + b(x - x^{(n)}) + c$
- As we know $q(x)$ passes through all three points
  - $q(x^{(n)}) = f(x^{(n)}) \Rightarrow c = f(x^{(n)})$
  - $q(x^{(n-1)}) = f(x^{(n-1)}) = a(x^{(n-1} - x^{(n)})^2 + b(x^{(n-1} - x^{(n)}) + f(x^{(n)})$
  - $q(x^{(n-2)}) = f(x^{(n-1)}) = a(x^{(n-2} - x^{(n)})^2 + b(x^{(n-2} - x^{(n)}) + f(x^{(n)})$
- Two equations and two unknowns: $a$ and $b$

# Solving for Coefficients

- Given the two equations, we can eliminate $a$ or $b$ to get equations for the two constants

$$a = \frac{(x^{(n-1)} - x^{(n)})[f(x^{(n-2)}) - f(x^{(n)})] - (x^{(n-2)} - x^{(n)})[f(x^{(n-1)}) - f(x^{(n)})]}{(x^{(n-2)} - x^{(n-1)})(x^{(n-2)} - x^{(n)})(x^{(n-1)} - x^{(n)})}$$

$$b = \frac{(x^{(n-2)} - x^{(n)})^2[f(x^{(n-1)}) - f(x^{(n)})] - (x^{(n-1)} - x^{(n)})^2[f(x^{(n-2)}) - f(x^{(n)})]}{(x^{(n-2)} - x^{(n-1)})(x^{(n-2)} - x^{(n)})(x^{(n-1)} - x^{(n)})}$$

- Now have $a$, $b$ and $c$, so we can use the general formula to calculate the zeros for $x^{(n+1)} - x^{(n)}$
  - $x^{(n+1)} - x^{(n)} = \dfrac{-b \pm \sqrt{b^2 - 4ac}}{2a}$