# Physics 77

## Introduction to Computational Techniques in Physics

Spring 2019

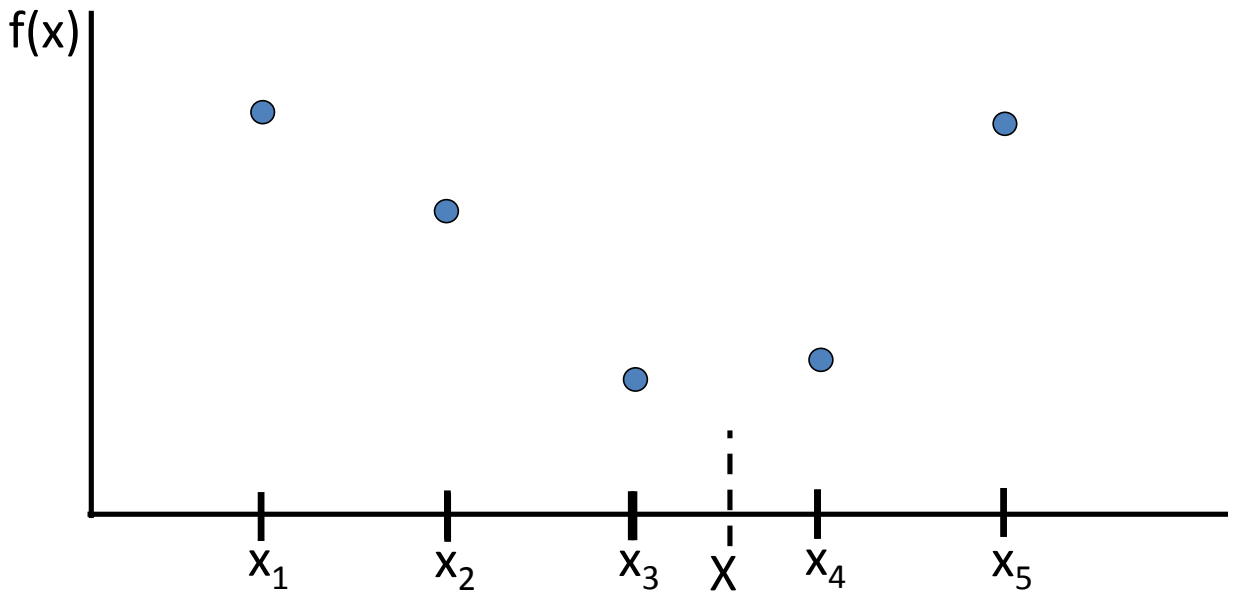
## Approximate Methods in 1D: Interpolation, Root Finding, Minimization

Amin Jazaeri, Yury Kolomensky

See also Python workbook Lecture08.ipynb

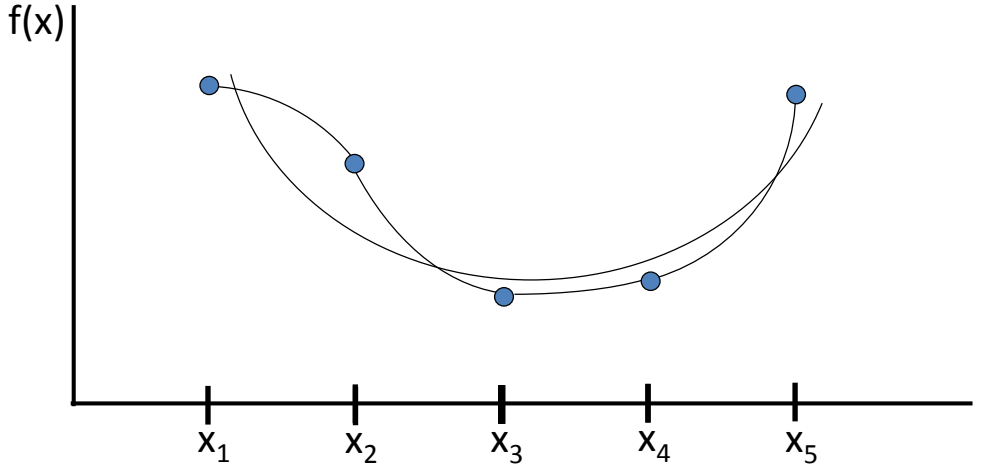
# Approximations

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

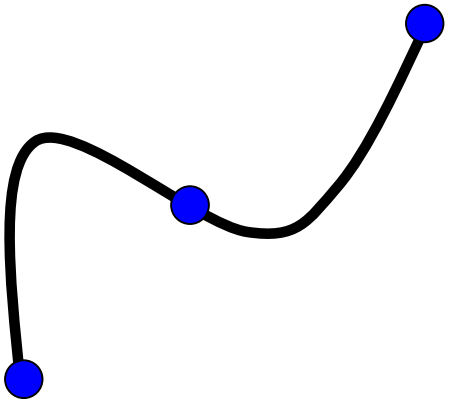


# Interpolation vs Fitting

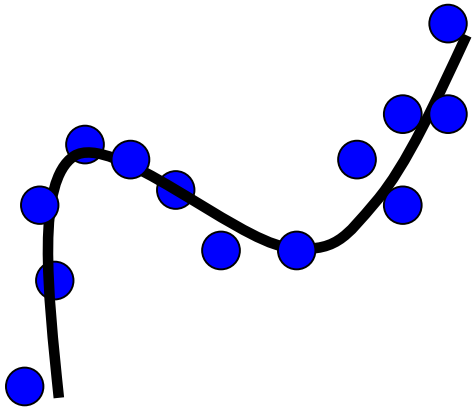
- We can *fit* a function to the data points, and evaluate that function in between the data points
  - E.g. a least-squares fit
  - Requires an analytical (typically) form of the function
  - Does not guarantee that the curve would go through the known points precisely
  - Appropriate when the data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



# Interpolation vs Fit



Interpolation



Fit

# Basis Functions

- There are many choices of function that can go through the data and those are called Basis Functions.
- Families of functions commonly used for interpolation include
  - Polynomials
  - Piecewise polynomials
  - Trigonometric functions
  - Exponential functions
  - Rational functions
- For now we will focus on interpolation by polynomials and piecewise polynomials

# Existence and Uniqueness

- Existence and uniqueness of interpolant depend on number of data points m and number of basis functions n
- If $m > n$, interpolant might or might not exist
- If $m < n$, interpolant is not unique
- If $m = n$, then the number of unknown parameters could be found exactly, so the data can be fit exactly.
- A good choice of basis function could be essential

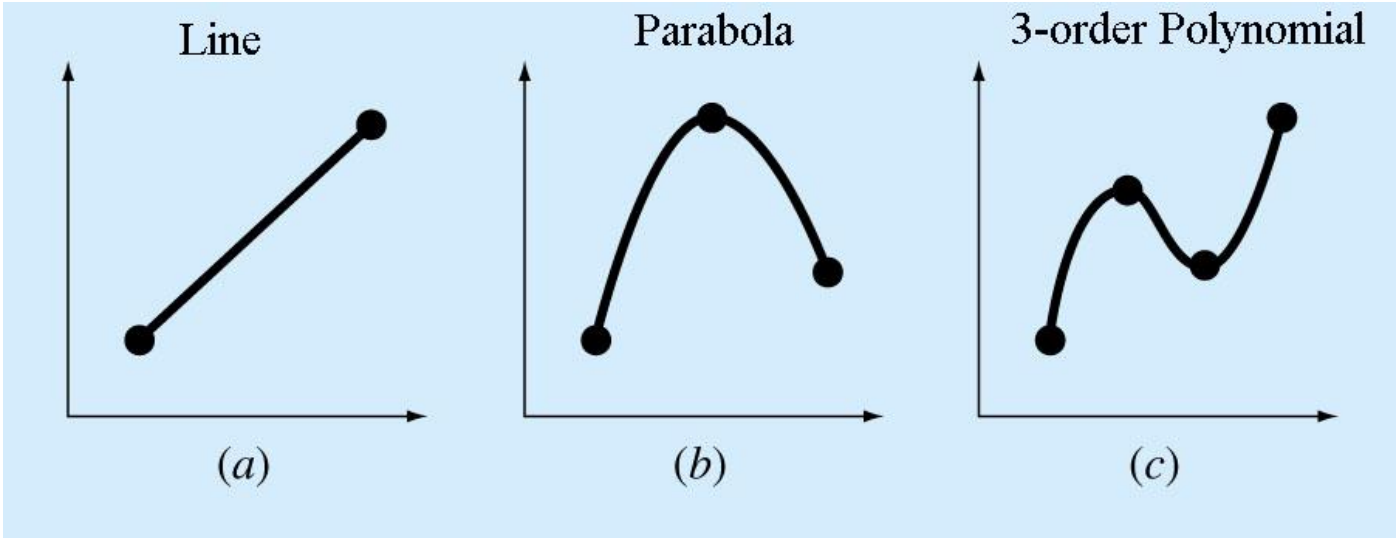
# Polynomial Interpolation

- Simplest and most common type of interpolation uses polynomials
- The simplest and most common method for this purpose is polynomial interpolation, where an $(n\text{-}1)^{\text{th}}$ order polynomial is solved that passes through $n$ data points:

$$f(x) = a_1 + a_2 x + a_3 x^2 + \cdots + a_n x^{n-1}$$

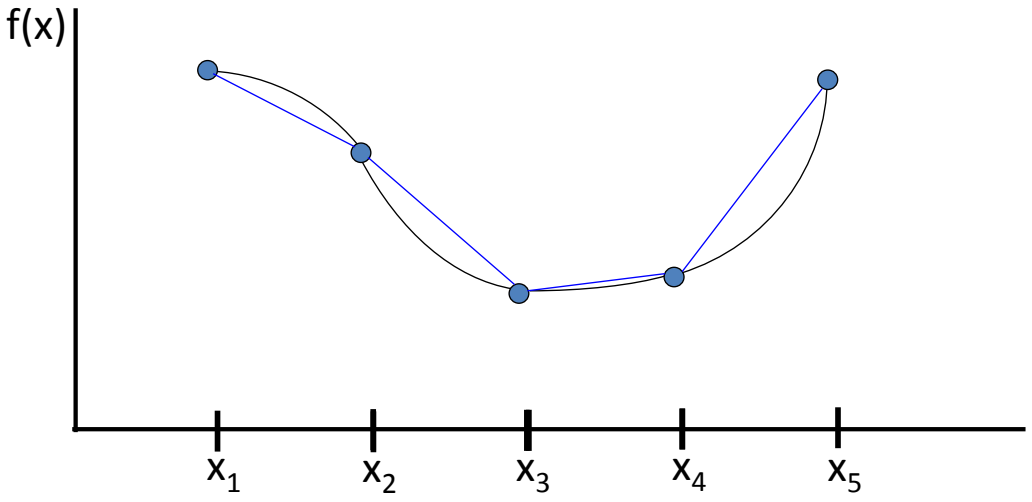
$$\begin{bmatrix} 1 & x_1 & \cdots & x_1^{n-2} & x_1^{n-1} \\ 1 & x_2 & \cdots & x_2^{n-2} & x_1^{n-1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{n-1} & \cdots & x_{n-1}^{n-2} & x_{n-1}^{n-1} \\ 1 & x_n & \cdots & x_n^{n-2} & x_n^{n-1} \end{bmatrix} \begin{Bmatrix} a_1 \\ a_2 \\ \vdots \\ a_{n-1} \\ a_n \end{Bmatrix} = \begin{Bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{Bmatrix}$$

# Choice of Polynomials



# Global vs Piece-wise Interpolation

- An $n^{th}$ order polynomial contains $n+1$ coefficients, so can go through n+1 points exactly
- Alternatively, can use a series of polynomials — called *splines* — to link points together
  - May want to require derivatives to be continuous
  - Splines are usually cubic polynomials, but example below shows a linear spline interpolation



# Linear Lagrange Interpolation

- Use Taylor's Series Expansion for *f(x):*

$$f(x_1) = f(x) + (x_1 - x) f'(x) + O(\Delta^2)$$

$$f(x_2) = f(x) + (x_2 - x) f'(x) + O(\Delta^2)$$

- Now we 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: p'(x)* is only a constant

# Linear Lagrange Interpolation

- Now multiply the top equation by $(x-x_2)$ and the bottom one by $(x-x_1)$

$$f(x_1)(x_2-x) = p(x)(x_2-x) + (x_1-x)(x_2-x)p'(x)$$

$$f(x_2)(x_1-x) = p(x)(x_1-x) + (x_1-x)(x_2-x)p'(x)$$

- Now subtract the bottom from the top:

$$f(x_1)(x_2-x) - f(x_2)(x_1-x) = p(x)(x_2-x) - p(x)(x_1-x)$$

$$f(x_1)(x_2-x) - f(x_2)(x_1-x) = p(x)[(x_2-x)-(x_1-x)]$$

$$f(x_1)(x_2-x) - f(x_2)(x_1-x) = p(x)(x_2-x_1)$$

# Linear Lagrange Interpolation

- Solve for *p(x)*:

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

- Or:

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

- Or

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

# Quadratic Lagrange Interpolation

- If we again Taylor expand and assume the function is a polynomial, p(x) of degree 2, then
  - $f(x_1)=p(x)+(x_1-x)p'(x)+(x_1-x)^2p''(x)/2$ --- (3)
  - $f(x_2)=p(x)+(x_2-x)p'(x)+(x_2-x)^2p''(x)/2$ --- (4)
  - $f(x_3)=p(x)+(x_3-x)p'(x)+(x_3-x)^2p''(x)/2$ --- (5)
- With a bit more algebra (fairly lengthy) we can eliminate both p'(x) and p''(x) using (3),(4),(5) to get

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

# Lagrange Polynomials

- In general,

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

# Disadvantages

- Although the computation of $P_n(x)$ is simple, the method is still not particularly efficient for large values of $n$.
- When $n$ is large and the data for $x$ is ordered, some improvement in efficiency can be obtained by considering only the data pairs in the vicinity of the $x$ value for which $P_n(x)$ is sought.
- The price of this improved efficiency is the possibility of a poorer approximation to $P_n(x)$.

# Newton's Interpolating Polynomials

- In general, $n$+1 data points can be fitted by an $n$th-order polynomial of the form

$$f_n(x) = b_0 + b_1(x - x_0) + \ldots + b_n(x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

- The following equations are used to evaluate the coefficients:

$$b_0 = f(x_0) = f[x_0]$$

$$b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = f[x_1, x_0]$$

$$b_2 = \frac{\dfrac{f(x_2) - f(x_1)}{x_2 - x_1} - \dfrac{f(x_1) - f(x_0)}{x_1 - x_0}}{x_2 - x_0} = f[x_2, x_1, x_0]$$

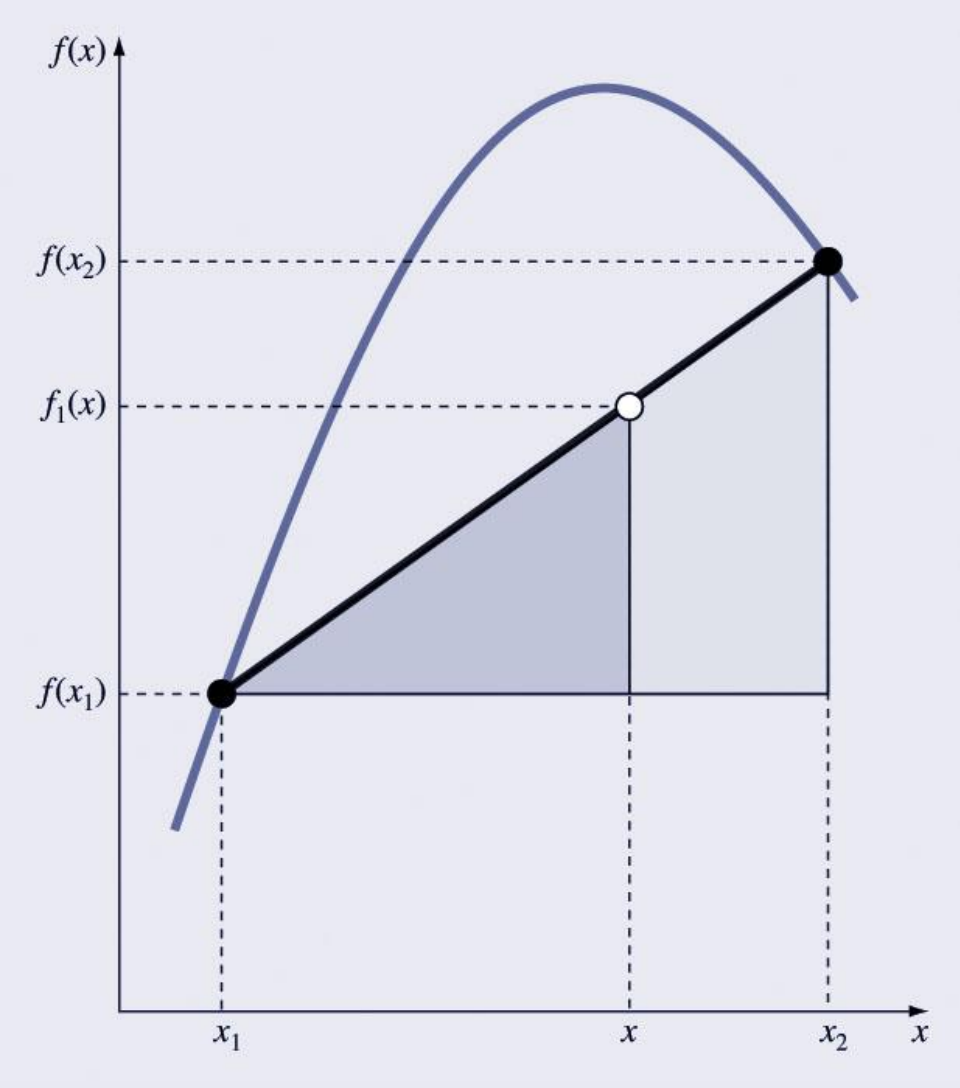
$$\vdots$$

$$b_n = f[x_n, x_{n-1}, \cdots, x_1, x_0]$$

# Newton Interpolating Polynomials

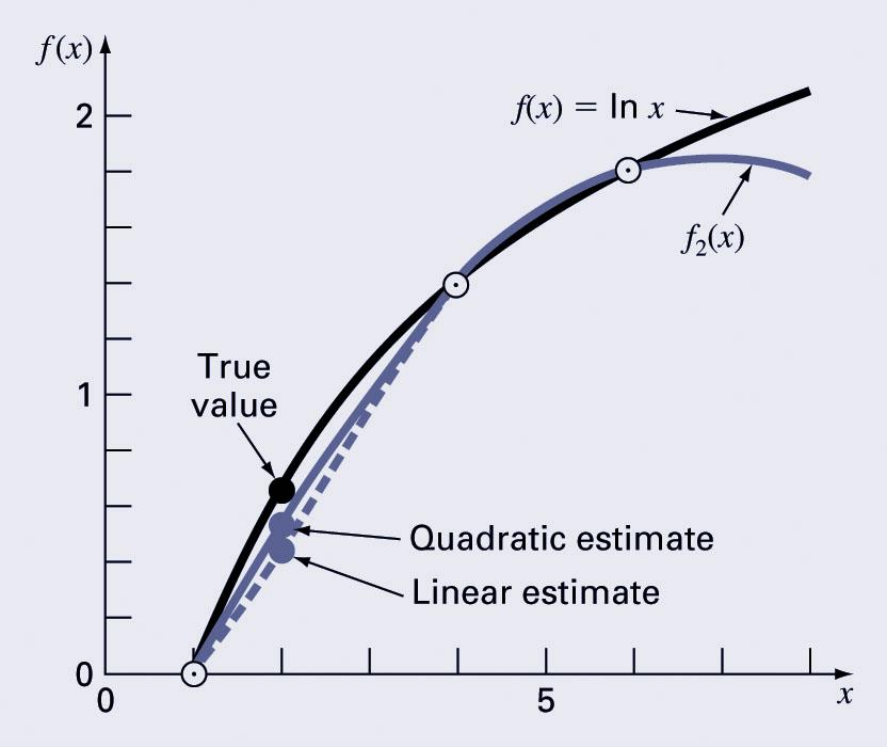
- The first-order Newton interpolating polynomial may be obtained from linear interpolation and similar triangles, as shown.
- The resulting formula based on known points $x_1$ and $x_2$ and the values of the dependent function at those points is:

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



# Newton Interpolating Polynomials

- The second-order Newton interpolating polynomial introduces some curvature to the line connecting the points, but still goes through the first two points.
- The resulting formula based on known points $x_1$, $x_2$, and $x_3$ and the values of the dependent function at those points is:




$$f_2(x) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1}(x - x_1) + \frac{\frac{f(x_3) - f(x_2)}{x_3 - x_2} - \frac{f(x_2) - f(x_1)}{x_2 - x_1}}{x_3 - x_1}(x - x_1)(x - x_2)$$

# Piecewise Polynomials

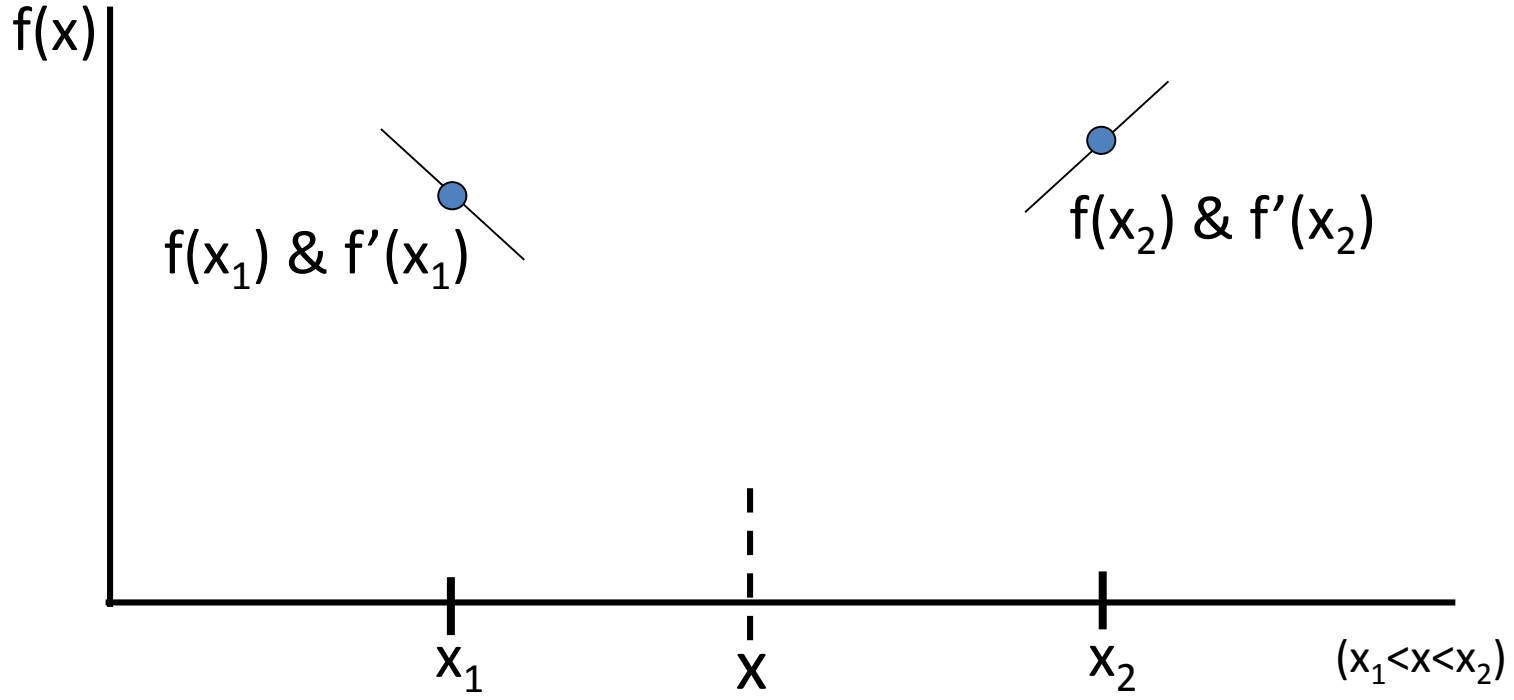
- Fitting single polynomial to large number of data points is likely to yield unsatisfactory oscillating behavior in interpolant.
- Piecewise polynomials provide alternative to practical and theoretical difficulties with high-degree polynomial interpolation. Main advantage of piecewise polynomial interpolation is that large number of data points can be fit with low-degree polynomials
- In piecewise interpolation of given data points $(x_i, y_i)$, different function is used in each subinterval $[x_i, x_{i+1}]$
- Abscissas $x_i$ are called knots or breakpoints, at which interpolant changes from one function to another

# Piecewise polynomials

- Simplest example is piecewise linear interpolation, in which successive pairs of data points are connected by straight lines
- Although piecewise interpolation eliminates excessive oscillation and nonconvergence, it appears to sacrifice smoothness of interpolating function
- We have many degrees of freedom in choosing piecewise polynomial interpolant, however, which can be exploited to obtain smooth interpolating function despite its piecewise nature

# Hermite Interpolation

- If we have both the function value, *and the derivative* we can *fully constrain* a higher order polynomial through two points



# Hermite Cubic Interpolating Using Two Points

- Given the general form for a cubic

$$p(x) = ax^3 + bx^2 + cx + d \quad \text{the derivative is}$$

$$p'(x) = 3ax^2 + 2bx + c$$

fitting the two points gives

$$f(x_1) = ax_1^3 + bx_1^2 + cx_1 + d$$

$$f(x_2) = ax_2^3 + bx_2^2 + cx_2 + d$$

fitting the derivative s gives

$$f'(x_1) = 3ax_1^2 + 2bx_1 + c$$

$$f'(x_2) = 3ax_2^2 + 2bx_2 + c$$

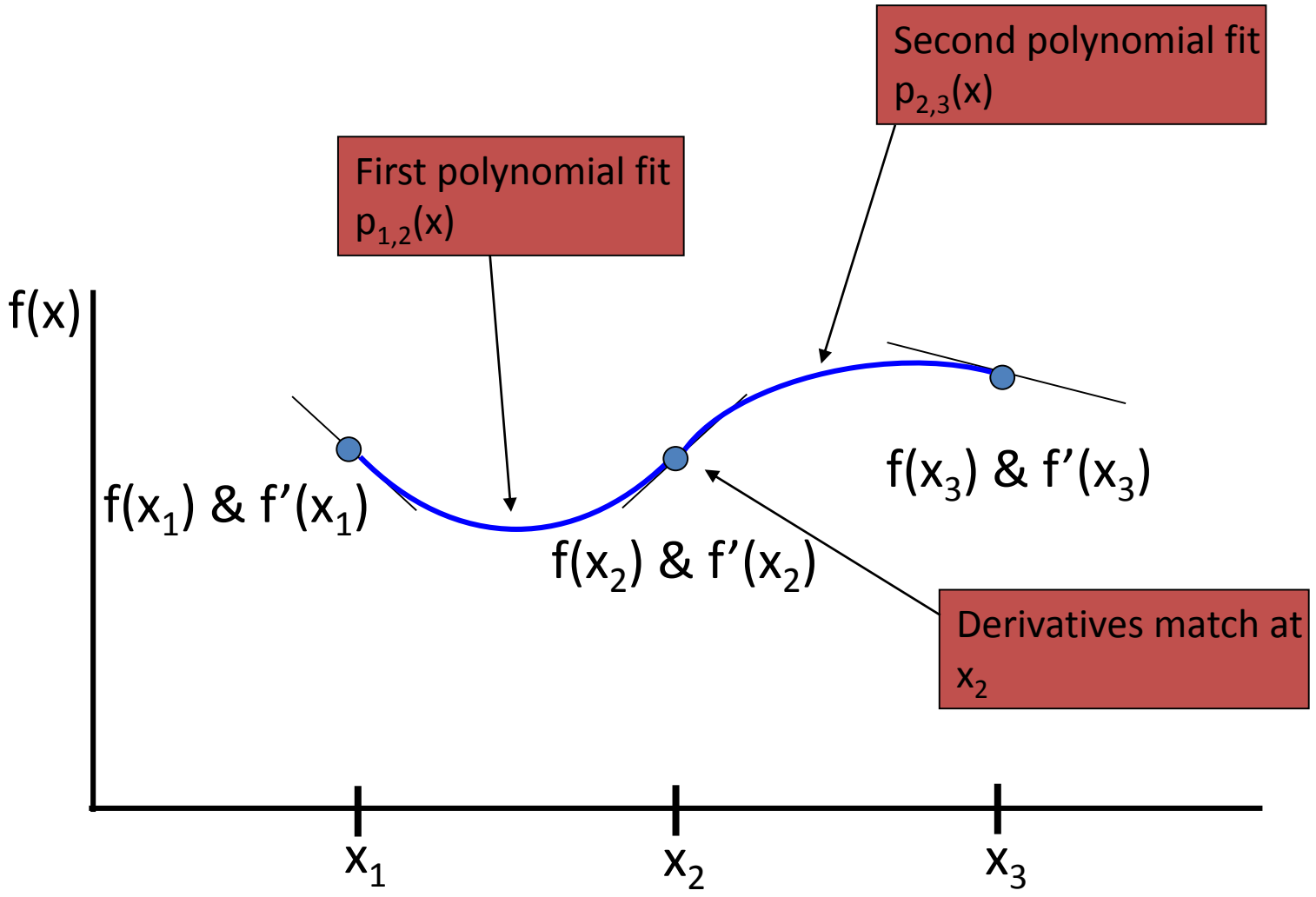
# Finding Hermite Polynomials

- While again somewhat lengthy, we can solve for p(x) in terms of the known values (subscript denotes for $x_1$,$x_2$)

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

- One important property of Hermite interpolation is that if we fit a different polynomial $p_{2,3}(x)$ between $x_2$ & $x_3$ then it will have the same derivative at $x_2$ as $p_{1,2}(x_2)$ *i.e.* $p'_{1,2}(x_2)=p'_{2,3}(x_2)$
  - Interpolation between successive pairs of points is continuous
  - The *derivatives* are also *continuous*
- Hermite interpolation is thus considered to be *smooth*

# Behavior of Hermite Polynomials at Boundaries



# Advantages of Hermite interpolation over Lagrange interpolation

- A series of H.I. polynomials will be continuous at the points, as will the derivatives (*i.e. smooth*)
- While a series of L.I. polynomials is continuous at the points the derivatives will *not* be continuous
- H.I. can fit a cubic to `local' data (*i.e.* $x_1<x<x_2$)
- L.I. requires four points to fit a cubic ($x_1<x_2<x<x_3<x_4$)

# Cubic Spline Interpolation

- If derivatives at each point are not known, can still require that the derivatives are continuous, up to 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, 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 piecewise cubic polynomial interpolant with continuous first derivative

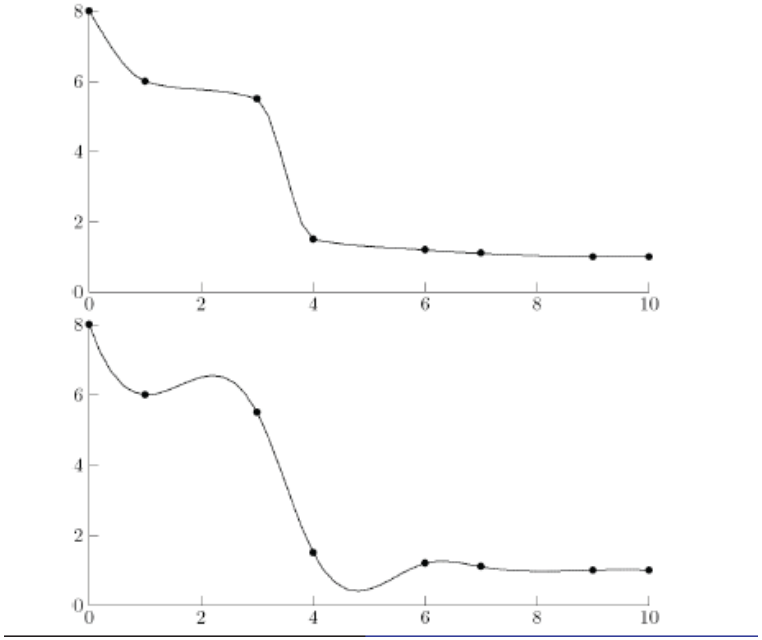
- Piecewise cubic polynomial with n knots has $4(n-1)$ parameters to be determined
- Requiring that it interpolate given data gives $2(n-1)$ equations
- Requiring that it have one continuous derivative gives $n-2$ additional equations, or total of $3n-4$, which still leaves n free parameters
- Thus, Hermite cubic interpolant is not unique, and remaining free parameters can be chosen so that result satisfies additional constraints

Spline is piecewise polynomial of degree k that is $k-1$ times continuously differentiable

- For example, linear spline is of degree 1 and has 0 continuous derivatives, i.e., it is continuous, but not smooth, and could be described as "broken line"
- Cubic spline is piecewise cubic polynomial that is twice continuously differentiable
- As with Hermite cubic, interpolating given data and requiring one continuous derivative imposes $3n-4$ constraints on cubic spline
- Requiring continuous second derivative imposes $n-2$ additional constraints, leaving 2 remaining free parameters

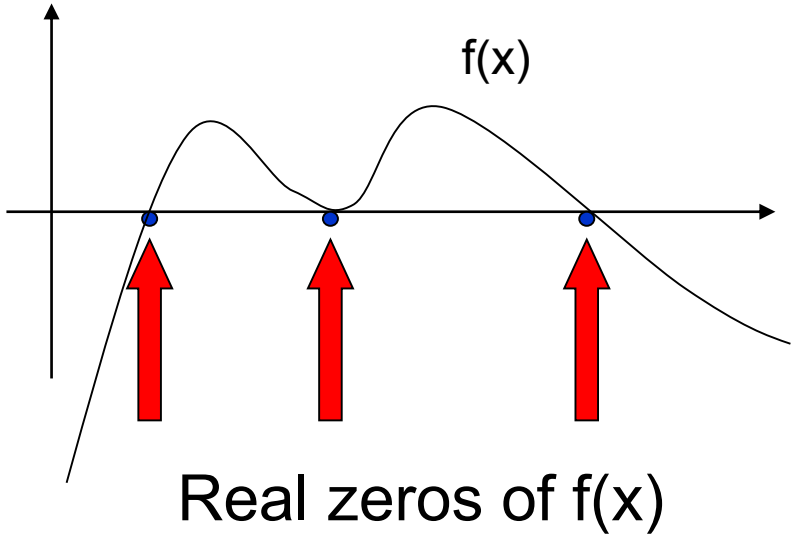
# Hermite vs. Spline

- Choice between Hermite cubic and spline interpolation depends on data to be fit and on purpose for doing interpolation
- If smoothness is of paramount importance, then spline interpolation may be most appropriate
- But Hermite cubic interpolant may have more pleasing visual appearance and allows flexibility to preserve monotonicity if original data are monotonic
- In any case, it is advisable to plot interpolant and data to help assess how well interpolating function captures behavior of original data



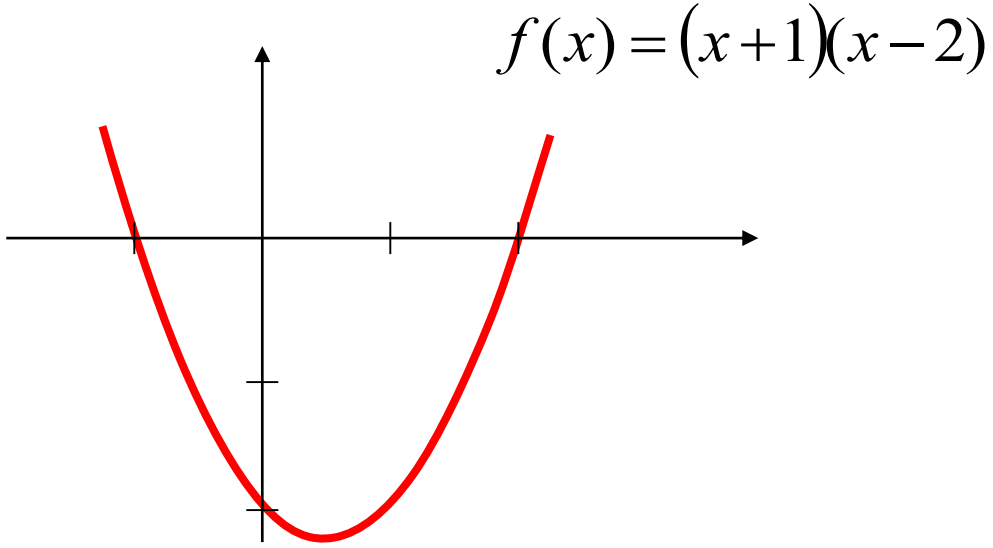
# Root Finding

- Many problems in Physics deal with finding roots of a function (i.e. $f(x)=0$).
- The real zeros of a function $f(x)$ are the values of x at which the graph of the function crosses (or touches ) the x-axis.



# Type of Roots

Simple Zeros

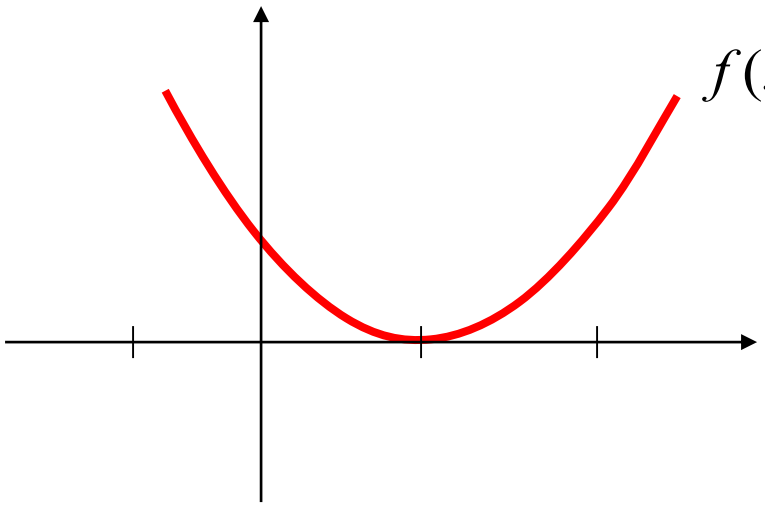



$$f(x) = (x+1)(x-2) = x^2 - x - 2$$

has two simple zeros (one at $x = 2$ and one at $x = -1$)

# Type of Roots

Multiple zeros

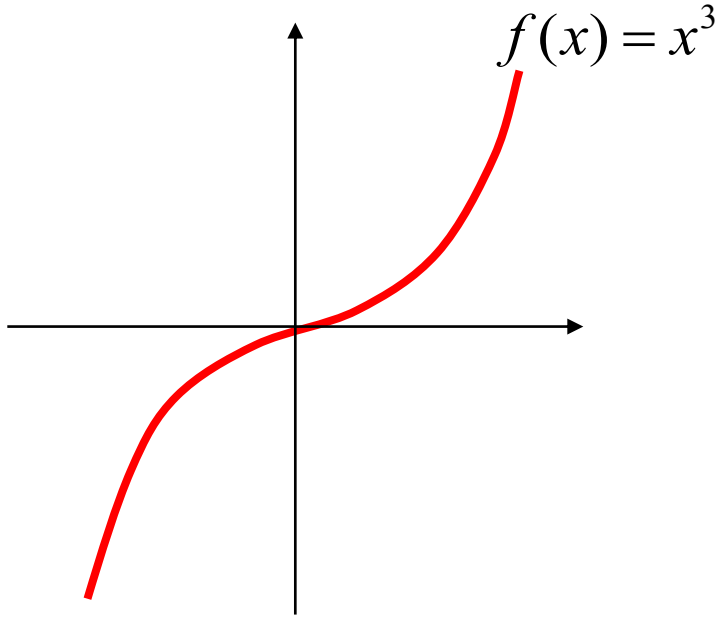



$$f(x)=(x-1)^2=x^2-2x+1$$

has double zeros (zero with mulip licity $=2$) at $x=1$

# Type of Roots

Multiple zeros

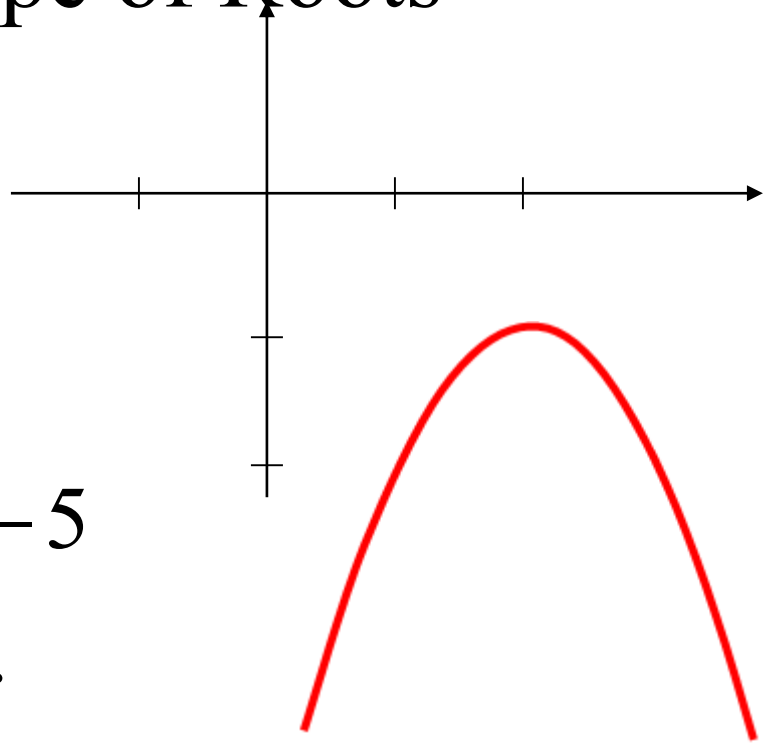



$f(x) = x^3$

has a zero with mulip licity $= 3$ at $x = 0$

# Type of Roots

- No Zeros



$f(x) = -x^2 + 4x - 5$

has no real zeros.

# Roots of a Polynomial

- Polynomials:

$$f_n(x) = \sum_{i=0}^{n} a_i x^i = 0 \quad \text{with } a_n = 1$$

$$n = 1: \; f_1(x) = a_0 + x \quad ; \quad x = -a_0 \text{ is the only root}$$

$$n = 2: \; f_2(x) = a_0 + a_1 x + x^2 \; ; x = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_0}}{2}$$

$$n = 3: \; f_2(x) = a_0 + a_1 x + a_2 x^2 + x^3 \; ; x = ??$$

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

# Polynomials

- Factored form:

$$P(x) = a_n(x - x_1)(x - x_2)\cdots(x - x_n)$$

- N roots.
- N+1 parameters.
- Both real and complex roots.
- No analytical solution for the polynomials of degree 5 or higher.

# Facts about Polynomials

- Any $n^{th}$ order polynomial has exactly n zeros (counting real and complex zeros with 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*** then the function and its first ***(m-1)*** derivatives are zero at ***x=r*** *and the* $\boldsymbol{m}^{th}$ *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 methods
  - Bracketing
  - Open methods

# 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 quickly to the solution (if at all)
  - Or diverges/oscillates
- Examples:
  - 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 ?
  - See Lecture08.ipynb

# Bracketing Methods

- In bracketing methods, the method starts with an interval that contains the root and a procedure is used to obtain a smaller interval containing the root.
- Examples of bracketing methods :
  - Bisection method
  - False position method

# Open Methods

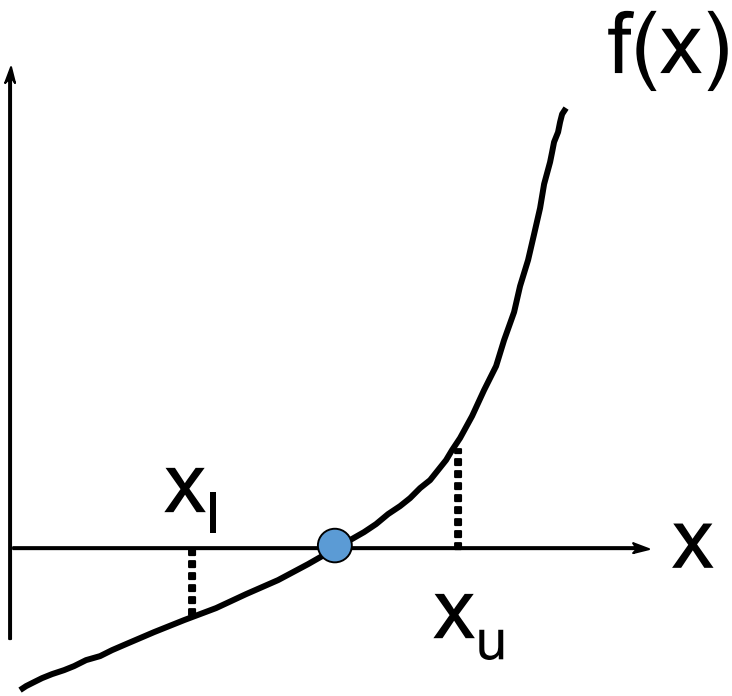
- In the open methods, the method starts with one or more initial guess points. In each iteration a new guess of the root is obtained.
- Open methods are usually more efficient than bracketing methods
- They may not converge to the a root.

# Iterative Root Finding Methods

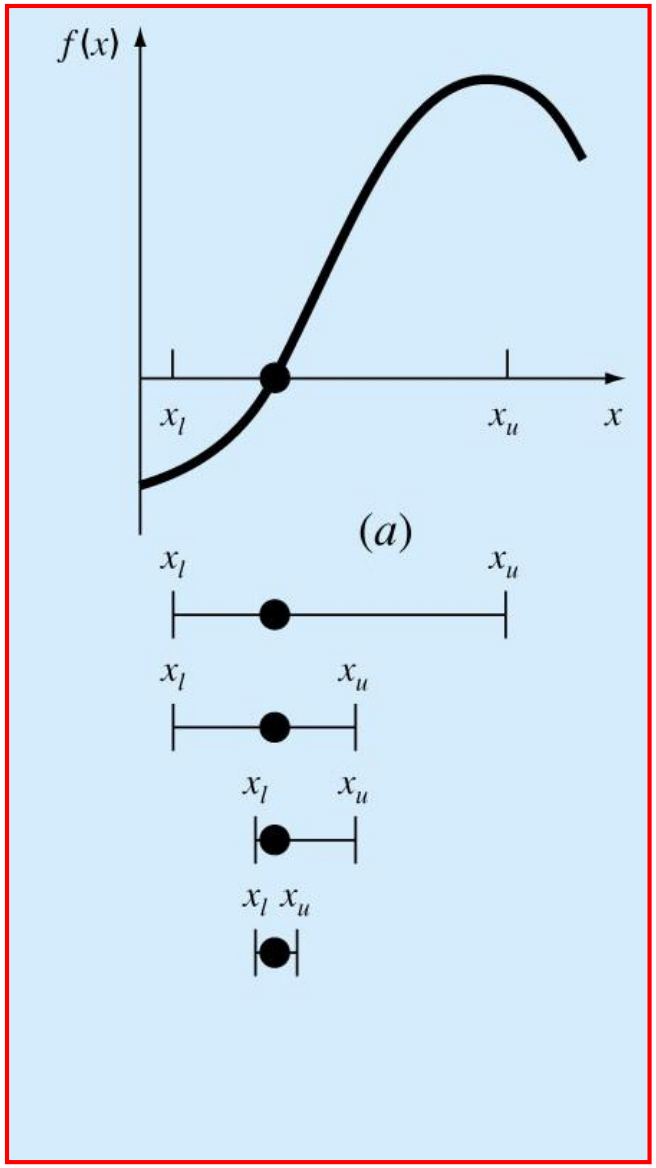
- It is clearly useful to construct numerical algorithms that will take any input f(x) and search for solutions
  - However, there is no perfect algorithm for a given function f(x)
    - Some converge fast for certain f(x) but slowly for others
    - Certain f(x) may not yield numerical solutions
- Above all, you should not use algorithms as black boxes
  - You need to know what the limitations are!
- We shall look at the following algorithms
  - Bisection
  - Newton-Raphson
  - Secant Method
  - Muller's Method

# Bracketing & Bisection

- Bracketing is a very simple idea!
- Given region bounded by $x_l$ and $x_u$, such that the sign of a function f(x) changes between $f(x_l)$ and $f(x_u)$, there must be *at least* one root
  - This is provided that the function does not have any infinities within the bracket – f(x) must be continuous within the region
- The root is said to be *bracketed* by the interval $(x_l,x_u)$

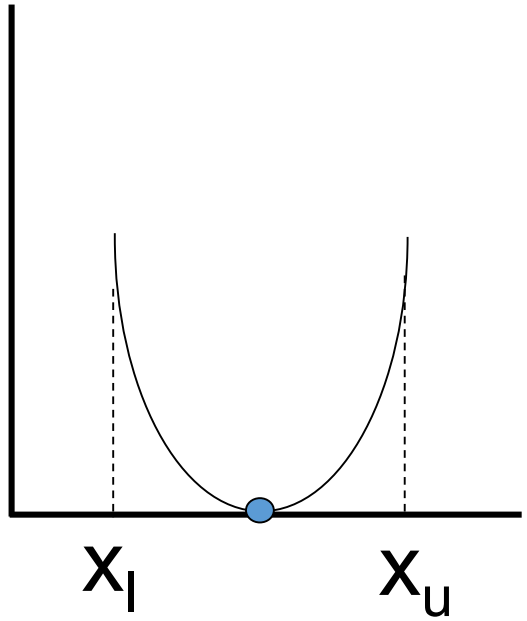


# Bracketing Method

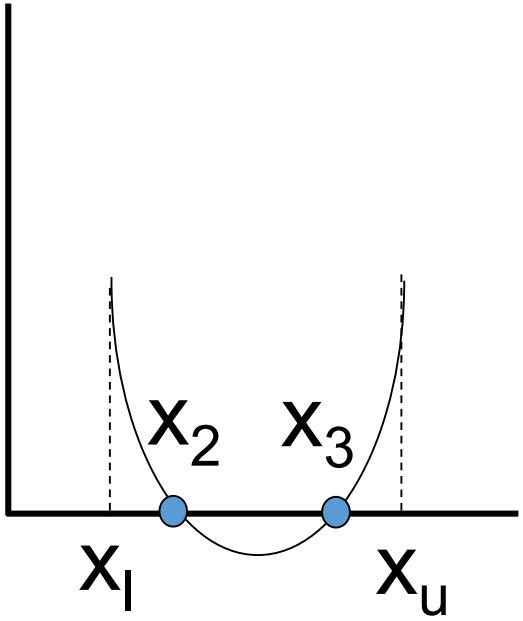


# Other Possibilities

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

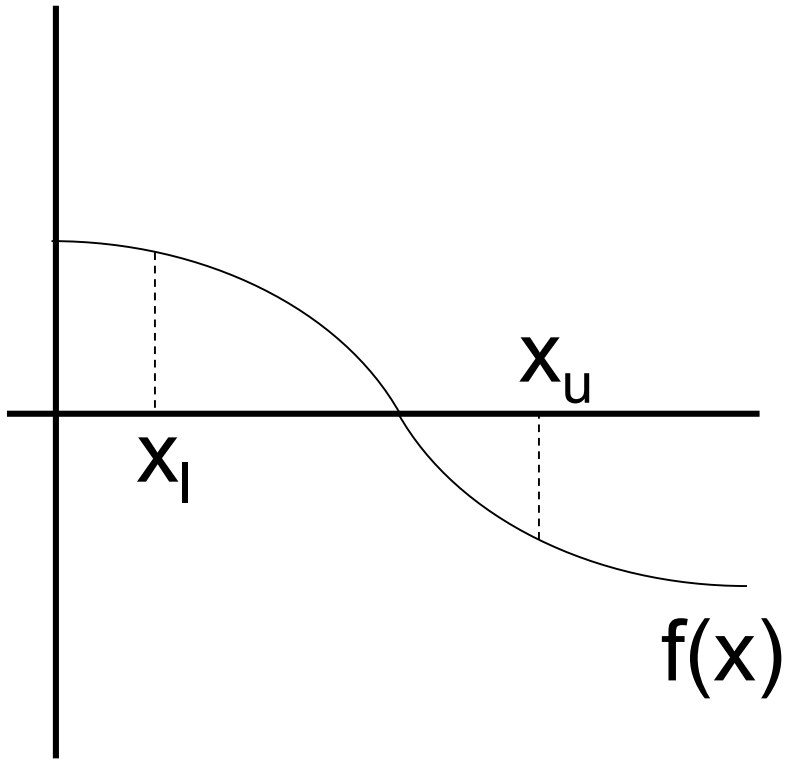



- There could even be 2n roots in this region for functions with roots relatively close to each other!



# Root finding by bisection

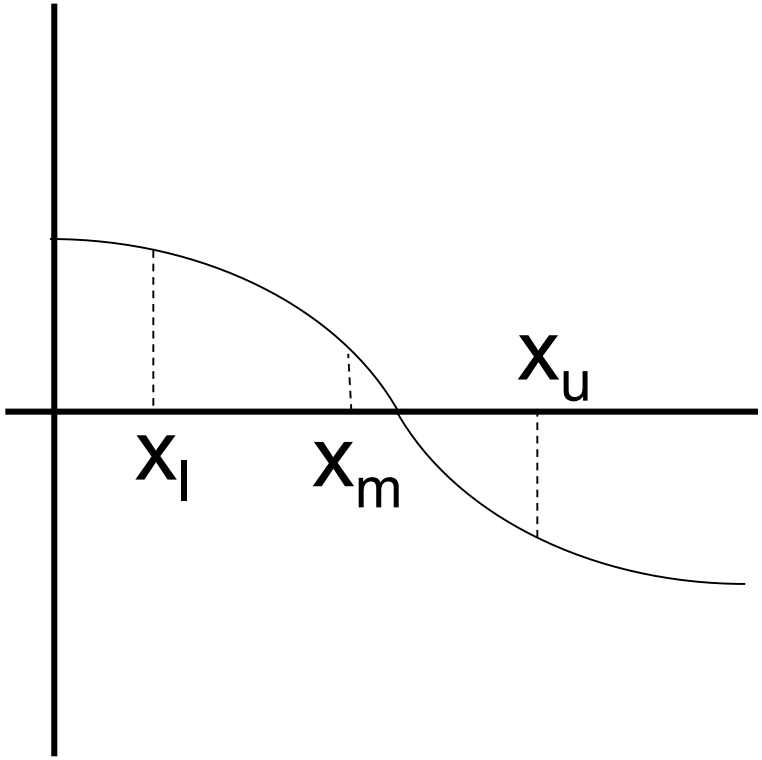
- **Step 1**: Given we know that there exists an $f(x_0)=0$ solution, choose $x_l$ and $x_u$ as the brackets for the root
  - Since the *signs of* $f(x_l)$ & $f(x_u)$ *must be different*, we must have $f(x_l)\times f(x_u) < 0$
  - This step may be non-trivial – we'll come back to it later




For example, for $f(x)=\cos x - x$ we know that $0<x_0<\pi/2$

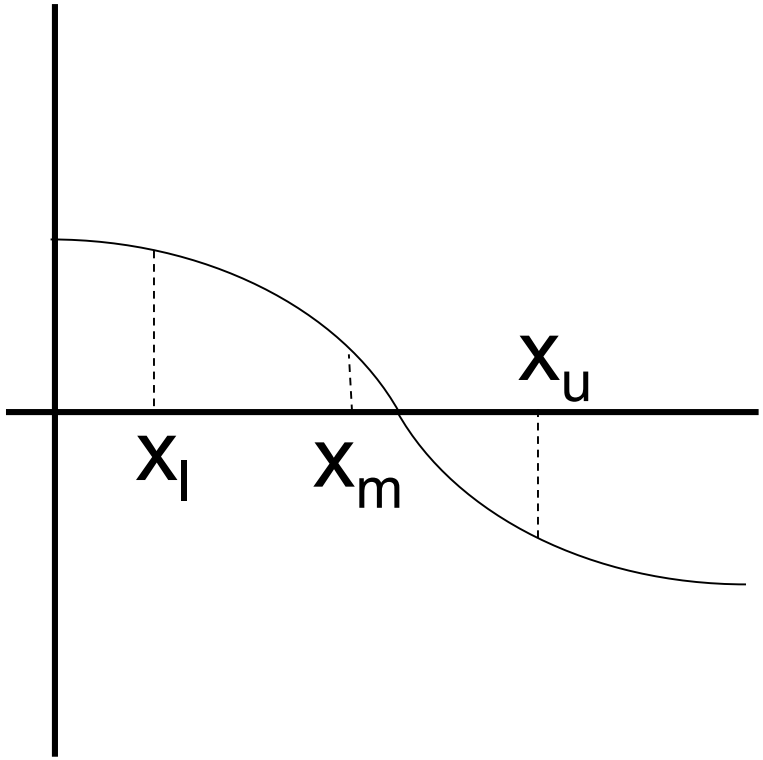
# Bisection: Find the Mid-Point

- Step 2: Let $x_m=0.5(x_l+x_u)$
  - *the mid point between the two brackets*
- This point must be closer to the root than one of $x_l$ or $x_u$
- The next step is to select the new bracket from $x_l,x_m,x_u$
- In this particular case new $x_l$ will be the same as the old $x_m$



# Bisection: Find the New Bracket

- Step 3:
  - If $f(x_l) \times f(x_m) < 0$
    - root lies between $x_l$ and $x_m$
    - Set $x_l = x_l$ ; $x_u = x_m$
  - If $f(x_l) \times f(x_m) > 0$
    - root lies between $x_m$ and $x_u$
    - Set $x_l = x_m$; $x_u = x_u$
  - If $f(x_l) \times f(x_m) = 0$
    - root is $x_m$ you can stop



# Final Step in Bisection

- Step 4: *Test accuracy* (we want to stop at some point)
  - $x_m$ is the best guess at this stage
  - Absolute error is $|x_m - x_0|$ but we don't know what $x_0$ is!
  - Error estimate: $\varepsilon = |x_m^n - x_m^{n-1}|$ where n corresponds to nth iteration
  - Check whether $\varepsilon < \delta$ where $\delta$ is the *tolerance* that you have to decide for yourself at the *start* of the root finding
    - If $\varepsilon < \delta$ then stop
    - If $\varepsilon > \delta$ then return to step 1

# Pros & Cons of Bisection

- Pro: Bisection is Robust
  - You will *always* find a root
  - For this fact alone it should be taken seriously
- Pro: can be programmed very straightforwardly
- Con: bisection converges slowly…
  - Width of the bracket: $x_u^n - x_l^n = w^n = w^{n-1}/2$
    - Width halves at each stage
  - We know in advance if the initial bracket's width $w^0$ after n steps the bracket will have width $w^0/2^n$
  - If we have a specified tolerance $\delta$ then this will be achieved when $n = log_2(w^0/\delta)$
  - Bisection is said to converge linearly since $w^{n+1} \propto w^n$

# Newton-Raphson Method

- For many numerical methods, the underlying algorithms begins with a Taylor expansion
- Consider the Taylor expansion of f(x) around a point $x_0$

$$f(x) \cong f(x_0) + (x - x_0) f'(x_0) + \ldots$$

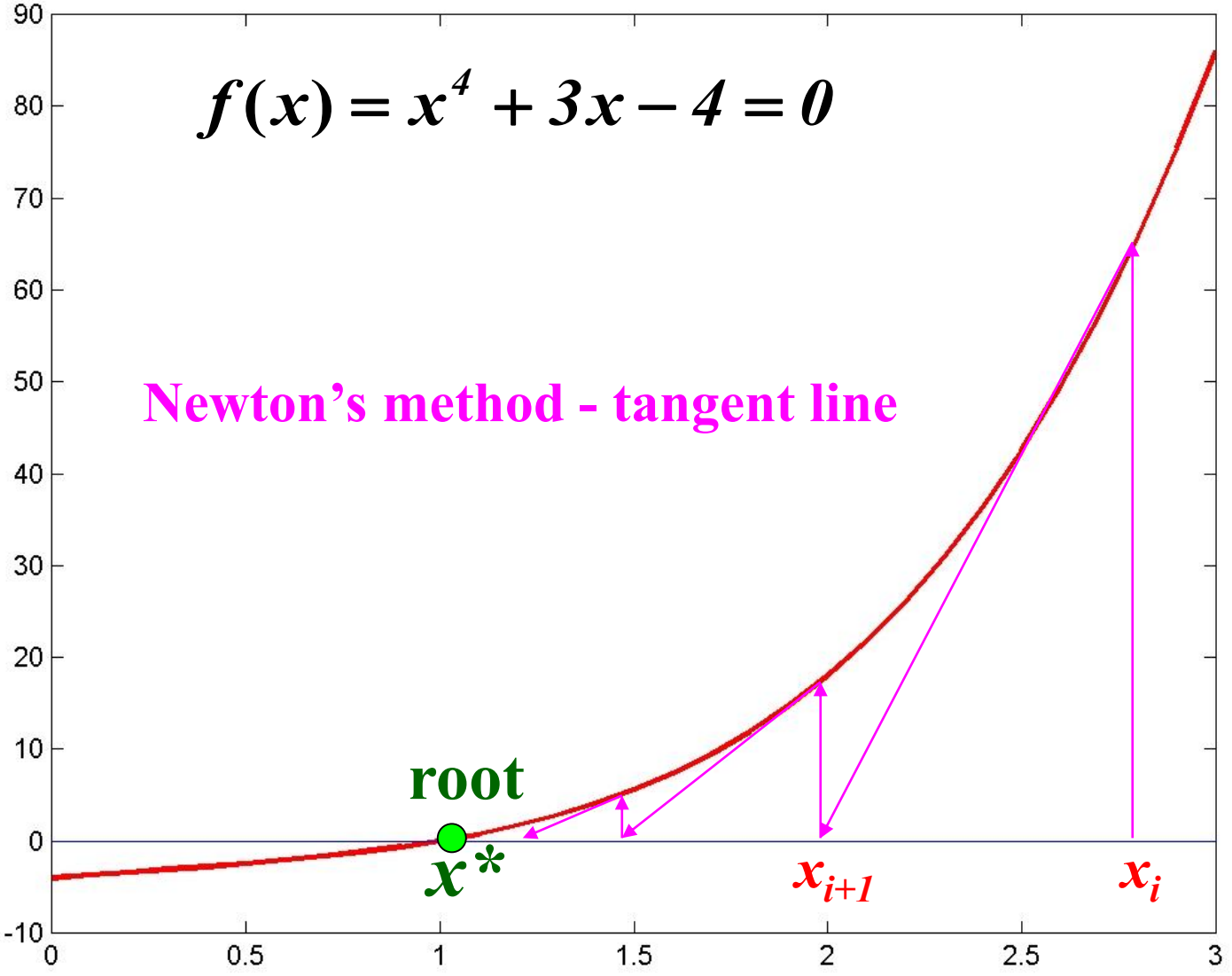
- If x is a root then

$$0 \approx f(x_0) + (x - x_0) f'(x_0)$$

$$\Rightarrow x \approx x_0 - \frac{f(x_0)}{f'(x_0)}$$

- This formula is applicable provided f'($x_0$) is not an extremum (f'($x_0$)≠0))

# Newton-Raphson Method



# Newton Raphson Method

- Step 1: Start at the point $(x_1, f(x_1))$.
- Step 2 : The intersection of the tangent of f(x) at this point and the x-axis.

  $x_2 = x_1 - f(x_1)/f`(x_1)$
- Step 3: Examine if $f(x_2) = 0$

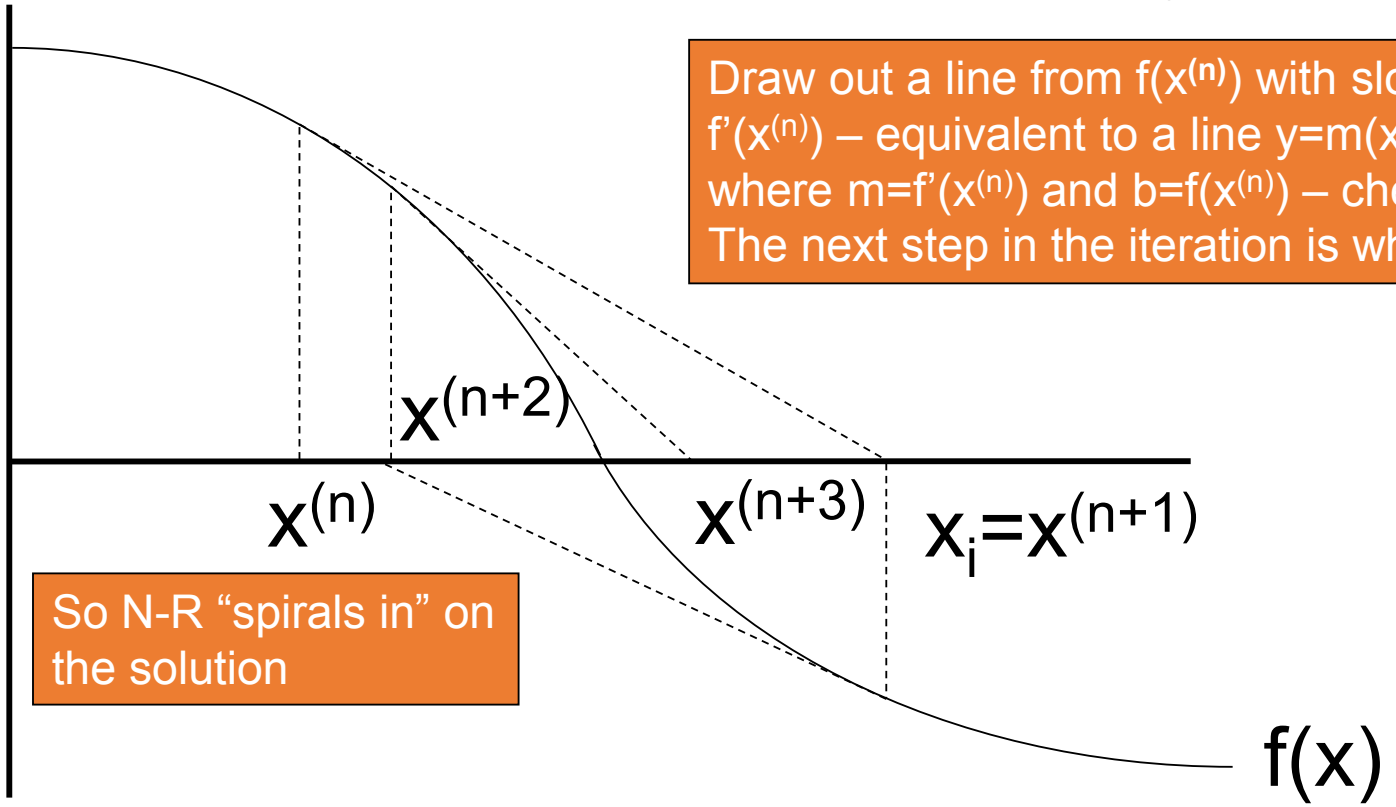
  or $abs(x_2 - x_1) < tolerance$,
- Step 4: If yes, solution $x_r = x_2$

  If not, $x_1 \leftarrow x_2$, repeat the iteration.

# Geometric Interpretation of Newton-Raphson

- The general step in the iteration process is

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



# Newton-Raphson's Method

- **Note that an evaluation of the derivative (slope) is required**
- **You may have to do this numerically**
- **Open Method – Convergence depends on the initial guess (not guaranteed)**
- **However, Newton method can converge very quickly (quadratic convergence)**

# Convergence

The nececessary and sufficient condition :

$$\left|\frac{F(x)F''(x)}{[F'(x)]^2}\right| < 1$$

with the error term of :

$$E(x) = K(\Delta x)^n$$

Doesn't converge when F(x) has multiple roots.

# Newton-Raphson Method

Although Newton-Raphson converges very rapidly, it may diverge and fail to find roots

a) if there are multiple roots

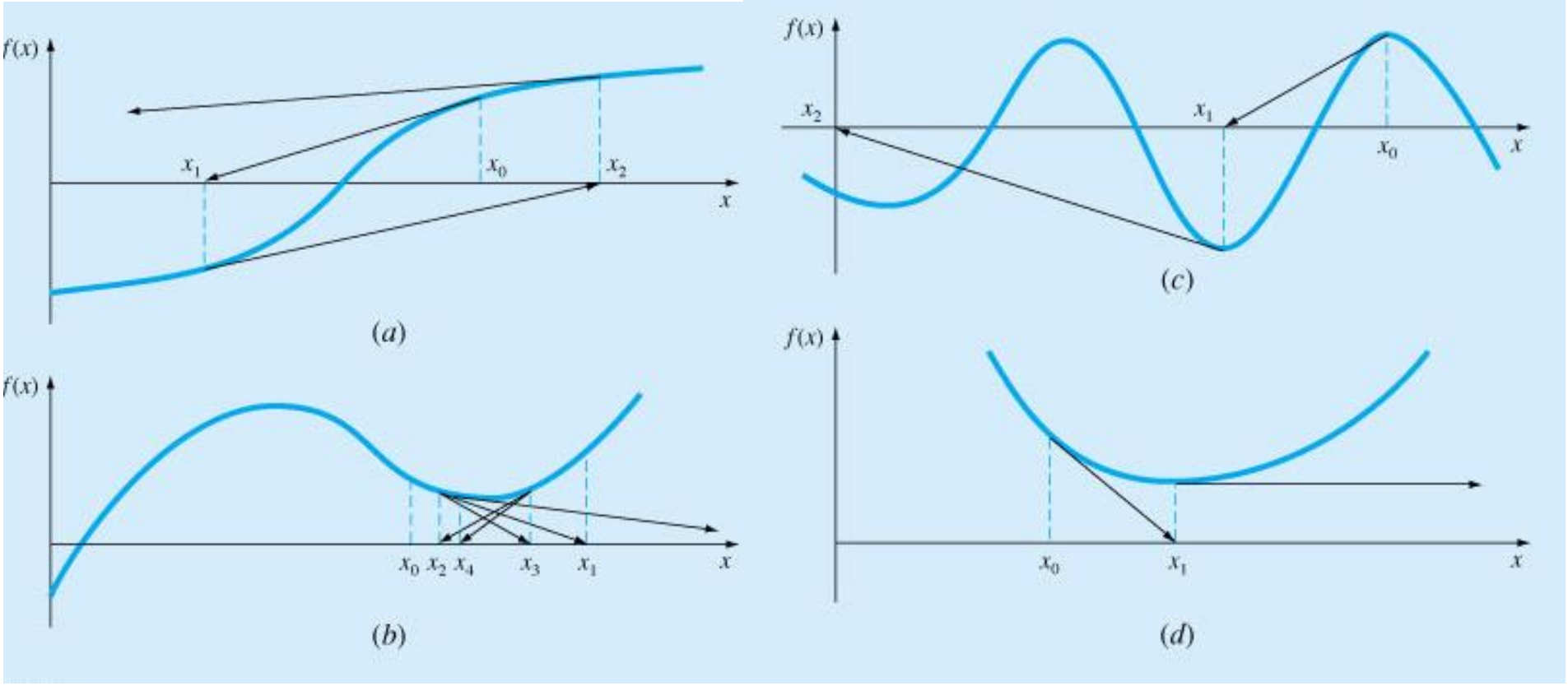
b) if an inflection point (f''=0) is near the root

c) if there is a local minimum or maximum (f'=0)

d) if a zero slope is reached

In Open Methods, Convergence is not always guaranteed

# Newton-Raphson Method




**Examples of poor convergence**

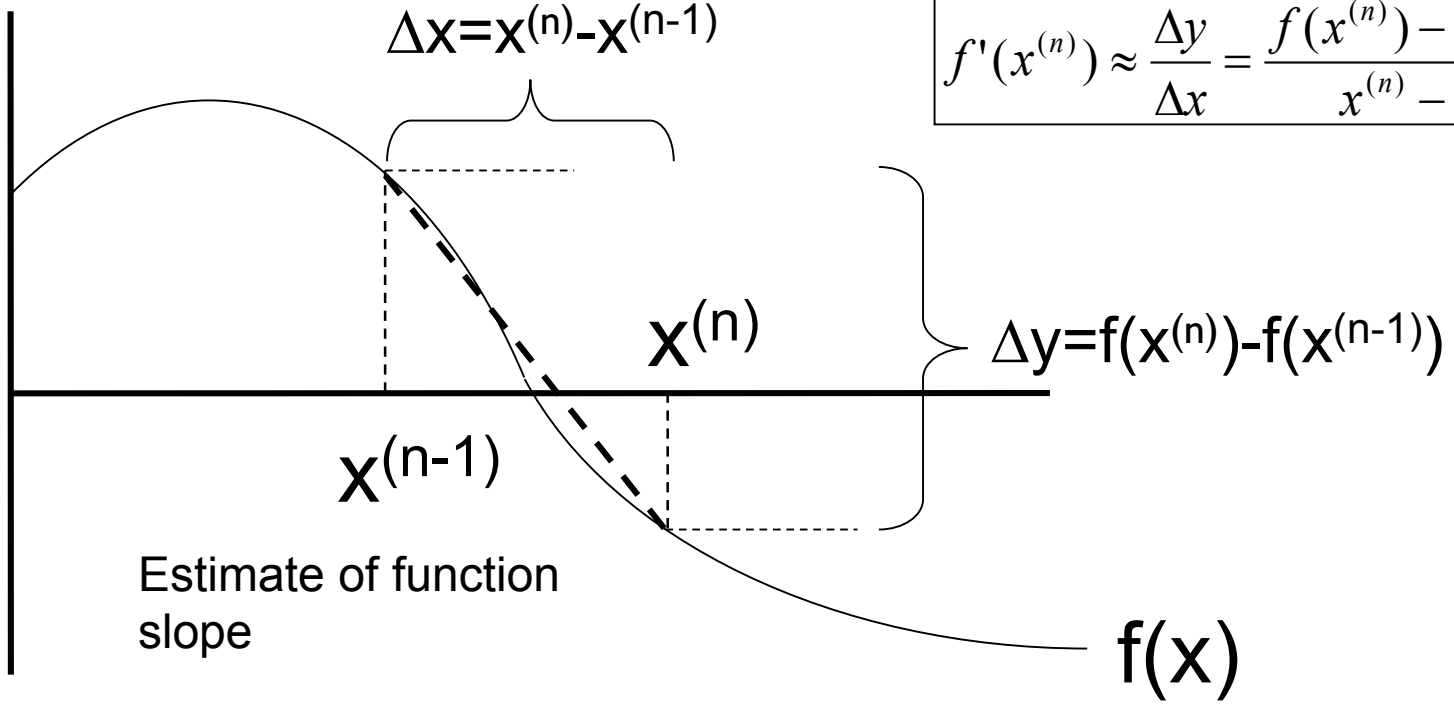
# Newton-Raphson-Bisection Hybrid

- Let's combine the two methods
  - Good convergence speed from NR
  - Guaranteed convergence from bisection
- Start with $x_l<x^{(n)}<x_u$ as being the 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 $x_{NR}^{(n+1)}=x_m^{(n+1)}=0.5(x_l+x_u)$, check and update $x_l$ and $x_u$ values for the new bracket
    - *i.e.* we throw away any bad NR steps and do a bisection step instead

# Secant Method

- One of the key issues in NR method is that you need the derivative
  - May not have it if the function is defined numerically
  - Derivative could be extremely hard to calculate
- You can get an estimate of the local slope using values of $x^{(n)}$,$x^{(n-1)}$, $f(x^{(n)})$ and $f(x^{(n-1)})$

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



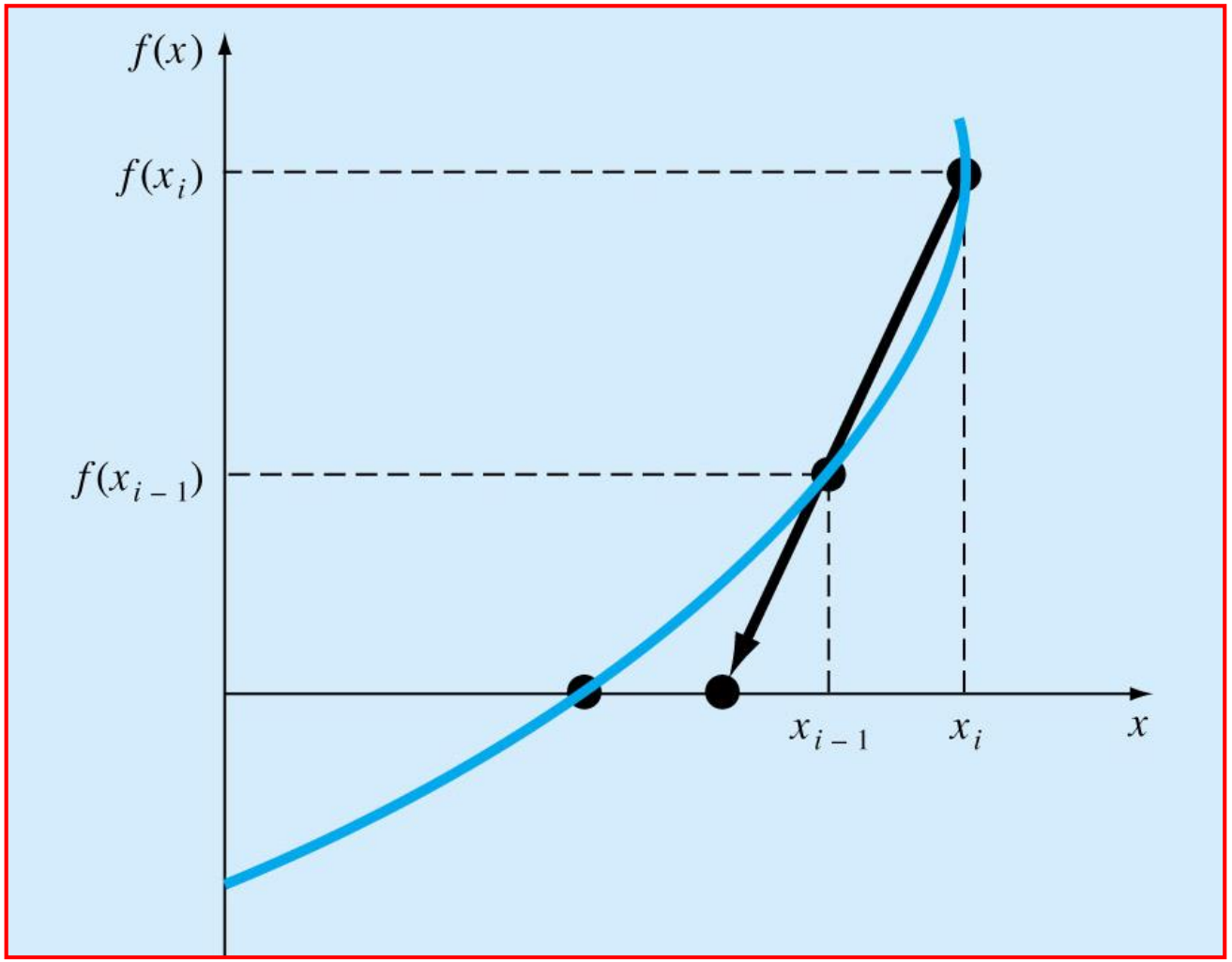
# Secant Method

- So once you initially select a pair of points bracketing the root ($x^{(n-1)}$, $x^{(n)}$), $x^{(n+1)}$ is given by

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

- You then iterate to a specific tolerance value exactly as in NR
- This method can converge almost as quickly as NR

# Secant Method




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

# Algorithm for Secant Method

- **Open Method**

**1. Begin with any two endpoints $[a, b] = [x_0, x_1]$**

**2. Calculate $x_2$ using the secant method formula**

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

**3. Replace $x_0$ with $x_1$, replace $x_1$ with $x_2$ and repeat from (2) until convergence is reached**

- **Use the two most recently generated points in subsequent iterations (not a bracket method!)**

# Secant Method

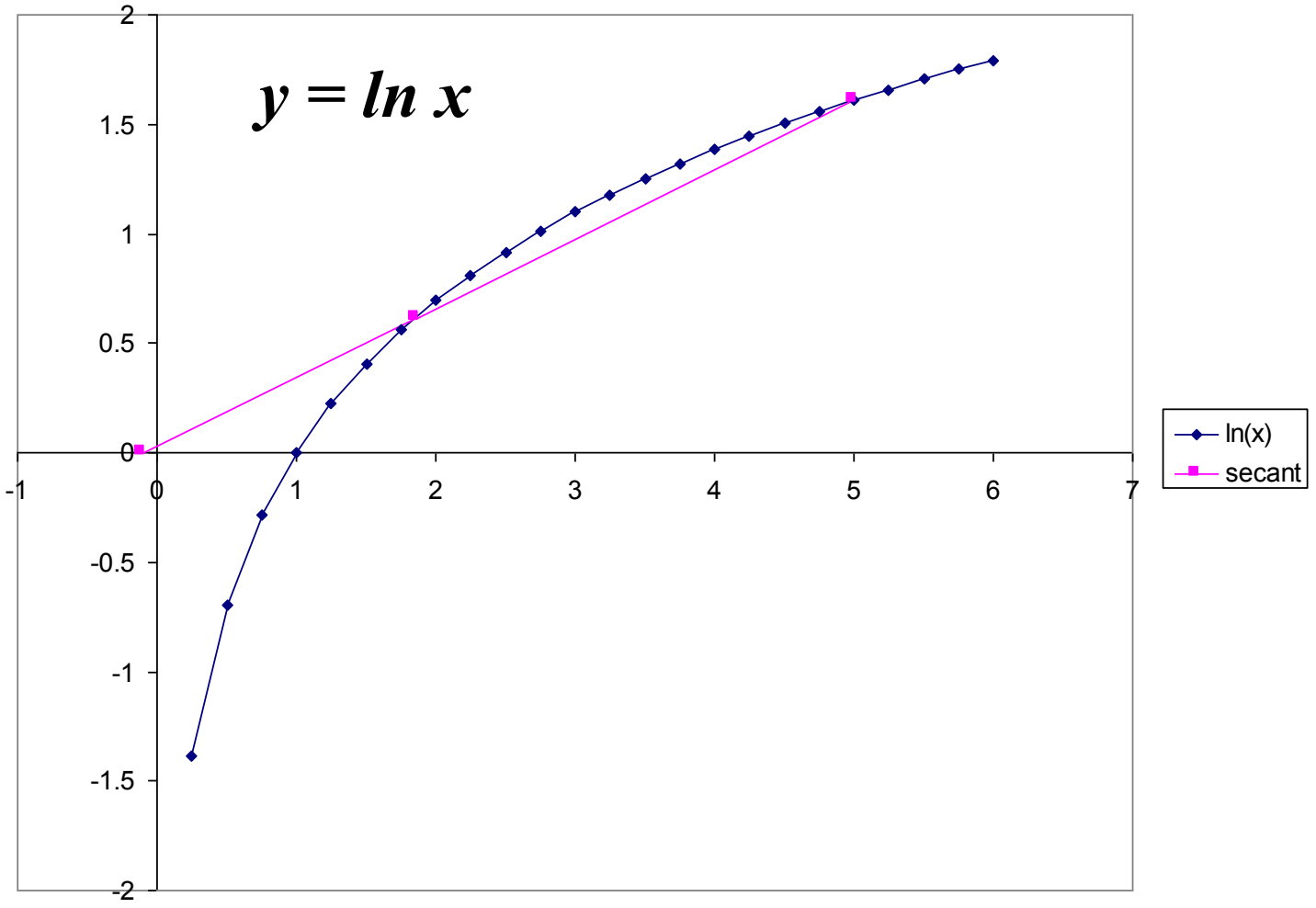
- **Advantage of the secant method -**
- **It can converge even faster and it doesn't need to bracket the root**

- **Disadvantage of the secant method -**
- **It is not guaranteed to converge!**
- **It may diverge (fail to yield an answer)**

# Issues with the Secant Method

- Given the initial values $x_l(=x^{(n-1)}),x_u(=x^{(n)})$ that bracket the root the calculation of $x^{(n+1)}$ is *local*
  - Easy to program, all values are easily stored
- Difficulty is with finding suitable $x_l,x_u$ since this is a global problem
  - Determined by f(x) over its entire range
  - Consequently difficult to program (no perfect method)

# Convergence not Guaranteed



# Müller-Brent Method

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

# Müller Method

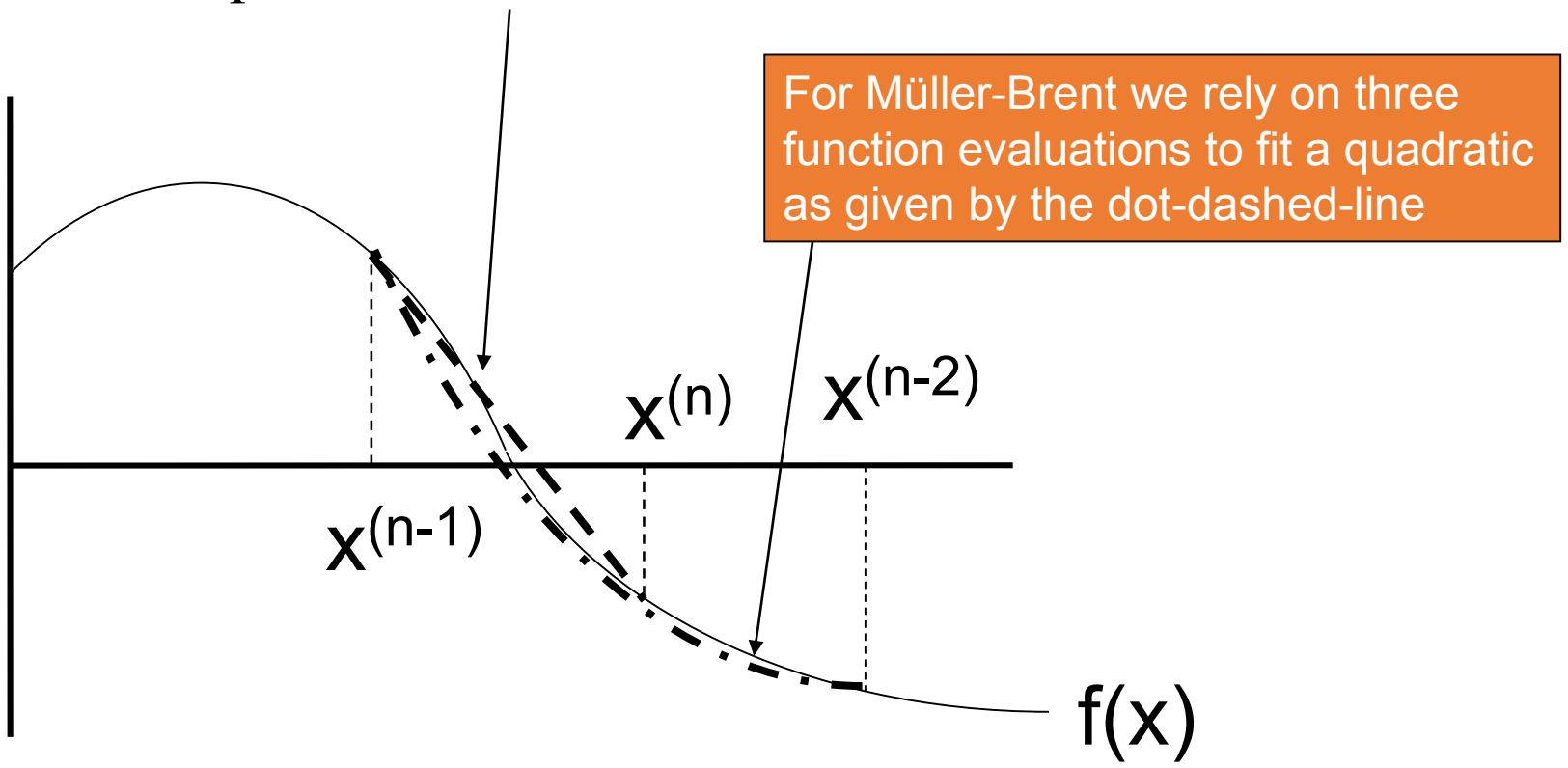
- The method consists of deriving the coefficients of parabola that goes through the three points:

1. 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

$$q(x) = a(x - x^{(n)})^2 + b(x - x^{(n)}) + c$$

- Since we know q(x) passes through all three points we have

$$q(x^{(n)}) = f(x^{(n)}) \Rightarrow c = f(x^{(n)})$$

$$q(x^{(n-1)}) = f(x^{(n-1)}) = a(x^{(n-1)} - x^{(n)})^2 + b(x^{(n-1)} - x^{(n)}) + f(x^{(n)})$$

$$q(x^{(n-2)}) = f(x^{(n-2)}) = a(x^{(n-2)} - x^{(n)})^2 + b(x^{(n-2)} - x^{(n)}) + f(x^{(n)})$$

- Which gives two equations for two unknowns, a & b

# Solving for Coefficients

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

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

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

- We now have a,b,c so we can use the general formula to calculate the zeros for $x^{(n+1)}$-$x^{(n)}$

$$x^{(n+1)} - x^{(n)} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$