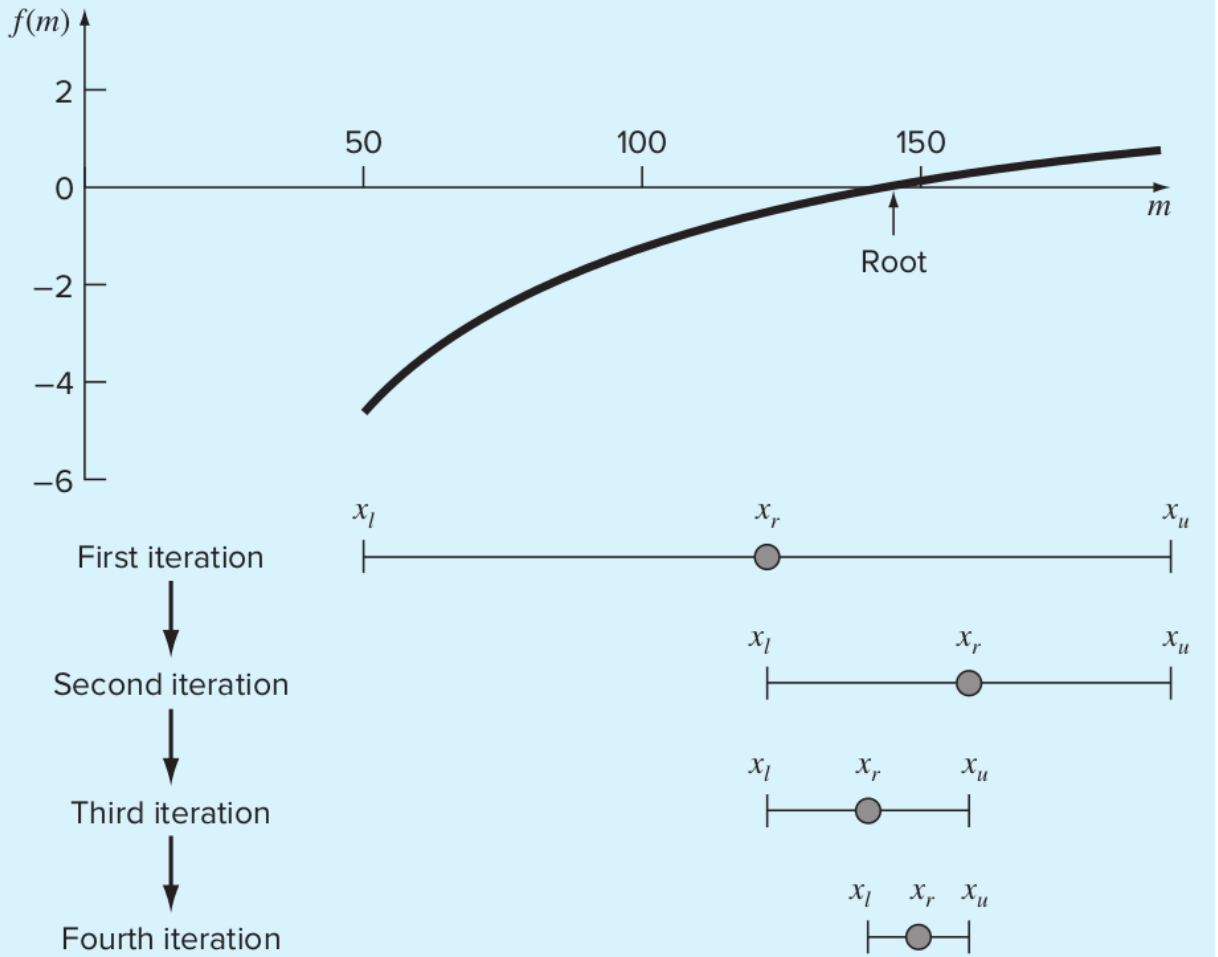
5.1 Bisection Method



- Bracketed method so needs 2 initial guesses.
- Usually faster than incremental search.

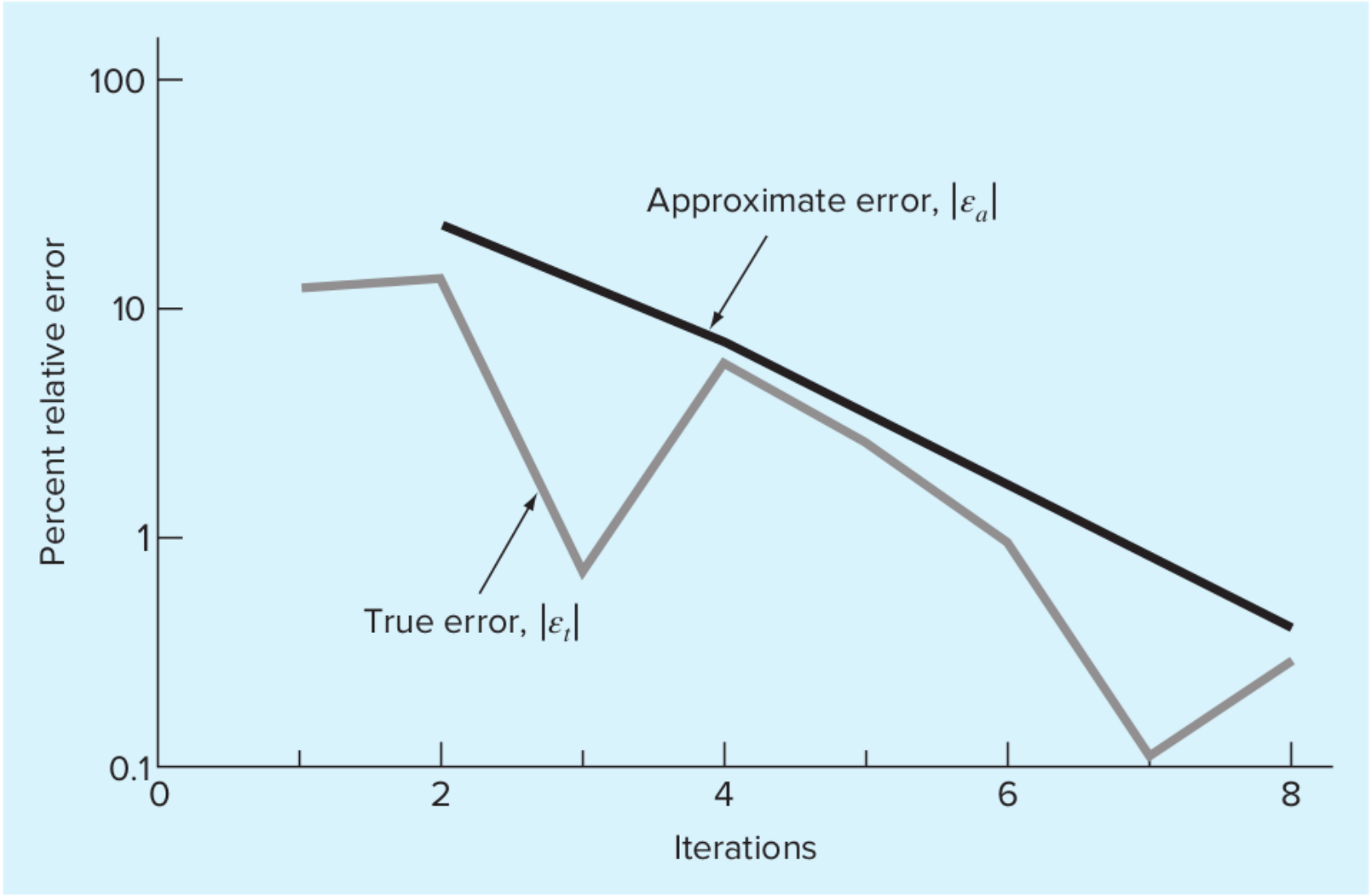
Algorithm

- Choose initial points *a* and *b* then check if $f(a)f(b) < 0$.
- If not must choose other points (maybe using Incremental Search).
- Calculate halfway point, x_0.
- Check which one of $f(a)f(x_o) < 0$ or $f(b)f(x_o) < 0$ then choose appropriate next interval.
- Repeat this process until stopping criterion is met (decimal places, number of iterations etc.).

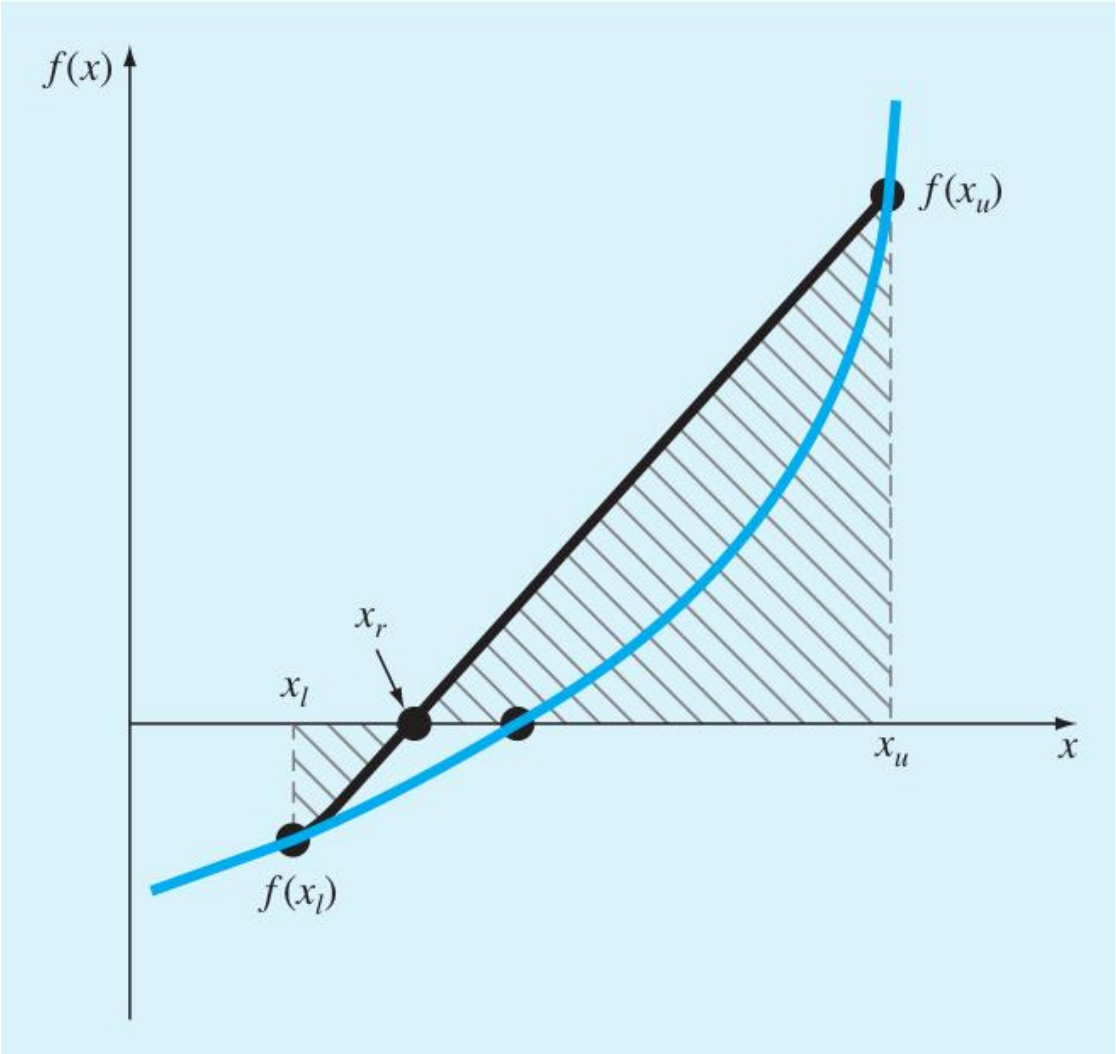
Convergence of Bisection Method

- Typical convergence tends to be quite erratic. True error goes up and down until it settles down near the root since the error is low when the root is near the midpoint of an interval, but high when the root is near the endpoints of an interval.

Iteration	a	b	x_0	$\lvert\varepsilon_a\rvert$ (%)	$\lvert\varepsilon_t\rvert$ (%)
1	50	200	125		12.43
2	125	200	162.5	23.08	13.85
3	125	162.5	143.75	13.04	0.71
4	125	143.75	134.375	6.98	5.86
5	134.375	143.75	139.0625	3.37	2.58
6	139.0625	143.75	141.4063	1.66	0.93
7	141.4063	143.75	142.5781	0.82	0.11
8	142.5781	143.75	143.1641	0.41	0.30


100
10
1
0.1
Percent relative error
Approximate error, $|\varepsilon_a|$
True error, $|\varepsilon_t|$
0
2
4
6
8
Iterations

5.2 False Position Method (Bracketed)



- Similar to Bisection Method but instead of bisecting the interval we draw a straight line between the bracketed points to get the next position.

- Usually faster than Bisection however sometimes Bisection is faster. It depends on the problem.

Algorithm

- Choose initial points *a* and *b* then check if *f(a)f(b)* < 0. If not must choose other points.

- Calculate next point from straight line:

$$x_0 = a - \frac{f(a)(b-a)}{f(b) - f(a)}$$

- Check which one of $f(a)f(x_o) < 0$ or $f(b)f(x_o) < 0$ then choose appropriate next interval.

- Repeat this process until stopping criterion is met (decimal places, number of iterations etc.).

5.3 Fixed-Point Iteration (Open)

- An open method that uses successive iterations to obtain a new *x* value from an old one.
- Rearrange $F(x) = 0 \rightarrow x = g(x)$.
- Write as an iterative formula $x_{i+1} = g(x_i)$.
- Only requires 1 initial guess unlike the bracketing methods.

EXAMPLE 1

Use simple fixed-point iteration to locate the root of $f(x) = e^{-x} - x$.

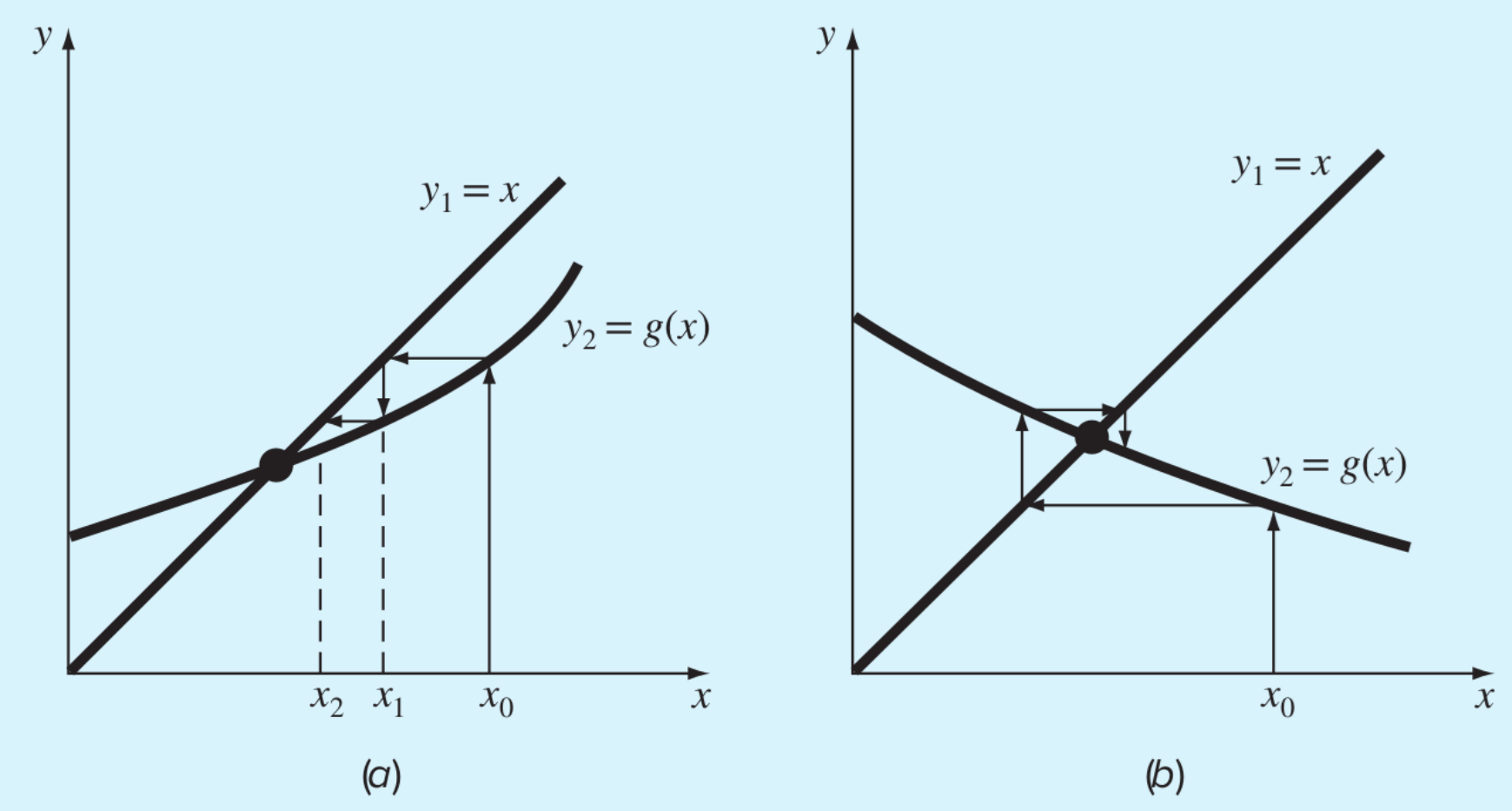
$$\longrightarrow \quad x_{i+1} = e^{-x_i}$$ **Iterative formula**

Try initial guess: $x_0 = 0$

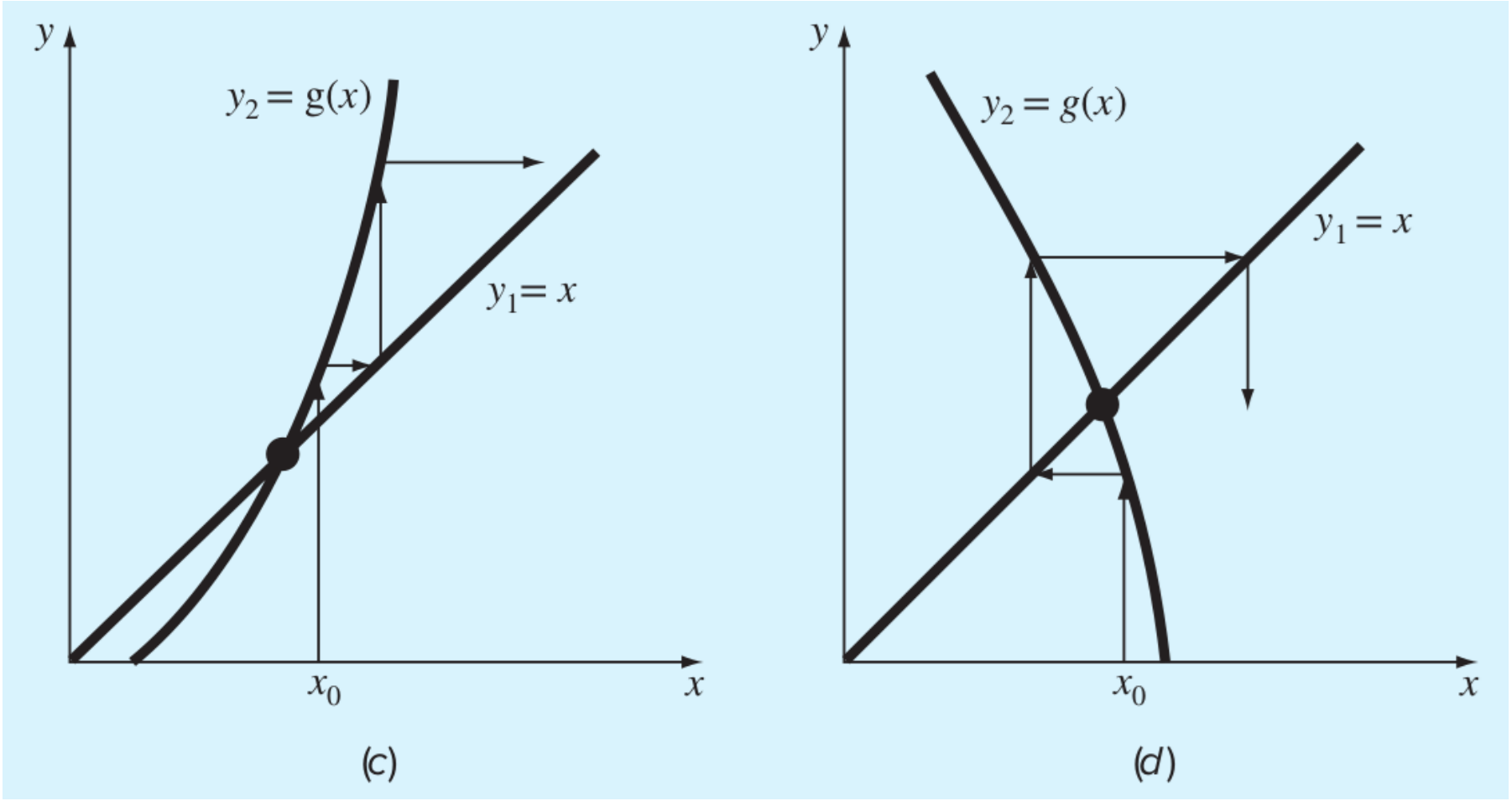
i	x_i	$\lvert\varepsilon_a\rvert$, %	$\lvert\varepsilon_t\rvert$, %	$\lvert\varepsilon_t\rvert_i / \lvert\varepsilon_t\rvert_{i-1}$
0	0.0000		100.000	
1	1.0000	100.000	76.322	0.763
2	0.3679	171.828	35.135	0.460
3	0.6922	46.854	22.050	0.628
4	0.5005	38.309	11.755	0.533
5	0.6062	17.447	6.894	0.586
6	0.5454	11.157	3.835	0.556
7	0.5796	5.903	2.199	0.573
8	0.5601	3.481	1.239	0.564
9	0.5711	1.931	0.705	0.569
10	0.5649	1.109	0.399	0.566

- Roughly linear convergence for $|\varepsilon_a|$.
- The true value for the root is 0.5671 (Lambert W Function).

- Method is convergent when $|g'| < 1$.



- Method is divergent when $|g'| > 1$.



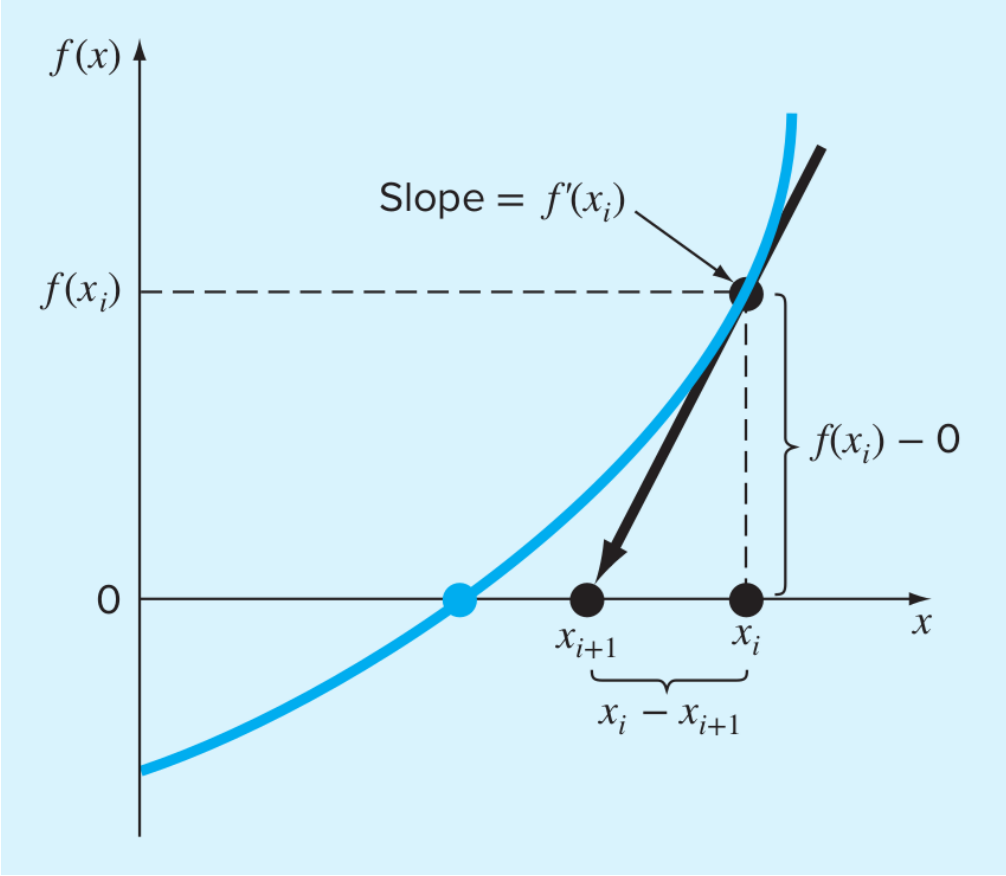
Algorithm

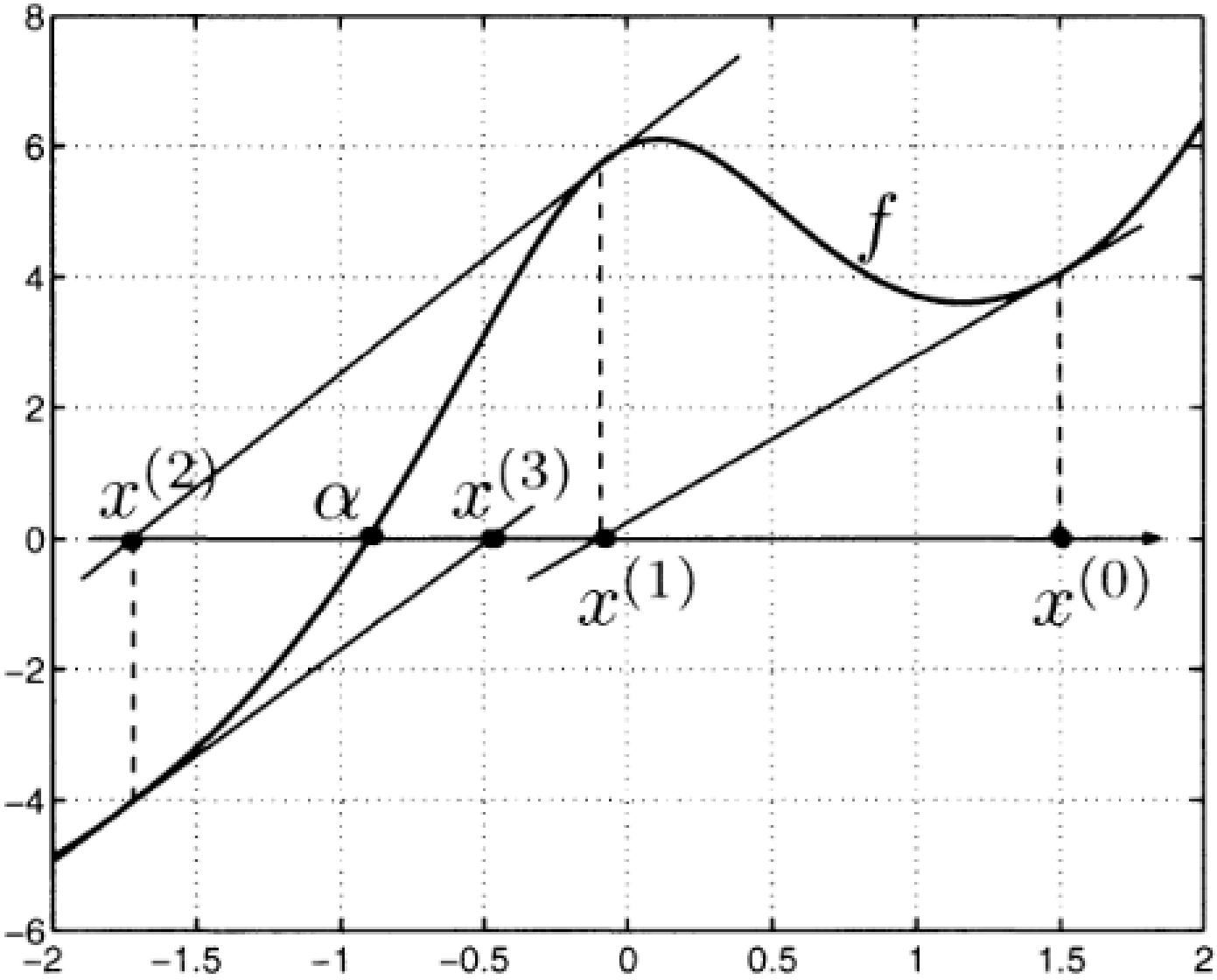
- Pick initial value.
- Calculate $x_1 = g(x_0)$.
- Check that $|g'(x_0)| < 1$ otherwise display an error/warning.
- Calculate $x_2 = g(x_1)$ etc. until stopping criterion is met.

5.4 Newton-Raphson Method (Open)

- Possibly the most commonly used method.
- Take initial guess and draw tangent line.
- Intersection with *x*-axis is the new *x* value.

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




8
6
4
2
0
−2
−4
−6
−2
−1.5
−1
−0.5
0
0.5
1
1.5
2
f
$x^{(2)}$
α
$x^{(3)}$
$x^{(1)}$
$x^{(0)}$

Algorithm

- Pick initial value.
- Calculate successive *x* values using: $x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$
- Continue until stopping criterion is met.

EXAMPLE 2

Use the Newton-Raphson method to locate a root of:

$$f(x) = e^{-x} - x$$

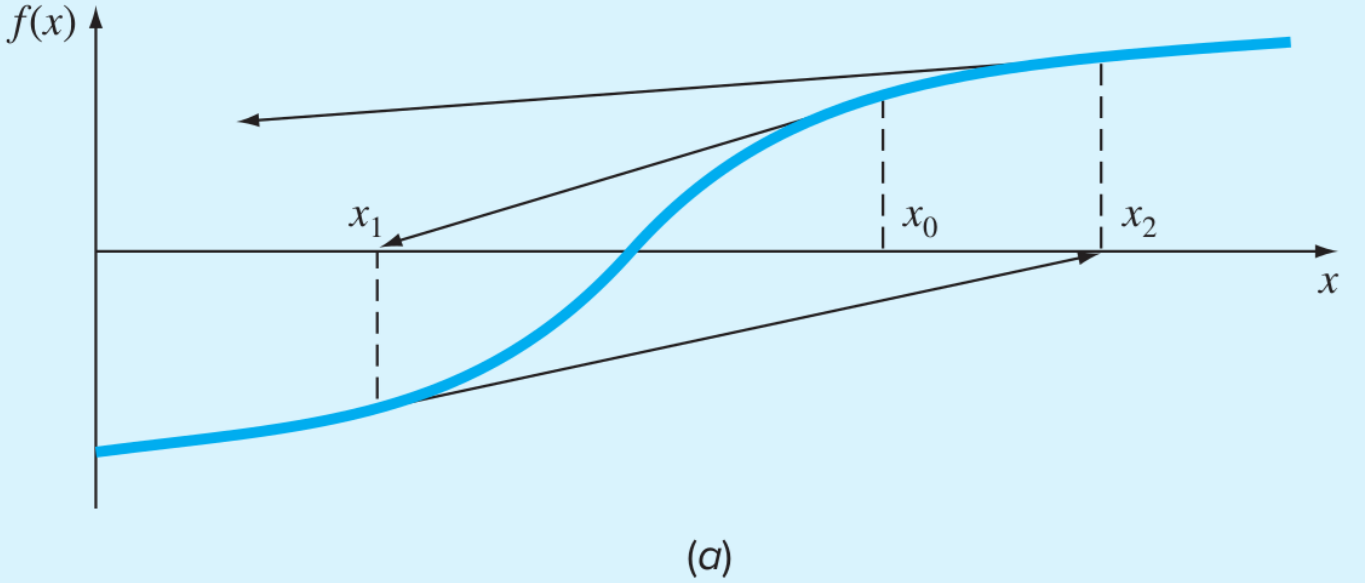
Calculate derivative $$f'(x) = -e^{-x} - 1$$

Write formula $$x_{i+1} = x_i - \frac{e^{-x_i} - x_i}{-e^{-x_i} - 1}$$

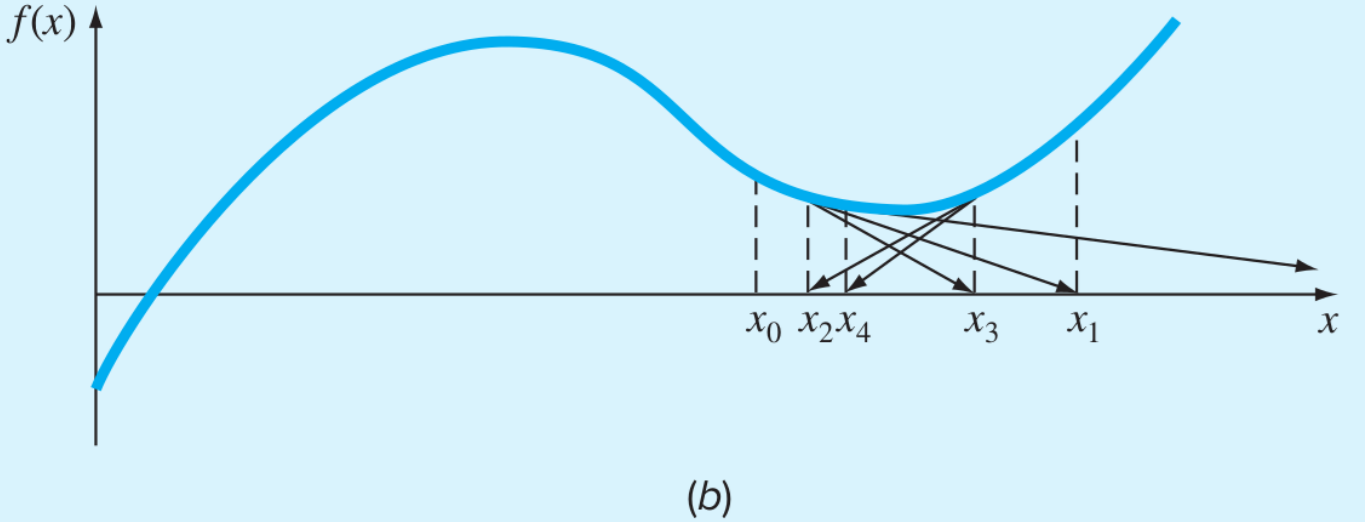
Try initial value $x_0 = 0$

i	x_i	$\lvert\varepsilon_t\rvert$, %
0	0	100
1	0.500000000	11.8
2	0.566311003	0.147
3	0.567143165	0.0000220
4	0.567143290	$<10^{-8}$

- Newton-Raphson usually converges quickly when it works.
- However sometimes the method may **diverge** or **oscillate**.
- We must put checks in our code just in case this happens. Usually we set a maximum number of iterations before breaking the loop just in case the solution is not converging.
- The next few slides show special cases where the Newton-Raphson Method fails.



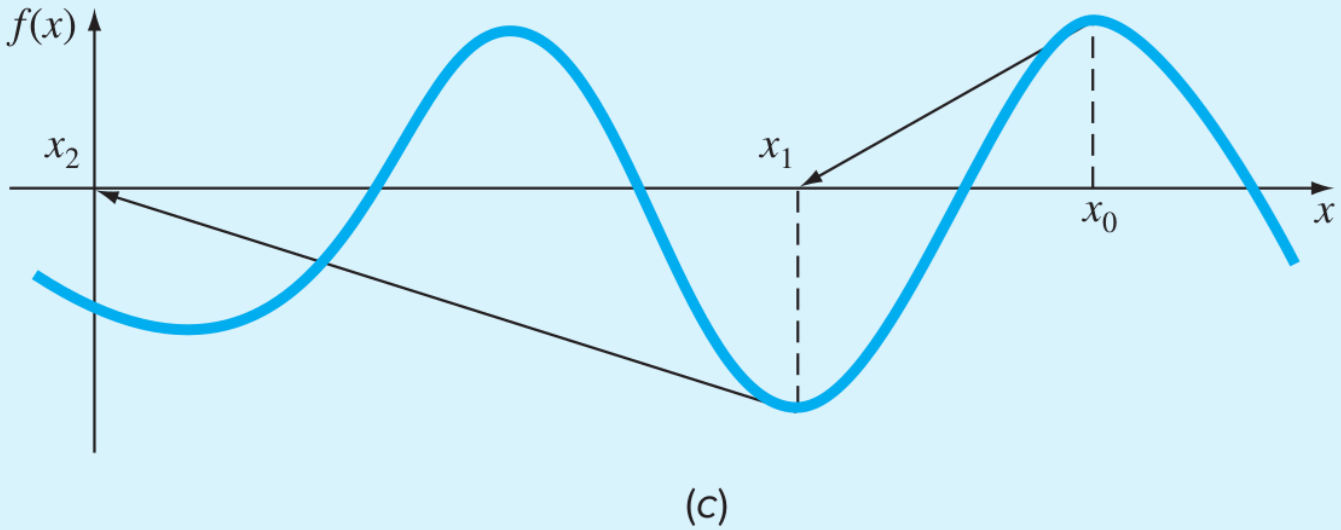
(a)



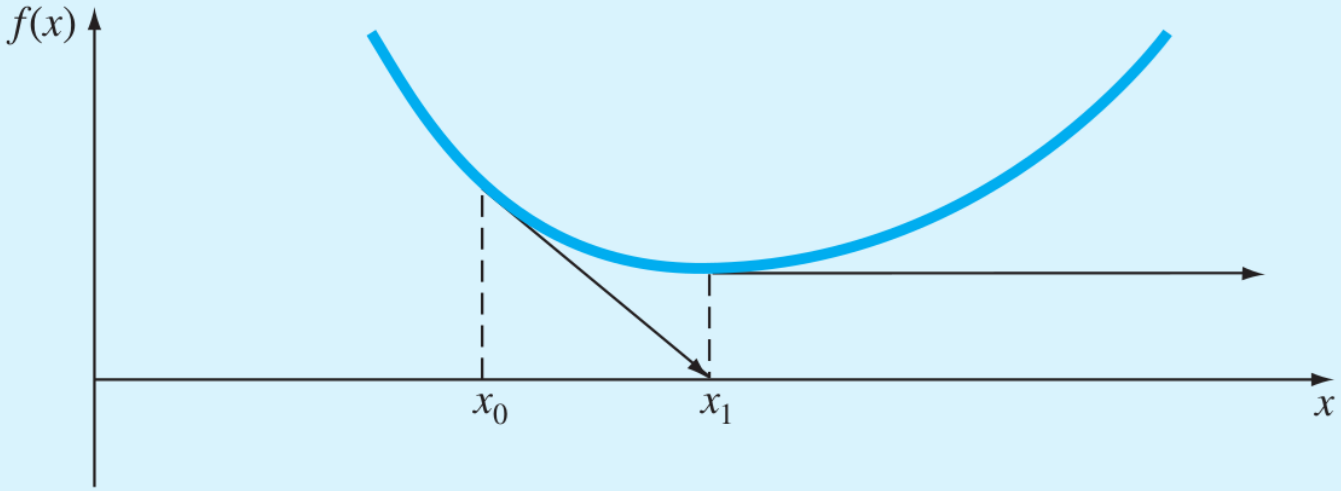
(b)

Near a point of inflection can cause divergence

Near a local minimum or maximum can cause oscillations



(c)



(d)

Root skipping can occur especially if roots are close together

A zero derivative causes division by 0 in the formula and hence divergence to infinity

5.5 Secant Method (Open)

- Sometimes the derivative can be difficult to evaluate for certain functions so the Newton-Raphson Method may not be suitable.

- Instead we can use the same formula but instead approximate the derivative with a finite difference known as the secant line (think back to calculus).

- The iterative formula becomes: $$x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)}$$

- Two initial guesses are needed, but not bracketing since the sign is not required to change.

5.6 Modified Secant Method (Open)

- Same as the Secant Method but we guess just one initial value and for the second one we choose a small distance away from the first one, Δx_i.

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

- Be careful choosing Δx_i since if it is too large we get large round-off errors but too small and the convergence is too slow.

5.7 Inverse Quadratic Interpolation (Open)

- We will cover interpolation more in detail later but for now this method uses an inverse quadratic formula and 3 initial points to generate an iterative formula.

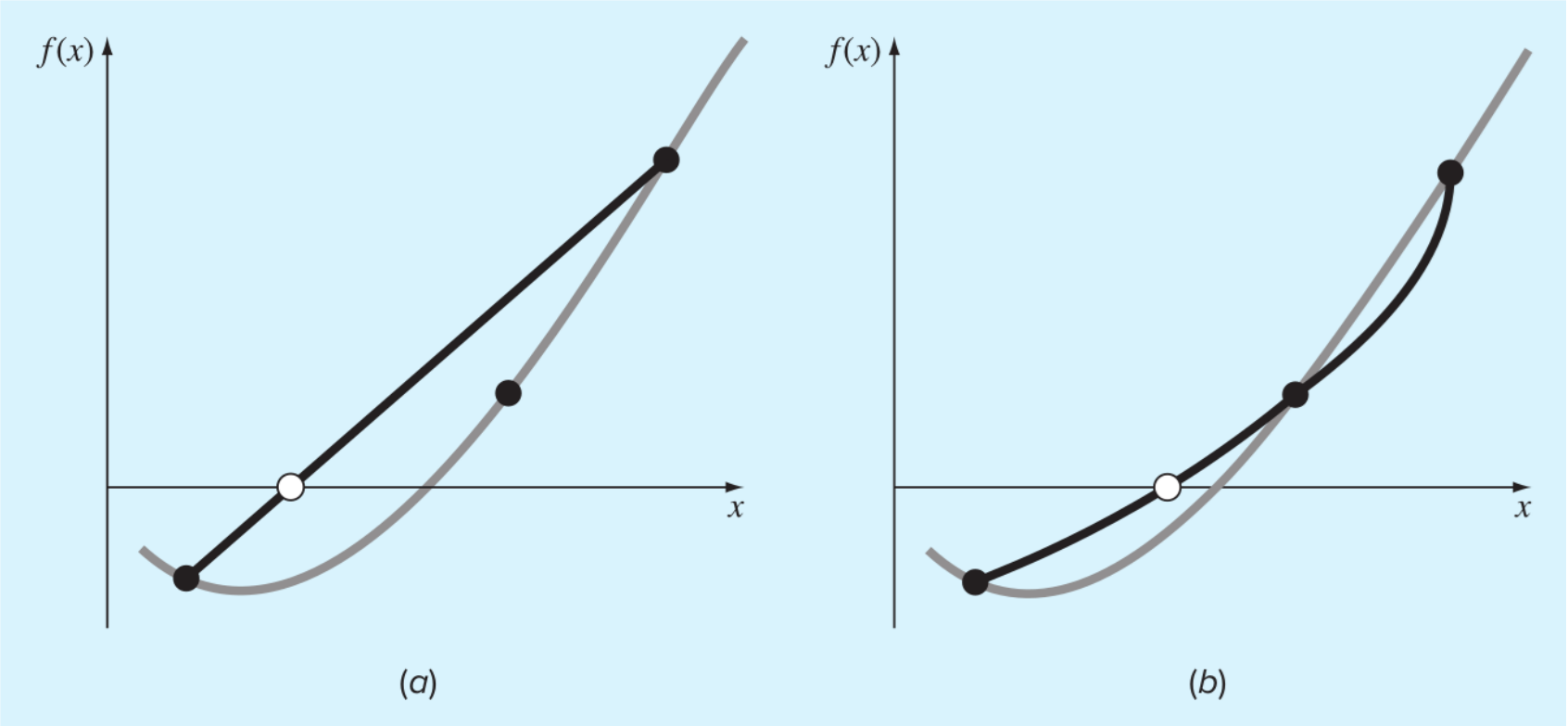
$$x_{i+1} = \frac{y_{i-1}\,y_i}{(y_{i-2}-y_{i-1})(y_{i-2}-y_i)}x_{i-2} + \frac{y_{i-2}y_i}{(y_{i-1}-y_{i-2})(y_{i-1}-y_i)}x_{i-1}$$

$$+ \frac{y_{i-2}\,y_{i-1}}{(y_i-y_{i-2})(y_i-y_{i-1})}x_i$$

- So we use the last 3 iterative points to calculate the next value of x_i.

- Fails if at any iterative step, two of the y values are the same.

- Graphs of False-Position vs. Inverse Quadratic Interpolation Method.



- Generally gives faster convergence than False-Position.

5.8 Brent's Method (Hybrid)

- Another commonly used method that is a hybrid between Bisection, Secant & Inverse Quadratic methods.

- The idea is to use the fast convergence of the Quadratic method when possible, but if it does not return a good result switch to Bisection which guarantees convergence.

- The Secant method is used if 2 points are the same at some iteration to overcome the weakness of the Quadratic method.

- The rules for determining when to switch between Quadratic and Bisection are fairly complex and will not be covered here.

5.9 MATLAB Polynomial Roots

- MATLAB has a built-in function for solving roots of polynomials.

- A polynomial, $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ is written in MATLAB as:

$$\mathrm{p} = [a_n, a_{n-1}, \ldots, a_1, a_0]$$

- We solve using the **roots** function.

- MATLAB also has the **poly** function that gives you the vector, *p*, if you tell it what the roots are. So this finds a polynomial that has the roots you give it.

EXAMPLE 3

Find the roots of $x^4 - 4x^3 - 7x^2 + 34x - 24 = 0$

```
>> p = [1 -4 -7 34 -24];
>> r = roots(p)
r =
-3
 4
 2
 1
>> r(2)
r =
     4
```

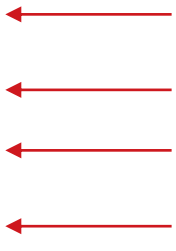
Accessing the 2nd root (for example)

Let's verify the result:

```
>>  px = @(x) x.^4 - 4*x.^3 - 7*x.^2 + 34*x - 24;
>> format shortg
>> px(r)

ans =

-3.2685e-13
     -7.1054e-14
                0
                0
```



Pretty close to machine epsilon ~ 0, due to round-off errors

Exactly 0

5.10 MATLAB General Function Roots

- For functions that are not polynomials we can use the MATLAB function **fzero**.

- It uses Brent's method to find the roots of a function given an initial guess.

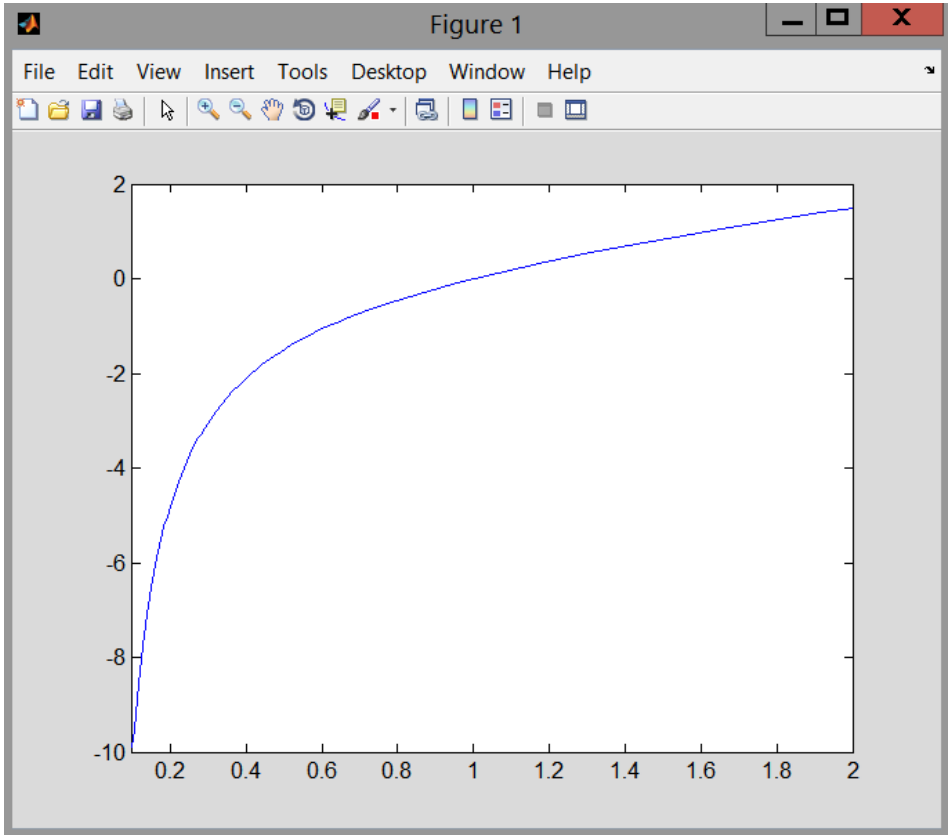
EXAMPLE 4

Find a root of $f(x) = \dfrac{x^2 - 1}{x}$

- Try plotting first to get an initial guess:

```
>> f = @(x) (x.^2 - 1)./x;
>> fplot(f,[0.1,2])
```

Plots functions on an interval



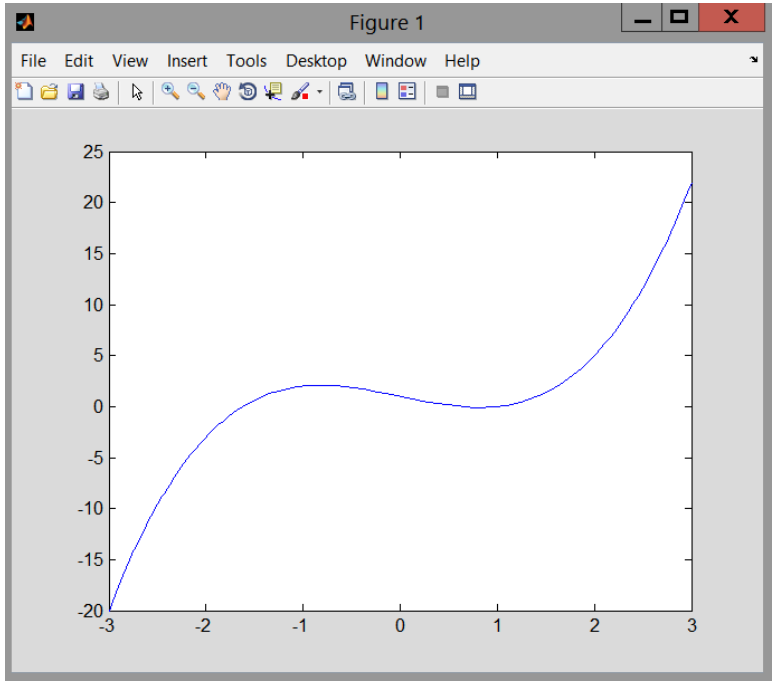
- Pick an initial value and use fzero.

```
>> r = fzero(f,2)
r =
     1
```

- We need slightly different syntax if we define the function in a file instead of as an anonymous function.

```
function y = f(x)
y = x^3 - 2*x + 1;
```

```
>> fplot(@f,[-3,3])
```




```
>> r = fzero(@f,2)
r =
     1

>> r = fzero(@f,-2)
r =
-1.6180
```

- Don’t forget to search for all the roots using different initial guesses.
- Plotting the function will help you but you must explore various intervals on the graph to see.
- You may need to adjust the axes or add a grid to see roughly where the roots are.