Chapter 6

Roots: Open Methods

Numerical Methods
Fall 2019

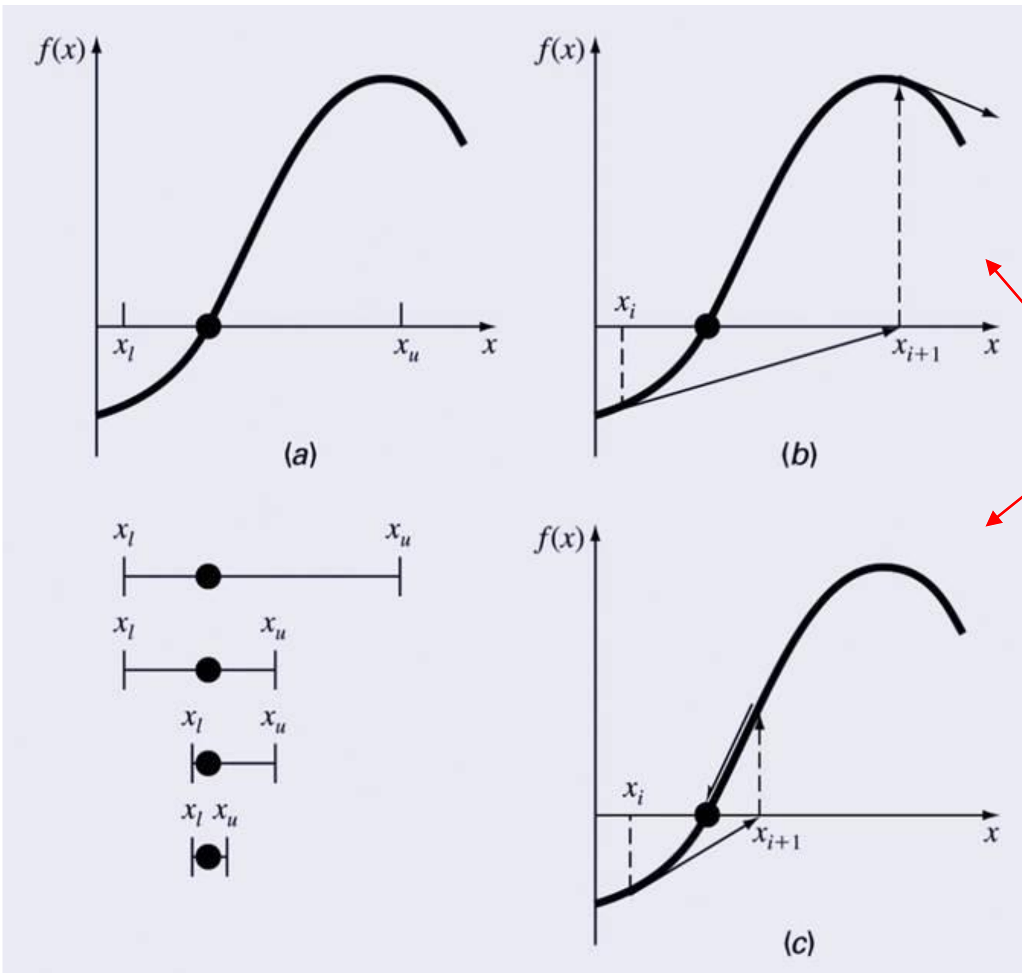
Open Methods

- *Open methods* differ from bracketing methods, in that open methods require only a single starting value or two starting values that do not necessarily bracket a root.

- Open methods may diverge as the computation progresses, but when they do converge, they usually do so much faster than bracketing methods.

Graphical Comparison of Methods

a) Bracketing method
b) Diverging open method
c) Converging open method – note speed!



fixed-point iteration
(one-point iteration)

Simple Fixed-Point Iteration

- Rearrange the function $f(x)=0$ so that x is on the left-hand side of the equation: $x=g(x)$
- Use the new function g to predict a new value of x, that is, $x_{i+1}=g(x_i)$
- The approximate error is given by:

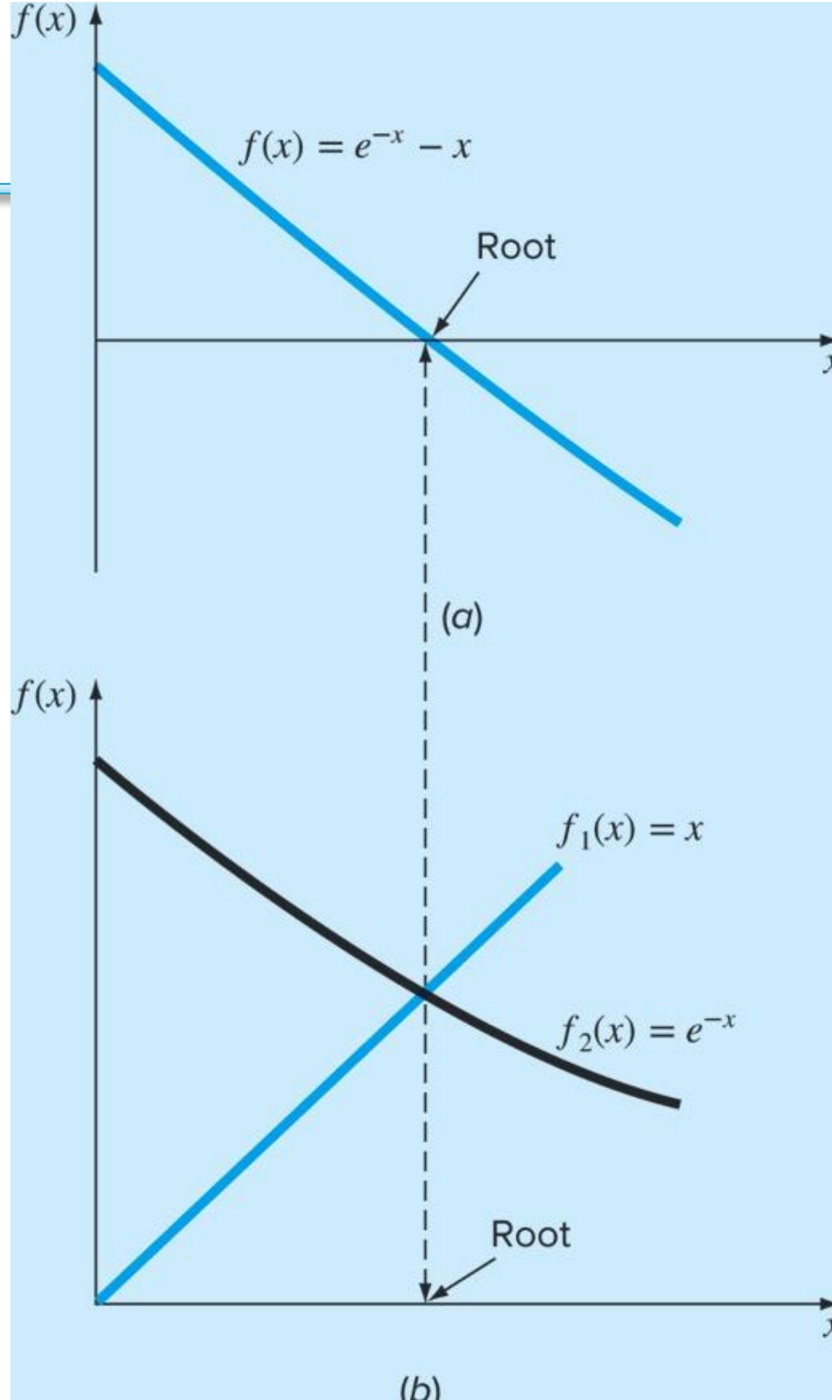
$$\varepsilon_a = \left|\frac{x_{i+1} - xi}{x_{i+1}}\right| 100\%$$

Example

- Solve $f(x)=e^{-x}-x$
- Re-write as $x = g(x)$ by isolating x (example: $x = e^{-x}$)
- Start with an initial guess (here, 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

Example

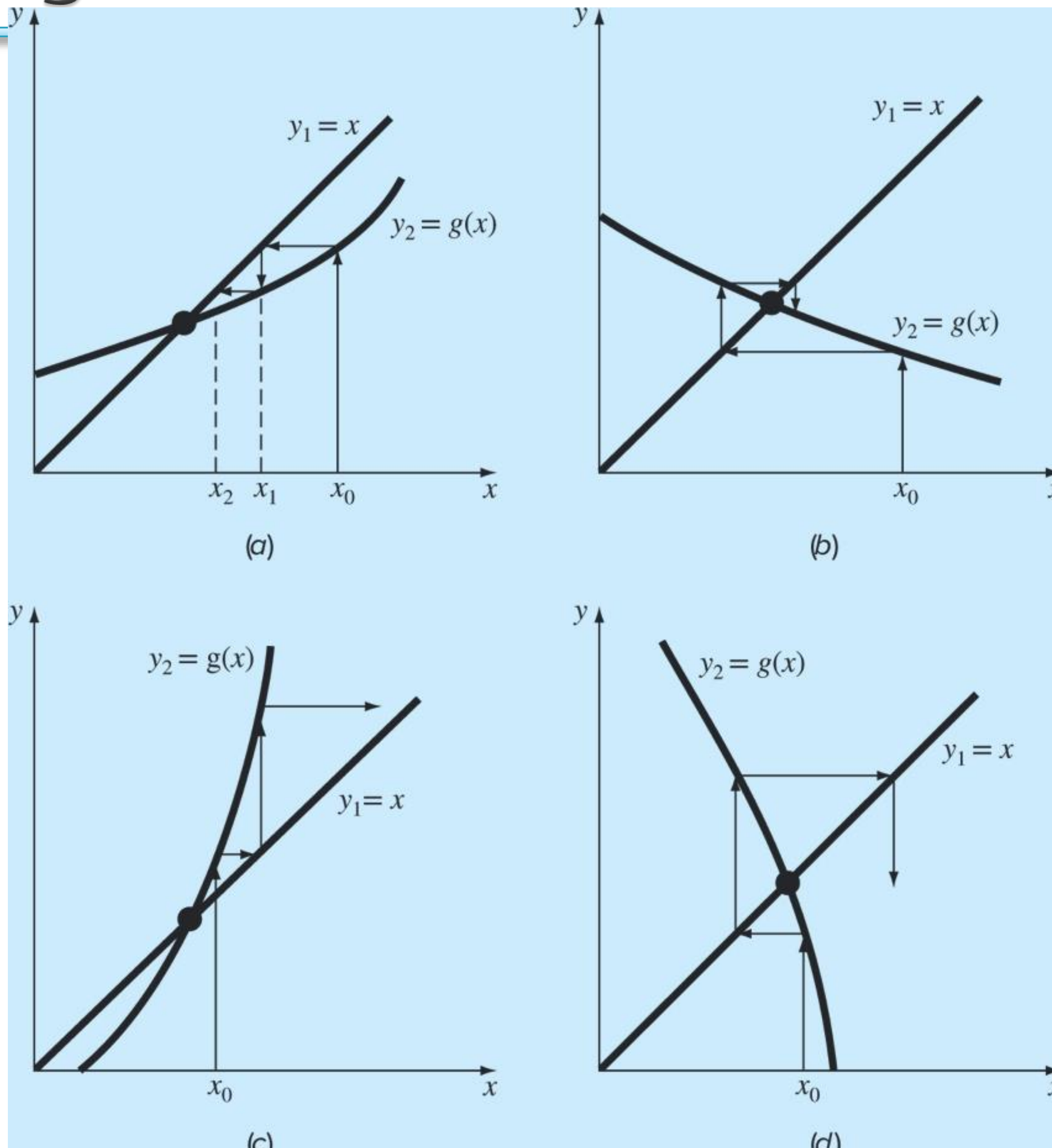


Convergence

- Convergence of the simple fixed-point iteration method requires that the derivative of $g(x)$ near the root has a magnitude less than 1.

a) Convergent, $0 \le g' < 1$
b) Convergent, $-1 < g' \le 0$
c) Divergent, $g' > 1$
d) Divergent, $g' < -1$

Convergence

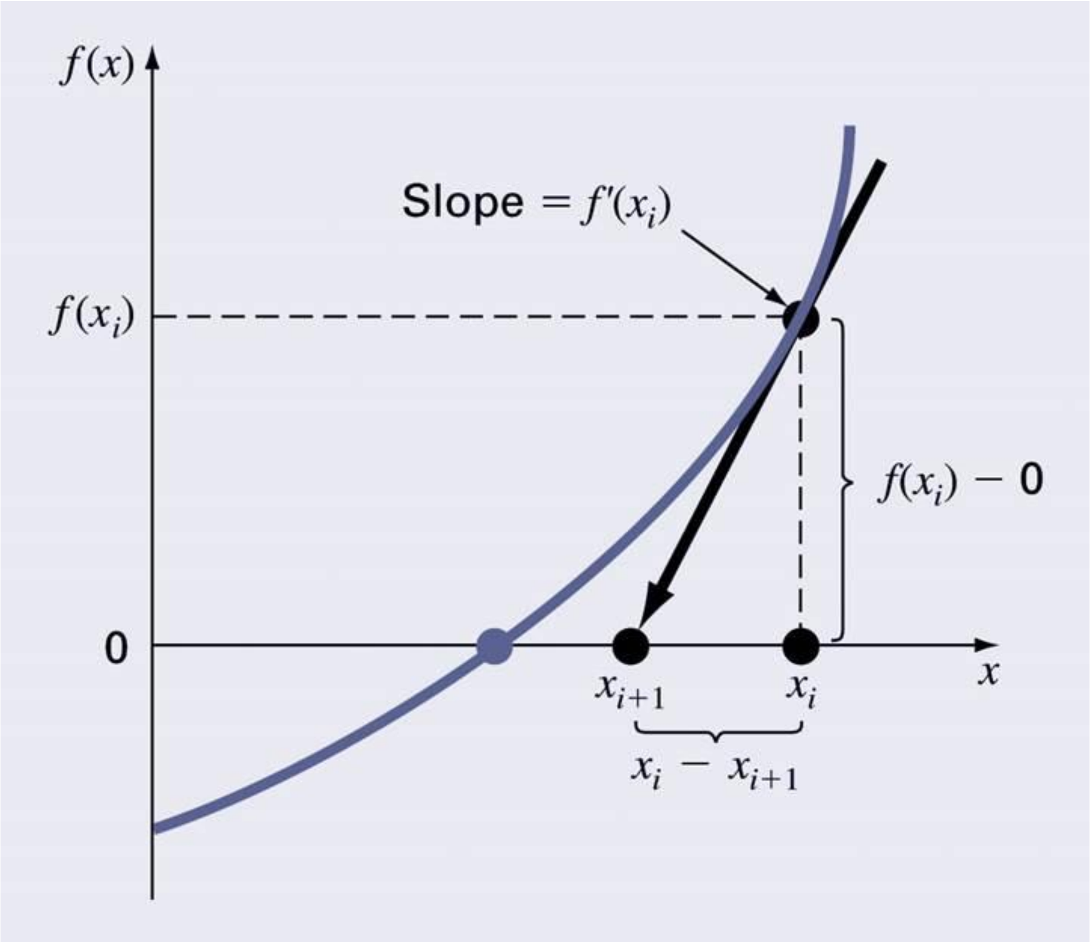


Newton–Raphson Method

- Based on forming the tangent line to the $f(x)$ curve at some guess x, then following the tangent line to where it crosses the x–axis.

$$f'(x_i) = \frac{f(x_i) - 0}{x_i - xi_{+1}}$$

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



Newton–Raphson Method

- Example

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

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

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

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	$<10^{-8}$

Pros and Cons

- Pro: The error of the *i*+1th iteration is roughly proportional to the square of the error of the *i*th iteration – this is called *quadratic convergence*

$$E_{t,i+1} = \frac{-f''(x_r)}{2f'(x_r)} E_{t,i}^2$$

- Con: Some functions show slow or poor convergence or divergence!

Pros and Cons

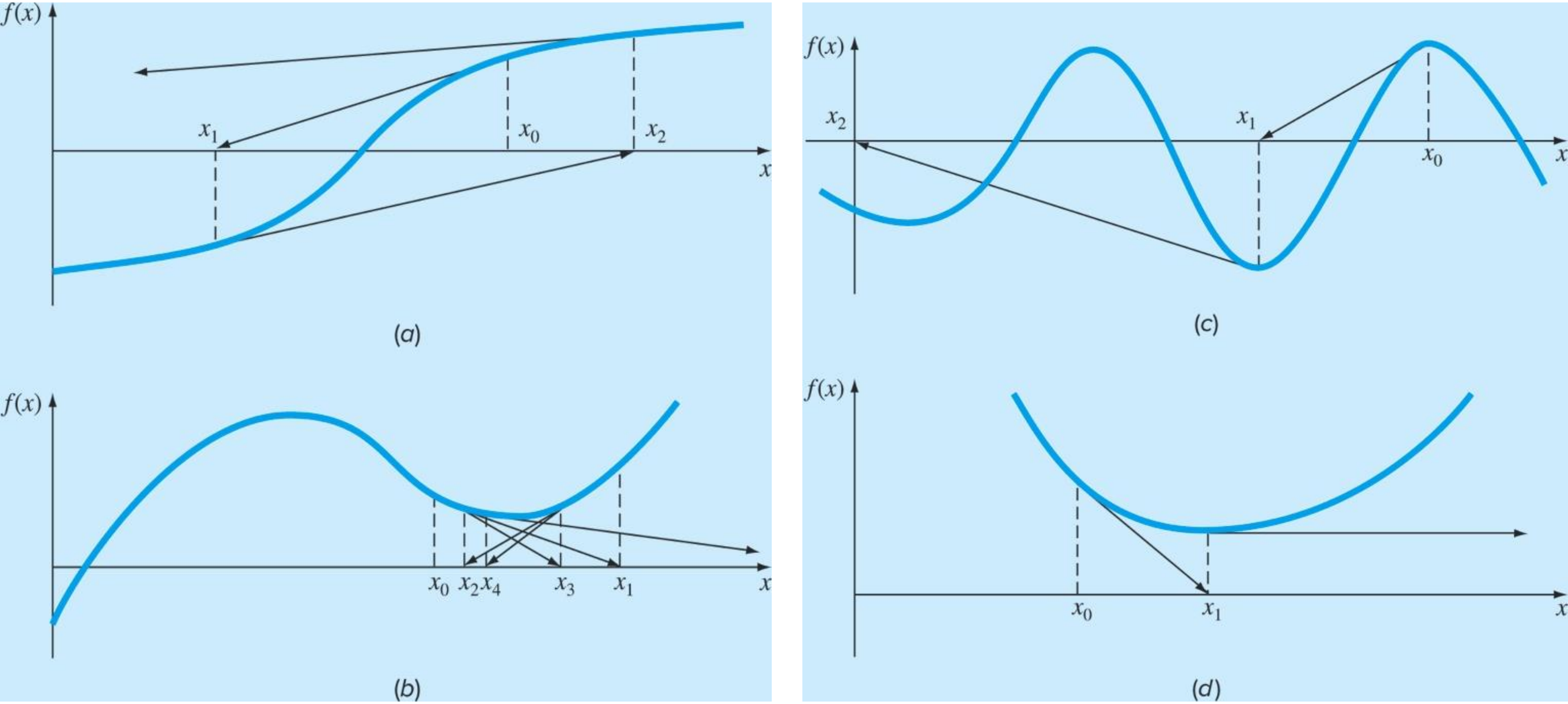
- Example

$$f(x) = x^{10} - 1$$

$$x_{i+1} = x_i - \frac{x_i^{10} - 1}{10x_i^9}$$

i	x_i	$\lvert\varepsilon_a\rvert$, %
0	0.5	
1	51.65	99.032
2	46.485	11.111
3	41.8365	11.111
4	37.65285	11.111
⋮		
40	1.002316	2.130
41	1.000024	0.229
42	1	0.002

Pros and Cons



Secant Methods (1)

- A potential problem in implementing the Newton–Raphson method is the evaluation of the derivative – there are certain functions whose derivatives may be difficult or inconvenient to evaluate.
- For these cases, the derivative can be approximated by a backward finite divided difference:

$$f'(xi) \cong \frac{f(x_{i-1}) - f(xi)}{x_{i-1} - xi}$$

Secant Methods (2)

- Substitution of this approximation for the derivative to the Newton–Raphson method equation gives:

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

- Note – this method requires *two* initial estimates of x but does *not* require an analytical expression of the derivative.

Modified Secant method

- Rather than using two arbitrary values to estimate the derivative, an alternative approach involves a fractional perturbation of the independent variable to estimate f'(x)

$$f'(x_i) \cong \frac{f(x_i + \delta x_i) - f(x_i)}{\delta x_i}$$

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

Modified Secant method

- Example: Use the modified secant method to determine the mass of the bungee jumper with a drag coefficient of 0.25 kg/m to have a velocity of 36 m/s after 4 s of free fall. Note: The acceleration of gravity is 9.81 m/s^2. Use an initial guess of 50 kg and a value of 10^{-6} for the perturbation fraction.

Modified Secant method

- Solution:

First iteration:

$$x_0 = 50 \qquad f(x_0) = -4.57938708$$

$$x_0 + \delta x_0 = 50.00005 \qquad f(x_0 + \delta x_0) = -4.579381118$$

$$x_1 = 50 - \frac{10^{-6}(50)(-4.57938708)}{-4.579381118 - (-4.57938708)}$$

$$= 88.39931(|\varepsilon_t| = 38.1\%;\ |\varepsilon_a| = 43.4\%)$$

Second iteration:

$$x_1 = 88.39931 \qquad f(x_1) = -1.69220771$$

$$x_1 + \delta x_1 = 88.39940 \qquad f(x_1 + \delta x_1) = -1.692203516$$

$$x_2 = 88.39931 - \frac{10^{-6}(88.39931)(-1.69220771)}{-1.692203516 - (-1.69220771)}$$

$$= 124.08970(|\varepsilon_t| = 13.1\%;\ |\varepsilon_a| = 28.76\%)$$

Modified Secant method

i	x_i	$\lvert\varepsilon_t\rvert$, %	$\lvert\varepsilon_a\rvert$, %
0	50.0000	64.971	
1	88.3993	38.069	43.438
2	124.0897	13.064	28.762
3	140.5417	1.538	11.706
4	142.7072	0.021	1.517
5	142.7376	4.1×10^{-6}	0.021
6	142.7376	3.4×10^{-12}	4.1×10^{-6}

MATLAB's `fzero` Function

- MATLAB's `fzero` provides the best qualities of both bracketing methods and open methods.
  - Using an initial guess:

    `x = fzero(`*`function`*`, `*`x0`*`)`

    `[x, fx] = fzero(`*`function`*`, `*`x0`*`)`
    - *`function`* is a function handle to the function being evaluated
    - *`x0`* is the initial guess
    - *`x`* is the location of the root
    - *`fx`* is the function evaluated at that root
  - Using an initial bracket:

    `x = fzero(`*`function`*`, [`*`x0 x1`*`])`

    `[x, fx] = fzero(`*`function`*`, [`*`x0 x1`*`])`
    - As above, except *`x0`* and *`x1`* are guesses that *must* bracket a sign change

`fzero` Options

- Options may be passed to fzero as a third input argument – the options are a data structure created by the `optimset` command

- *options* = optimset(‘par_1’, val_1, ‘par_2’, val_2,…)
  - par_n *is the name of the parameter to be set*
  - val_n is the value to which to set that parameter

  - The parameters commonly used with `fzero` are:
    - `display`: when set to ‘iter’ displays a detailed record of all the iterations
    - `tolx`: A positive scalar that sets a termination tolerance on x.

`fzero` Example

- `options = optimset('display', 'iter');`
  - Sets options to display each iteration of root finding process
- `[x, fx] = fzero(@(x) x^10-1, 0.5, options)`
  - Uses fzero to find roots of $f(x)=x^{10}-1$ starting with an initial guess of $x=0.5$.
- MATLAB reports `x=1, fx=0` after 35 function counts

Polynomials (1)

- Polynomials are a special type of nonlinear algebraic equation of the general form

$$\boxed{f_n(x) = a_1 x^n + a_2 x^{n-1} + \cdots + a_{n-1} x^2 + a_n x + a_{n+1}}$$

- MATLAB has a built in program called `roots` to determine all the roots of a polynomial – including imaginary and complex ones.
- `x = roots(c)`
  - `x` is a column vector containing the roots
  - `c` is a row vector containing the polynomial coefficients
- Example:
  - Find the roots of

    $f(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$

```
x = roots([1 -3.5 2.75 2.125 -3.875 1.25])
```

Polynomials (2)

- MATLAB's `poly` function can be used to determine polynomial coefficients if roots are given:
  - `b = poly([0.5 -1])`
    - Finds $f(x)$ where $f(x) = 0$ for $x = 0.5$ and $x = -1$
    - MATLAB reports `b = [1.000 0.5000 -0.5000]`
    - This corresponds to $f(x) = x^2 + 0.5x - 0.5$
- MATLAB's `polyval` function can evaluate a polynomial at one or more points:

```
>> a = [1 -3.5 2.75 2.125 -3.875 1.25];
```

    - If used as coefficients of a polynomial, this corresponds to $f(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$
  - `polyval(a, 1)`
    - This calculates $f(1)$, which MATLAB reports as −0.2500

Homework #2

- 연습문제 풀이 (교재 3판)
  - 5장 연습문제 5.5, 5.7번. 각 문제의 (a) (b) (c) 를 계산기만으로 풀기
  - 6장 연습문제 6.2 (a) (b), 6.3 (a) (b) (c) (d) 계산기만으로 풀기
  - 6.3(e)는 MATLAB code 작성으로 원하는 사람만.

- 주의: 답지에는 풀이과정을 모두 적어야 함
- 제출일: 10월 2일 수업시간