Fixed-Point Iteration

Prof. Seungchul Lee
Industrial AI Lab.

Numerical Approach

- For the given equation

$$f(x) = 0 \implies x = g(x)$$

$$x = f(x) + x = g(x)$$
$$x = -f(x) + x = g(x)$$

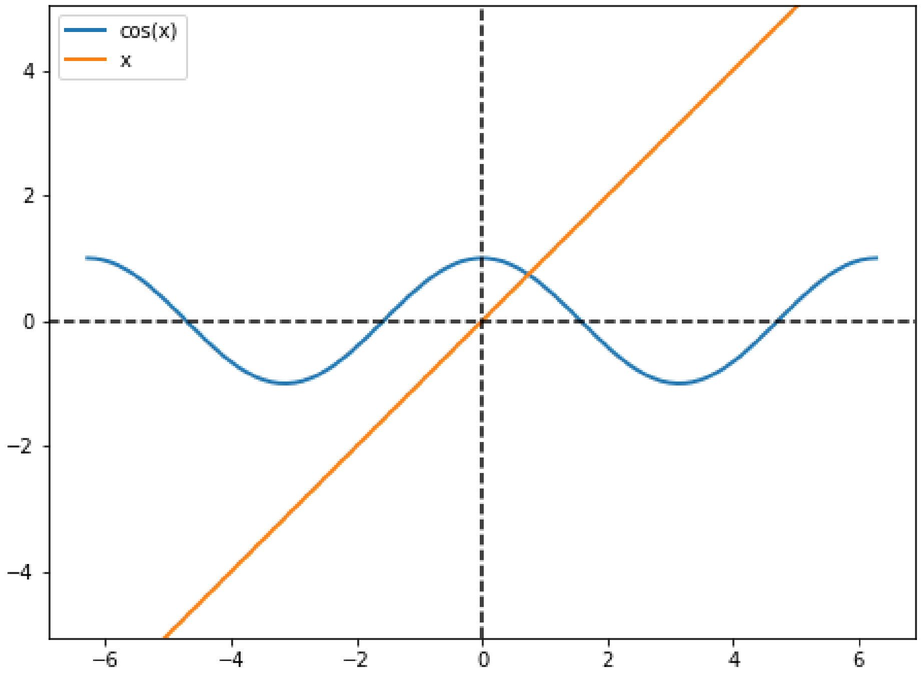
- Goal: numerically find the solution of $x = g(x)$

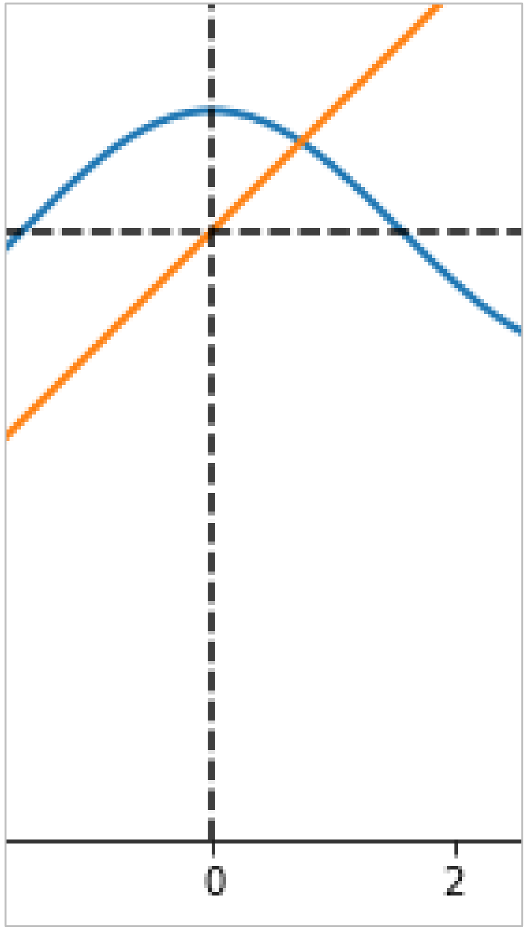
- Main idea:
  - Make a guess of the solution, x_k
  - If $g(x_k)$ is 'nice', then hopefully, $g(x_k)$ will be closer to the answer. If so, we can iterate

Iteration Algorithm

- Goal: numerically find the solution of $x = g(x)$
  1) Choose an initial point x_0
  2) Do the iteration $x_{k+1} = g(x_k)$ until meeting stopping criteria

- Example of $x = \cos x$





Naïve Approach

- $x = \cos x$

```python
# naive approach

x = 0.3
print (np.cos(x))
print (np.cos(np.cos(x)))
print (np.cos(np.cos(np.cos(x))))
print (np.cos(np.cos(np.cos(np.cos(x)))))
print (np.cos(np.cos(np.cos(np.cos(np.cos(x))))))
print (np.cos(np.cos(np.cos(np.cos(np.cos(np.cos(x)))))))
print (np.cos(np.cos(np.cos(np.cos(np.cos(np.cos(np.cos(x))))))))
```

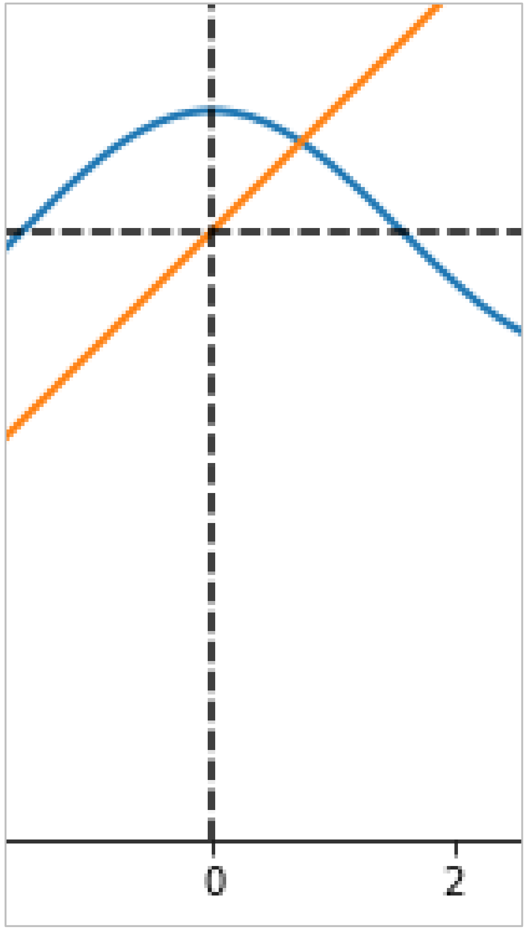
```
0.955336489125606
0.5773340444711864
0.8379206831271269
0.6690097308223832
0.7844362247423562
0.7077866472756374
0.7598027552852303
```

```python
# better way

x = 10
for i in range(24):
    x = np.cos(x)

print (x)
```

```
0.7390735444682907
```

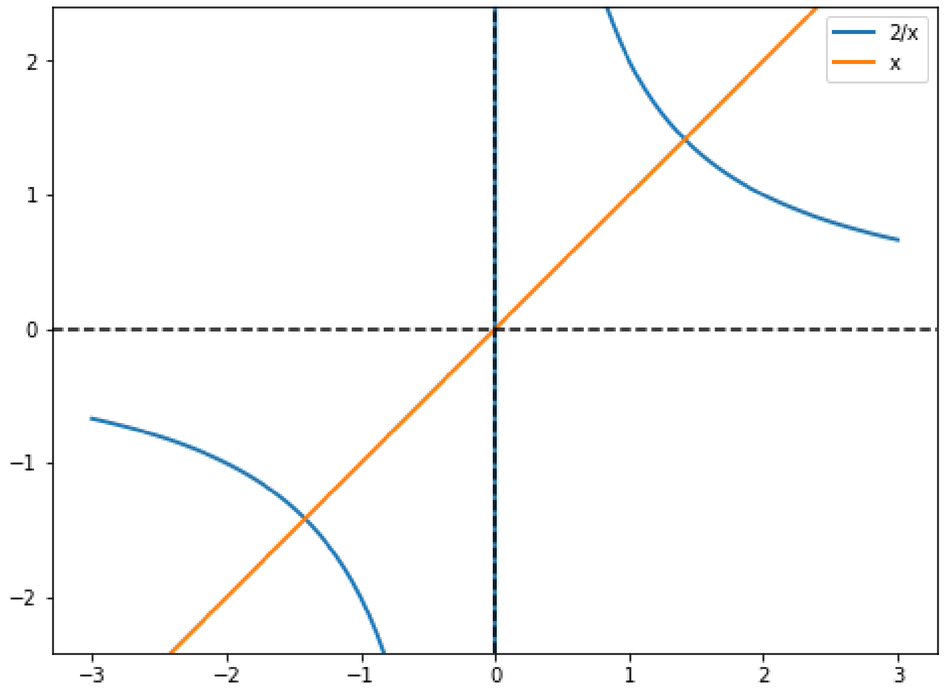


Example

- $x^2 = 2, x = \pm\sqrt{2}$

```
x = 2
for i in range(10):
    x = 2/x
    print (x)
```

```
1.0
2.0
1.0
2.0
1.0
2.0
1.0
2.0
1.0
2.0
```



Convergence Check (or Analysis)

- Let r be the exact solution, $r = g(r)$

$$
\begin{aligned}
x_{k+1} &= g(x_k) \\
\text{error:} \quad e_k &= x_k - r \\
e_{k+1} &= x_{k+1} - r = g(x_k) - g(r) \\
&= g'(\eta)(x_k - r) = g'(\eta)e_k, \quad \eta \in (x_k, r)
\end{aligned}
$$

$$
\Longrightarrow \quad |e_{k+1}| \leq |g'(\eta)||e_k|
$$

If $|g'(\eta)| < 1$, error decreases (iteration converges)

- $x^2 = 2$ or $x = \frac{2}{x}$

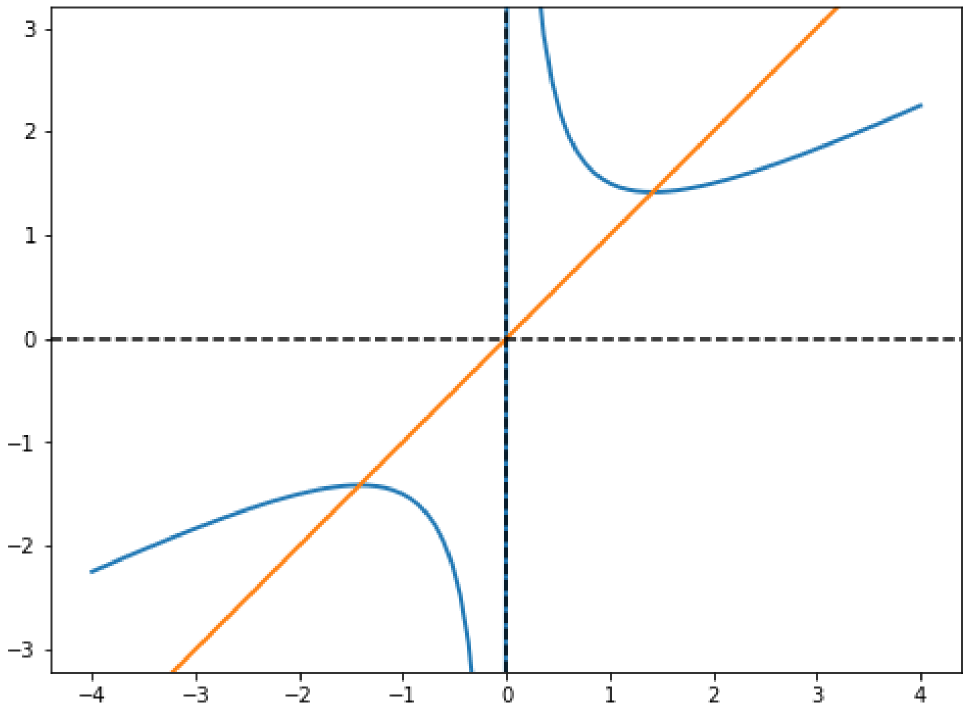
Example

- $x = \frac{2}{x}$
- $x = \frac{1}{2}\left(x + \frac{2}{x}\right)$, kind of damping

```python
# How to overcome
# Use an idea of a fixed point +   kind of *|damping|*

x = 3
for i in range(10):
    x = (x + 2/x)/2
    print (x)
```

```
1.8333333333333333
1.4621212121212122
1.414998429894803
1.4142137800471977
1.4142135623731118
1.414213562373095
1.414213562373095
1.414213562373095
1.414213562373095
1.414213562373095
```



System of Linear Equations

$$\begin{aligned} 4x_1 - x_2 + x_3 &= 7 \\ 4x_1 - 8x_2 + x_3 &= -21 \\ -2x_1 + x_2 + 5x_3 &= 15 \end{aligned}$$

```python
# matrix inverse

A = np.array([[4, -1, 1], [4, -8, 1], [-2, 1, 5]])
b = np.array([[7, -21, 15]]).T

x = np.linalg.inv(A).dot(b)

print (x)
```

```
[[2.]
 [4.]
 [3.]]
```

- This solution only possible for small size problems.
- There are many iterative methods for large problems.

Iterative Methods for System of Linear Equations

$$\begin{aligned} 4x_1 - x_2 + x_3 &= 7 \\ 4x_1 - 8x_2 + x_3 &= -21 \\ -2x_1 + x_2 + 5x_3 &= 15 \end{aligned} \quad \Longrightarrow \quad \begin{aligned} x_1 &= \frac{1}{4}x_2 - \frac{1}{4}x_3 + \frac{7}{4} \\ x_2 &= \frac{1}{2}x_1 + \frac{1}{8}x_3 + \frac{21}{8} \\ x_3 &= \frac{2}{5}x_1 - \frac{1}{5}x_2 + \frac{15}{5} \end{aligned}$$

- In a matrix form

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{2} & 0 & \frac{1}{8} \\ \frac{2}{5} & -\frac{1}{5} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} \frac{7}{4} \\ \frac{21}{8} \\ 3 \end{bmatrix}$$

- Iteration

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{2} & 0 & \frac{1}{8} \\ \frac{2}{5} & -\frac{1}{5} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{bmatrix} + \begin{bmatrix} \frac{7}{4} \\ \frac{21}{8} \\ 3 \end{bmatrix}$$

Iterative Methods for System of Linear Equations

```python
# Iterative way

A = np.array(([[0, 1/4, -1/4 ],
               [4/8, 0, 1/8],
               [2/5, -1/5, 0]]))
b = np.array([[7/4, 21/8, 15/5]]).T

# initial point
x = np.array([[1, 1, 2]]).T

A = np.asmatrix(A)
b = np.asmatrix(b)
x = np.asmatrix(x)

for i in range(20):
    x = A*x + b

print (x)
```

```
[[2.]
 [4.]
 [3.]]
```

$$\begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ x_3^{(k+1)} \end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{4} & -\frac{1}{4} \\ \frac{1}{2} & 0 & \frac{1}{8} \\ \frac{2}{5} & -\frac{1}{5} & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ x_3^{(k)} \end{bmatrix} + \begin{bmatrix} \frac{7}{4} \\ \frac{21}{8} \\ 3 \end{bmatrix}$$

Try This One

$$4x_1 - x_2 + x_3 = 7$$
$$4x_1 - 8x_2 + x_3 = -21$$
$$-2x_1 + x_2 + 5x_3 = 15$$

$$\Longrightarrow \quad \begin{aligned} x_1 &= -3x_1 + x_2 - x_3 + 7 \\ x_2 &= 4x_1 - 7x_2 + x_3 + 21 \\ x_3 &= 2x_1 - x_2 + -4x_3 + 15 \end{aligned}$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \leftarrow \begin{bmatrix} -3 & 1 & -1 \\ 4 & -7 & 1 \\ 2 & -1 & -4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 7 \\ 21 \\ 15 \end{bmatrix}$$

```python
# think about why this one does not work

A = np.array(([[3, 1, -1 ],
               [4, 7, 1],
               [2, -1, -4]]))
b = np.array([[7, 21, 15]]).T

# initial point
x = np.array([[1, 2, 2]]).T

for i in range(10):
    x = A.dot(x) + b

print (x)
```

```
[[1076845340]
 [ 660465147]
 [-237408283]]
```

?

- Convergence check

$$x \leftarrow Ax + b$$

$$\begin{aligned} x_{k+1} &= Ax_k + b \\ &= A(Ax_{k-1} + b) + b = A^2 X_{k-1} + Ab + b \\ &\vdots \\ &= A^{k+1}x_0 + A^k b + \cdots + Ab + b \end{aligned}$$

Convergence Check

- Eigenvalue of A

```
# stability, check eigenvalue of A

A = np.array(([[0, 1/4, -1/4 ],
               [4/8, 0, 1/8],
               [2/5, -1/5, 0]]))

np.linalg.eig(A)
```

```
(array([ 0.33471648+0.j        , -0.16735824+0.28987297j,
        -0.16735824-0.28987297j]),
 array([[-0.52873647+0.j        , -0.10993842+0.35839967j,
         -0.10993842-0.35839967j],
        [-0.83865529+0.j        ,  0.40608435-0.36739725j,
          0.40608435+0.36739725j],
        [-0.13074806+0.j        ,  0.74804945+0.j        ,
          0.74804945-0.j        ]]))
```

```
# stability, check eigenvalue of A

A = np.array(([[3, 1, -1 ],
               [4, 7, 1],
               [2, -1, -4]]))

np.linalg.eig(A)
```

```
(array([ 7.82147369,  1.65496639, -3.47644008]),
 array([[-0.21213573, -0.71387137,  0.17464301],
        [-0.97612458,  0.60136269, -0.15942549],
        [ 0.04668226, -0.3588183 ,  0.97163951]]))
```