Thomas algorithm 설명회

Team. Two-rillion

금융공학과 박수빈
금융공학과 서보규
금융공학과 선두연
금융공학과 전민욱
e-비즈니스학과 심푸름

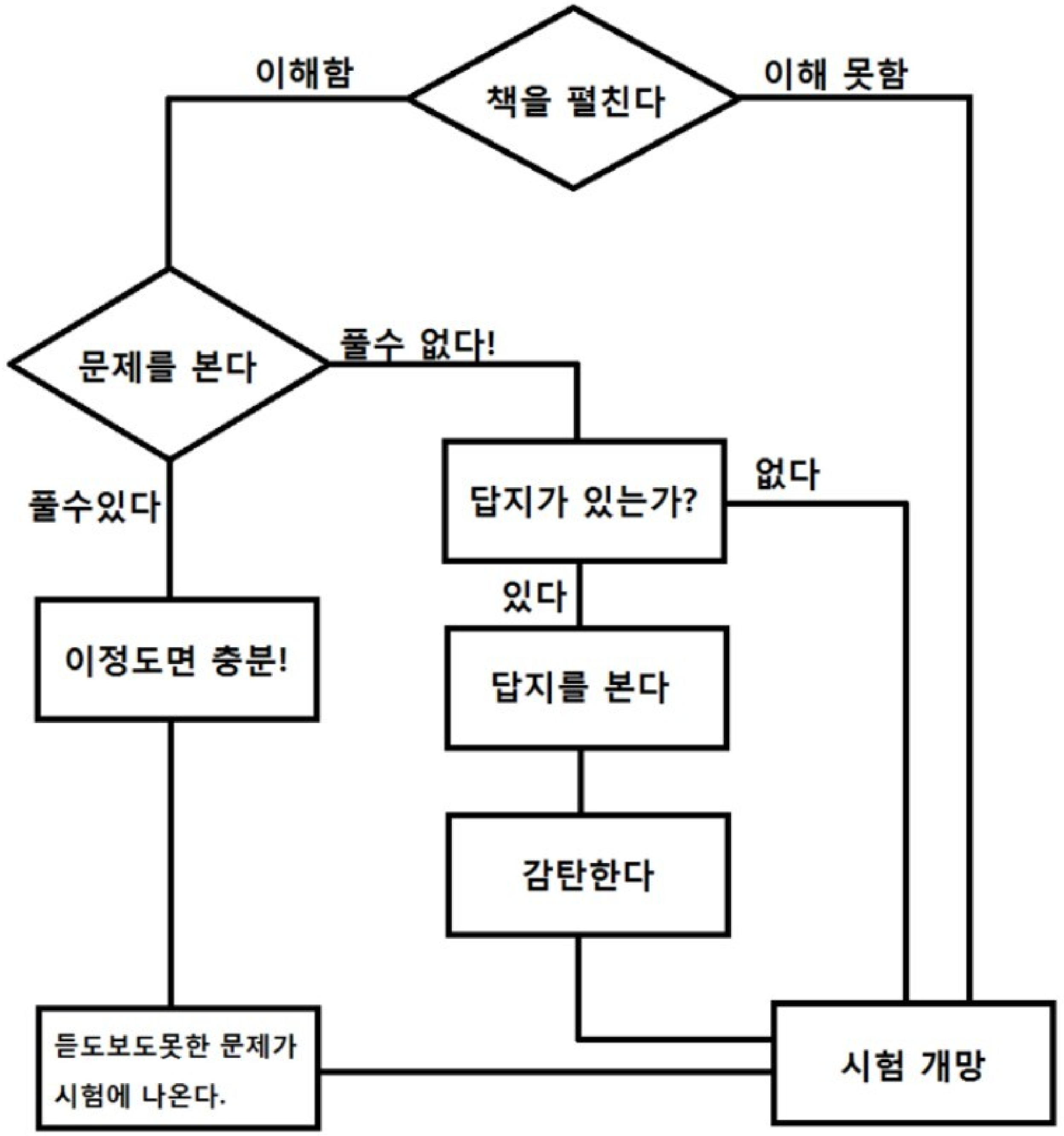
23-2 가을학기 이산수학

목차

01

알고리즘과 시간복잡도

1 알고리즘 (algorithm)



주어진 문제를 **논리적**으로 **해결**하기 위해

필요한 **절차, 방법, 명령어**들을 모아 놓은 것

1 시간복잡도(Time Complexity)

컴퓨터 프로그램의 **입력값**과

연산 수행 시간의 **상관관계**를 나타내는 척도

O	Ω	Θ
Big Oh	Omega	Theta
Upper Bound	Lower Bound	Upper & Lower Bound
Worst Case	Best Case	Average 'ish'

Big-O를 쓰는 이유?

프로그래밍 = **최악**의 경우를 고려

02

삼중대각행렬이란

2 삼중대각행렬

$$A = \begin{pmatrix} a_{11} & a_{12} & 0 & 0 & 0 & 0 & 0 \\ a_{21} & a_{22} & a_{23} & 0 & 0 & 0 & 0 \\ 0 & a_{32} & a_{33} & a_{34} & 0 & 0 & 0 \\ 0 & 0 & a_{43} & a_{44} & a_{45} & 0 & 0 \\ 0 & 0 & 0 & a_{54} & a_{55} & a_{56} & 0 \\ 0 & 0 & 0 & 0 & a_{65} & a_{66} & a_{67} \\ 0 & 0 & 0 & 0 & 0 & a_{76} & a_{77} \end{pmatrix}$$

계수 행렬의 **대각선** 요소와

왼쪽 및 오른쪽에 이웃한 요소 외에는 **전부 0**인 행렬

03

삼중대각행렬의 풀이

3 역행렬 풀이

2x2 행렬

$$A^{-1} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1}$$

$$= \frac{1}{ad-bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

3x3 행렬

$$A^{-1} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^{-1}$$

$$= \frac{1}{|A|} \begin{bmatrix} ei-fh & -(bi-ch) & bf-ce \\ -(di-fg) & ai-cg & -(af-cd) \\ dh-eg & -(ah-bg) & ae-bd \end{bmatrix}$$

$$= \frac{1}{|A|} \begin{bmatrix} ei-fh & ch-bi & bf-ce \\ fg-di & ai-cg & cd-af \\ dh-eg & bg-ah & ae-bd \end{bmatrix}$$

행렬이 길어질수록 풀이가 점점 더 **복잡**해짐

3 가우스 소거

$$\left[\begin{array}{rrr|r} 1 & 3 & 1 & 9 \\ 1 & 1 & -1 & 1 \\ 3 & 11 & 5 & 35 \end{array}\right] \to \left[\begin{array}{rrr|r} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 2 & 2 & 8 \end{array}\right] \to \left[\begin{array}{rrr|r} 1 & 3 & 1 & 9 \\ 0 & -2 & -2 & -8 \\ 0 & 0 & 0 & 0 \end{array}\right] \to \left[\begin{array}{rrr|r} 1 & 0 & -2 & -3 \\ 0 & 1 & 1 & 4 \\ 0 & 0 & 0 & 0 \end{array}\right]$$

$$M = \begin{pmatrix} -1 & 1 & 2 \\ 3 & -1 & 1 \\ -1 & 3 & 4 \end{pmatrix}$$

3 가우스 소거

$$
(\,A \mid I\,) \to \left(\begin{array}{ccc|ccc} -1 & 1 & 2 & 1 & 0 & 0 \\ 3 & -1 & 1 & 0 & 1 & 0 \\ -1 & 3 & 4 & 0 & 0 & 1 \end{array}\right) \to \left(\begin{array}{ccc|ccc} 1 & -1 & -2 & -1 & 0 & 0 \\ 0 & 1 & 3.5 & 1.5 & 0.5 & 0 \\ 0 & 0 & 1 & 0.8 & 0.2 & -0.2 \end{array}\right)
$$

$$
\to \left(\begin{array}{ccc|ccc} -1 & 1 & 2 & 1 & 0 & 0 \\ 0 & 2 & 7 & 3 & 1 & 0 \\ 0 & 2 & 2 & -1 & 0 & 1 \end{array}\right) \to \left(\begin{array}{ccc|ccc} 1 & -1 & 0 & 0.6 & 0.4 & -0.4 \\ 0 & 1 & 0 & -1.3 & -0.2 & 0.7 \\ 0 & 0 & 1 & 0.8 & 0.2 & -0.2 \end{array}\right)
$$

$$
\to \left(\begin{array}{ccc|ccc} -1 & 1 & 2 & 1 & 0 & 0 \\ 0 & 2 & 7 & 3 & 1 & 0 \\ 0 & 0 & -5 & -4 & -1 & 1 \end{array}\right) \to \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & -0.7 & 0.2 & 0.3 \\ 0 & 1 & 0 & -1.3 & -0.2 & 0.7 \\ 0 & 0 & 1 & 0.8 & 0.2 & -0.2 \end{array}\right)
$$

3 Thomas Algorithm 풀이

$$\begin{bmatrix} b_1 & c_1 & 0 \\ a_2 & b_2 & c_2 \\ 0 & a_3 & b_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

$$b_1 x_1 + c_1 x_2 = d_1$$

$\rightarrow$ *divided by* b_1

$$x_1 + \frac{c_1}{b_1} x_2 = \frac{d_1}{b_1}$$

$$x_1 + c'_1 x_2 = d'_1 \quad , \; c'_1 = \frac{c_1}{b_1} \quad , \; d'_1 = \frac{d_1}{b_1}$$

3 Thomas Algorithm 경우

3x3 행렬

$$\begin{bmatrix} 1 & c'_1 & 0 \\ a_2 & b_2 & c_2 \\ 0 & a_3 & b_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} d'_1 \\ d_2 \\ d_3 \end{bmatrix}$$

$$\begin{aligned} a_2x_1+b_2x_2+c_2x_3 &= d_2 \\ -\ a_2x_1+a_2c'_1x_2 &= a_2d'_1 \end{aligned}$$

$$\Rightarrow (b_2-a_2c'_1)x_2+c_2x_3 = d_2-a_2d'_1$$

$$\Rightarrow x_2+\frac{c_2}{b_2-a_2c'_1}x_3 = \frac{d_2-a_2d'_1}{b_2-a_2c'_1}$$

$$\Rightarrow x_2+c'_2x_3 = d'_2 \quad , c'_2=\frac{c_2}{b_2-a_2c'_1} \quad , d'_2=\frac{d_2-a_2d'_1}{b_2-a_2c'_1}$$

3 Thomas Algorithm 경우

3x3 행렬

$$\begin{bmatrix} 1 & c'_1 & 0 \\ 0 & 1 & c'_2 \\ 0 & a_3 & b_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} d'_1 \\ d'_2 \\ d_3 \end{bmatrix}$$

$$\begin{array}{rll} & a_3x_2 + \quad b_3x_3 & = d_3 \\ - & a_3x_2 + a_3c'_2x_3 & = a_3d'_2 \end{array}$$

$$\Rightarrow (b_3 - a_3c'_2)x_3 = d_3 - a_3d'_2$$

$$\Rightarrow x_3 = \frac{d_3 - a_3d'_2}{b_3 - a_3c'_2}$$

$$\Rightarrow x_3 = d'_3 \quad , \; d'_3 = \frac{d_3 - a_3d'_2}{b_3 - a_3c'_2}$$

3 Thomas Algorithm 경우

3x3 행렬

$$\begin{bmatrix} 1 & c'_1 & 0 \\ 0 & 1 & c'_2 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} d'_1 \\ d'_2 \\ d'_3 \end{bmatrix} \uparrow$$

X_3 = d'3 **아래**부터 **순차적** 대입

3 Thomas Algorithm 일반화

$$c'_i = \begin{cases} \dfrac{c_i}{b_i} & ; i = 1 \\ \dfrac{c_i}{b_i - a_i c'_{i-1}} & ; i = 2,3,\ldots,n-1 \end{cases}$$

$$x_n = d'_n$$

$$x_i = d'_i - c'_i x_{i+1} \qquad ; i = n-1,\, n-2,\ldots,1$$

$$d'_i = \begin{cases} \dfrac{d_i}{b_i} & ; i = 1 \\ \dfrac{d_i - a_i d'_{i-1}}{b_i - a_i c'_{i-1}} & ; i = 2,3,\ldots,n \end{cases}$$

04

역행렬과 Thomas Algorithm pseudo code & 시간복잡도

4 역행렬의 시간 복잡도

```
function inverse(matrix A):
    n = size(A)
    I = identity_matrix(n)  # n x n 단위 행렬을 생성

    # 행렬 A에 항등행렬 I을 오른쪽으로 결합
    augmented_matrix = concatenate(A, I, axis=1)

    # 가우시안 소거법 적용하여 좌측 부분(A)을 항등행렬로 변환
    for i from 1 to n:
        # 대각 요소를 1로 만들기 위해 현재 행 스케일링
        scaling_factor = 1 / augmented_matrix[i][i]
        for j from 1 to 2 * n:  # 2 * n은 augmented_matrix의 총 열의 개수
            augmented_matrix[i][j] *= scaling_factor

       # 현재 열의 다른 행 요소들을 제거
        for k from 1 to n:
            if i != k:
                # 현재 열의 위와 아래 요소들을 0으로 만들기 위한 행 연산 적용
                factor = augmented_matrix[k][i]
                for j from 1 to 2 * n:
                    augmented_matrix[k][j] -= factor * augmented_matrix[i][j]

    # 확장된 행렬의 오른쪽 부분에는 A의 역행렬이 형성됨
    inverse_A = get_right_half(augmented_matrix)

    return inverse_A
```

4 Thomas Algorithm 시간복잡도

```
# TDMA(thomas algorithm)
Input: n (the number of unknowns)
       a[2..n], b[1..n], c[1..n-1] (the tridiagonal matrix coefficients)
       d[1..n] (the right-hand side)

1. Initialize temporary variables:
   alpha[1..n], beta[1..n], x[1..n]

2. Forward Sweep:
   alpha[1] = b[1]
   beta[1] = d[1] / alpha[1]

   for i from 2 to n: # time complexity: O(n)
       alpha[i] = b[i] - (a[i] * c[i-1]) / alpha[i-1]
       beta[i] = (d[i] - a[i] * beta[i-1]) / alpha[i]

3. Backward Sweep: # time complexity: O(n)
   x[n] = beta[n]

   for i from n-1 to 1:
       x[i] = beta[i] - (c[i] * x[i+1]) / alpha[i]

4. x[1..n] now contains the solution to the linear system.

Output: x[1..n] (the solution to the linear system)

# Time Complexity = Forward sweep: O(n) + Backward sweep: O(n) = O(n)
```

4 Thomas Algorithm 시간복잡도

```
# TDMA(thomas algorithm)
Input: n (the number of unknowns)
       a[2..n], b[1..n], c[1..n-1] (the tridiagonal matrix coefficients)
       d[1..n] (the right-hand side)

1. Initialize temporary variables:
   alpha[1..n], beta[1..n], x[1..n]
```

```
2. Forward Sweep:
   alpha[1] = b[1]
   beta[1] = d[1] / alpha[1]

   for i from 2 to n: # time complexity: O(n)
       alpha[i] = b[i] - (a[i] * c[i-1]) / alpha[i-1]
       beta[i] = (d[i] - a[i] * beta[i-1]) / alpha[i]
```

```
3. Backward Sweep: # time complexity: O(n)
   x[n] = beta[n]

   for i from n-1 to 1:
       x[i] = beta[i] - (c[i] * x[i+1]) / alpha[i]

4. x[1..n] now contains the solution to the linear system.

Output: x[1..n] (the solution to the linear system)

# Time Complexity = Forward sweep: O(n) + Backward sweep: O(n) = O(n)
```

where $a_1 = 0$ and $c_n = 0$.

$$\begin{bmatrix} b_1 & c_1 & & & 0 \\ a_2 & b_2 & c_2 & & \\ & a_3 & b_3 & \ddots & \\ & & \ddots & \ddots & c_{n-1} \\ 0 & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}.$$

***i* =1 일 때**

$$\begin{bmatrix} b_1 & c_1 & 0 \\ a_2 & b_2 & c_2 \\ 0 & a_3 & b_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$$

$b_1x_1 + c_1x_2 = d_1$

$\rightarrow$ *divided by* b_1

$x_1 + \frac{c_1}{b_1}x_2 = \frac{d_1}{b_1}$ alpha

$x_1 + c'_1x_2 = d'_1 \ , \ c'_1 = \frac{c_1}{b_1} \ , \ d'_1 = \frac{d_1}{b_1}$ beta

***i*=2 일 때**

$$\begin{bmatrix} 1 & c'_1 & 0 \\ a_2 & b_2 & c_2 \\ 0 & a_3 & b_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} d'_1 \\ d_2 \\ d_3 \end{bmatrix}$$

$$\begin{aligned} a_2x_1 + b_2x_2 + c_2x_3 &= d_2 \\ -\ a_2x_1 + a_2c'_1x_2 &= a_2d'_1 \\ \Rightarrow (b_2 - a_2c'_1)x_2 + c_2x_3 &= d_2 - a_2d'_1 \\ \Rightarrow x_2 + \frac{c_2}{b_2 - a_2c'_1}x_3 &= \frac{d_2 - a_2d'_1}{b_2 - a_2c'_1} \\ \Rightarrow x_2 + c'_2x_3 = d'_2 \ , \ c'_2 = \frac{c_2}{b_2 - a_2c'_1} \ &, \ d'_2 = \frac{d_2 - a_2d'_1}{b_2 - a_2c'_1} \end{aligned}$$

beta

alpha

$\vdots$

$$c'_i = \begin{cases} \dfrac{c_i}{b_i} & ; i = 1 \\ \dfrac{c_i}{b_i - a_ic'_{i-1}} & ; i = 2,3,\dots,n-1 \end{cases}$$

alpha

$$d'_i = \begin{cases} \dfrac{d_i}{b_i} & ; i = 1 \\ \dfrac{d_i - a_id'_{i-1}}{b_i - a_ic'_{i-1}} & ; i = 2,3,\dots,n \end{cases}$$

beta

$x_n = d'_n$

$x_i = d'_i - c'_ix_{i+1} \quad ; i = n-1, n-2, \dots, 1$

4 Thomas Algorithm 시간복잡도

<전반부 for문>

i	연산횟수
1	1
2	6
3	6
⋮	⋮
n	6
$n+1$	1

$\Rightarrow 2+6(n-1)$

$= 6n-4$ 이므로 $O(n)$

<후반부 for문>

i	연산횟수
1	0
2	3
3	3
⋮	⋮
n	3
$n+1$	1

$\Rightarrow 1+3(n-1) = 3n-2$

$= O(n)$

05

실제 코드의 시간 비교

5 Thomas Algorithm 실제 코드

```
import numpy as np
import time

# TMDA 알고리즘을 위한 함수
def thomas(a, b, c, d):
    """ A는 삼중 대각 행렬의 계수 행렬이며, d는 RHS(우변) 행렬입니다"""
    N = len(a)  # 행렬의 크기 (a, b, c, d의 길이)
    cp = np.zeros(N, dtype='float64')  # 변환된 c 또는 c'를 저장하는 배열
    dp = np.zeros(N, dtype='float64')  # 변환된 d 또는 d'를 저장하는 배열
    X = np.zeros(N, dtype='float64')  # 미지수를 저장하는 배열

    # Forward Sweep 수행
    # Python에서 0으로 인덱싱된 Equation 1
    cp[0] = c[0] / b[0]  # c의 변환된 값 계산
    dp[0] = d[0] / b[0]  # d의 변환된 값 계산
    # Equation 2, ...., N (Python에서 1부터 N-1까지 인덱싱)
    for i in np.arange(1, N, 1):
        dnum = b[i] - a[i] * cp[i - 1]  # 분모 계산
        cp[i] = c[i] / dnum  # c의 변환된 값 계산
        dp[i] = (d[i] - a[i] * dp[i - 1]) / dnum  # d의 변환된 값 계산

    # Back Substitution 수행
    X[(N - 1)] = dp[N - 1]  # 마지막 xn을 구함

    for i in np.arange((N - 2), -1, -1):  # x[i]를 얻기 위해 x[i+1]을 사용
        X[i] = (dp[i]) - (cp[i]) * (X[i + 1])

    return X
```

5 Thomas Algorithm vs 역행렬

```python
def inv_vs_thomas_algorithm(a, b, c, d):
    nSize = 100  # 행렬 크기
    a = np.random.randn(nSize)  # a 배열을 무작위로 생성
    b = np.random.randn(nSize)  # b 배열을 무작위로 생성
    c = np.random.randn(nSize)  # c 배열을 무작위로 생성
    d = np.random.randn(nSize)  # d 배열을 무작위로 생성

    trid = np.zeros((nSize, nSize))  # 삼중 대각 행렬 초기화

    for i in range(nSize):
        trid[i, i] = b[i]  # 대각 원소 (b) 초기화
        if i > 0:
            trid[i, i - 1] = a[i]  # 하부 대각 원소 (a) 초기화
            trid[i - 1, i] = c[i - 1]  # 상부 대각 원소 (c) 초기화

    x_inv = np.linalg.inv(trid) @ d  # 역행렬을 사용하여 x_inv 계산
    x_thomas = thomas(a, b, c, d)  # Thomas 알고리즘을 사용하여 x_thomas 계산

    print('역행렬을 사용한 결과 :', x_inv)  # 역행렬을 사용한 결과 출력
    print('Thomas 알고리즘을 사용한 결과 :', x_thomas)  # Thomas 알고리즘을 사용한 결과 출력
    print(np.allclose(x_inv, x_thomas))  # 두 결과의 일치 여부 확인

    nIteration = 10000  # 반복 횟수
    start_time = time.time()
    for i in range(nIteration):
        x_inv = np.linalg.inv(trid) @ d  # 역행렬을 사용한 계산
    time1 = time.time() - start_time  # 경과 시간 계산

    start_time = time.time()
    for i in range(nIteration):
        x_inv = thomas(a, b, c, d)  # Thomas 알고리즘을 사용한 계산
    time2 = time.time() - start_time  # 경과 시간 계산

    print('역행렬을 사용한 경과 시간: {:.5f}초'.format(time1))
    print('Thomas 알고리즘을 사용한 경과 시간: {:.5f}초'.format(time2))
```

5 Thomas Algorithm vs 역행렬

size = 10

```
a = [0,-1,-1,1]
b = [2,2,2,1]
c = [-1,-1,-1,0]
d = [0,0,1,0]

inv_vs_thomas_algorithm(a,b,c,d)
```

```
역행렬을 사용한 결과 : [  0.57272766   4.57402786  21.20539873  14.17093307   2.90472178
   6.24957338  -6.5916429  -17.56264017 -20.65961415 -30.31411767]
Thomas 알고리즘을 사용한 결과 : [  0.57272766   4.57402786  21.20539873  14.17093307   2.90472178
   6.24957338  -6.5916429  -17.56264017 -20.65961415 -30.31411767]
True
역행렬을 사용한 결과 시간: 1.57430초
Thomas 알고리즘을 사용한 결과 시간: 0.22072초
```

size = 30

역행렬을 사용한 결과 시간: 5.37267초
Thomas 알고리즘을 사용한 결과 시간: 0.61603초

size = 100

역행렬을 사용한 결과 시간: 25.55699초
Thomas 알고리즘을 사용한 결과 시간: 2.16515초

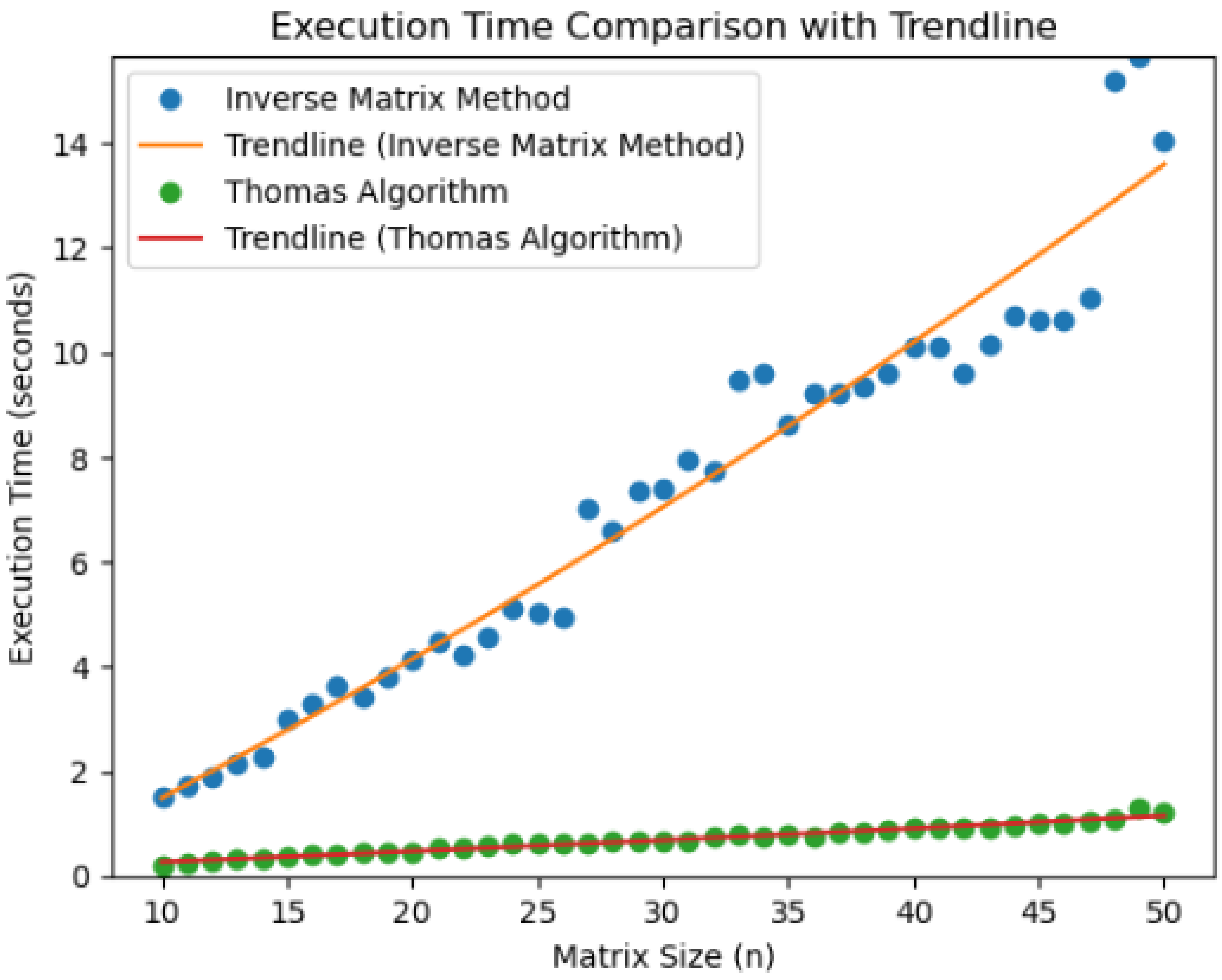
size = 300

역행렬을 사용한 결과 시간: 108.62173초
Thomas 알고리즘을 사용한 결과 시간: 6.13093초

size = 1000

역행렬을 사용한 결과 시간: 1029.58139초
Thomas 알고리즘을 사용한 결과 시간: 24.32044초

5 Thomas Algorithm vs 역행렬



Thomas Algorithm이

역행렬보다 효율적

06

토마스 알고리즘의 예시 (FDM)

6 유한차분법(FDM)

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

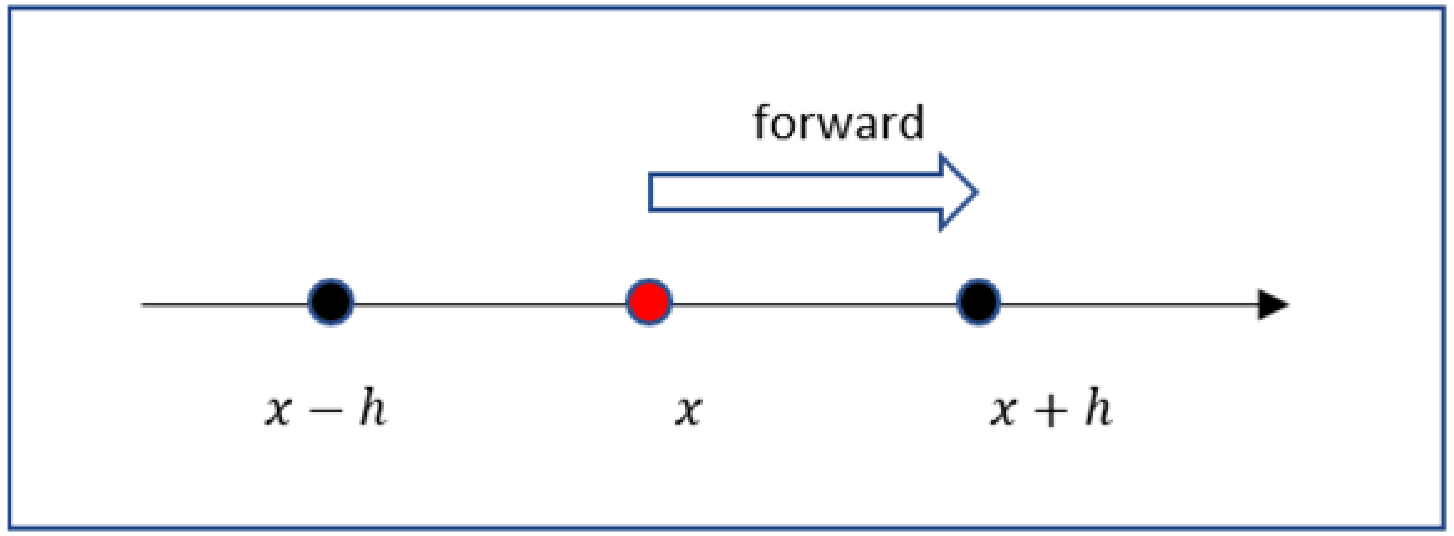
$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$ (1)

$$f'(x) \approx \frac{f(x-h) - f(x)}{-h} = \frac{f(x) - f(x-h)}{h}$$ (2)

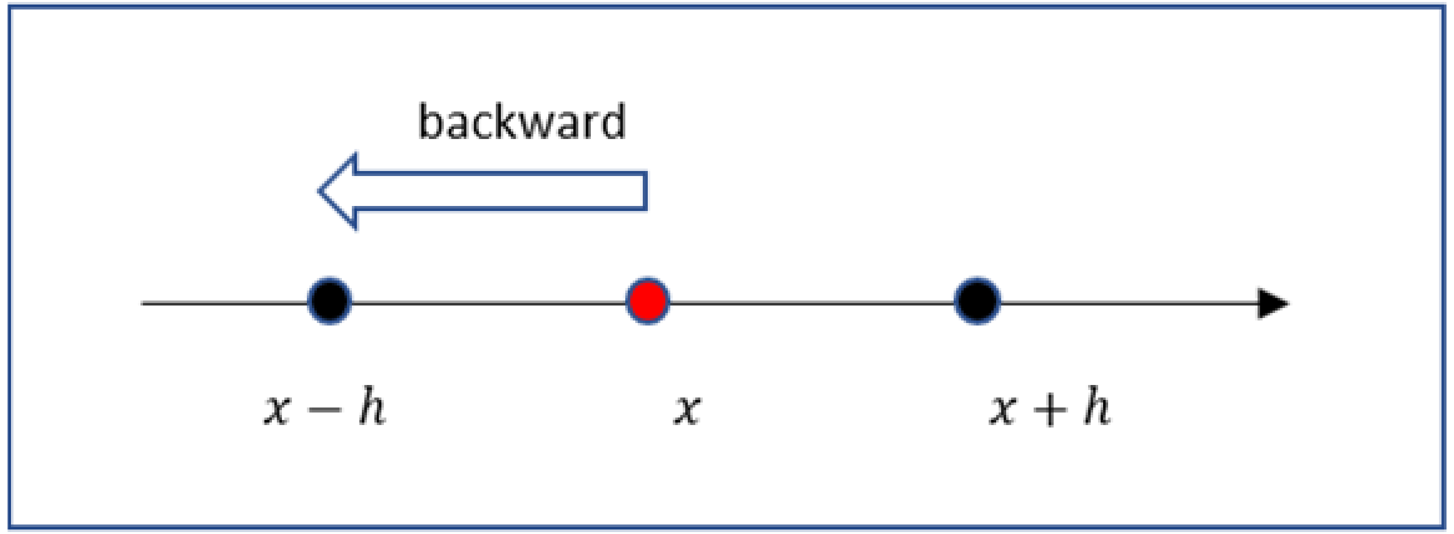
$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$ [(1)+(2)]/2

6 차분 과정

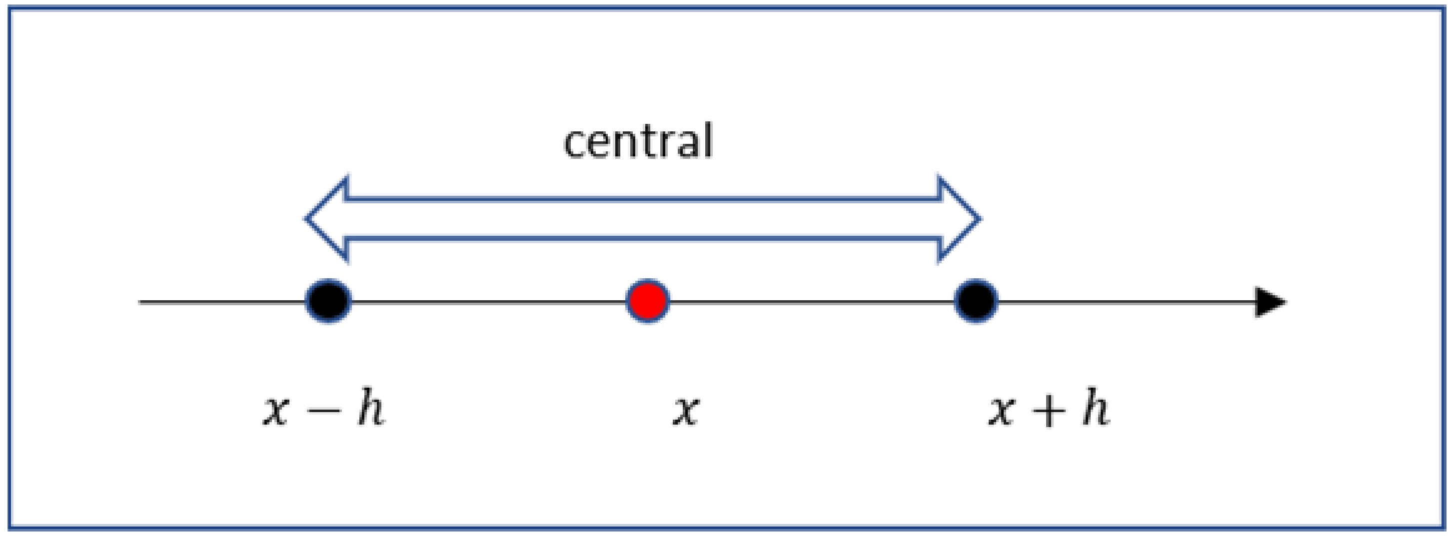
(1)



(2)



[(1)+(2)]/2



6 차분 과정

구분	유형	수식
$f'(x)$	forward difference	$\frac{f(x+h)-f(x)}{h}$
	Backward difference	$\frac{f(x)-f(x-h)}{h}$
	central difference	$\frac{f(x+h)-f(x-h)}{2h}$
$f''(x)$	central difference	$\frac{f(x+h)-2f(x)+f(x-h)}{h^2}$

6 FDM 실제 적용 – 상미분 방정식

$$f'(x) + pf(x) = q\ , f(0) = r \qquad (1)$$

다음과 같은 일계 상미분방정식을 풀어보자(p, q, r은 상수). x의 최대값을 x_{max}라 하고

$[0, x_{max}]$을 N등분하면

$$0 = x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N = x_{max}$$

x_{i+1}과 x_i사이의 간격을 h라 하자. 그리고 $f(x_i) = u_i$라고 정의하자.

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

6 FDM 실제 적용 – 상미분 방정식

$$\frac{u_{i+1} - u_i}{h} + pu_i = q\,,\ x_0 = r \qquad (2)$$

$$u_{i+1} + (ph - 1)u_i = qh\,,\ \ x_0 = r \qquad (3)$$

$$u_{i+1} = \frac{1}{1 + ph}u_i + \frac{qh}{1 + ph}\,,\ x_0 = r \qquad (4)$$

위 점화식을 통해 $\boxed{u_0, u_1, \dots, u_N}$ 을 구할 수 있다.

6 FDM 실제 적용 – 이계상미분 방정식

$0 \le x \le 1$

$$f''(x) + pf'(x) + qf(x) = 0\,, f(0) = y_0\,, f(1) = y_1 \qquad (1)$$

$$0 = x_0 < x_1 < x_2 < \cdots < x_{N-1} < x_N = x_{max}$$

x_{i+1}과 x_i사이의 간격을 h라 하자. 그리고 $f(x_i) = u_i$라고 정의

$$f''(x_i) = \frac{f(x_i+h) - 2f(x_i) + f(x_i-h)}{h^2} = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{h^2} = \frac{u_i - 2u_i + u_{i-1}}{h^2}$$

$$f'(x_i) = \frac{u_{i+1} - u_i}{h} \qquad \frac{u_i - 2u_i + u_{i-1}}{h^2} + p\frac{u_{i+1} - u_i}{h} + qu_i = 0$$

$$u_0 = y_0\,, u_N = y_1$$

6 FDM 실제 적용 – 이계상미분 방정식

$$\frac{u_i - 2u_i + u_{i-1}}{h^2} + p\frac{u_{i+1} - u_i}{h} + qu_i = 0$$

$$\frac{1}{h^2}u_{i-1} - \left(\frac{2}{h^2} + \frac{p}{h} - q\right)u_i + \left(\frac{1}{h^2} + \frac{p}{h}\right)u_{i+1} = 0\,, \qquad i \geq 1 \qquad (2)$$

$$a_i = \frac{1}{h^2}$$

$$b_i = -\left(\frac{2}{h^2} + \frac{p}{h} - q\right)$$

$$c_i = \left(\frac{1}{h^2} + \frac{p}{h}\right)$$

$$a_i u_{i-1} + b_i u_i + c_i u_{i+1} = 0$$

6 FDM 실제 적용 – 이계상미분 방정식

$i = 1$일 때 $b_1 u_1 + c_1 u_2 = \ -a_1 y_0$

$i = N-1$일 때 $a_{N-1} u_{N-2} + b_{N-1} u_{N-1} = \ -c_{N-1} y_1$

$$\begin{pmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & \ddots & & \\ & & & \ddots & \ddots & \\ & & & a_{N-2} & b_{N-2} & c_{N-2} \\ & & & & a_{N-1} & b_{N-1} \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{N-2} \\ u_{N-1} \end{pmatrix} = \begin{pmatrix} -a_1 y_0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ -c_{N-1} y_1 \end{pmatrix}$$

$$\boxed{Tu = k}$$

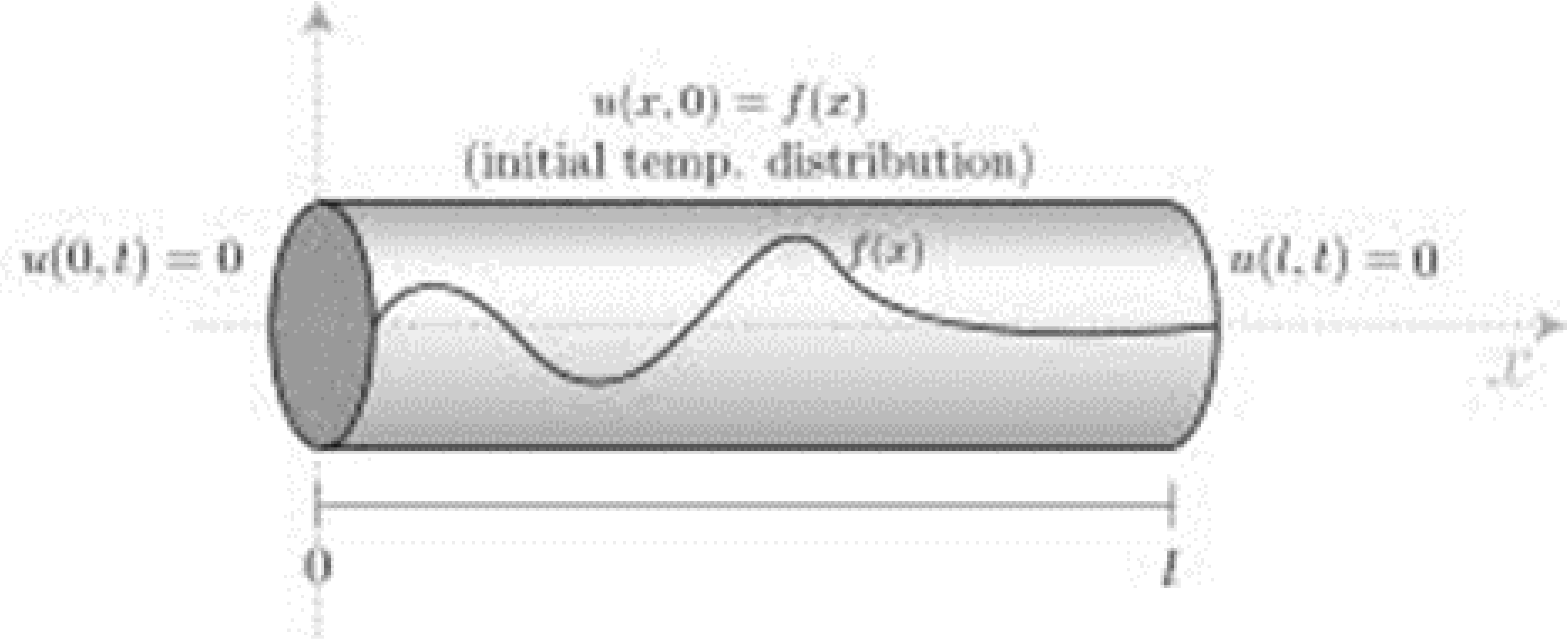
6 FDM과 편미분 방정식(Heat Equation)

시간 t와 변위 x에 관한 편미분방정식 $u(t,x)$가 $\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$의 형태일 때

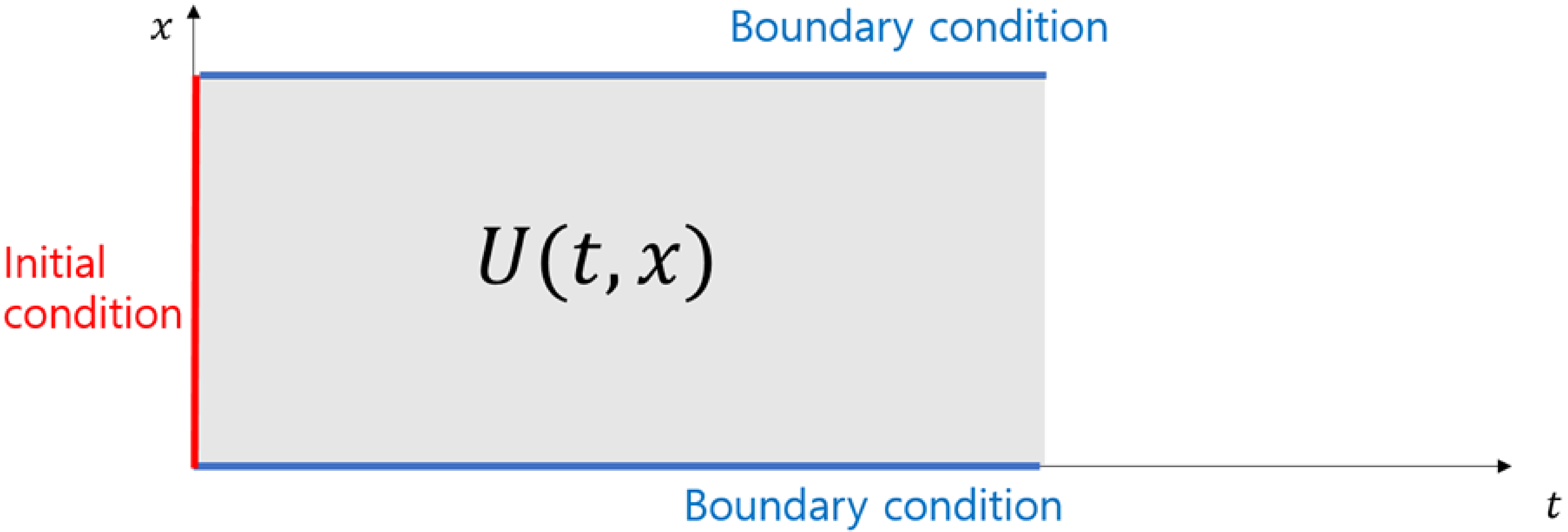
$t \geq 0$, $x \in [0, L]$, α는 상수

초기조건(initial condition): $u(0,x) = f(x), \forall x \in [0, L]$

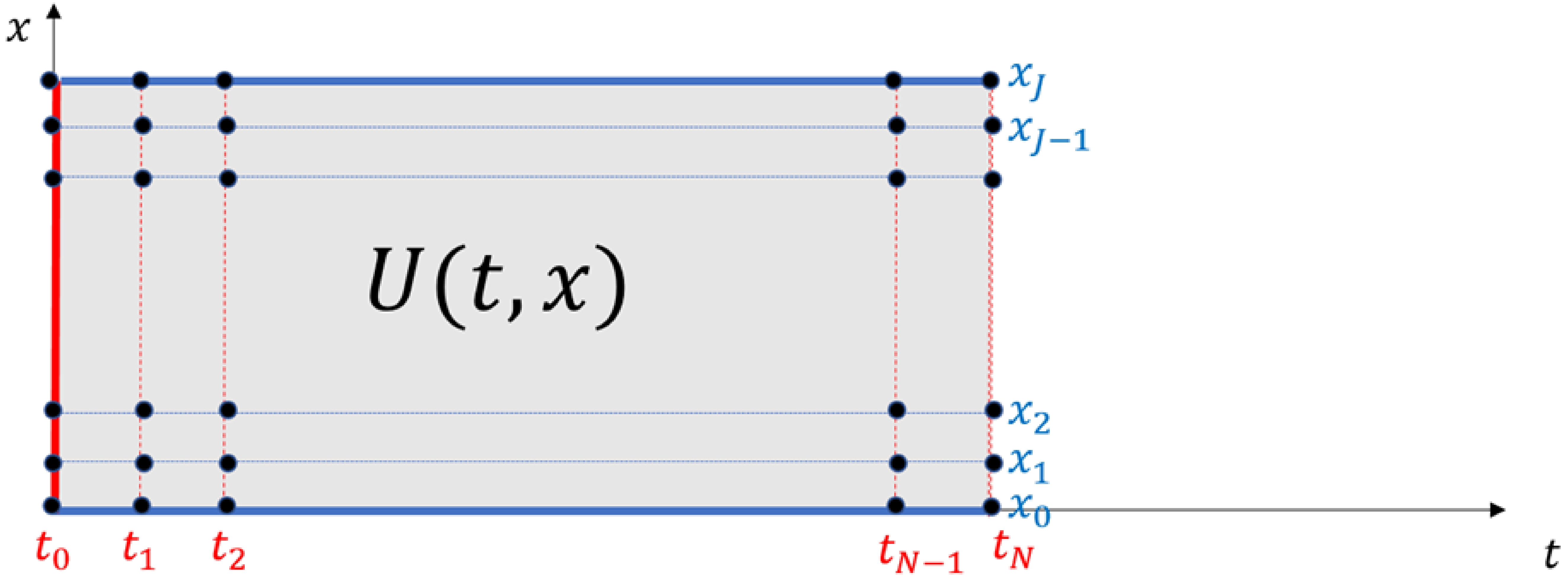
경계조건(boundary condition): $u(t,0) = 0,\ u(t,L) = 0$



6 FDM과 편미분 방정식(Heat Equation)



6 FDM과 편미분 방정식(Heat Equation)



6 블랙 숄즈 방정식(Black Scholes Equation)

파생 상품의 두 가지 특징

① **기초자산**의 가격 S_t에 연동해 **가치가 결정**

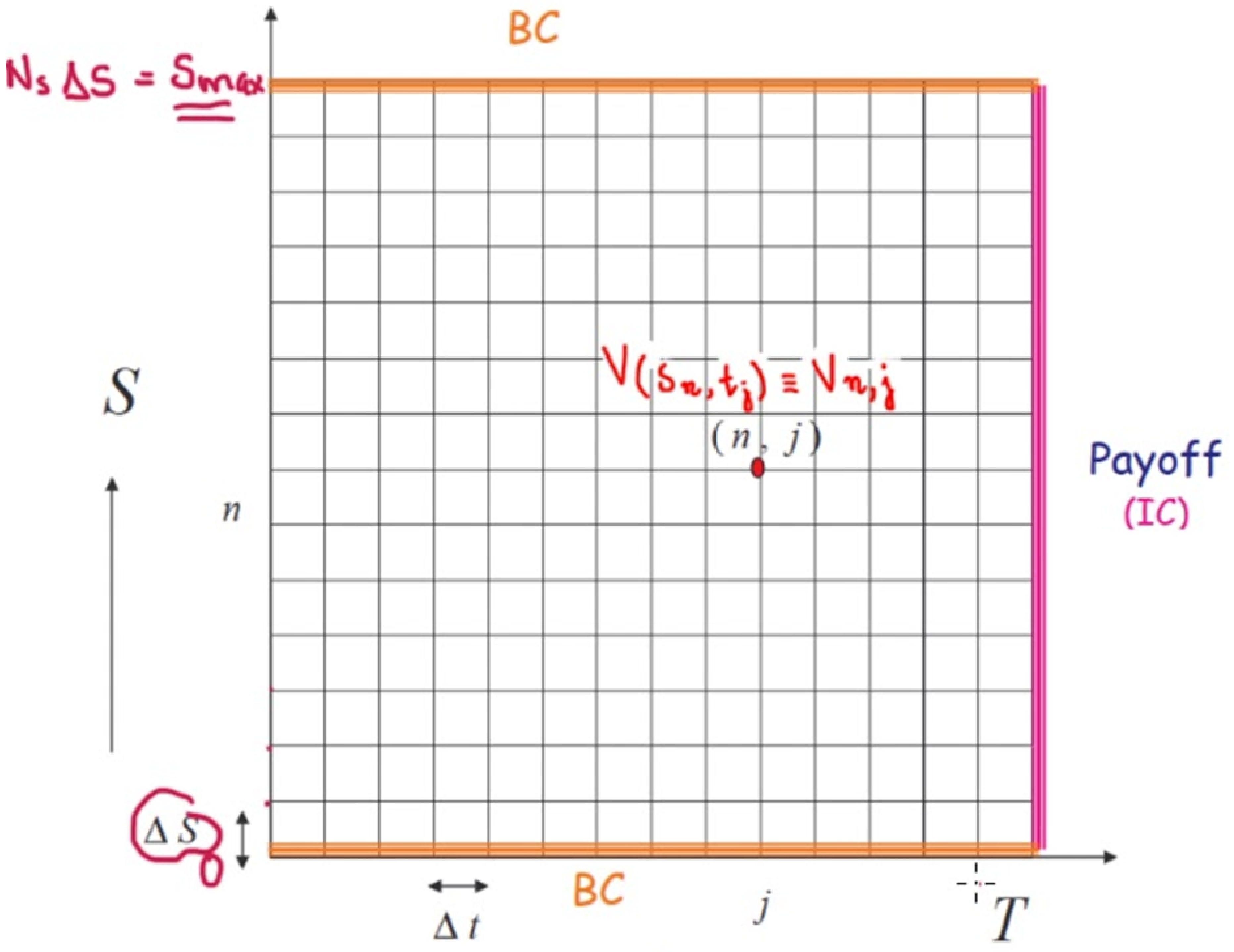
② 시간에 따라 가치가 변화 → **만기 T = 수익(payoff) 결정**

파생상품 가격 f = $f(t, S_t)$

(r은 무위험 이자율(risk-free interest rate), q는 연속 배당)

$$\left[\; f_t(t,S_t) + (r-q)S_t f_S(t,S_t)dt + \frac{1}{2}\sigma^2 S_t^2 f_{SS}(t,S_t)dt - rf(t,S_t) = 0 \;\right]$$

6 블랙 숄즈 방정식(Black Scholes Equation)



Team. TWO-rillion