

[수치해석] Interpolation

Fitting

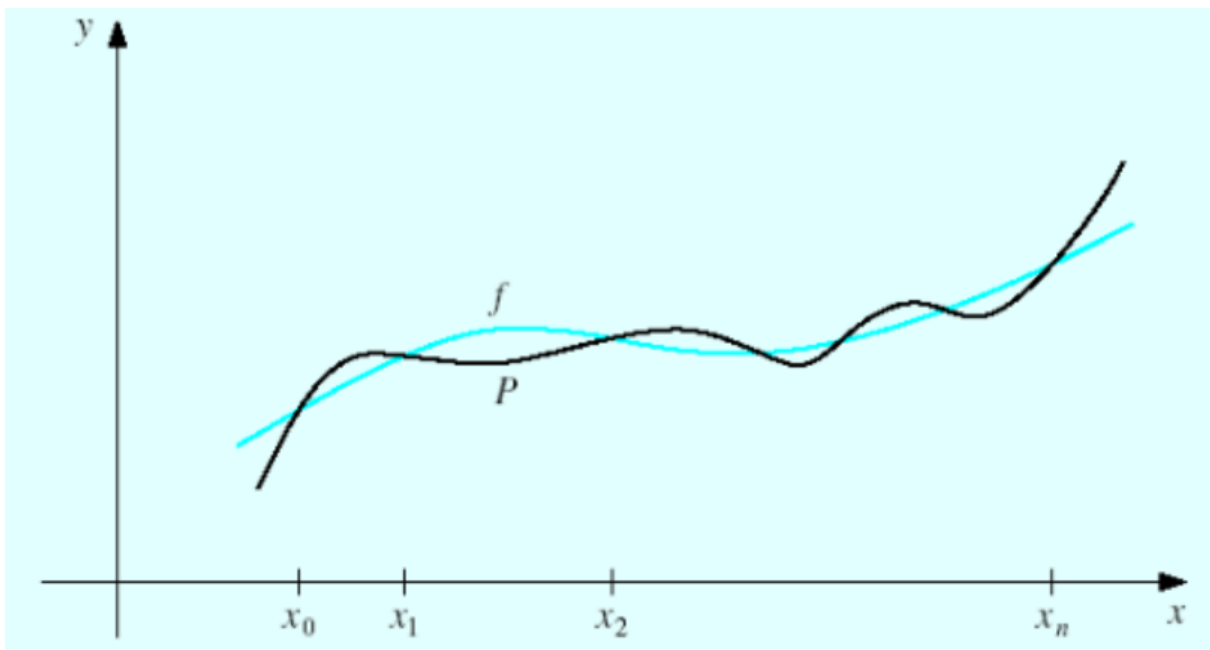
Exact Fitting: 특정 데이터를 모두 지나는 함수/곡선을 표현하는 방법.

1. **Extrapolation:** 데이터가 존재하지 않는 범위에서 값을 추정하는 것
2. **Interpolation:** 데이터가 존재하는 범위 내에서 값을 추정하는 것

Apporximation Fitting: 모든 데이터를 지나는 대신, 오차가 최소화되도록 하는 함수/곡선을 찾는 방법

Weierstrass Approximation Theorem

주어진 연속 함수 $f(x)$ 에 대해 구간 전체에서 $\|f(x) - P(x)\| < \epsilon$ 을 만족하도록 하는 다항식 $P(x)$ 가 항상 존재한다.



Approximation error: $\|f(x) - P(x)\|$

그럼 이제 모든 데이터를 정확히 지나는 **다항식을 얻는 방법**에 알아보자.

Lagrange Interpolating Polynomial

실제 함수 **f**에 대해 $(x_0, f_0), (x_1, f_1), \dots, (x_n, f_n)$ 데이터를 모두 지나는 **n차 P(x)**를 얻고 싶은 상황을 생각해보자.

- (n+1)개의 데이터를 모두 지나는 n차 다항식은 단 하나 존재한다.

각 데이터에 대해 **Basis function**을 먼저 정의할 수 있다.

$$\begin{aligned} L_o(x) &= \frac{g_o(x)}{g_o(x_o)} \\ &= \frac{(x-x_1)(x-x_2)\cdots(x-x_n)}{(x_o-x_1)(x_o-x_2)\cdots(x_o-x_n)} \\ &= \begin{cases} 1, & x = x_o \\ 0, & x = x_1, x_2, \dots, x_n \end{cases} \end{aligned}$$

$$L_i(x) = \prod_{k=0, k \neq i}^n \left(\frac{x-x_k}{x_i-x_k} \right)$$

위 **Basis function**을 바탕으로 하나의 다항식을 얻을 수 있다.

$$P_n(x) = L_0(x)f_0 + L_1(x)f_1 + \dots + L_n(x)f_n$$

- f_i : 함수값

이 방법으로 **Unique**한 다항식을 얻을 수 있지만 **너무 복잡하다는 단점**이 있다.

위 방식으로 얻은 다항식의 **Interpolation error**를 구해보자.

$$\text{Error} = f(x) - P_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n)u(x)$$

- 모든 데이터 (x_0, x_1, \dots, x_n) 을 통과하기 때문에 해당 점에서는 0이 되도록 하기 위해 $(x - x_i)$ 항을 곱한다.
- $u(x)$: Error function

$$f(x) - P_n(x) - (x - x_0)(x - x_1) \cdots (x - x_n)u(x) = 0$$

$$v(t) = f(t) - P_n(t) - (t - x_0)(t - x_1) \cdots (t - x_n)u(x) \text{라고 하자.}$$

- t 가 변수, x 는 상수 취급한다.

$v(t) = 0$ 은 **n+2개의 Solution**을 갖게 된다.

- $x_0 \sim x_n$ 의 n+1개의 Data point
- $t = x$
 - $v(x) = f(x) - P_n(x) - (x - x_0)(x - x_1) \cdots (x - x_n)u(x) = 0$

이제, $v(t)$ 를 t 에 대해 (n+1)번 미분하자. 아래 정리에 의해 $v^{(n+1)}(t) = 0$ 은 **한 개의 Solution**만을 갖는다.

※ Generalized Roll's Theorem

$$\left[\begin{array}{ll} v'(t) = 0 & \Rightarrow (n+1) \text{ solutions} \\ v''(t) = 0 & \Rightarrow n \text{ solutions} \\ \vdots & \vdots \\ v^{(n+1)}(t) = 0 & \Rightarrow \text{1 solutions} \end{array} \right. \quad t = \xi$$

실제로 미분해보자.

$$v^{(n+1)}(\xi) = f^{(n+1)}(\xi) - 0 - (n+1)!u(x) = 0$$

- $(x - x_0)(x - x_1) \cdots (x - x_n)$ 을 미분하면 $(n+1)!$
- $P(t)$ 는 n 차 다항식이다.

위 식을 $u(x)$ 에 대해 정리할 수 있다.

$$u(x) = \frac{f^{(n+1)}(\epsilon)}{(n+1)!}$$

따라서 **원래의 Error**는 다음과 같이 정리할 수 있다.

$$f(x) - P_n(x) = (x - x_0)(x - x_1) \cdots (x - x_n) \frac{f^{(n+1)}(\epsilon)}{(n+1)!}$$

Difference

함수를 모르기 때문에 데이터로부터 미분값도 근사해야 한다. 이런 경우, **Difference**를 사용한다.

Difference	정의	의미
Forward	$\nabla f_i = f_{i+1} - f_i$	한 칸 앞으로의 변화량
Backward	$\nabla f_i = f_i - f_{i-1}$	한 칸 뒤로의 변화량
Central	$\nabla f_{i+\frac{1}{2}} = f_{i+1} - f_i$	가운데 점 기준 변화량

Divided Difference

주어진 데이터만 가지고 기울기를 추측하는 방법

1. First-order divided difference

$$f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0}$$

- 평균값 정리에 의해 $f[x_0, x_1] = f'(\xi_1)(x_0 \leq \xi_1 \leq x_1)$ 가 성립한다.

2. Second-order divided difference

$$\begin{aligned} f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\ &= \frac{f_0}{(x_0 - x_1)(x_0 - x_2)} + \frac{f_1}{(x_1 - x_0)(x_1 - x_2)} + \frac{f_2}{(x_2 - x_0)(x_2 - x_1)} \end{aligned}$$

- 기울기의 차이의 평균 → 2차 미분과 비슷

3. N-order divided difference

$$\begin{aligned} f[x_0, \dots, x_n] &= \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{x_n - x_0} \\ &= \frac{f_0}{(x_0 - x_1) \dots (x_0 - x_n)} + \dots + \frac{f_n}{(x_n - x_0)(x_n - x_{n-1})} \end{aligned}$$

- Second-order divided difference을 자연스럽게 확장한 것과 동일하다.

Newton's Interpolation

실제 함수 f 에 대해 $(x_0, f_0), (x_1, f_1), \dots, (x_n, f_n)$ 데이터를 모두 지나는 n 차 $P(x)$ 를 Divided Difference를 이용하여 Lagrange Interpolation보다 간단하게 구하는 방법

아래 다항식이 공식이다.

$$P_n(x) = f_0 + (x - x_0)a_1 + (x - x_0)(x - x_1)a_2 + \cdots + (x - x_0) \cdots (x - x_{n-1})a_n$$

- 적절한 a_i 를 구해서 함수값을 맞춘다.

Newton's Interpolation이 간단한 이유

기존 Data에 하나의 Data가 추가될 때, Lagrange처럼 L_i 를 전부 새로 계산하는 대신 하나의 항을 하나만 추가하기만 하면 된다.

$$(x - x_0) \cdots (x - x_{n-1})a_n$$

계수 구하는 방법

a_n 을 예시로 알아보자.

$$P_n(x_n) - P_{n-1}(x_n) = (x_n - x_0) \cdots (x_n - x_{n-1})a_n$$

$P_n(x_n) = f_n$ 이기 때문에, 아래 공식처럼 변환할 수 있다.

$$f_n - P_{n-1}(x_n) = (x_n - x_0) \cdots (x_n - x_{n-1})a_n$$

$$\Rightarrow a_n = \frac{f_n - P_{n-1}(x_n)}{(x_n - x_0) \cdots (x_n - x_{n-1})}$$

a_n 이 **N-order Divided Difference**와 동일한 것을 확인할 수 있다.

따라서 아래와 같이 정리할 수 있다.

$$P_1(x) = f_0 + (x - x_0)\mathcal{A}[x_0, x_1]$$

:

$$P_n(x) = f_0 + (x - x_0)\mathcal{A}[x_0, x_1] + \cdots + (x - x_0)(x - x_1) \cdots (x - x_{n-1})\mathcal{A}[x_0, \cdots, x_n]$$

결국 각 $P_i(x) = P(x_i)$ 는 아래와 같이 행렬로 나타낼 수 있다.

- 식 $\mathbf{A} \cdot \mathbf{a} = \mathbf{y}$ 의 형태이다.
- **Linear equation**을 푸는 것과 동일해진다.

$$\begin{bmatrix} 1 & & & & 0 \\ 1 & x_1 - x_0 & & & \\ 1 & x_2 - x_0 & (x_2 - x_0)(x_2 - x_1) & & \\ \vdots & \vdots & & \ddots & \\ 1 & x_k - x_0 & \dots & \dots & \prod_{j=0}^{k-1} (x_k - x_j) \end{bmatrix} \begin{bmatrix} a_0 \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} y_0 \\ \vdots \\ y_k \end{bmatrix}$$

Forward Difference version

Forward Difference가 가능한 상황을 가정하면 더 쉽다.

각 데이터 포인트끼리 **동일한 간격**을 가진다.

- $h = x_{i+1} - x_i$
- $x_i = x_0 + i \cdot h$

이런 상황에선 미분이 더 쉽게 표현된다.

Derivation

n=1

$$a_1 = f[x_0, x_1] = \frac{f_1 - f_0}{x_1 - x_0} = \frac{f_1 - f_0}{h} = \frac{1}{1!h} \Delta f_0$$

n=2

$$\begin{aligned} a_2 = f[x_0, x_1, x_2] &= \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} \\ &= \frac{\frac{\Delta f_1}{h} - \frac{\Delta f_0}{h}}{x_2 - x_0} = \frac{\Delta f_1 - \Delta f_0}{2h^2} = \frac{1}{2!h^2} \Delta^2 f_0 \end{aligned}$$

Newton's Interpolation도 더 쉽게 표현할 수 있다. a_n 이 아래와 같이 정리된다.

$$\begin{aligned} a_n = f[x_0, \dots, x_n] &= \frac{f[x_1, \dots, x_n] - f[x_0, \dots, x_{n-1}]}{nh} \\ &= \frac{1}{n!h^n} \Delta^n f_0 \end{aligned}$$

결국, **다항식** P_n 또한 아래와 같이 정리된다.

$$\begin{aligned} P_n(x) &= f_0 + s\Delta f_0 + \frac{s(s-1)}{2!} \Delta^2 f_0 + \dots + \frac{s(s-1)(s-n+1)}{n!} \Delta^n f_0 \\ &= \sum_{k=0}^n \underbrace{\binom{s}{k}}_{\text{Binomial coef.}} \Delta^k f_0 \end{aligned} \quad (x = x_0 + sh)$$

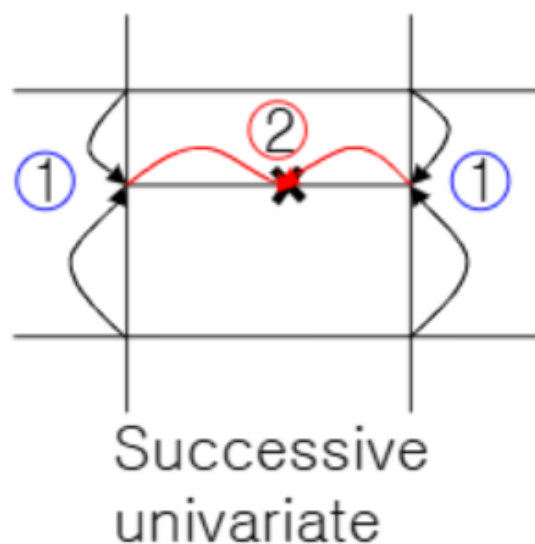
Newton's Interpolation의 Error를 정리하면 다음과 같다.

$$E(x) = \binom{s}{n+1} h^{n+1} f^{(n+1)}(\xi) = O(h^{n+1}), \quad x_0 \leq \xi \leq x_n$$
$$\cong \binom{s}{n+1} \Delta^{n+1} f_0$$

Interpolate of Multivariate Function

1. Successive univariate polynomial

한 변수에 대해 Interpolation을 반복하여 다변수 Interpolation을 구현하는 방법



특징

- 구현 쉬움
- 다변수 형태를 정확히 반영하지 못하는 경우가 있음

Bilinear interpolation

가로(X) 방향과 세로(Y) 방향으로 두 번 보간하여 (X, Y) 값을 예측하는 방법

주변 4개의 점을 이용하여 하나의 점을 예측한다.

예측하고자 하는 점: (x, y) , $Q_{11}: (x_1, y_1)$, $Q_{21}: (x_2, y_1)$, $Q_{12}: (x_1, y_2)$, $Q_{22}: (x_2, y_2)$ 일 때 동작은 다음과 같다.

1. y_1 의 x 방향으로 R_1 을 구한다.

$$\bullet R_1 = \frac{x-x_1}{x_2-x_1}Q_{21} + \frac{x_2-x}{x_2-x_1}Q_{11}$$

2. y_2 의 x 방향으로 R_2 를 구한다.

$$\bullet R_2 = \frac{x-x_1}{x_2-x_1}Q_{22} + \frac{x_2-x}{x_2-x_1}Q_{12}$$

3. y 방향으로 Interpolation을 한다.

$$\bullet (x, y) = \frac{y_2-y}{y_2-y_1}R_1 + \frac{y-y_1}{y_2-y_1}R_2$$

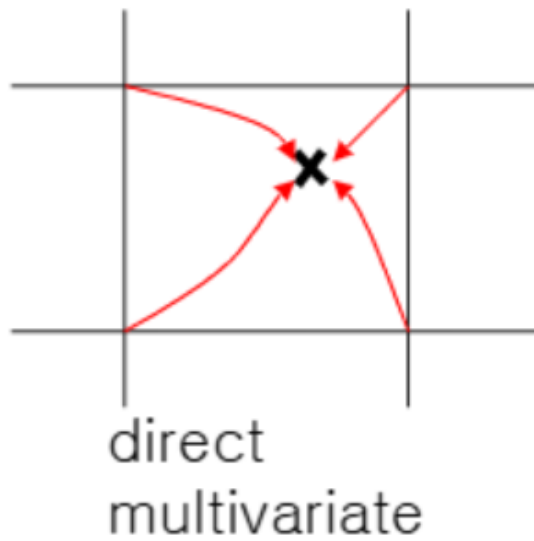
Bicubic interpolation

주변 16개의 Pixel을 사용해서 예측하는 방법

- Bilinear interpolation에 비해 훨씬 부드러운 결과를 얻을 수 있다.
- Bilinear interpolation에서는 Weight가 1차이지만 Bicubic interpolation은 Weight가 3차이다.

2. Direct multivariate polynomial

다변수 다항식을 이용하여 직접 해결하는 방법



특징

- 정확도 증가
- 계산 비용 커짐

주어진 $y = f(x)$ 에서 **y**를 생성하는 **x**를 찾고 싶은 경우에는 역함수를 이용한다.

하지만 역함수를 찾기 어려울 때는 아래 방법을 이용할 수 있다.

Inverse Interpolation

┃ $f(x)$ 를 근사하는 Interpolation polynomial을 만들고 그 다항식을 거꾸로 풀어서 x 를 얻는법

1. $f(x)$ 를 Newton's Polynomial을 이용하여 근사했다.

$$f(x) \cong P_n(x) = f_i + (x - x_i)f[x_i, x_{i+1}] + (x - x_i)(x - x_{i+1})f[x_i, x_{i+1}, x_{i+2}] + \dots + (x - x_i) \cdots (x - x_{i+n-1})f[x_i, \dots, x_{i+n}]$$

2. 가장 먼저 First-order divided difference의 x 에 대해 정리하자.

Solve for x

$$x = \frac{f(x) - f_i - \{(x - x_i)(x - x_{i+1})f[x_i, x_{i+1}, x_{i+2}] + \dots\}}{f[x_i, x_{i+1}]} + x_i$$

1st approximation

$$x_1 = \frac{f(x) - f_i}{f[x_i, x_{i+1}]} + x_i$$

x_1 을 구하기 위해서 2차 이상의 항은 무시한다.

3. (2)에서 x_1 을 구했기 때문에 x_1 을 이용하여 Second-order divided difference를 x 에 대해 정리한다.

$$x_2 = \frac{f(x) - f_i - (x_1 - x_i)(x_1 - x_{i+1})f[x_i, x_{i+1}, x_{i+2}]}{f[x_i, x_{i+1}]} + x_i$$

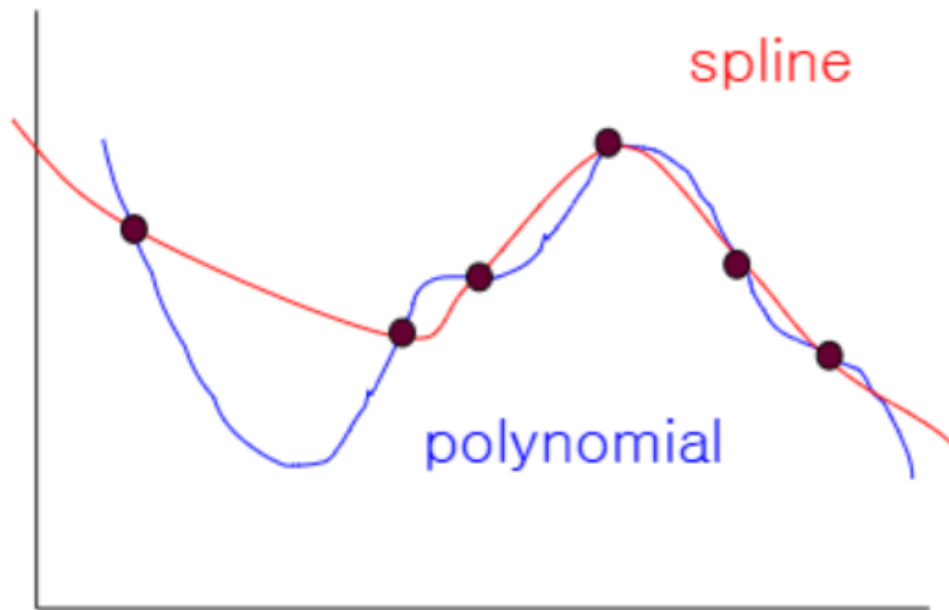
4. 위 과정을 수렴할 때까지 반복한다.

안정적이고 좋은 예측을 위한 Interpolation 기법에 대해서도 살펴보자.

Spline Interpolation

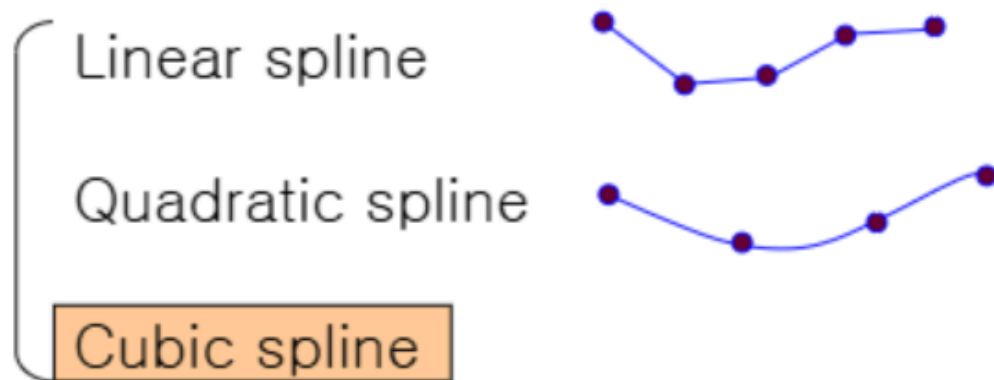
| Spline을 이용하여 Polynomial interpolation에 비해 안정적이고 정확한 곡선을 만드는 방법

데이터 포인트 간의 각 간격마다 다항식을 다르게 설정하여 연결하는 방법이다.



여러 종류의 **Spline**이 있다.

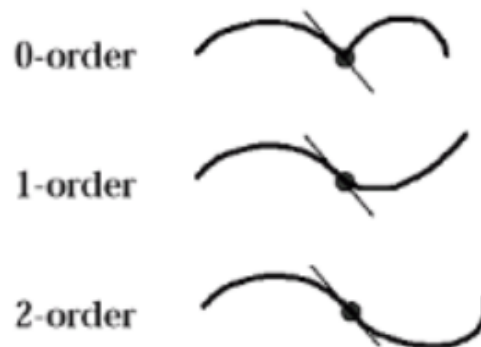
- Spline은 각 간격마다의 다항식이 어떻게 연결되어야 하는지를 정한다.



- $f'_{i-1}(x_i) = f'_i(x_i)$
- $f''_{i-1}(x_i) = f''_i(x_i)$

- Cubic spline은 각 **Data점**에서 기울기와 곡률까지 같도록 한다.

Continuity 관점에서 보면 **Linear - 0 order / Quadratic - 1 order / Cubic - 2 order** 와 대응 된다.



Cubic spline interpolation

Cubic의 경우 각 간격에 **3차 다항식**을 사용한다.

- $f_i(x) = a(x - x_i)^3 + b(x - x_i)^2 + c(x - x_i) + d$

데이터가 $(n+1)$ 개라면 구간은 n 개이다.

- 각 구간마다 미지수가 4개이므로 미지수는 총 $4n$ 개이다.
- $4n$ 개의 미지수를 구하기 위해선 $4n$ 개의 식이 필요하다.

"구간마다 구간의 양 끝 값이 같다."와 "연결되는 구간의 각 2차 기울기와 각 1차 기울기가 연결되어야 한다."는 조건으로 $4n-2$ 개의 식을 얻을 수 있다.

$$1) f_i(x_i) = y_i, \quad i = 0, 1, \dots, n-1 \quad (n)$$

$$2) f_i(x_{i+1}) = y_{i+1}, \quad i = 0, 1, \dots, n-1 \quad (n)$$

3) continuity of f'

$$f'_{i-1}(x_i) = f'_i(x_i), \quad i = 1, 2, \dots, n-1 \quad (n-1)$$

4) continuity of f''

$$f''_{i-1}(x_i) = f''_i(x_i), \quad i = 1, 2, \dots, n-1 \quad (n-1)$$

따라서 2개의 식을 추가로 얻기 위해, **Boundary condition**을 추가하는 여러 방법이 있다.

$$\left\{ \begin{array}{l} \text{Method 1 : } f''_0(x_0) = 0, f''_{n-1}(x_{n-1}) = 0 \quad \text{natural cubic} \\ \text{Method 2 : } f''_0(x_0) = f''_0(x_1), f''_{n-1}(x_{n-1}) = f''_{n-1}(x_n) \\ \text{Method 3 : } f''_1(x_1) = \frac{f''_0(x_0) + f''_2(x_2)}{2}, f''_{n-1}(x_{n-1}) = \frac{f''_{n-2}(x_{n-2}) + f''_{n-1}(x_n)}{2} \end{array} \right.$$

- 방법 1: 양 끝 구간을 직선으로 만든다.

- 방법 2: 양 끝 구간의 곡률을 상수 \rightarrow **양 끝을 포물선**으로 부드럽게 만든다.
- 방법 3: x_0, x_1, x_2 에서 2계 도함수의 변화율(기울기)이 일정 \rightarrow 첫 번째에서 두 번째 구간까지를 **하나의 3차 곡선**으로 잇는다.