Math as Code (Python 버전)

이 문서는 복잡한 수학 표기법을 Python 코드와 비교하여, 개발자들이 수학적 개념을 프로그래밍 관점에서 더 쉽게 이해할 수 있도록 돕는 참고 자료입니다.

서문

수학 기호는 저자, 문맥, 그리고 연구 분야(선형대수학, 집합론 등)에 따라 각기 다른 의미를 가질 수 있습니다. 본 가이드는 기호의 모든 용도를 포괄하지는 않지만, 실제 어떻게 활용되는지 보여주기 위해 블로그 포스트나 출판물 등의 참고 자료를 적극 인용하였습니다.

더 자세한 목록은 위키피디아 - 수학 기호 목록을 참조해 주시기 바랍니다.

참고로, 여기에 제시된 많은 코드 예제는 부동 소수점 값을 사용하므로 수치적으로 완벽하게 안정적이지 않을 수 있습니다. 수치 해석상의 주의 사항에 대해서는 Mikola Lysenko의 Robust Arithmetic Notes를 참고하세요.

목차

변수 명명 규칙

수학에서도 분야에 따라 다양한 명명 규칙이 존재하며 항상 일관되지는 않습니다. 그러나 일반적인 수학 문헌에서는 다음과 같은 패턴이 자주 발견됩니다.

- s - 스칼라(Scalar, 단일 숫자): 이탤릭체 소문자
- **x** - 벡터(Vector, 2D 점 등): 굵은 소문자
- **A** - 행렬(Matrix, 3D 변환 등): 굵은 대문자
- θ - 상수 및 특수 변수(예: 극좌표의 각도 세타): 이탤릭체 소문자 그리스 문자

본 가이드에서도 이 관례를 따릅니다.

NumPy와 배열 프로그래밍

NumPy는 파이썬에서 과학 계산을 수행하기 위한 핵심 라이브러리로, 강력한 **배열 프로그래밍** 기능을 제공합니다. 수학적 표기를 코드로 옮길 때, 파이썬과 NumPy가 네임스페이스를 공유하며 특별한 문법적 설탕(Syntactic sugar)을 통해 서로 접근한다고 생각하면 이해가 쉽습니다. 특히 벡터와 행렬 연산 시, 파이썬의 기본 문법과 NumPy 문법은 성능 면에서 큰 차이를 보이므로 주의가 필요합니다. 관례적으로 `import numpy as np` 형식을 사용합니다.

등호 및 부등호 기호

등호 =와 유사하지만 의미가 다른 여러 기호가 존재합니다.

- = : 동등 (두 값이 같음)
- ≠ : 부등 (두 값이 다름)
- ≈ : 근사 (값이 거의 같음, 예: π ≈ 3.14159)
- := : 정의 (A를 B라는 수식으로 정의함)

파이썬 코드로 표현하면 다음과 같습니다.

```
## 동등 여부 확인
2 == 3

## 부등 여부 확인
2 != 3

## 근사치 비교
import math
# math.isclose는 허용 오차를 설정할 수 있습니다.
math.isclose(math.pi, 3.14159, abs_tol=1e-5)

from numpy.testing import assert_almost_equal
# 5째 자리까지 일치하는지 확인
assert_almost_equal(math.pi, 3.14159, 5)

def almost_equal(x, y, epsilon=1e-7):
    """직접 구현하는 경우: 두 값의 차이가 아주 작은 값(엡실론)보다 작은지 확인합니다."""
    return abs(x - y) < epsilon
```

수학에서는 :=, =:, = 기호가 **정의(Definition)**의 의미로 쓰일 수 있습니다. 예를 들어 아래는 x를 $2kj$라는 수식의 별칭으로 정의한다는 뜻입니다.

$x := 2kj$

파이썬에서는 = 연산자를 사용하여 변수를 정의하고 값을 할당합니다.

```
x = 2 * k * j
```

반면, $x = 2kj$와 같은 수식은 문맥에 따라 단순히 "값이 같다"는 동등성을 나타낼 수도 있습니다.

주의: 코드에서 =는 **할당 명령**이고 ==는 **논리 비교**로 명확히 구분되지만, 수학 문헌에서는 = 기호가 문맥에 따라 정의와 비교 두 가지 의미로 혼용되므로 주의 깊게 살펴봐야 합니다.

제곱근과 복소수

제곱근(Square root) 연산은 다음과 같은 성질을 갖습니다.

$\sqrt{x^2} = x$

프로그래밍에서는 주로 `sqrt` 함수를 사용합니다.

```
import math
print(math.sqrt(2)) # 결과: 1.4142135623730951
```

복소수(Complex numbers)는 $a + ib$ 형태의 수로, a는 실수부, b는 허수부입니다. 여기서 허수 단위 i는 다음과 같이 정의됩니다.

$i = \sqrt{-1}$

파이썬은 기본적으로 `complex` 타입과 `cmath` 표준 모듈을 통해 복소수 연산을 지원합니다. (파이썬에서는 허수 단위로 i 대신 j를 사용합니다)

```
complex(1, 1)  # (1+1j)

import cmath
cmath.sqrt(complex(-1, 0)) # (0+1j)
```

내적 및 외적 연산

점 · 기호와 십자 × 기호는 문맥에 따라 스칼라 곱셈이나 벡터 연산으로 나뉩니다.

스칼라 곱셈

단순한 숫자의 곱셈에서는 두 기호가 동일한 의미로 쓰입니다.

$5 \cdot 4 = 5 \times 4$

대부분의 프로그래밍 언어에서는 별표 *를 사용합니다.

벡터 및 행렬 연산

벡터 간의 요소별 곱셈(Element-wise product)에는 주로 기호를 생략하거나 열린 점 ◦을 사용하여 아다마르 곱(Hadamard product)임을 명시합니다.

$3\boldsymbol{k} \circ \boldsymbol{j}$

내적 (Dot Product)

점 · 기호는 두 벡터의 내적을 의미하기도 합니다. 연산 결과가 스칼라 값이므로 '스칼라 곱'이라고도 불립니다.

$\boldsymbol{k} \cdot \boldsymbol{j}$

```
k = [0, 1, 0]
j = [1, 0, 0]
# numpy를 이용한 내적 계산
d = np.dot(k, j) # 결과: 0
```

외적 (Cross Product)

십자 × 기호는 두 벡터의 외적을 의미합니다.

$\boldsymbol{k} \times \boldsymbol{j}$

```
k = [0, 1, 0]
j = [1, 0, 0]
# numpy를 이용한 외적 계산
result = np.cross(k, j) # 결과: [ 0, 0, -1 ]
```

합산 기호 시그마 Σ

대문자 그리스 문자 Σ (시그마)는 수열의 합산을 나타냅니다.

$\sum_{i=1}^{100} i$

위 식은 변수 i가 1부터 시작하여 100이 될 때까지 모든 값을 더한다는 뜻입니다. 파이썬에서는 다음과 같이 표현할 수 있습니다.

```
sum(range(1, 101)) # 결과: 5050
```

곱셈 기호 파이 Π

대문자 파이는 시그마와 구조는 비슷하지만, 합산 대신 모든 원소를 곱하는 연산을 수행합니다.

$\prod_{i=1}^{6} i$

```
from functools import reduce
def multiply(x, y): return x * y
# 1부터 6까지의 곱 (팩토리얼 6!)
reduce(multiply, range(1, 7)) # 결과: 720
```

파이프 기호 ||

수평 막대 기호는 문맥에 따라 여러 의미로 해석됩니다.

- **절댓값**: $|x|$ - 숫자의 크기
- **유클리드 노름(Norm)**: $\| \boldsymbol{v} \|$ - 벡터의 길이
- **행렬식(Determinant)**: $\det(\boldsymbol{A})$ - 행렬의 특성값

햇 기호 â

문자 위의 "햇(Hat)" 기호는 길이가 1인 단위 벡터(Unit vector)를 의미합니다.

$\hat{a}$

기하학에서는 일반 벡터를 자신의 길이로 나누어 단위 벡터로 정규화(Normalization)하여 방향만을 나타낼 때 자주 사용합니다.

소속 기호 ∈, ∉

집합론에서 ∈ 기호는 특정 원소가 집합에 포함되는지를 나타냅니다.

$A = \{3, 9, 14\}, 3 \in A$

파이썬에서는 `in` 키워드가 정확히 같은 역할을 수행합니다.

```
A = {3, 9, 14}
3 in A # True
```

주요 수 집합

- $\mathbb{R}$: 실수 집합 (Python의 `float`에 대응)
- $\mathbb{Q}$: 유리수 집합
- $\mathbb{Z}$: 정수 집합 (Python의 `int`에 대응)
- $\mathbb{N}$: 자연수 집합 (0 또는 1부터 시작하는 양의 정수)
- $\mathbb{C}$: 복소수 집합 (Python의 `complex`에 대응)

함수 표기 f

함수는 입력값을 받아 규칙에 따라 출력값을 반환하는 매핑 관계입니다. $f(x) = x^2$는 코드로 다음과 같습니다.

```
def square(x):
    return x**2
```

조각 정의 함수 (Piecewise function)

입력 조건에 따라 서로 다른 수식을 적용하는 함수입니다.

$f(x) = \begin{cases} \frac{x^2-x}{x} & \text{if } x \geqslant 1 \\ 0 & \text{otherwise} \end{cases}$

```
def f(x):
    if x ⩾ 1:
        return (x**2 - x) / x
    else:
        return 0
```

프라임 기호 ′

프라임 기호(′)는 기존 변수와 비슷하지만 다른 변수를 나타낼 때(예: 변환 후의 값) 사용하거나, 함수의 **도함수(Derivative)**를 나타낼 때 씁니다.

$f(x) = x^2$ 일 때, 도함수는 $f'(x) = 2x$ 입니다.

바닥 및 천장 함수 ⌊, ⌉

- $\lfloor x \rfloor$: 내림 (작거나 같은 최대 정수)
- $\lceil x \rceil$: 올림 (크거나 같은 최소 정수)
- $\lfloor x \rceil$: 반올림

```
import math
math.floor(4.8) # 4
math.ceil(3.1)  # 4
# Python 3의 round()는 .5일 때 가장 가까운
짝수 정수로 반올림합니다.
round(4.5)      # 4
```

논리 기호 및 화살표

- **논리적 함의**: $A \Rightarrow B$ (A가 참이면 B도 반드시 참임)
- **부등식**: <, >, ⩽, ⩾
- **논리곱(AND)과 논리합(OR)**: ∧ (AND), ∨ (OR)

논리 부정 기호 ¬, ~, !

명제의 반대(NOT)를 나타냅니다. $x \neq y \Leftrightarrow \neg(x = y)$

구간 표기 (Intervals)

실수 집합 내의 특정 범위를 나타냅니다.

- $(0, 1)$: 0과 1을 포함하지 않는 열린 구간
- $[0, 1)$: 0은 포함하고 1은 포함하지 않는 반-열린 구간
- $(0, 1]$: 0은 포함하지 않고 1은 포함하는 반-열린 구간
- $[0, 1]$: 0과 1을 모두 포함하는 닫힌 구간