R언어의 아름다움

부제: 정선우 귀 막아

Siwon Yun

KDMHS - 21wp

March 17, 2024

Contents

1. R언어의 이름 선택에 대하여

왜 Vector인가?

C언어서 'Array'라 불리우는 것은 R언어에서는 'Vector'라고 명명됩니다.

왜 Vector인가?

C언어서 'Array'라 불리우는 것은 R언어에서는 'Vector'라고 명명됩니다.

4. Elements of $\mathbb{R}^n$ (tuples of n real numbers) are vectors. $\mathbb{R}^n$ is more abstract than polynomials, and it is the concept we focus on in this book. For instance,

$$\boldsymbol{a} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \in \mathbb{R}^3 \tag{2.1}$$

is an example of a triplet of numbers. Adding two vectors $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^n$ component-wise results in another vector: $\boldsymbol{a} + \boldsymbol{b} = \boldsymbol{c} \in \mathbb{R}^n$. Moreover, multiplying $\boldsymbol{a} \in \mathbb{R}^n$ by $\lambda \in \mathbb{R}$ results in a scaled vector $\lambda\boldsymbol{a} \in \mathbb{R}^n$. Considering vectors as elements of $\mathbb{R}^n$ has an additional benefit that it loosely corresponds to arrays of real numbers on a computer. Many programming languages support array operations, which allow for convenient implementation of algorithms that involve vector operations.

Figure: Definition of Vector: 4th
Mathematics for Machine Learning@2020, M. Deisenroth, A. Faisal, and C. Ong.

다양한 분야서 Vector가 활용되는 만큼, Vector의 정의는 수학에서 보편적인 내용을 포함합니다. 예컨대, 다항식 또한 Vector의 범위에 포함됩니다. 실수 집합의 모임은 cs 분야서 흔히 사용되는 Vector의 개념입니다.

벡터 공간

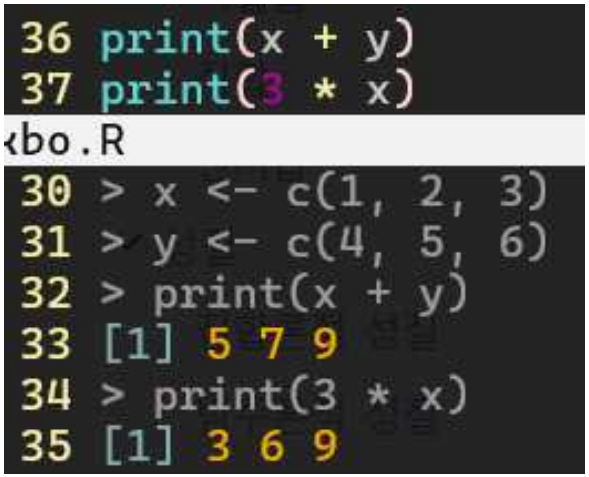
문서 토론

위키백과, 우리 모두의 백과사전.

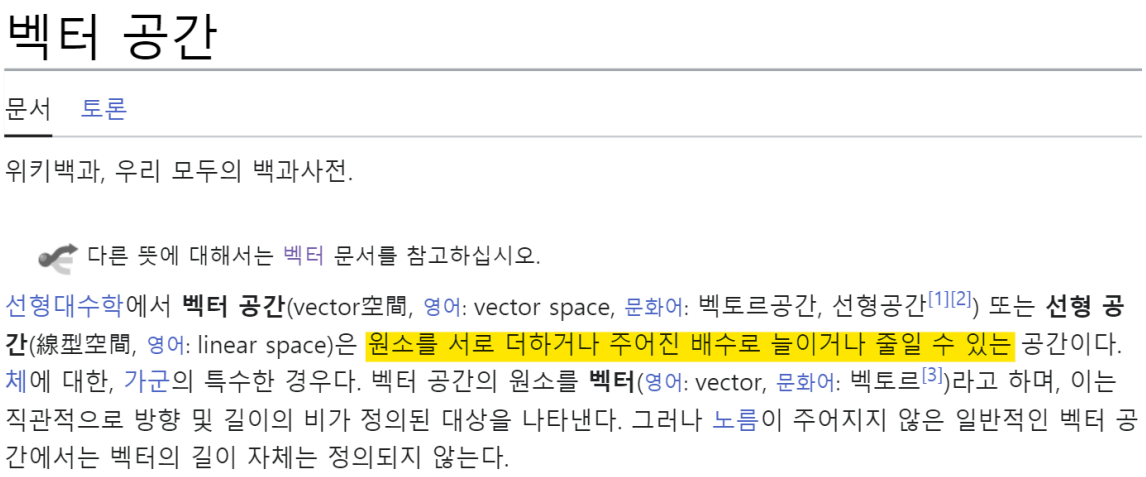
다른 뜻에 대해서는 벡터 문서를 참고하십시오.

선형대수학에서 **벡터 공간**(vector空間, 영어: vector space, 문화어: 벡토르공간, 선형공간[1][2]) 또는 **선형 공간**(線型空間, 영어: linear space)은 원소를 서로 더하거나 주어진 배수로 늘이거나 줄일 수 있는 공간이다. 체에 대한, 가군의 특수한 경우다. 벡터 공간의 원소를 **벡터**(영어: vector, 문화어: 벡토르[3])라고 하며, 이는 직관적으로 방향 및 길이의 비가 정의된 대상을 나타낸다. 그러나 노름이 주어지지 않은 일반적인 벡터 공간에서는 벡터의 길이 자체는 정의되지 않는다.

(a) Vector@Wikipedia



(b) Vector in R



벡터 공간

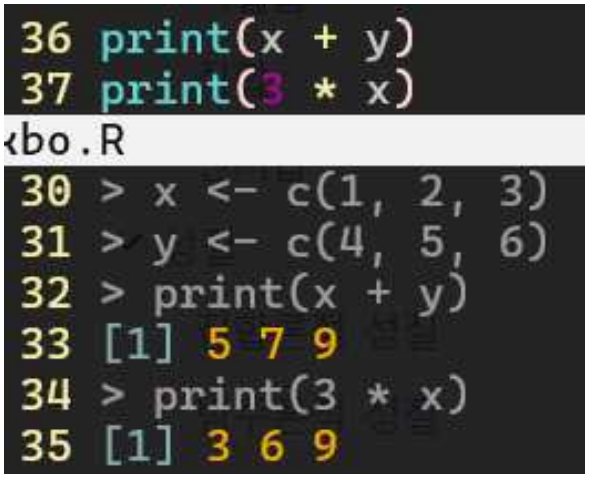
문서 토론

위키백과, 우리 모두의 백과사전.

다른 뜻에 대해서는 벡터 문서를 참고하십시오.

선형대수학에서 **벡터 공간**(vector空間, 영어: vector space, 문화어: 벡토르공간, 선형공간[1][2]) 또는 **선형 공간**(線型空間, 영어: linear space)은 원소를 서로 더하거나 주어진 배수로 늘이거나 줄일 수 있는 공간이다. 체에 대한, 가군의 특수한 경우다. 벡터 공간의 원소를 **벡터**(영어: vector, 문화어: 벡토르[3])라고 하며, 이는 직관적으로 방향 및 길이의 비가 정의된 대상을 나타낸다. 그러나 노름이 주어지지 않은 일반적인 벡터 공간에서는 벡터의 길이 자체는 정의되지 않는다.

(c) Vector@Wikipedia



```
36 print(x + y)
37 print(3 * x)
xbo.R
30 > x <- c(1, 2, 3)
31 > y <- c(4, 5, 6)
32 > print(x + y)
33 [1] 5 7 9
34 > print(3 * x)
35 [1] 3 6 9
```

(d) Vector in R

R에서의 Vector는 Wikipedia에 적혀 있는 Vector의 조건을 만족합니다. 또한 Figure 1는 많은 프로그래밍 언어에서 실수 배열을 Vector라고 부른다고 합니다.

즉, R언어에서 Vector를 Vector라 부르는 것은 당연한 일입니다.

왜 = 대신 <-를 사용하는가?

등호(=)는 우리말로 표현하려 한다면, 중의적인 표현을 가지고 있습니다.
아래는 "미적분"(김홍종 저)의 머리말에 담긴 말을 참고하여 정리했습니다.

> 수학 기호는 수학 기호로서 이해해야 하지만,...
> 등호는 우리말과 잘 어울리지 않습니다. 등호를 술어로 해석하여 '점 $O = (0, 0)$을 원점으로 한다'라는 문장을 읽는 것은 자연스럽지 아니합니다.
> 등호를 '즉'으로 이해하는 것이 더 현명할 수 있습니다: '점 O, 즉 $(0, 0)$을 원점으로 한다'.

왜 = 대신 <-를 사용하는가?

등호(=)는 우리말로 표현하려 한다면, 중의적인 표현을 가지고 있습니다. 아래는 "미적분"(김홍종 저)의 머리말에 담긴 말을 참고하여 정리했습니다.

> 수학 기호는 수학 기호로서 이해해야 하지만,...
> 등호는 우리말과 잘 어울리지 않습니다. 등호를 술어로 해석하여 '점 $O = (0, 0)$을 원점으로 한다'라는 문장을 읽는 것은 자연스럽지 아니합니다.
> 등호를 '즉'으로 이해하는 것이 더 현명할 수 있습니다: '점 O, 즉 $(0, 0)$을 원점으로 한다'.

프로그래밍 언어에서는 등호의 중의적 표현을 확실한 표현으로 바꿀 필요가 있습니다.

프로그래밍에서는 Bool type이 필수적으로 존재하여, 등호를 사용하는 것이 더욱 어렵습니다. 예컨대, 'f(a = 1)'은 다음과 같이 3가지 방식으로 해석될 수 있습니다.

(a) a 이름을 가진 argument에 1을 대입한다.

(b) a와 1을 논리 연산하고 argument로 전달한다.

(c) 변수 a에 1을 할당하고, a를 argument로 전달한다.

R언어 및 C언어를 포함한 대부분의 언어는 (b)를 비교하기 위해 '=='연산자를 사용합니다. C언어는 argument에 기본값 설정을 금하여 (a) 상황을 피합니다.

프로그래밍에서는 Bool type이 필수적으로 존재하여, 등호를 사용하는 것이 더욱 어렵습니다. 예컨대, 'f(a = 1)'은 다음과 같이 3가지 방식으로 해석될 수 있습니다.

(a) a 이름을 가진 argument에 1을 대입한다.

(b) a와 1을 논리 연산하고 argument로 전달한다.

(c) 변수 a에 1을 할당하고, a를 argument로 전달한다.

R언어 및 C언어를 포함한 대부분의 언어는 (b)를 비교하기 위해 '==' 연산자를 사용합니다. C언어는 argument에 기본값 설정을 금하여 (a) 상황을 피합니다. R언어는 '←'와 '='를 구분하여 (a) 상황과 (c) 상황을 구분합니다.

```
143 > f <- function(a = "default value", b) {
144 +   print(a)
145 + }
146 > f(b = c(1, 2, 3))
147 [1] "default value"
148 > f(b <- c(1, 2, 3))
149 [1] 1 2 3
```

Figure: R언어의 argument 전달 방식

R언어에서 '←'는 할당, '='는 argument 대입으로 생각하기를 제안합니다.

왜 1.9.0 이전에는 변수 이름에 '_'를 사용 못할까?

Gemini:

R 1.9 이전 버전에서는 변수 이름에 _ (언더스코어)를 사용할 수 없었던 이유는 다음과 같습니다.
1. 역사적 이유
R 언어는 1990년대 초에 개발되었으며 당시에는 _가 특별한 의미를 가지고 있었습니다. 예를 들어, _는 내부 함수 또는 변수를 나타내는 데 사용되었습니다. 따라서 변수 이름에 _를 사용하면 코드의 의미가 모호해지고 오류가 발생할 수 있었습니다.

왜 1.9.0 이전에는 변수 이름에 '_'를 사용 못할까?

Gemini:

> R 1.9 이전 버전에서는 변수 이름에 _ (언더스코어)를 사용할 수 없었던 이유는 다음과 같습니다.
> 1. 역사적 이유
> R 언어는 1990년대 초에 개발되었으며 당시에는 _가 특별한 의미를 가지고 있었습니다. 예를 들어, _는 내부 함수 또는 변수를 나타내는 데 사용되었습니다. 따라서 변수 이름에 _를 사용하면 코드의 의미가 모호해지고 오류가 발생할 수 있었습니다.

LaTeX에서 'a_3'는 'a_3'와 같이 표현됩니다.
즉, '_'는 '내부의 존재를 가져옴'과 같이 해석 가능합니다. R언어 또한 '_'를 이와 같이 해석하여 1.9.0 이전에는 변수 이름에 '_'를 사용할 수 없었을 것으로 추측됩니다.

2. 왜 R언어의 vector는 1부터 시작하는가?

1부터 vs 0부터

programming language에서 0부터 시작하는가, 1부터 시작하는가는 거대한 논쟁거리입니다. 인기있는 프로그래밍 언어인 C의 자료 구조인 Array 에서는 0부터 시작하지만, 1부터 시작하는 언어도 다수 존재합니다. 아래 대표적인 프로그래밍 언어를 시작 숫자로 구별하였습니다.

0 기반 인덱스 언어	1 기반 인덱스 언어
C, Python, Java, Rust	R, Julia, MATLAB, Roblox

1부터 vs 0부터

programming language에서 0부터 시작하는가, 1부터 시작하는가는 거대한 논쟁거리입니다. 인기있는 프로그래밍 언어인 C의 자료 구조인 Array 에서는 0부터 시작하지만, 1부터 시작하는 언어도 다수 존재합니다. 아래 대표적인 프로그래밍 언어를 시작 숫자로 구별하였습니다.

0 기반 인덱스 언어	1 기반 인덱스 언어
C, Python, Java, Rust	R, Julia, MATLAB, Roblox

비교적 수학/계산과학 분야서 많이 사용되는 언어는 1기반 인덱스를 사용하는 경향이 있습니다.

'1부터 vs 0부터'는 아직 논쟁거리

'1부터 vs 0부터'는 아직 프로그래밍 언어 분야의 논쟁거리입니다.
책 '줄리아 머신러닝, 딥러닝, 강화학습'(김태훈 저)에서는 "0 기반 인덱스는 첫 번째 원소로부터의 거리 개념이고, 1 기반 인덱스는 첫 번째, 두 번째 등 자연스러운 카운팅 개념이다."라고 설명합니다.

'1부터 vs 0부터'는 아직 논쟁거리

'1부터 vs 0부터'는 아직 프로그래밍 언어 분야의 논쟁거리입니다.
책 '줄리아 머신러닝, 딥러닝, 강화학습'(김태훈 저)에서는 "0 기반 인덱스는 첫 번째 원소로부터의 거리 개념이고, 1 기반 인덱스는 첫 번째, 두 번째 등 자연스러운 카운팅 개념이다."라고 설명합니다.

해당 논쟁은 수학에서 또한 논쟁거리입니다.
우리는 수열을 배우며 '$(a_1, a_2, a_3, \dots)$'과 같은 표현을 사용합니다. 즉, 우리가 배운 수열은 1부터 시작합니다. 비단 "미적분"(김홍종 저)에 따르면 "정의역이 $\mathbb{N}_0$인 것이 더 자연스럽다. 오늘날은 수의 시작, 즉 처음 수를 0으로 보는 견해가 지배적이다."라고 합니다.

'1부터 vs 0부터'는 아직 논쟁거리

'1부터 vs 0부터'는 아직 프로그래밍 언어 분야의 논쟁거리입니다.
책 '줄리아 머신러닝, 딥러닝, 강화학습'(김태훈 저)에서는 "0 기반 인덱스는 첫 번째 원소로부터의 거리 개념이고, 1 기반 인덱스는 첫 번째, 두 번째 등 자연스러운 카운팅 개념이다."라고 설명합니다.

해당 논쟁은 수학에서 또한 논쟁거리입니다.
우리는 수열을 배우며 '$(a_1, a_2, a_3, \dots)$'과 같은 표현을 사용합니다. 즉, 우리가 배운 수열은 1부터 시작합니다. 비단 "미적분"(김홍종 저)에 따르면 "정의역이 $\mathbb{N}_0$인 것이 더 자연스럽다. 오늘날은 수의 시작, 즉 처음 수를 0으로 보는 견해가 지배적이다."라고 합니다.

이와 같이 인덱스의 시작을 0으로 또는 1로 시작하는 것은 아직 논쟁거리입니다.

비단 1 기반 인덱스와 0 기반 인덱스는 효율적인 코드를 작성하는데 차이가 발생합니다. 메모리에 '1, 2, 3, 4, 5, 6'과 같이 저장되어 있을 때, 3×2 행렬을 표기하면, 0 기반 인덱스는 행을 우선적으로, 1 기반 인덱스는 열을 우선적으로 표기합니다.

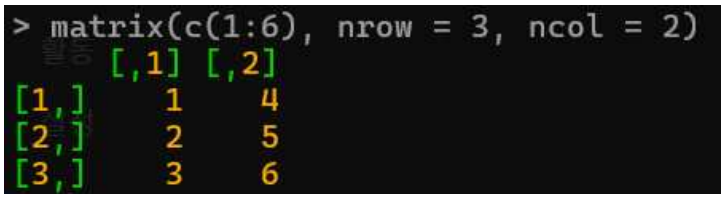
```
> matrix(c(1:6), nrow = 3, ncol = 2)
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

(a) Matrix in R

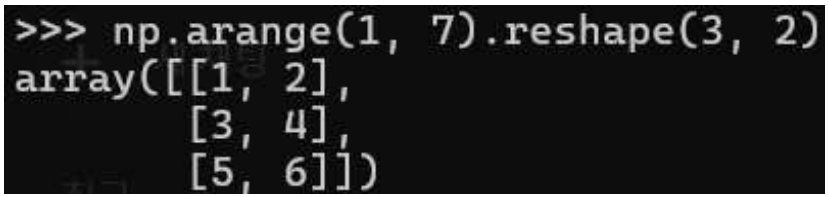
```
>>> np.arange(1, 7).reshape(3, 2)
array([[1, 2],
       [3, 4],
       [5, 6]])
```

(b) Matrix in Python

비단 1 기반 인덱스와 0 기반 인덱스는 효율적인 코드를 작성하는데 차이가 발생합니다. 메모리에 '1, 2, 3, 4, 5, 6'과 같이 저장되어 있을 때, 3×2 행렬을 표기하면, 0 기반 인덱스는 행을 우선적으로, 1 기반 인덱스는 열을 우선적으로 표기합니다.



(c) Matrix in R



(d) Matrix in Python

즉, 인덱스 시작 값을 생각하여 행렬 연산시 메모리를 방문하는 순서를 구성하면 보다 효율적인 코드 작성이 가능합니다.
(하지만 R에서는 행렬에 대하여 element-wise 연산이 필요하다면 vector 화를 합시다...)