개발자에게 필요 없는 수학은 없다

Math for Programmers

프로그래머를 위한 수학

파이썬으로 하는 3D 그래픽스, 시뮬레이션, 머신러닝

* 연습문제 정답 및 풀이 이용안내 *

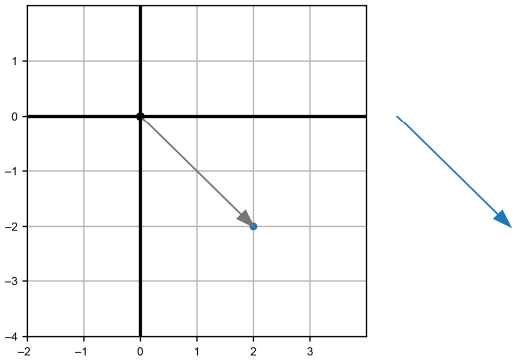
2.1.3 연습문제 정답 및 풀이

연습문제 | 2.1

$(-1,-4)$

연습문제 | 2.2

점 $(2,-2)$를 평면 상의 점과 화살표로 표현해보면 다음과 같다.



연습문제 | 2.3

공룡의 외양을 이루는 점 벡터 전체는 다음과 같다.

```
dino_vectors = [(6,4), (3,1), (1,2), (-1,5), (-2,5), (-3,4), (-4,4),
    (-5,3), (-5,2), (-2,2), (-5,1), (-4,0), (-2,1), (-1,0), (0,-3),
    (-1,-4), (1,-4), (2,-3), (1,-2), (3,-1), (5,1)
]
```

연습문제 | 2.4

```
draw(
    Points(*dino_vectors),
    Polygon(*dino_vectors)
)
```

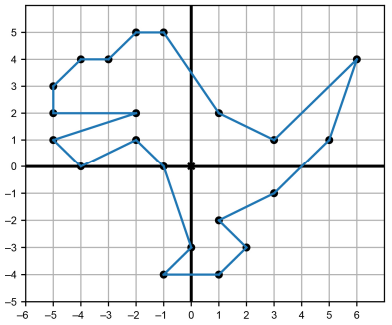


그림 다각형으로 그려진 공룡

연습문제 | 2.5

주어진 순서쌍은 -10에서 10까지의 정수 x에 대해서만 나타낸 함수 $y=x^2$의 그래프를 그린다.

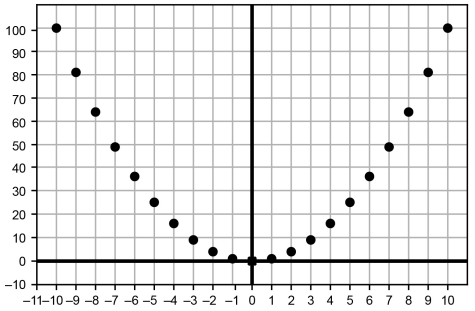


그림 함수 $y=x^2$ 그래프 상의 점

이 그래프를 생성하고자 `draw` 함수에 2개의 키워드 인자를 사용했다. `grid` 키워드 인자에 $(1,10)$을 주면 1단위당 수직 격자선을 그리며, 10단위당 수평 격자선을 그린다. `nice_aspect_ratio` 키워드 인자에 `False`를 설정하면 이 그래프는 x축과 y축 척도를 동일하게 설정할 필요가 없음을 알려준다.

```
draw(
    Points(*[(x,x**2) for x in range(-10,11)]),
    grid=(1,10),
    nice_aspect_ratio=False
)
```

2.2.4 연습문제 정답 및 풀이

연습문제 | 2.6

$\mathbf{u}=(-2,0)$, $\mathbf{v}=(1.5,1.5)$, $\mathbf{w}=(4,1)$이므로 결과는 다음과 같다.

$\mathbf{u}+\mathbf{v}=(-0.5,1.5)$
$\mathbf{v}+\mathbf{w}=(5.5,2.5)$
$\mathbf{u}+\mathbf{w}=(2,1)$
$\mathbf{u}+\mathbf{v}+\mathbf{w}=(3.5,2.5)$

연습문제 | 2.7 미니 프로젝트

```
def add(*vectors):
    return (sum([v[0] for v in vectors]), sum([v[1] for v in vectors]))
```

연습문제 | 2.8

```
def translate(translation, vectors):
    return [add(translation, v) for v in vectors]
```

연습문제 | 2.9 미니 프로젝트

임의의 두 벡터 $\mathbf{u}$와 $\mathbf{v}$를 실수 a,b,c,d에 대하여 $\mathbf{u}=(a,b)$, $\mathbf{v}=(c,d)$라고 표기할 수 있다. 이때 벡터합은 $\mathbf{u}+\mathbf{v}=(a+c,b+d)$이다. 한편, $\mathbf{v}+\mathbf{u}=(c+a,d+b)$이다. 실수의 덧셈에서는 더하는 순서가 중요하지 않기 때문에 두 순서쌍은 같다고 할 수 있다. 시각적으로 두 벡터를 임의로 선택해 삼각형법을 적용해보자. 순서대로 벡터를 더하면 동일한 합벡터가 된다는 것을 다시 한 번 확인할 수 있다.

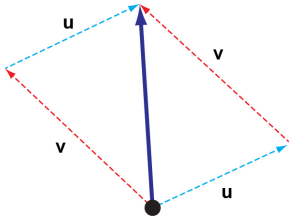
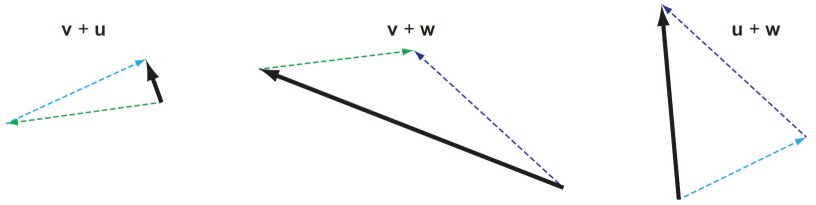


그림 삼각형법으로 각 순서에 따라 더해도 같은 합벡터가 도출된다.

$\mathbf{u}+\mathbf{v}$ 또는 $\mathbf{v}+\mathbf{u}$ 순으로 계산(점선)해도 결과 벡터(실선)가 같으므로 순서는 중요하지 않다. 기하학적인 관점에서 삼각형법을 적용하면 $\mathbf{u}$와 $\mathbf{v}$는 순서대로 평행사변형을 이루고 이때 벡터합의 길이는 평행사변형의 대각선 길이와 같다.

연습문제 | 2.10

가능한 조합을 모두 삼각형법으로 더해서 길이를 측정할 수 있다.



결과를 보면 $\mathbf{v}+\mathbf{u}$가 가장 짧은 벡터이다. $\mathbf{u}$와 $\mathbf{v}$가 거의 반대 방향을 가리키고 있어서 서로를 거의 상쇄하기 때문이다. 가장 긴 벡터는 $\mathbf{v}+\mathbf{w}$이다.

연습문제 | 2.11 미니 프로젝트

시행착오를 거쳐 보면 수직 방향과 수평 방향 모두에서 공룡을 겹치지 않고 적절한 경계를 갖도록 평행이동할 수 있다. 그림을 명확하게 그리기 위해 격자선, 축, 원점, 공룡의 꼭짓점은 제거했다. 예시 코드는 다음과 같다.

```
def hundred_dinos():
    translations = [(12*x,10*y)
                    for x in range(-5,5)
                    for y in range(-5,5)]
    dinos = [Polygon(*translate(t, dino_vectors),color=blue)
                for t in translations]
    draw(*dinos, grid=None, axes=None, origin=None)

hundred_dinos()
```

결과는 다음과 같다.

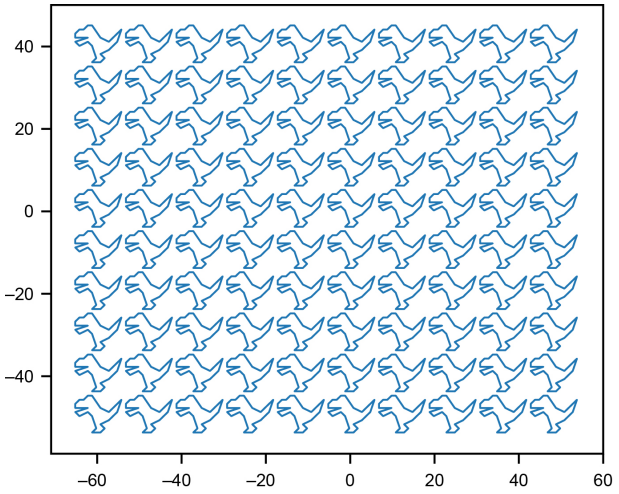


그림 공룡 100마리(죽기 살기로 뛰어야 할 듯!)

연습문제 | 2.12

벡터합 $(3,-2)+(1,1)+(-2,-2)$의 결과는 $(2,-3)$이다. x성분 $(2,0)$은 방향이 오른쪽이고 길이는 2단위인 벡터이며, y성분 $(0,-3)$은 방향이 아래쪽이고 길이는 3단위인 벡터이다. 따라서 y성분이 길다.

연습문제 | 2.13

$(-6,-6)$의 성분은 $(-6,0)$과 $(0,-6)$이며, 길이는 모두 6이다. $(-6,-6)$의 길이는 6^2+6^2의 양의 제곱근으로, 근사적으로 8.485이다.

$(5,-12)$의 성분은 $(5,0)$과 $(0,-12)$이며, 길이는 각각 5와 12이다. $(5,-12)$의 길이는 $5^2+12^2=25+144=169$의 양의 제곱근으로, 13이다.

연습문제 | 2.14

x성분 $(1,0)$의 길이가 1이고 벡터 $\mathbf{v}$의 길이가 6이므로, y성분의 길이를 b라고 할 때 식 $1^2+b^2=6^2=36$이 성립해야 한다. $b^2=35$이므로 y성분의 길이는 근사적으로 5.916이다. 하지만 y성분의 길이를 알았다는 사실만으로 y성분의 방향을 알 수가 없다. 이를 고려하면 벡터 $\mathbf{v}$는 $(1,5.916)$ 또는 $(1,-5.916)$ 중 하나이다.

연습문제 | 2.15

```
>>> max(dino_vectors, key=length)
(6, 4)
```

연습문제 | 2.16

$N = (\sqrt{2}, \sqrt{3})$의 값은 근사적으로 $(1.4142135623730951, 1.7320508075688772)$이다. π를 각 좌표에 곱하면 다음을 얻는다.

$$(4.442882938158366, 5.441398092702653)$$

다음 그림에서 볼 수 있듯 스칼라곱을 계산한 벡터는 원래 벡터보다 길다.

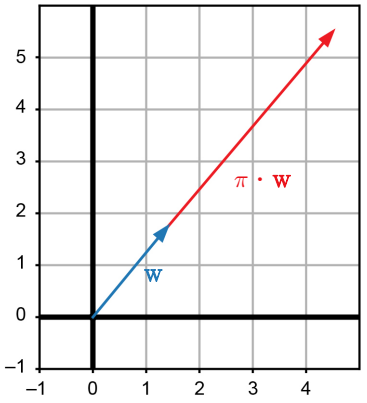


그림 (더 짧은) 원래 벡터와 (더 긴) 스칼라곱을 계산한 벡터

연습문제 | 2.17

```
def scale(scalar,v):
    return (scalar * v[0], scalar * v[1])
```

연습문제 | 2.18 미니 프로젝트

벡터 (a,b)의 길이를 $|(a,b)|$로 나타내자. 문제에서 주어진 바에 따르면 다음이 성립한다.

$$c = \sqrt{a^2+b^2} = |(a,b)|$$

이를 통해 (sa,sb)의 길이를 계산할 수 있다.

$$\begin{aligned}|(sa,sb)| &= \sqrt{(sa)^2+(sb)^2} = \sqrt{s^2a^2+s^2b^2} \\ &= \sqrt{s^2\cdot(a^2+b^2)} = |s|\cdot\sqrt{a^2+b^2} \\ &= |s|\cdot c\end{aligned}$$

s가 음수가 아닌 한, s는 그 절댓값과 같다. 즉 $s=|s|$이다. 따라서 스칼라곱을 한 벡터의 길이는 우리가 보이고자 한 sc이다.

연습문제 | 2.19 미니 프로젝트

$r=0$이라고 하면, 모든 벡터 $r\cdot\mathbf{u}+s\cdot\mathbf{v}$는 $(-1,-1)$에서 $(1,1)$ 사이의 선분 위에 위치해 있다. r이 0이 아니면, 해당 선분에서 $(-1,1)$ 또는 $-(-1,1)$ 방향으로 3단위만큼 벗어날 수 있다. 가능한 점들이 이루는 영역은 $(2,4)$, $(4,2)$, $(2,-4)$, $(4,-2)$를 꼭짓점으로 하는 평행사변형이다. 이를 검증하기 위해 r과 s의 범위 안에서 점들을 랜덤하게 테스트해볼 수 있다.

```
from random import uniform
u = (-1,1)
v = (1,1)
def random_r():
    return uniform(-3,3)
def random_s():
    return uniform(-1,1)

possibilities = [add(scale(random_r(), u), scale(random_s(), v))
                 for i in range(0,500)]
draw(
    Points(*possibilities)
)
```

이 코드를 실행하면 다음과 같이 주어진 제약조건 아래에서 $r\cdot\mathbf{u}+s\cdot\mathbf{v}$가 도달하는 점들을 보여주는 그림을 얻는다.

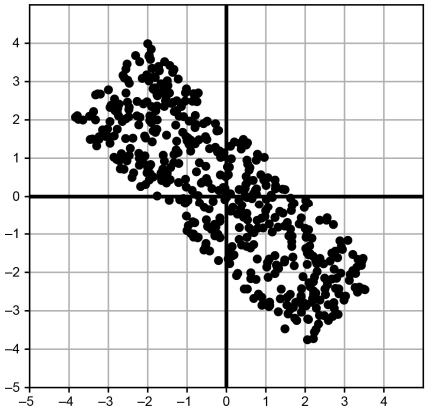


그림 주어진 제약조건 아래에서 $r \cdot \mathbf{u} + s \cdot \mathbf{v}$가 될 수 있는 점들의 위치

연습문제 | 2.20

(a,b)에 대한 역벡터의 좌표는 $(-a,-b)$이지만, 길이는 바뀌지 않는다.

$$\sqrt{(-a)^2+(-b)^2} = \sqrt{(-a)\cdot(-a)+(-b)\cdot(-b)} = \sqrt{a^2+b^2}$$

따라서 벡터 $(-a,-b)$는 (a,b)와 길이가 같다.

연습문제 | 2.21

벡터 $\mathbf{v}_3$와 $\mathbf{v}_7$이 서로의 역벡터로 짝을 이룬다.

연습문제 | 2.22

2차원 벡터 $\mathbf{u}$의 좌표가 (a,b)라고 하자. 벡터 $\mathbf{u}$의 역벡터의 좌표는 $(-a,-b)$이다. 따라서 다음이 성립한다.

$$\mathbf{u} + (-\mathbf{u}) = (a,b) + (-a,-b) = (a-a,b-b) = (0,0)$$

따라서 답은 $(0,0)$이다. 기하학적으로 벡터를 따라갔다가 역벡터를 따라가면 원점 $(0,0)$으로 돌아옴을 의미한다.

연습문제 | 2.23

$\mathbf{v} - \mathbf{w} = (-2.5, 0.5)$
$\mathbf{u} - \mathbf{v} = (-3.5, -1.5)$
$\mathbf{w} - \mathbf{v} = (2.5, -0.5)$

연습문제 | 2.24

```
def subtract(v1,v2):
    return (v1[0] - v2[0], v1[1] - v2[1])
```

연습문제 | 2.25

거리란 두 입력 벡터의 차에 대한 길이이다.

```
def distance(v1,v2):
    return length(subtract(v1,v2))
```

둘레를 구하려면 리스트 상에 이웃한 벡터끼리의 거리를 더해야 하는데, 이때 마지막 벡터와 첫 벡터 간의 거리도 포함해야 한다.

```
def perimeter(vectors):
    distances = [distance(vectors[i], vectors[(i+1)%len(vectors)])
                 for i in range(0,len(vectors))]
    return sum(distances)
```

적합성 확인(sanity check) 차원에서 각 변의 길이가 1인 정사각형을 사용해볼 수 있다.

```
>>> perimeter([(1,0),(1,1),(0,1),(0,0)])
4.0
```

이를 통해 공룡의 둘레를 계산할 수 있다.

```
>>> perimeter(dino_vectors)
44.77115093694563
```

연습문제 | 2.26 미니 프로젝트

n이 1을 기준으로 13단위 이내의 거리에, m이 -1을 기준으로 13단위 이내의 거리에 있을 때 정수로 이루어진 가능한 순서쌍 (n,m)을 탐색하면 충분하다.

```
for n in range(-12,15):
    for m in range(-14, 13):
        if distance((n,m), (1,-1)) == 13 and n > m > 0:
            print((n,m))
```

v의 찾아본 결과 중 하나는 $(13,4)$이다. **u** $=(1,-1)$을 기준으로 오른쪽으로 12단위, 위쪽으로 5단위에 있으며, 따라서 **u**에서 **v**로의 변위는 $(12,5)$이다.

2.3.4 연습문제 정답 및 풀이

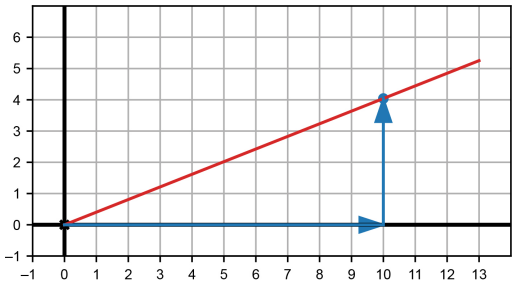
연습문제 | 2.27

```
>>> length((-1.34,2.68))
2.9963310898497184
```

충분히 3에 가깝다!

연습문제 | 2.28

그림과 같이 이 선은 점 (10,4)의 근처를 지나므로 $4/10 = 0.4$는 $\tan(22°)$의 좋은 근삿값이다.



연습문제 | 2.29

각 37°의 사인값은 대략 3/5이다. 이 각을 따라 거리가 5단위 증가할 때마다 위쪽으로 3단위씩 증가한다는 뜻이다. 따라서 거리가 15단위라는 것은 수직 성분이 $3/5 \cdot 15 = 9$임을 알 수 있다.

각 37°의 코사인값은 대략 4/5이다. 이 각을 따라 거리가 5단위 증가할 때마다 오른쪽으로 4단위씩 증가한다는 뜻이다. 따라서 수평 성분이 $4/5 \cdot 15 = 12$임을 알 수 있다. 요약하자면 극좌표 (15,37°)는 근사적으로 데카르트 좌표 (12,9)에 대응한다.

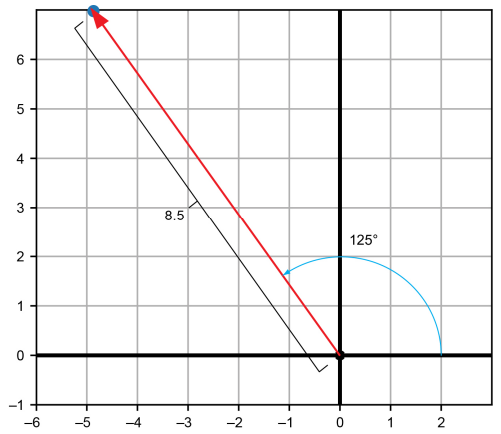
연습문제 | 2.30

최종 좌표의 각 성분은 다음과 같다.

$$x = r \cdot \cos(\theta) = 8.5 \cdot -0.574 = -4.879$$

$$y = r \cdot \sin(\theta) = 8.5 \cdot 0.819 = 6.962$$

다음 그림은 최종 좌표가 $(-4.879, 6.962)$임을 보여준다. 각 125°와 이동한 경로도 확인할 수 있다.



연습문제 | 2.31

0°에서 수직 거리는 0이므로 $\sin(0°) = 0$이다. 거리가 1단위씩 늘어나면 그만큼 수평 거리 단위가 늘어난다. 따라서 $\cos(0°) = 1$이다.

90°(반시계방향으로 사분원만큼 회전)에서는 거리가 1단위 늘어나면 그만큼 양의 수직 단위가 늘어난다. 따라서 $\sin(90°) = 1$이지만 $\cos(90°) = 0$이다.

마지막으로 180°에서는 거리가 1단위 늘어나면 그만큼 x축에서 음의 단위로 늘어난다. 따라서 $\cos(180°) = -1$이고 $\sin(180°) = 0$이다.

연습문제 | 2.32

각 변의 길이는 실제로 피타고라스 정리를 만족한다. 피타고라스 정리에 각 변의 길이를 대입하면 다음과 같다.

$$\sqrt{\left(\frac{1}{2}\right)^2+\left(\frac{\sqrt{3}}{2}\right)^2}=\sqrt{\frac{1}{4}+\frac{3}{4}}=\sqrt{\frac{4}{4}}=1$$

각 삼각함수의 값은 적절히 선택된 두 길이의 비율로 구해진다. 각 정의에 따라 사인, 코사인, 탄젠트값을 계산하면 다음과 같다.

$$\sin(30°)=\frac{\left(\frac{1}{2}\right)}{1}=0.500,\ \cos(30°)=\frac{\left(\frac{\sqrt{3}}{2}\right)}{1}\approx 0.866,\ \tan(30°)=\frac{\left(\frac{1}{2}\right)}{\left(\frac{\sqrt{3}}{2}\right)}\approx 0.577$$

연습문제 | 2.33

[연습문제 2.32]에서 등장한 삼각형을 회전하고 대칭이동하더라도 변의 길이나 각에는 변화가 없다.

각 변의 비율을 정의대로 계산해 60°에서의 삼각함수 값을 구하면 다음과 같다.

$$\sin(60°)=\frac{\left(\frac{\sqrt{3}}{2}\right)}{1}\approx 0.866$$

$$\cos(60°)=\frac{\left(\frac{1}{2}\right)}{1}=0.500$$

$$\tan(60°)=\frac{\left(\frac{\sqrt{3}}{2}\right)}{\left(\frac{1}{2}\right)}\approx 1.732$$

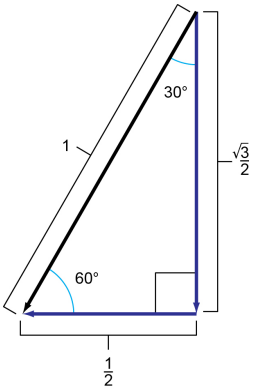
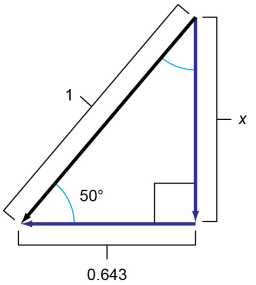


그림 [연습문제 2.32]에서 삼각형을 회전한 그림

연습문제 | 2.34

50°에 대한 코사인값이 0.643이므로, 오른쪽과 같이 삼각형을 그릴 수 있다.



코사인값 $0.643/1 = 0.643$을 이 삼각형의 두 변 길이로부터 구했다고 생각할 수 있다. 이제 미지수인 변의 길이 x를 구하기 위해 피타고라스 정리를 사용해보자.

$$\sqrt{0.643^2 + x^2} = 1$$
$$0.643^2 + x^2 = 1$$
$$0.413 + x^2 = 1$$
$$x^2 = 0.587$$
$$x = 0.766$$

따라서 $\sin(50°) = 0.766/1 = 0.766$이며, $\tan(50°) = 0.766/0.643 = 1.192$이다.

연습문제 | 2.35

$116.57° \cdot (1\text{라디안}/57.296°) = 2.035\text{라디안}$

```
>>> from math import tan
>>> tan(2.035)
-1.9972227673316139
```

연습문제 | 2.36

원을 그리며 한 바퀴 돌면 2π라디안이므로, 각 $\pi/6$는 원의 1/12에 해당한다. 원형 피자를 12조각으로 자르는 상상을 하면서 x축의 양의 방향에서부터 각의 크기를 늘리면 각 $10\pi/6$은 1판(1바퀴)이 되기엔 2조각이 모자람을 알 수 있다. 이는 각 $10\pi/6$가 원점을 기준으로 오른쪽 아래에 있음을 의미한다. 해당 방향으로 나아가면 수평 변위가 양수이고 수직 변위가 음수이기 때문에 코사인값은 양수일 것이며 사인값은 음수일 것이다.

```
>>> from math import pi, cos, sin
>>> sin(10*pi/6)
-0.8660254037844386
>>> cos(10*pi/6)
0.5000000000000001
```

연습문제 | 2.37

극좌표 점들의 리스트를 포함한 코드는 다음과 같다.

```
polar_coords = [(cos(x*pi/100.0), 2*pi*x/1000.0) for x in range(0,1001)]
vectors = [to_cartesian(p) for p in polar_coords]
draw(Polygon(*vectors, color=green))
```

이 결과는 5장의 꽃잎으로 이루어진 꽃이다.

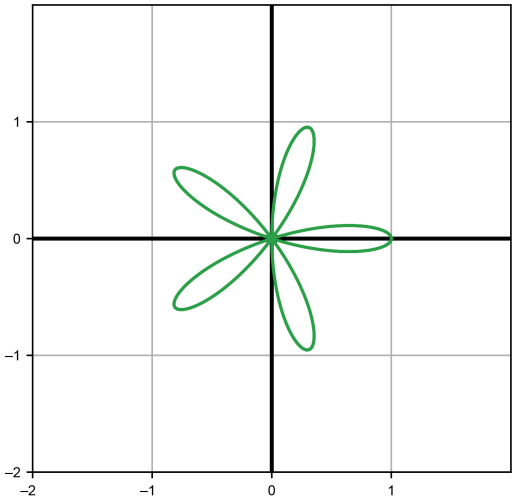


그림 1,001개의 연결된 점을 그린 꽃잎 모양

연습문제 | 2.38

탄젠트값이 $-3/2 = -1.5$에 근접하는 각을 찾기 위해, 파이썬을 사용해 $\pi/2$와 π 사이에서 정답을 이진 탐색(binary search)하듯이 구하는 예시를 살펴보자.

```
>>> from math import tan, pi
>>> pi, pi/2
(3.141592653589793, 1.5707963267948966)
>>> tan(1.8)
-4.286261674628062
>>> tan(2.5)
-0.7470222972386603
>>> tan(2.2)
-1.3738230567687946
```

```
>>> tan(2.1)
-1.7098465429045073
>>> tan(2.15)
-1.5289797578045665
>>> tan(2.16)
-1.496103541616277
>>> tan(2.155)
-1.5124173422757465
>>> tan(2.156)
-1.5091348993879299
>>> tan(2.157)
-1.5058623488727219
>>> tan(2.158)
-1.5025996395625054
>>> tan(2.159)
-1.4993467206361923
```

해당 값은 2.158과 2.159 사이에 있다.

연습문제 | 2.39

탄젠트값이 $-3/2$인 다른 점으로 $(3,-2)$가 있다. 파이썬의 `math.atan` 함수로 이 점에 대한 각을 구할 수 있다.

```
>>> from math import atan
>>> atan(-3/2)
-0.982793723247329
```

이 값은 시계방향으로 1/4바퀴보다 약간 적게 회전한 것이다.

연습문제 | 2.40

$(1,1)$는 극좌표로 표현하면 $(\sqrt{2},\pi/4)$이다.

$(1,-1)$은 극좌표로 표현하면 $(\sqrt{2},-\pi/4)$이다.

2.4.2 연습문제 정답 및 풀이

연습문제 | 2.42

```
def rotate(angle, vectors):
    polars = [to_polar(v) for v in vectors]
    return [to_cartesian((l, a+angle)) for l,a in polars]
```

연습문제 | 2.43

```
def regular_polygon(n):
    return [to_cartesian((1, 2*pi*k/n)) for k in range(0,n)]
```

연습문제 | 2.44

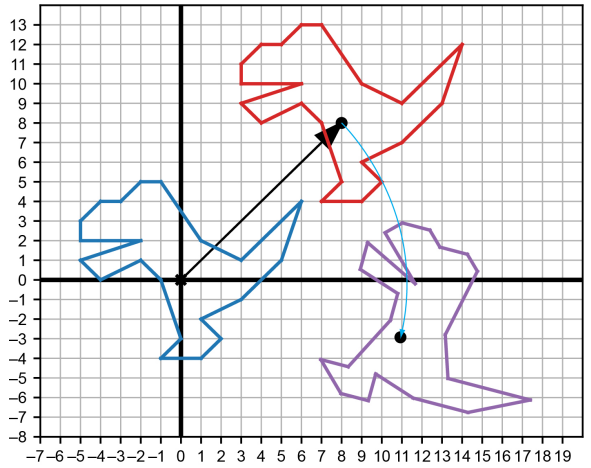


그림 평행이동 뒤 회전한 공룡

결과는 같지 **않다**. 일반적으로 회전이동과 평행이동의 순서를 바꾸면 결과도 다르다.

3.1.3 연습문제 정답 및 풀이

연습문제 | 3.1

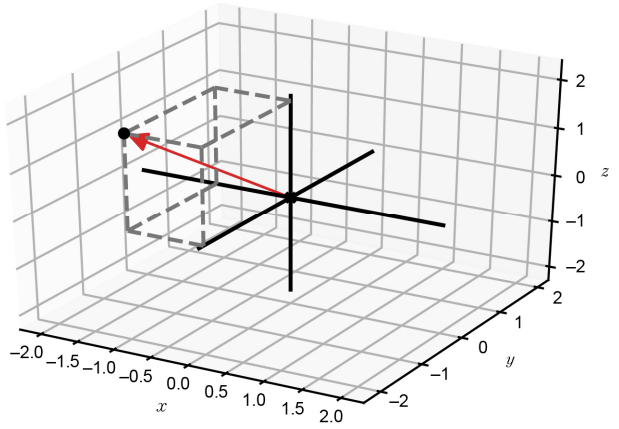


그림 벡터 $(-1,-2,2)$와 벡터를 3차원처럼 보이게 하는 박스

연습문제 | 3.2 미니 프로젝트

정육면체에 있는 8개의 꼭짓점과 12개의 모서리를 나열하는 게 엄청 지루한 일은 아니지만, 여기서는 편의상 리스트 컴프리헨션을 사용해보자. 꼭짓점은 가능한 두 값의 리스트인 `[1, -1]`로 x, y, z의 범위를 정한 뒤 결과를 모아 구했다. 모서리는 각 좌표축 방향을 따라 모서리를 네 개씩 묶어서 3개의 집합을 만들었다. 예를 들어 양끝 점의 y좌표와 z좌표는 동일하지만 $x=-1$부터 $x=1$까지 이어지는 선분은 4개 존재한다.

```
pm1 = [1,-1]
vertices = [(x,y,z) for x in pm1 for y in pm1 for z in pm1]
edges = [((-1,y,z),(1,y,z)) for y in pm1 for z in pm1] +\
            [((x,-1,z),(x,1,z)) for x in pm1 for z in pm1] +\
            [((x,y,-1),(x,y,1)) for x in pm1 for y in pm1]
draw3d(
    Points3D(*vertices,color=blue),
    *[Segment3D(*edge) for edge in edges]
)
```

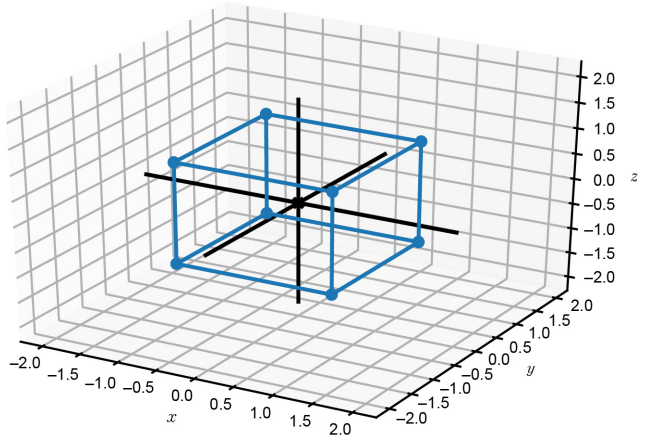


그림 좌표가 $+1$ 또는 -1인 모든 꼭짓점으로 이루어진 정육면체

3.2.6 연습문제 정답 및 풀이

연습문제 | 3.3

앞에서 만든 add 함수를 사용해 벡터합을 구할 수 있다.

```
>>> add((4,0,3),(-1,0,1))
(3, 0, 4)
```

두 벡터를 삼각형법으로 그리려면 원점에서 각 점으로 가는 화살표를 그리고, 각 점에서 벡터합 $(3,0,4)$를 그리면 된다. 2차원에서의 Arrow 객체와 마찬가지로 Arrow3D는 화살표 머리의 위치를 나타내는 벡터를 취하며, 꼬리가 원점이 아니라면 선택적으로 꼬리의 위치를 나타내는 벡터를 취한다.

```
draw3d(
    Arrow3D((4,0,3),color=red),
    Arrow3D((-1,0,1),color=blue),
    Arrow3D((3,0,4),(4,0,3),color=blue),
    Arrow3D((-1,0,1),(3,0,4),color=red),
    Arrow3D((3,0,4),color=purple)
)
```

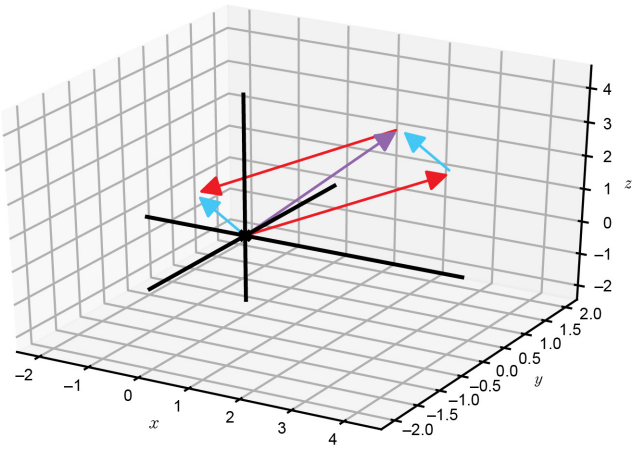


그림 삼각형법으로 구한 벡터합 $(4,0,3)+(-1,0,1)=(-1,0,1)+(4,0,3)=(3,0,4)$

연습문제 | 3.4

첫 번째 zip은 길이가 5이다. 두 입력 벡터 각각이 5개의 좌푯값을 가지므로, zip(*vectors1)은 각각 2개의 원소를 갖는 5개의 튜플을 포함한다. 마찬가지로 zip(*vectors2)는 길이가 2이다. zip(*vectors2)의 두 원소는 각각 모든 x성분을 포함한 튜플과 y성분을 포함한 튜플이다.

연습문제 | 3.5 미니 프로젝트

삼각형법을 사용해 이 벡터들이 꼬리에 꼬리를 물도록 그려보면 나선(helix) 모양을 이룬다.

```
from math import sin, cos, pi
vs = [(sin(pi*t/6), cos(pi*t/6), 1.0/3) for t in range(0,24)]

running_sum = (0,0,0)
arrows = []
for v in vs:
    next_sum = add(running_sum, v)
    arrows.append(Arrow3D(next_sum, running_sum))
    running_sum = next_sum
print(running_sum)
draw3d(*arrows)
```

중간합을 저장하는 변수는 삼각형법 덧셈의 시작점인 (0, 0, 0)에서 시작한다.

직전 벡터가 이번 벡터의 꼬리를 물도록 기존 중간합에 이번 벡터를 더한다. 새로운 화살표는 기존 중간합과 다음 중간합을 연결한다.

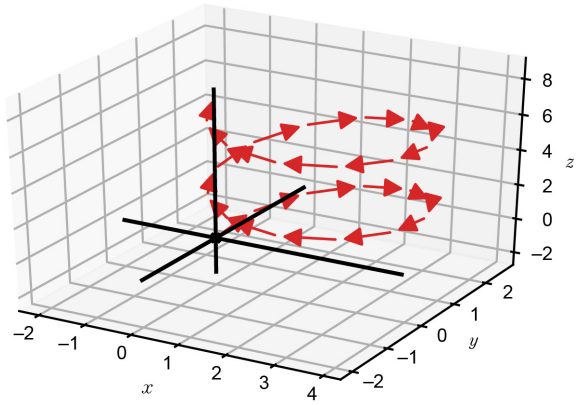


그림 3차원에서 24개의 벡터의 합을 구하기

그 합은 (-4.440892098500626e-16, -7.771561172376096e-16, 7.9999999999999964)이며 약 $(0,0,8)$이다.

연습문제 | 3.6

컴프리헨션을 사용해서 벡터의 각 좌푯값에 스칼라를 곱한다. 이 코드는 튜플로 변환되는 제너레이터 컴프리헨션(generator comprehension)[1]이다.

```
def scale(scalar,v):
    return tuple(scalar * coord for coord in v)
```

연습문제 | 3.7

$\mathbf{u}=(1,-1,-1)$과 $\mathbf{v}=(0,0,2)$이므로 다음과 같다.

$$(\mathbf{v}-\mathbf{u})=(0-1,0-(-1),2-(-1))=(-1,1,3)$$

$$\frac{1}{2}\cdot(\mathbf{v}-\mathbf{u})=\left(-\frac{1}{2},\frac{1}{2},\frac{3}{2}\right)$$

$$\mathbf{u}+\frac{1}{2}\cdot(\mathbf{v}-\mathbf{u})=\left(\frac{1}{2},-\frac{1}{2},\frac{1}{2}\right)$$

신기하게도 $\mathbf{u}+\frac{1}{2}\cdot(\mathbf{v}-\mathbf{u})$는 점 $\mathbf{u}$와 점 $\mathbf{v}$ 사이의 중점이다.

연습문제 | 3.8

$(1,1)$의 길이는 $\sqrt{1^2+1^2}=\sqrt{2}$이다. $(1,1,1)$의 길이는 $\sqrt{1^2+1^2+1^2}=\sqrt{3}$이다. 예상했겠지만 고차원 벡터에 대한 거리 공식도 동일하다. 따라서 $(1,1,1,1)$의 길이도 동일한 방법으로 구하면 $\sqrt{1^2+1^2+1^2+1^2}=\sqrt{4}=2$이다.

---

1 (옮긴이) 코드에 `tuple` 생성자가 없어도 되지 않을까 생각할 수 있지만, 파이썬에서 튜플 컴프리헨션은 정의되어 있지 않으며, 괄호 부분은 제너레이터 컴프리헨션이다. 때문에 `tuple` 생성자로 변환해주어야 한다.

연습문제 | 3.9 미니 프로젝트

다음 코드는 (임의로 고른) 100 미만의 자연수 세 개(내림차순 정렬) 조합을 탐색한다.

```
def vectors_with_whole_number_length(max_coord=100):
    for x in range(1,max_coord):
        for y in range(1,x+1):
            for z in range(1,y+1):
                if length((x,y,z)).is_integer():
                    yield (x,y,z)
```

이 코드는 좌푯값과 길이가 모두 정수인 869개의 벡터를 찾아준다. 이 중 가장 짧은 벡터는 $(2,2,1)$로 길이는 3이다. 가장 긴 벡터는 $(99,90,70)$이며 길이는 150이다.

연습문제 | 3.10

$(-1,-1,2)$의 길이는 약 2.45이므로 이 벡터에 스칼라 $\frac{1}{2.45}$을 곱해 길이를 1로 만들 수 있다.

```
>>> length((-1,-1,2))
2.449489742783178
>>> s = 1/length((-1,-1,2))
>>> scale(s,(-1,-1,2))
(-0.4082482904638631, -0.4082482904638631, 0.8164965809277261)
>>> length(scale(s,(-1,-1,2)))
1.0
```

각 좌푯값을 소수점 셋째 자리에서 반올림하면 구하는 벡터는 $(-0.41,-0.41,0.82)$이다.

3.3.5 연습문제 정답 및 풀이

연습문제 | 3.11

$\mathbf{u}$와 $\mathbf{v}$을 선택할 때만 사잇각이 직각보다 작으므로, 내적 $\mathbf{u}\cdot\mathbf{v}$만이 양수이다. 또한 $\mathbf{u}\cdot\mathbf{w}$는 $\mathbf{v}\cdot\mathbf{w}$보다 더 작은 음수이다. $\mathbf{u}$가 $\mathbf{v}$보다 길이가 더 길고 $\mathbf{w}$와 더 멀리 떨어져 있기 때문이다. 따라서 $\mathbf{u}\cdot\mathbf{v}>\mathbf{v}\cdot\mathbf{w}>\mathbf{u}\cdot\mathbf{w}$이다.

연습문제 | 3.12

$(-1,-1,1)$과 $(1,2,1)$의 내적은 $-1\cdot1+-1\cdot2+1\cdot1=-2$이다. 내적이 음수이므로 두 벡터의 사잇각은 $90°$보다 크다.

연습문제 | 3.13 미니 프로젝트

$\mathbf{u}$와 $\mathbf{v}$의 좌표를 $\mathbf{u}=(a,b,c)$, $\mathbf{v}=(d,e,f)$라고 하자. 그러면 $\mathbf{u}\cdot\mathbf{v}=ad+be+cf$이다. $s\mathbf{u}=(sa,sb,sc)$이고 $s\mathbf{v}=(sd,se,sf)$이므로, 두 내적 $(s\mathbf{u})\cdot\mathbf{v}$와 $\mathbf{u}\cdot(s\mathbf{v})$를 각각 전개해 각 결과를 증명할 수 있다.

$$
\begin{aligned}
(s\mathbf{u})\cdot\mathbf{v} &= (sa,\ sb,\ sc)\cdot(d,\ e,\ f) && \text{좌표로 바꾸어 쓰기}\\
&= sad+sbe+scf && \text{내적 연산 수행하기}\\
&= s(ad+be+cf) && \text{인수 } s\text{로 묶어서 처음 계산한 내적임을 발견하기}\\
&= s(\mathbf{u}\cdot\mathbf{v})
\end{aligned}
$$

또한 다른 곱도 같은 방법으로 확인할 수 있다.

$$
\begin{aligned}
\mathbf{u}\cdot(s\mathbf{v}) &= (a,\ b,\ c)\cdot(sd,\ se,\ sf)\\
&= asd+bse+csf\\
&= s(ad+be+cf)\\
&= s(\mathbf{u}\cdot\mathbf{v})
\end{aligned}
$$

연습문제 | 3.14 미니 프로젝트

벡터의 좌표를 (a,b,c)라 두면 내적은 $a\cdot a+b\cdot b+c\cdot c$이다.
벡터의 길이가 $\sqrt{a\cdot a+b\cdot b+c\cdot c}$이므로 내적이 길이의 제곱임을 알 수 있다.

연습문제 | 3.15 미니 프로젝트

같은 방향(예를 들어 x축의 양의 방향)인 두 벡터는 길이가 정해지면 내적값이 가장 크다.

```
>>> dot((3,0),(7,0))
21
```

반대 방향(예를 들어 각각 y축의 양의 방향과 y축의 음의 방향)인 두 벡터는 길이가 정해지면 내적값이 가장 작다.

```
>>> dot((0,3),(0,-7))
-21
```

극좌표를 사용하면 길이가 3과 7이면서 각의 크기가 랜덤인 벡터를 어렵지 않게 더 생성할 수 있다.

```
from vectors import to_cartesian
from random import random
from math import pi

def random_vector_of_length(l):
    return to_cartesian((l, 2*pi*random()))

pairs = [(random_vector_of_length(3), random_vector_of_length(7))
           for i in range(0,3)]
for u,v in pairs:
    print("u = %s, v = %s" % (u,v))
    print("length of u : %f, length of v : %f, dot product :%f" %
                (length(u), length(v), dot(u,v)))
```

연습문제 | 3.16

길이와 라디안으로 적절히 변환한 사잇각의 크기를 앞에서 유도했던 내적 공식에 대입하면 내적 계산 결과를 파이썬으로 계산할 수 있다.

```
>>> 3.61 * 1.44 * cos(101.3 * pi / 180)
-1.0186064362303022
```

소수점 셋째 자리까지 반올림하면 답은 (c)이다.

연습문제 | 3.17 미니 프로젝트

벡터 $(3,4)$는 $(4,3)$과 비교하면 x축의 양의 방향에서부터 반시계방향으로 더 회전한 상태이므로, $(4,3)$의 각에서 $(3,4)$의 각을 빼면 답을 구할 수 있다. 그 결과는 **(d)**이다.

```
>>> from vectors import to_polar
>>> r1,t1 = to_polar((4,3))
>>> r2,t2 = to_polar((3,4))
>>> t1-t2
-0.2837941092083278
>>> t2-t1
0.2837941092083278
```

연습문제 | 3.18

두 벡터의 내적은 $(1,1,1)\cdot(-1,-1,1)=1\cdot(-1)+1\cdot(-1)+1\cdot 1=-1$이다. 두 벡터의 길이가 모두 $\sqrt{3}\approx 1.732$이므로 $-1=\sqrt{3}\cdot\sqrt{3}\cdot\cos(\theta)$이고, $\cos(\theta)=-1/3$이다. 이는 두 벡터의 사잇각이 약 1.911라디안 또는 $109.5°$임을 알려준다. 따라서 답은 **(c)**이다.

3.4.5 연습문제 정답 및 풀이

연습문제 | 3.19

(a) 상단에서 내려다보면 x축과 y축이 여느 때와 같아 보이며 z축은 우리를 향한다. 따라서 이 그림은 우리의 좌표축 설정 방법을 따른다.

(b) z축이 우리를 향하고 있는데, $+y$방향은 $+x$방향에서 시계방향으로 $90°$ 회전해 있다. 이 그림은 우리의 좌표축 설정 방법과 다르다.

(c) z축의 양의 방향의 한 점(박스의 왼쪽 면)에서 바라보면 $+y$방향이 $+x$방향에서 반시계방향으로 $90°$ 회전한 것임을 볼 수 있다. 이 그림은 우리의 좌표축 설정 방법을 따른다.

(d) 박스의 좌측에서 바라보면 $+z$방향이 우리를 향해 있으며, $+y$방향도 마찬가지로 $+x$방향에서 반시계방향에 있다. 이 그림은 우리의 좌표축 설정 방법을 따른다.

따라서 이 책에서 사용하는 좌표축 설정 방법은 (a), (c), (d)이다.

연습문제 | 3.20

거울에 비친 상은 축 배치 방향이 반대이다. 그림과 같은 관점에서 z축과 y축은 같은 방향을 가리킨다. x축은 원래 y축에서 시계방향으로 회전해야 하지만, 그림에 비친 상에서는 반시계방향으로 회전한다.

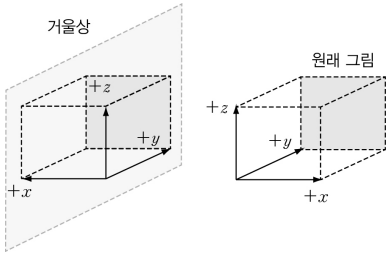


그림 x축, y축, z축과 거울에 비친 상

연습문제 | 3.21

오른손 집게손가락이 z축의 양의 방향인 $(0,0,3)$을 가리키도록 하고 다른 손가락들은 y축의 음의 방향인 $(0,-2,0)$을 가리키도록 회전하면 엄지손가락은 x축의 양의 방향을 가리킨다. 따라서 $(0,0,3)\times(0,-2,0)$은 x축의 양의 방향을 가리킨다.

연습문제 | 3.22

$(1,-2,1)$에 스칼라 -6을 곱하면 $(-6,12,-6)$이므로 두 벡터는 정반대의 방향을 가리킨다. 두 벡터로 평행사변형을 생성해도 면적은 없으므로 외적의 길이는 0이다. 길이가 0인 벡터는 오직 $(0,0,0)$뿐이다.

연습문제 | 3.23 미니 프로젝트

그림에서 벡터 $\mathbf{u}$는 밑변을 정의하므로 밑변의 길이는 $|\mathbf{u}|$이다. $\mathbf{v}$의 머리에서 밑변으로 수선을 내리면 직각삼각형을 그릴 수 있다. $\mathbf{v}$의 길이는 빗변이고, 삼각형 수선의 길이는 우리가 구하려는 높이이다. 사인 함수의 정의에 따라 높이는 $|\mathbf{v}| \cdot \sin(\theta)$이다.

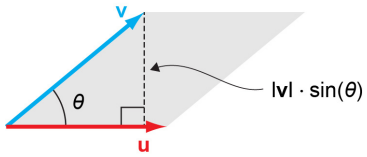


그림 평행사변형의 한 각에 대한 사인으로 만든 평행사변형 면적 공식

평행사변형 밑변의 길이는 $|\mathbf{u}|$이고 높이는 $|\mathbf{v}| \cdot \sin(\theta)$이므로 면적은 $|\mathbf{u}| \cdot |\mathbf{v}| \cdot \sin(\theta)$이다.

연습문제 | 3.24

주어진 두 벡터 모두 xz평면 위에 있으므로 외적은 y축에 있다. 오른쪽 집게손가락이 $(1,0,1)$ 방향을 가리키게 하고 나머지 손가락들이 $(-1,0,0)$을 가리키도록 회전하면 엄지손가락은 $-y$ 방향을 가리킨다.

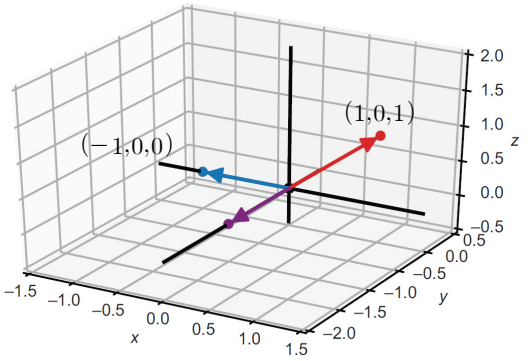


그림 $(1,0,1)$과 $(-1,0,0)$의 외적

두 벡터의 길이와 사잇각을 구해 외적의 크기를 구할 수도 있지만 좌표를 살펴보면 밑변의 길이와 높이를 바로 알 수 있다. 둘 다 1이므로 외적의 길이는 1이다. 외적은 길이가 1이고 $-y$ 방향을 가리키는 벡터 $(0,-1,0)$이므로 정답은 **(b)**이다.

연습문제 | 3.25

어떤 $\mathbf{v}$를 선택하더라도 z좌표는 0이다.

```
>>> cross((0,0,1),(1,2,3))
(-2, 1, 0)
>>> cross((0,0,1),(-1,-1,0))
(1, -1, 0)
>>> cross((0,0,1),(1,-1,5))
(1, 1, 0)
```

$\mathbf{u}=(0,0,1)$이므로 u_x와 u_y 모두 0이다. 이는 외적 공식에 등장하는 항 $u_xv_y-u_yv_x$가 v_x와 v_y에 상관없이 0임을 의미한다. 기하학적으로도 당연하다. 외적의 결과는 두 입력 벡터 모두에 수직이어야 하는데, $(0,0,1)$에 수직이려면 z성분은 반드시 0이어야 한다.

연습문제 | 3.26 미니 프로젝트

$\mathbf{u}=(u_x,u_y,u_z)$이고 $\mathbf{v}=(v_x,v_y,v_z)$라고 두자. $(\mathbf{u}\times\mathbf{v})\cdot\mathbf{u}$를 다음과 같이 쓸 수 있다.

$$(\mathbf{u}\times\mathbf{v})\cdot\mathbf{u}=(u_yv_z-u_zv_y,u_zv_x-u_xv_y,u_xv_y-u_yv_x)\cdot(u_x,u_y,u_z)$$

외적의 내적 전개하기

이 내적을 전개하면 여섯 개의 항이 등장함을 알 수 있다. 각 항은 다른 항에 의해 소거된다.

$$=(u_yv_z-u_zv_y)u_x+(u_zv_x-u_xv_z)u_y+(u_xv_y-u_yv_x)u_z$$

$$=u_yv_zu_x-u_zv_yu_x+u_zv_xu_y-u_xv_zu_y+u_xv_yu_z-u_yv_xu_z$$

완전히 전개하면 모든 항이 없어진다.

모든 항이 없어지기 때문에 결과는 0이다. 여백이 부족해서 $(\mathbf{u}\times\mathbf{v})\cdot\mathbf{v}$의 결과는 보이지 않겠지만 같은 일이 일어나서 (여섯 개의 항이 등장해 각 항이 다른 항에 의해 소거되어) 결과는 0이 될 것이다. 이는 $(\mathbf{u}\times\mathbf{v})$가 $\mathbf{u}$와 $\mathbf{v}$ 모두에 수직임을 의미한다.

3.5.4 연습문제 정답 및 풀이

연습문제 | 3.27 미니 프로젝트

8면체의 위 꼭짓점은 $(0,0,1)$이다. 이 점은 xy평면에 위치한 네 개의 점에 네 개의 변으로 연결된다. 마찬가지로 8면체의 아래 꼭짓점은 $(0,0,-1)$이며 xy평면에 위치한 네 개의 점에 연결된다. 마지막으로 xy평면에 위치한 네 개의 점은 사각형을 그리며 연결된다.

```
top = (0,0,1)
bottom = (0,0,-1)
xy_plane = [(1,0,0),(0,1,0),(-1,0,0),(0,-1,0)]
edges = [Segment3D(top,p) for p in xy_plane] +\
            [Segment3D(bottom, p) for p in xy_plane] +\
            [Segment3D(xy_plane[i],xy_plane[(i+1)%4]) for i in
range(0,4)]
draw3d(*edges)
```

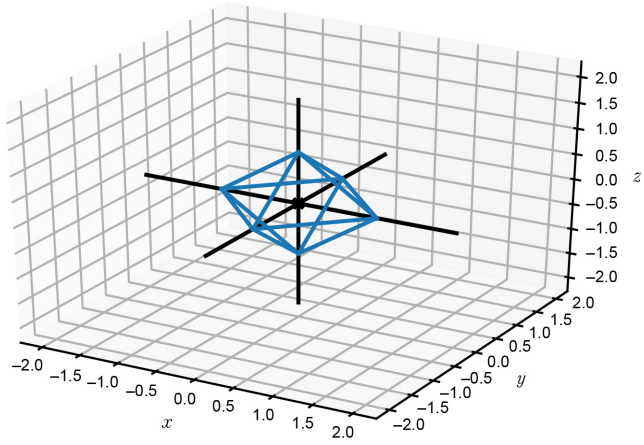


그림 8면체의 변을 찾아낸 결과

연습문제 | 3.28

그렇지 않다. 예를 들어 $[(0,1,0),(0,0,1),(1,0,0)]$도 동일한 세 개의 점의 집합이며, 이 순서로 나열하면 외적 또한 같은 방향을 가리킨다.

4.1.5 연습문제 정답 및 풀이

연습문제 | 4.1

```
def translate_by(translation):
    def new_function(v):
        return add(translation,v)
    return new_function
```

연습문제 | 4.2

`translate_by((0,0,-20))`을 `polygon_map`을 이루는 모든 다각형의 모든 벡터에 적용하면 된다.

```
draw_model(polygon_map(translate_by((0,0,-20)), load_triangles()))
```

기억하겠지만 z축 5단위 위에서 찻주전자를 바라보고 있다. 이 변환은 찻주전자를 20단위만큼 더 멀리 옮기므로 원래 찻주전자에 비해 더 작아 보인다. 실행 가능한 전체 코드는 소스 코드의 translate_teapot_down_z.py에 있다.

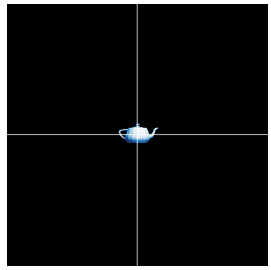
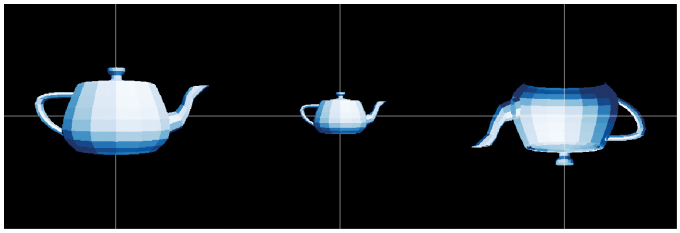


그림 z축의 아래로 20단위만큼 평행이동한 찻주전자

연습문제 | 4.3 미니 프로젝트

결과를 보기 위해 `scale_by(0.5)`와 `scale_by(-1)`을 해볼 수 있다.

```
draw_model(polygon_map(scale_by(0.5), load_triangles()))
draw_model(polygon_map(scale_by(-1), load_triangles()))
```



(a) 원래 찻주전자 (b) 0.5배만큼 축소 (c) −1배만큼 확대·축소

그림 스칼라 값에 따른 찻주전자의 변화

위 그림에서 살펴볼 수 있듯이 `scale_by(0.5)`는 찻주전자를 원래 크기의 절반으로 축소한다. `scale_by(-1)`의 결과는 찻주전자를 180°만큼 회전한 것처럼 보이지만 이 상황은 다소 복잡하다. 실제로는 회전했을 뿐만 아니라 안팎이 뒤집혔기 때문이다! 각 삼각형이 대칭이동(reflection)되어서 각 법선벡터는 이제 찻주전자의 표면 바깥쪽이 아니라 안쪽을 향한다.

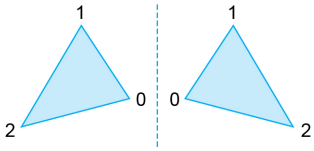


그림 대칭이동과 삼각형의 축 배치 방향

부연설명하면 위 그림과 같이 대칭이동은 삼각형의 축 배치 방향을 바꾼다. 왼쪽 삼각형의 꼭짓점에 반시계방향으로 번호를 매겼지만, 오른쪽의 대칭이동된 삼각형에서는 시계방향으로 번호가 붙는다. 이러한 삼각형의 법선벡터는 반대 방향을 향한다.

찻주전자를 회전하면 결과가 적절히 렌더링 되지 않음을 알 수 있다. 이 때문에 그래픽스에서 대칭이동을 주의해야 한다!

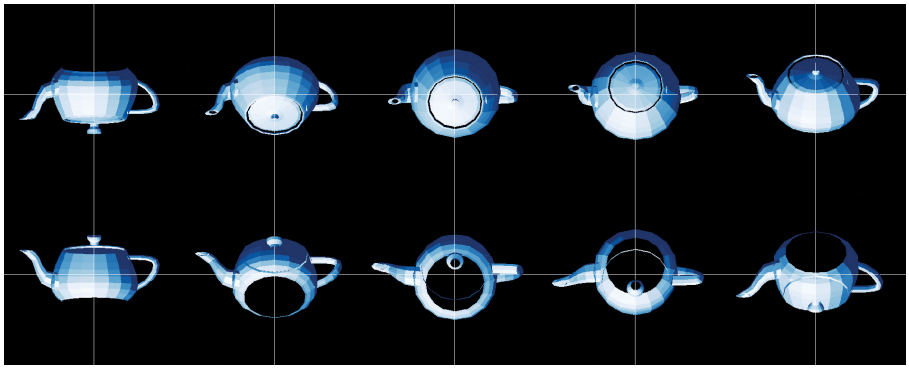


그림 대칭이동된 찻주전자의 회전

실제로 대칭이동된 찻주전자를 회전해보면 적절하게 보이지 않는다. 어떤 특징은 나타나지만 숨겨야 한다. 예를 들어 오른쪽 아래 프레임에서는 뚜껑과 속이 빈 바닥을 모두 볼 수 있다.

연습문제 | 4.4

주어진 순서대로 두 함수를 합성해서 `polygon_map`에 적용해 보자.

```
draw_model(polygon_map(compose(scale2, translate1left), load_triangles()))
```

순서를 바꿨음에도 원래 찻주전자에 비해 두 배 큰 결과를 얻지만, 이 찻주전자는 왼쪽으로 좀 더 평행이동되었다. 평행이동 후에 두 배 확대를 했기 때문에, 평행이동으로 이동한 거리도 두 배 늘어난 것이다. 소스 파일 scale_translate_teapot.py와 translate_scale_teapot.py를 실행해 결과를 비교할 수 있다.

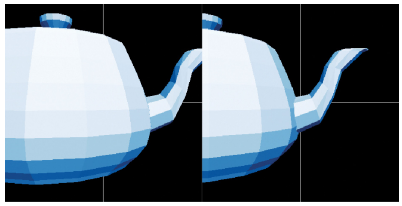


그림 찻주전자를 확대한 뒤 평행이동(왼쪽) vs 찻주전자를 평행이동한 뒤 확대(오른쪽)

연습문제 | 4.5

벡터에 이 변환을 적용하면 1.5배 확대·축소한 뒤 0.4배 확대·축소한다. 최종 확대·축소 배율은 0.6이다. 변환을 적용한 도형은 원래 도형의 크기의 60%이다.

연습문제 | 4.6

```
def compose(*args):
    def new_function(input):
        state = input
        for f in reversed(args):
            state = f(state)
        return state
    return new_function
```

compose가 리턴할 함수를 정의하며 시작한다.

현재 상태가 입력과 동일하도록 설정한다.

합성함수의 안쪽 함수를 먼저 적용해야 하므로 역순으로 입력 함수를 순회한다. 예를 들어 compose(f,g,h)(x)는 f(g(h(x)))와 같아야 하므로 h가 처음 적용된다.

각 단계에서 다음 함수를 적용해 상태를 업데이트 한다. 최종 상태는 올바른 순서로 모든 함수를 적용한 상태이다.

제대로 됐는지 확인하기 위해 함수를 만들어 합성해볼 수 있다.

```
def prepend(string):
    def new_function(input):
        return string + input
    return new_function

f = compose(prepend("P"), prepend("y"), prepend("t"))
```

이제 f("hon")을 실행하면 문자열 "Python"을 리턴한다. 일반적으로 어떤 문자열이 주어져도 문자열을 "Pyt" 앞에 덧붙인다.

연습문제 | 4.7

함수 curry2(f)의 리턴값은 새로운 함수여야 한다. 이 함수는 호출될 때마다 새로운 함수를 만들어낸다.

```
def curry2(f):
    def g(x):
        def new_function(y):
            return f(x,y)
        return new_function
    return g
```

예를 들어 앞에서 정의한 scale_by 함수를 다음과 같이 정의할 수도 있다.

```
>>> scale_by = curry2(scale)
>>> scale_by(2)((1,2,3))
(2, 4, 6)
```

연습문제 | 4.8

이 합성은 y축을 중심으로 $\pi/2$만큼 시계방향으로 회전한 것과 같다. 순서를 바꾸면 y축을 중심으로 $\pi/2$만큼 반시계방향으로 회전한 것과 같다.

연습문제 | 4.9

```
def stretch_x(scalar,vector):
    x,y,z = vector
    return (scalar*x, y, z)

def stretch_x_by(scalar):
    def new_function(vector):
        return stretch_x(scalar,vector)
    return new_function
```

4.2.5 연습문제 정답 및 풀이

연습문제 | 4.10

$\mathbf{v}=(x,y)$라고 하면 $s\mathbf{v}=(sx,sy)$, $S(s\mathbf{v})=(s^2x^2,s^2y^2)=s^2\cdot(x^2,y^2)\,s^2\cdot S(\mathbf{v})$이다. s 값 대부분과 벡터 $\mathbf{v}$에 대해 $S(s\mathbf{v})=s^2\cdot S(\mathbf{v})$는 $s\cdot S(\mathbf{v})$와 같지 않다. 반례는 $s=2$이고 $\mathbf{v}=(1,1,1)$가 있다. $S(s\mathbf{v})=(4,4,4)$이지만 $s\cdot S(\mathbf{v})=(2,2,2)$이다. 이 반례는 S가 일차변환이 아님을 보여준다.

연습문제 | 4.11

임의의 벡터 $\mathbf{v}$에 대해 $\mathbf{v}+\mathbf{0}=\mathbf{v}$이다. T가 벡터합을 보존하려면 $T(\mathbf{v}+\mathbf{0})=T(\mathbf{v})+T(\mathbf{0})$이어야 한다. $T(\mathbf{v}+\mathbf{0})=T(\mathbf{v})$이므로 $T(\mathbf{v})=T(\mathbf{v})+T(\mathbf{0})$, 즉 $\mathbf{0}=T(\mathbf{0})$이어야 한다. T는 이를 만족하지 않으므로, T는 일차변환일 수 없다.

연습문제 | 4.12

임의의 벡터 $\mathbf{v}$와 $\mathbf{w}$에 대해 $I(\mathbf{v}+\mathbf{w})=\mathbf{v}+\mathbf{w}=I(\mathbf{v})+I(\mathbf{w})$이고, 임의의 스칼라 s에 대해 $I(s\mathbf{v})=s\mathbf{v}=s\cdot I(\mathbf{v})$이다. 두 등식은 항등변환이 벡터합과 스칼라곱을 보존함을 보여준다.

연습문제 | 4.13

중점은 $\frac{1}{2}(5,3)+\frac{1}{2}(-2,1)=\left(\frac{5}{2},\frac{3}{2}\right)+\left(-1,\frac{1}{2}\right)=\left(\frac{3}{2},2\right)$이다. 다음 그림처럼 비율을 맞추어 그려보면 결과가 맞음을 확인할 수 있다.

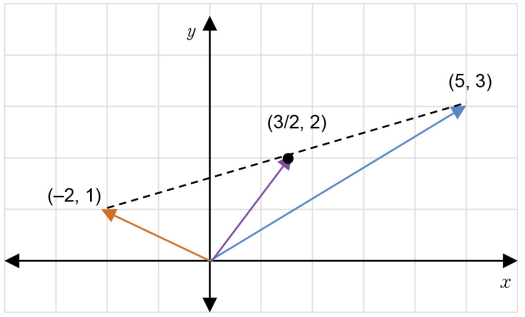


그림 $(5,3)$과 $(-2,1)$을 연결하는 선분의 중점은 $\left(\frac{3}{2},2\right)$

연습문제 | 4.14

처음에는 점 사이 간격이 일정하지만 변환 후의 그림에서는 x좌표와 y좌표가 증가할수록 수직 방향과 수평 방향에서의 간격이 각각 증가한다.

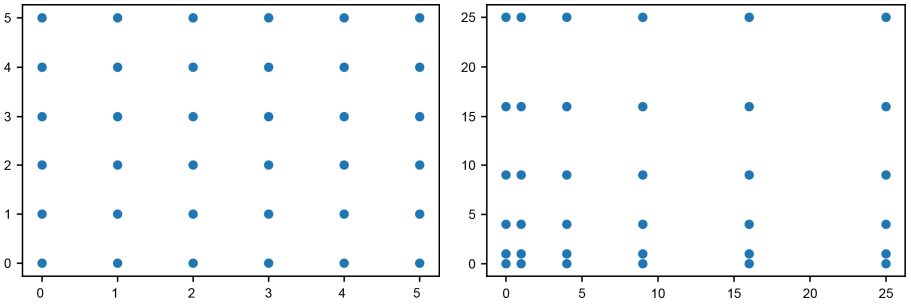


그림 격자점의 간격을 불균일하게 변경하는 변환 S

연습문제 | 4.16

다음은 벡터합을 보존하는 벡터 변환으로 x축에 대한 대칭이동의 예시이다.

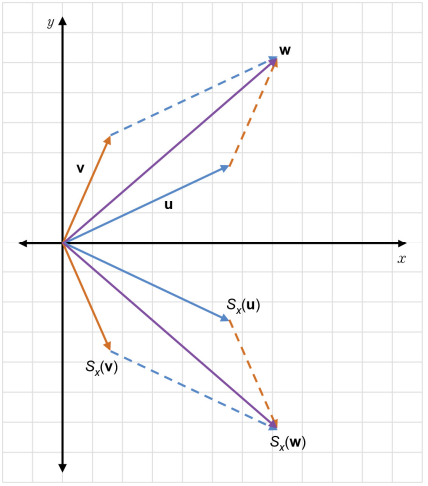


그림 벡터합을 보존하는 x축에 대한 대칭이동 S_x

다음은 x축에 대한 대칭이동이 스칼라곱을 보존함을 보이는 예시이다. 즉 $S_x(s\mathbf{v})$는 $sS_x(\mathbf{v})$가 있을 자리에 위치한다.

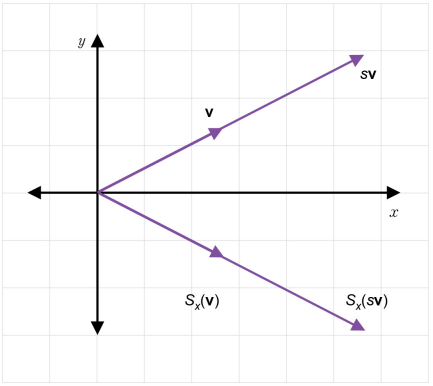


그림 x축에 대한 대칭이동은 스칼라곱을 보존

S_x가 일차변환임을 **증명**(prove)하려면 유사한 그림을 모든 벡터합과 모든 스칼라곱에 대해 그려서 보여줘야 한다. 그렇게 하려면 그림을 무수히 많이 그려야 하므로 대수적인 증명법을 사용하는 게 낫다. (두 기준을 대수적으로 보이는 법을 생각해낼 수 있는가?)

연습문제 | 4.17 미니 프로젝트

임의의 벡터합 $\mathbf{u}+\mathbf{v}=\mathbf{w}$에 대해 $S(T(\mathbf{u}))+S(T(\mathbf{v}))=S(T(\mathbf{w}))$가 성립하고 임의의 스칼라곱 $s\mathbf{v}$에 대해 $S(T(s\mathbf{v}))=s\cdot S(T(\mathbf{v}))$가 성립하면 합성변환 $S(T(\mathbf{v}))$는 일차변환이다. 이는 반드시 만족해야 하는 일차변환의 정의를 서술했을 뿐이다.

이제 합성변환이 일차변환이라는 명제가 참인 이유를 살펴보자. 임의의 두 입력 벡터 $\mathbf{u}$와 $\mathbf{v}$가 주어질 때, $\mathbf{u}+\mathbf{v}=\mathbf{w}$가 성립한다고 하자. 그러면 T의 선형성으로 인해 $T(\mathbf{u})+T(\mathbf{v})=T(\mathbf{w})$이다. 또한 이 합이 성립하므로 S의 선형성은 S가 이 합을 보존할 것임을 안다. 즉, $S(T(\mathbf{u}))+S(T(\mathbf{v}))=S(T(\mathbf{w}))$이다. 이는 $S(T(\mathbf{v}))$가 벡터합을 보존함을 의미한다.

마찬가지로 임의의 스칼라곱 $s\mathbf{v}$에 대해 T의 선형성은 $s\cdot T(\mathbf{v})=T(s\mathbf{v})$임을 알려준다. 따라서 S의 선형성은 $s\cdot S(T(\mathbf{v}))=S(T(s\mathbf{v}))$도 성립함을 말해준다. 이는 $S(T(\mathbf{v}))$가 스칼라곱을 보존함을 의미하며, 따라서 $S(T(\mathbf{v}))$는 앞서 말한 선형성 정의 전부를 만족한다. 그러므로 두 일차변환의 합성변환이 일차변환이라고 결론지을 수 있다.

연습문제 | 4.18

특정한 축에 대한 회전은 축 위의 점에 아무런 영향을 끼치지 않는다. 따라서 $T(\mathbf{e}_1)$이 x축 위에 있음을 고려할 때 $T(\mathbf{e}_1)=\mathbf{e}_1=(1,0,0)$이다. yz평면에 위치한 $\mathbf{e}_2=(0,1,0)$을 x축을 기준으로 반시계방향으로 90°만큼 회전하면 y축의 양의 방향으로 1단위에 위치한 점을 가리키는 벡터가 z축의 양의 방향으로 1단위에 위치한 점을 가리킨다. 따라서 $T(\mathbf{e}_2)=\mathbf{e}_3=(0,0,1)$이다. 마찬가지로 $\mathbf{e}_3$를 반시계방향으로 회전하면 z축의 양의 방향이 y축의 음의 방향이 된다. $T(\mathbf{e}_3)$가 새로운 방향에서도 여전히 길이가 1이기 때문에 $T(\mathbf{e}_3)$는 $-\mathbf{e}_2$로, 다시 말해 $(0,-1,0)$이다.

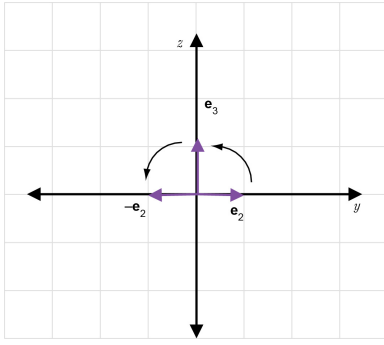


그림 $\mathbf{e}_2$와 $\mathbf{e}_3$를 각각 $\mathbf{e}_3$와 $-\mathbf{e}_2$에 대응시키는 90° 회전이동

연습문제 | 4.19

```
from vectors import *
def linear_combination(scalars,*vectors):
    scaled = [scale(s,v) for s,v in zip(scalars,vectors)]
    return add(*scaled)
```

이 함수를 이용하면 본문에서 설명한 것과 같은 결과를 얻음을 확인할 수 있다.

```
>>> linear_combination([1,2,3], (1,0,0), (0,1,0), (0,0,1))
    (1, 2, 3)
```

연습문제 | 4.20

제시된 대로 `transform`을 각 표준 기저 벡터에 적용하면 된다.

```
def transform_standard_basis(transform):
    return transform((1,0,0)), transform((0,1,0)), transform((0,0,1))
```

이 함수를 이용하면 (부동소수점 표현으로 인한 오차 범위 내에서) `rotate_x_by(pi/2)`에 대한 출력 결과를 찾으려고 한 [연습문제 4.18]의 답을 검산할 수 있다.

```
>>> from math import *
>>> transform_standard_basis(rotate_x_by(pi/2))
((1, 0.0, 0.0), (0, 6.123233995736766e-17, 1.0), (0, -1.0,
    1.2246467991473532e-16))
```

이 벡터는 근사적으로 $(1,0,0)$, $(0,0,1)$, $(0,-1,0)$이다.

연습문제 | 4.21

$\mathbf{v}=(-1,1,2)=-\mathbf{e}_1+\mathbf{e}_2+2\mathbf{e}_3$이므로 $B(\mathbf{v})=B(-\mathbf{e}_1+\mathbf{e}_2+2\mathbf{e}_3)$이다. B가 일차변환이므로 B는 일차결합을 보존한다. 따라서 $B(\mathbf{v})=-B(\mathbf{e}_1)+B(\mathbf{e}_2)+2\cdot B(\mathbf{e}_3)$가 성립한다. 이제 $B(\mathbf{v})=-(0,0,1)+(2,1,0)+2\cdot(-1,0,-1)=(0,1,-3)$이다.

연습문제 | 4.22

$A(B(\mathbf{e}_1))$는 $B(\mathbf{e}_1)=(0,0,1)=\mathbf{e}_3$에 A를 적용해서 얻는다. 이미 $A(\mathbf{e}_3)=(0,1,1)$임을 알기에 $A(B(\mathbf{e}_1))=(0,1,1)$이다.

$A(B(\mathbf{e}_2))$는 $B(\mathbf{e}_2)=(2,1,0)$에 A를 적용해서 얻는다. 이 벡터는 $A(\mathbf{e}_1)$, $A(\mathbf{e}_2)$, $A(\mathbf{e}_3)$와 스칼라 $(2,1,0)$의 일차결합이다. 따라서 $A(B(\mathbf{e}_2))$는 $2\cdot(1,1,1)+1\cdot(1,0,-1)+0\cdot(0,1,1)=(3,2,1)$이다.

$A(B(\mathbf{e}_3))$는 $B(\mathbf{e}_3)=(-1,0,-1)$에 A를 적용해서 얻는다. 이는 일차변환 $-1\cdot(1,1,1)+0\cdot(1,0,-1)\pm1\cdot(0,1,1)=(-1,-2,-2)$이다.

모든 표준 기저 벡터에 대한 A와 B의 합성 결과를 알았으니 임의의 벡터 $\mathbf{v}$에 대해 $A(B(\mathbf{v}))$를 계산할 수 있다.

5.1.6 연습문제 정답 및 풀이

연습문제 | 5.1

```
def infer_matrix(n, transformation):
    def standard_basis_vector(i):
        return tuple(1 if i==j else 0 for j in range(1,n+1))          ❶
    standard_basis = [standard_basis_vector(i) for i in range(1,n+1)] ❷
    cols = [transformation(v) for v  in standard_basis]               ❸
    return tuple(zip(*cols))                                          ❹
```

❶ i번째 좌표만 1이고 나머지 좌표는 0인 튜플로 i번째 표준 기저 벡터를 생성한다.
❷ n개의 벡터로 이루어진 리스트로 표준 기저를 생성한다.
❸ 표준 기저 벡터에 대응하는 일차변환을 적용한 결과를 행렬의 열벡터로 정의한다.
❹ 관례에 따라 열벡터 리스트 대신 행에 대한 튜플로 행렬을 재형성한다.

`rotate_z_by(pi/2)`와 같은 일차변환으로 이 코드를 테스트할 수 있다.

```
>>> from transforms import rotate_z_by
>>> from math import pi
>>> infer_matrix(3,rotate_z_by(pi/2))
((6.123233995736766e-17, -1.0, 0.0), (1.0, 1.2246467991473532e-16, 0.0), (0, 0, 1))
```

연습문제 | 5.2

행렬의 첫 번째 행과 벡터의 내적은 $-2.5 \cdot 1.3 + 0.3 \cdot -0.7 = -3.46$이다. 행렬의 두 번째 행과 벡터의 내적은 $-2.5 \cdot 6.5 + 0.3 \cdot 3.2 = -15.29$이다. 이 두 값은 출력 벡터의 좌표이며, 그 결과는 다음과 같다.

$$\begin{pmatrix} 1.3 & 0.7 \\ 6.5 & 3.2 \end{pmatrix} \begin{pmatrix} -2.5 \\ 0.3 \end{pmatrix} = \begin{pmatrix} -3.46 \\ -15.29 \end{pmatrix}$$

연습문제 | 5.3 미니 프로젝트

먼저 `random_matrix` 함수에 행의 개수, 열의 개수, 성분의 최댓값과 최솟값을 명시하기 위한 인자를 넘겨주도록 작성해야 한다.

```
from random import randint
def random_matrix(rows,cols,min=-2,max=2):
    return tuple(
        tuple(
        randint(min,max) for j in range(0,cols))
        for i in range(0,rows)
    )
```

그러면 0에서 10 사이의 성분을 갖는 3×3 랜덤 행렬을 다음과 같이 생성할 수 있다.

```
>>> random_matrix(3,3,0,10)
((3, 4, 9), (7, 10, 2), (0, 7, 4))
```

연습문제 | 5.4

운이 좋은 경우만 제외하면 결과는 모두 다를 것이다. 행렬 쌍은 순서를 뒤집어서 곱하면 대부분 결과가 다르게 나온다. 입력 순서에 상관없이 같은 결과를 내는 연산을 수학 용어로 가환성(commutative)이 있다[1]고 한다. 어떤 두 수 x, y를 선택하더라도 $xy=yx$가 성립하기 때문에, 수의 곱셈은 가환성이 있는 연산이다. 하지만 행렬곱은 가환성이 **없는데**, 두 정사각행렬 A, B에 대해 AB가 BA와 언제나 같은 건 아니기 때문이다.

연습문제 | 5.5

2차원 또는 3차원에서 항등변환을 표준 기저 벡터에 작용하면 변형 없이 그대로 놔둔다. 따라서 각 차원에서 이 변환에 대한 행렬의 각 열은 표준 기저 벡터이다. 2차원과 3차원에서 **항등행렬**(identity matrix)은 각각 I_2, I_3라고 표기하며 다음과 같다.

$$I_2=\begin{pmatrix}1&0\\0&1\end{pmatrix},\ I_3=\begin{pmatrix}1&0&0\\0&1&0\\0&0&1\end{pmatrix}$$

1 (옮긴이) 쉽게 말하면 '행렬의 곱셈은 교환법칙이 성립한다.'라고 표현할 수 있다.

연습문제 | 5.6

다음 함수는 소스 파일 matrix_transform_teapot.py에 포함되어 있다.

```
def transform(v):
    m = ((2,1,1),(1,2,1),(1,1,2))
    return multiply_matrix_vector(m,v)

draw_model(polygon_map(transform, load_triangles()))
```

그림 찻주전자의 모든 꼭짓점에 해당 행렬을 적용한 결과

이 코드를 실행해보면 찻주전자의 앞부분이 x, y, z가 모두 양수인 영역에서 늘어나 있음을 볼 수 있다. 이는 표준 기저 벡터 모두가 양수 좌푯값을 가진 벡터 $(2,1,1)$, $(1,2,1)$, $(1,1,2)$로 각각 변환되었기 때문이다.

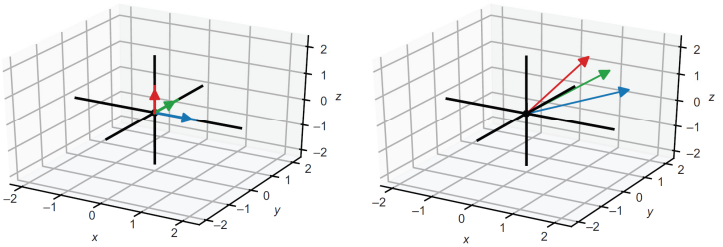


그림 주어진 행렬로 정의된 일차변환이 표준 기저 벡터에 끼친 영향

스칼라가 모두 양수일 때 새 벡터의 일차결합은 표준 기저의 일차결합보다 $+x$, $+y$, $+z$ 방향으로 더 늘어나게 한다.

연습문제 | 5.7

```
def multiply_matrix_vector(matrix,vector):
    return tuple(
        sum(vector_entry * matrix_entry
            for vector_entry, matrix_entry in zip(row,vector))
        for row in matrix
    )
```

연습문제 | 5.8

[연습문제 5.7]의 풀이를 단순화하여 다음과 같이 나타낼 수 있다.

```
def multiply_matrix_vector(matrix,vector):
    return tuple(
        dot(row,vector)
        for row in matrix
    )
```

연습문제 | 5.9 미니 프로젝트

지금은 2차원만 증명해보자. 3차원에서 증명할 땐 구조는 동일하지만 기재할 게 더 많다. 2×2 행렬 A의 각 성분이 수 a, b, c, d로 이루어졌다고 하자. A가 두 벡터 $\mathbf{u}$와 $\mathbf{v}$에 어떻게 작용하는지 살펴보자.

$$A=\begin{pmatrix}a & b\\ c & d\end{pmatrix},\ \mathbf{u}=\begin{pmatrix}u_1\\ u_2\end{pmatrix},\ \mathbf{v}=\begin{pmatrix}v_1\\ v_2\end{pmatrix}$$

$A\mathbf{u}$와 $A\mathbf{v}$를 구하고자 각 행렬 곱셈을 명확히 써볼 수 있다.

$$A\mathbf{u}=\begin{pmatrix}a & b\\ c & d\end{pmatrix}\begin{pmatrix}u_1\\ u_2\end{pmatrix}=\begin{pmatrix}au_1+bu_2\\ cu_1+du_2\end{pmatrix}$$

$$A\mathbf{v}=\begin{pmatrix}a & b\\ c & d\end{pmatrix}\begin{pmatrix}v_1\\ v_2\end{pmatrix}=\begin{pmatrix}av_1+bv_2\\ cv_1+dv_2\end{pmatrix}$$

이제 $A\mathbf{u}+A\mathbf{v}$와 $A(\mathbf{u}+\mathbf{v})$를 계산해서 결과가 같은지 볼 수 있다.

$$A\mathbf{u}+A\mathbf{v}=\begin{pmatrix}au_1+bu_2\\ cu_1+du_2\end{pmatrix}+\begin{pmatrix}av_1+bv_2\\ cv_1+dv_2\end{pmatrix}=\begin{pmatrix}au_1+av_1+bu_2+bv_2\\ cu_1+cv_1+du_2+dv_2\end{pmatrix}$$

$$A(\mathbf{u}+\mathbf{v})=\begin{pmatrix}a & b\\ c & d\end{pmatrix}\begin{pmatrix}u_1+v_1\\ u_2+v_2\end{pmatrix}=\begin{pmatrix}a(u_1+v_1)+b(u_2+v_2)\\ c(u_1+v_1)+d(u_2+v_2)\end{pmatrix}=\begin{pmatrix}au_1+av_1+bu_2+bv_2\\ cu_1+cv_1+du_2+dv_2\end{pmatrix}$$

이 결과는 임의의 2×2 행렬에 대해 해당 행렬을 곱해서 정의한 2차원 벡터 변환이 벡터합을 보존함을 보여준다. 마찬가지로 임의의 수 s에 대해 다음이 성립한다.

$$s\mathbf{v}=\begin{pmatrix}sv_1\\ sv_2\end{pmatrix}$$

$$s(A\mathbf{v})=\begin{pmatrix}s(av_1+bv_2)\\ s(cv_1+dv_2)\end{pmatrix}=\begin{pmatrix}sav_1+sbv_2\\ scv_1+sdv_2\end{pmatrix}$$

$$A(s\mathbf{v})=\begin{pmatrix}a(sv_1)+b(sv_2)\\ c(sv_1)+d(sv_2)\end{pmatrix}=\begin{pmatrix}sav_1+sbv_2\\ scv_1+sdv_2\end{pmatrix}$$

$s \cdot (A\mathbf{v})$와 $A(s\mathbf{v})$의 결과가 같으므로 행렬 A의 곱셈이 스칼라곱도 보존함을 볼 수 있다. 이 두 사실은 임의의 2×2 행렬의 곱셈이 2차원 벡터의 일차변환임을 의미한다.

연습문제 | 5.10

행렬 A와 행렬 B가 정의하는 일차변환을 각각 수행하는 두 함수 `transform_a`와 `transform_b`를 구현하자. 그 뒤 `compose` 함수를 사용해 이 두 함수를 결합한다.

```
from transforms import compose

a = ((1,1,0),(1,0,1),(1,-1,1))
b = ((0,2,1),(0,1,0),(1,0,-1))

def transform_a(v):
    return multiply_matrix_vector(a,v)

def transform_b(v):
    return multiply_matrix_vector(b,v)

compose_a_b = compose(transform_a, transform_b)
```

이제 `infer_matrix` 함수를 사용해 이 두 일차변환의 합성에 대응하는 행렬을 구한 뒤 행렬곱 AB와 비교할 수 있다.

```
>>> infer_matrix(3, compose_a_b)
((0, 3, 1), (1, 2, 0), (1, 1, 0))
>>> matrix_multiply(a,b)
((0, 3, 1), (1, 2, 0), (1, 1, 0))
```

연습문제 | 5.11 미니 프로젝트

이 문제를 푸는 한 가지 방법으로 두 행렬을 쓴 뒤 그 곱이 항등행렬인지 확인하는 작업을 반복하는 방법이 있다. 다른 방법은 일차변환 측면에서 생각하면 된다. 두 행렬을 곱해서 항등행렬이 나온다면 대응하는 각각의 일차변환을 합성할 때 항등변환이 된다는 뜻이다.

이를 염두에 두고 합성했을 때 항등변환이 되는 2차원 일차변환 두 개를 생각해보자. 주어진 2차원 벡터에 순서대로 적용했을 때, 이 일차변환들은 출력으로 원래 벡터를 리턴해야 한다. 그러한 일차변환 쌍은 시계방향으로 90°만큼 회전이동한 다음 시계방향으로 270°만큼 회전이동하는 것이다. 두 일차변환을 적용하면 모든 벡터가 원래 위치로 돌아가는 360° 회전이 된다. 270° 회전이동과 90° 회전이동에 대응하는 행렬은 다음과 같으며, 그 곱은 항등행렬이다.

$$\begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

연습문제 | 5.12

1보다 크거나 같은 정수 지수만큼 거듭제곱하는 구현은 다음과 같다.

```
def matrix_power(power,matrix):
    result = matrix
    for _ in range(1,power):
        result = matrix_multiply(result,matrix)
    return result
```

5.2.6 연습문제

연습문제 | 5.13

이 행렬은 3개의 행과 5개의 열이 있으므로 답은 (b)이다.

연습문제 | 5.14

2차원 열벡터는 행이 2개이고 열이 1개이므로 2×1 행렬이다.

2차원 행벡터는 행이 1개이고 열이 2개이므로 1×2 행렬이다.

같은 원리로 3차원 열벡터와 3차원 행벡터는 각각 3×1 행렬, 1×3 행렬이다.

연습문제 | 5.16

(a) 2×2 행렬과 4×4 행렬의 곱셈은 유효하지 않다. 첫 번째 행렬은 2개의 열만 있지만 두 번째 행렬은 4개의 열이 있다.

(b) 2×4 행렬과 4×2 행렬의 곱셈은 유효하다. 첫 번째 행렬은 4개의 열이 있고 두 번째 행렬은 4개의 행이 있다. 이 곱셈의 결과는 2×2 행렬이다.

(c) 3×1 행렬과 1×8 행렬의 곱셈은 유효하다. 첫 번째 행렬의 열은 1개이고 두 번째 행렬의 행도 1개이다. 이 곱셈의 결과는 3×8 행렬이다.

(d) 3×3 행렬과 2×3 행렬의 곱셈은 유효하지 않다. 첫 번째 행렬은 3개의 열이 있지만 두 번째 행렬은 2개의 행이 있다.

연습문제 | 5.17

첫 번째 행렬의 열 개수와 두 번째 행렬의 행 개수가 같아야 하므로 각 행렬의 차원을 $m\times n$과 $n\times k$라고 하자. 그러면 $mn=15$이고 $nk=6$이다. 이를 만족하는 경우는 2가지이다.

❶ $m=5, n=3, k=2$인 경우 5×3 행렬과 3×2 행렬을 곱해 5×2 행렬이 된다.

❷ $m=15, n=1, k=6$인 경우 15×1 행렬과 1×6 행렬을 곱해 15×6 행렬이 된다.

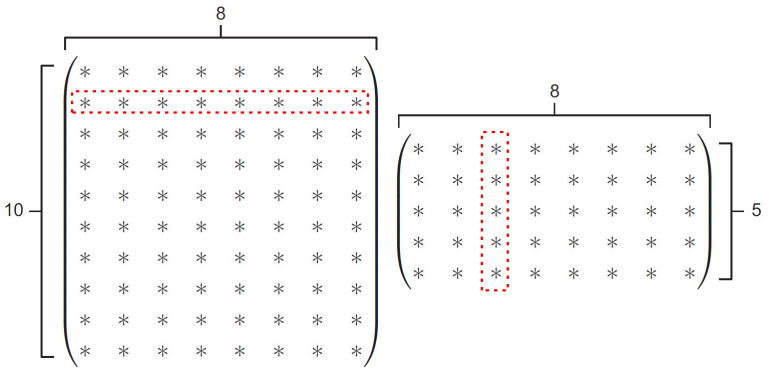
연습문제 | 5.18

```
def transpose(matrix):
    return tuple(zip(*matrix))
```

`zip(*matrix)`를 호출하면 행렬의 열로 구성된 리스트를 리턴하므로, 이를 튜플로 만들어준다. 이러한 동작은 임의의 입력 행렬의 행과 열을 바꿔주는 효과가 있으며, 열벡터를 행벡터로 변환하고 행벡터를 열벡터로 변환한다.

```
>>> transpose(((1,),(2,),(3,)))
((1, 2, 3),)
>>> transpose(((1, 2, 3),))
((1,), (2,), (3,))
```

연습문제 | 5.19



첫 번째 행렬의 행은 10개의 성분이 있지만 두 번째 행렬의 열은 5개의 성분만 있다. 따라서 이 행렬 곱셈은 계산할 수 없다.

연습문제 | 5.20

유효한 행렬곱 중 하나는 BC로, 2×3 행렬과 3×5 행렬의 곱은 2×5 행렬이다. 다른 하나는 CA로, 3×5 행렬과 5×7 행렬을 곱하면 3×7 행렬이다. 세 행렬의 곱 중 BCA가 유일하게 유효하며, 곱하는 순서는 상관이 없다. 즉, $(BC)A$가 2×5 행렬과 5×7 행렬을 곱한 행렬이고 $B(CA)$가 2×3 행렬과 3×7 행렬을 곱한 행렬인데 둘 다 2×7 행렬이다.

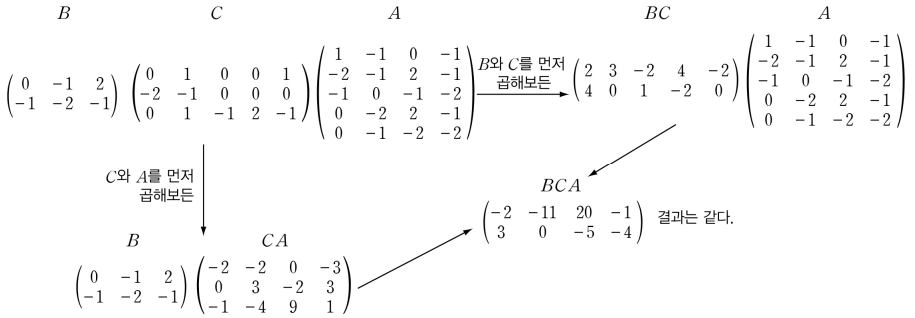


그림 세 행렬을 다른 순서로 곱하기

연습문제 | 5.21

yz평면으로의 투영은 x좌표를 없앤다. 이 연산의 행렬은 다음과 같다.

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

마찬가지로 xz평면으로의 투영은 y좌표를 없앤다. 이 연산의 행렬은 다음과 같다.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

예를 들어 $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ z \end{pmatrix}$이고 $\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} y \\ z \end{pmatrix}$이다.

연습문제 | 5.22

확인할 수 있는 함수 중에 xy평면으로의 투영이 있다. 3차원 벡터를 입력으로 받아 2차원 벡터를 리턴한다. 이 일차변환을 파이썬 함수로 구현한 뒤 이 일차변환의 2×3 행렬을 추론해 보자.

```
>>> def project_xy(v):
...     x,y,z = v
...     return (x,y)
...
>>> infer_matrix(3,project_xy)
((1, 0, 0), (0, 1, 0))
```

참고로 이 함수의 인자로 **입력** 벡터의 차수를 제공해야 올바른 표준 기저 벡터를 만들어 `project_xy`의 동작을 확인할 수 있다. 3차원 표준 기저 벡터가 `project_xy`에 전달되면 2차원 벡터를 자동으로 출력하며, 이를 통해 행렬의 각 열이 출력된다.

연습문제 | 5.23

이러한 행렬은 다음과 같다.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

입력 벡터의 1, 2, 4, 5번째 좌표가 출력 벡터의 4개의 좌표를 형성함을 알 수 있다.

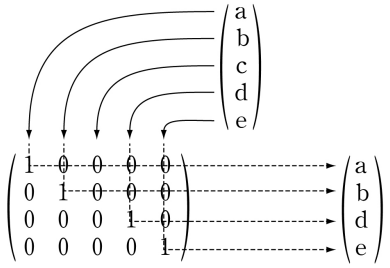


그림 출력 벡터의 위치를 결정하는 행렬 내의 성분

연습문제 | 5.24 미니 프로젝트

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} l \\ e \\ m \\ o \\ n \\ s \end{pmatrix} = \begin{pmatrix} 0+0+0+0+0+s \\ 0+0+0+0+o+0 \\ l+0+0+0+0+0 \\ 0+e+0+0+0+0 \\ 0+0+m+0+0+0 \\ 0+0+0+0+n+0 \end{pmatrix} = \begin{pmatrix} s \\ o \\ l \\ e \\ m \\ n \end{pmatrix}$$

그림 6차원 벡터 (l,e,m,o,n,s)를 (s,o,l,e,m,n)으로 변환하는 행렬

연습문제 | 5.25

M은 3×3 행렬이고 N은 2×2 행렬이며, P와 Q는 둘 다 2×3이다. M과 M을 곱한 $MM=M^2$은 유효하며 3×3 행렬이다. $NN=N^2$도 마찬가지로 유효하며 2×2 행렬이다. 이와 별도로 PM, QM, NP, NQ가 유효하며 모두 3×2 행렬이다.

5.3.5 연습문제 정답 및 풀이

연습문제 | 5.26

`[(x,y,2) for x,y in dino_vectors]`를 사용한 뒤 동일한 3×3 행렬을 적용하면 공룡은 벡터 $(3,1)$이 아니라 벡터 $(6,2)$만큼(주어진 벡터의 2배만큼) 평행이동한다. 이는 벡터 $(0,0,1)$이 $(3,1)$만큼 평행이동하고 이 행렬이 일차변환에 해당하기 때문이다.

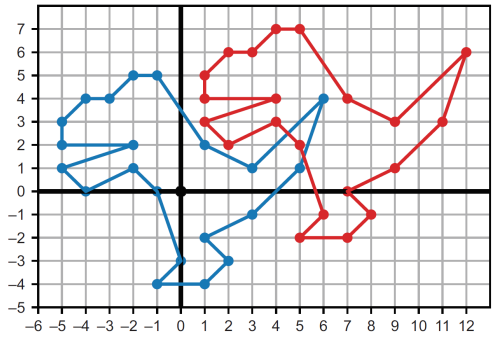


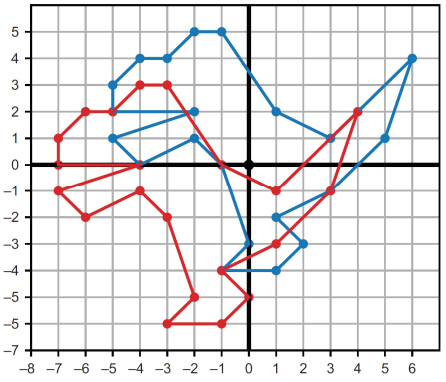
그림 평면 $z=2$ 상의 공룡은 같은 행렬에 의해 원래의 2배만큼 평행이동

연습문제 | 5.27

원래 행렬에서 3,1을 각각 $-2,-2$로 바꾸면 다음을 얻는다.

$$\begin{pmatrix}1&0&2\\0&1&2\\0&0&1\end{pmatrix}$$

당연히 공룡은 벡터 $(-2,-2)$만큼 평행이동하며, 왼쪽 아래로 이동한다.



연습문제 | 5.28

3차원 벡터의 초기 z좌표를 수 z라고 하면 이 행렬은 z좌표를 가만히 둔다.

$$\begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} ax+by+cz \\ dx+ey+fz \\ 0x+0y+z \end{pmatrix}$$

연습문제 | 5.29 미니 프로젝트

2차원 벡터를 45°만큼 회전시키는 2×2 행렬을 구하자.

```
>>> from vectors import rotate2d
>>> from transforms import *
>>> from math import pi
>>> rotate_45_degrees = curry2(rotate2d)(pi/4)  ❶
>>> rotation_matrix = infer_matrix(2,rotate_45_degrees)
>>> rotation_matrix
((0.7071067811865476, -0.7071067811865475), (0.7071067811865475,
0.7071067811865476))
```

❶ rotate2d를 실행해 2차원 입력 벡터를 45°(= $\pi/4$ 라디안)만큼 회전시키는 함수를 생성한다.

이 행렬을 근사적으로 나타내면 다음과 같다.

$$\begin{pmatrix} 0.707 & -0.707 \\ 0.707 & 0.707 \end{pmatrix}$$

비슷하게 $\frac{1}{2}$배수로 확대·축소하는 행렬을 구할 수 있다.

$$\begin{pmatrix} 0.5 & 0 \\ 0 & 0.5 \end{pmatrix}$$

이 두 행렬을 곱하면 다음 코드와 같이 두 변환을 동시에 수행할 수 있다.

```
>>> from matrices import *
>>> scale_matrix = ((0.5,0),(0,0.5))
>>> rotate_and_scale = matrix_multiply(scale_matrix,rotation_matrix)
>>> rotate_and_scale
((0.3535533905932738, -0.35355339059327373), (0.35355339059327373,
0.3535533905932738))
```

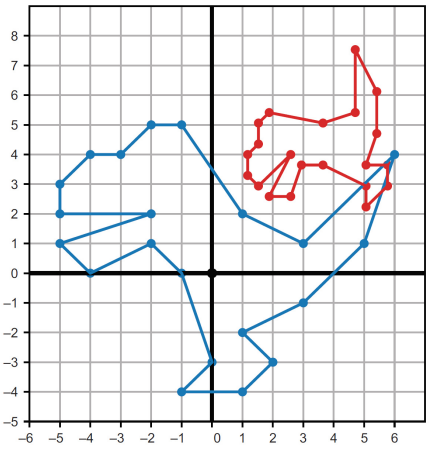
다음 행렬은 평면 $z=1$에 있는 공룡을 $(2,2)$만큼 평행이동하는 3×3 행렬이다.

$$\begin{pmatrix} 1 & 0 & 2 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix}$$

이 행렬의 왼쪽 위에 2×2 회전과 확대·축소를 동시에 하는 행렬을 집어 넣으면 원하는 최종 행렬을 구할 수 있다.

```
>>> ((a,b),(c,d)) = rotate_and_scale
>>> final_matrix = ((a,b,2),(c,d,2),(0,0,1))
>>> final_matrix
((0.3535533905932738, -0.35355339059327373, 2), (0.35355339059327373,
0.3535533905932738, 2), (0, 0, 1))
```

이 공룡을 평면 $z=1$로 옮긴 뒤, 3차원에서 이 행렬을 적용하고 2차원으로 투영하자. 다음과 같이 행렬 곱셈 한 번으로 회전이동, 확대·축소, 평행이동을 모두 적용한 공룡을 얻는다.



연습문제 | 5.30

공룡이 평면 $z=1$에 있다면 다음 행렬은 평행이동 **없이** 90°만큼 회전이동을 수행한다.

$$\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

먼저 회전이동한 뒤 회전이동하려면, 이 회전이동 행렬과 평행이동 행렬을 곱한다.

$$\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}=\begin{pmatrix} 0 & -1 & -1 \\ 1 & 0 & 3 \\ 0 & 0 & 1 \end{pmatrix}$$

이 행렬은 평행이동 전에 회전이동하는 행렬과는 다르다. 이 경우 평행이동 벡터 $(3,1)$은 90° 회전에 영향을 받기 때문에 실제 효과는 $(-1,3)$만큼 평행이동된다.

연습문제 | 5.31

앞과 동일하게 설정하되 4차원 벡터에 5번째 좌표 1을 부여해서 5차원 벡터로 바꾸면 된다.

```
def translate_4d(translation):
    def new_function(target):
        a,b,c,d = translation
        x,y,z,w = target
        matrix = (
            (1,0,0,0,a),
            (0,1,0,0,b),
            (0,0,1,0,c),
            (0,0,0,1,d),
            (0,0,0,0,1))
        vector = (x,y,z,w,1)
        x_out,y_out,z_out,w_out,_ = multiply_matrix_vector(matrix,vector)
        return (x_out,y_out,z_out,w_out)
    return new_function
```

평행이동이 정상 동작함을 확인해보자. 두 벡터를 더한 효과가 나온다.

```
>>> translate_4d((1,2,3,4))((10,20,30,40))
(11, 22, 33, 44)
```

6.1.7 연습문제 정답 및 풀이

연습문제 | 6.1

```
class Vec3(Vector):
    def __init__(self,x,y,z):
        self.x = x
        self.y = y
        self.z = z
    def add(self,other):
        return Vec3(self.x + other.x,
                    self.y + other.y,
                    self.z + other.z)
    def scale(self,scalar):
        return Vec3(scalar * self.x,
                    scalar * self.y,
                    scalar * self.z)
    def __eq__(self,other):
        return (self.x == other.x
                and self.y == other.y
                and self.z == other.z)
    def __repr__(self):
        return "Vec3({},{},{})".format(self.x, self.y, self.z)
```

연습문제 | 6.2 미니 프로젝트

CoordinateVector 클래스 구현을 위해 2장과 3장에 등장한 차원 독립적인 연산 add와 scale을 사용할 수 있다. 이 클래스에 차원이 구현되어 있지 않다보니 어떤 차원에서 동작하는지 모르기 때문에 CoordinateVector를 인스턴스화할 수 없다.

```
from abc import abstractproperty
from vectors import add, scale
class CoordinateVector(Vector):
    @abstractproperty
    def dimension(self):
        pass
    def __init__(self,*coordinates):
        self.coordinates = tuple(x for x in coordinates)
    def add(self,other):
        return self.__class__(*add(self.coordinates, other.coordinates))
    def scale(self,scalar):
        return self.__class__(*scale(scalar, self.coordinates))
    def __repr__(self):
        return "{}{}".format(self.__class__.__qualname__, self.coordinates)
```

예를 들어 차원을 6이라고 선택하면 구체화된 클래스를 만들어 인스턴스화할 수 있다.

```
class Vec6(CoordinateVector):
    def dimension(self):
        return 6
```

벡터합과 스칼라곱, 그 외의 정의는 CoordinateVector 베이스 클래스에서 가져온다.

```
>>> Vec6(1,2,3,4,5,6) + Vec6(1, 2, 3, 4, 5, 6)
Vec6(2, 4, 6, 8, 10, 12)
```

CoordinateVector 클래스는 특정한 좌표 벡터를 여럿 구현해야 할 때 반복 작업을 줄여준다. 예를 들어 Vec6 클래스를 구현하려면 CoordinateVector를 상속받고 차원을 6으로 설정하면 된다.

연습문제 | 6.3

```
from abc import ABCMeta, abstractmethod, abstractproperty

class Vector(metaclass=ABCMeta):
    ...
    @classmethod
    @abstractproperty
    def zero():
        pass

    def __neg__(self):
        return self.scale(-1)
```

어떠한 벡터공간에서도 영벡터 값은 오직 하나이기 때문에 zero는 클래스 메서드이다.

zero는 추상 프로퍼티이기도 한데 zero가 뭔지 아직 구체화하지 않았기 때문이다.

산술 부정 연산자를 오버로딩하기 위한 특별한 메서드 명이다.

부모 클래스에 __neg__의 정의가 포함되어 있으므로 자식 클래스가 __neg__를 구현할 필요가 없다. 하지만 각 클래스에서 zero는 구현해야 한다.

```
        class Vec2(Vector):
            ...
            def zero():
                return Vec2(0,0)
```

연습문제 | 6.4

테스트 함수가 일반적이었으므로 Vec3 객체에 대한 동등성 함수와 100개의 랜덤 입력을 제공하면 된다.

```
def random_vec3():
    return Vec3(random_scalar(),random_scalar(),random_scalar())

def approx_equal_vec3(v,w):
    return isclose(v.x,w.x) and isclose(v.y,w.y) and isclose(v.z, w.z)

for i in range(0,100):
    a,b = random_scalar(), random_scalar()
    u,v,w = random_vec3(), random_vec3(), random_vec3()
    test(approx_equal_vec3,a,b,u,v,w)
```

연습문제 | 6.5

영벡터는 테스트하려는 클래스마다 다르기 때문에 함수에 인자로 전달해주어야 한다.

```
def test(zero,eq,a,b,u,v,w):
    ...
    assert eq(zero + v, v)
    assert eq(0 * v, zero)
    assert eq(-v + v, zero)
```

zero 메서드를 구현한 어떠한 벡터 클래스도 테스트할 수 있다. [연습문제 6.3]을 참고하자.

```
for i in range(0,100):
    a,b = random_scalar(), random_scalar()
    u,v,w = random_vec2(), random_vec2(), random_vec2()
    test(Vec2.zero(), approx_equal_vec2, a,b,u,v,w)
```

연습문제 | 6.6

보면 알겠지만 덧셈에서도 이런 확인을 해야 한다!

```
class Vec2(Vector):
    ...
    def add(self,other):
        assert self.__class__ == other.__class__
        return Vec2(self.x + other.x, self.y + other.y)
    ...

def __eq__(self,other):
        return (self.__class__ == other.__class__
            and self.x == other.x and self.y == other.y)
```

Vector의 다른 자식 클래스에도 이러한 확인 작업을 추가하면 안전하다.

연습문제 | 6.7

```
class Vector(metaclass=ABCMeta):
    ...
    def __truediv__(self, scalar):
        return self.scale(1.0/scalar)
```

이를 구현하면 Vec2(1,2)/2와 같은 나눗셈을 해서 Vec2(0.5, 1.0)을 얻는다.

6.2.6 연습문제 정답 및 풀이

연습문제 | 6.8

랜덤하게 생성된 스칼라를 벡터로 두고 수 0을 영벡터로 두자. math.isclose를 동등성 테스트라고 두어서 랜덤 테스트를 100번 통과하면 된다.

```
for i in range(0,100):
    a,b = random_scalar(), random_scalar()
    u,v,w = random_scalar(), random_scalar(), random_scalar()
    test(0, isclose, a,b,u,v,w)
```

연습문제 | 6.9 미니 프로젝트

작업은 대부분 랜덤 데이터 생성과 날짜 및 시간을 다루는 근사적 동등성 테스트 구축으로 이루어져 있다. 다음을 확인하라.

```
from math import isclose
from random import uniform, random, randint
from datetime import datetime, timedelta

def random_time():
    return CarForSale.retrieved_date - timedelta(days=uniform(0,10))

def approx_equal_time(t1, t2):
    test = datetime.now()
    return isclose((test-t1).total_seconds(), (test-t2).total_seconds())

def random_car():
    return CarForSale(randint(1990,2019), randint(0,250000),
              27000. * random(), random_time())

def approx_equal_car(c1,c2):
    return (isclose(c1.model_year,c2.model_year)
            and isclose(c1.mileage,c2.mileage)
            and isclose(c1.price, c2.price)
            and approx_equal_time(c1.posted_datetime, c2.posted_datetime))

for i in range(0,100):
    a,b = random_scalar(), random_scalar()
    u,v,w = random_car(), random_car(), random_car()
    test(CarForSale.zero(), approx_equal_car, a,b,u,v,w)
```

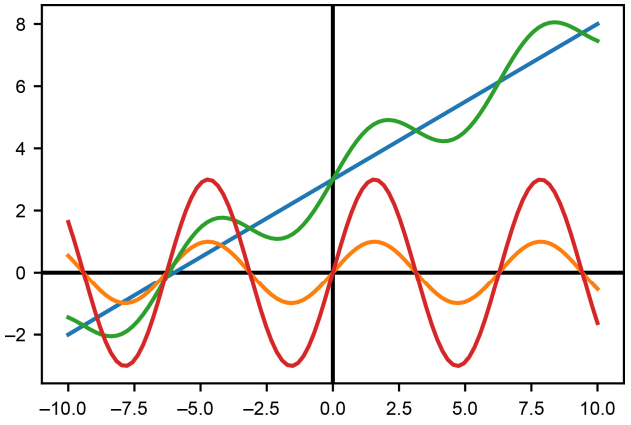
연습문제 | 6.10

```
class Function(Vector):
    def __init__(self, f):
        self.function = f
    def add(self, other):
        return Function(lambda x: self.function(x) + other.function(x))
    def scale(self, scalar):
        return Function(lambda x: scalar * self.function(x))
    @classmethod
    def zero(cls):
        return Function(lambda x: 0)
    def __call__(self, arg):
        return self.function(arg)

f = Function(lambda x: 0.5 * x + 3)
g = Function(sin)

plot([f, g, f+g, 3*g], -10, 10)
```

마지막 줄의 결과는 다음과 같이 플로팅된다.



객체 f와 g는 벡터처럼 동작하므로 이 객체들을 더하거나 객체에 스칼라를 곱할 수 있다. 또한 이 객체들은 함수처럼 동작하기 때문에 플로팅할 수도 있다.

연습문제 | 6.11 미니 프로젝트

우리의 관심사는 주로 잘 동작하는 연속 함수에 있기 때문에, 다음과 같이 몇몇 랜덤 입력값에 대해 두 함숫값이 가까운지 체크하는 정도면 되겠다.

```
def approx_equal_function(f,g):
    results = []
    for _ in range(0,10):
        x = uniform(-10,10)
        results.append(isclose(f(x),g(x)))
    return all(results)
```

아쉽게도 위의 코드는 오해의 소지가 있는 결과를 줄 수 있다. 다음 코드를 실행해보면 이 함수는 `True`를 리턴하기 때문이다.

```
approx_equal_function(lambda x: (x*x)/x, lambda x: x)
```

함수의 동등성을 계산하는 문제는 결정불가능성(undecidable) 문제의 일종임이 밝혀졌다. 즉, 임의의 두 함수가 동등한지를 보장할 수 있는 알고리즘은 없음이 증명되어 있다.

연습문제 | 6.12 미니 프로젝트

함수의 동등성을 테스트하기는 어려우며, 함수를 랜덤하게 생성하기도 어렵다. 여기서는 (다음 절에서 다룰) `Polynomial` 클래스를 사용해 랜덤하게 다항함수를 몇 개 생성하였다. [연습문제 6.11]에 등장한 `approx_equal_function`을 사용하면 테스트를 통과함을 확인할 수 있다.

```
def random_function():
    degree = randint(0,5)
    p = Polynomial(*[uniform(-10,10) for _ in range(0,degree)])
    return Function(lambda x: p(x))
for i in range(0,100):
    a,b = random_scalar(), random_scalar()
    u,v,w = random_function(), random_function(), random_function()
    test(Function.zero(), approx_equal_function, a,b,u,v,w)
```

연습문제 | 6.13 미니 프로젝트

Function2(Vector) 클래스는 Function 클래스와 정의가 크게 다르진 않지만, 모든 함수가 두 개의 인자를 받는다는 점이 다르다.

```
class Function(Vector):
    def __init__(self, f):
        self.function = f
    def add(self, other):
        return Function(lambda x,y: self.function(x,y) + other.function(x,y))
    def scale(self, scalar):
        return Function(lambda x,y: scalar * self.function(x,y))
    @classmethod
    def zero(cls):
        return Function(lambda x,y: 0)
    def __call__(self, *args):
        return self.function(*args)
```

예를 들어 $f(x,y)=x+y$와 $g(x,y)=x-y+1$의 합은 $2x+1$이다. 이를 다음과 같이 확인할 수 있다.

```
>>> f = Function(lambda x,y:x+y)
>>> g = Function(lambda x,y: x-y+1)
>>> (f+g)(3,10)
7
```

연습문제 | 6.14

9×9 행렬은 성분이 81개다. 따라서 행렬을 결정하는 데 81개의 독립적인 수(또는 좌표)가 있다. 따라서 이 행렬로 이루어진 공간은 81차원 벡터공간이며 (d)가 정답이다.

연습문제 | 6.15 미니 프로젝트

```
class Matrix(Vector):
    @abstractproperty
    def rows(self):
        pass
    @abstractproperty
    def columns(self):
        pass
```

```
    def __init__(self,entries):
        self.entries = entries
    def add(self,other):
        return self.__class__(
            tuple(
                tuple(self.entries[i][j] + other.entries[i][j]
                        for j in range(0,self.columns()))
                for i in range(0,self.rows())))
    def scale(self,scalar):
        return self.__class__(
            tuple(
                tuple(scalar * e for e in row)
                for row in self.entries))
    def __repr__(self):
        return "%s%r" % (self.__class__.__qualname__, self.entries)
    def zero(self):
        return self.__class__(
            tuple(
                tuple(0 for i in range(0,self.columns()))
                for j in range(0,self.rows())))
```

연습문제 | 6.16

```
def random_matrix(rows, columns):
    return tuple(
        tuple(uniform(-10,10) for j in range(0,columns))
        for i in range(0,rows)
    )

def random_5_by_3():
    return Matrix5_by_3(random_matrix(5,3))

def approx_equal_matrix_5_by_3(m1,m2):
    return all([
        isclose(m1.matrix[i][j],m2.matrix[i][j])
        for j in range(0,3)
        for i in range(0,5)
    ])

for i in range(0,100):
    a,b = random_scalar(), random_scalar()
    u,v,w = random_5_by_3(), random_5_by_3(), random_5_by_3()
    test(Matrix5_by_3.zero(), approx_equal_matrix_5_by_3, a,b,u,v,w)
```

이제 고정된 크기의 행렬에 대한 벡터공간을 나타내는 어느 클래스라도 구현할 수 있다. 다음은 2×2 행렬 예시이다.

```
class Matrix2_by_2(Matrix):
    def rows(self):
        return 2
    def columns(self):
        return 2
```

그러면 2×2 행렬을 벡터로 간주하여 계산할 수 있다.

```
>>> 2 * Matrix2_by_2(((1,2),(3,4))) + Matrix2_by_2(((1,2),(3,4)))
Matrix2_by_2((3, 6), (9, 12))
```

연습문제 | 6.19

어떤 이미지를 선택하더라도 `ImageVector("my_image.jpg") + ImageVector.zero()`의 결과를 살펴보면 된다.

연습문제 | 6.20

예를 들어 다음 코드를 $s=0.1,\ 0.2, 0.3,\ ...,\ 0.9,\ 1.0$으로 두어 각각 실행시켜보자.

```
s * ImageVector("inside.JPG") + (1-s) * ImageVector("outside.JPG")
```

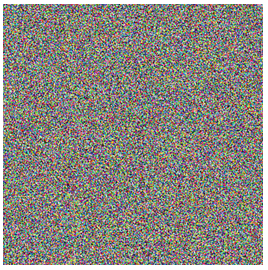
이미지를 연달아 배치하면 다음 그림과 유사한 결과를 얻을 것이다.

그림 가중치가 서로 다른 두 이미지의 가중평균

연습문제 | 6.21

랜덤한 이미지는 각 픽셀의 RGB값을 랜덤하게 설정해도 얻을 수 있다. 예를 들어 다음과 같이 할 수 있다.

```
def random_image():
    return ImageVector([(randint(0,255), randint(0,255), randint(0,255))
                            for i in range(0,300 * 300)])
```



이 결과는 아날로그 TV에 낀 노이즈처럼 뿌옇게 보이지만, 그 자체가 중요하진 않다. 단위 테스트에서는 각 픽셀을 비교해야 한다. 다음과 같은 근사적인 동등성 테스트를 통해 단위 테스트를 수행할 수 있다.

```
def approx_equal_image(i1,i2):
    return all([isclose(c1,c2)
        for p1,p2 in zip(i1.pixels,i2.pixels)
        for c1,c2 in zip(p1,p2)])

for i in range(0,100):
    a,b = random_scalar(), random_scalar()
    u,v,w = random_image(), random_image(), random_image()
    test(ImageVector.zero(), approx_equal_image, a,b,u,v,w)
```

6.3.7 연습문제 정답 및 풀이

연습문제 | 6.22

이 영역에 있는 점들로 일차결합을 하면 영역을 벗어나는 경우가 많이 있다. 더 명확한 증거로는 이 영역은 영벡터를 포함하지 않으므로 벡터공간일 수 없다. 영벡터는 (스칼라가 0인) 임의 벡터의 스칼라곱이다. 따라서 영벡터는 임의의 벡터공간과 그 부분공간에 포함되어야 한다.

연습문제 | 6.23

이 벡터들은 실수 y에 대해 $(0,y)$ 꼴이고 y축에 놓여 있다. $(0,y)$ 꼴 벡터에 대한 덧셈과 스칼라곱은 실수와 마찬가지이다. 0이 부가적으로 따라올 뿐이다. 이 벡터공간은 $\mathbb{R}$이 변장한 것이므로 1차원 벡터공간이라고 결론내릴 수 있다. 좀 더 엄밀하게 증명하고 싶다면 벡터공간에 대한 모든 성질을 명시적으로 확인해보자.

연습문제 | 6.24

$$(1,0) = \frac{1}{2} \cdot (1,1) - \frac{1}{2}(-1,1)$$
$$(1,1) = 2 \cdot (1,0) + (-1,1)$$
$$(-1,1) = (1,1) - 2 \cdot (1,0)$$

연습문제 | 6.25

$(1,0)$은 y좌표에 기여할 수 없으므로 일차결합에 $y(1,1)$이 포함되어야 한다. 식이 성립하려면 $(1,0)$이 $(x-y)$ 단위만큼 필요하다.

$$(x,y) = (x-y) \cdot (1,0) + y(1,1)$$

연습문제 | 6.26

벡터의 일차결합은 벡터공간 법칙에 따라 스칼라곱으로 단순화할 수 있다. 예를 들어 일차결합 $a \cdot \mathbf{v} + b \cdot \mathbf{v}$는 $(a+b) \cdot \mathbf{v}$와 같다.

연습문제 | 6.27

이 직선이 부분공간이 아닌 간단한 이유는 원점(영벡터)를 포함하지 않기 때문이다. 다른 이유는 원점을 지나지 않는 직선은 평행하지 않은 두 벡터를 점으로 가지기 때문이다. 이 두 벡터의 생성공간은 평면 전체이며, 직선보다 더 크다.

연습문제 | 6.28

집합 $\{\mathbf{e}_1, \mathbf{e}_2\}$의 생성공간은 일차결합 $a \cdot \mathbf{e}_1 + b \cdot \mathbf{e}_2 = a \cdot (1,0,0) + b \cdot (0,1,0) = (a,b,0)$으로 이루어져 있다. a와 b의 선택에 따라 $z=0$인 xy평면에 있는 임의의 점에 대응된다. 같은 논증으로 두 벡터 $\{\mathbf{e}_2, \mathbf{e}_3\}$는 $x=0$인 yz평면을 생성한다. 두 벡터 $\{\mathbf{e}_1, \mathbf{e}_3\}$는 $y=0$인 xz평면을 생성한다.

연습문제 | 6.29

x좌표에 $(-2,1)$만 기여할 수 있으므로 일차변환에는 $2.5 \cdot (-2,1)$이 있어야 한다. 이 벡터는 $(-5,2.5)$이므로 y좌표에 1.5단위를 더해야 한다. 즉 $0.5 \cdot (0,3)$을 더해야 한다. 따라서 일차변환은 다음과 같다.

$$(-5,4) = 0.5 \cdot (0,3) + 2.5 \cdot (-2,1)$$

연습문제 | 6.30 미니 프로젝트

알아내기 쉽지 않지만 앞의 두 벡터의 일차결합으로 세 번째 벡터를 도출할 수 있다.

$$-3 \cdot (1,2,0) + (5,0,5) = (2,-6,5)$$

여기서 세 번째 벡터가 중복이므로 이 벡터 집합은 일차종속이다. 이 집합은 3차원 공간 전체가 아니라 2차원 부분공간만을 생성한다.

연습문제 | 6.31

선형사상은 일차결합을 보존해야 한다는 정의를 직접 활용하자. f는 실수의 일차결합을 보존하지 않음을 확인하면 된다. 예를 들어 $f(1+1)=2a+b$이지만 $f(1)+f(1)=(a+b)+(a+b)=2a+2b$이다. 이 둘은 $b=0$이 아니라면 같지 않다.

다른 방법으로는 일차함수 $\mathbb{R}\to\mathbb{R}$이 1×1 행렬로 표현됨을 이용할 수 있다. 1차원 열벡터 $[x]$와 1×1 행렬 $[a]$를 곱하면 $[ax]$를 얻는다. 5장에서 구현한 내용으로 이 결과를 확인할 수 있다. 함수 $\mathbb{R}\to\mathbb{R}$가 선형사상이려면 1×1 행렬 곱셈과 일치해야 한다. 따라서 스칼라를 곱하는 것과도 일치해야 한다.

연습문제 | 6.32

Vec2가 보유한 데이터는 a와 b가 아니라 x와 y였다. 이 점만 제외하면 기능이 같다. 이제 해야 할 일은 `__call__`을 구현하는 것이다.

```
class LinearFunction(Vec2):
    def __call__(self,input):
        return self.x * input + self.y
```

연습문제 | 6.33

두 일차함수의 일차결합이 다른 일차함수임을 확인해야 한다. $f(x)=ax+b$이고 $g(x)=cx+d$일 때, $r\cdot f+s\cdot g$는 다음을 리턴한다.

$$r\cdot f+s\cdot g=r\cdot(ax+b)+s\cdot(cx+d)=rax+b+scx+d=(ra+sc)\cdot x+(b+d)$$

$(ra+sc)$와 $(b+d)$는 스칼라이므로 원하는 꼴이 되었다. 따라서 일차함수는 일차결합에 닫혀 있으며 부분공간을 만든다고 결론내릴 수 있다.

연습문제 | 6.34

3×3 행렬 9개로 이루어진 기저 중 한 사례는 다음과 같다.

$$\begin{pmatrix}1&0&0\\0&0&0\\0&0&0\end{pmatrix}\begin{pmatrix}0&1&0\\0&0&0\\0&0&0\end{pmatrix}\begin{pmatrix}0&0&1\\0&0&0\\0&0&0\end{pmatrix}$$

$$\begin{pmatrix}0&0&0\\1&0&0\\0&0&0\end{pmatrix}\begin{pmatrix}0&0&0\\0&1&0\\0&0&0\end{pmatrix}\begin{pmatrix}0&0&0\\0&0&1\\0&0&0\end{pmatrix}$$

$$\begin{pmatrix}0&0&0\\0&0&0\\1&0&0\end{pmatrix}\begin{pmatrix}0&0&0\\0&0&0\\0&1&0\end{pmatrix}\begin{pmatrix}0&0&0\\0&0&0\\0&0&1\end{pmatrix}$$

이 집합은 일차독립이다. 각 행렬이 임의의 일차결합에서 고유한 성분에만 기여하기 때문이다. 또한 이 집합의 일차결합으로 어떠한 3×3 행렬도 만들어낼 수 있기 때문에 공간 전체를 생성한다. 일차결합에서 이 행렬 중 하나와 곱한 계수는 결과 행렬의 한 성분만 결정하기 때문이다. 9개의 벡터가 3×3 행렬의 공간에 대한 기저이므로 이 공간은 9차원이다.

연습문제 | 6.35 미니 프로젝트

다음 구현은 `LinearFunction`과 많이 비슷하지만 계수는 2개가 아니라 3개 있으며, `__call__` 함수는 2차 항을 포함한다.

```
class QuadraticFunction(Vector):
    def __init__(self,a,b,c):
        self.a = a
        self.b = b
        self.c = c
    def add(self,v):
        return QuadraticFunction(self.a + v.a,
                                 self.b + v.b,
                                 self.c + v.c)
    def scale(self,scalar):
        return QuadraticFunction(scalar * self.a,
                                 scalar * self.b,
                                 scalar * self.c)
    def __call__(self,x):
        return self.a * x * x + self.b * x + self.c
    @classmethod
    def zero(cls):
        return QuadraticFunction(0,0,0)
```

덧붙여 ax^2+bx+c가 집합 $\{x^2, x, 1\}$의 일차결합과 비슷함에 주목할 수 있다. 게다가 집합 내의 세 함수는 공간을 생성하는데, 세 함수 중 어떤 것도 다른 두 함수의 일차결합으로 표현될 수 없다. 예를 들어 일차함수를 더해봤자 x^2 항이 될 수 없다. 따라서 이 집합은 기저의 한 사례이다. 벡터가 3개이므로 함수로 이루어진 공간에서 3차원 부분공간에 해당된다고 결론지을 수 있다.

연습문제 | 6.36 미니 프로젝트

$\frac{1}{9}\cdot(4x+1)-\frac{22}{9}\cdot(x-2)=-2x+5$이다. 여러분도 손으로 쉽게 확인할 수 있다.
만약 확인하지 못했어도 이런 교묘한 문제를 푸는 법은 다음 장에서 다룬다.

연습문제 | 6.37 미니 프로젝트

```
class Polynomial(Vector):
    def __init__(self, *coefficients):
        self.coefficients = coefficients
    def __call__(self,x):
        return sum(coefficient * x ** power
                   for (power,coefficient)
                   in enumerate(self.coefficients))
    def add(self,p):
        return Polynomial([a + b
                           for a,b
                           in zip(self.coefficients,
                                  p.coefficients)])
    def scale(self,scalar):
        return Polynomial([scalar * a
                           for a in self.coefficients])
        return "$ %s $" % (" + ".join(monomials))
    @classmethod
    def zero(cls):
        return Polynomial(0)
```

모든 다항함수 집합에 대한 한 기저는 무한집합 $\{1, x, x^2, x^3, x^4, \cdots\}$이다. x의 모든 거듭제곱을 쓸 수 있다면 임의의 다항함수를 일차결합으로 만들어낼 수 있다.

연습문제 | 6.38

두 번째 기저 벡터는 다음 위치에 1을 채워 넣어 만들 수 있다. 이미지의 왼쪽 가장 위에 위치한 픽셀이 희미한 녹색을 띨 것이다.

```
ImageVector([
    (0,1,0), (0,0,0), (0,0,0), ..., (0,0,0),
    (0,0,0), (0,0,0), (0,0,0), ..., (0,0,0),
    ...
])
```

두 번째 기저 벡터에서 1은 두 번째 슬롯으로 이동하였다.

다른 행은 비어있다.

연습문제 | 6.39

```
def solid_color(r,g,b):
    return ImageVector([(r,g,b) for _ in range(0,300*300)])
```

연습문제 | 6.40 미니 프로젝트

```
image_size = (300,300)
total_pixels = image_size[0] * image_size[1]
square_count = 30
square_width = 10

def ij(n):
    return (n // image_size[0], n % image_size[1])

def to_lowres_grayscale(img):

    matrix = [
        [0 for i in range(0,square_count)]
        for j in range(0,square_count)
    ]
    for (n,p) in enumerate(img.pixels):
        i,j = ij(n)
        weight = 1.0 / (3 * square_width * square_width)
        matrix[i // square_width][ j // square_width] += (sum(p) * weight)
    return matrix
```

그림을 30×30 크기 격자로 쪼갠다.

이 함수는 ImageVector를 입력으로 받아 30개의 값으로 이루어진 배열이 30개 있는 배열을 리턴한다. 이 2차원 배열의 각 성분은 사각형별 흑백 값을 나타낸다.

```
def from_lowres_grayscale(matrix):
    def lowres(pixels, ij):
        i,j = ij
        return pixels[i // square_width][ j // square_width]
    def make_highres(limg):
        pixels = list(matrix)
        triple = lambda x: (x,x,x)
        return ImageVector([triple(lowres(matrix, ij(n))) for n in
range(0,total_pixels)])
    return make_highres(matrix)
```

이 두 번째 함수는 30×30 행렬을 입력으로 받아 각 행렬 값이 나타내는 밝기로 칠해진 10×10 픽셀 블록으로 이미지를 생성한다.

`from_lowres_grayscale(to_lowres_grayscale(img))`을 호출하면 이 장에서 이미 보여준 방법에 따라 이미지 `img`가 변형된다.

7.1.4 연습문제 정답 및 풀이

연습문제 | 7.1

회전변환을 먼저 해야 한다. 그렇지 않으면 평행이동 벡터도 회전변환의 각만큼 회전하기 때문이다. 예를 들어 다음과 같은 코드로 나타낼 수 있다.

```
class PolygonModel():
    ...
    def transformed(self):
        rotated = [vectors.rotate2d(self.rotation_angle, v) for v in
self.points]
        return [vectors.add((self.x,self.y),v) for v in rotated]
```

연습문제 | 7.2

```
width, height = 400, 400
def to_pixels(x,y):
    return (width/2 + width * x / 20, height/2 - height * y / 20)
```

7.2.7 연습문제 정답 및 풀이

연습문제 | 7.3

한 가지 가능성은 $\mathbf{u}=\mathbf{0}=(0,0)$이다. 이 경우에 직선이 자동으로 원점을 지난다. 이때 점 $\mathbf{u}+0\cdot\mathbf{v}$는 $\mathbf{v}$에 관계없이 원점이다. $\mathbf{u}$와 $\mathbf{v}$가 서로 스칼라배라고 가정하자. 예를 들어 $\mathbf{u}=s\cdot\mathbf{v}$라고 하면 $\mathbf{u}-s\cdot\mathbf{v}=\mathbf{0}$이 직선 위에 있으므로 직선이 원점을 지난다.

연습문제 | 7.4

그렇지 않다. t값에 관계없이 $\mathbf{u}+t\cdot\mathbf{v}=\mathbf{u}+t\cdot(0,0)=\mathbf{u}$이기 때문이다. $\mathbf{v}=\mathbf{0}=(0,0)$일 때 $\mathbf{u}+t\cdot\mathbf{v}$ 꼴의 점은 $\mathbf{u}$이다.

연습문제 | 7.5

$\mathbf{v}=(-1,3)$을 $(2,-6)$처럼 $\mathbf{v}$의 스칼라배로 바꾸는 방법을 써보자. $(2,2)+t\cdot(-1,3)$ 꼴의 점은 $t=-2\cdot s$일 때 점 $(2,2)+s\cdot(2,-6)$과 일치한다. 또한 $\mathbf{u}$를 직선 상의 임의의 점으로 바꿀 수 있다. 직선 위에 $(2,2)+1\cdot(-1,3)=(1,5)$가 있기 때문에 $(1,5)+t\cdot(2,-6)$도 같은 직선을 나타내는 방정식이다.

연습문제 | 7.6

그렇지 않다. $a=b=0$이면 방정식은 $0\cdot x+0\cdot y=c$이고 직선을 나타내지 않는다. $c=0$이면 이 방정식은 항상 참이고, $c\neq 0$이면 이 방정식은 언제나 참이 아니다. 어느 쪽이든 x와 y의 관계를 설명하지 않으므로 직선을 나타내지 않는다.

연습문제 | 7.7

이 직선을 $6x+3y=9$로도 나타낼 수 있다. 사실 방정식의 양변에 0이 아닌 동일한 수를 곱하면 같은 직선에 대한 다른 방정식을 얻을 수 있다.

연습문제 | 7.8

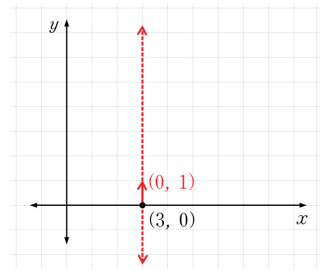
7.3.1절 내용을 참고하라.

연습문제 | 7.9

$2 \cdot 0 + 7 = 7$이고 $2 \cdot (3.5) + 0 = 7$이므로 주어진 방정식을 만족한다.

연습문제 | 7.10

$(3,0) + t \cdot (0,1)$은 수직선 $x = 3$을 생성한다.



$x = 3$은 이미 직선의 방정식의 표준형이지만, 공식에 대입해 확인할 수 있다. 이 직선의 첫 번째 점은 $(x_1, y_1) = (3,0)$이고 두 번째 점은 $(3,0) + (0,1) = (3,1) = (x_2, y_2)$이다. 따라서 다음이 성립한다.

$$a = y_2 - y_1 = 1,$$
$$b = x_1 - x_2 = 0,$$
$$c = x_1 y_2 - x_2 y_2 = 3 \cdot 1 - 1 \cdot 0 = 3$$

이를 대입하면 $1 \cdot x + 0 \cdot y = 3$ 또는 간단히 $x = 3$이다.

연습문제 | 7.11

앞에서 쓴 공식을 파이썬으로 번역하기만 하면 된다.

```
def standard_form(v1, v2):
    x1, y1 = v1
    x2, y2 = v2
    a = y2 - y1
    b = x1 - x2
    c = x1 * y2 - y1 * x2
    return a,b,c
```

연습문제 | 7.12 미니 프로젝트

실험을 더 쉽게 수행하고자 `do_segments_intersect`를 수정해서 4번의 검사 각각에 대한 `True/False`값을 리스트로 리턴하게 하자.

```
def segment_checks(s1,s2):
    u1,u2 = s1
    v1,v2 = s2
```

```
    l1, l2 = distance(*s1), distance(*s2)
    x,y = intersection(u1,u2,v1,v2)
    return [
        distance(u1, (x,y)) <= l1,
        distance(u2, (x,y)) <= l1,
        distance(v1, (x,y)) <= l2,
        distance(v2, (x,y)) <= l2
    ]
```

일반적으로 선분의 한 끝점과 다른 끝점 사이의 거리가 한 끝점과 교점 간의 거리보다 가까우면 검사가 실패한다.

두 직선 $y=0$과 $x=0$ 위의 선분을 사용해 원점에서 교차하는 다른 답을 찾아두었다. 각 답은 네 번의 검사 중 정확히 한 번에서만 실패한다. 의심스럽다면 어떻게 가능한지 살펴볼 수 있도록 그려보라.

```
>>> segment_checks(((-3,0),(-1,0)),((0,-1),(0,1)))
[False, True, True, True]
>>> segment_checks(((1,0),(3,0)),((0,-1),(0,1)))
[True, False, True, True]
>>> segment_checks(((-1,0),(1,0)),((0,-3),(0,-1)))
[True, True, False, True]
>>> segment_checks(((-1,0),(1,0)),((0,1),(0,3)))
[True, True, True, False]
```

연습문제 | 7.13

가장 높이 위치한 꼭짓점부터 반시계방향 순서대로 나열하면 꼭짓점은 $(2,7),(1,5),(2,3),(6,2),(6,4),(6,6),(4,6)$이다. 레이저 빔의 끝점은 $(1,1)$과 $(7,7)$로 가정할 수 있다.

```
>>> from asteroids import PolygonModel
>>> asteroid = PolygonModel([(2,7), (1,5), (2,3), (4,2), (6,2), (7,4), (6,6),
(4,6)])
>>> asteroid.does_intersect([(0,0),(7,7)])
True
```

이 결과를 통해 레이저가 소행성을 맞췄음을 확인할 수 있다! 반면에 $(0,0)$에서 $(0,7)$로 y축의 양의 방향으로 발사하면 소행성을 맞추지 못한다.

```
>>> asteroid.does_intersect([(0,0),(0,7)])
False
```

연습문제 | 7.14

먼저 PolygonModel에 다각형을 형성하는 (변환된) 선분을 리턴하는 작업을 중복해서 작성하지 않도록 segments() 메서드를 도입하는 게 편리하다. 그 뒤에 다른 다각형의 모든 선분과 현재 다각형을 대상으로 does_intersect를 수행했을 때 True를 리턴하는지 확인하면 된다.

```
class PolygonModel():
    ...
    def segments(self):
        point_count = len(self.points)
        points = self.transformed()
        return [(points[i], points[(i+1)%point_count])
                for i in range(0,point_count)]

    def does_collide(self, other_poly):
        for other_segment in other_poly.segments():
            if self.does_intersect(other_segment):
                return True
        return False
```

겹치는 정사각형과 겹치지 않는 정사각형을 만들어 does_collide 메서드가 제대로 탐지하는지를 테스트해볼 수 있다. 실제로 잘 동작한다.

```
>>> square1 = PolygonModel([(0,0), (3,0), (3,3), (0,3)])
>>> square2 = PolygonModel([(1,1), (4,1), (4,4), (1,4)])
>>> square1.does_collide(square2)
True
>>> square3 = PolygonModel([(-3,-3),(-2,-3),(-2,-2),(-3,-2)])
>>> square1.does_collide(square3)
False
```

연습문제 | 7.15 미니 프로젝트

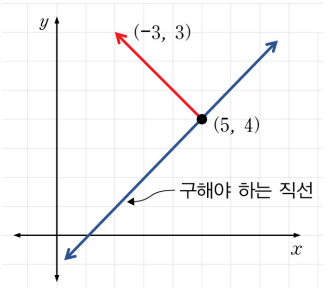
예를 들어 $\mathbf{w} = (0,0)$이면 연립일차방정식에 의해 표현되는 두 직선이 동일하다. 잘 모르겠다면 그래프를 그려봐라! 이때의 해는 임의의 실수 a에 대해 $\mathbf{v} = (a, -2a)$ 꼴이다. $\mathbf{w} = (0,0)$일 때 $\mathbf{v}$에 대한 무수히 많은 해 중 몇 개를 소개한다.

$$\begin{pmatrix}2 & 1\\ 4 & 2\end{pmatrix}\begin{pmatrix}1\\ -2\end{pmatrix} = \begin{pmatrix}2 & 1\\ 4 & 2\end{pmatrix}\begin{pmatrix}-4\\ 8\end{pmatrix} = \begin{pmatrix}2 & 1\\ 4 & 2\end{pmatrix}\begin{pmatrix}10\\ -20\end{pmatrix} = \begin{pmatrix}0\\ 0\end{pmatrix}$$

7.3.5 연습문제 정답 및 풀이

연습문제 | 7.16

그림을 그려 점 $(5,4)$를 지나고 $(-3,3)$에 수직인 직선의 방정식을 구해보자.



직선의 모든 점 (x,y)에 대해 벡터 $(x-5,y-4)$는 직선과 평행하므로 $(-3,3)$에 수직이다. 이는 $(x-5,y-4)$와 $(-3,3)$의 내적은 직선 위의 임의의 점 (x,y)에 대해 0임을 의미한다. 이 방정식을 전개하면 $-3x+15+3y-12=0$이고, 정리하면 $-3x+3y=-3$이다. 양변을 -3으로 나누면 동치이면서 더 단순한 방정식 $x-y=1$을 얻는다.

연습문제 | 7.17 미니 프로젝트

(a_1,a_2,a_3,a_4)와 (b_1,b_2,b_3,b_4)가 모두 해라면 두 해의 일차결합도 해임을 보일 수 있다. 이는 해집합이 이 벡터들의 모든 일차결합을 포함한다는 뜻으로, 해집합이 벡터 부분공간이 됨을 의미한다.

먼저 (a_1,a_2,a_3,a_4)와 (b_1,b_2,b_3,b_4)가 두 일차방정식을 만족하는 공통해라고 가정하자. 이는 다음 식이 성립함을 의미한다.

$$\begin{aligned} a_1+2a_2+2a_3+a_4&=0\\ b_1+2b_2+2b_3+b_4&=0\\ a_1-a_4&=0\\ b_1-b_4&=0 \end{aligned}$$

두 스칼라 c와 d를 선택해서 만든 일차결합 $c(a_1,a_2,a_3,a_4)+d(b_1,b_2,b_3,b_4)$는 $(ca_1+db_1, ca_2+db_2, ca_3+db_3, ca_4+db_4)$와 같다. 이 일차결합이 두 방정식의 해인지 알고 싶다면 x_1, x_2, x_3, x_4에 각 좌푯값을 대입해보면 확인할 수 있다.

첫 번째 방정식에서 $x_1+2x_2+2x_3+x_4$는 다음과 같다.

$$\begin{aligned}&(ca_1+db_1)+2(ca_2+db_2)+2(ca_3+db_3)+(ca_4+db_4)\\&=ca_1+db_1+2ca_2+2db_2+2ca_3+2db_3+ca_4+db_4 \quad \text{전개한다.}\\&=c(a_1+2a_2+2a_3+a_4)+d(b_1+2b_2+2b_3+b_4) \quad c,\ d\text{로 묶는다.}\\&=c\cdot 0+d\cdot 0=0 \quad a_1+2a_2+2a_3+a_4\text{와 } b_1+2b_2+2b_3+b_4 \text{ 모두 } 0\text{이다.}\end{aligned}$$

이 일차결합은 첫 번째 방정식의 해이다. 마찬가지로 두 번째 방정식에 이 일차결합을 대입해서 해가 됨을 확인할 수 있다.

$$(ca_1+db_1)-(ca_4+db_4)=c(a_1-a_4)+d(b_1-b_4)=c\cdot 0+d\cdot 0=0$$

임의의 두 해를 임의로 일차결합해도 해이기 때문에 해집합은 모든 일차결합을 포함한다. 따라서 이 해집합은 4차원의 벡터 부분공간이다.

연습문제 | 7.18

평면의 모든 점 (x,y,z)에 대해 벡터 $(x-1,y-1,z-1)$은 $(1,1,1)$에 수직이다. 평면 위의 어떤 점에 해당하는 임의의 x, y, z값에 대해서도 $(x-1,y-1,z-1)\cdot(1,1,1)=0$이다. 이 식을 전개하면 $(x-1)+(y-1)+(z-1)=0$이므로 평면에 대한 일차방정식의 표준형 $x+y+z=3$을 얻는다.

연습문제 | 7.19 미니 프로젝트

주어진 점을 p_1, p_2, p_3라고 하면 두 벡터 p_3-p_1과 p_2-p_1의 차는 평면과 평행하다. 따라서 두 벡터의 외적 $(p_2-p_1)\times(p_3-p_1)$은 평면에 수직이다. (세 점 p_1, p_2, p_3가 삼각형을 만들면 문제가 없다. 이 경우 두 벡터의 차는 평행하지 않기 때문이다.) 평면의 점 및 평면과 수직인 벡터가 있으면 일차방정식의 표준형을 구하는 작업을 반복하면 된다.

```
from vectors import *

def plane_equation(p1,p2,p3):
    parallel1 = subtract(p2,p1)
    parallel2 = subtract(p3,p1)
    a,b,c = cross(parallel1, parallel2)
    d = dot((a,b,c), p1)
    return a,b,c,d
```

예를 들어 다음은 [연습문제 7.18]에 등장한 평면 $x+y+z=3$의 세 점이다.

```
>>> plane_equation((1,1,1), (3,0,0), (0,3,0))
(3, 3, 3, 9)
```

결과로 나타난 (3,3,3,9)는 $3x+3y+3z=9$이며 $x+y+z=3$와 동치이다. 그 말은 제대로 했음을 의미한다!

연습문제 | 7.20

먼저 행렬 방정식을 축약하지 않고 써보자.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} & a_{16} & a_{17} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} & a_{26} & a_{27} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} & a_{36} & a_{37} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} & a_{46} & a_{47} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} & a_{56} & a_{57} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix}$$

좌변의 행렬에는 $5 \cdot 7 = 35$개의 항이 있으므로 이 연립일차방정식에는 a_{ij}가 35개 있다. 미지수인 변수는 $x_1, x_2, ..., x_7$으로 총 7개이며, 방정식은 행별로 하나씩 총 5개 있다. 행렬 곱셈을 하면 전체 연립일차방정식을 얻을 수 있다.

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + a_{14}x4 + a_{15}x_5 + a_{16}x_6 + a_{17}x_7 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + a_{24}x4 + a_{25}x_5 + a_{26}x_6 + a_{27}x_7 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + a_{34}x4 + a_{35}x_5 + a_{36}x_6 + a_{37}x_7 &= b_3 \\ a_{41}x_1 + a_{42}x_2 + a_{43}x_3 + a_{44}x4 + a_{45}x_5 + a_{46}x_6 + a_{47}x_7 &= b_4 \\ a_{51}x_1 + a_{52}x_2 + a_{53}x_3 + a_{54}x4 + a_{55}x_5 + a_{56}x_6 + a_{57}x_7 &= b_5 \end{aligned}$$

이렇게 모두 쓰면 지루하므로 축약해서 나타내는 게 좋다.

연습문제 | 7.21

이 방정식의 좌변은 i가 1에서 3까지인 x_i 꼴 항을 더한 것이다. 따라서 $x_1+x_2+x_3=1$이다. 이 식은 변수가 3개인 일차방정식의 표준형이며 그 해집합은 3차원 공간에서 평면을 이룬다.

연습문제 | 7.22

다음은 세 평면 $z+y=0$, $z-y=0$, $z=3$의 그래프이다.

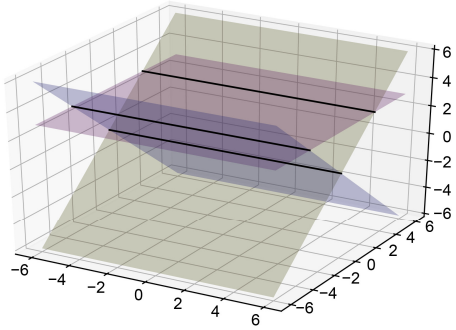


그림 세 평면이 한 교점을 갖지 않고 그 중 어느 두 평면도 평행하지 않은 경우

그림에 세 평면 중 두 평면의 교집합을 표시했는데, 이 교집합들은 평행선이다. 이 직선들은 절대 만나지 않기 때문에 세 평면 모두가 교차하는 단일 점은 없다. 이 예시는 6장에서 살펴본 바와 비슷하다. 세 벡터 중 어느 두 벡터끼리 평행하지 않더라도 세 벡터는 일차종속일 수 있다.

연습문제 | 7.23

(a) 두 개의 일차방정식과 두 개의 미지수가 있을 때, 유일한 해가 **있을 수도 있다**. 두 방정식은 평면에서 각각 직선을 나타내며, 두 직선이 평행하지 않으면 한 점에서 교차하기 때문이다.

(b) 두 개의 일차방정식과 일곱 개의 미지수가 있을 때, 유일한 해는 **있을 수 없다**. 각 방정식으로 정의되는 6차원 초평면이 서로 평행하지 않아도 해공간이 5차원이기 때문이다.

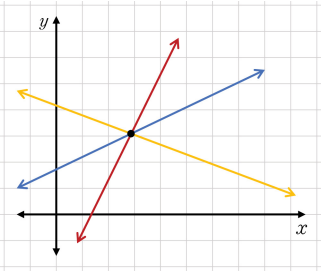


그림 한 점에서 교차하는 평면의 세 선

(c) 다섯 개의 일차방정식과 다섯 개의 미지수가 있을 때, 유일한 해가 **있을 수도 있다**. 단, 이 연립일차방정식이 독립일 때에만 해가 유일하다.

(d) 세 개의 일차방정식과 두 개의 미지수가 있을 때, 유일한 해가 **있을 수도 있다**. 세 번째 직선이 두 직선의 교점을 지나야 하는데, 드물긴 하지만 그런 일이 있을 수도 있다.

연습문제 | 7.24

세 평면 $z-y=0$, $z+y=0$, $z+x=0$은 단일 점 $(0,0,0)$에서 교차한다. 임의로 선택한 평면은 대부분 이와 같이 유일한 점에서 교차한다.

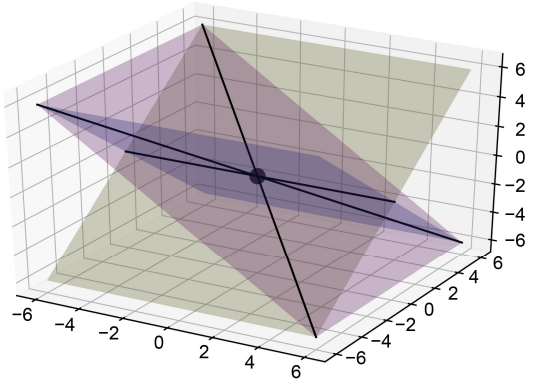


그림 단일 점에서 교차하는 세 평면

세 평면 $z-y=0$, $z+y=0$, $z=0$의 교집합은 직선(x축)이다. 이 방정식을 풀어보면 y와 z 모두 0이 되어야 하지만 x는 모든 식에 등장하지 않으므로 아무런 제약을 받지 않는다. 따라서 x축 위의 임의의 벡터 $(x,0,0)$은 이 연립일차방정식의 해이다.

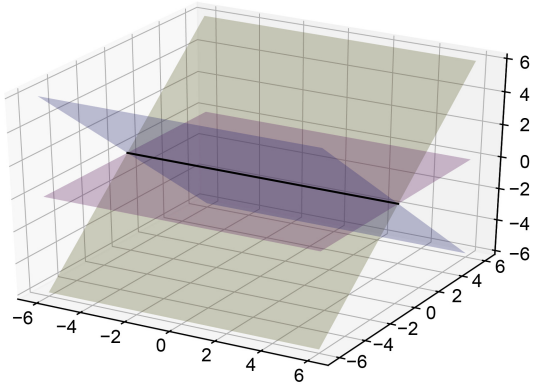


그림 교집합이 직선을 이루는 세 평면

마지막으로, 세 방정식이 모두 같은 평면을 나타내면 그 전체 평면은 하나의 해 집합이다. 예를 들어 $z-y=0$, $2z-2y=0$, $3z-3y=0$은 모두 동일한 평면을 나타낸다. 이러한 연립일차방정식의 해집합은 평면 전체이다.

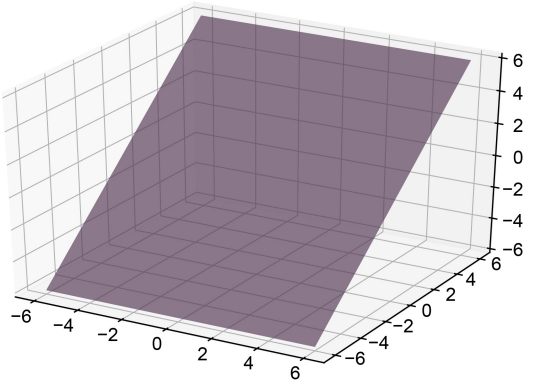


그림 모두 겹쳐진 세 개의 동일한 평면

연습문제 | 7.25

일차방정식 4개가 좌푯값을 명시하므로 해가 $(0,1,x_3,-1,3)$ 꼴임을 알 수 있다. x_3값을 구하려면 마지막 방정식을 풀어야 한다. $x_1+x_2+x_3=-2$이므로 $0+1+x_3=-2$이고, 따라서 x_3는 -3이다. 따라서 유일한 해는 $(0,1,-3,-1,3)$이다. 이 연립방정식을 행렬 꼴로 변환하면 NumPy로 이 연립방정식을 풀고 답이 맞음을 확인할 수 있다.

```
>>> matrix =
np.array(((0,0,0,0,1),(0,1,0,0,0),(0,0,0,1,0),(1,0,0,0,0),(1,1,1,0,0)))
>>> vector = np.array((3,1,-1,0,-2))
>>> np.linalg.solve(matrix,vector)
array([ 0., 1., -3., -1., 3.])
```

연습문제 | 7.26 미니 프로젝트

NumPy를 사용해 주어진 행렬의 역행렬을 계산한다.

```
>>> matrix = np.array(((1,1,-1),(0,2,-1),(1,0,1)))
>>> vector = np.array((-1,3,2))
>>> inverse = np.linalg.inv(matrix)
>>> inverse
array([[ 0.66666667, -0.33333333,  0.33333333],
       [-0.33333333,  0.66666667,  0.33333333],
       [-0.66666667,  0.33333333,  0.66666667]])
```

이 역행렬과 원래 행렬을 곱하면 대각성분은 1이고 나머지는 0인 항등행렬을 얻지만 수치 오차가 약간 발생할 수 있다.

```
>>> np.matmul(inverse,matrix)
array([[ 1.00000000e+00,  1.11022302e-16, -1.11022302e-16],
       [ 0.00000000e+00,  1.00000000e+00,  0.00000000e+00],
       [ 0.00000000e+00,  0.00000000e+00,  1.00000000e+00]])
```

연립일차방정식의 해를 구하는 묘책은 바로 행렬 방정식의 양변에 이 역행렬을 곱하는 것이다. 여기서는 가독성을 고려해 역행렬의 값을 근사하였다. 아래 식에서 좌변의 첫 번째 곱셈은 행렬과 그 역행렬의 곱이므로 쉽게 정리할 수 있다.

$$\begin{pmatrix} 0.667 & -0.333 & 0.333 \\ -0.333 & 0.667 & 0.333 \\ -0.667 & 0.333 & 0.667 \end{pmatrix} \begin{pmatrix} 1 & -1 & 0 \\ 0 & -1 & -1 \\ 1 & 0 & 2 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0.667 & -0.333 & 0.333 \\ -0.333 & 0.667 & 0.333 \\ -0.667 & 0.333 & 0.667 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0.667 & -0.333 & 0.333 \\ -0.333 & 0.667 & 0.333 \\ -0.667 & 0.333 & 0.667 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}$$

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 0.667 & -0.333 & 0.333 \\ -0.333 & 0.667 & 0.333 \\ -0.667 & 0.333 & 0.667 \end{pmatrix} \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}$$

이를 통해 해 (x,y,z)를 명확히 구할 수 있다. 이제 행렬과 벡터 간의 곱셈 하나만 남았다. `numpy.matmul`은 행렬과 벡터의 곱셈도 수행할 수 있다.

```
>>> np.matmul(inverse, vector)
array([-1., 3., 3.])
```

이 결과는 7.3.2절에서 풀이 프로그램을 통해 구한 답과 같다.

7.4.2 연습문제 정답 및 풀이

연습문제 | 7.27

이 문제는 다음 방정식이 참이 되는 a, b값을 찾는 문제와 동치이다.

$$a\begin{pmatrix}10\\1\end{pmatrix}+b\begin{pmatrix}3\\2\end{pmatrix}=\begin{pmatrix}5\\5\end{pmatrix}$$

또는 다음 행렬 방정식을 만족시키는 벡터 (a,b)를 찾는 문제와 동치이다.

$$\begin{pmatrix}10 & 3\\1 & 2\end{pmatrix}\begin{pmatrix}a\\b\end{pmatrix}=\begin{pmatrix}5\\5\end{pmatrix}$$

이 방정식의 해는 NumPy로 구할 수 있다.

```
>>> matrix = np.array(((10,3),(1,2)))
>>> vector = np.array((5,5))
>>> np.linalg.solve(matrix,vector)
array([-0.29411765, 2.64705882])
```

이는 구해야 할 일차결합이 다음과 같다는 뜻이다. 직접 확인해보라!

$$-0.29411765\cdot\begin{pmatrix}10\\1\end{pmatrix}+2.64705882\cdot\begin{pmatrix}3\\2\end{pmatrix}=\begin{pmatrix}5\\5\end{pmatrix}$$

연습문제 | 7.28

풀어야 할 연립일차방정식은 다음과 같다.

$$\begin{pmatrix}0 & 0 & 1 & 0\\0 & -2 & -2 & 0\\1 & -1 & 0 & -2\\1 & -1 & 2 & 1\end{pmatrix}\begin{pmatrix}a\\b\\c\\d\end{pmatrix}=\begin{pmatrix}3\\0\\6\\9\end{pmatrix}$$

여기서 4×4 행렬의 각 열은 만들고자 하는 일차결합에 포함된 벡터이다. NumPy는 이 연립일차방정식의 해를 알려준다.

```
>>> matrix = np.array(((0, 0, 1, 0), (0, -2, -2, 0), (1, -1, 0, -2), (1, -1, 2,
1)))
>>> vector = np.array((3,0,6,9))
>>> np.linalg.solve(matrix,vector)
array([ 1., -3., 3., -1.])
```

이는 구하려는 일차결합이 다음과 같음을 의미한다.

$$1 \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} - 3 \cdot \begin{pmatrix} 0 \\ -2 \\ -1 \\ -1 \end{pmatrix} + 3 \cdot \begin{pmatrix} 1 \\ -2 \\ 0 \\ 2 \end{pmatrix} - \begin{pmatrix} 0 \\ 0 \\ -2 \\ 1 \end{pmatrix} = \begin{pmatrix} 3 \\ 0 \\ 6 \\ 9 \end{pmatrix}$$

8.1.4 연습문제 정답 및 풀이

연습문제 | 8.1

4.5시간에 걸쳐 이동한 총 거리는 77905 − 77641 = 264마일이다. 따라서 평균 속력은 264마일/4.5시간, 즉 약 58.7mph이다.

연습문제 | 8.2

```
def secant_line(f,x1,x2):
    def line(x):
        return f(x1) + (x-x1) * (f(x2)-f(x1))/(x2-x1)
    return line
```

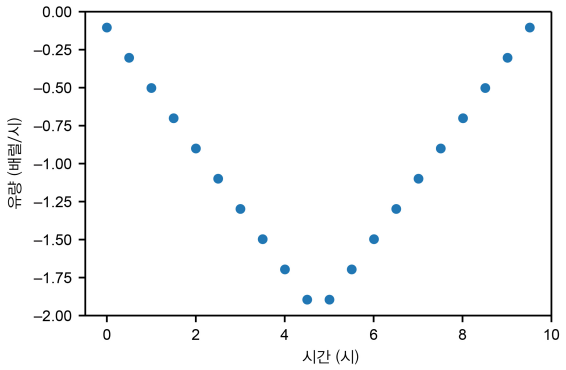
연습문제 | 8.3

```
def plot_secant(f,x1,x2,color='k'):
    line = secant_line(f,x1,x2)
    plot_function(line,x1,x2,c=color)
    plt.scatter([x1,x2],[f(x1),f(x2)],c=color)
```

8.2.3 연습문제 정답 및 풀이

연습문제 | 8.4

`plot_interval_flow_rates(decreasing_volume,0,10,0.5)`를 실행해보면 평균 유량이 5시 시점 직전에 가장 낮음(절댓값이 가장 큰 음수가 됨)을 알 수 있다.

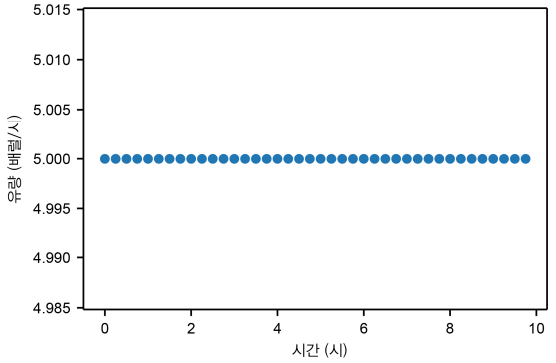


연습문제 | 8.5

`linear_volume_function(t)`는 상수 a와 b에 대한 $V(t) = at + b$ 꼴이다. 예를 들어 다음과 같은 코드를 실행하자.

```
def linear_volume_function(t):
    return 5*t + 3

plot_interval_flow_rates(linear_volume_function,0,10,0.25)
```



이 그래프는 부피가 시간에 따라 일차함수로 증가하면 시간에 따른 평균 유량은 일정함을 보여준다.

8.3.4 연습문제 정답 및 풀이

연습문제 | 8.6

이 함수가 직선이었다면 모든 점에서 할선과 같은 값을 가진다. 하지만 0.999시에서 1.001시까지 할선의 값은 $t=1$시에서 volume 함수의 값과는 다르다.

```
>>> volume(1)
2.878125
>>> secant_line(volume,0.999,1.001)(1)
2.8781248593749997
```

연습문제 | 8.7

```
>>> average_flow_rate(volume,7.9,8.1)
0.7501562500000007
>>> average_flow_rate(volume,7.99,8.01)
0.750001562499996
>>> average_flow_rate(volume,7.999,8.001)
0.7500000156249458
>>> average_flow_rate(volume,7.9999,8.0001)
0.7500000001554312
```

$t=8$에서 순간변화율은 0.75배럴/시로 보인다.

연습문제 | 8.8

구간을 작게 만들수록 해당 구간에서 할선의 기울기는 특정한 수로 수렴하지 않고 계속 커지기만 한다.

```
>>> average_flow_rate(sign,-0.1,0.1)
10.0
>>> average_flow_rate(sign,-0.01,0.01)
100.0
>>> average_flow_rate(sign,-0.001,0.001)
1000.0
>>> average_flow_rate(sign,-0.000001,0.000001)
1000000.0
```

이는 sign 함수가 $x=0$에서 -1에서 1로 급격히 증가하기 때문이다. 이로 인해 확대하더라도 직선처럼 보이진 않는다.

8.4.4 연습문제 정답 및 풀이

연습문제 | 8.9

처음 6시간 동안 석유가 탱크에 약 1.13배럴 주입되는데, 마지막 4시간 동안 탱크에 주입된 약 3.24배럴보다 작다.

```
>>> volume_change(flow_rate,0,6,0.01)
1.1278171874999996
>>> volume_change(flow_rate,6,10,0.01)
3.2425031249999257
```

9.1.4 연습문제 정답 및 풀이

연습문제 | 9.1

지금 이 순간에 $x'(t) = v_x = -3$이므로 소행성은 x축의 음의 방향인 왼쪽으로 이동하는 중이다. 동시에 $y'(t) = v_y = 1$이므로 소행성은 y축의 양의 방향인 위쪽으로도 이동하는 중이다. 따라서 **(b)**가 정답이다.

9.4.1 연습문제 정답 및 풀이

연습문제 | 9.2 미니 프로젝트

여러 답을 쉽게 확인할 수 있도록 총 시간과 타임스텝 개수도 매개변수로 포함했다.

```
def eulers_method(s0,v0,a,total_time,step_count):
    trajectory = [s0]
    s = s0
    v  = v0
    dt = total_time/step_count
    for _ in range(0,step_count):
        s = add(s,scale(dt,v))
        v  = add(v,scale(dt,a))
        trajectory.append(s)
    return trajectory
```

각 타임스텝 dt의 길이는 전체 경과시간을 타임스텝 개수로 나눈 것이다.

매 타임스텝에서 위치와 속도를 갱신한 뒤, 위치의 리스트인 궤적(trajectory)에 최신 위치를 다음 위치로 추가한다.

연습문제 | 9.3 미니 프로젝트

[연습문제 9.2]의 eulers_method 함수 구현에서 s와 v의 갱신 순서만 바꿔주면 된다.

```
def eulers_method_overapprox(s0,v0,a,total_time,step_count):
    trajectory = [s0]
    s = s0
    v  = v0
    dt = total_time/step_count
    for _ in range(0,step_count):
        v  = add(v,scale(dt,a))
        s = add(s,scale(dt,v))
        trajectory.append(s)
    return trajectory
```

동일한 입력을 놓고 보면 이 함수는 원래 구현한 값에 비해 더 커진 y좌표의 근삿값을 준다. 다음 그림에서 궤적을 자세히 살펴보면 첫 번째 타임스텝에서부터 객체가 y방향으로 이미 움직이고 있음을 볼 수 있다.

```
eulers_method_overapprox((0,0),(1,0),(0,0.2),10,10)
```

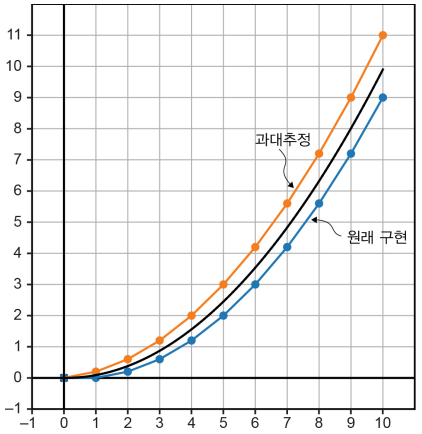


그림 오일러 방법으로 처음 구현한 궤적과 새로 구현한 궤적. 비교하고자 실제 궤적을 검은 선으로 표시했다.

연습문제 | 9.4 미니 프로젝트

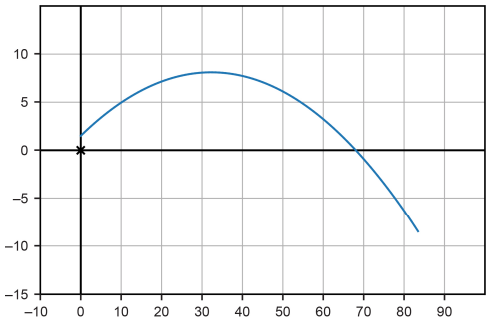
야구공의 초기 속도는 $(30 \cdot \cos(20°),\ 30 \cdot \sin(20°))$이다. [연습문제 9.2]의 `eulers_ method` 함수를 사용해 야구공의 움직임을 시뮬레이션해보자.

```
from math import pi,sin,cos
angle = 20 * pi/180
s0 = (0,1.5)
v0 = (30*cos(angle),30*sin(angle))

a = (0,-9.81)

result = eulers_method(s0,v0,a,3,100)
```

다음 그림에 결과로 얻은 궤적을 플로팅했다. 야구공은 공중에서 포물선을 그린 뒤 x축의 양의 방향으로 약 67m 떨어진 대지에 떨어졌다. 다만 야구공이 대지에 도착하면 멈추도록 설정하지 않았으므로 이 궤적은 지하로 계속 내려간다.



연습문제 | 9.5 미니 프로젝트

서로 다른 각도를 시뮬레이션하는 작업을 함수로 패키징할 수 있다. 초기 위치를 $(0,0)$으로 두고 다음 그림과 같이 여러 궤적을 살펴볼 수 있다. 야구공은 $45°$일 때 가장 멀리 나아감을 알 수 있다. 참고로 야구공이 지면에 닿기 직전의 움직임만 살펴보기 위해 y성분이 음인 궤적은 모두 걸러냈다.

```
def baseball_trajectory(degrees):
    radians = degrees * pi/180
    s0 = (0,0)
    v0 = (30*cos(radians),30*sin(radians))
    a = (0,-9.81)
    return [(x,y) for (x,y) in eulers_method(s0,v0,a,10,1000) if y>=0]
```

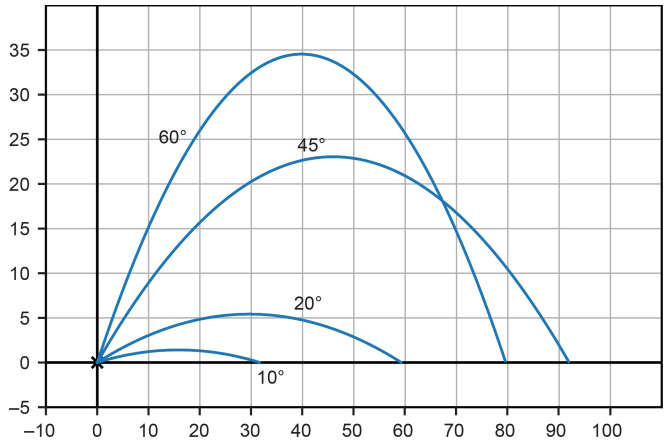
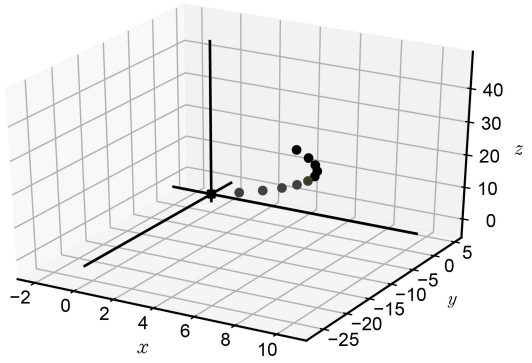


그림 초기 속력이 $30\ \mathrm{m/s}$일 때 다양한 각도로 야구공 던지기

연습문제 | 9.6 미니 프로젝트

우리가 구현한 `eulers_method`는 3차원 벡터도 다룰 수 있다! 다음 코드 조각으로 만든 그림은 3차원 궤적을 보여준다.

```
from draw3d import *
traj3d = eulers_method((0,0,0), (1,2,0), (0,-1,1), 10, 10)
draw3d(
    Points3D(*traj3d)
)
```



정확도를 높이기 위해 타임스텝을 1,000개로 두고 실행하면 마지막 위치를 찾을 수 있다.

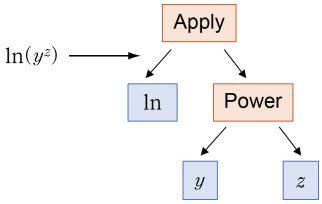
```
>>> eulers_method((0,0,0), (1,2,0), (0,-1,1), 10, 1000)[-1]
(9.999999999999831, -29.949999999999644, 49.94999999999933)
```

결과는 $(10,-30,50)$에 매우 가깝다. 실제로도 10초 후 객체의 정확한 위치는 $(10,-30,50)$이다.

10.2.4 연습문제 정답 및 풀이

연습문제 | 10.1

가장 바깥쪽에는 `Apply` 컴비네이터가 있다. 적용하는 함수는 자연로그 $\ln$이며 인자는 y^z이다. 한편, y^z는 밑이 y이고 지수가 z인 거듭제곱(`Power`)이다. 그 결과는 다음과 같다.



연습문제 | 10.2

$\ln(y^z)$은 두 변수 y와 z의 함수로 볼 수 있다. 파이썬으로 바로 번역하면 다음과 같은데, 파이썬에서는 $\ln$은 `log`라고 한다.

```
from math import log
def f(y,z):
    return log(y**z)
```

수식 트리는 다음과 같이 작성한다.

```
Apply(Function("ln"), Power(Variable("y"), Variable("z")))
```

연습문제 | 10.3

이 수식은 $3 \cdot (y+z)$를 나타낸다. 연산 순서를 고려하여 괄호를 표기하였다.

연습문제 | 10.4

Quotient 컴비네이터는 피제수(numerator, 분모)라고 하는 위의 수식과 제수(denominator, 분자)라는 아래 수식으로 이루어진 두 수식을 저장해야 한다.

```
class Quotient():
    def __init__(self,numerator,denominator):
        self.numerator = numerator
        self.denominator = denominator
```

문제에 주어진 수식은 합 $a+b$와 수 2의 나눗셈이다.

```
Quotient(Sum(Variable("a"),Variable("b")),Number(2))
```

연습문제 | 10.5

Difference 컴비네이터는 두 수식을 저장하고, 첫 번째 수식에 두 번째 수식을 뺀 결과를 나타낸다.

```
class Difference():
    def __init__(self,exp1,exp2):
        self.exp1 = exp1
        self.exp2 = exp2
```

문제에 주어진 수식 b^2-4ac는 수식 b^2과 수식 $4ac$의 차이며 다음과 같이 나타낼 수 있다.

```
Difference(
    Power(Variable('b'),Number(2)),
    Product(Number(4),Product(Variable('a'), Variable('c'))))
```

연습문제 | 10.6

Negative 컴비네이터는 수식 1개만 입력으로 받는 클래스이다.

```
class Negative():
    def __init__(self,exp):
        self.exp = exp
```

x^2+y를 부정하려면 이 식을 Negative 생성자로 넘겨주면 된다.

```
Negative(Sum(Power(Variable("x"),Number(2)),Variable("y")))
```

연습문제 | 10.7

타이핑할 내용을 줄이기 위해 각 변수 및 제곱근 함수에 먼저 이름을 붙이자.

```
A = Variable('a')
B = Variable('b')
C = Variable('c')
Sqrt = Function('sqrt')
```

그러면 이제 주어진 대수식을 원소와 컴비네이터로 구조로 적절하게 만들어 번역하기만 하면 된다. 맨 위에서 내려다보면 이 구조는 합(분모)과 곱(분자)의 나눗셈이다.

```
Quotient(
    Sum(
        Negative(B),
        Apply(
            Sqrt,
            Difference(
                Power(B,Number(2)),
                Product(Number(4), Product(A,C))))),
    Product(Number(2), A))
```

연습문제 | 10.8 미니 프로젝트

10장 소스 코드에서 expressions.py 파일을 살펴보기 바란다.

10.3.4 연습문제 정답 및 풀이

연습문제 | 10.9

distinct_variables의 결과를 이용하면 변수가 등장하는지 여부를 쉽게 체크할 수 있지만, 여기서는 처음부터 구현해보자.

```
def contains(exp, var):
    if isinstance(exp, Variable):
        return exp.symbol == var.symbol
    elif isinstance(exp, Number):
        return False
    elif isinstance(exp, Sum):
        return any([contains(e,var) for e in exp.exps])
    elif isinstance(exp, Product):
        return contains(exp.exp1,var) or contains(exp.exp2,var)
    elif isinstance(exp, Power):
        return contains(exp.base, var) or contains(exp.exponent, var)
    elif isinstance(exp, Apply):
        return contains(exp.argument, var)
    else:
        raise TypeError("Not a valid expression.")
```

연습문제 | 10.10

이 구현은 10.3.1절의 distinct_variables 함수와 매우 닮았다.

```
def distinct_functions(exp):
    if isinstance(exp, Variable):
        return set()
    elif isinstance(exp, Number):
        return set()
    elif isinstance(exp, Sum):
        return set().union(*[distinct_functions(exp) for exp in exp.exps])
    elif isinstance(exp, Product):
        return distinct_functions(exp.exp1).union(distinct_functions(exp.exp2))
    elif isinstance(exp, Power):
        return distinct_functions(exp.base).union(distinct_functions(exp.exponent))
    elif isinstance(exp, Apply):
        return set([exp.function.name]).union(distinct_functions(exp.argument))
    else:
        raise TypeError("Not a valid expression.")
```

연습문제 | 10.11

```
def contains_sum(exp):
    if isinstance(exp, Variable):
        return False
    elif isinstance(exp, Number):
        return False
    elif isinstance(exp, Sum):
        return True
    elif isinstance(exp, Product):
        return contains_sum(exp.exp1) or contains_sum(exp.exp2)
    elif isinstance(exp, Power):
        return contains_sum(exp.base) or contains_sum(exp.exponent)
    elif isinstance(exp, Apply):
        return contains_sum(exp.argument)
    else:
        raise TypeError("Not a valid expression.")
```

연습문제 | 10.12 미니 프로젝트

10장의 단계별 주피터 노트북을 살펴보거나 파이썬 클래스에서 `__repr__`나 다른 특수 메서드에 대해 논의하는 [부록 B]를 살펴보라.

연습문제 | 10.13 미니 프로젝트

10장에 대한 단계별 노트북을 참고하라.

연습문제 | 10.14 미니 프로젝트

구현은 단계별 노트북을 살펴보라. 구현한 뒤에는 다음과 같이 실행해볼 수 있다.

```
>>> Power(Variable("x"),Number(2))._python_expr()
'(x) ** (2)'
>>> Power(Variable("x"),Number(2)).python_function(x=3)
9
```

10.4.5 연습문제 정답 및 풀이

연습문제 | 10.15

```
def p(x):
    return x**5
plot_function(derivative(p), 0, 1)
plot_function(lambda x: 5*x**4, 0, 1)
```

다음 그림에서 볼 수 있듯이 두 그래프는 정확히 겹친다.

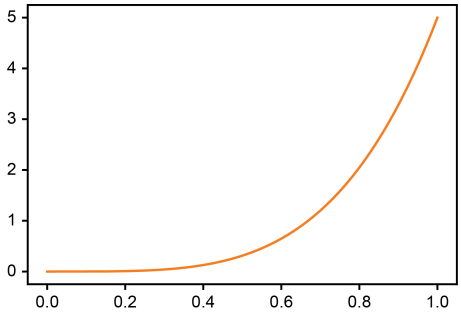


그림 $5x^4$의 그래프와 x^5의 (순간변화율을 근사한) 도함수

연습문제 | 10.16 미니 프로젝트

f와 g를 벡터로 보면 두 함수를 더하고 스칼라곱할 수 있다. $(f+g)(x)=f(x)+g(x)$이고 $(c\cdot f)(x)=c\cdot f(x)$임을 떠올려보기 바란다. 또한 **일차변환**은 벡터합과 스칼라곱을 보존하는 변환이다.

도함수를 취하는 연산(the operation of taking derivative)을 연산자(operator) D로 표기할 때, D는 함수를 입력으로 받아 도함수를 출력으로 리턴한다고 생각할 수 있다. $Df=f'$라고 표기하면 그렇게 보인다. 두 함수의 합에 대한 도함수는 각 함수의 도함수의 합인데, 다시 말해 다음이 성립한다.

$$D(f+g)=Df+Dg$$

어떤 함수에 수 c를 곱해 만든 함수의 도함수는 c와 원래 함수의 도함수를 곱한 것이다.

$$D(c\cdot f)=c\cdot Df$$

이 두 규칙은 D가 일차변환임을 의미한다. 특히 두 함수의 일차결합의 도함수가 각 도함수의 일차결합과 같음에 주목하라.

$$D(a\cdot f+b\cdot g)=a\cdot Df+b\cdot Dg$$

연습문제 | 10.17 미니 프로젝트

$g(x)^{-1}$의 도함수는 합성함수의 미분법에 의해 $-g(x)^{-2} \cdot g'(x)$이다. 이를 다시 쓰면 다음과 같다.

$$-\frac{g'(x)}{g(x)^2}$$

몫 $f(x)/g(x)$의 도함수는 곱 $f(x) \cdot g(x)^{-1}$의 도함수와 같다. 따라서 곱의 미분법을 적용하면 다음과 같다.

$$f'(x)g(x)^{-1} - \frac{g'(x)}{g(x)^2}f(x) = \frac{f'(x)}{g(x)} - \frac{f(x)g'(x)}{g(x)^2}$$

첫 번째 항의 분모, 분자에 $g(x)$를 곱하면 두 항의 분모가 같아지므로 두 항을 더할 수 있다.

$$\frac{f'(x)}{g(x)} - \frac{f(x)g'(x)}{g(x)^2} = \frac{f'(x)g(x)}{g(x)^2} - \frac{f(x)g'(x)}{g(x)^2} = \frac{f'(x)g(x) - f(x)g'(x)}{g(x)^2}$$

연습문제 | 10.18

이 식에는 두 번의 곱셈이 있는데, 다행히도 어느 순서로 곱의 미분법을 적용해도 같은 결과를 얻는다. $\sin(x) \cdot \cos(x)$의 도함수는 다음과 같다.

$$\sin(x) \cdot (-\sin(x)) + \cos(x) \cdot \cos(x) = \cos(x)^2 - \sin(x)^2$$

$\ln(x)$의 도함수는 $1/x$이고 곱의 미분법을 이용하면 전체 곱의 도함수는 다음과 같다.

$$\ln(x)(\cos(x)^2 - \sin(x)^2) + \frac{\sin(x)\cos(x)}{x}$$

연습문제 | 10.19

합성함수의 미분법을 두 번 적용해야 한다. 합성함수의 미분법의 앞 부분은 $f'(g(h(x)))$인데, 여기에 $g(h(x))$의 도함수를 곱해야 한다. 이 도함수는 $g'(h(x))$에 안쪽 $h(x)$의 도함수를 곱한 것이다. $g(h(x))$의 도함수는 $h'(x) \cdot g'(h(x))$이므로, $f(g(h(x)))$의 도함수는 $h'(x) \cdot g'(h(x)) \cdot f'(g(h(x)))$이다.

10.5.4 연습문제 정답 및 풀이

연습문제 | 10.20

곱을 이루는 두 수식 중 하나가 Number 클래스의 인스턴스인지만 확인하면 된다. 일반적으로는 곱의 각 항이 도함수의 대상이 되는 변수를 포함하는지 확인하는 방법이 있다. 예를 들어 x에 대해 $(3+\sin(5^a))f(x)$의 도함수를 구할 때에는 앞의 항에 x가 없으므로 곱의 미분법을 사용하지 않아도 된다. 따라서 앞의 항을 (x에 대해) 미분하면 0이다. 기존 연습문제에서 등장한 contains(expression, variable)을 사용해 이를 확인할 수 있다.

```
class Product(Expression):
    ...
    def derivative(self,var):
        if not contains(self.exp1, var):
            return Product(self.exp1, self.exp2.derivative(var))
        elif not contains(self.exp2, var):
            return Product(self.exp1.derivative(var), self.exp2)
        else:
            return Sum(
                Product(self.exp1.derivative(var), self.exp2),
                Product(self.exp1, self.exp2.derivative(var)))
```

만약 첫 번째 수식이 변수에 따라 변하지 않으면 두 번째 수식의 도함수와 첫 번째 수식의 곱을 리턴한다.

두 번째 수식이 변수에 따라 변하지 않으면 첫 번째 수식의 도함수와 수정하지 않은 두 번째 수식의 곱을 리턴한다.

둘 다 아니면, 일반적인 곱의 미분법을 적용한다.

연습문제 | 10.21

거듭제곱의 미분법을 사용하면 x에 대한 제곱근 x의 도함수가 $\frac{1}{2} \cdot x^{-\frac{1}{2}}$인데, $\frac{1}{2} \cdot \frac{1}{x^{1/2}} = \frac{1}{2\sqrt{x}}$과 같이 쓸 수도 있다.

이 도함수 공식을 다음과 같이 수식으로 코드화할 수 있다.

```
_function_bindings = {
    ...
    "sqrt": math.sqrt
}
_derivatives = {
    ...
    "sqrt": Quotient(Number(1), Product(Number(2), Apply(Function("sqrt"), _var)))
}
```

10.6.3 연습문제 정답 및 풀이

연습문제 | 10.22

이 문제는 도함수가 0인 함수를 구하라는 뜻이다. 임의의 상수함수는 모든 점에서 기울기가 0이므로 도함수가 0이다. 따라서 부정적분은 다음과 같다.

$$\int f(x)dx = \int 0dx = C$$

SymPy에서 `Integer(0)`이라는 코드는 수 0을 수식으로 만들어주며, x에 대한 이 수식의 적분은 다음과 같다.

```
>>> Integer(0).integrate(x)
0
```

함수 0은 0의 역도함수 중 하나이다. 적분상수를 더하면 $0+C$ 즉 C인데, 우리가 구한 결과와 같다. 임의의 상수함수는 상수가 0인 함수의 역도함수 중 하나이다.

연습문제 | 10.23

[힌트]에서 시작하자. $x\sin(x)$의 도함수는 곱의 미분법에 의해 $\sin(x)+x\cos(x)$이다. 우리가 원하는 결과에 가깝지만 $\sin(x)$ 항이 추가로 붙어있다. 도함수에 $-\sin(x)$ 항이 나타나도록 만들 수 있다면 불필요한 $\sin(x)$를 제거할 수 있는데, 다행히 $\cos(x)$의 도함수는 $-\sin(x)$이다. 따라서 $x\sin(x)+\cos(x)$의 도함수는 $\sin(x)+x\cos(x)-\sin(x)=x\cos(x)$이다. 그리고 우리가 구하는 부정적분은 다음과 같다.

$$\int x\cos(x)dx = x\sin(x)+\cos(x)+C$$

이 답은 SymPy에서 확인할 수 있다.

```
>>> (x*cos(x)).integrate(x)
x*sin(x) + cos(x)
```

이와 같이 도함수를 곱의 한 항으로 취급하는 리버스 엔지니어링적 접근을 **부분적분**(integration by parts)이라고 하는데, 미분적분학 교사라면 누구나 즐겨 사용한다.

연습문제 | 10.24

$f'(x) = x^2$이면 $f(x)$는 아마 x^3을 포함할 것이다. 거듭제곱의 미분법은 지수를 하나 낮추기 때문이다. x^3의 도함수는 $3x^2$이므로 이 결과의 $1/3$에 해당하는 함수를 얻고 싶다. 즉 우리가 찾는 식은 $x^3/3$이며, 도함수는 x^2이다. 따라서 구하는 식은 다음과 같다.

$$\int x^2 dx = \frac{x^3}{3} + C$$

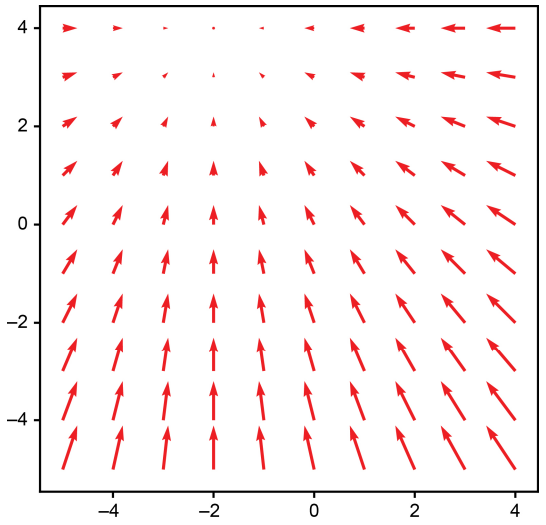
SymPy를 실행하면 이 식이 정답임을 확인할 수 있다.

```
>>> (x**2).integrate(x)
x**3/3
```

11.3.2 연습문제 정답 및 풀이

연습문제 | 11.1

이 벡터장은 변위 벡터 $(-2,4)-(x,y)$와 같은데, 이 벡터는 점 (x,y)에서 $(-2,4)$로 향한다. 이 벡터장의 모든 벡터는 $(-2,4)$를 향할 것이다. 벡터장을 그려 확인할 수 있다.



연습문제 | 11.2 미니 프로젝트

모든 위치 (x,y)에서 질량이 m인 물체는 두 중력 $m\cdot\mathbf{g}_1(x,y)$과 $m\cdot\mathbf{g}_2(x,y)$를 받는다. 이 힘의 벡터합은 $m(\mathbf{g}_1(x,y)+\mathbf{g}_2(x,y))$이다. 단위 질량당 물체가 받는 알짜힘은 $\mathbf{g}_1(x,y)+\mathbf{g}_2(x,y)$인데, 이는 전체 중력장 벡터가 각 블랙홀로 인한 중력장 벡터의 합임을 확인해준다. 총 중력장을 계산해보자.

$$\begin{aligned}\mathbf{g}(x,y)&=\mathbf{g}_1(x,y)+\mathbf{g}_2(x,y)\\&=0.1\cdot(-3-x,4-y)+0.1\cdot(2-x,1-y)\end{aligned}$$

우변을 성분끼리 더한 뒤, 각 성분을 2로 다시 묶자.

$$\begin{aligned}\mathbf{g}(x,y)&=0.1\cdot 2\cdot(0.5-x,2.5-y)\\&=0.2\cdot(0.5-x,2.5-y)\end{aligned}$$

이 중력장은 $(0.5,2.5)$에 위치한 중력 0.2인 블랙홀 한 개의 중력장과 동일하다.

연습문제 | 11.3 미니 프로젝트

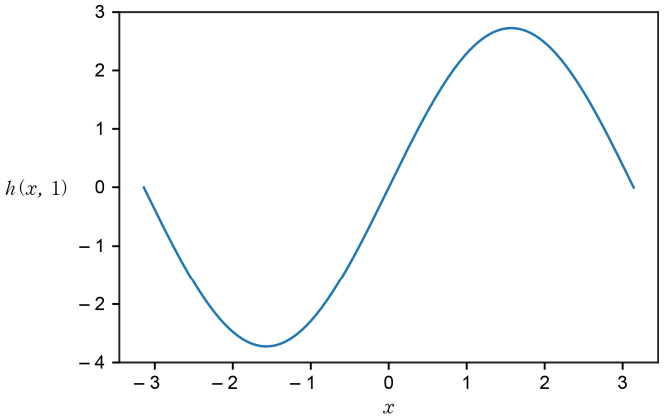
전체 구현은 소스 코드를 참고하라. 주요 추가 사항은 게임 루프를 반복할 때마다 각 블랙홀에 대해 move 메서드를 호출한다는 점이며, 이때 중력의 원천으로 나머지 블랙홀의 리스트를 전달한다.

```
for bh in black_holes:
    others = [other for other in black_holes if other != bh]
    bh.move(milliseconds, (0,0), others)
```

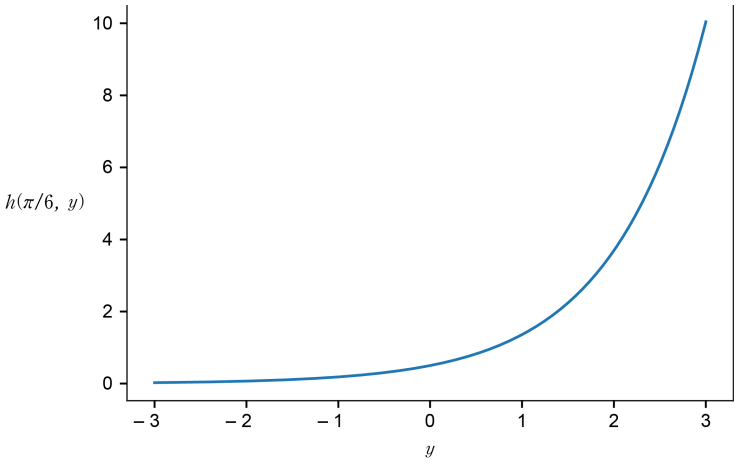
11.5.5 연습문제 정답 및 풀이

연습문제 | 11.4

$y=1$일 때 $h(x,y)$의 단면은 x에 대한 함수 $h(x,1)=e^1\sin(x)=e\cdot\sin(x)$로 다음과 같이 나타내어진다.



$x=\pi/6$인 경우 $h(x,y)$값은 y에만 의존한다. 즉, $h(\pi/6,y)=e^y\sin(\pi/6)=e^y/2$이다. 그래프는 다음과 같다.



연습문제 | 11.5

x에 대한 $e^y\sin(x)$의 편도함수는 y를 상수 취급해 얻는다. 따라서 e^y 또한 상수 취급하며 결과는 다음과 같다.

$$\frac{\partial h}{\partial x}=e^y\cos(x)$$

마찬가지로 x를 상수로 간주하면 $\sin(x)$ 또한 상수로 보고 y에 대한 편도함수를 얻을 수 있다.

$$\frac{\partial h}{\partial y}=e^y\sin(x)$$

그라디언트 $\nabla h(x,y)$는 각 편도함수를 각 성분으로 하는 벡터장이다.

$$\nabla h(x,y)=\left(\frac{\partial h}{\partial x},\frac{\partial h}{\partial y}\right)=(e^y\cos(x),e^y\sin(x))$$

$(x,y)=(\pi/6,1)$에서 이 벡터장의 값은 다음과 같이 계산한다.

$$\nabla h\left(\frac{\pi}{6},1\right)=\left(e^1\cos\left(\frac{\pi}{6}\right),e^1\sin\left(\frac{\pi}{6}\right)\right)=\frac{e}{2}\cdot(\sqrt{3},1)$$

연습문제 | 11.6

2장을 복습하는 차원으로 풀어보자. 두 벡터의 내적이 $(-5,2)\cdot(2,5)=-10+10=0$이므로 두 벡터는 수직이다.

연습문제 | 11.7 미니 프로젝트

$U(x,y)=\frac{1}{2}(x^2+y^2)$임을 떠올리기 바란다. $U(-5,2)$값은 14.5로, 점 $(x,y,z)=(-5,2,14.5)$는 3차원 상에서 $U(x,y)$ 그래프 위의 점이다.

$U(x,y)$를 가장 잘 근사하는 평면의 식을 고민하기 전에, 함수 $f(x)$를 가장 잘 근사하는 직선을 얻는 방법을 복습해보자. 점 x_0에서 함수 $f(x)$를 가장 잘 근사하는 직선은 점 $(x_0,f(x_0))$을 지나고 기울기가 $f'(x_0)$인 직선이다. 여기서 $f(x)$값과 $f(x)$의 미분계수가 함수 $f(x)$를 가장 잘 근사하는 직선과 일치함을 보장할 수 있다.

이 모델을 따라서 $(x,y)=(-5,2)$의 함숫값과 두 편미분계수가 **모두** 일치하는 평면 $p(x,y)$를 구해보자. $p(-5,2)=14.5$, $\partial p/\partial x=-5$, $\partial p/\partial y=2$이다. $p(x,y)$는 평면이므로 어떤 두 수 a,b에 대해 $p(x,y)=ax+by+c$ 꼴이다(왜 그런지 기억하는가?). 그러므로 각 편도함수는 다음과 같다.

$$\frac{\partial p}{\partial x}=a\text{이고 }\frac{\partial p}{\partial y}=b$$

이로부터 $p(x,y)=-5x+2y+c$이어야 하며, $p(-5,2)=14.5$를 만족해야 하므로 $c=-14.5$이다. 그러므로 U를 가장 잘 근사하는 평면의 식은 $p(x,y)=-5x+2y-14.5$이다.

이제 xy평면에 평행하고 평면 $p(x,y)$에 포함되면서 $(-5,2)$를 지나는 직선을 구해보자. 이러한 직선은 $p(x,y)=p(-5,2)$인 점 (x,y)의 집합인데, 직선 위의 점 (x,y)과 점 $(-5,2)$의 고도는 다르지 않기 때문이다.

$p(x,y)=p(-5,2)$라면 $-5x+2y-14.5=-5\cdot-5+2\cdot2-14.5$이다. 식을 정리하여 직선의 방정식 $-5x+2y=29$를 얻는다.[1] 이 직선은 $(-5,2,14.5)+r\cdot(2,5,0)$이라는 벡터 집합과 동등하며, 이때 r은 실수이다. 따라서 이 직선은 $(2,5,0)$과 확실히 평행한다.

1 **(옮긴이)** 수학적으로 이 방정식은 평면의 방정식에 가깝지 직선의 방정식이라고 할 수 없다. 고도에 대한 조건이 빠져있기 때문이다. 직선의 방정식을 올바르게 쓰면 $-5x+2y=29$, $z=p(-5,2)=14.5$이다.

12.1.4 연습문제 정답 및 풀이

연습문제 | 12.1

50°에서는 포탄이 x축의 양의 방향으로 약 40.1m만큼 나아가고 130°에서는 x축의 음의 방향으로 약 40.1m를 간다.

```
>>> landing_position(trajectory(50))
40.10994684444007
>>> landing_position(trajectory(130))
-40.10994684444007
```

이는 x축의 양의 방향에서 130°가 x축의 음의 방향에서 50°와 같기 때문이다.

연습문제 | 12.2 미니 프로젝트

개선한 함수는 다음과 같다. 이 함수는 매 초가 지날 때마다 가장 가까운 시점에 해당하는 인덱스를 찾고 각 인덱스에서 (x,z)값을 산점도(scatter plot)로 표현한다.

```
def plot_trajectories(*trajs,show_seconds=False):
    for traj in trajs:
        xs, zs = traj[1], traj[2]
        plt.plot(xs,zs)
        if show_seconds:
            second_indices = []
            second = 0
            for i,t in enumerate(traj[0]):
                if t>= second:
                    second_indices.append(i)
                    second += 1
            plt.scatter([xs[i] for i in second_indices],
                        [zs[i] for i in second_indices])
    ...
```

이를 통해 플로팅하는 궤적 각각의 경과 시간을 나타낼 수 있으며, 예는 다음과 같다.

```
plot_trajectories(
    trajectory(20),
    trajectory(45),
    trajectory(60),
    trajectory(80),
    show_seconds=True)
```

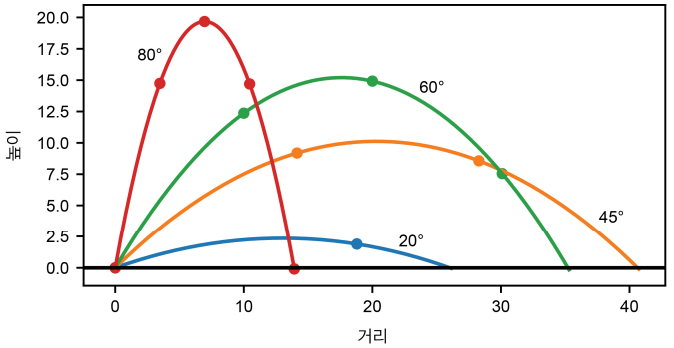


그림 매 초에 해당하는 지점을 나타낸 4개의 궤도 플롯

연습문제 | 12.3

```
test_angles = range(0,181,5)
hang_times = [hang_time(trajectory(theta)) for theta in test_angles]
plt.scatter(test_angles, hang_times)
```

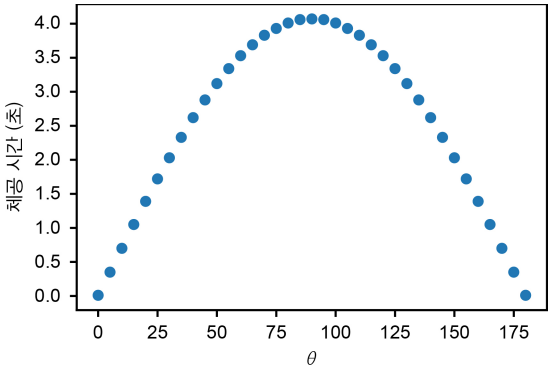


그림 발사 각도에 대한 함수로 포탄의 체공 시간을 나타낸 플롯

발사각도가 90°일 때 체공 시간은 약 4초로 가장 길다. $\theta = 90°$가 수직 성분이 가장 큰 초기 속도를 주기 때문에 참이다.

연습문제 | 12.4 미니 프로젝트

```
def plot_trajectory_metric(metric,thetas,**settings):
    plt.scatter(thetas,
                [metric(trajectory(theta,**settings))
                 for theta in thetas])
```

[연습문제 12.3]에서 그린 플롯은 다음과 같이 실행하면 그릴 수 있다.

```
plot_trajectory_metric(hang_time, range(0,181,5))
```

연습문제 | 12.5 미니 프로젝트

[연습문제 12.4]의 미니 프로젝트에 등장한 `plot_trajectory_metric` 함수를 사용한다면 다음 코드를 실행해 간단히 할 수 있다.

```
plot_trajectory_metric(landing_position,range(0,90,5), height=10)
```

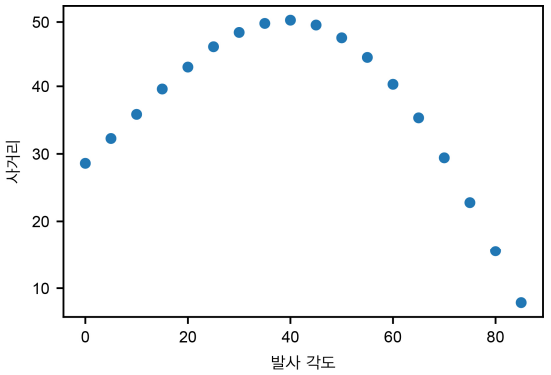


그림 발사 높이가 10m일 때 포탄의 사거리 대 발사 각도의 플롯

10m 높이에서의 최적 발사 각도는 약 40°이다.

12.2.4 연습문제 정답 및 풀이

연습문제 | 12.6

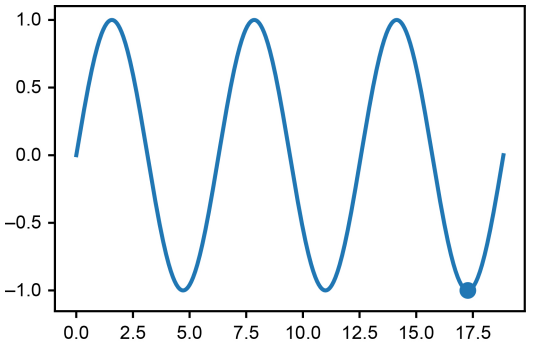
포탄의 초기 속도가 $v = |\mathbf{v}|$일 때, 체공 시간은 $t = 2v_z/g = 2v\sin(\theta)/g$이다. 이 식은 $\sin(\theta)$가 최대일 때 최대화된다. 미적분을 쓸 필요는 없다. $0 \le \theta \le 180°$에서 $\sin(\theta)$의 최댓값은 $\theta = 90°$에서 구하기 때문이다. 다른 매개변수가 모두 일정하면 포탄은 바로 위를 향해 발사할 때 가장 오랫동안 공중에 머물러 있다.

연습문제 | 12.7

$\sin(x)$의 도함수는 $\cos(x)$이며 다음이 성립한다.

$$\cos\left(\frac{11\pi}{2}\right) = \cos\left(\frac{3\pi}{2} + 4\pi\right) = \cos\left(\frac{3\pi}{2}\right) = 0$$

따라서 $\sin(x)$의 도함수는 $x = 11\pi/2$에서 0이다. $\sin(11\pi/2) = \sin(3\pi/2) = -1$이고 사인함수의 범위는 -1과 1 사이이기 때문에 이것이 국소 최솟값임을 확인할 수 있다. 이를 확인하기 위해 $\sin(x)$의 플롯을 그리면 다음과 같다.



연습문제 | 12.8

어떤 양수 x에서 극솟값을 가지고 어떤 음수 x에서 극댓값을 가지는 함수 $f(x)$를 플로팅하여 볼 수 있다.

도함수는 $f'(x)=3x^2-1$이므로 $3x^2-1=0$인 x를 구해야 한다. 근의 공식이 필요 없을 정도로 풀이는 간단하다. $3x^2-1=0$이면 $x^2=1/3$이므로 $x=-1/\sqrt{3}$ 또는 $x=1/\sqrt{3}$이다. 이 x값들은 $f(x)$가 극댓값 및 극솟값을 갖는다.

극댓값은 $f\left(\frac{-1}{\sqrt{3}}\right)=\frac{-1}{3\sqrt{3}}-\frac{-1}{\sqrt{3}}=\frac{2}{3\sqrt{3}}$이고,

극솟값은 $f\left(\frac{1}{\sqrt{3}}\right)=\frac{1}{3\sqrt{3}}-\frac{1}{\sqrt{3}}=\frac{-2}{3\sqrt{3}}$이다.

연습문제 | 12.9 미니 프로젝트

이 이차함수의 도함수는 $q'(x)=2ax+b$이다. $x=-b/2a$일 때 도함수의 값은 0이다.

a가 양수일 때, 매우 작은 x값에서 도함수가 음수인데, x값이 증가하면서 $x=-b/2a$에서 도함수는 0이고 그 이후로는 도함수가 양수이다. 이는 q가 $x=-b/2a$보다 작을 때는 감소하며 $x=-b/2a$보다 클 때에는 증가함을 의미한다. 즉, $x=-b/2a$일 때 $q(x)$가 **최솟값**을 가진다.

a가 음수이면 정반대이다. 결론적으로 a가 양수이면 $x=-b/2a$에서 $q(x)$가 최솟값을 가지고 a가 음수이면 $x=-b/2a$에서 $q(x)$가 최댓값을 가진다.

12.3.4 연습문제 정답 및 풀이

연습문제 | 12.10

$|\mathbf{v}| = v$가 포탄의 초기 속도라고 할 때, 초기 속도 벡터의 크기가 v와 같은지 확인하라. 즉, 벡터 $(v\cos\theta\cos\phi,\ v\cos\theta\sin\phi,\ v\sin\theta)$의 길이가 v임을 나타내라.

힌트 사인, 코사인의 정의와 피타고라스 정리에 의해 모든 x에 대해 $\sin^2 x + \cos^2 x = 1$이다.

정답 및 풀이

$(v\cos\theta\cos\phi, v\cos\theta\sin\phi, v\sin\theta)$의 크기는 다음과 같다.

$$\begin{aligned}\sqrt{v^2\cos^2\theta\cos^2\phi + v^2\cos^2\theta\sin^2\phi + v^2\sin^2\theta} &= \sqrt{v^2(\cos^2\theta\cos^2\phi + \cos^2\theta\sin^2\phi + \sin^2\theta)} \\ &= \sqrt{v^2(\cos^2\theta(\cos^2\phi + \sin^2\phi) + \sin^2\theta)} \\ &= \sqrt{v^2(\cos^2\theta \cdot 1 + \sin^2\theta)} \\ &= \sqrt{v^2 \cdot 1} \\ &= v\end{aligned}$$

연습문제 | 12.11

다음 식에서부터 시작해보자.

$$d = \frac{-v_z \cdot v_{xy}}{\frac{g}{2} - Bv_x^2 + Cv_y^2}$$

$v_z = v\sin\theta$, $v_{xy} = v\cos\theta$, $v_y = v\cos\theta\sin\phi$, $v_x = v\cos\theta\cos\phi$를 대입하면

$$d(\theta,\phi) = \frac{-v^2\sin\theta cos\theta}{\frac{g}{2} - Bv^2\cos^2\theta\cos^2\phi + Cv^2\cos^2\theta\sin^2\phi}$$

분모를 약간 정리하면 다음과 같다.

$$d(\theta,\phi) = \frac{-v^2\sin\theta cos\theta}{\frac{g}{2} + v^2\cos^2\theta \cdot (C\sin^2\phi - B\cos^2\phi)}$$

연습문제 | 12.12 미니 프로젝트

항력으로 인한 가속도를 시뮬레이션에 추가해야 한다. 항력이 $-\alpha \mathbf{v}$이므로 항력으로 인한 가속도는 $-\alpha \mathbf{v}/\mathrm{m}$이다. 객체의 질량은 바뀌지 않으므로 α/m을 항력에 관한 상수 매개변수로 사용할 수 있다. 항력으로 인한 가속도의 성분은 각각 $v_x\alpha/\mathrm{m}$, $v_y\alpha/\mathrm{m}$, $v_z\alpha/\mathrm{m}$이다. 다음은 기존 코드에서 변경된 부분이다.

```
def trajectory3d(theta,phi,speed=20,height=0,dt=0.01,g=-9.81,
                 elevation=flat_ground, drag=0):
    ...
    while z >= elevation(x,y):
        t += dt
        vx -= (drag * vx) * dt
        vy -= (drag * vy) * dt
        vz += (g - (drag * vz)) * dt
        ...
    return ts, xs, ys, zs
```

항력에 비례해 vx와 vy를 감소시킨다.

z속도(vz)를 중력과 항력의 효과를 반영해 변화시킨다.

0.1과 같이 작은 drag 상수조차 포탄을 눈에 띄게 느리게 만든다. 이로 인해 포탄의 궤적은 항력이 없는 궤적까지 도달하지 못한다.

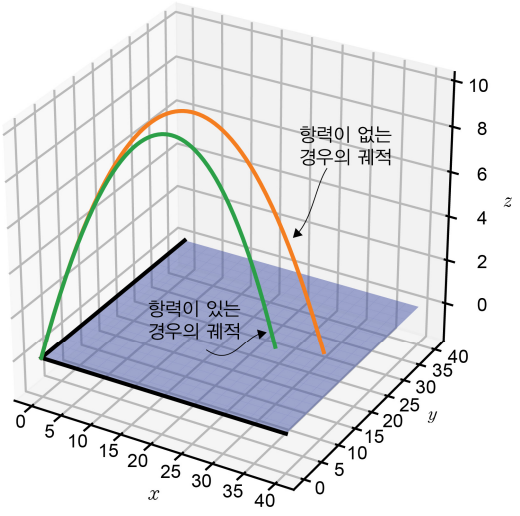


그림 drag = 0일 때와 drag = 0.1일 때 포탄의 궤적

12.4.5 연습문제 정답 및 풀이

연습문제 | 12.13

히트맵을 이미 플로팅해둔 뒤에, 다음을 실행하면 무작위로 택한 점 20개에 대해 경사상승법을 플로팅할 수 있다.

```
from random import uniform
for x in range(0,20):
    gap = gradient_ascent_points(landing_distance,
                                 uniform(0,90),
                                 uniform(0,360))
    plt.plot(*gap,c='k')
```

결과는 모든 길이 같은 곳으로 이어짐을 보여준다.

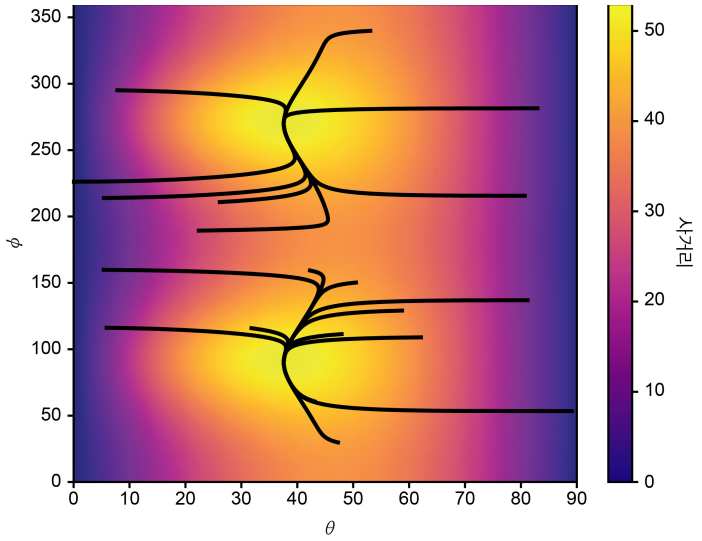


그림 무작위로 선택된 시작점 20개에 대한 경사상승법 경로

연습문제 | 12.15

기교를 부리자면 $\phi = 180°$에서 경사상승법을 시작하면 된다. 상황의 대칭성을 고려하면 $\phi = 180°$에서는 언제나 $\partial r/\partial\phi = 0$이다. 그러면 경사상승법이 $\phi = 180°$이 나타내는 선을 벗어날 이유가 없어지기 때문이다.

```
>>> gradient_ascent(landing_distance,0,180)
(46.122613357930206, 180.0)
```

이 θ값은 $\phi = 0°$ 또는 $\phi = 180°$로 고정할 때 최적 발사 각도이며, 오르막을 향해 발사하므로 최악의 각도이다.

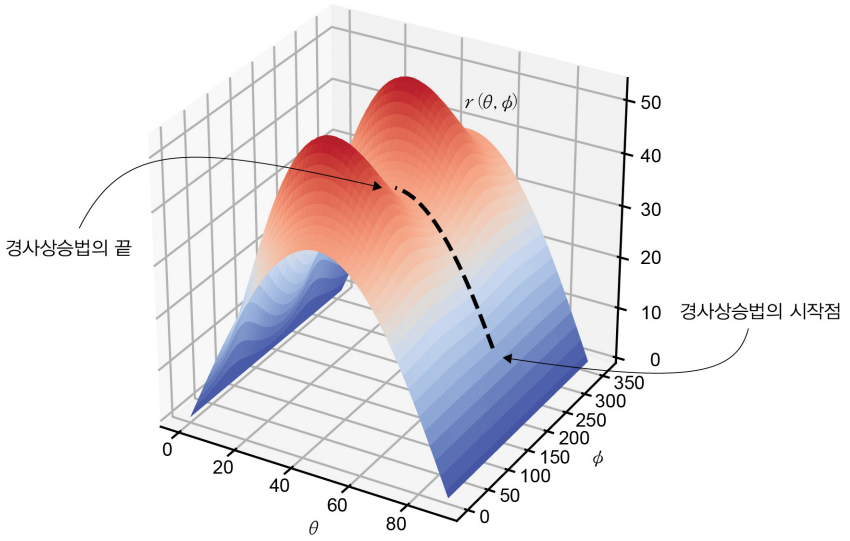


그림 $\partial r/\partial\phi = 0$인 단면에서 경사상승법을 초기화하는 기교

연습문제 | 12.16

경사상승법을 계산할 때 상승이 나아가는 빠르기를 rate라는 매개변수로 나타내자. rate가 클수록 현재 계산된 그라디언트를 신뢰하여 해당 방향으로 더 멀리 나아간다고 하자.

```
def gradient_ascent_points(f,xstart,ystart,rate=1,tolerance=1e-6):
    ...
    while length(grad) > tolerance:
        x += rate * grad[0]
        y += rate * grad[1]
        ...
    return xs, ys
```

다음 함수는 경사상승법 과정이 수렴할 때까지 단계 수를 세어준다.

```
def count_ascent_steps(f,x,y,rate=1):
   gap = gradient_ascent_points(f,x,y,rate=rate)
   print(gap[0][-1],gap[1][-1])
   return len(gap[0])
```

rate 매개변수가 1인 경우 원래 경사상승법은 855단계가 필요하다.

```
>>> count_ascent_steps(landing_distance,36,83)
855
```

rate=1.5일 때에는 단계별로 그라디언트의 1.5배만큼 나아간다. 당연하게도 최대점에 더 빨리 도달해서 568단계면 된다.

```
>>> count_ascent_steps(landing_distance,36,83,rate=1.5)
568
```

rate에 여러 값을 시도해 보면 rate가 증가할수록 더 적은 단계만 거쳐도 최적해에 도달함을 알 수 있다.

```
>>> count_ascent_steps(landing_distance,36,83,rate=3)
282
>>> count_ascent_steps(landing_distance,36,83,rate=10)
81
>>> count_ascent_steps(landing_distance,36,83,rate=20)
38
```

그러나 과욕은 금물이다. rate를 20으로 하면 분명 더 적은 단계로 정답에 도달하지만, 중간 단계에서는 정답에서 너무 벗어나서 다음 단계에 되돌아온다. rate를 너무 크게 설정하면 알고리즘은 정답에서 더더욱 벗어날 수 있으며, 그런 경우에는 수렴하지 않고 발산한다.

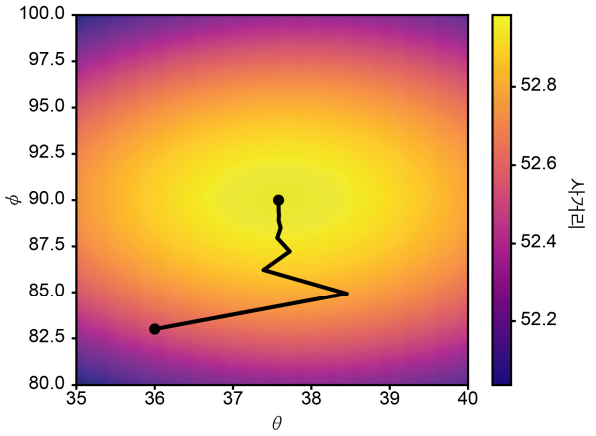


그림 rate가 20일 때 경사상승법. 경사상승법 알고리즘은 초기에 최대 θ값을 넘어가는 바람에 뒤로 다시 돌아와야 한다.

rate를 40으로 올려보면 경사상승법은 더 이상 수렴하지 않는다. 매번 이동할 때마다 정답에서 더 멀어지므로 이런 식으로 매개변수 공간을 탐색하면 무한대로 간다.

연습문제 | 12.17

이 결과는 그다지 아름답지 않다. (발사체가 지면에 닿는 시점을 결정하는 것 같은 상황에서) 시뮬레이션한 결과가 수치 추정에 의존하고, 그래서 발사 각도가 조금만 변해도 추정값들이 급격하게 변하기 때문이다. 편미분계수 $\partial r/\partial\theta$를 계산할 때 앞에서 등장한 편미분계수 근사 계산기를 사용한 경우, 단면 $r(\theta,270°)$의 플롯은 다음과 같다.

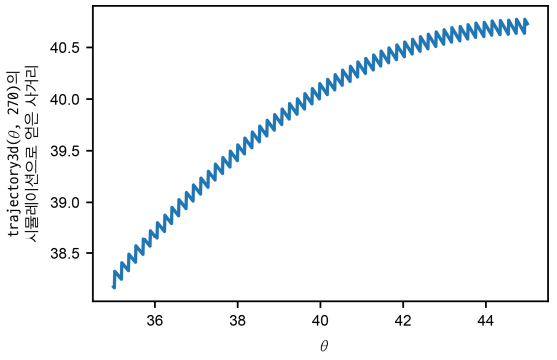


그림 시뮬레이션 궤적의 단면은 시뮬레이터가 매끄러운 함수 $r(\theta,\phi)$를 생성하지 못함을 보여준다.

편도함수의 값이 크게 변동하기 때문에 경사상승법은 랜덤한 방향으로 이동한다.

13.2.3 연습문제 정답 및 풀이

연습문제 | 13.1

주파수 44,100Hz는 350으로 나누어떨어지며 44100/350 = 126이다.

10,000이 63개, −10,000이 63개인 수열을 350번 반복하면 1초짜리 오디오를 만들 수 있다. 결과로 얻은 음은 A보다 낮은 소리이며 실제로 F이다.

```
form = np.repeat([10000,-10000],63)
arr = np.tile(form,350)
sound = pygame.sndarray.make_sound(arr)
sound.play()
```

13.3.4 연습문제 정답 및 풀이

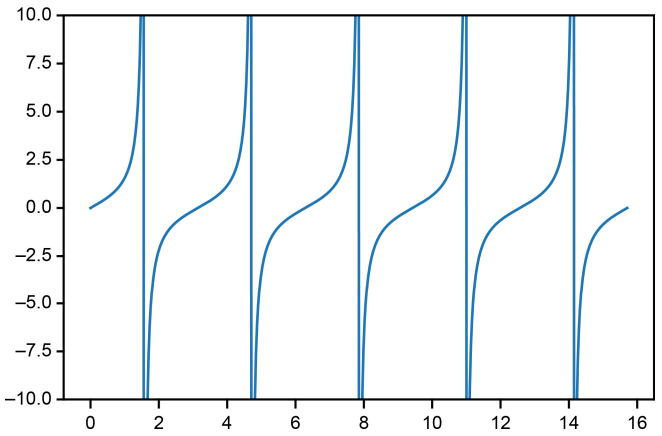
연습문제 | 13.2

탄젠트 함수는 매 주기마다 무한히 커진다. y값의 범위를 적절히 제한해 플로팅해보면 도움이 된다.

```
from math import tan
plot_function(tan,0,5*pi)
plt.ylim(-10,10)
```

y범위가 $-10 < y < 10$가 되도록 그래프 화면을 제한한다.

주기함수 $\tan(t)$의 그래프는 다음과 같다.



$\tan(t)$는 $\cos(t)$와 $\sin(t)$값에만 의존하므로, 적어도 2π마다 반복되어야 한다. 사실 2π마다 **두 번** 반복된다. 그래프에서 주기가 π임을 확인할 수 있다.

연습문제 | 13.3

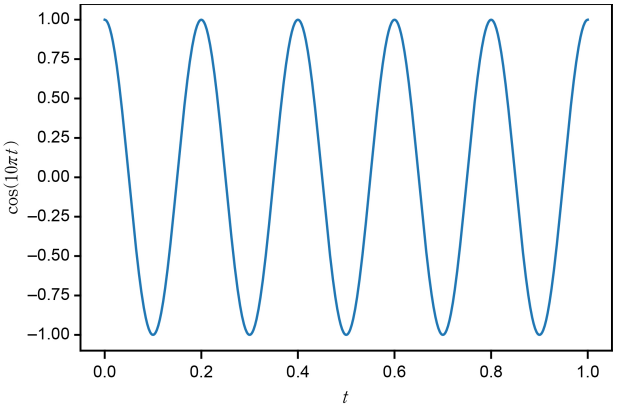
$\sin(t)$의 주파수는 $1/(2\pi)$이고, 사인 함수의 인자에 3π를 곱하면 주파수도 3π배 늘어난다. 결과로 얻은 주파수는 $(3\pi)/(2\pi) = 3/2$이다. 주기는 주파수의 역수이므로 $2/3$이다.

연습문제 | 13.4

$\cos(t)$의 기본 주파수는 $1/2\pi$이므로 $\cos(kt)$의 주파수는 $k/(2\pi)$이다. 이 값이 5와 같으려면 $k=10\pi$이어야 한다. 따라서 구하려는 함수는 $\cos(10\pi t)$이다.

```
>>> plot_function(lambda t: cos(10*pi*t),0,1)
```

이 그래프는 다음과 같은데, $t=0$와 $t=1$ 사이에서 5번 반복한다.



13.5.6 연습문제 정답 및 풀이

연습문제 | 13.6

각 내적은 다음과 같다.

$$a_1 = \mathbf{v} \cdot \mathbf{u}_1 = (3,4,5) \cdot (2,0,0) = 6$$
$$a_2 = \mathbf{v} \cdot \mathbf{u}_2 = (3,4,5) \cdot (0,1,1) = 9$$
$$a_3 = \mathbf{v} \cdot \mathbf{u}_3 = (3,4,5) \cdot (1,0,-1) = -2$$

이 결과로 만든 일차결합 $6 \cdot (2,0,0) + 9 \cdot (0,1,1) - 2 \cdot (1,0,-1) = (16,9,2)$는 $(3,4,5)$와 같지 않다. 기저 벡터 $\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3$의 길이가 1이 아니고, 서로 수직이 아니기 때문에 이렇게 계산하면 올바른 결과가 나오지 않는다.

연습문제 | 13.7 미니 프로젝트

내적 $\langle f,f \rangle$는 다음과 같이 적분식으로 표현된다.

$$2 \cdot \int_0^1 f(t) \cdot f(t)dt = 2 \cdot \int_0^1 k \cdot kdt = 2k^2$$

(0부터 1까지 상수 함수 k^2의 넓이는 k^2이다.) $2k^2$이 1이 되려면 $k^2 = 1/2$이어야 하고, k가 양수라고 가정하면 $k = \sqrt{1/2} = 1/\sqrt{2}$이다.

연습문제 | 13.8

```
def fourier_series(a0,a,b):
    def result(t):
        cos_terms = [an*cos(2*pi*(n+1)*t) for (n,an) in enumerate(a)]
        sin_terms = [bn*sin(2*pi*(n+1)*t) for (n,bn) in enumerate(b)]
        return a0/sqrt(2) + sum(cos_terms) + sum(sin_terms)
    return result
```

일차결합에서 계수 a_0와 상수 함수 $f(t) = 1/\sqrt{2}$가 곱해져야 하므로, t값과 무관하게 fourier_series의 result는 $a_0/\sqrt{2}$를 리턴한다.

연습문제 | 13.9 미니 프로젝트

진폭을 8,000으로, 주파수를 441로 설정한 톱니파 함수를 작성한 뒤 샘플링하여 PyGame에 전달하자.

```
def modified_sawtooth(t):
    return 8000 * sawtooth(441*t)
arr = sample(modified_sawtooth,0,1,44100)
sound = pygame.sndarray.make_sound(arr)
sound.play()
```

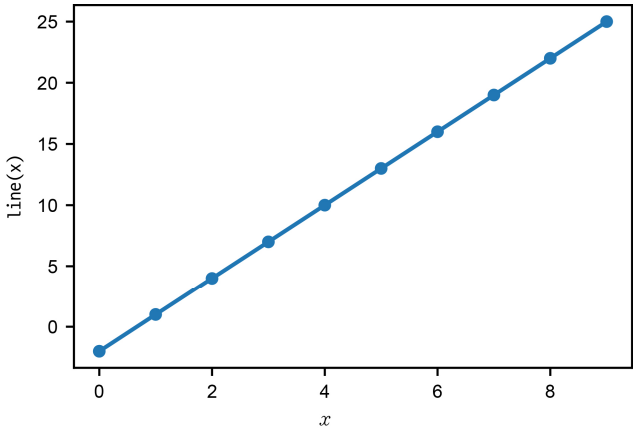
사람들은 종종 톱니파 소리를 바이올린과 같은 현악기 소리에 비유한다.

14.1.4 연습문제 정답 및 풀이

연습문제 | 14.1

다음은 일차함수 및 일차함수의 그래프에 있는 몇 가지 점이다.

```
def line(x):
    return 3*x-2
points = [(x,line(x)) for x in range(0,10)]
```



`sum_error(line,points)`와 `sum_squared_error(line,points)` 둘 다 0을 리턴하는데, 모든 점에서 직선까지 거리가 0이기 때문이다.

연습문제 | 14.2

```
>>> sum_squared_error(lambda x:2*x-1,test_data)
23.1942461283472
>>> sum_squared_error(lambda x:x+0.5,test_data)
16.607900877665685
```

함수 $x+0.5$는 `sum_squared_error`의 값이 더 작다. 그러므로 `test_data`에 더 잘 피팅된다.

연습문제 | 14.3

지금까지 구한 가장 적합한 함수는 $p(x) = 22500 - 0.1 \cdot x$를 나타낸 p3이다. 더 적합한 함수를 구하려면 비용이 감소될 때까지 상수를 바꿔볼 수 있다. 여기서 얻을 수 있는 관찰에는 $b = 25000$을 $b = 22500$으로 줄여서 함수가 더 적합하다는 점이다. 이 값을 더 줄이면 적합도는 더 개선된다. $b = 20000$인 새 함수 p4를 정의하면 다음과 같다.

```
def p4(x):
    return 20000 - 0.1 * x
```

또한 sum_squared_error는 더 작아짐을 알 수 있다.

```
>>> sum_squared_error(p4, prius_mileage_price)
18958453681.560005
```

이전 세 함숫값 중 어떤 것보다도 이 값이 낮기에, p4가 데이터에 더 적합하다.

14.2.3 연습문제 정답 및 풀이

연습문제 | 14.4

계수는 a만 구하면 된다. 제곱 오차 합은 $f(3)=a\cdot 3$과 4의 차를 제곱한 $(3a-4)^2$으로, 이를 전개하면 $9a^2-24a+16$이다. 이 식은 a에 관한 비용 함수 $c(a)=9a^2-24a+16$이다.

여기서 가장 좋은 a값은 비용을 최소화하는 값이다. 그러한 a값은 비용 함수의 도함수를 0으로 만든다. 10장의 미분법을 사용하면 $c'(a)=18a-24$이다. 이 도함수는 $a=4/3$일 때 0으로 최적합 직선은 $f(x)=\frac{4}{3}x$임을 의미한다. 이 직선은 원점과 점 $(4,3)$을 모두 지난다.

연습문제 | 14.5

$x=0$일 때 $ax+b$값은 $b=80000$이다. 주행거리가 0인 시점에서 스포츠카를 8만 달러에 판다고 기대한다는 뜻이다. $a=-0.4$는 x가 1단위씩 증가할 때마다 함숫값 $ax+b$가 0.4단위 비율로 감소함을 의미한다. 문제에서는 스포츠카를 1마일 운전할 때마다 가치가 평균적으로 40센트씩 감소한다는 뜻이다.

14.3.3 연습문제 정답 및 풀이

연습문제 | 14.6

계수 a, b에 대한 식 $f(x)=ax+b$를 사용해 테스트 데이터에 대한 비용 함수부터 작성하자.

```
def test_data_linear_cost(a,b):
    def f(x):
        return a*x+b
    return sum_squared_error(f,test_data)
```

이 함수를 최소화하는 a값, b값은 최적합 일차함수를 준다. 문제에 따르면 a, b는 각각 2, 0과 가까울 거라 예상되나, 최소화하려는 함수를 이해하는 차원에서 이 값들이 나타내는 점 주변의 히트맵을 플로팅할 수 있다.

```
scalar_field_heatmap(test_data_linear_cost,-0,4,-2,2)
```

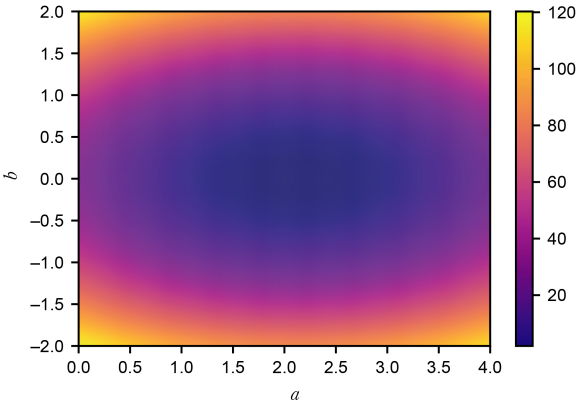


그림 a와 b의 함수에 해당하는, 테스트 데이터에 대한 $ax+b$의 비용

예상대로 $(a,b)=(2,0)$ 부근에서 비용 함수의 최솟값이 존재하는 듯하다. 경사하강법을 사용해 이 함수를 최소화하면 정확한 값을 구할 수 있다.

```
>>> gradient_descent(test_data_linear_cost,1,1)
(2.103718204728344, 0.0021207385859157535)
```

이것은 테스트 데이터에 대한 최적합 직선이 약 $2.10372 \cdot x+0.00212$임을 의미한다.

14.4.3 연습문제 정답 및 풀이

연습문제 | 14.7

$r=3$이라고 두면 함수는 e^{-3x}이다. x가 $1/3$ 단위만큼 증가할 때마다 e^{-3x}가 e**배** 감소함을 확인하고자 한다. 이 함수를 파이썬에서 정의하면 다음과 같다.

```
def test(x):
    return exp(-3*x)
```

이 함수는 $x=0$일 때 1이며 x에 $1/3$을 더할 때마다 e배 감소함을 관찰할 수 있다.

```
>>> test(0)
1.0
>>> from math import e
>>> test(1/3), test(0)/e
(0.36787944117144233, 0.36787944117144233)
>>> test(2/3), test(1/3)/e
(0.1353352832366127, 0.1353352832366127)
>>> test(1), test(2/3)/e
(0.049787068367863944, 0.04978706836786395)
```

위의 각 경우에서 `test` 함수의 입력에 $1/3$을 더할수록 기존 결과를 e로 나눈 결과가 나옴을 확인할 수 있다.

연습문제 | 14.8

가격 함수는 $p(x)=18700\cdot e^{-0.00000768\cdot x}$이다. 여기서 $q=18700$달러는 초기 가격을 나타내긴 하지만 가격의 감소 속도와는 관계가 없다. 따라서 나머지 항 $e^{rx}=e^{-0.00000768\cdot x}$에 주목해서 10,000마일 동안 얼마나 변화하는지를 보면 된다. $x=0$일 때 이 식의 값은 1이며, $x=10000$일 때 이 값은 다음과 같다.

```
>>> exp(r * 10000)
0.9422186306357088
```

이는 10,000마일을 주행한 프리우스의 가격이 초기 가격의 94.2%에 불과해 5.8%p 감소했음을 의미한다. 지수함수의 동작을 고려하면 주행거리가 얼마였든 10,000마일만큼 증가하는 모든 경우에 참일 것이다.

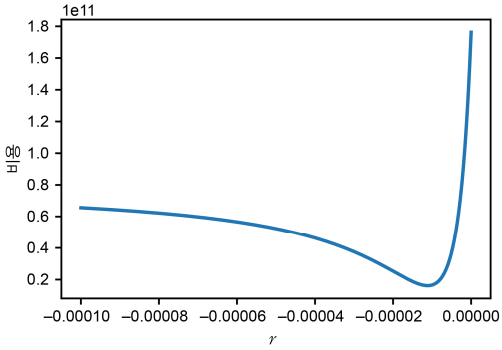
연습문제 | 14.9

주어진 지수함수로부터 계수 r을 미지수 하나로 갖는 비용 함수를 새로 작성해보자.

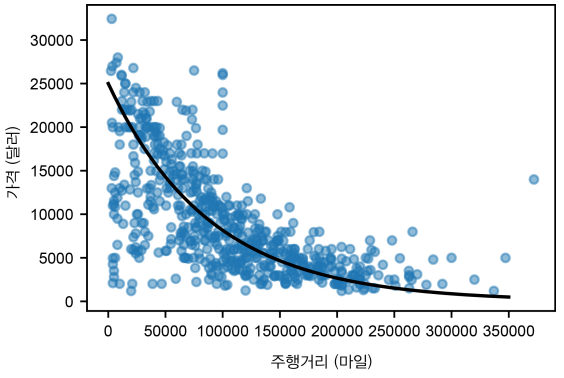
```
def exponential_cost2(r):
    def f(x):
        return 25000 * exp(r*x)
    return sum_squared_error(f,prius_mileage_price)
```

다음 플롯은 비용 함수를 최소화하는 r값이 -10^{-4}과 0 사이에 있음을 확인해준다.

```
plot_function(exponential_cost2,-1e-4,0)
```



대략 $r = -10^{-5}$ 값이 비용 함수를 최소화하는 것으로 보인다. 이 함수를 자동으로 최소화하려면 경사하강법 알고리즘을 1차원 형태로 작성하거나 다른 최소화 알고리즘을 사용해야 한다. 원한다면 그렇게 해도 되겠으나 이 경우는 매개변수가 하나만 있으므로 추정과 확인을 반복해서 $r = -1.12 \cdot 10^{-5}$이 최소 비용을 주는 r의 근삿값임을 확인할 수 있다. 이는 최적합 함수가 $p(x) = 25000 \cdot e^{-0.0000112 \cdot x}$임을 의미한다. 가격 데이터와 함께 새로운 지수함수의 그래프를 그려보면 다음과 같다.



15.1.3 연습문제 정답 및 풀이

연습문제 | 15.1

예측이 맞았는지 틀렸는지만 추적하기보다 참·거짓, 양성·음성 각각을 추적하는 게 좋다.

```
def test_classifier(classifier, data, verbose=False):
    true_positives = 0
    true_negatives = 0
    false_positives = 0
    false_negatives = 0
    for mileage, price, is_bmw in data:
        predicted = classifier(mileage,price)
        if predicted and is_bmw:
            true_positives += 1
        elif predicted:
            false_positives += 1
        elif is_bmw:
            false_negatives += 1
        else:
            true_negatives += 1
    if verbose:
        print("true positives %f" % true_positives)
        print("true negatives %f" % true_negatives)
        print("false positives %f" % false_positives)
        print("false negatives %f" % false_negatives)

    total = true_positives + true_negatives

    return total / len(data)
```

verbose로 각 개수를 화면에 출력할지 명시한다(False인 이유는 매번 출력하지 않도록 한다).

추적할 변수는 4개이다.

자동차가 프리우스인지 BMW인지, 올바로 분류했는지 아닌지에 따라 네 개의 변수 중 하나만을 1만큼 증가시킨다.

각 변수의 결과를 출력한다.

올바로 분류한 포인트(참 양성, 참 음성)의 개수를 데이터셋의 튜플 개수로 나누어 리턴한다.

bmw_finder 함수에 대해 실행하면 이 함수는 다음 텍스트를 출력한다.

```
true positives 18.000000
true negatives 100.000000
false positives 0.000000
false negatives 82.000000
```

이 분류기는 거짓 양성인 경우가 없으므로 자동차가 BMW가 **아니면**(프리우스라면) 언제나 올바로 식별한다. 아직 만족하면 안 된다. 이 분류기는 BMW인데 BMW가 아니라고 예측(거짓 음성)하는 경우가 82가지이기 때문이다. [연습문제 15.2]에서 총 성공률(정확도)을 개선하기 위해 제약조건을 완화할 것이다.

연습문제 | 15.2

[연습문제 15.1]을 풀어봤다면 `bmw_finder` 분류기는 데이터를 보고 BMW가 아니라고 단정 짓는 경우가 너무 빈번함을 알 수 있다. BMW라고 예상한 경우가 18+0번, BMW가 아니라고 예상한 경우는 100+82번이다! 경곗값(threshold) 가격을 20,000달러로 낮추면 결과가 달라지는지 관찰해보자.

```
def bmw_finder2(mileage,price):
    if price > 20000:
        return 1
    else:
        return 0
```

경곗값을 낮추어보니 `bmw_finder`의 성공률(정확도)이 다행히 73.5%로 개선되었다.

```
>>> test_classifier(bmw_finder2, all_car_data)
0.735
```

15.2.4 연습문제 정답 및 풀이

연습문제 | 15.3 미니 프로젝트

다음 함수는 주어진 상수 기준 가격(cutoff_price)에 대해 분류 함수를 생성한다. 이렇게 얻은 분류기는 자동차가 기준 가격보다 비싸면 1을, 기준 가격과 같거나 저렴하면 0을 리턴한다.

```
def constant_price_classifier(cutoff_price):
    def c(x,p):
        if p > cutoff_price:
            return 1
        else:
            return 0
    return c
```

이 분류기를 test_classify 함수에 전달하면 분류기의 정확도를 측정할 수 있다. 다음 코드는 임의의 값을 기준 가격(cutoff_price)으로 설정해 정확도를 측정하는 과정을 자동화하는 보조 함수이다.

```
def cutoff_accuracy(cutoff_price):
    c = constant_price_classifier(cutoff_price)
    return test_classifier(c,all_car_data)
```

엄밀하게 따지면 최적 기준 가격은 all_car_data 리스트에 있는 가격 값 중에서 어떤 두 값 사이에 존재한다. 하지만 리스트 가격 값 중에는 반드시 최적 기준 가격이 존재하므로 이 리스트에서 최적 기준 가격을 찾아내면 충분하다. 이러한 작업은 파이썬에서 max 함수로 빠르게 진행할 수 있다. max 함수의 키워드 인자 key를 우리가 최대화해야 하는 함수를 지정할 때 쓸 수 있기 때문이다. 우리는 리스트 가격 값 중에서 최적 기준 가격을 구하려 하고, 이는 cutoff_accuracy 함수를 최대화하는 가격을 리스트 가격 중에서 찾는 작업과 동일하다.

```
>>> max(all_prices,key=cutoff_accuracy)
17998.0
```

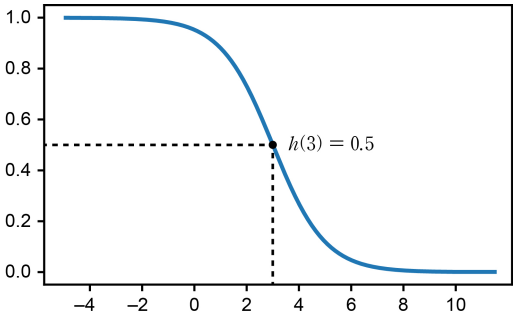
이 결과는 상수 기준 가격에 따라 자동차가 BMW 5 시리즈인지 프리우스인지 분류하는 분류기를 사용할 때, 해당 데이터셋에 대한 최적 기준 가격이 17,998달러임을 알려준다. 이 최적 기준 가격을 적용한 분류기의 정확도는 79.5%로 상당히 높다.

```
>>> test_classifier(constant_price_classifier(17998.0), all_car_data)
0.795
```

15.3.5 연습문제 정답 및 풀이

연습문제 | 15.4

함수를 $y(x)=3-x$이라고 두면 $y(3)=0$이다. x가 음수이고 절댓값이 커질 때, $y(x)$는 양의 무한대를 향하며 x가 양수이고 절댓값이 커질 때, $y(x)$는 음의 무한대를 향한다. 이는 $y(x)$의 결과를 시그모이드 함수에 대입해서 문제의 조건을 만족하는 함수를 만들 수 있음을 의미한다. 다시 말하면 $h(x)=\sigma(y(x))=\sigma(3-x)$가 성립하며, 그래프는 다음과 같다.



연습문제 | 15.5 미니 프로젝트

히트맵을 살펴보면 함수 $f(x,p)$는 왼쪽과 아래쪽으로 갈수록 함숫값이 작아진다. 이러한 성질은 식에서도 관찰할 수 있다. x나 p를 감소시키면 $f=p-ax-b=p+0.35\cdot x-0.56$의 값이 점점 줄어든다. 따라서 $f(x,p)$는 $(x,p)=(0,0)$일 때 최솟값이 $f(0,0)=-0.56$으로 존재한다.

15.4.4 연습문제 정답 및 풀이

연습문제 | 15.6

참고로 Matplotlib의 `plot` 함수에서 플로팅한 직선의 색을 설정하는 키워드 인자로 `c`를 사용하기 때문에, 구현할 함수의 인자로 `a`, `b`, `c` 대신 다른 이름을 붙였다.

```
def plot_line(acoeff,bcoeff,ccoeff,**kwargs):
    a,b,c = acoeff, bcoeff, ccoeff
    if b == 0:
        plt.plot([c/a,c/a],[0,1])
    else:
        def y(x):
            return (c-a*x)/b
        plt.plot([0,1],[y(0),y(1)],**kwargs)
```

연습문제 | 15.7

시그모이드 함수는 $\sigma(x)=\frac{1}{1+e^{-x}}$이므로 다음과 같이 쓰면 된다.

$$\sigma(ax+by-c)=\frac{1}{1+e^{c-ax-by}}$$

연습문제 | 15.8 미니 프로젝트

방정식이 $x^2+y^2-1=0$, 즉 $x^2+y^2=1$이면 언제나 $\sigma(x^2+y^2-1)=0.5$이다. 방정식 $x^2+y^2-1=0$의 해는 원점에서 거리가 1인 점들의 집합이자 반지름이 1인 원이다. 원 내부의 점과 원점 사이의 거리는 1보다 작으므로 $x^2+y^2<1$이며, 이는 $\sigma(x^2+y^2-1)<0.5$임을 알려준다. 원 외부의 점은 $x^2+y^2>1$을 만족하므로 $\sigma(x^2+y^2-1)>0.5$이다. 원점에서 방향에 관계없이 멀어질수록 함수의 그래프는 1에 다가가며, 원 내부에서 원점으로 다가갈수록 함수의 그래프는 감소하여 원점에서 약 0.27이라는 최솟값을 가진다. 이 그래프를 그려보면 다음과 같다.

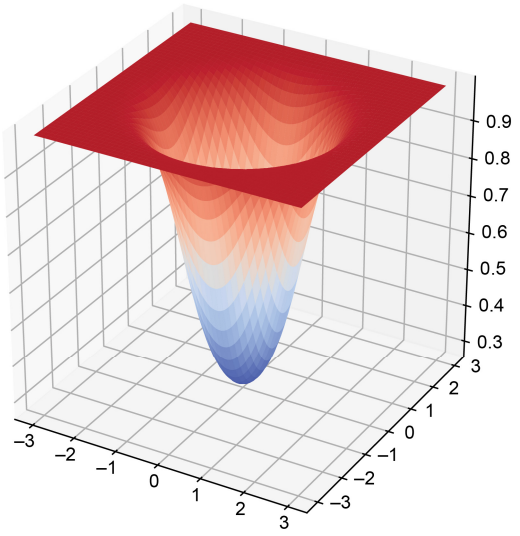


그림 $\sigma(x^2+y^2-1)$의 그래프. 반지름이 1인 원의 내부에서는 함숫값이 0.5보다 작고, 원 밖에서는 어느 방향에서든지 함숫값이 증가하여 1까지 도달한다.

연습문제 | 15.9 미니 프로젝트

두 함수는 같은 함수가 아니다. x나 y가 증가하면 $2x+y-1$보다 $4x+2y-2$가 더 급격하게 증가한다. 때문에 $\sigma(4x+2y-2)$가 더 가파르다.

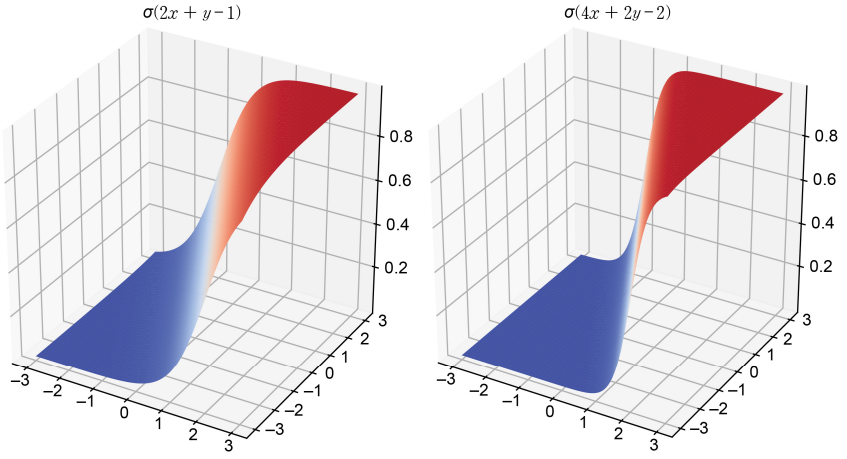


그림 좌측 그래프보다 가파른 우측 로지스틱 함수의 그래프

연습문제 | 15.10 미니 프로젝트

직선 $ax+by=c$는 $z(x,y)=ax+by-c=0$을 만족하는 점의 집합이다. 7장에서 살펴봤듯이 $z(x,y)=ax+by-c$의 그래프는 3차원 공간에서 평면을 나타낸다. 따라서 직선 위의 점에서 특정 방향으로 이동할 때 $z(x,y)$값이 증가했다면 반대 방향으로 이동할 때 $z(x,y)$값이 감소한다. $z(x,y)$의 그라디언트는 $\nabla z(x,y)=(a,b)$이며, 벡터 (a,b)가 나타내는 방향으로 갈 때 $z(x,y)$가 가장 가파르게 증가함을 의미한다. 따라서 $(-a,-b)$ 방향으로 가면 $z(x,y)$가 가장 가파르게 감소한다. 두 방향 모두 직선이 뻗어나가는 방향과는 수직이다.

15.5.4 연습문제 정답 및 풀이

연습문제 | 15.11

gradient_descent3에서 결과를 리턴하기 직전에 print(steps)라는 한 줄을 삽입하기만 하면 된다.

```
def gradient_descent3(f,xstart,ystart,zstart,tolerance=1e-6,max_steps=1000):
    ...
    print(steps)
    return x,y,z
```

다음과 같이 경사하강법을 실행하자.

```
gradient_descent3(logistic_cost,1,1,1,max_steps=8000)
```

출력된 수는 7244이며, 이는 이 알고리즘이 7,244단계 후에 수렴함을 의미한다.

연습문제 | 15.12 미니 프로젝트

임의 차원의 벡터를 수의 리스트로 모델링하자. 벡터 $\mathbf{v}=(v_1,v_2,...,v_n)$에서의 i번째 좌표에 대한 편도함수를 구하려면 i번째 좌표 x_i에 대한 도함수를 구해야 한다. 그러려면 다음 함수 f를 먼저 얻어야 한다.

$$f(v_1,v_2,...,v_{i-1},x_i,v_{i+1},...,v_n)$$

다시 말해서 벡터 $\mathbf{v}$에서 i번째 성분을 제외한 나머지 성분을 모두 f에 대입해서 변수 x_i만 남겨두어야 한다. 이를 통해 단일 변수 x_i에 대한 함수를 얻었다면 이 함수에 대한 도함수가 i번째 좌표에 대한 편도함수이다. 편도함수를 구하는 위 과정을 코드로 작성하면 다음과 같다.

```
def partial_derivative(f,i,v,**kwargs):
    def cross_section(x):
        arg = [(vj if j != i else x) for j,vj in enumerate(v)]
        return f(*arg)
    return approx_derivative(cross_section, v[i], **kwargs)
```

파이썬에서 좌표는 0번째부터 시작한다는 점과 f에 대한 입력 차수는 $\mathbf{v}$의 길이로부터 알 수 있음을 주의하기 바란다.

남은 작업은 비교적 쉽다. 그라디언트를 만들려면 n개의 편도함수를 구한 뒤 리스트에 순서대로 넣으면 된다.

```
def approx_gradient(f,v,dx=1e-6):
    return [partial_derivative(f,i,v) for i in range(0,len(v))]
```

이제 차원에 상관없는 경사하강법을 수행하기 위해 x,y,z와 같이 명명된 좌표 변수를 조작하는 부분을 모두 $\mathbf{v}$라는 좌표 벡터를 나타내는 리스트에 대한 리스트 연산으로 대체하겠다.

```
def gradient_descent(f,vstart,tolerance=1e-6,max_steps=1000):
    v = vstart
    grad = approx_gradient(f,v)
    steps = 0
    while length(grad) > tolerance and steps < max_steps:
        v  = [(vi - 0.01 * dvi) for vi,dvi in zip(v,grad)]
        grad = approx_gradient(f,v)
        steps += 1
    return v
```

문제에서 주어진 테스트 함수를 구현할 때에도 임의 차원의 입력 벡터를 받아서 각 성분에서 1씩 뺀 값의 제곱을 합한 결과를 리턴하는 일반화된 함수로 작성하면 된다.

```
def sum_squares(*v):
    return sum([(x-1)**2 for x in v])
```

이 함수는 제곱의 합이며 제곱은 0보다 작을 수 없으므로 이 함수도 0보다 작을 수 없다. 0은 입력 벡터 $\mathbf{v}$의 모든 성분이 1이면 얻을 수 있고, 이 결과가 최솟값이다. 우리가 작성한 경사하강법은 이를 (수치 오차 범위 내에서) 확인해주며, 다음 결과를 봐도 아무 문제가 없다! 단, 다음은 시작 벡터 $\mathbf{v}$가 5차원이므로 계산 과정에서 등장하는 모든 벡터가 자동으로 5차원임에 주의하라.

```
>>> v = [2,2,2,2,2]
>>> gradient_descent(sum_squares,v)
[1.0000002235452137,
 1.0000002235452137,
 1.0000002235452137,
 1.0000002235452137,
 1.0000002235452137]
```

연습문제 | 15.13 미니 프로젝트

이 함수에 대해 경사하강법을 실행하면 수렴하지 않는 것처럼 보인다. 결정 경계가 수렴해도 a, b, c값이 한도 끝도 없이 계속 증가하기 때문이다. 경사하강법이 더 많은 로지스틱 함수를 탐색하는 과정에서 같은 방향의 직선이지만 로지스틱 함수가 무한히 가팔라지는 상황임을 의미한다. 이러한 상황은 경사하강법이 대부분의 점에 피팅하도록 유인하면서도 이미 잘못 분류된 점을 무시하기 때문에 발생한다. 앞에서 언급했듯이 로지스틱 함수가 가장 높은 확신 정도를 갖는 데이터 포인트가 사실은 부정확하게 분류된다면, 이 상황에 패널티를 부여해 해결할 수 있다. 우리가 만든 `logistic_cost` 함수는 이런 식으로 패널티를 부여하기 때문에 잘 동작한다.

16.2.4 연습문제 정답 및 풀이

연습문제 | 16.1

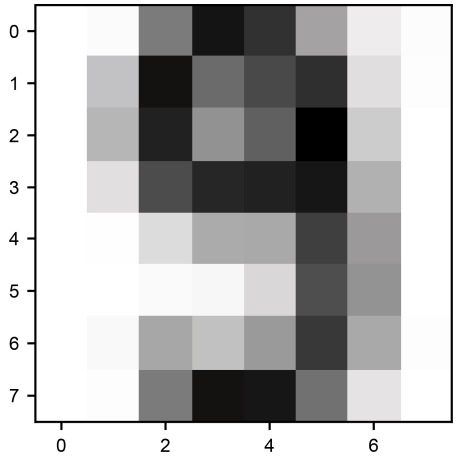
이 배열에서 가장 큰 수는 `9.98060276e-01`로 약 0.998이며, 배열의 다섯 번째 성분이자 그 인덱스는 4이다. 이 출력 결과는 분류기가 이미지를 4로 분류했음을 뜻한다.

연습문제 | 16.2 미니 프로젝트

아래 코드는 정수 i를 입력으로 받아 숫자 i를 나타내는 데이터셋의 이미지들 전체의 평균을 낸다. 숫자 이미지는 덧셈과 스칼라곱을 지원하는 NumPy 배열로 표현되어 있기에, 일반적인 파이썬 `sum` 함수와 나눗셈 연산자를 써서 평균을 낼 수 있다.

```
def average_img(i):
    imgs = [img for img,target in zip(digits.images[1000:],
digits.target[1000:]) if target==i]
    return sum(imgs) / len(imgs)
```

위의 코드를 입력한 뒤 `average_image(9)`를 실행하면 숫자 9를 나타내는 모든 이미지의 평균을 나타내는 8×8 행렬을 계산한다. 그 모습은 다음과 같다.



연습문제 | 16.3 미니 프로젝트

```
avg_digits = [np.matrix.flatten(average_img(i)) for i in range(10)]
def compare_to_avg(v):
    return [np.dot(v,avg_digits[i]) for i in range(10)]
```

이 분류기는 테스트 데이터셋에서 85%의 확률로 정확하게 답한다. 나쁘지 않은 결과이다!

```
>>> test_digit_classify(compare_to_avg)
0.853
```

16.3.5 연습문제 정답 및 풀이

연습문제 | 16.4

위 첨자는 계층을 나타내고 아래 첨자는 계층 내의 뉴런을 나타낸다. 따라서 활성 a_2^3는 계층 3의 두 번째 뉴런에 대응한다. 그림에서 이 활성의 값은 0.9이다.

연습문제 | 16.5

계층 5에 있는 10개의 뉴런은 각각 계층 6에 있는 12개의 뉴런과 연결되어 있다. 따라서 연결은 총 120개이다.

연습문제 | 16.6

l은 연결이 도달하는 계층을 나타내므로 $l=5$이다. 두 인덱스 i와 j는 각각 계층 l의 뉴런과 $l-1$의 뉴런에 대응하므로 $i=7$이고 $j=3$이다. 따라서 이 가중치는 w_{73}^5라고 부른다.

연습문제 | 16.7

이러한 가중치는 존재하지 않는다. 계층 3의 세 번째 뉴런에 도달해야 하는데 계층 3에는 뉴런이 두 개뿐이다.

연습문제 | 16.8

직전 계층 활성은 a_1^2, a_2^2, a_2^3이며, a_1^3에 연결된 가중치는 w_{11}^3, w_{12}^3, w_{13}^3이다. 활성 a_1^3에 대한 편향은 b_1^3이라고 기재하면 수식은 다음과 같다.

$$a_1^3 = \sigma(w_{11}^3 a_1^2 + w_{12}^3 a_2^2 + w_{13}^3 a_3^2 + b_1^3)$$

연습문제 | 16.9 미니 프로젝트

구현은 이 책의 소스 코드에서 살펴볼 수 있다.

16.5.4 연습문제 정답 및 풀이

연습문제 | 16.11

테스트 예시의 시작점을 알려주기 위해 start라는 키워드 인자를 사용하였다. 기존과 마찬가지로 키워드 인자 test_count는 테스트할 예시의 개수를 나타낸다.

```
def test_digit_classify(classifier,start=0,test_count=1000):
    correct = 0
    end = start + test_count
    for img, target in zip(digits.images[start:end],
digits.target[start:end]):
        v  = np.matrix.flatten(img) / 15
        output = classifier(v)
        answer = list(output).index(max(output))
        if answer == target:
            correct += 1
    return (correct/test_count)
```

테스트할 데이터셋에서 끝을 나타내는 인덱스 end를 계산한다.

시작 인덱스 start와 (실제로는 포함되지 않는) 마지막 인덱스 end 사이에서 테스트 데이터셋을 순회한다.

학습이 끝난 다층 퍼셉트론은 새로운 숫자 이미지의 96.2%를 올바르게 식별한다.

```
>>> test_digit_classify(sklearn_trained_classify,start=1000,test_count=500)
0.962
```

연습문제 | 16.12

먼저 주어진 숫자에 대해 이상적인 출력 벡터를 알려주는 함수를 작성하자. 예를 들어 출력 벡터 y는 숫자 5에 대해 인덱스 5에서만 1이고 나머지는 0이다.

```
def y_vec(digit):
    return np.array([1 if i == digit else 0 for i in range(0,10)])
```

테스트할 예시의 비용은 분류기의 출력과 이상적인 출력 결과 간 거리의 제곱이다. 이는 각 좌표의 차를 제곱한 뒤 합한 것이다.

```
def cost_one(classifier,x,i):
    return sum([(classifier(x)[j] - y_vec(i)[j])**2 for j in range(10)])
```

앞에서 분류기의 비용은 훈련 예시 1,000개에 대한 평균 비용으로 정의하였다.

```
def total_cost(classifier):
    return sum([cost_one(classifier,x[j],y[j]) for j in range(1000)])/1000.
```

예상했듯이 예측 정확도가 10%인 랜덤 생성 다층 퍼셉트론은 예측 정확도가 100%인 scikit-learn이 생성한 다층 퍼셉트론에 비해 비용이 훨씬 높다.

```
>>> total_cost(nn.evaluate)
8.995371023185067
>>> total_cost(sklearn_trained_classify)
5.670512721637246e-05
```

연습문제 | 16.13 미니 프로젝트

실제로 해보면 문제를 하나 발견할텐데, 두 가중치 행렬이 16×64와 10×16 크기라고 생각했겠지만 `MLPClassifier`의 `coefs_` 속성은 각각 64×16 행렬과 16×10 행렬이다. 지금까지 가중치 행렬의 행을 저장하는 관례를 따랐는데, scikit-learn은 가중치 행렬의 열을 저장하는 관례를 따른 것 같다. 이를 고치는 간단한 방법이 있다.

NumPy 배열 클래스에는 [연습문제 5.18]에서 다룬 행렬의 **전치**(transpose), 즉 행렬의 행이 열이 되도록 성분을 재배치한 행렬을 리턴하는 `T`라는 속성이 있다. 이 기교를 염두에 두고 가중치와 편향을 신경망에 대입하여 테스트하면 다음과 같다.

```
>>> nn = MLP([64,16,10])
>>> nn.weights = [w.T for w in mlp.coefs_]
>>> nn.biases = mlp.intercepts_
>>> test_digit_classify(nn.evaluate, start=1000,
                        test_count=500) 0.962
```

scikit-learn의 다중 퍼셉트론에서 추출한 가중치 행렬의 전치행렬을 구해서 관례에 따르도록 만든 뒤 가중치 행렬에 설정한다.

scikit-learn 다중 퍼셉트론의 편향을 신경망의 편향에 설정한다.

새로 설정한 가중치와 편향을 설정한 신경망으로 분류 작업의 성능을 테스트한다.

이 분류기는 훈련용 데이터셋 뒤에 테스트한 500개의 이미지에 대해서 96.2%의 정확도를 보이며, 이는 scikit-learn에서 직접 생성한 다층 퍼셉트론의 정확도와 같다.

16.6.3 연습문제 정답 및 풀이

연습문제 | 16.14 미니 프로젝트

SymPy에서 이 도함수에 대한 식을 간단히 구할 수 있다.

```
>>> from sympy import *
>>> X = symbols('x')
>>> diff(1 / (1+exp(-X)),X)
exp(-x)/(1 + exp(-x))**2
```

이를 수학식으로 표기하면 다음과 같다.

$$\frac{e^{-x}}{(1+e^{-x})^2}=\frac{e^{-x}}{1+e^{-x}}\cdot\frac{1}{1+e^{-x}}=\frac{e^{-x}}{1+e^{-x}}\cdot\sigma(x)$$

이 식이 $\sigma(x)(1-\sigma(x))$와 같음을 보이려면 우리에게 별로 중요하지 않은 대수학적 기법을 사용해야 하지만, 식이 성립함을 확인하는 건 중요하다. e^{-x}에 1을 더하고 빼도 같다는 점을 이용하면 다음을 얻는다.

$$\begin{aligned}\frac{e^{-x}}{(1+e^{-x})^2}=\frac{e^{-x}}{1+e^{-x}}\cdot\sigma(x)&=\frac{1+e^{-x}-1}{1+e^{-x}}\cdot\sigma(x)\\&=\left(\frac{1+e^{-x}}{1+e^{-x}}-\frac{1}{1+e^{-x}}\right)\cdot\sigma(x)\\&=\left(1-\frac{1}{1+e^{-x}}\right)\cdot\sigma(x)\\&=\sigma(x)(1-\sigma(x))\end{aligned}$$

C.4 연습문제 정답 및 풀이

연습문제 | C.1

이 책의 소스 코드 중 draw_model.py에 정답이 수록되어 있으며, 사용 예제는 draw_teapot_glrotatef.py를 확인하자.

연습문제 | C.2

프레임률에 따라 다르다. 주어진 `glRotatef`를 호출하면 매 프레임마다 1° 씩 관점이 회전한다. 60fps를 가정하면 초당 60° 씩 회전하므로 1바퀴(360°) 회전을 하는 데 6초가 걸린다.

연습문제 | C.3 미니 프로젝트

소스 코드 중 teapot.py 파일을 살펴보면 `load_triangles()`의 구현을 얻을 수 있다.

연습문제 | C.4 미니 프로젝트

소스 코드 중 animated_octahedron.py 파일에는 매 프레임마다 `glRotatef`의 `angle` 매개변수를 업데이트해서 초당 360/5 = 72° 만큼 8면체를 회전시키는 예제가 있다. 찻주전자나 8면체에 대해 비슷한 수정을 해볼 수 있다.