3.1 선형대수와 해석기하의 기초

선형대수는 숫자 데이터의 계산에만 사용되는 것이 아니다. 직선과 화살표, 이미지 등을 다루는 기하학에서도 선형대수는 중요한 역할을 한다. 이 절에서는 선형대수를 기하학에서 어떻게 응용하고 선형대수의 연산이 기하학적으로 어떤 의미를 가지는지 알아본다.

벡터의 기하학적 의미

N차원 벡터 a는 N차원의 공간에서

- 벡터 a의 값으로 표시되는 **점(point)** 또는
- 원점과 벡터 a의 값으로 표시되는 점을 연결한 **화살표(arrow)**

라고 생각할 수 있다.

예를 들어 2차원 벡터

$$a = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} \tag{3.1.1}$$

는 2차원 공간에서 x 좌표가 a_1, y 좌표가 a_2인 점으로 생각할 수도 있고 또는 원점에서 이 점을 가리키는 화살표로 생각할 수도 있다. 벡터를 화살표로 생각하는 경우에는 길이와 방향을 고정시킨 채 **평행이동**할 수 있다.

(앞으로 나오는 그림은 모두 맷플롯리브 패키지로 그린 그림이다. 이 코드는 파이썬으로 이러한 그림도 제작할 수 있다는 것을 보이기 위한 것일 뿐 이번 절의 내용과는 관계없으므로 그림 코드의 내용은 무시해도 된다.)

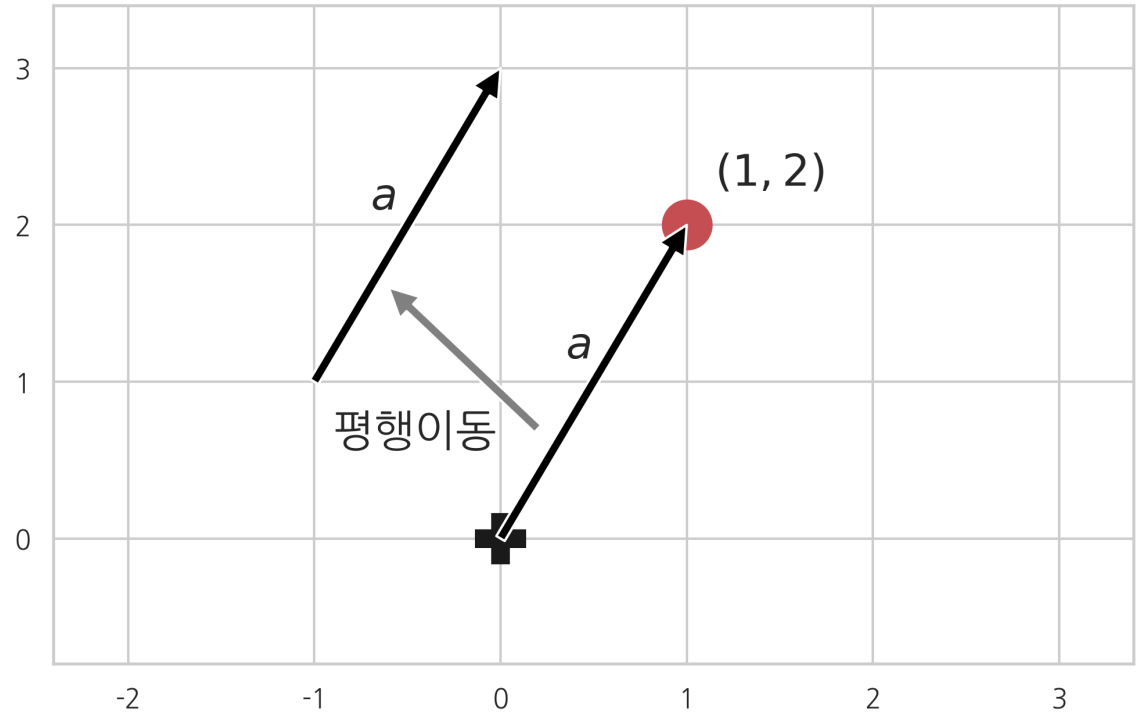
In [1]:

```
import numpy as np
import matplotlib.pylab as plt

plt.rc("font", size=18)  # 그림의 폰트 크기를 18로 고정
gray = {"facecolor": "gray"}
black = {"facecolor": "black"}
red = {"facecolor": "red"}
green = {"facecolor": "green"}
blue = {"facecolor": "blue"}
```

In [2]:

```
a = np.array([1, 2])
plt.plot(0, 0, 'kP', ms=20)
plt.plot(a[0], a[1], 'ro', ms=20)
plt.annotate('', xy=[-0.6, 1.6], xytext=(0.2, 0.7), arrowprops=gray)
plt.annotate('', xy=a, xytext=(0, 0), arrowprops=black)
plt.annotate('', xy=a + [-1, 1], xytext=(-1, 1), arrowprops=black)
plt.text(0.35, 1.15, "$a$")
plt.text(1.15, 2.25, "$(1,2)$")
plt.text(-0.7, 2.1, "$a$")
plt.text(-0.9, 0.6, "평행이동")
plt.xticks(np.arange(-2, 4))
plt.yticks(np.arange(-1, 4))
plt.xlim(-2.4, 3.4)
plt.ylim(-0.8, 3.4)
plt.show()
```



벡터의 길이

벡터 a의 길이는 놈(norm) $\|a\|$으로 정의한다.

$$\|a\| = \sqrt{a^T a} = \sqrt{a_1^2 + \cdots + a_N^2} \tag{3.1.2}$$

넘파이 linalg 서브 패키지의 `norm()` 명령으로 벡터의 길이를 계산할 수 있다. 위에서 예로 든 2차원 벡터 $a = [a_1 \;\; a_2]^T$의 길이는 $\sqrt{5} \approx 2.236$이다.

In [3]:

```
a = np.array([1, 2])
np.linalg.norm(a)
```

Out[3]:

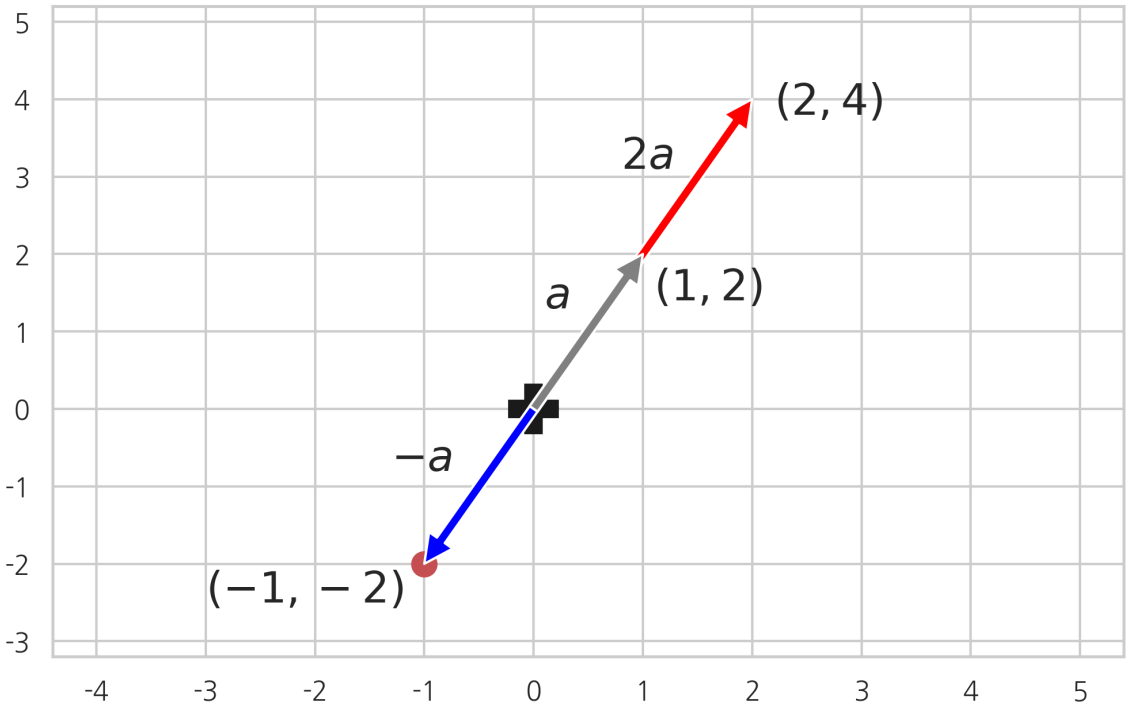
2.23606797749979

스칼라와 벡터의 곱

양의 실수와 벡터를 곱하면 벡터의 방향은 변하지 않고 실수의 크기만큼 벡터의 길이가 커진다. 만약 음의 실수를 곱하면 벡터의 방향이 반대가 된다.

In [4]:

```
a = np.array([1, 2])
b = 2 * a
c = -a
plt.annotate('', xy=b, xytext=(0, 0), arrowprops=red)
plt.text(0.8, 3.1, "$2a$")
plt.text(2.2, 3.8, "$(2, 4)$")
plt.annotate('', xy=a, xytext=(0, 0), arrowprops=gray)
plt.text(0.1, 1.3, "$a$")
plt.text(1.1, 1.4, "$(1, 2)$")
plt.plot(c[0], c[1], 'ro', ms=10)
plt.annotate('', xy=c, xytext=(0, 0), arrowprops=blue)
plt.text(-1.3, -0.8, "$-a$")
plt.text(-3, -2.5, "$(-1, -2)$")
plt.plot(0, 0, 'kP', ms=20)
plt.xticks(np.arange(-5, 6))
plt.yticks(np.arange(-5, 6))
plt.xlim(-4.4, 5.4)
plt.ylim(-3.2, 5.2)
plt.show()
```



단위벡터

길이가 1인 벡터를 **단위벡터(unit vector)**라고 한다. 예를 들어 다음과 같은 벡터들은 모두 단위벡터다.

$$a = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \quad b = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \quad c = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix} \tag{3.1.3}$$

영벡터가 아닌 임의의 벡터 x에 대해 다음 벡터는 벡터 x와 같은 방향을 가리키는 단위벡터가 된다.

$$\frac{x}{\|x\|} \tag{3.1.4}$$

In [5]:

```
a = np.array([1, 0])
b = np.array([0, 1])
c = np.array([1/np.sqrt(2), 1/np.sqrt(2)])
np.linalg.norm(a), np.linalg.norm(b), np.linalg.norm(c)
```

Out[5]:

```
(1.0, 1.0, 0.9999999999999999)
```

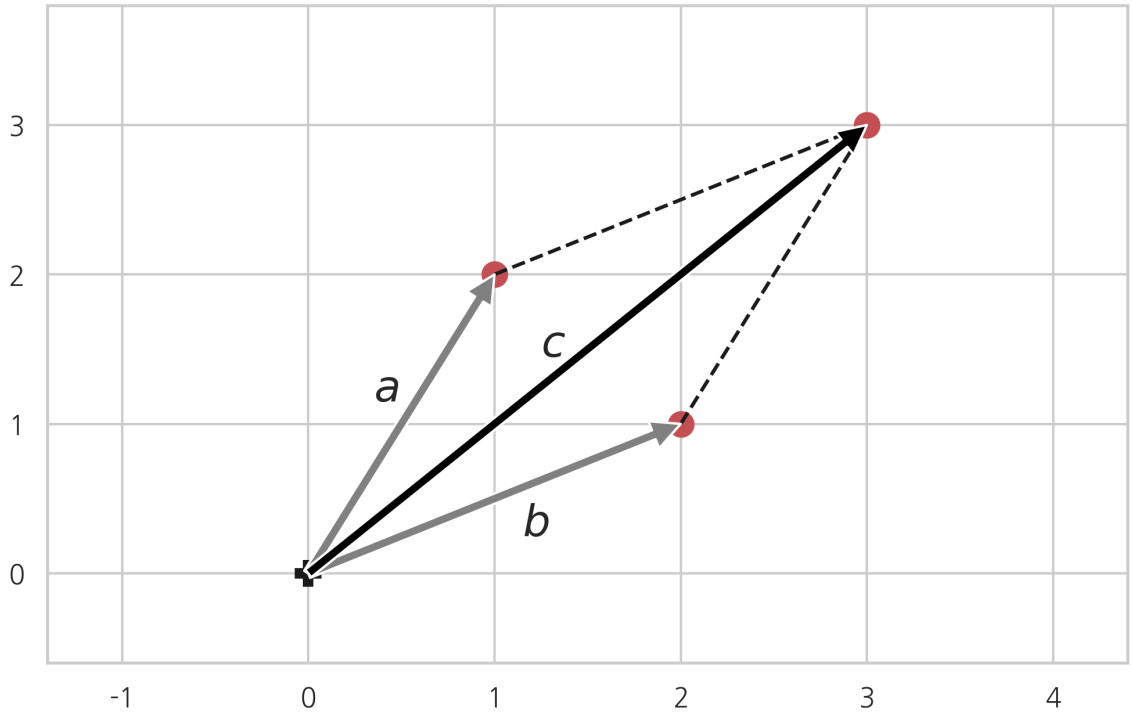
벡터의 합

벡터와 벡터의 합도 벡터가 된다. 이때 **두 벡터의 합은 그 두 벡터를 이웃하는 변으로 가지는 평행사변형의 대각선 벡터**가 된다.

$$a = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad \rightarrow \quad c = a + b = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \tag{3.1.5}$$

In [6]:

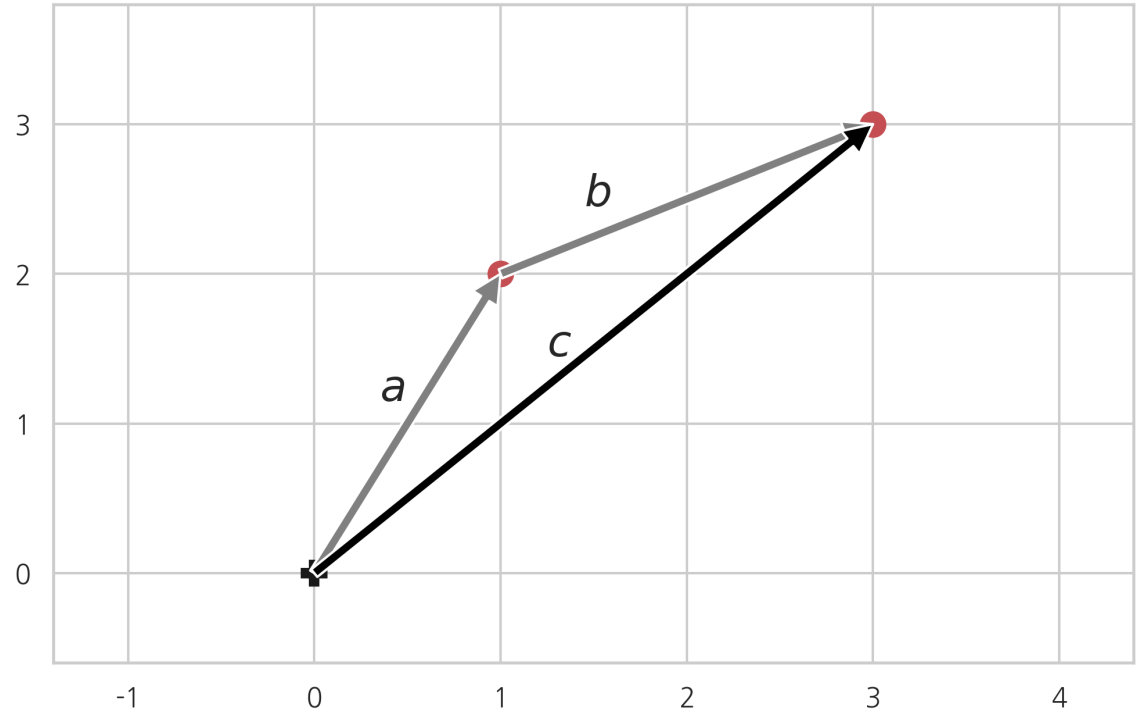
```
a = np.array([1, 2])
b = np.array([2, 1])
c = a + b
plt.annotate('', xy=a, xytext=(0, 0), arrowprops=gray)
plt.annotate('', xy=b, xytext=(0, 0), arrowprops=gray)
plt.annotate('', xy=c, xytext=(0, 0), arrowprops=black)
plt.plot(0, 0, 'kP', ms=10)
plt.plot(a[0], a[1], 'ro', ms=10)
plt.plot(b[0], b[1], 'ro', ms=10)
plt.plot(c[0], c[1], 'ro', ms=10)
plt.plot([a[0], c[0]], [a[1], c[1]], 'k--')
plt.plot([b[0], c[0]], [b[1], c[1]], 'k--')
plt.text(0.35, 1.15, "$a$")
plt.text(1.15, 0.25, "$b$")
plt.text(1.25, 1.45, "$c$")
plt.xticks(np.arange(-2, 5))
plt.yticks(np.arange(-1, 4))
plt.xlim(-1.4, 4.4)
plt.ylim(-0.6, 3.8)
plt.show()
```



또는 벡터를 더하고자 하는 벡터의 끝점으로 평행이동했을 때 이동한 벡터가 가리키는 점의 위치로 생각할 수도 있다.

In [7]:

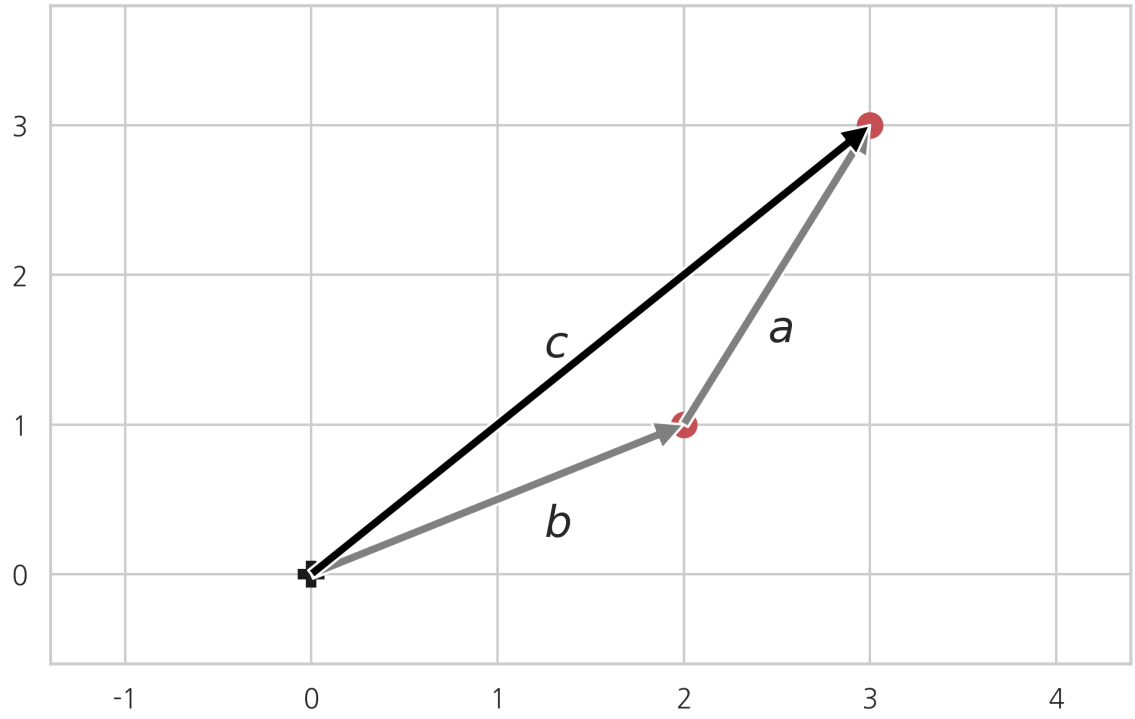
```
a = np.array([1, 2])
b = np.array([2, 1])
c = a + b
plt.annotate('', xy=a, xytext=(0, 0), arrowprops=gray)
plt.annotate('', xy=c, xytext=a, arrowprops=gray)
plt.annotate('', xy=c, xytext=(0, 0), arrowprops=black)
plt.plot(0, 0, 'kP', ms=10)
plt.plot(a[0], a[1], 'ro', ms=10)
plt.plot(c[0], c[1], 'ro', ms=10)
plt.text(0.35, 1.15, "$a$")
plt.text(1.45, 2.45, "$b$")
plt.text(1.25, 1.45, "$c$")
plt.xticks(np.arange(-2, 5))
plt.yticks(np.arange(-1, 4))
plt.xlim(-1.4, 4.4)
plt.ylim(-0.6, 3.8)
plt.show()
```



둘 중 어느 벡터를 평행이동해도 결과는 마찬가지다.

In [8]:

```
a = np.array([1, 2])
b = np.array([2, 1])
c = a + b
plt.annotate('', xy=b, xytext=(0, 0), arrowprops=gray)
plt.annotate('', xy=c, xytext=b, arrowprops=gray)
plt.annotate('', xy=c, xytext=(0, 0), arrowprops=black)
plt.plot(0, 0, 'kP', ms=10)
plt.plot(b[0], b[1], 'ro', ms=10)
plt.plot(c[0], c[1], 'ro', ms=10)
plt.text(2.45, 1.55, "$a$")
plt.text(1.25, 0.25, "$b$")
plt.text(1.25, 1.45, "$c$")
plt.xticks(np.arange(-2, 5))
plt.yticks(np.arange(-1, 4))
plt.xlim(-1.4, 4.4)
plt.ylim(-0.6, 3.8)
plt.show()
```



벡터의 선형조합

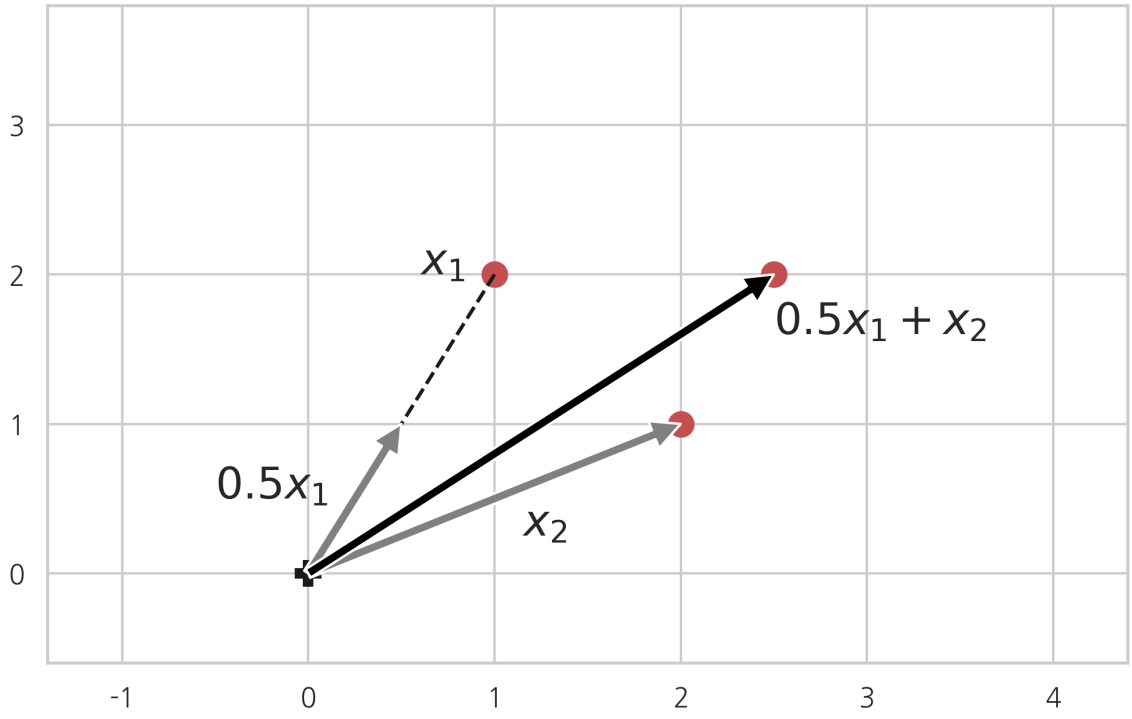
지금까지 벡터의 스칼라곱이 어떤 새로운 벡터가 되고 두 벡터의 합이 어떤 새로운 벡터가 되는지 살펴보았다. 여러 개의 벡터를 스칼라곱을 한 후 더한 것을 선형조합(linear combination)이라고 한다.

$$c_1 x_1 + c_2 x_2 + \cdots + c_N x_N \tag{3.1.6}$$

이 식에서 $c_1, \cdots, c_N$은 스칼라 계수다.

In [9]:

```
x1 = np.array([1, 2])
x2 = np.array([2, 1])
x3 = 0.5 * x1 + x2
plt.annotate('', xy=0.5*x1, xytext=(0, 0), arrowprops=gray)
plt.annotate('', xy=x2, xytext=(0, 0), arrowprops=gray)
plt.annotate('', xy=x3, xytext=(0, 0), arrowprops=black)
plt.plot(0, 0, 'kP', ms=10)
plt.plot(x1[0], x1[1], 'ro', ms=10)
plt.plot(x2[0], x2[1], 'ro', ms=10)
plt.plot(x3[0], x3[1], 'ro', ms=10)
plt.plot([x1[0], 0], [x1[1], 0], 'k--')
plt.text(0.6, 2.0, "$x_1$")
plt.text(-0.5, 0.5, "$0.5x_1$")
plt.text(1.15, 0.25, "$x_2$")
plt.text(2.5, 1.6, "$0.5x_1 + x_2$")
plt.xticks(np.arange(-2, 5))
plt.yticks(np.arange(-1, 4))
plt.xlim(-1.4, 4.4)
plt.ylim(-0.6, 3.8)
plt.show()
```



연습 문제 3.1.1

벡터 x_1, x_2가 다음과 같을 때, $c_1x_1 + c_2x_2$가 다음 벡터와 같아지는 선형조합 계수 c_1, c_2를 찾아라. (힌트: 연립방정식의 해를 이용한다.)

$$x_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \tag{3.1.7}$$

(1)

$$c_1x_1 + c_2x_2 = \begin{bmatrix} 3 \\ 1 \end{bmatrix} \tag{3.1.8}$$

(2)

$$c_1x_1 + c_2x_2 = \begin{bmatrix} -1 \\ -1 \end{bmatrix} \tag{3.1.9}$$

벡터의 차

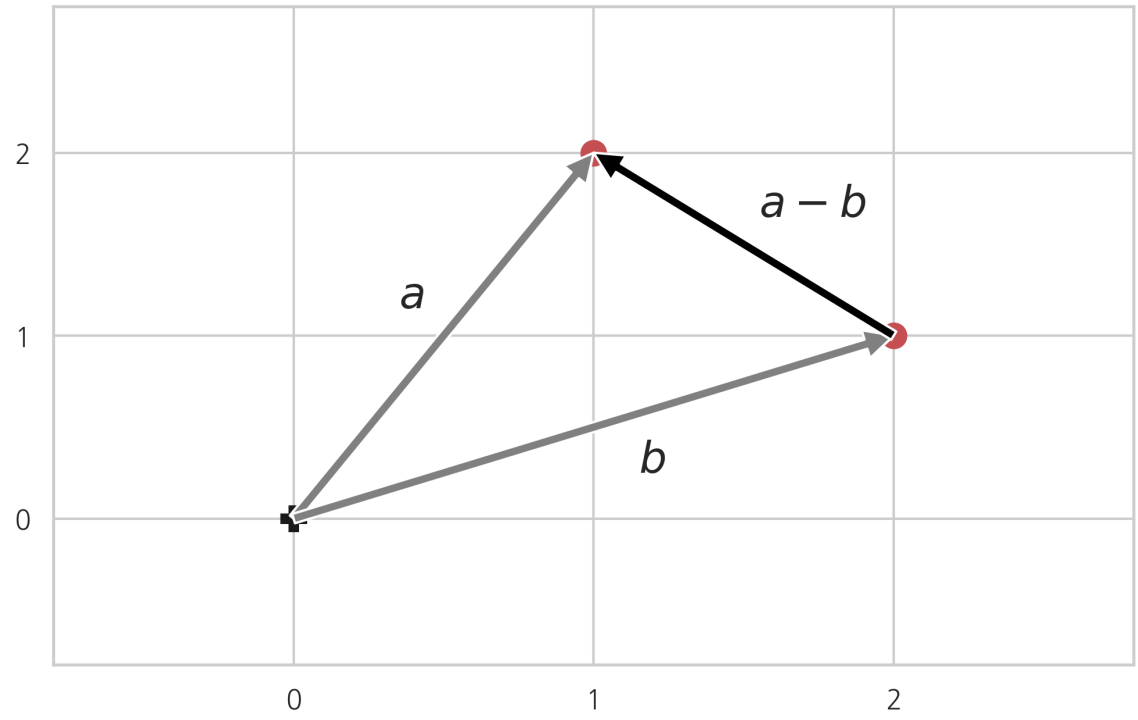
벡터의 차 $a - b = c$는 벡터 b가 가리키는 점으로부터 벡터 a가 가리키는 점을 연결하는 벡터다. 그 이유는 벡터 b에 벡터 $a - b$를 더하면, 즉 벡터 b와 벡터 $a - b$를 연결하면 벡터 a가 되어야 하기 때문이다.

$$a - b = c \tag{3.1.10}$$

$$b + c = b + (a - b) = a \tag{3.1.11}$$

In [10]:

```
a = np.array([1, 2])
b = np.array([2, 1])
c = a - b
plt.annotate('', xy=a, xytext=(0, 0), arrowprops=gray)
plt.annotate('', xy=b, xytext=(0, 0), arrowprops=gray)
plt.annotate('', xy=a, xytext=b, arrowprops=black)
plt.plot(0, 0, 'kP', ms=10)
plt.plot(a[0], a[1], 'ro', ms=10)
plt.plot(b[0], b[1], 'ro', ms=10)
plt.text(0.35, 1.15, "$a$")
plt.text(1.15, 0.25, "$b$")
plt.text(1.55, 1.65, "$a-b$")
plt.xticks(np.arange(-2, 5))
plt.yticks(np.arange(-1, 4))
plt.xlim(-0.8, 2.8)
plt.ylim(-0.8, 2.8)
plt.show()
```



Word2Vec

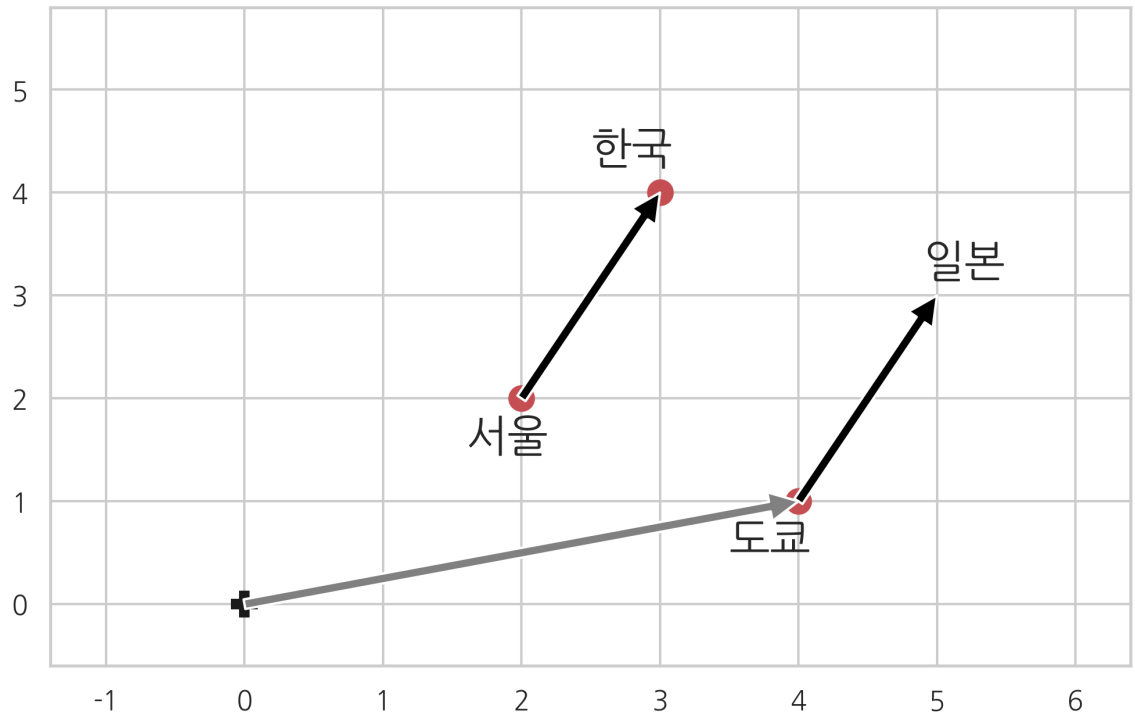
나중에 인공신경망 부분에서 공부하게 될 **word2vec** 방법을 이용하면 단어(word)를 공간에서 점 또는 벡터(vector)로 표현할 수 있다. word2vec으로 만들어진 벡터는 단어의 의미에 따라 다음처럼 평행사변형 관계를 가질 수 있다.

$$\text{일본} = \text{도쿄} + (\text{한국} - \text{서울}) \tag{3.1.12}$$

`한국 - 서울` 은 `서울` 에서 `한국` 으로 향하는 벡터다. 즉 의미론적으로 **수도 이름을 나라 이름으로 바꾸는 행위(action)**에 비유할 수 있다. 이러한 행위를 `도쿄` 에 대해서 적용한 결과가 `도쿄 + (한국 - 서울)` 이다. word2vec 학습 결과에서 이렇게 계산한 위치에 가장 가까이 있는 단어를 찾으면 `도쿄` 가 나온다.

In [11]:

```
a = np.array([2, 2])
b = np.array([3, 4])
c = np.array([4, 1])
d = a + (c - a)
e = b + (c - a)
plt.annotate('', xy=b, xytext=a, arrowprops=black)
plt.annotate('', xy=e, xytext=d, arrowprops=black)
plt.annotate('', xy=c, xytext=[0, 0], arrowprops=gray)
plt.plot(0, 0, 'kP', ms=10)
plt.plot(a[0], a[1], 'ro', ms=10)
plt.plot(b[0], b[1], 'ro', ms=10)
plt.plot(c[0], c[1], 'ro', ms=10)
plt.text(1.6, 1.5, "서울")
plt.text(2.5, 4.3, "한국")
plt.text(3.5, 0.5, "도쿄")
plt.text(4.9, 3.2, "일본")
plt.xticks(np.arange(-2, 7))
plt.yticks(np.arange(-1, 6))
plt.xlim(-1.4, 6.4)
plt.ylim(-0.6, 5.8)
plt.show()
```



연습 문제 3.1.2

남자배우, 여자배우, 남자, 여자, 이렇게 4가지 단어에 대응하는 4개의 벡터에 대해 위와 같은 관계가 성립한다고 가정하자. 다음 식을 완성하라.

> 남자배우 = 여자배우 + ?

유클리드 거리

두 벡터가 가리키는 점 사이의 거리를 **유클리드 거리(Euclidean distance)**라고 한다. 두 벡터의 유클리드 거리는 **벡터의 차의 길이**로 구할 수 있다.

벡터의 놈의 정의와 벡터의 차의 정의에서 유클리드 거리는 다음처럼 구한다.

$$\begin{aligned} \|a-b\| &= \sqrt{\sum_{i=1}(a_i-b_i)^2} \\ &= \sqrt{\sum_{i=1}(a_i^2 - 2a_ib_i + b_i^2)} \\ &= \sqrt{\sum_{i=1}a_i^2 + \sum_{i=1}b_i^2 - 2\sum_{i=1}a_ib_i} \\ &= \sqrt{\|a\|^2 + \|b\|^2 - 2a^Tb} \end{aligned} \tag{3.1.13}$$

즉,

$$\|a-b\|^2 = \|a\|^2 + \|b\|^2 - 2a^Tb \tag{3.1.14}$$

벡터의 내적과 삼각함수

두 벡터의 내적은 다음처럼 벡터의 길이 $\|a\|$, $\|b\|$ 와 두 벡터 사이의 각도 θ의 코사인 함수값으로 계산할 수도 있다.

$$a^Tb = \|a\|\|b\|\cos\theta \tag{3.1.15}$$

여기에서 $\cos\theta$는 **코사인(cosine)**이라고 하는 함수이다. 코사인은 사인(sine)이라고 하는 함수와 함께 정의할 수 있다. 사인과 코사인을 합쳐서 삼각함수라고 한다.

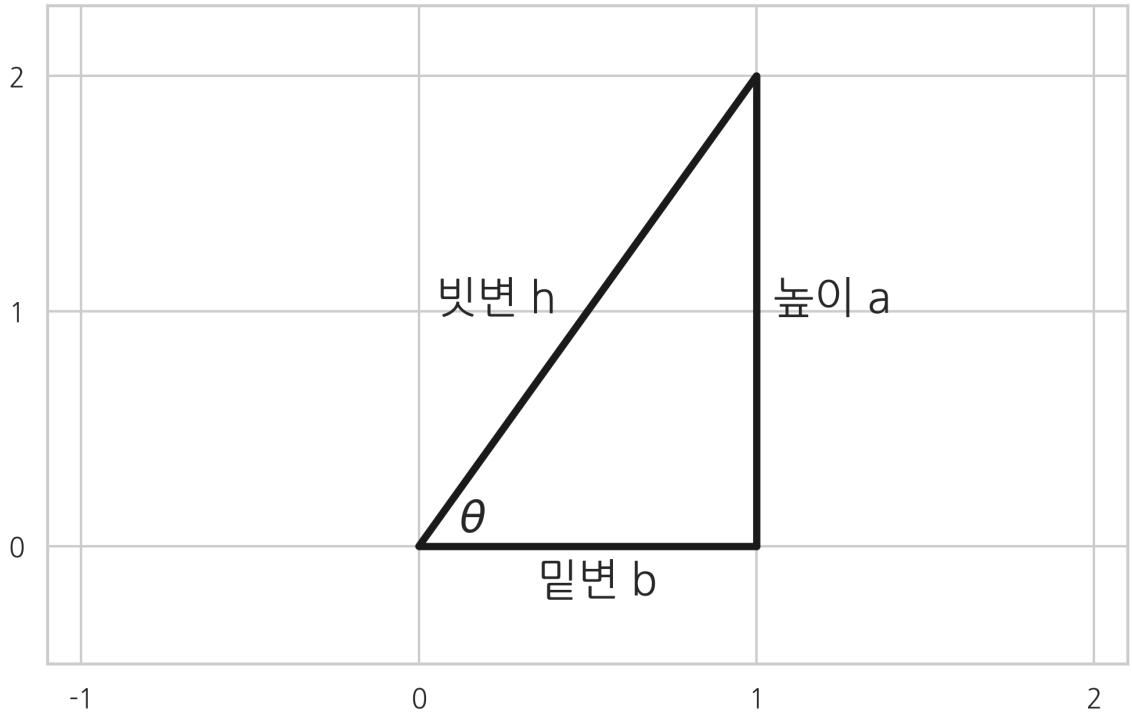
사인 $\sin\theta$의 값은 θ라는 각을 가지는 직각 삼각형에서 빗변(hypotenuse)과 높이(opposite)의 비율을 뜻한다. 코사인 $\cos\theta$의 값은 θ라는 각을 가지는 직각 삼각형에서 빗변(hypotenuse)과 밑변(adjacent)의 비율을 뜻한다.

$$\sin\theta = \frac{a}{h} \tag{3.1.16}$$

$$\cos\theta = \frac{b}{h} \tag{3.1.17}$$

In [12]:

```
plt.plot([0, 1], [0, 2], 'k-', lw=3)
plt.plot([0, 1], [0, 0], 'k-', lw=3)
plt.plot([1, 1], [0, 2], 'k-', lw=3)
plt.text(0.05, 1, "빗변 h")
plt.text(0.35, -0.2, "밑변 b")
plt.text(1.05, 1, "높이 a")
plt.text(0.12, 0.06, r"$\theta$")
plt.xticks(np.arange(-2, 4))
plt.yticks(np.arange(-1, 4))
plt.xlim(-1.1, 2.1)
plt.ylim(-0.5, 2.3)
plt.show()
```



$\sin\theta$의 값은 θ가 0에 가까워질수록 0에 가까워지고 θ가 90°에 가까워질수록 1에 가까워진다.

$$\sin 0^\circ = 0 \tag{3.1.18}$$

$$\sin 90^\circ = 1 \tag{3.1.19}$$

반대로 $\cos\theta$의 값은 θ가 0에 가까워질수록 1에 가까워지고 θ가 90°에 가까워질수록 0에 가까워진다.

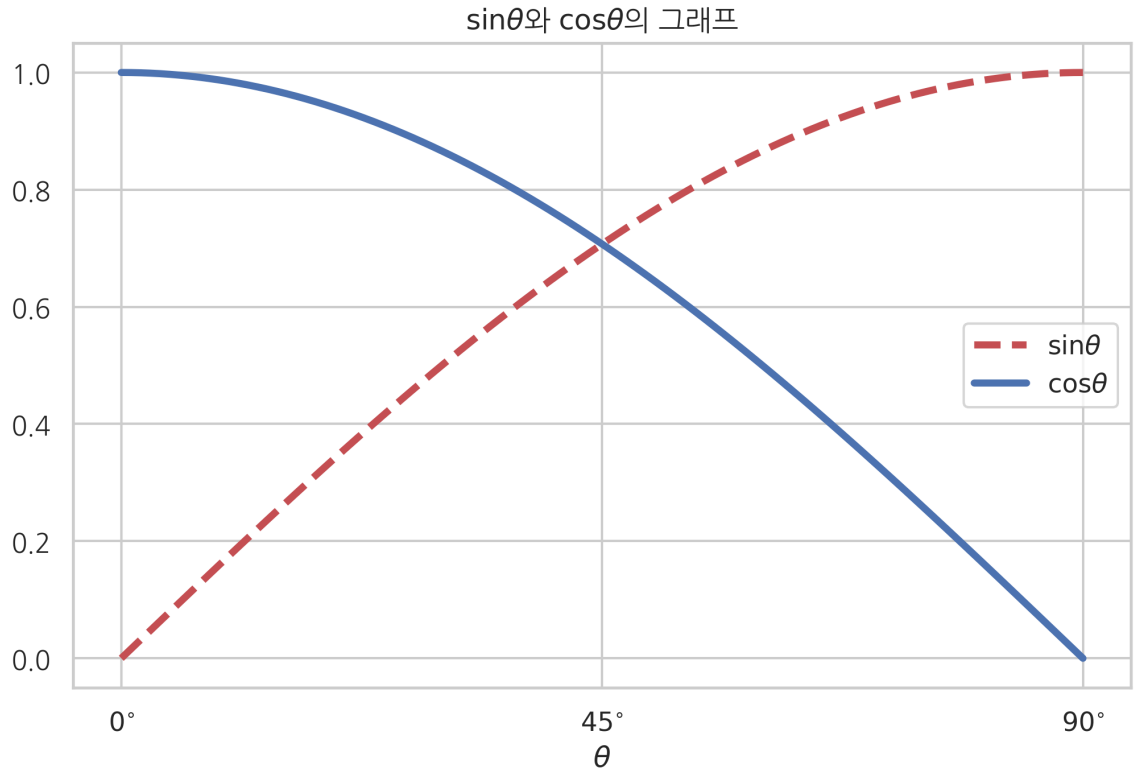
$$\cos 0^\circ = 1 \tag{3.1.20}$$

$$\cos 90^\circ = 0 \tag{3.1.21}$$

함수의 그래프로 표현하면 다음과 같다.

In [13]:

```
x = np.linspace(0, np.pi/2, 100)
y1 = np.sin(x)
y2 = np.cos(x)
plt.plot(x, y1, 'r--', lw=3,  label=r"$\sin\theta$")
plt.plot(x, y2, 'b-', lw=3, label=r"$\cos\theta$")
plt.legend()
plt.xticks([0, np.pi/4, np.pi/2], [r'$0^{\circ}$', r'$45^{\circ}$', r'$90^{\circ}$'])
plt.xlabel(r"$\theta$")
plt.title(r"$\sin\theta$와 $\cos\theta$의 그래프")
plt.show()
```



직교

두 벡터 a와 b가 이루는 각이 90도이면 서로 **직교(orthogonal)**라고 하며 $a \perp b$로 표시한다.

$\cos 90^\circ = 0$이므로 서로 직교인 두 벡터의 내적은 0이 된다.

$$a^T b = b^T a = 0 \quad \leftrightarrow \quad a \perp b \tag{3.1.22}$$

예를 들어 다음 두 벡터는 서로 직교한다.

$$a = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \;\; b = \begin{bmatrix} -1 \\ 1 \end{bmatrix} \quad \rightarrow \quad a^T b = \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ 1 \end{bmatrix} = -1 + 1 = 0 \tag{3.1.23}$$

In [14]:

```
a = np.array([1, 1])
b = np.array([-1, 1])
a @ b
```

Out[14]:

```
0
```

연습 문제 3.1.3

(1) 다음 벡터에 대해 직교하는 단위벡터를 찾아라.

$$x = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{3.1.24}$$

(2) 다음 벡터에 대해 직교하는 단위벡터를 찾아라.

$$x = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \tag{3.1.25}$$

(3) 다음 두 벡터에 대해 모두 직교하는 단위벡터를 찾아라.

$$x = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \tag{3.1.26}$$

정규직교

만약 N개의 단위벡터 $v_1, v_2, \cdots, v_N$가 서로 직교하면 **정규직교(orthonormal)**라고 한다.

$$\|v_i\| = 1 \quad \leftrightarrow \quad v_i^T v_i = 1 \tag{3.1.27}$$

$$v_i^T v_j = 0 \quad (i \neq j) \tag{3.1.28}$$

연습 문제 3.1.4

직교하는 두 N차원 벡터 a, b에 대해 다음 식이 성립함을 보여라

$$\|a + b\|^2 = \|a\|^2 + \|b\|^2 \tag{3.1.29}$$

$N = 2$일 때 이 식은 피타고라스의 정리가 된다.

연습 문제 3.1.5

정규직교하는 세 개의 3차원 벡터 v_1, v_2, v_3로 이루어진 행렬 V에 대해서 다음 등식이 성립함을 보여라.

$$V = \begin{bmatrix} v_1 & v_2 & v_3 \end{bmatrix} \tag{3.1.30}$$

(1)

$$V^T V = I \tag{3.1.31}$$

(2)

$$V^{-1} = V^T \tag{3.1.32}$$

코사인 유사도

두 벡터의 방향이 비슷할수록 벡터가 비슷하다고 간주하여 두 벡터 사이의 각의 코사인값을 **코사인 유사도(cosine similarity)**라고 한다. 코사인값은 각도가 0일때 가장 커지므로 두 벡터가 같은 방향을 가리키고 있으면 코사인 유사도가 최댓값 1을 가진다.

$$\text{cosine similarity} = \cos\theta = \frac{x^T y}{\|x\|\|y\|} \tag{3.1.33}$$

코사인 유사도는 나중에 공부할 추천시스템(recommender system)에서 사용자의 취향이 얼마나 비슷한지를 계산할 때 사용된다. 코사인 유사도를 이용하면 다음처럼 **코사인 거리(cosine distance)**도 정의할 수 있다.

$$\text{cosine distance} = 1 - \text{cosine similarity} = 1 - \frac{x^T y}{\|x\|\|y\|} \tag{3.1.34}$$

연습 문제 3.1.6

a, b, c, 3명의 사용자가 4개의 영화에 준 평점을 다음처럼 벡터로 표현하였다.

$$a = \begin{bmatrix} 4 \\ 5 \\ 2 \\ 2 \end{bmatrix}, \;\; b = \begin{bmatrix} 4 \\ 0 \\ 2 \\ 0 \end{bmatrix}, \;\; c = \begin{bmatrix} 2 \\ 2 \\ 0 \\ 1 \end{bmatrix} \tag{3.1.35}$$

(1) a, b, c 사이의 유클리드 거리를 구하라. 어느 두 사용자가 가장 가까운가? 또 어느 두 사용자가 가장 멀리 떨어져 있는가?

(2) a, b, c 사이의 코사인 거리를 구하라. 어느 두 사용자가 가장 가까운가? 또 어느 두 사용자가 가장 멀리 떨어져 있는가?

벡터의 분해와 성분

어떤 두 벡터 a, b의 합이 다른 벡터 c가 될 때 c가 두 벡터 **성분(component)** a, b으로 **분해(decomposition)**된다고 말한다.

연습 문제 3.1.7

다음 벡터를 두 개의 벡터로 분해하는 방법을 두 가지 이상 찾고 평면 위에 각각 화살표로 표기하라.

$$x = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \tag{3.1.36}$$

투영성분과 직교성분

벡터 a를 다른 벡터 b에 직교하는 성분과 벡터 b에 평행한 성분으로로 분해할 수 있는데, 평행한 성분을 벡터 b에 대한 **투영성분(projection)**, 벡터 b에 직교하는 성분을 벡터 b에 대한 **직교성분(rejection)**이라고 하며 각각 다음과 같이 표기한다.

$$a^{\parallel b} \tag{3.1.37}$$

$$a^{\perp b} \tag{3.1.38}$$

투영성분의 길이는 다음처럼 구할 수 있다.

$$\|a^{\parallel b}\| = \|a\| \cos\theta = \frac{\|a\|\|b\| \cos\theta}{\|b\|} = \frac{a^T b}{\|b\|} = \frac{b^T a}{\|b\|} = a^T \frac{b}{\|b\|} \tag{3.1.39}$$

만약 벡터 b 자체가 이미 단위벡터이면 **단위벡터에 대한 투영길이는 내적**이 된다.

$$\|a^{\parallel b}\| = a^T b \tag{3.1.40}$$

투영성분 성분 벡터는 투영성분 길이와 벡터 b 방향의 단위벡터의 곱이다.

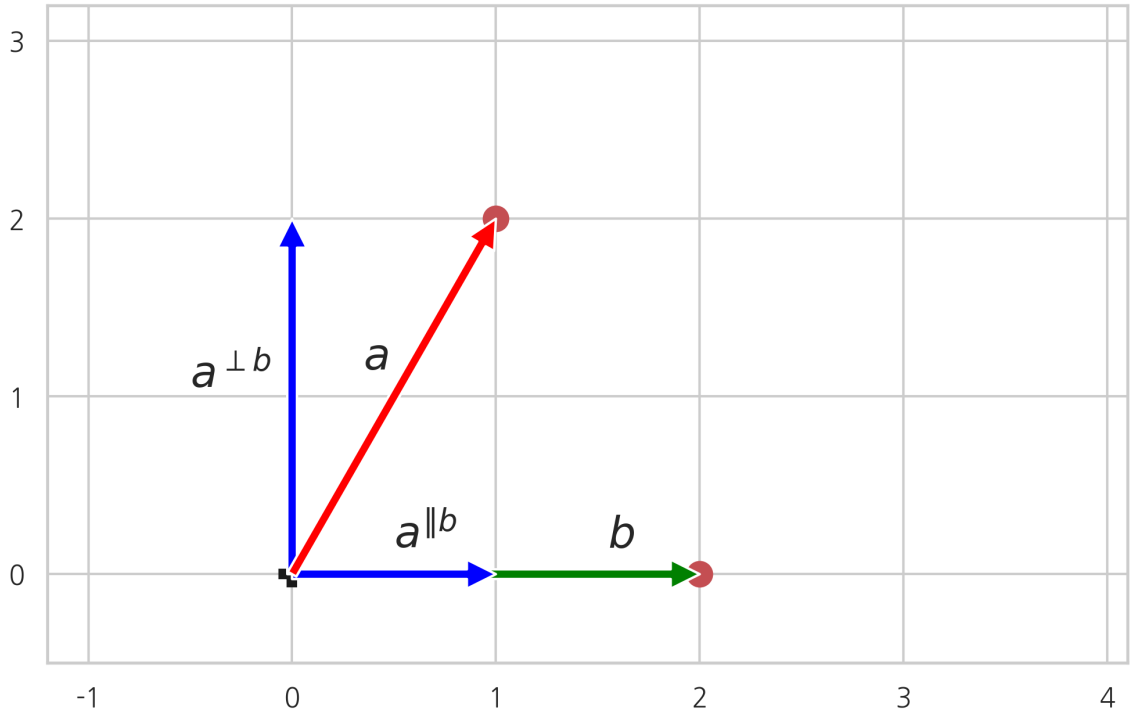
$$a^{\parallel b} = \frac{a^T b}{\|b\|} \frac{b}{\|b\|} = \frac{a^T b}{\|b\|^2} b \tag{3.1.41}$$

직교성분 벡터는 원래의 벡터에서 투영성분 성분 벡터를 뺀 나머지다.

$$a^{\perp b} = a - a^{\parallel b} \tag{3.1.42}$$

In [15]:

```
a = np.array([1, 2])
b = np.array([2, 0])
a2 = (a @ b) / np.linalg.norm(b) * np.array([1, 0])
a1 = a - a2
plt.annotate('', xy=b, xytext=(0, 0), arrowprops=green)
plt.annotate('', xy=a2, xytext=(0, 0), arrowprops=blue)
plt.annotate('', xy=a1, xytext=(0, 0), arrowprops=blue)
plt.annotate('', xy=a, xytext=(0, 0), arrowprops=red)
plt.plot(0, 0, 'kP', ms=10)
plt.plot(a[0], a[1], 'ro', ms=10)
plt.plot(b[0], b[1], 'ro', ms=10)
plt.text(0.35, 1.15, "$a$")
plt.text(1.55, 0.15, "$b$")
plt.text(-0.5, 1.05, "$a^{₩perp b}$")
plt.text(0.50, 0.15, "$a^{₩Vert b}$")
plt.xticks(np.arange(-10, 10))
plt.yticks(np.arange(-10, 10))
plt.xlim(-1.2, 4.1)
plt.ylim(-0.5, 3.2)
plt.show()
```



연습 문제 3.1.8

$$a = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad b = \begin{bmatrix} 2 \\ 0 \end{bmatrix} \tag{3.1.43}$$

일 때, 투영성분 $a^{\|b}$, 직교성분 $a^{\perp b}$를 구하라.

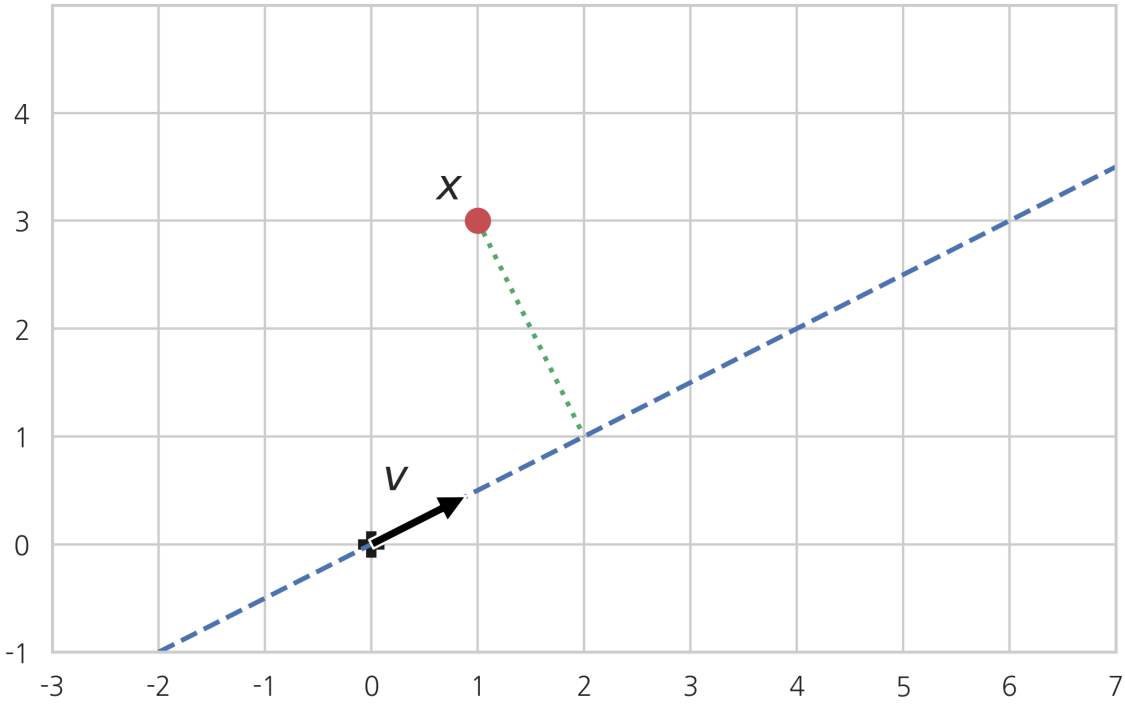
연습 문제 3.1.9

만약 v가 원점을 지나는 직선의 방향을 나타내는 단위벡터라고 하자. 이때 그 직선 위에 있지 않는 어떤 점 x와 그 직선과의 거리의 제곱이 다음과 같음을 증명하라.

$$\|x\|^2 - (x^T v)^2 \tag{3.1.44}$$

In [16]:

```
v = np.array([2, 1]) / np.sqrt(5)
x = np.array([1, 3])
plt.plot(0, 0, 'kP', ms=10)
plt.annotate('', xy=v, xytext=(0, 0), arrowprops=black)
plt.plot([-2, 8], [-1, 4], 'b--', lw=2)
plt.plot([1, 2], [3, 1], 'g:', lw=2)
plt.plot(x[0], x[1], 'ro', ms=10)
plt.text(0.1, 0.5, "$v$")
plt.text(0.6, 3.2, "$x$")
plt.xticks(np.arange(-3, 15))
plt.yticks(np.arange(-1, 5))
plt.xlim(-3, 7)
plt.ylim(-1, 5)
plt.show()
```



직선의 방정식

어떤 벡터 w가 있을 때

- 원점에서 출발한 벡터 w가 가리키는 점을 지나면서
- 벡터 w에 수직인

직선의 방정식을 구해보자.

위 두 조건을 만족하는 직선 상의 임의의 점을 가리키는 벡터를 x라고 하면, 벡터 x가 가리키는 점과 벡터 w가 가리키는 점을 이은 벡터 $x - w$는 조건에 따라 벡터 w와 직교해야 한다. 따라서 다음 식이 성립한다.

$$w^T(x - w) = 0 \tag{3.1.45}$$

정리하면 다음과 같아진다.

$$w^T(x - w) = w^T x - w^T w = w^T x - \|w\|^2 \tag{3.1.46}$$

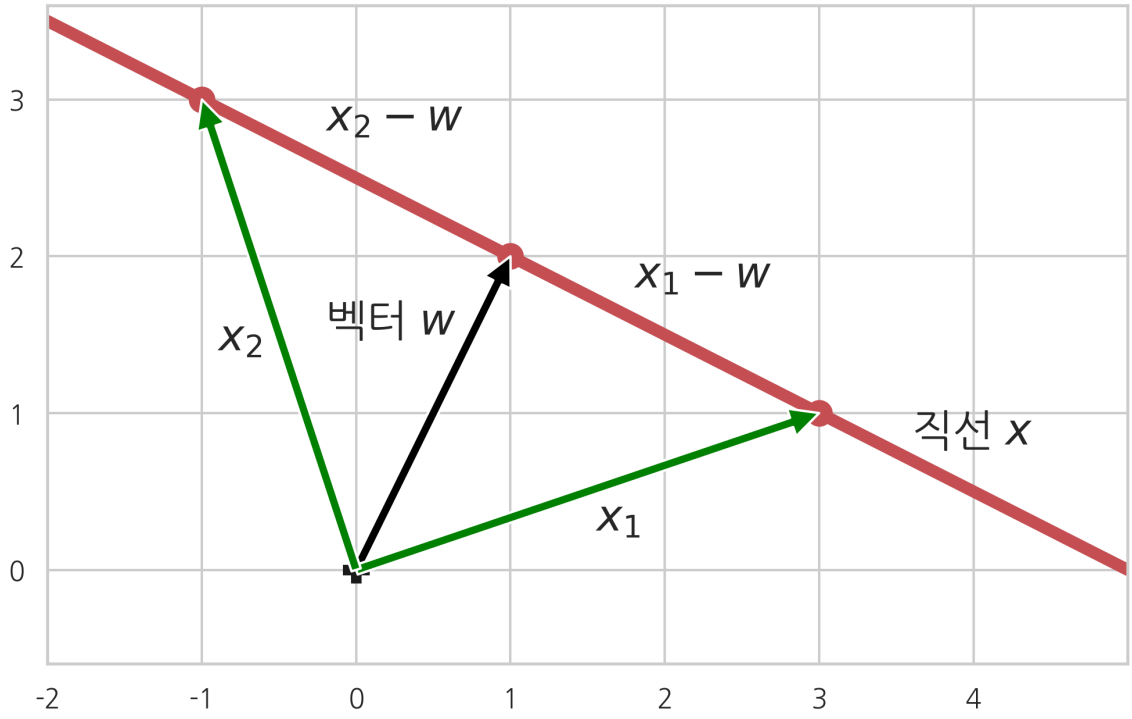
$$w^T x - \|w\|^2 = 0 \tag{3.1.47}$$

이 직선과 원점 사이의 거리는 벡터 w의 놈 $\|w\|$이다.

$$\|w\| \tag{3.1.48}$$

In [17]:

```
w = np.array([1, 2])
x1 = np.array([3, 1])
x2 = np.array([-1, 3])
plt.annotate('', xy=w, xytext=(0, 0), arrowprops=black)
plt.annotate('', xy=x1, xytext=(0, 0), arrowprops=green)
plt.annotate('', xy=x2, xytext=(0, 0), arrowprops=green)
plt.plot(0, 0, 'kP', ms=10)
plt.plot(w[0], w[1], 'ro', ms=10)
plt.plot(x1[0], x1[1], 'ro', ms=10)
plt.plot(x2[0], x2[1], 'ro', ms=10)
plt.plot([-3, 5], [4, 0], 'r-', lw=5)
plt.text(-0.2, 1.5, "벡터 $w$")
plt.text(1.55, 0.25, "$x_1$")
plt.text(-0.9, 1.40, "$x_2$")
plt.text(1.8, 1.8, "$x_1 - w$")
plt.text(-0.2, 2.8, "$x_2 - w$")
plt.text(3.6, 0.8, "직선 $x$")
plt.xticks(np.arange(-2, 5))
plt.yticks(np.arange(-1, 5))
plt.xlim(-2, 5)
plt.ylim(-0.6, 3.6)
plt.show()
```


3
2
1
0
-2
-1
0
1
2
3
4
$x_2 - w$
$x_1 - w$
벡터 w
x_2
x_1
직선 x

예를 들어

$$w = \begin{bmatrix} 1 \\ 2 \end{bmatrix} \tag{3.1.49}$$

일 때

$$\|w\|^2 = 5 \tag{3.1.50}$$

$$\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - 5 = x_1 + 2x_2 - 5 = 0 \tag{3.1.51}$$

$$x_1 + 2x_2 = 5 \tag{3.1.52}$$

이 방정식은 벡터 w가 가리키는 점 $(1, 2)$를 지나면서 벡터 w에 수직인 직선을 뜻한다. 이 직선과 원점 사이의 거리는 $\|w\| = \sqrt{5}$이다.

이번에는 벡터 w가 가리키는 점을 지나야 한다는 조건을 없애고 단순히

- 벡터 w에 수직인

직선 x의 방정식을 구해보자.

이때는 직선이 w가 아니라 w와 방향이 같고 길이가 다른 벡터 $w' = cw$을 지날 것이다. c는 양의 실수이다.

위에서 했던 방법으로 다시 직선의 방정식을 구하면 다음과 같다.

$$w'^T x - \|w'\|^2 = cw^T x - c^2\|w\|^2 = 0 \tag{3.1.53}$$

$$w^T x - c\|w\|^2 = 0 \tag{3.1.54}$$

여기에서 $c\|w\|^2$는 임의의 수가 될 수 있으므로 단순히 벡터 w에 수직인 직선의 방정식은 다음과 같이 나타낼 수 있다.

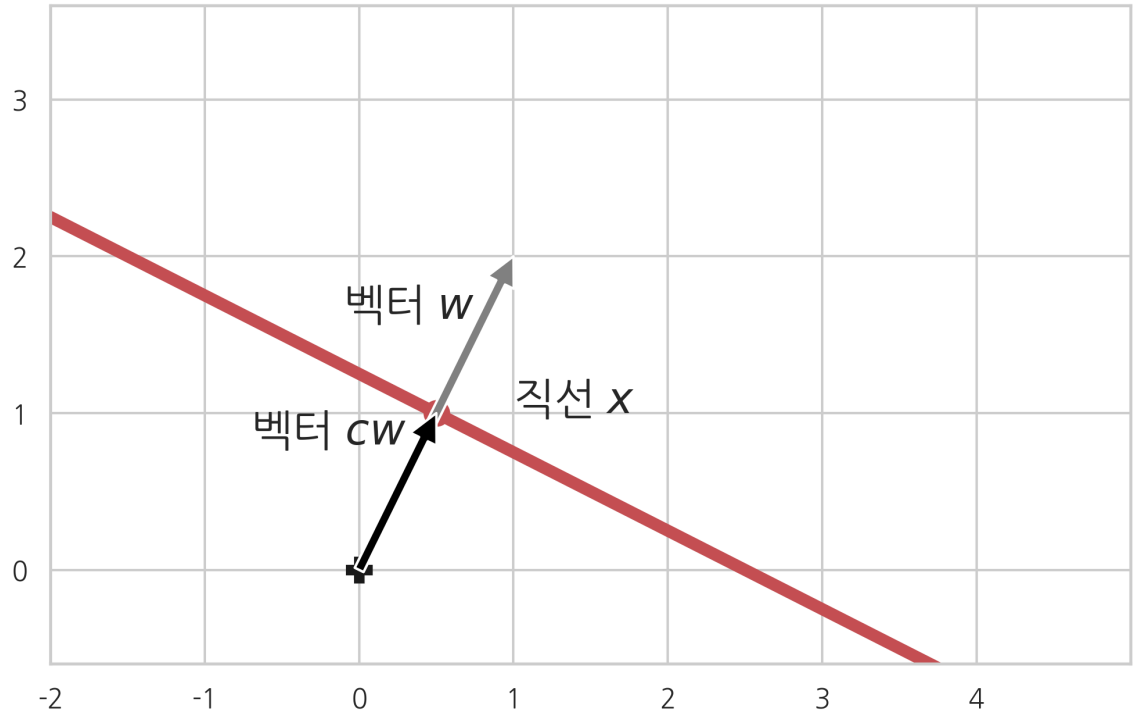
$$w^T x - w_0 = 0 \tag{3.1.55}$$

이 직선과 원점 사이의 거리는 다음과 같다.

$$c\|w\| = \frac{w_0}{\|w\|} \tag{3.1.56}$$

In [18]:

```
w = np.array([1, 2])
plt.annotate('', xy=w, xytext=(0, 0), arrowprops=gray)
plt.annotate('', xy=0.5 * w, xytext=(0, 0), arrowprops=black)
plt.plot(0, 0, 'kP', ms=10)
plt.plot(0.5 * w[0], 0.5 * w[1], 'ro', ms=10)
plt.plot([-2, 5], [2.25, -1.25], 'r-', lw=5)
plt.text(-0.7, 0.8, "벡터 $cw$")
plt.text(-0.1, 1.6, "벡터 $w$")
plt.text(1, 1, "직선 $x$")
plt.xticks(np.arange(-2, 5))
plt.yticks(np.arange(-1, 5))
plt.xlim(-2, 5)
plt.ylim(-0.6, 3.6)
plt.show()
```



예를 들어 $c = 0.5$이면 벡터 $w = [1, 2]^T$에 수직이고 원점으로부터의 거리가 $\frac{\sqrt{5}}{2}$인 직선이 된다.

$$x_1 + 2x_2 - 2.5 = 0 \tag{3.1.57}$$

연습 문제 3.1.10

직선 $w^T x - w_0 = 0$과 원점 사이의 거리가 다음과 같다는 것을 증명하라.

$$\frac{w_0}{\|w\|} \tag{3.1.58}$$

직선과 점의 거리

이번에는 직선 $w^T x - \|w\|^2 = 0$ 과 이 직선 위에 있지 않은 점 x' 사이의 거리를 구해보자.

벡터 w에 대한 벡터 x'의 투영성분 $x'^{\|w}$의 길이는 다음과 같다.

$$\|x'^{\|w}\| = \frac{w^T x'}{\|w\|} \tag{3.1.59}$$

직선과 점 x' 사이의 거리는 이 길이에서 원점에서 직선까지의 거리 $\|w\|$를 뺀 값의 절댓값이다.

$$\left| \|x'^{\|w}\| - \|w\| \right| = \left| \frac{w^T x'}{\|w\|} - \|w\| \right| = \frac{\left| w^T x' - \|w\|^2 \right|}{\|w\|} \tag{3.1.60}$$

직선의 방정식이 $w^T x - w_0 = 0$이면 직선과 점의 거리는 다음과 같다.

$$\frac{\left| w^T x' - w_0 \right|}{\|w\|} \tag{3.1.61}$$

이 공식은 나중에 분류 방법의 하나인 서포트 벡터 머신(SVM: Support Vector Machine)에서 사용된다.

연습 문제 3.1.11

직선의 방정식이 $w^T x - w_0 = 0$이면 직선과 점의 거리는 다음과 같다는 것을 증명하라.

$$\frac{\left| w^T x' - w_0 \right|}{\|w\|} \tag{3.1.62}$$

In [19]:

```
w = np.array([1, 2])
x1 = np.array([4, 3])
x2 = np.array([1, 2]) * 2
plt.annotate('', xy=x1, xytext=(0, 0), arrowprops=gray)
plt.annotate('', xy=x2, xytext=(0, 0), arrowprops=gray)
plt.annotate('', xy=w, xytext=(0, 0), arrowprops=red)
plt.plot(0, 0, 'kP', ms=10)
plt.plot(w[0], w[1], 'ro', ms=10)
plt.plot(x1[0], x1[1], 'ro', ms=10)
plt.plot([-3, 7], [4, -1], 'r-', lw=5)
plt.plot([2, 4], [4, 3], 'k:', lw=2)
plt.plot([3, 4], [1, 3], 'k:', lw=2)
plt.text(0.1, 0.9, "$w$")
plt.text(4.2, 3.1, "$x'$")
plt.text(1.5, 2.4, "$x'^{\Vert w}$")
plt.xticks(np.arange(-3, 15))
plt.yticks(np.arange(-1, 5))
plt.xlim(-3, 7)
plt.ylim(-1, 5)
plt.show()
```