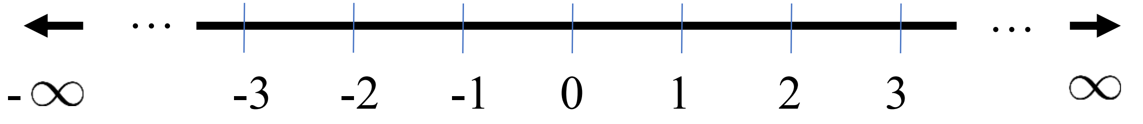
게임 수학 – 강의 9
복소수의 곱과 회전, 그리고 쿼터니언

동명대학교 게임공학과

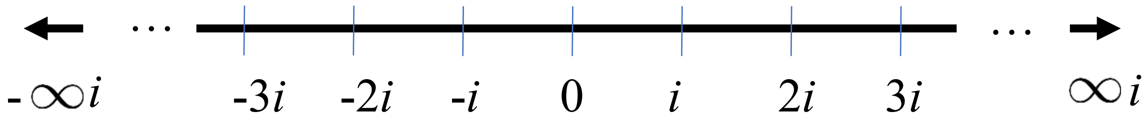
강영민

허수imaginary number

- 허수란?
  - 제곱해서 −1이 되는 수를 단위 i로 하는 수 체계
  - 실수 집합에서는 아무리 찾아도 제곱해서 음수가 되는 수는 찾을 수 없음
- 실수의 단위는 1, 허수의 단위는 i
- 실수는 수직선으로 표현 가능



- 허수도 수직선으로 표현 가능



복소수complex number

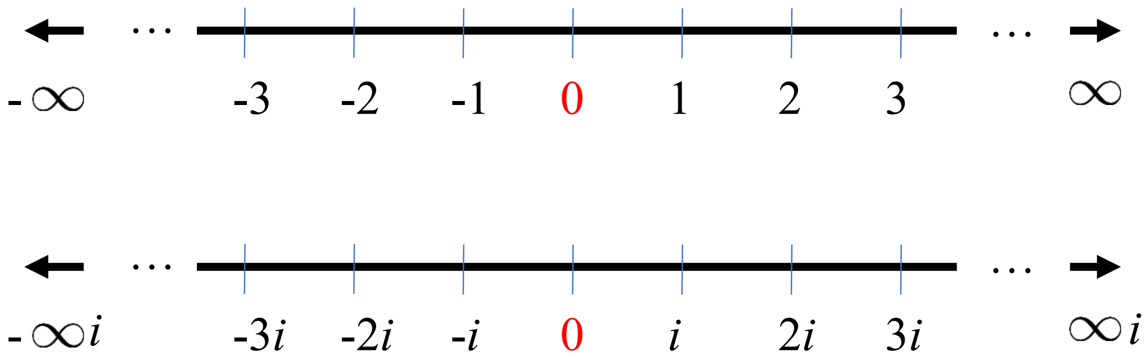
- 실수와 허수가 함께 만드는 수
  - 실수 + 허수

- 실수 + 실수 ➔ 실수 $\alpha + \beta = \gamma$
- 허수 + 허수 ➔ 허수 $\alpha i + \beta i = \gamma i$

- 실수와 허수의 덧셈 : $\alpha + \beta i = ?$
  - 실수 α와 허수의 크기 β는 서로 **연산이 되지 않음**
    - 더 줄일 수 없는 표현 : $\alpha + \beta i$ ← 복소수

복소수complex number는 어디에 있나?

- 실수와 허수 모두 수직선에 표시할 수 있음

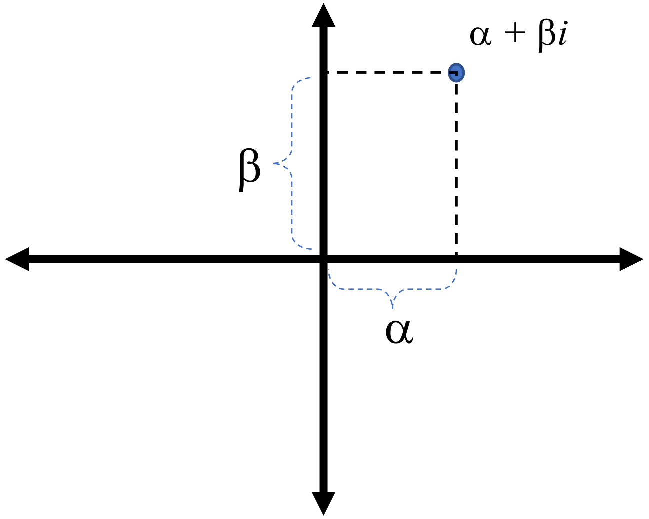


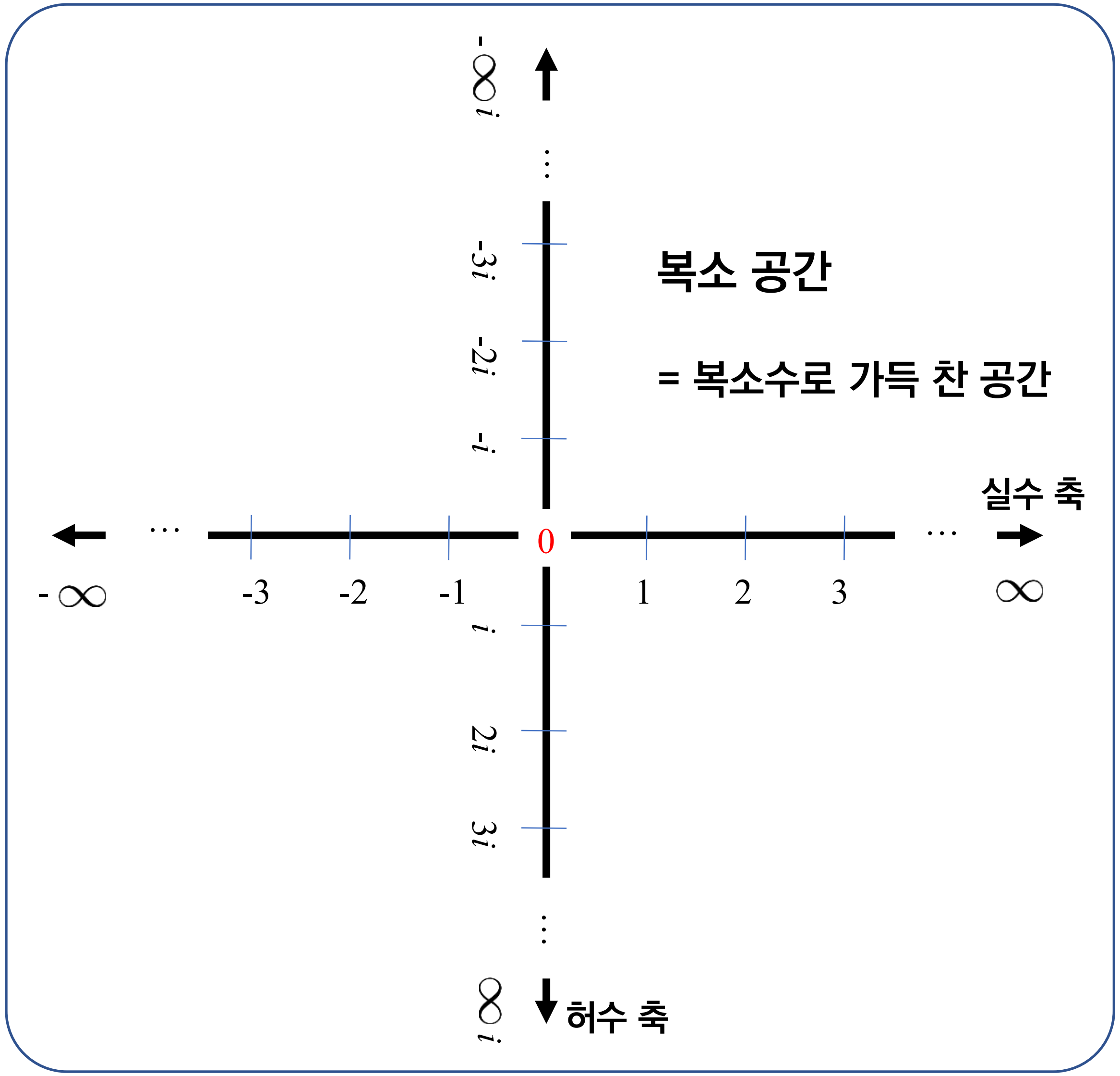
- 복소수는 어디에 있나?
  - 실수 수직선에도 없고, 허수 수직선에도 찍을 수 없음
  - 두 수직선의 밖에 있음
  - 두 수직선은 하나의 동일한 값을 가짐: 0

복소수complex number는 어디에 있나?

- 실수 수직선과 허수 수직선이 공유하는 값은 한 곳에
  - 실수는 가로축
  - 허수는 세로축
  - 실수와 허수도 아닌 어마어마하게 넓은 공간이 생성된다

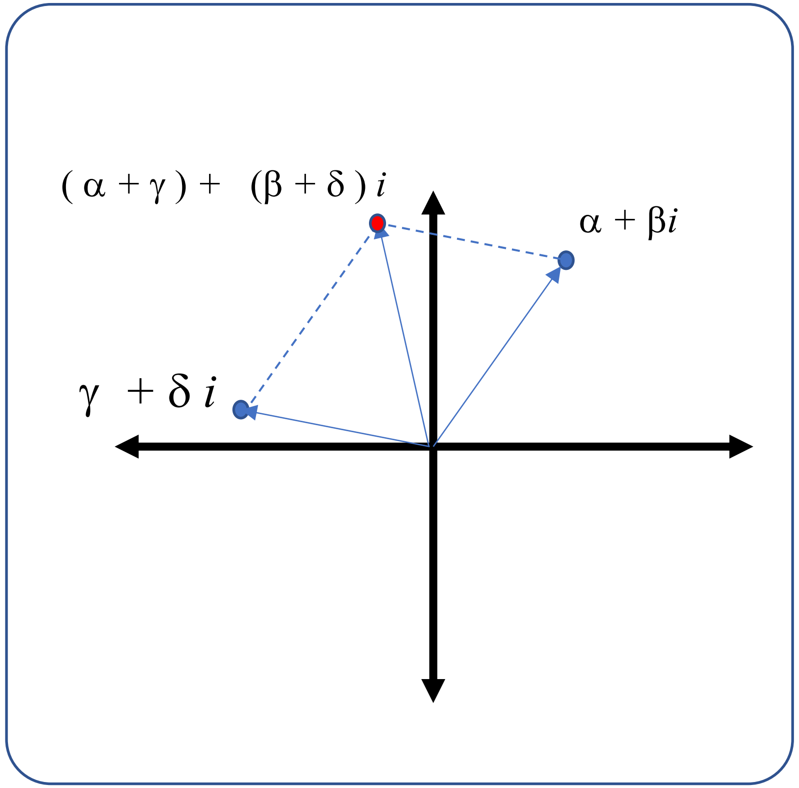
- (x, y) 좌표처럼 실수와 허수로 좌표 표현 가능
  - $(\alpha , \beta i)$
  - 이곳이 바로 $\alpha + \beta i$





복소수는 벡터와 비슷?

- 벡터처럼 덧셈과 뺄셈을 하면 각 축별로 이루어짐
  - 덧셈
    - 복소수 표현: $(\alpha + \beta i) + (\gamma + \delta i) = (\alpha + \gamma) + (\beta + \delta) i$
    - 벡터 표현 : $(\alpha, \beta) + (\gamma, \delta) = (\alpha + \gamma, \beta + \delta)$
  - 뺄셈
    - 복소수 표현: $(\alpha + \beta i) - (\gamma + \delta i) = (\alpha - \gamma) + (\beta - \delta) i$
    - 벡터 표현 : $(\alpha, \beta) - (\gamma, \delta) = (\alpha - \gamma, \beta - \delta)$



복소수의 곱셈은

- 복소수의 곱셈은?
  - 벡터는 스칼라 곱인 점곱과 벡터 곱인 가위곱이 존재
  - 복소수의 곱셈은
    - 실수와 허수는 곱셈이 가능 (허수를 실수배 하는 일)
    - 허수끼리 곱하면 음수
    - 두 복소수를 자연스럽게 곱할 수 있음
    - $(\alpha + \beta i)(\gamma + \delta i) = \alpha(\gamma + \delta i) + \beta i(\gamma + \delta i)$

$= \alpha\gamma + \alpha\delta i + \beta\gamma i + \beta\delta i^2$: 청색은 허수, 적색은 실수

$= \alpha\gamma - \beta\delta + (\alpha\delta + \beta\gamma) i$

복소수의 곱셈이 무슨 의미가 있을까?

- 복소수의 곱이 기하적으로 의미를 가질까?
  - 복소수 곱을 벡터의 변환으로 이해해 보자
- $(\alpha + \beta i)(\gamma + \delta i) = \alpha\gamma - \beta\delta + (\alpha\delta + \beta\gamma)i$
- $(\alpha + \beta i)(x + y i) = \alpha x - \beta y + (\alpha y + \beta x) i$
  $= x' + y' i$

$$\begin{aligned} x' &= \alpha x - \beta y \\ y' &= \beta x + \alpha y \end{aligned}$$

선형변환

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

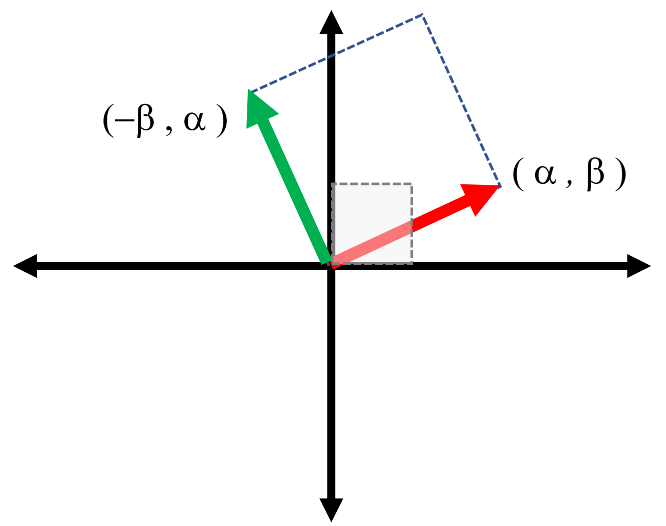
$(\alpha + \beta i)$를 $(x + y i)$에 곱한다는 것은
복소공간의 점 (x, y)에 행렬

$\begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix}$ 를 곱해서 변환하는 것과 같다

복소수의 곱셈은 어떤 변환을 하나

- 행렬을 그려보자

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \Leftrightarrow (\alpha + \beta i)(x + y i)$$

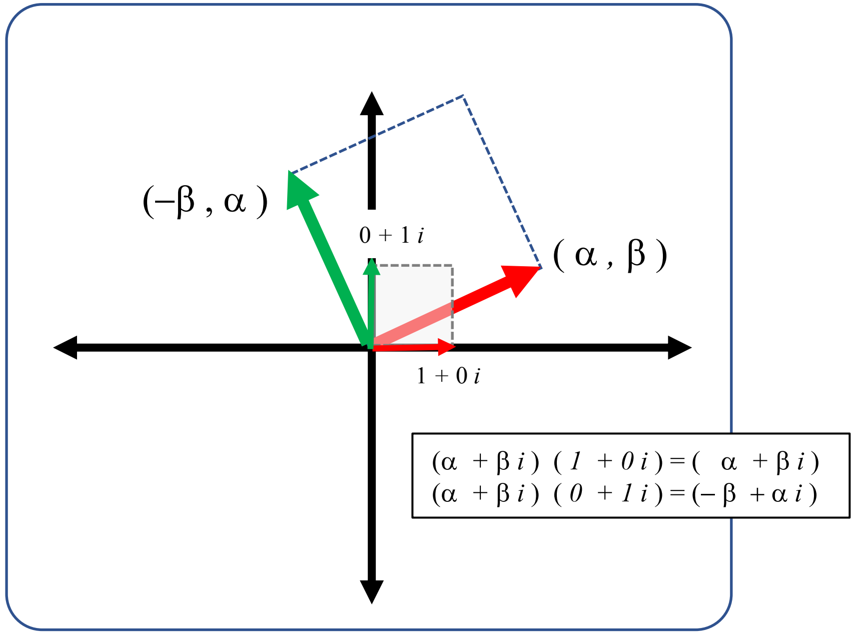


- 변환의 기하적 의미
  - 오른쪽 그림의 회색 사각형 = 축에 정렬된 단위 사각형
  - 이 단위 사각형이
    - 행렬의 1열 벡터를 표시하는 붉은 축과 2열 벡터인 녹색 축이 만드는 공간으로 옮겨짐
  - 이 붉은 색 축과 녹색 축은 언제나 직교 = 두 축의 내적은 언제나 0 $(-\alpha\beta + \alpha\beta)$
    - 공간이 크기만 커지거나 줄어들지 찌그러지지 않음
    - 축에 따라 크기가 바뀐 공간이 회전을 하게 됨

복소수의 곱셈은 어떤 변환을 하나

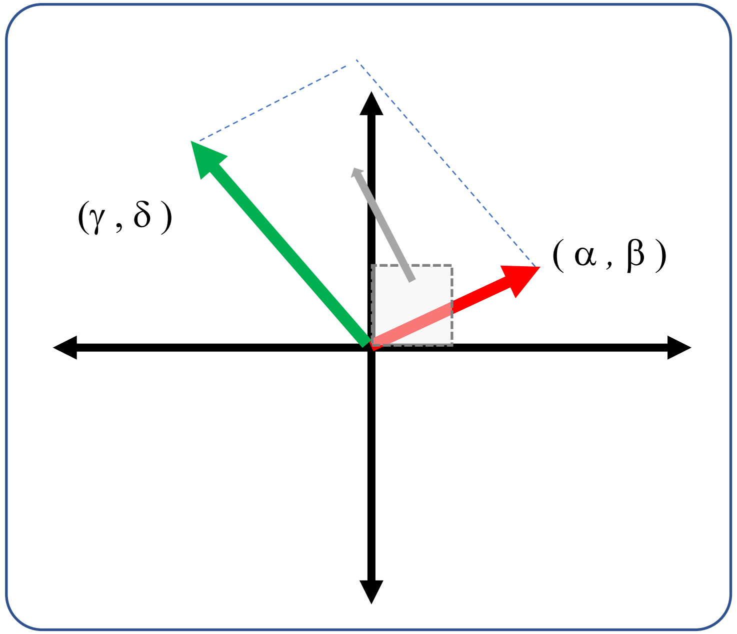
- 행렬을 그려보자

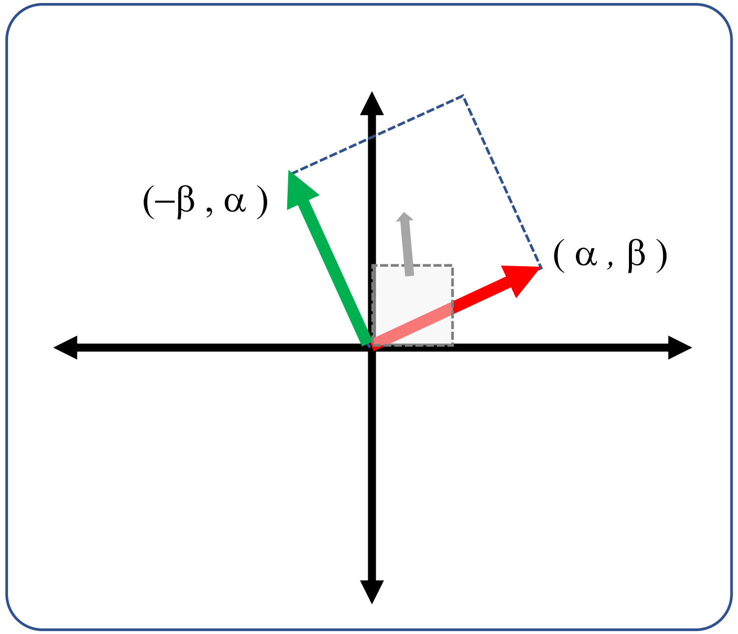
$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \alpha & -\beta \\ \beta & \alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \Leftrightarrow (\alpha + \beta i)(x + yi)$$



행렬곱셈과 복소수 곱셈의 비교

$$\begin{pmatrix} \alpha & \gamma \\ \beta & \delta \end{pmatrix} \times \qquad (\alpha + \beta i) \times$$





복소수를 곱하는 것은 복소평면 내의 변환이다.
행렬이 직교축을 임의의 축으로 옮겨 놓는다면,
복소수 곱하기는 회전과 스케일 둘만 적용되어
새로운 좌표축도 여전히 직교로 남게 변환된다.

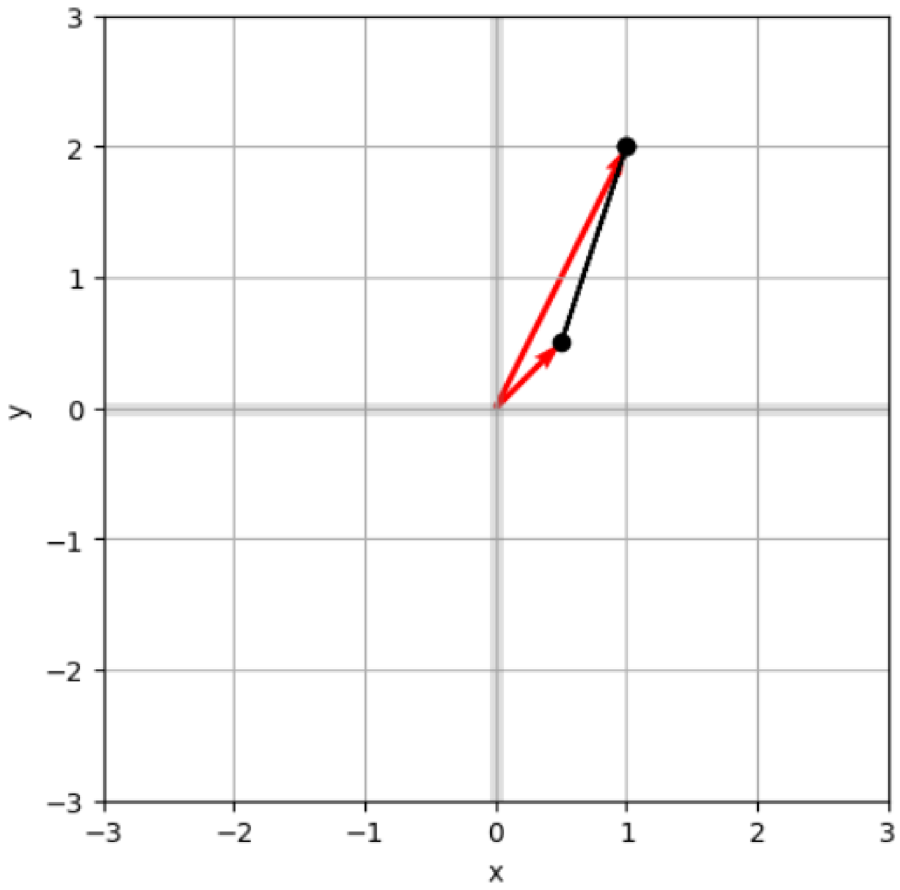
복소수 가시화

```
!wget https://raw.githubusercontent.com/dknife/linalg/main/tool/visualizer.py
from visualizer import *
!rm visualizer.py
```

```
def drawComplex(ax, c_list, color='black'):
    xList = []
    yList = []
    for c in c_list:
        xList.append(c.real)
        yList.append(c.imag)
    xList.append(c_list[0].real)
    yList.append(c_list[0].imag)
    plt.plot(xList, yList, marker="o", color=color)

def drawComplexAsVec(ax, c_list, color='black'):
    for c in c_list:
        v = np.array([c.real, c.imag])
        draw_vec2d(ax, v, color=color)
```

복소수 가시화

```
mySpace = axis2d(x=[-3, 3], y=[-3, 3])

a = 1 + 2j
b = 0.5 + 0.5j

drawComplex(mySpace, [a, b])
drawComplexAsVec(mySpace, [a, b], color='red')
```

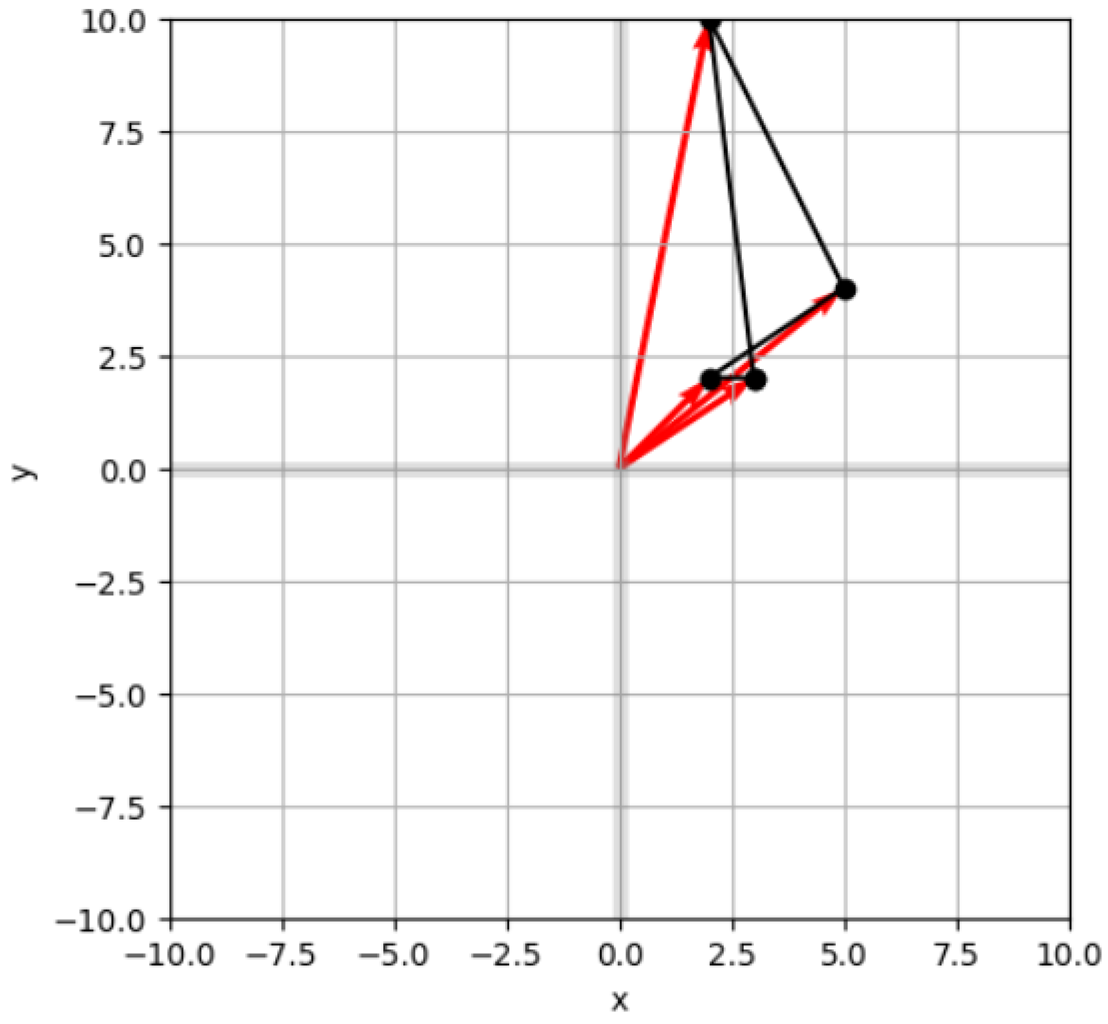



```
a = 3 + 2j
b = 2 + 2j

c = a + b
d = a * b

print(c, d)
```

```
(5+4j) (2+10j)
```

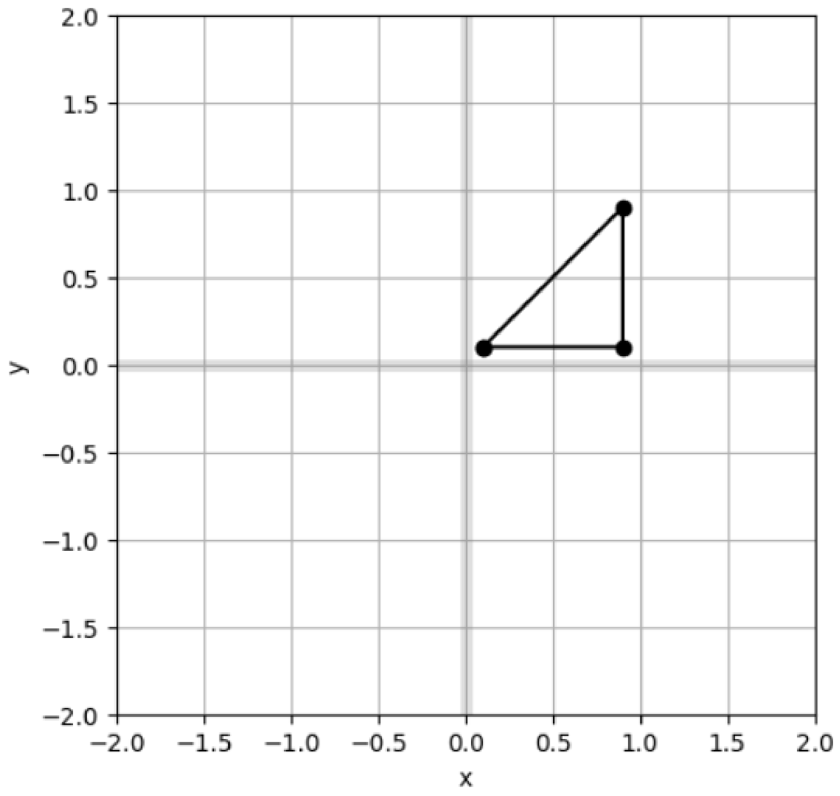
복소수 가시화

```
mySpace = axis2d(x=[-10, 10], y=[-10, 10])
drawComplex(mySpace, [a, b, c, d])
drawComplexAsVec(mySpace, [a, b, c, d], color='red')
```



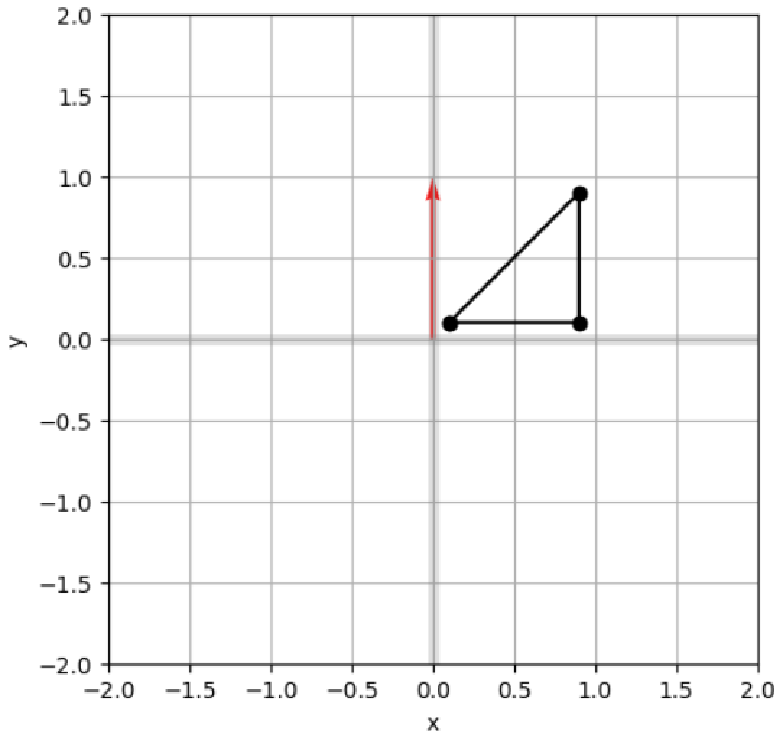
복소수로 표현하는 도형

```
# 삼각형을 하나 그려보자
a, b, c = 0.1+0.1j , 0.9+0.1j, 0.9+0.9j

mySpace = axis2d(x=[-2, 2], y=[-2, 2])
drawComplex(mySpace, [a, b, c])
```



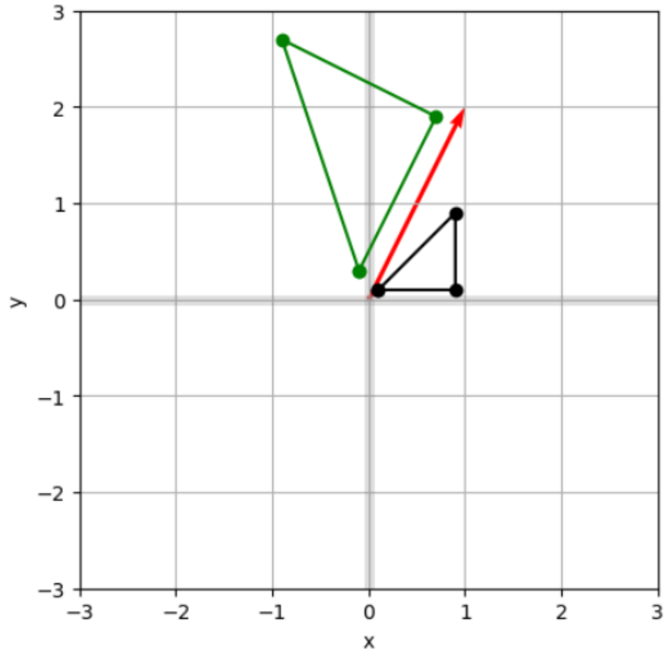
복소수로 표현하는 도형에 곱해질 복소수

```
a, b, c = 0.1+0.1j , 0.9+0.1j, 0.9+0.9j

rot_complex = 1j;

mySpace = axis2d(x=[-2, 2], y=[-2, 2])
drawComplex(mySpace, [a, b, c])
drawComplexAsVec(mySpace, [rot_complex], color='red')
```



복소수로 표현하는 도형에 곱해질 복소수

```
# 삼각형을 하나 그려보자
a, b, c = 0.1+0.1j , 0.9+0.1j, 0.9+0.9j


# 변환 복소수를 하나 정하자
rot_complex = 1+2j;

# 변환을 실행하자
A, B, C = rot_complex * a, rot_complex * b, rot_complex * c

# 모양을 그려보자
mySpace = axis2d(x=[-3, 3], y=[-3, 3])
drawComplex(mySpace, [a, b, c])
drawComplex(mySpace, [A, B, C], color='green')
drawComplexAsVec(mySpace, [rot_complex], color='red')
```



복소수 곱하기로 표현하는 회전 변환

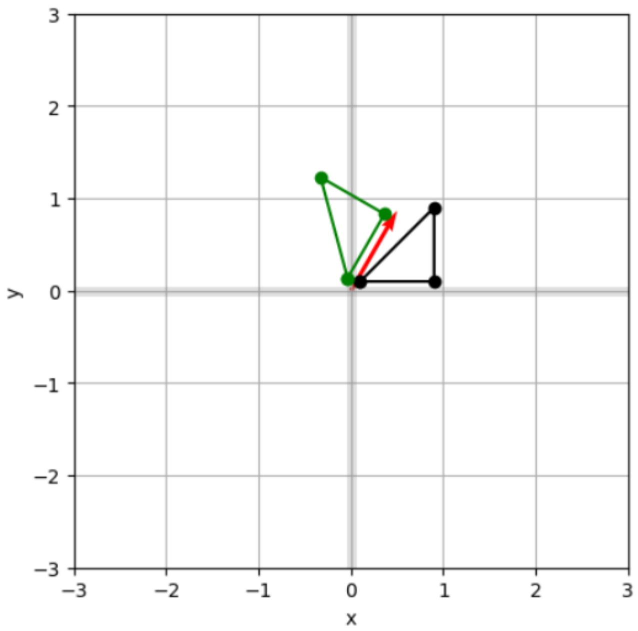
- 회전 행렬

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \Leftrightarrow (\cos\theta + \sin\theta\, i)(x + y\, i)$$

회전을 실시하자

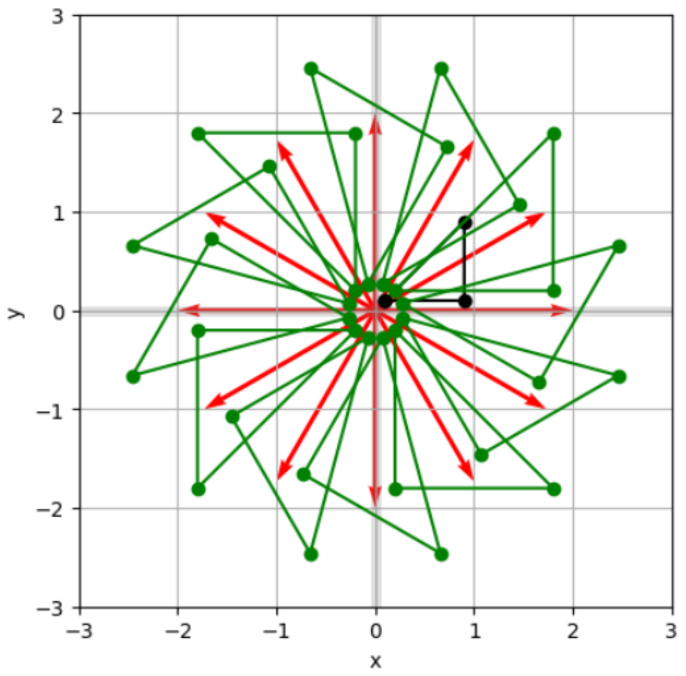
```python
# 삼각형을 하나 그려보자
a, b, c = 0.1+0.1j , 0.9+0.1j, 0.9+0.9j


# 회전 복소수를 만들자.
angle = 60
angle_radian = np.deg2rad(angle)
COS = np.cos(angle_radian)
SIN = np.sin(angle_radian)

rot_complex = complex(COS, SIN)
print(rot_complex)


# 변환을 실행하자
A, B, C = rot_complex * a, rot_complex * b, rot_complex * c

# 모양을 그려보자
mySpace = axis2d(x=[-3, 3], y=[-3, 3])
drawComplex(mySpace, [a, b, c])
drawComplex(mySpace, [A, B, C], color='green')
drawComplexAsVec(mySpace, [rot_complex], color='red')
```

```
(0.5000000000000001+0.8660254037844386j)
```



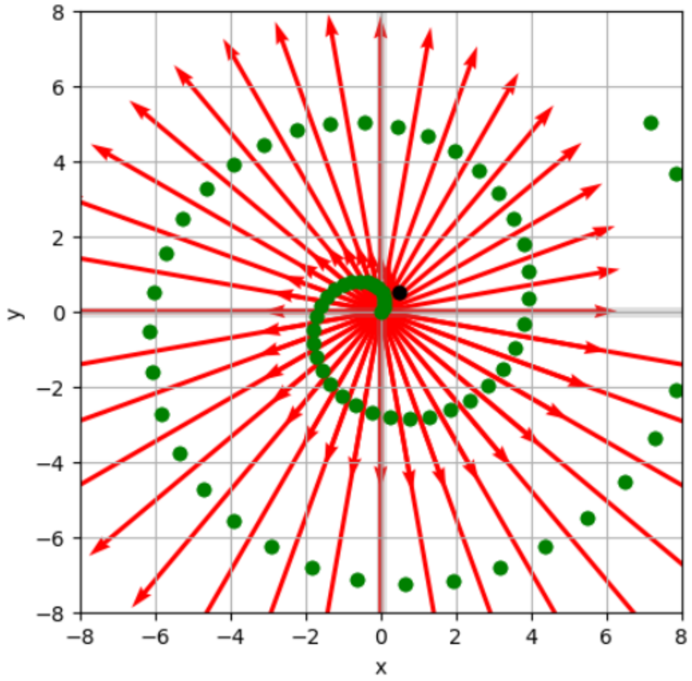
회전을 실시하자

```
# 삼각형을 하나 그려보자
a, b, c = 0.1+0.1j , 0.9+0.1j, 0.9+0.9j


# 모양을 그려보자
mySpace = axis2d(x=[-3, 3], y=[-3, 3])
drawComplex(mySpace, [a, b, c])

# 회전 복소수를 만들자.
for angle in range(0, 360, 30):
    angle_radian = np.deg2rad(angle)
    COS = np.cos(angle_radian)
    SIN = np.sin(angle_radian)

    # 변환을 실행하자
    rot_complex = 2 * complex(COS, SIN)
    A, B, C = rot_complex * a, rot_complex * b, rot_complex * c
    drawComplex(mySpace, [A, B, C], color='green')
    drawComplexAsVec(mySpace, [rot_complex], color='red')
```

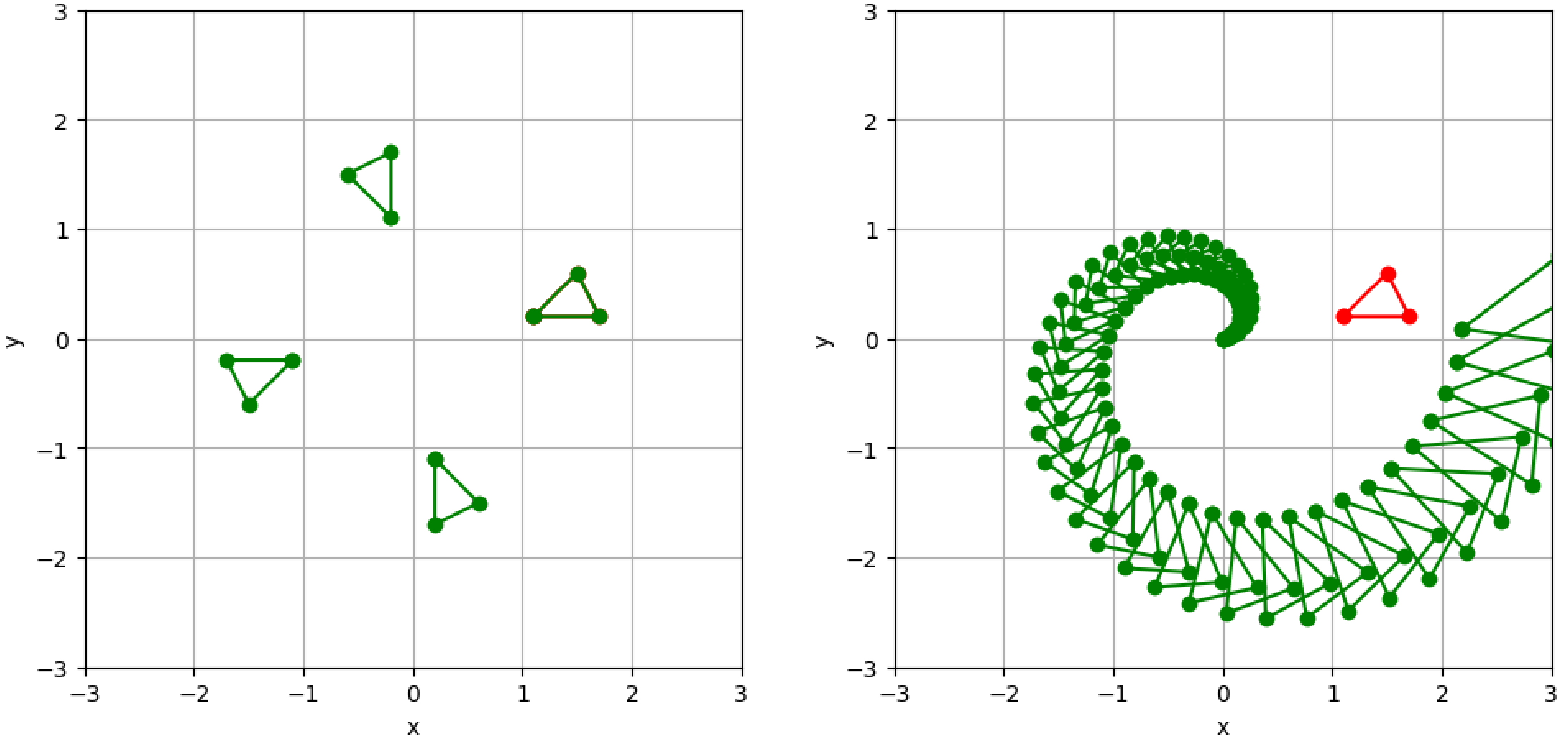


회전을 실시하자

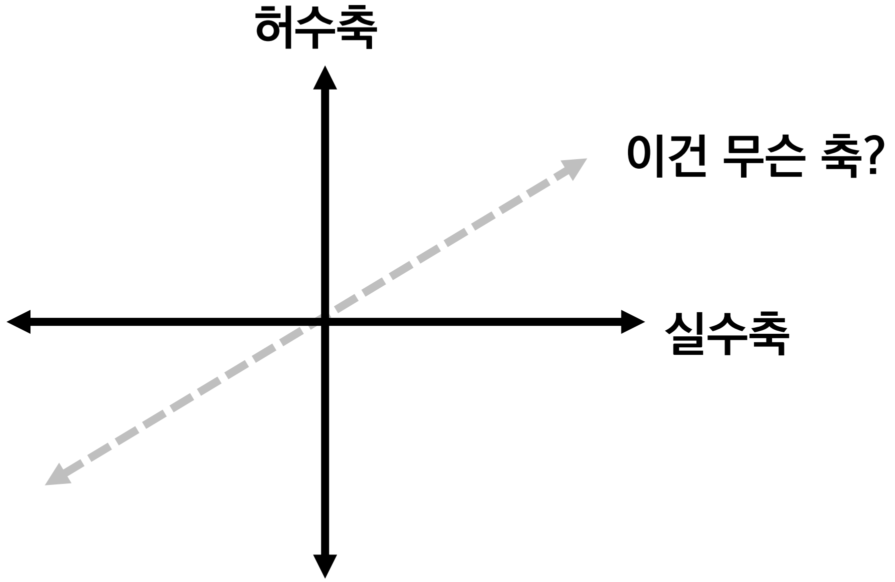
```python
# 점을 하나 정하자
a = 0.5+0.5j


# 모양을 그려보자
mySpace = axis2d(x=[-8, 8], y=[-8, 8])
drawComplex(mySpace, [a])

# 회전 복소수를 만들자.
for angle in range(0, 720, 10):
    angle_radian = np.deg2rad(angle)
    COS = np.cos(angle_radian)
    SIN = np.sin(angle_radian)

    # 변환을 실행하자
    rot_complex = angle_radian * complex(COS, SIN)
    A = rot_complex * a
    drawComplex(mySpace, [A], color='green')
    drawComplexAsVec(mySpace, [rot_complex], color='red')
```



복소수를 이용한 기하객체의 회전



2찡웏 곴갃엉섕늯〉



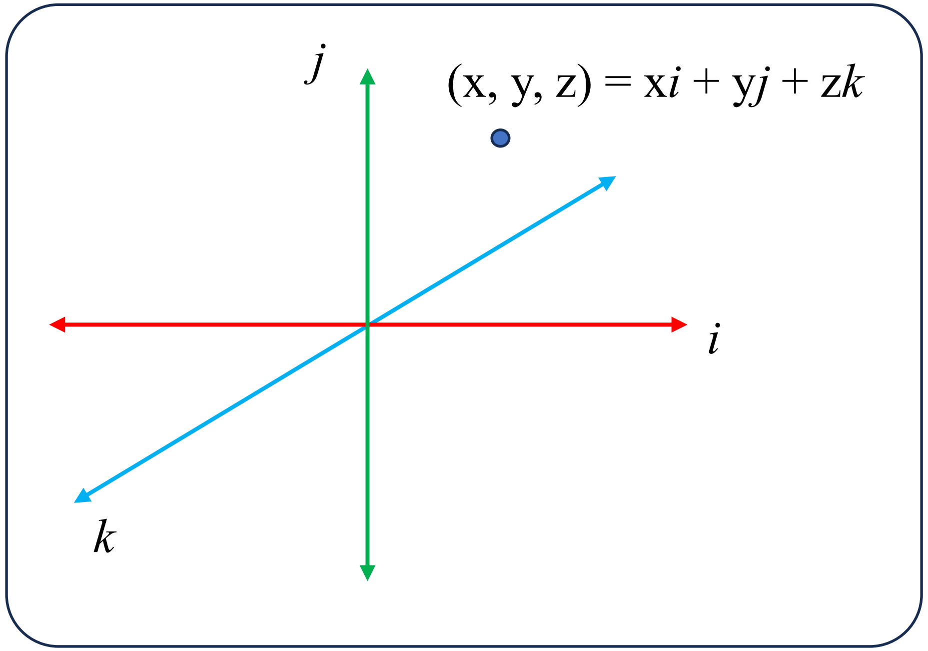
새로운 허수를 도입하자

- 2차원 복소 평면은
  - 두 개의 축 (실수, 허수)
    - 각각의 기저는 1과 i
    - 기저라는 것은 그 축의 모든 값이 이 기저를 늘이거나 줄인 값으로 표현할 수 있는 단위

  - 새로운 축이 필요하다면 i 말고 j라는 새로운 허수를 추가하면 어떨까?
- 모든 축을 허수로 다룬다면…
  - i, j, k 세 종류의 허수가 3차원 공간의 세 축의 기저가 되자

3차원 공간의 좌표

- 세 허수를 가진 복소수 표현
  - (x, y, z) = xi + yj + zk
- 실수는 사라진 것인가?
  - 3차원 공간 좌표를 표현하는 공간은 허수만 사용
  - 다른 용도로 실수축 사용
    - 벡터와 좌표의 구분 등
- 실수 + 허수 표현
  - 기저
    - (1, (i, j, k)) : 1은 실수부의 기저, i, j, k는 허수축 각각의 기저



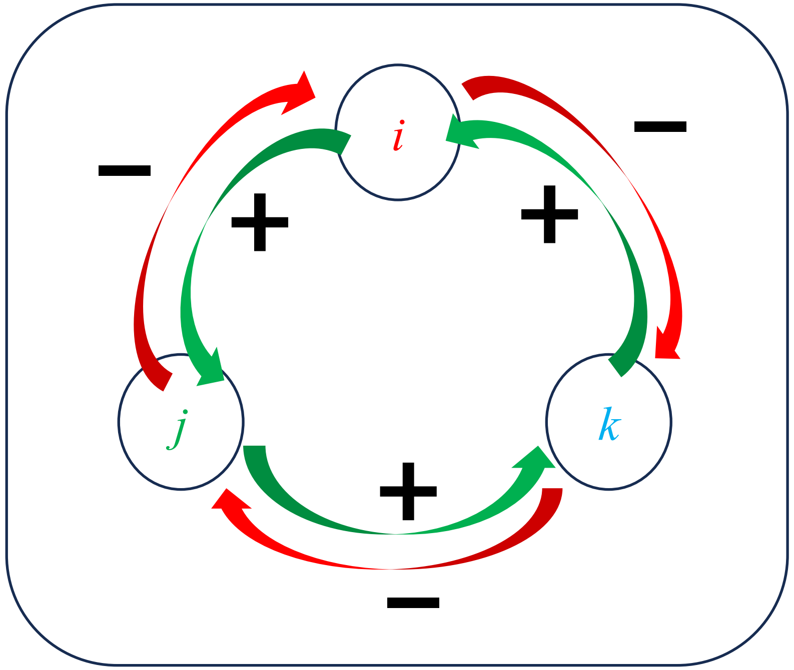
3차원 공간의 좌표를 곱할 수 있다?

- 복소수는 곱할 수 있다
  - 허수인 i, j, k의 제곱은 모두 −1
  - 실수와 허수의 곱은 허수 크기를 변경
  - 서로 다른 허수 사이의 곱은?
    - 다른 허수를 만든다

$ij = k$
$jk = i$
$ki = j$
$ji = -k$
$kj = -i$
$ik = -j$

$$(x_1, y_1, z_1)\ (x_2, y_2, z_2)$$
$$= (x_1 i + y_1 j + z_1 k)\ (x_2 i + y_2 j + z_2 k)$$

이렇게 곱하면
무엇을 얻을까?



3차원 공간의 좌표를 곱할 수 있다?

$\mathbf{u} = (x_1, y_1, z_1)$
$\mathbf{v} = (x_2, y_2, z_2)$

이렇게 곱하면
무엇을 얻을까?

$$(x_1, y_1, z_1)\ (x_2, y_2, z_2) = (x_1 i + y_1 j + z_1 k)\ (x_2 i + y_2 j + z_2 k)$$

$$= x_1 i\,(x_2 i + y_2 j + z_2 k) + y_1 j\,(x_2 i + y_2 j + z_2 k) + z_1 k\,(x_2 i + y_2 j + z_2 k)$$
$$= x_1 i\, x_2 i + x_1 i\, y_2 j + x_1 i\, z_2 k + y_1 j\, x_2 i + y_1 j\, y_2 j + y_1 j\, z_2 k + z_1 k\, x_2 i + z_1 k\, y_2 j + z_1 k\, z_2 k$$
$$= -x_1 x_2 + x_1 y_2 k - x_1 z_2 j - y_1 x_2 k - y_1 y_2 + y_1 z_2 i + z_1 x_2 j - z_1 y_2 i - z_1 z_2$$
$$= -x_1 x_2 - y_1 y_2 - z_1 z_2 + y_1 z_2 i - z_1 y_2 i + z_1 x_2 j - x_1 z_2 j + x_1 y_2 k - y_1 x_2 k$$
$$= -x_1 x_2 - y_1 y_2 - z_1 z_2 + (y_1 z_2 - z_1 y_2) I + (z_1 x_2 - x_1 z_2) j + (x_1 y_2 - y_1 x_2) k$$
$$= -\mathbf{u} \bullet \mathbf{v} + \mathbf{u} \times \mathbf{v}$$

스칼라곱 벡터곱

그런데 이 결과는 3개의 숫자로
표현된 원래의 좌표 방식으로 표현이 불가능
$\mathbf{u} \times \mathbf{v}$ 만 i, j, k 축 성분으로 표현 가능하고
$\mathbf{u} \bullet \mathbf{v}$ 는 실수축이 필요 ➔ 네 개의 기저 필요
$(1, (i, j, k))\ :\ (-\mathbf{u} \bullet \mathbf{v}, (\mathbf{u} \times \mathbf{v}))$

3차원 좌표를 곱하니 4개의 숫자가 필요

➔ 3차원 좌표는 4개의 숫자로 표현되는 수의 집합에서 실수 부분만 0인 부분집합

$\mathbf{u} = (x_1, y_1, z_1)$

$\mathbf{v} = (x_2, y_2, z_2)$

➔

$\mathbf{u} = (0, x_1, y_1, z_1) = 0 + x_1 i + y_1 j + z_1 k$

$\mathbf{v} = (0, x_2, y_2, z_2) = 0 + x_2 i + y_2 j + z_2 k$

사원수(四元數)quaternion

사원수의 표현 방법: $(s, \mathbf{v})$

$s \in \mathbb{R}$
$\mathbf{v} \in \mathbb{R}^3$

쿼터니언 연산: 덧셈과 뺄셈

$$\mathbf{p} + \mathbf{q} = (s_p, \mathbf{v}_p) + (s_q, \mathbf{v}_q) = (s_p + s_q , \mathbf{v}_p + \mathbf{v}_q)$$

$$\mathbf{p} = (s_p, \mathbf{v}_p)$$
$$\mathbf{q} = (s_q, \mathbf{v}_q)$$

4차원 벡터로 이해하면 됨

$$\mathbf{p} - \mathbf{q} = (s_p, \mathbf{v}_p) - (s_q, \mathbf{v}_q) = (s_p - s_q , \mathbf{v}_p - \mathbf{v}_q)$$

쿼터니언의 곱셈: 일반적 경우

$\mathbf{p} = (s_p, \mathbf{v}_p)$
$\mathbf{q} = (s_q, \mathbf{v}_q)$

$$
\begin{aligned}
\mathbf{p}\,\mathbf{q} = (s_p + \mathbf{v}_p)(s_q + \mathbf{v}_q) \\
&= s_p s_q + s_p \mathbf{v}_q + s_q \mathbf{v}_p + \mathbf{v}_p \mathbf{v}_q \\
&= s_p s_q + s_p \mathbf{v}_q + s_q \mathbf{v}_p + (-\mathbf{v}_p \bullet \mathbf{v}_q + \mathbf{v}_p \times \mathbf{v}_q) \\
&= s_p s_q - \mathbf{v}_p \bullet \mathbf{v}_q + s_p \mathbf{v}_q + s_q \mathbf{v}_p + \mathbf{v}_p \times \mathbf{v}_q
\end{aligned}
$$

$$
\begin{aligned}
\mathbf{p}\,\mathbf{q} = (s_p, \mathbf{v}_p)(s_q, \mathbf{v}_q) \\
&= (s_p s_q - \mathbf{v}_p \bullet \mathbf{v}_q,\ s_p \mathbf{v}_q + s_q \mathbf{v}_p + \mathbf{v}_p \times \mathbf{v}_q)
\end{aligned}
$$

쿼터니언의 곱셈: 스칼라 원소가 0이면

$$\mathbf{p} = (0, \mathbf{v}_p)$$
$$\mathbf{q} = (0, \mathbf{v}_q)$$

$$\begin{aligned}\mathbf{p}\,\mathbf{q} &= (0 + \mathbf{v}_p)\,(0 + \mathbf{v}_q)\\ &= \mathbf{v}_p\,\mathbf{v}_q\\ &= (-\mathbf{v}_p \bullet \mathbf{v}_q + \mathbf{v}_p \times \mathbf{v}_q)\\ &= -\mathbf{v}_p \bullet \mathbf{v}_q + \mathbf{v}_p \times \mathbf{v}_q\end{aligned}$$

$$\begin{aligned}\mathbf{p}\,\mathbf{q} &= (0\,,\, \mathbf{v}_p)\,(0\,,\, \mathbf{v}_q)\\ &= (-\mathbf{v}_p \bullet \mathbf{v}_q\,,\, \mathbf{v}_p \times \mathbf{v}_q)\end{aligned}$$

사원수의 연산규칙

$$\hat{p} + \hat{q} = \hat{q} + \hat{p}$$

$$(\hat{p} + \hat{q}) + \hat{r} = \hat{p} + (\hat{q} + \hat{r})$$

$$\lambda\hat{p} = \hat{p}\lambda$$

$$-\lambda\hat{p} = \lambda(-\hat{p})$$

$$\hat{p}\hat{q} \neq \hat{q}\hat{p}$$

$$\mathbf{p}\,\mathbf{q} = (s_p, \mathbf{v}_p)(s_q, \mathbf{v}_q) = (s_p s_q - \mathbf{v}_p \bullet \mathbf{v}_q,\ s_p\mathbf{v}_q + s_q\mathbf{v}_p + \mathbf{v}_p \times \mathbf{v}_q)$$

$$\mathbf{q}\,\mathbf{p} = (s_q, \mathbf{v}_q)(s_p, \mathbf{v}_p) = (s_p s_q - \mathbf{v}_p \bullet \mathbf{v}_q,\ s_p\mathbf{v}_q + s_q\mathbf{v}_p + \mathbf{v}_q \times \mathbf{v}_p)$$

켤레 사원수

- 켤레 사원수(공액 사원수, conjugate)
  - 어떤 사원수 $\hat{p} = (d_p, \mathbf{v}_p)$의 켤레 사원수를 $\hat{p}^*$라고 표현
  - 이 켤레 이 사원수는 $(d_p, -\mathbf{v}_p)$의 값을 가짐

$$\hat{p} = (d_p, \mathbf{v}_p) \Rightarrow \hat{p}^* = (d_p, -\mathbf{v}_p)$$

- 사원수의 크기는 벡터의 크기와 같은 방식으로 구한다.

$$\begin{aligned}
|\hat{q}| &= \sqrt{d_q d_q + a_q a_q + b_q b_q + c_q c_q} \\
&= \sqrt{d_q^2 + a_q^2 + b_q^2 + c_q^2} \\
&= \sqrt{d_q^2 + \mathbf{v}^{\mathrm{T}}\mathbf{v}} \\
&= \sqrt{d_q^2 + \mathbf{v} \cdot \mathbf{v}} \\
&= \sqrt{\hat{q}\hat{q}^*}
\end{aligned}$$

켤레 사원수

- 사원수의 항등원은 $\hat{i}$는 $(1, 0, 0, 0)$
- 사원수 $\hat{q}$의 역원 $\hat{q}^{-1}$은 $\hat{q}^*/|\hat{q}|$

$$\hat{q}\hat{i} = \hat{i}\hat{q} = \hat{q}$$
$$\hat{q}\hat{q}^{-1} = \hat{q}^{-1}\hat{q} = \hat{q}\hat{q}^*/|q| = \hat{i}$$

- 켤레 사원수의 크기는 서로 동일하다.

$$|\hat{q}| = |\hat{q}^*|$$

- 다음과 같은 연산 규칙도 중요

$$(\hat{q} + \hat{r})^* = \hat{q}^* + \hat{r}^*$$
$$(\hat{q}\hat{r})^* = \hat{r}^*\hat{q}^*$$

사원수 연산법칙 종합

$$\hat{p} + \hat{q} = \hat{q} + \hat{p}$$

$$(\hat{p} + \hat{q}) + \hat{r} = \hat{p} + (\hat{q} + \hat{r})$$

$$\lambda\hat{p} = \hat{p}\lambda$$

$$-\lambda\hat{p} = \lambda(-\hat{p})$$

$$\hat{p}\hat{q} \neq \hat{q}\hat{p}$$

$$\hat{p} = (d_p, \mathbf{v}_p) \implies \hat{p}^* = (d_p, -\mathbf{v}_p)$$

$$|\hat{q}| = \sqrt{\hat{q}\hat{q}^*}$$

$$\hat{q}\hat{i} = \hat{q} \implies \hat{i} = (1, 0, 0, 0)$$

$$\hat{q}\hat{p} = \hat{i} \implies \hat{p} = \hat{q}^{-1} = \hat{q}^*/|\hat{q}|$$

$$\hat{q}\hat{q}^{-1} = \hat{q}^{-1}\hat{q} = \hat{q}\hat{q}^*/|q| = \hat{i}$$

$$|\hat{q}| = |\hat{q}^*|$$

$$(\hat{q} + \hat{r})^* = \hat{q}^* + \hat{r}^*$$

$$(\hat{q}\hat{r})^* = \hat{r}^*\hat{q}^*$$

사원수 연산 구현

```
[2] quaternion1 = np.array([1, 2, 3, 4])
    quaternion2 = np.array([2, 1, 1, 2])
```

```
[3] def quat_add(q1, q2) :
        result = q1 + q2
        return result
```

```
[4] quat_add(quaternion1, quaternion2)
```

```
array([3, 3, 4, 6])
```

```
[5] def quat_mult(q1, q2) :
        s1, v1 = q1[0], q1[1:]
        s2, v2 = q2[0], q2[1:]
        scalar = s1 * s2 - np.dot(v1, v2)
        vector = s1*v2 + s2*v1 + np.cross(v1, v2)
        return np.array([scalar, vector[0], vector[1], vector[2]])
```

```
[7] quat_mult(quaternion1, quaternion2)
```

```
array([-11,   7,   7,   9])
```

```
def quat_norm(q) :
    return np.linalg.norm(q)

def quat_conjugate(q):
    s, v = q[0], q[1:]
    v = -v
    return np.array([s, v[0], v[1], v[2]])
```

```
quat_conjugate(quaternion1)
```

```
array([ 1, -2, -3, -4])
```

```
[ ] q1_norm = quat_norm(quaternion1)
    q1_conjugate_norm = quat_norm(quat_conjugate(quaternion1))

    print(q1_norm, q1_conjugate_norm)
```

```
5.477225575051661 5.477225575051661
```

사원수와 회전: 곱셈

- 행렬로 표현했던 회전은 사원수를 이용하여 표현 가능
- 어떤 좌표 $\mathbf{p}(x, y, z)$는 사원수 표현으로는 $\hat{p} = (0, (x, y, z)) = (0, \mathbf{p})$
- 이 좌표에 다음과 같은 사원수 $\hat{q}$를 곱하면 어떻게 되는지 보자.

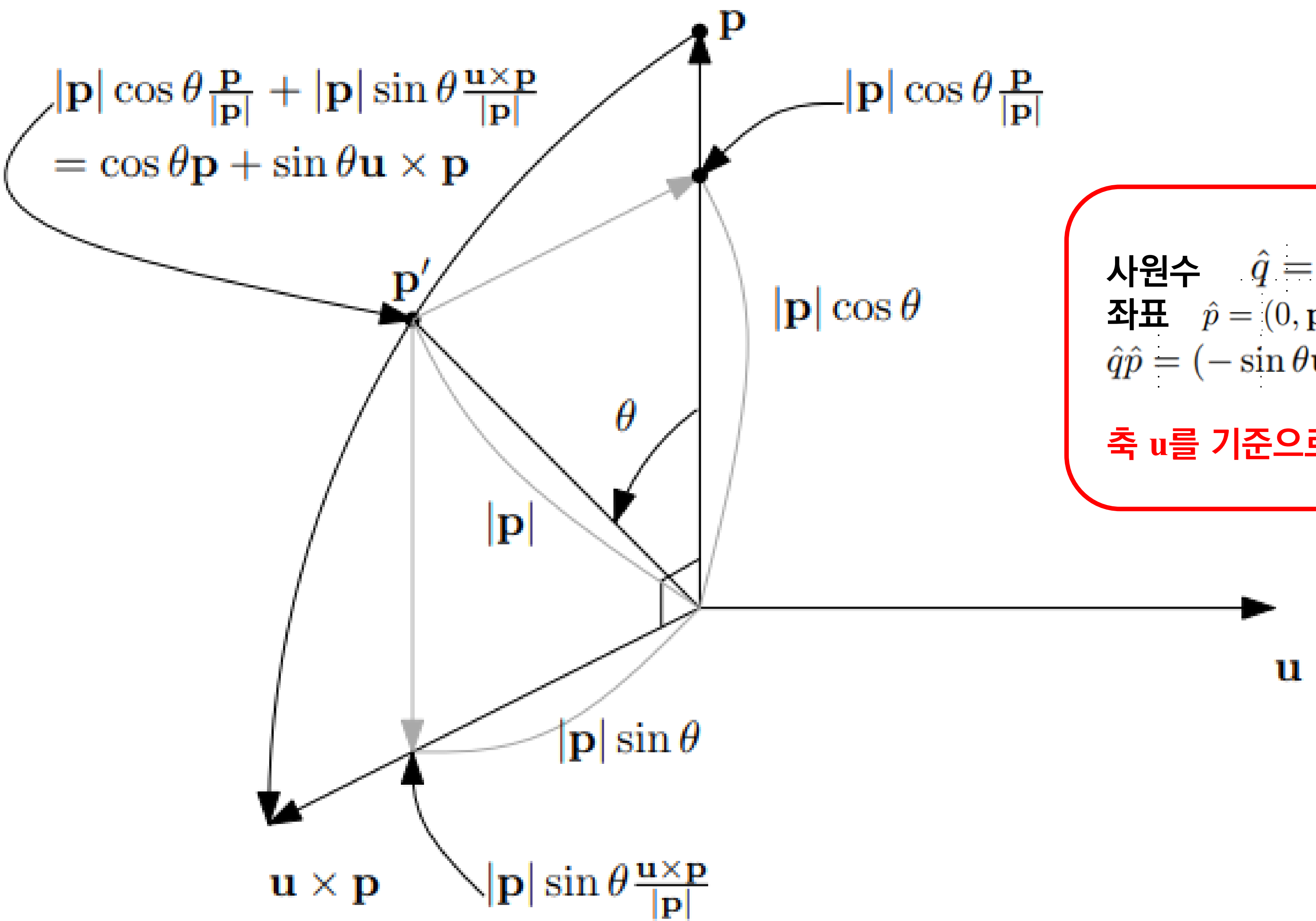
$$\hat{p} = (0, \mathbf{p})$$

$$\hat{q} = (\cos\theta, \sin\theta\mathbf{u}), |\mathbf{u}| = 1, |\hat{q}| = 1$$

- 벡터 $\mathbf{u}$는 단위벡터 ($\mathbf{u}$가 어떤 방향이나 축을 표현)

$$\hat{p}' = (d_{p'}, \mathbf{p}') = \hat{q}\hat{p} = (-\sin\theta\mathbf{u} \cdot \mathbf{p}, \cos\theta\mathbf{p} + \sin\theta\mathbf{u} \times \mathbf{p})$$

- **p**와 **u**가 서로 직교하는 경우



사원수 $\hat{q} = (\cos\theta, \sin\theta\mathbf{u})$ 를
좌표 $\hat{p} = (0, \mathbf{p})$ 에 곱해 얻는
$\hat{q}\hat{p} = (-\sin\theta\mathbf{u}\cdot\mathbf{p}, \cos\theta\mathbf{p} + \sin\theta\mathbf{u}\times\mathbf{p})$ 는

축 **u**를 기준으로 θ만큼 회전된 위치

$\mathbf{p}$와 $\mathbf{u}$가 서로 직교하지 않는 일반적 경우

- 스칼라 부분 $-\sin\theta\mathbf{u}\cdot\mathbf{p}$이 0이 아님
- 스칼라 값이 0이 될 수 있도록 사원수 곱하기를 두 번 수행
- 하나의 사원수 $\hat{q}$를 곱하는 것이 아니라 그 역원 $\hat{q}^{-1}$도 같이 곱함

$$\hat{p}' = \hat{q}\hat{p}\hat{p}^* = (\cos\theta, \sin\theta\mathbf{u})(0, \mathbf{p})(\cos\theta, -\sin\theta\mathbf{u})$$

$$\hat{q}\hat{p}\hat{p}^* = (-\sin\theta\mathbf{u}\cdot\mathbf{p}, \cos\theta\mathbf{p} + \sin\theta\mathbf{u}\times\mathbf{p})(\cos\theta, -\sin\theta\mathbf{u})$$

사원수 곱셈 연산법에 따라 계산하면 다음을 얻는다.

$$
\begin{aligned}
\hat{q}\hat{p}\hat{q}^* &= (s, \mathbf{v}) \\
s &= -\sin\theta\cos\theta\mathbf{u}\cdot\mathbf{p} + \sin\theta\cos\theta\mathbf{u}\cdot\mathbf{p} + \sin^2\theta(\mathbf{u}\times\mathbf{p})\cdot\mathbf{u} \\
\mathbf{v} &= \cos^2\theta\mathbf{p} \\
&\quad + \sin\theta\cos\theta\mathbf{u}\times\mathbf{p} \\
&\quad + (\sin^2\theta\mathbf{u}\cdot\mathbf{p})\mathbf{u} \\
&\quad - \sin\theta\cos\theta\mathbf{p}\times\mathbf{u} \\
&\quad - \sin^2\theta\mathbf{u}\times\mathbf{p}\times\mathbf{u}
\end{aligned}
$$

$\mathbf{u}\times\mathbf{p}$와 $\mathbf{u}$는 서로 수직이므로, 이 둘의 내적 $(\mathbf{u}\times\mathbf{p})\cdot\mathbf{u}$이 0이다. 따라서 스칼라 부분인 s가 0.

$$
\boxed{\hat{q}\hat{p}\hat{q}^* = (0, (\cos^2\theta - \sin^2\theta)\mathbf{p} + 2\sin\theta\cos\theta\mathbf{u}\times\mathbf{p} + (2\sin^2\theta\mathbf{u}\cdot\mathbf{p})\mathbf{u})}
$$

$$\begin{aligned} \cos 2\theta &= \cos^2\theta - \sin^2\theta \\ \sin 2\theta &= 2\sin\theta\cos\theta \end{aligned}$$

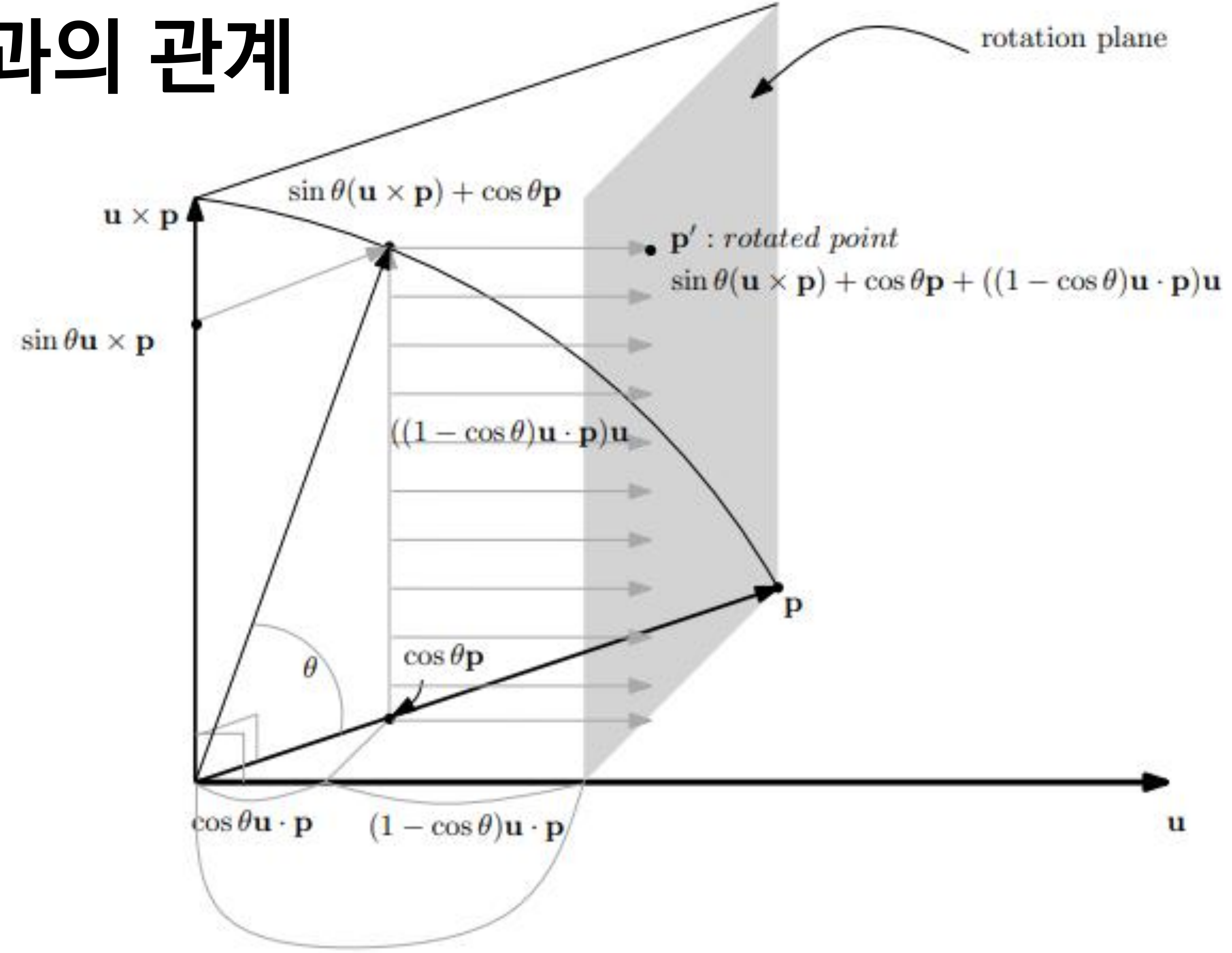
이 항등식을 적용하면 다음을 얻을 수 있다.

$$\hat{q}\hat{p}\hat{q}^* = (0, (\cos 2\theta\mathbf{p} + \sin 2\theta(\mathbf{u} \times \mathbf{p}) + (2\sin^2\theta\mathbf{u}\cdot\mathbf{p})\mathbf{u})$$

- $1 = \cos^2\theta + \sin^2\theta$이므로 $\sin^2\theta = 1 - \cos^2\theta$
- $2\sin^2\theta$는 $\sin^2\theta + \sin^2\theta = \sin^2\theta + 1 - \cos^2\theta$
- $1 - (\cos^2\theta - \sin^2\theta)$이므로 다음 성립
  - $2\sin^2\theta = 1 - \cos 2\theta$

$$\hat{q}\hat{p}\hat{q}^* = (0, (\cos 2\theta\mathbf{p} + \sin 2\theta(\mathbf{u} \times \mathbf{p}) + ((1 - \cos 2\theta)\mathbf{u}\cdot\mathbf{p})\mathbf{u})$$

회전과의 관계


rotation plane
$\sin\theta(\mathbf{u}\times\mathbf{p})+\cos\theta\mathbf{p}$
$\mathbf{u}\times\mathbf{p}$
$\mathbf{p}'$: *rotated point*
$\sin\theta(\mathbf{u}\times\mathbf{p})+\cos\theta\mathbf{p}+((1-\cos\theta)\mathbf{u}\cdot\mathbf{p})\mathbf{u}$
$\sin\theta\mathbf{u}\times\mathbf{p}$
$((1-\cos\theta)\mathbf{u}\cdot\mathbf{p})\mathbf{u}$
$\mathbf{p}$
$\cos\theta\mathbf{p}$
θ
$\cos\theta\mathbf{u}\cdot\mathbf{p}$
$(1-\cos\theta)\mathbf{u}\cdot\mathbf{p}$
$\mathbf{u}$

쿼터니언을 이용한 회전

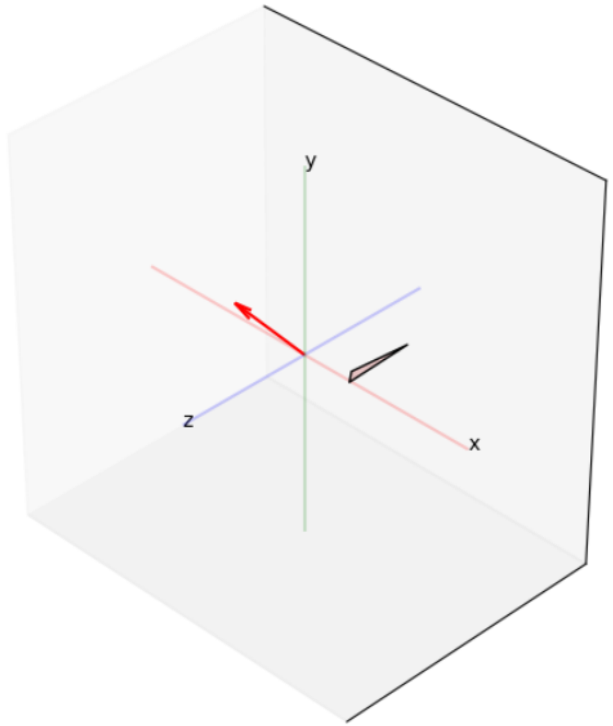
$$\mathbf{p}' = \sin\theta(\mathbf{u} \times \mathbf{p}) + \cos\theta\mathbf{p} + ((1 - \cos\theta)\mathbf{u} \cdot \mathbf{p})\mathbf{u}$$

어떤 점 p를 u 축을 중심으로 θ 만큼 회전하여 p′를 얻고 싶을 때

- $\hat{p} = (0, \mathbf{p})$
- $\hat{q} = (\cos\frac{\theta}{2}, \sin\frac{\theta}{2}\mathbf{u})$
- $\hat{p}' = (0, \mathbf{p}') = \hat{q}\hat{p}\hat{q}^*$

```python
import numpy as np
import math
```

```python
# Define the vertices of the triangle
p0 = np.array([2.2, 1.1, 1.5])
p1 = np.array([3.2, 2.2, 1.0])
p2 = np.array([2.1, 1.2, 1.3])
triangle = np.array([p0, p1, p2])
color = np.array([1, 0, 0]) * 0.7

u = np.array([-1, 1, 1])

my_axis = axis3d(x=[-4, 4], y=[-4,4], z=[-4,4])
draw_polygons(my_axis, [triangle], facecolors=[color], edgecolors='black', alpha=0.2)
draw_vec3d(my_axis, u)
setCam(my_axis, [1, 1, 1])
```

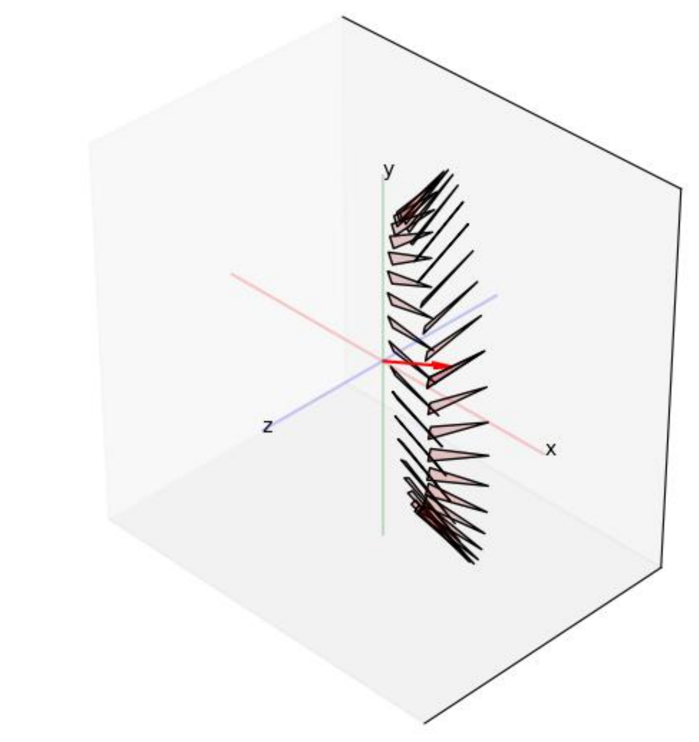


```python
def quat_mult(q1, q2) :
    s1, v1 = q1[0], q1[1:]
    s2, v2 = q2[0], q2[1:]
    scalar = s1 * s2 - np.dot(v1, v2)
    vector = s1*v2 + s2*v1 + np.cross(v1, v2)
    return np.array([scalar, vector[0], vector[1], vector[2]])
```

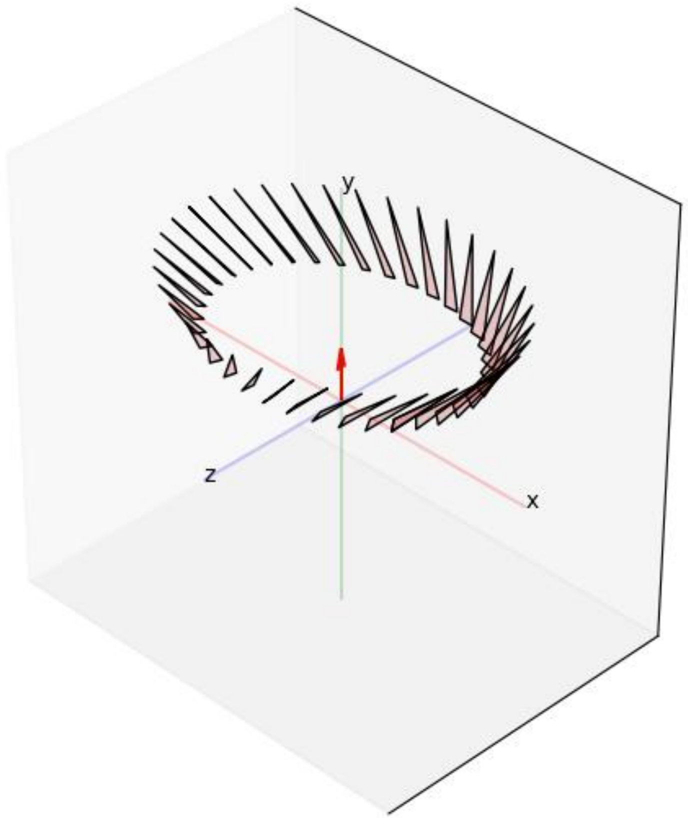
```python
def quat_conjugate(q):
    s, v = q[0], q[1:]
    v = -v
    return np.array([s, v[0], v[1], v[2]])
```

```python
def make_rotation_quaternion(axis, angle_degree):
    # q = (cos(angle/2), sin(angle/2) u)
    # rotation
    # q p q* ==> p'
    angle = np.deg2rad(angle_degree)
    u = axis / np.linalg.norm(axis)
    C = math.cos(angle/2)
    S = math.sin(angle/2)
    return np.array([C, S*u[0], S*u[1], S*u[2]])
```

```python
# Define the vertices of the triangle
p0 = np.array([2.2, 1.1, 1.5])
p1 = np.array([3.2, 2.2, 1.0])
p2 = np.array([2.1, 1.2, 1.3])
triangle = np.array([p0, p1, p2])
color = np.array([1, 0, 0]) * 0.7

u = np.array([1, 0, -1])



my_axis = axis3d(x=[-4, 4], y=[-4,4], z=[-4,4])
draw_polygons(my_axis, [triangle], facecolors=[color], edgecolors='black', alpha=0.2)

for angle in range(0,360,10):
    rot = make_rotation_quaternion(u, angle)
    rot_conj = quat_conjugate(rot)

    q0 = np.array([0, p0[0], p0[1], p0[2]])
    q1 = np.array([0, p1[0], p1[1], p1[2]])
    q2 = np.array([0, p2[0], p2[1], p2[2]])

    q0 = quat_mult( quat_mult(rot, q0), rot_conj)
    q1 = quat_mult( quat_mult(rot, q1), rot_conj)
    q2 = quat_mult( quat_mult(rot, q2), rot_conj)

    new_triangle = np.array([q0[1:], q1[1:], q2[1:]])

    draw_polygons(my_axis, [new_triangle], facecolors=[color], edgecolors='black', alpha=0.2)

draw_vec3d(my_axis, u)
setCam(my_axis, [1, 1, 1])
```



```python
# Define the vertices of the triangle
p0 = np.array([2.2, 1.1, 1.5])
p1 = np.array([3.2, 2.2, 1.0])
p2 = np.array([2.1, 1.2, 1.3])
triangle = np.array([p0, p1, p2])
color = np.array([1, 0, 0]) * 0.7

u = np.array([0, 1, 0])




my_axis = axis3d(x=[-4, 4], y=[-4,4], z=[-4,4])
draw_polygons(my_axis, [triangle], facecolors=[color], edgecolors='black', alpha=0.2)

for angle in range(0,360,10):
    rot = make_rotation_quaternion(u, angle)
    rot_conj = quat_conjugate(rot)

    q0 = np.array([0, p0[0], p0[1], p0[2]])
    q1 = np.array([0, p1[0], p1[1], p1[2]])
    q2 = np.array([0, p2[0], p2[1], p2[2]])

    q0 = quat_mult(quat_mult(rot, q0), rot_conj)
    q1 = quat_mult(quat_mult(rot, q1), rot_conj)
    q2 = quat_mult(quat_mult(rot, q2), rot_conj)

    new_triangle = np.array([q0[1:], q1[1:], q2[1:]])

    draw_polygons(my_axis, [new_triangle], facecolors=[color], edgecolors='black', alpha=0.2)

draw_vec3d(my_axis, u)
setCam(my_axis, [1, 1, 1])
```



```
# Define the vertices of the triangle
p0 = np.array([2.0, 0.0, 2.0])
p1 = np.array([2.5, 0.0, 1.0])
p2 = np.array([2.25, 0.0, 0.0])
triangle = np.array([p0, p1, p2])
color = np.array([1, 0, 0]) * 0.7

u = np.array([0, 1, 0])



my_axis = axis3d(x=[-4, 4], y=[-4,4], z=[-4,4])
draw_polygons(my_axis, [triangle], facecolors=[color], edgecolors='black', alpha=0.2)

rot = make_rotation_quaternion(u, 90)
rot_conj = quat_conjugate(rot)

q0 = np.array([0, p0[0], p0[1], p0[2]])
q1 = np.array([0, p1[0], p1[1], p1[2]])
q2 = np.array([0, p2[0], p2[1], p2[2]])

q0 = quat_mult(quat_mult(rot_conj, q0), rot)
q1 = quat_mult(quat_mult(rot_conj, q1), rot)
q2 = quat_mult(quat_mult(rot_conj, q2), rot)

new_triangle = np.array([q0[1:], q1[1:], q2[1:]])

draw_polygons(my_axis, [new_triangle], facecolors=[color], edgecolors='black', alpha=0.2)

draw_vec3d(my_axis, u)
setCam(my_axis, [1, 1, 1])
```