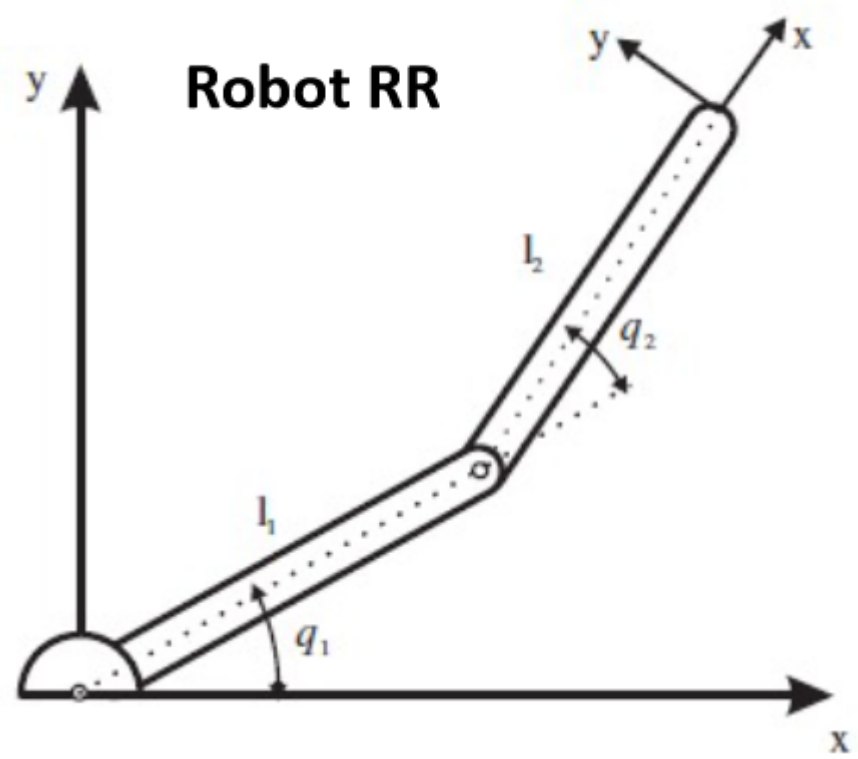
Práctica 2

Javier Gálvez Obispo

3 de mayo de 2021

Ejercicio 1

En este ejercicio nos piden implementar una función que resuelva el problema cinemático directo para un manipulador **RR**. Dados los ángulos de giro de las articulaciones, q_1 y q_2, junto con las longitudes de los eslabones, l_1 y l_2, tenemos que calcular la posición del extremo del robot (x, y).



Utilizamos el método geometrico para obtener los valores de x e y en función de los parámetros dados:

$$x = l_1 cos(q_1) + l_2 cos(q_1 + q_2)$$

$$y = l_1 sen(q_1) + l_2 sen(q_1 + q_2)$$

Entonces, nuestra función **pcd** simplemente tiene que calcular el punto (x, y) tal y como se indica arriba y devolver el resultado obtenido.

```
def pcd(q1, q2, l1, l2):
    x = l1*np.cos(q1) + l2*np.cos(q1+q2)
    y = l1*np.sin(q1) + l2*np.sin(q1+q2)
    return(x, y)
```

Ejercicio 2

Utilizamos ahora la función recién definida **pcd** para dibujar la trayectoria seguida por el extremo del robot. Dadas dos listas, *q1s* y *q2s* con los ángulos de giro (en radianes) que toman la primera y la segunda articulación respectivamente a lo largo de la trayectoria, la función calcula y muestra todas las posiciones por las que pasa el extremo del robot.

Además, nos piden que centremos el origen de coordenadas de la figura en función de las longitudes de los eslabones, estableciendo una longitud máxima para cada eje de $\pm(l_1 + l_2 + 1)$.

```
def dibujar_trayectoria_pcd(q1s, q2s, l1, l2):
    # Calculamos todos los puntos y los pintamos
    puntos = [pcd(q1, q2, l1, l2) for q1, q2 in zip(q1s, q2s)]
    plt.plot(*zip(*puntos))

    # Centrar el origen de coordenadas
    rango = np.arange(-l1-l2-1, l1+l2+2)
    plt.xticks(rango)
    plt.yticks(rango)

    plt.show()
```

Ejercicio 3

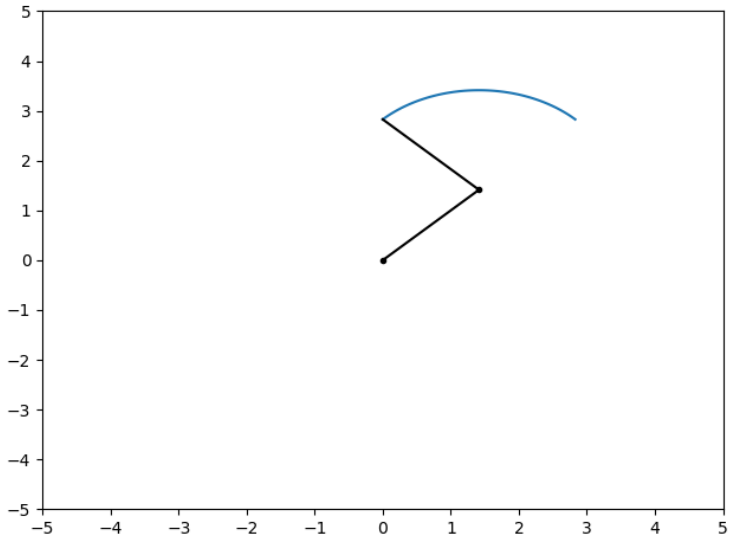
Para facilitar el análisis de la trayectoria seguida, implementamos ahora una función que dibuje los eslabones del robot RR. Esta función será llamada por la función **dibujar_trayectoria_pcd** anterior para que se muestre tanto la trayectoria seguida como el robot en la última posición en la que se encuentra.

Utilizamos el siguiente código ya dado en la práctica para pintar el robot:

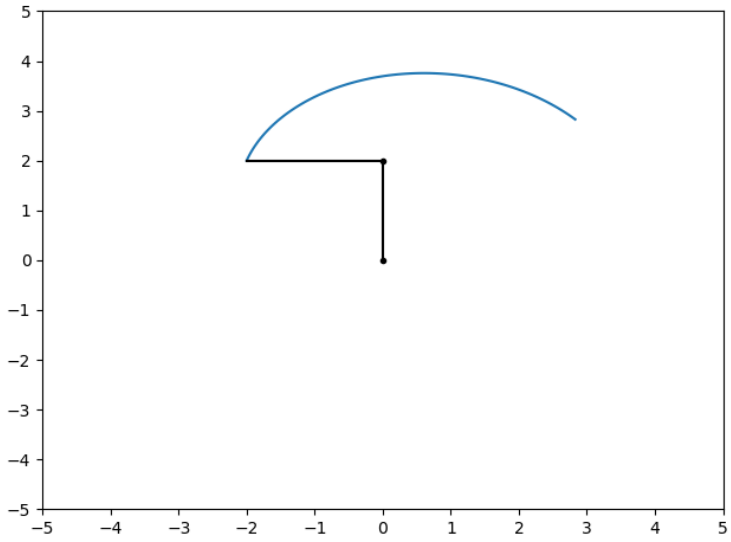
```
def dibujar_robot(q1, q2, l1, l2):
    x0, y0 = 0, 0                       # Posición de la articulación 1
    x1, y1 = pcd(q1, 0, l1, 0)          # Posición de la articulación 2
    x2, y2 = pcd(q1, q2, l1, l2)        # Posición del extremo del robot

    x, y = [x0, x1, x2], [y0, y1, y2]   # Coordenadas de la trayectoria
    plt.plot(x, y, 'k')                 # Traza la trayectoria
    plt.plot(x0, y0, 'k.')              # Dibuja la articulación 1
    plt.plot(x1, y1, 'k.')              # Dibuja la articulación 2
```

Ejercicio 4

Para comprobar el correcto funcionamiento de todas las funciones anteriores mostramos varios ejemplos trazando distintas trayectorias. Primero trazamos una trayectoria con 100 puntos en la que q_1 siempre vale $\pi/4$ y q_2 toma valores entre 0 y $\pi/2$.



Y otro ejemplo en el que q_1 toma valores entre $\pi/4$ y $\pi/2$ y q_2 entre 0 y $\pi/2$



En ambos ejemplos las longitudes de los eslabones son $l_1 = l_2 = 2$.

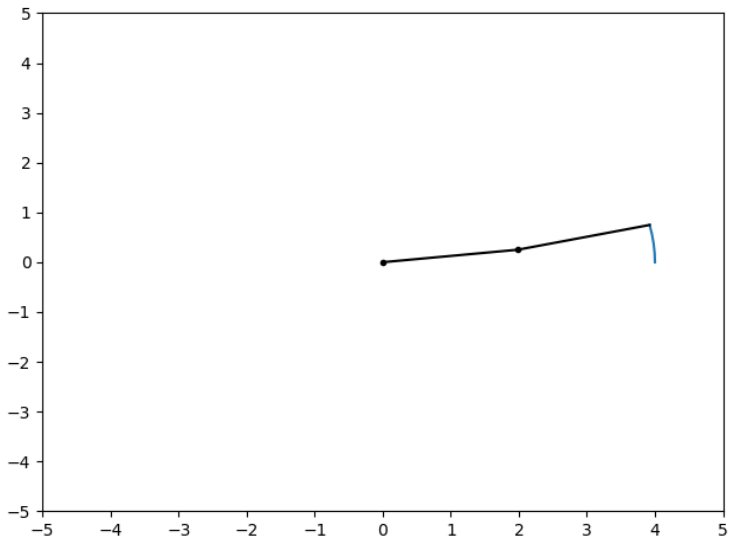
Ejercicio 5

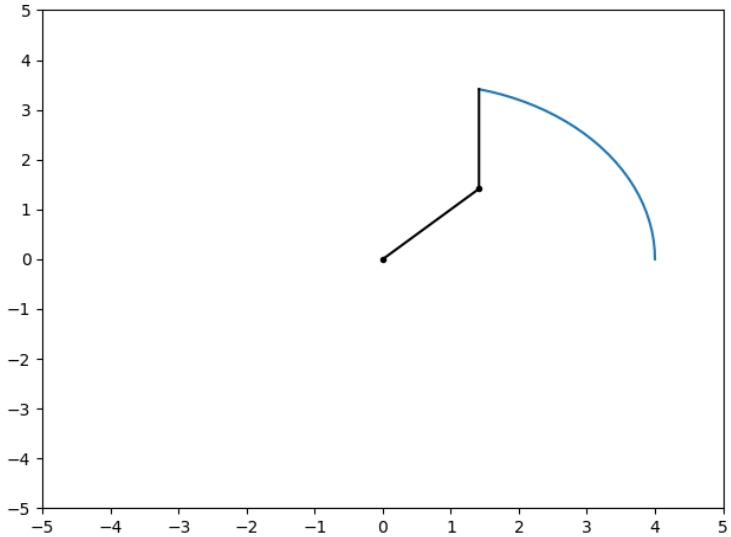
Implementamos ahora una función que muestre una animación paso a paso del movimiento del robot siguiendo la trayectoria indicada.
Nos dan el siguiente código:

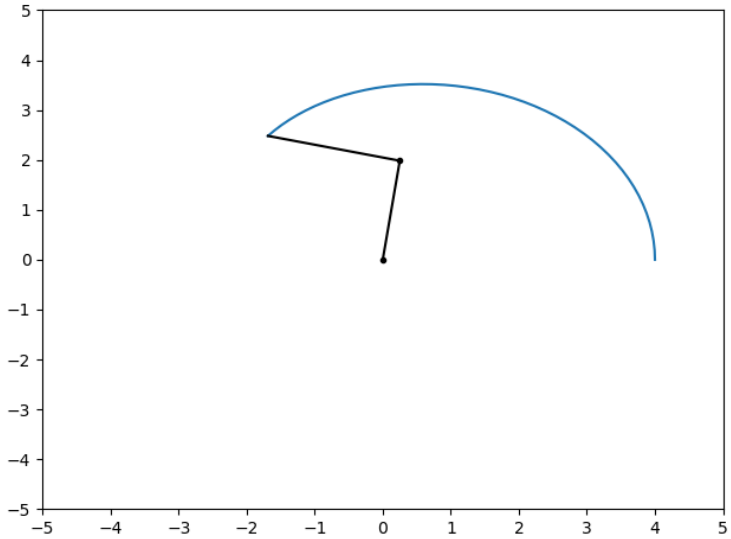
```
def animacion_trayectoria_pcd(q1s, q2s, l1, l2):
    n = min(len(q1s), len(q2s))
    for i in range(1, n):
        plt.clf()
        dibujar_trayectoria_pcd(q1s[0:i], q2s[0:i], l1, l2)
        plt.pause(0.001)
```

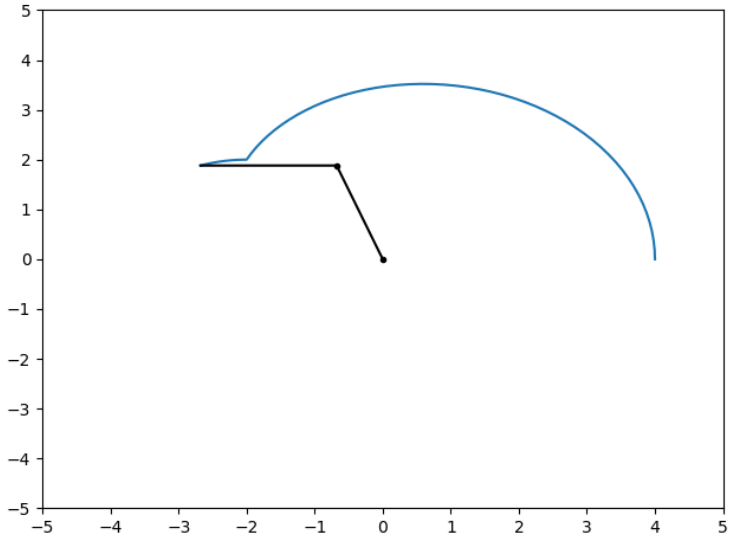
Ejercicio 6

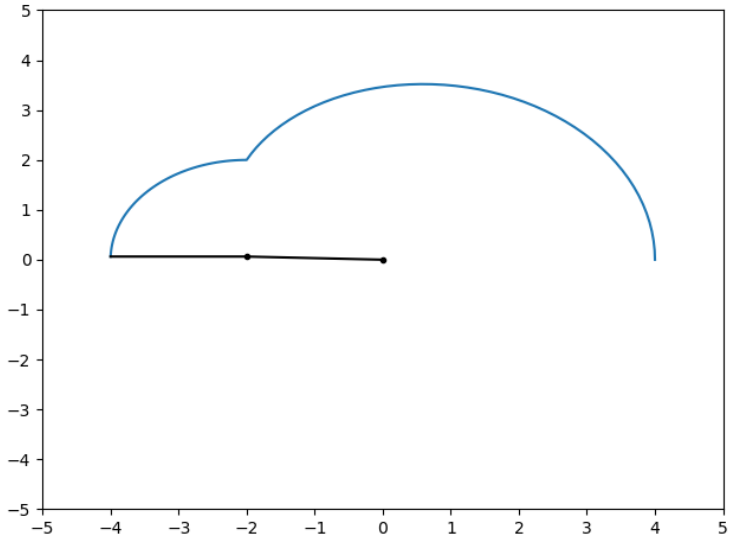
Comprobamos el correcto funcionamiento de la función anterior con un ejemplo en el que q_1 y q_2 toman los siguientes valores:
$q_1 \rightarrow 0, \pi/100, 2\pi/100, ..., \pi$
$q_2 \rightarrow 0, \pi/100, 2\pi/100, ..., 50\pi/100, 49\pi/100, 48\pi/100, ..., 0$
Mostramos varios estados de la animación:




5
4
3
2
1
0
−1
−2
−3
−4
−5
−5
−4
−3
−2
−1
0
1
2
3
4
5


5
4
3
2
1
0
−1
−2
−3
−4
−5
−5
−4
−3
−2
−1
0
1
2
3
4
5





Ejercicio 7

Tratamos ahora el problema cinemático inverso para un manipulador RR. Implementamos una función que dadas las coordenadas del extremo del robot (x, y) y las longitudes de los eslabones l_1 y l_2, obtenga los valores de q_1 y q_2. Volvemos a hacer uso del método geométrico para obtener los valores de q_1 y q_2 en función de x e y.

Utilizamos el ejemplo de la diapositiva 13 del tema 5, lo adaptamos al

caso 2D y obtenemos los siguientes resultados:

$$q_1 = arctan\frac{y}{x} - arctan\frac{l_2 * sen(q_2)}{l_1 + l_2 * cos(q_2)}$$

$$q_2 = arctan\frac{sen(q_2)}{cos(q_2)}$$

donde

$$cos(q_2) = \frac{x^2 + y^2 - l_1^2 - l_2^2}{2 * l_1 * l_2}$$

$$sen(q_2) = \pm\sqrt{1 - cos^2(q_2)}$$

Entonces, nuestra función **pci** simplemente tiene que calcular los ángulos q_1 y q_2 tal y como se indica arriba y devolver el resultado obtenido.

```
def pci(x, y, l1, l2):
    cosq2 = (x**2 + y**2 — l1**2 — l2**2) / (2*l1*l2)
    sinq2 = np.sqrt(1—cosq2**2)

    q1 = np.arctan(y/x) — np.arctan(l2*sinq2 / (l1 + l2*cosq2))
    q2 = np.arctan(sinq2/cosq2)

    return (q1, q2)
```

Ejercicio 8

Implementamos ahora funciones equivalentes a **dibujar_trayectoria_pcd** y **animacion_trayectoria_pci** pero para el caso del problema cinemático inverso.

```
def dibujar_trayectoria_pci(xs, ys, l1, l2):
    # Pintamos la trayectoria
    plt.plot(xs, ys)

    # Pintamos el robot
    q1, q2 = pci(xs[—1], ys[—1], l1, l2)
    dibujar_robot(q1, q2, l1, l2)

    # Centrar el origen de coordenadas
    rango = np.arange(—l1—l2—1, l1+l2+2)
    plt.xticks(rango)
```

```
    plt.yticks(rango)
    plt.show()
```

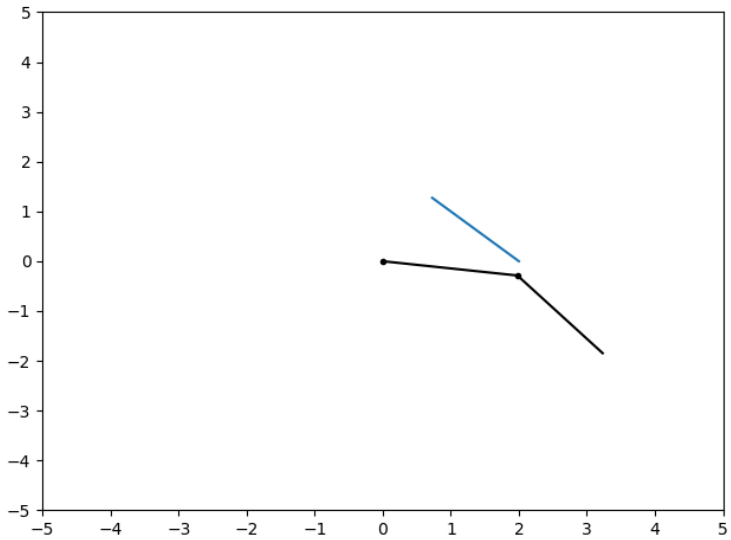
```
def animacion_trayectoria_pci(xs, ys, l1, l2):
    n = min(len(xs), len(ys))
    for i in range(1, n):
        plt.clf()
        dibujar_trayectoria_pci(xs[0:i], ys[0:i], l1, l2)
        plt.pause(0.001)
```

Ejercicio 9

Comprobamos el correcto funcionamiento mostrando el movimiento del robot cuando el extremo se mueve:

- Desde el punto (2, 0) hasta el punto (0, 2) en línea recta.
- Desde el punto (1, 1) hasta el punto (-2, 1) en línea recta.
- Desde el punto (1, 0) hasta el punto (-1, 0) en línea recta.
- Desde el punto (-2, 0) hasta el punto (2, 1) en línea recta.

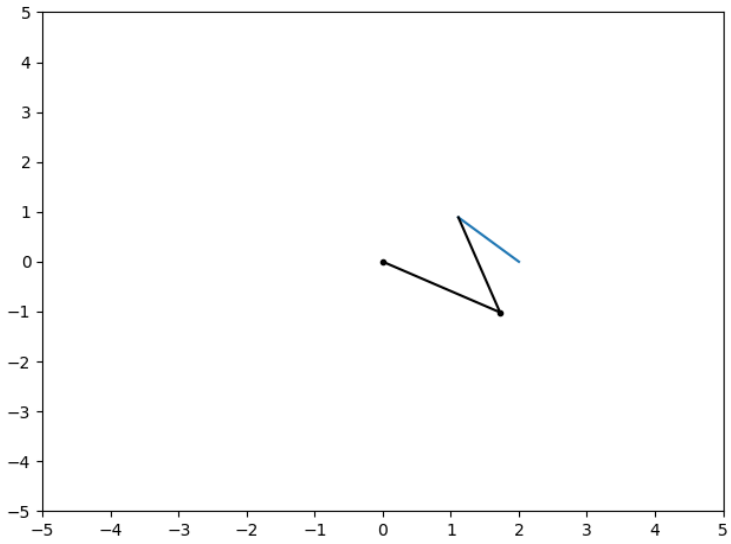
Mostramos un estado para el primer caso:



Podemos observar que el robot no se pinta correctamente. Esto se debe a que la función utilizada **np.arctan** devuelve un ángulo en el intervalo $[-\pi/2, \pi/2]$, es decir, no tiene en cuenta todos los cuadrantes. Para solucionar este problema modificamos la función **pci** para que haga uso de la función **np.arctan2** que sí devuelve el ángulo eligiendo correctamente el cuadrante.

```
def pci(x, y, l1, l2):
    cosq2 = (x**2 + y**2 — l1**2 — l2**2) / (2*l1*l2)
    sinq2 = np.sqrt(1—cosq2**2)

    q1 = np.arctan2(y, x) — np.arctan2(l2*sinq2, (l1 + l2*cosq2))
    q2 = np.arctan2(sinq2, cosq2)

    return (q1, q2, q1_neg, q2_neg)
```

Volvemos a mostrar un estado del primer ejemplo propuesto:



Vemos que ahora si se corresponde la posición del robot con la trayectoria seguida.

Ejercicio 10

Modificamos las funciones **pci** y **dibujar_trayectoria_pci** para que tengan en cuenta que hay dos posibles soluciones para los ángulos de forma que se utilice siempre la mejor solución y así evitar los cambios bruscos en el movimiento del robot. Primero hacemos que **pci** devuelva todas las soluciones:

```
def pci(x, y, l1, l2):
    cosq2 = (x**2 + y**2 — l1**2 — l2**2) / (2*l1*l2)
    sinq2 = np.sqrt(1—cosq2**2)

    q1 = np.arctan2(y, x) — np.arctan2(l2*sinq2, (l1 + l2*cosq2))
    q2 = np.arctan2(sinq2, cosq2)

    q1_neg = np.arctan2(y, x) — np.arctan2(l2*(—sinq2), (l1 + l2*cosq2))
    q2_neg = np.arctan2(—sinq2, cosq2)

    return (q1, q2, q1_neg, q2_neg)
```

Adaptamos **dibujar_trayectoria_pci** para que tenga en cuenta el ángulo utilizado en el paso anterior y así poder elegir la solución más próxima a este. Simplemente calculamos el valor absoluto de la diferencia entre el ángulo anterior y los dos posibles soluciones y nos quedamos con la que produzca una menor variación. Hay un caso especial en el que la mejor solución no es la de menor variación debido a que los ángulos del cuarto cuadrante se miden en sentido horario ($[-\pi/2, 0]$) mientras que los demás cuadrantes se miden en sentido antihorario. Por esto, hay que comprobar cuando el ángulo pasa del cuarto cuadrante al tercero ya que la diferencia será grande aunque en realidad la variación entre los ángulos es pequeña.

```
def dibujar_trayectoria_pci(xs, ys, l1, l2, qi_anteriores):
    plt.plot(xs, ys)

    q1, q2, q1_neg, q2_neg = pci(xs[—1], ys[—1], l1, l2)

    if qi_anteriores is not None:
        variacion_q1 = abs(qi_anteriores[0]—q1)
        variacion_q1_neg = abs(qi_anteriores[0]—q1_neg)

        # Los dos primeros casos son para el cambio del tercer cuadrante
        # al cuarto que se pasa de 3pi/2 a —pi/2 o el caso contrario.
        # Entonces, la diferencia de los ángulos es muy grande aunque
        # en realidad la variación en el espacio es pequeña
        if variacion_q1 > np.pi + np.pi/2:
            qi_usados = (q1, q2)
        elif variacion_q1_neg > np.pi + np.pi/2:
            qi_usados = (q1_neg, q2_neg)
```

```
        # Se compara cuál de los dos ángulos es más próximo
        elif variacion_q1 < variacion_q1_neg:
            qi_usados = (q1, q2)
        else:
            qi_usados = (q1_neg, q2_neg)
    else:
        qi_usados = (q1, q2)

    dibujar_robot(*qi_usados, l1, l2)

    # Centrar el origen de coordenadas
    rango = np.arange(—l1—l2—1, l1+l2+2)
    plt.xticks(rango)
    plt.yticks(rango)

    plt.show()

    return qi_usados
```

Archivos de código

cinematico_directo.py

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

def pcd(q1, q2, l1, l2):
    x = l1*np.cos(q1) + l2*np.cos(q1+q2)
    y = l1*np.sin(q1) + l2*np.sin(q1+q2)
    return (x, y)

def dibujar_trayectoria_pcd(q1s, q2s, l1, l2):
    # Calculamos todos los puntos y los pintamos
    puntos = [pcd(q1, q2, l1, l2) for q1, q2 in zip(q1s, q2s)]
    plt.plot(*zip(*puntos))

    dibujar_robot(q1s[-1], q2s[-1], l1, l2)

    # Centrar el origen de coordenadas
    rango = np.arange(-l1-l2-1, l1+l2+2)
    plt.xticks(rango)
    plt.yticks(rango)

    plt.show()

def dibujar_robot(q1, q2, l1, l2):
    x0, y0 = 0, 0                       # Posición de la articulación 1
    x1, y1 = pcd(q1, 0, l1, 0)          # Posición de la articulación 2
    x2, y2 = pcd(q1, q2, l1, l2)        # Posición del extremo del robot

    x, y = [x0, x1, x2], [y0, y1, y2]   # Coordenadas de la trayectoria
    plt.plot(x, y, 'k')                 # Traza la trayectoria
    plt.plot(x0, y0, 'k.')              # Dibuja la articulación 1
    plt.plot(x1, y1, 'k.')              # Dibuja la articulación 2

def animacion_trayectoria_pcd(q1s, q2s, l1, l2):
    n = min(len(q1s), len(q2s))
    for i in range(1, n):
        plt.clf()
        dibujar_trayectoria_pcd(q1s[0:i], q2s[0:i], l1, l2)
        plt.pause(0.001)
```

```
if __name__ == "__main__":
    n = 100
    l1 = l2 = 2

    q1s = [np.pi / 4] * n
    q2s = np.linspace(0, np.pi/2, n)
    dibujar_trayectoria_pcd(q1s, q2s, l1, l2)

    q1s = np.linspace(np.pi/4, np.pi/2, n)
    q2s = np.linspace(0, np.pi/2, n)
    dibujar_trayectoria_pcd(q1s, q2s, l1, l2)

    q1s = [i*np.pi / n for i in range(n+1)]
    q2s = q1s[:n//2+1] + q1s[n//2-1::-1]
    animacion_trayectoria_pcd(q1s, q2s, l1, l2)
```

cinematico_inverso.py

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

from cinematico_directo import dibujar_robot

def pci(x, y, l1, l2):
    cosq2 = (x**2 + y**2 - l1**2 - l2**2) / (2*l1*l2)
    sinq2 = np.sqrt(1-cosq2**2)

    # q1 = np.arctan(y/x) - np.arctan(l2*sinq2 / (l1 + l2*cosq2))
    # q2 = np.arctan(sinq2/cosq2)

    q1 = np.arctan2(y, x) - np.arctan2(l2*sinq2, (l1 + l2*cosq2))
    q2 = np.arctan2(sinq2, cosq2)

    q1_neg = np.arctan2(y, x) - np.arctan2(l2*(-sinq2), (l1 + l2*cosq2))
    q2_neg = np.arctan2(-sinq2, cosq2)

    return (q1, q2, q1_neg, q2_neg)
```

```
def dibujar_trayectoria_pci(xs, ys, l1, l2, qi_anteriores):
    # Pintamos la trayectoria
    plt.plot(xs, ys)

    # Pintamos el robot
    q1, q2, q1_neg, q2_neg = pci(xs[-1], ys[-1], l1, l2)

    if qi_anteriores is not None:
        variacion_q1 = abs(qi_anteriores[0]-q1)
        variacion_q1_neg = abs(qi_anteriores[0]-q1_neg)

        # Los dos primeros casos son para el cambio del tercer cuadrante
        # al cuarto que se pasa de 3pi/2 a -pi/2 o el caso contrario.
        # Entonces, la diferencia de los ángulos es muy grande aunque
        # en realidad la variación en el espacio es pequeña
        if variacion_q1 > np.pi + np.pi/2:
            qi_usados = (q1, q2)
        elif variacion_q1_neg > np.pi + np.pi/2:
            qi_usados = (q1_neg, q2_neg)

        # Se compara cuál de los dos ángulos es más próximo
        elif variacion_q1 < variacion_q1_neg:
            qi_usados = (q1, q2)
        else:
            qi_usados = (q1_neg, q2_neg)
    else:
        qi_usados = (q1, q2)

    dibujar_robot(*qi_usados, l1, l2)

    # Centrar el origen de coordenadas
    rango = np.arange(-l1-l2-1, l1+l2+2)
    plt.xticks(rango)
    plt.yticks(rango)

    plt.show()

    return qi_usados

def animacion_trayectoria_pci(xs, ys, l1, l2):
    n = min(len(xs), len(ys))
```

```
    qi_anteriores = None
    for i in range(1, n):
        plt.clf()
        qi_anteriores = dibujar_trayectoria_pci(xs[0:i], ys[0:i],
                                                l1, l2, qi_anteriores)
        plt.pause(0.001)

n = 100
l1 = l2 = 2

xs = np.linspace(2, 0, n)
ys = np.linspace(0, 2, n)
animacion_trayectoria_pci(xs, ys, l1, l2)

xs = np.linspace(1, -2, n)
ys = [1] * n
animacion_trayectoria_pci(xs, ys, l1, l2)

xs = np.linspace(1, -1, n)
ys = [0] * n
animacion_trayectoria_pci(xs, ys, l1, l2)

xs = np.linspace(-2, 2, n)
ys = np.linspace(0, 1, n)
animacion_trayectoria_pci(xs, ys, l1, l2)
```