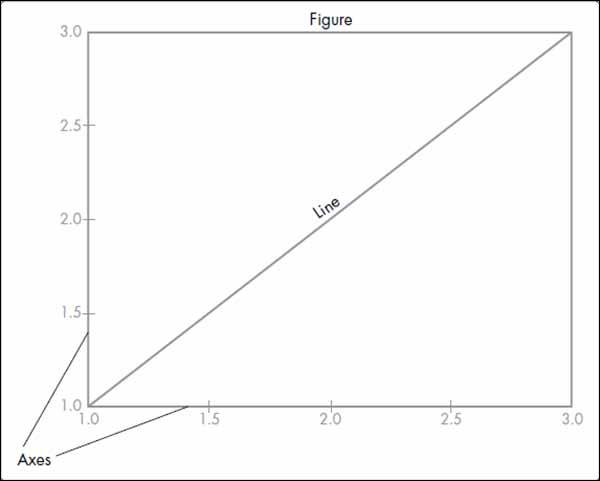
Capítulo 6: Dibujo de formas geométricas y fractales

### **Dibujar formas geométricas con los parches de Matplotlib**

En matplotlib, *los par* ches nos permiten dibujar formas geométricas, a cada una de las cuales nos referimos como un *parche*. Puedes especificar, por ejemplo, el radio y el centro de un círculo para añadir el círculo correspondiente a tu gráfico. Esto es bastante diferente de cómo hemos utilizado matplotlib hasta ahora, que ha consistido en suministrar las *coordenadas* *x* e *y* de los puntos a trazar. Sin embargo, antes de que podamos escribir un programa para utilizar la función de parches, tendremos que entender un poco mejor cómo se crea un gráfico con matplotlib. Considera el siguiente programa, que traza los puntos (1, 1), (2, 2) y (3, 3) utilizando matplotlib:

>>> import matplotlib.pyplot as plt  
>>> x = [1, 2, 3]  
>>> y = [1, 2, 3]  
>>> plt.plot(x, y)  
[<matplotlib.lines.Line2D object at 0x7fe822d67a20>]  
>>> plt.show()

Este programa crea una ventana matplotlib que muestra una línea que pasa por los puntos dados. En realidad, cuando se llama a la función plt.plot(), se crea un objeto Figure, dentro del cual se crean los ejes y, finalmente, se trazan los datos dentro de los ejes (ver [Figura 6-1](ch06.html#ch6fig1))[.1](footnote.html#fn03)



*Figura 6-1: Arquitectura de un gráfico matplotlib*

El siguiente programa vuelve a crear este gráfico, pero también crearemos explícitamente el objeto Figure y le añadiremos ejes, en lugar de limitarnos a llamar a la función plot() y confiar en ella para crearlos:

>>> import matplotlib.pyplot as plt  
>>> x = [1, 2, 3]  
>>> y = [1, 2, 3]  
➊ >>> fig = plt.figure()  
➋ >>> ax = plt.axes()  
>>> plt.plot(x, y)  
[<matplotlib.lines.Line2D object at 0x7f9bad1dcc18>]  
>>> plt.show()  
>>>

Aquí, creamos el objeto Figure utilizando la función figure() en ➊, y luego creamos los ejes utilizando la función axes() en ➋. La función axes() también añade los ejes al objeto Figure. Las dos últimas líneas son iguales que en el programa anterior. Esta vez, cuando llamamos a la función plot(), ésta ve que ya existe un objeto Figure con un objeto Axes y procede directamente a trazar los datos que se le suministran.

Además de crear manualmente los objetos Figure y Axes, puedes utilizar dos funciones diferentes del módulo pyplot para obtener una referencia a los objetos Figure y Axes actuales. Cuando llamas a la función gcf(), te devuelve una referencia al Figure actual, y cuando llamas a la función gca(), te devuelve una referencia al Axes actual. Una característica interesante de estas funciones es que cada una creará el objeto respectivo si aún no existe. El funcionamiento de estas funciones quedará más claro cuando las utilicemos más adelante en este capítulo.

#### ***Dibujar un círculo***

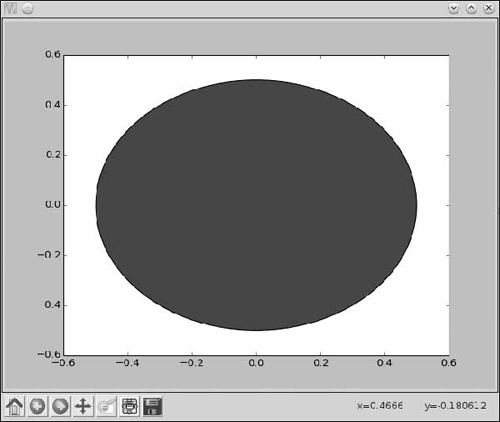
Para dibujar un círculo, puedes añadir el parche Circle al objeto actual Axes, como se demuestra en el siguiente ejemplo:

'''  
Example of using matplotlib's Circle patch  
'''  
import matplotlib.pyplot as plt  
  
def create\_circle():  
➊ circle = plt.Circle((0, 0), radius = 0.5)  
return circle  
  
def show\_shape(patch):  
➋ ax = plt.gca()  
ax.add\_patch(patch)  
plt.axis('scaled')  
plt.show()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
➌ c = create\_circle()  
show\_shape(c)

En este programa, hemos separado la creación del objeto parche Circle y la adición del parche a la figura en dos funciones: create\_circle() y show\_shape(). En create\_circle(), hacemos un círculo con centro en (0, 0) y radio de 0,5 creando un objeto Circle con las coordenadas del centro (0, 0) pasadas como tupla y con el radio de 0,5 pasado mediante el argumento de palabra clave del mismo nombre en ➊. La función devuelve el objeto Circle creado.

La función show\_shape() está escrita de forma que funcione con cualquier parche de matplotlib. Primero obtiene una referencia al objeto Axes actual utilizando la función gca() en ➋. Después, añade el parche que se le ha pasado utilizando la función add\_patch() y, por último, llama a la función show() para mostrar la figura. Aquí llamamos a la función axis() con el parámetro scaled, que básicamente indica a matplotlib que ajuste automáticamente los límites de los ejes. Necesitaremos esta declaración en todos los programas que utilicen parches para escalar automáticamente los ejes. Por supuesto, también puedes especificar valores fijos para los límites, como vimos en el [capítulo 2](ch02.html#ch02).

En ➌, llamamos a la función create\_circle() utilizando la etiqueta c para referirnos al objeto Circle devuelto. A continuación, llamamos a la función show\_shape(), pasando c como argumento. Cuando ejecutes el programa, verás una ventana matplotlib que muestra el círculo (ver [Figura 6-2](ch06.html#ch6fig2)).



*Figura 6-2: Un círculo con centro en (0, 0) y radio de 0,5*

Como puedes ver, el círculo no parece exactamente un círculo. Esto se debe a la relación de aspecto automática, que determina la relación entre la longitud de los *ejes* *x* e *y*. Si insertas la sentencia ax.set\_aspect('equal') después de ➋, verás que el círculo sí parece un círculo. La función set\_aspect() se utiliza para establecer la relación de aspecto del gráfico; utilizando el argumento equal, pedimos a matplotlib que establezca la relación de la longitud de los *ejes* *x* e y en 1:1.

Tanto el color de las aristas como el color de las caras (color de relleno) del parche pueden cambiarse utilizando los argumentos de palabra clave ec y fc. Por ejemplo, si pasas fc='g' y ec='r' crearás un círculo con un color de cara verde y un color de borde rojo.

Matplotlib admite otros parches, como Ellipse, Polygon y Rectangle.

#### ***Crear figuras animadas***

A veces podemos querer crear figuras con formas en movimiento. El soporte para animación de Matplotlib nos ayudará a conseguirlo. Al final de esta sección, crearemos una versión animada del programa de dibujo de trayectorias de proyectiles.

Primero, veamos un ejemplo más sencillo. Dibujaremos una figura de matplotlib con un círculo que empieza siendo pequeño y crece hasta un radio determinado indefinidamente (a menos que se cierre la ventana de matplotlib):

'''  
A growing circle  
'''  
  
from matplotlib import pyplot as plt  
from matplotlib import animation  
  
def create\_circle():  
circle = plt.Circle((0, 0), 0.05)  
return circle  
  
def update\_radius(i, circle):  
circle.radius = i\*0.5  
return circle,  
  
def create\_animation():  
➊ fig = plt.gcf()  
ax = plt.axes(xlim=(-10, 10), ylim=(-10, 10))  
ax.set\_aspect('equal')  
circle = create\_circle()  
➋ ax.add\_patch(circle)  
➌ anim = animation.FuncAnimation(  
fig, update\_radius, fargs = (circle,), frames=30, interval=50)  
plt.title('Simple Circle Animation')  
plt.show()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
create\_animation()

Empezaremos importando el módulo animation del paquete matplotlib. La función create\_animation() realiza aquí la funcionalidad principal. Obtiene una referencia al objeto Figure actual utilizando la función gcf() en ➊ y luego crea los ejes con límites de -10 y 10 tanto para *el* *eje* *x* como para el *eje y*. Después, crea un objeto Circle que represente un círculo con radio 0,05 y centro en (0, 0) y añade este círculo a los ejes actuales en ➋. A continuación, creamos un objeto FuncAnimation ➌, que pasa los siguientes datos sobre la animación que queremos crear:

fig Este es el objeto Figure actual.

update\_radius Esta función se encargará de dibujar *cada* fotograma. Toma dos argumentos: un número de fotograma que se le pasa automáticamente al llamarla y el objeto parche que queremos actualizar cada fotograma. Esta función también debe devolver el objeto.

fargs Esta tupla consta de todos los argumentos que hay que pasar a la función update\_radius() aparte del número de fotograma. Si no hay argumentos que pasar, no es necesario especificar este argumento.

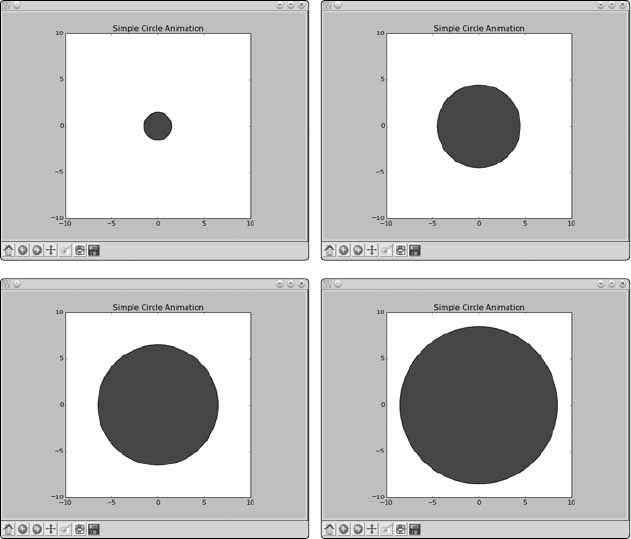
frames Es el número de fotogramas de la animación. Nuestra función update\_radius() es llamada así muchas veces. Aquí, hemos elegido arbitrariamente 30 fotogramas.

interval Este es el intervalo de tiempo en milisegundos entre dos fotogramas. Si tu animación parece demasiado lenta, disminuye este valor; si parece demasiado rápida, auméntalo.

A continuación, establecemos un título mediante la función title() y, por último, mostramos la figura mediante la función show().

Como ya hemos dicho, la función update\_radius() se encarga de actualizar la propiedad del círculo que cambiará en cada fotograma. Aquí, fijamos el radio en i\*0.5, donde i es el número de fotograma. Como resultado, verás un círculo que crece en cada fotograma durante 30 fotogramas, por lo que el radio del círculo más grande es 15. Como los límites de los ejes están fijados en -10 y 10, esto da el efecto de que el círculo sobrepasa las dimensiones de la figura. Cuando ejecutes el programa, verás tu primera figura animada, como se muestra en la [Figura 6-3](ch06.html#ch6fig3).

Observarás que la animación continúa hasta que cierras la ventana de matplotlib. Éste es el comportamiento por defecto, que puedes cambiar estableciendo el argumento de la palabra clave en repeat=False cuando crees el objeto FuncAnimation.



*Figura 6-3: Animación de un círculo simple*

**OBJETO FUNCANIMACIÓN Y PERSISTENCIA**

Probablemente hayas observado en el programa del círculo animado que asignamos el objeto FuncAnimation creado a la etiqueta anim aunque no volvamos a utilizarlo en ningún otro lugar. Esto se debe a un problema con el comportamiento actual de matplotlib: no almacena ninguna referencia al objeto FuncAnimation, por lo que está sujeto a la recolección de basura por parte de Python. Esto significa que la animación no se creará. Crear una etiqueta que haga referencia al objeto evita que esto ocurra.

Para saber más sobre este tema, puedes seguir las discusiones en [*https://github.com/matplotlib/matplotlib/issues/1656/.*](https://github.com/matplotlib/matplotlib/issues/1656/)

#### ***Animar la trayectoria de un proyectil***

En el [Capítulo 2](ch02.html#ch02), dibujamos la trayectoria de una pelota en movimiento de proyectil. Aquí, nos basaremos en este dibujo, haciendo uso del soporte de animación de matplotlib para animar la trayectoria, de modo que se acerque más a la demostración de cómo verías viajar una pelota en la vida real:

'''  
Animate the trajectory of an object in projectile motion  
'''  
  
from matplotlib import pyplot as plt  
from matplotlib import animation  
import math  
  
g = 9.8  
  
def get\_intervals(u, theta):  
  
t\_flight = 2\*u\*math.sin(theta)/g  
intervals = []  
start = 0  
interval = 0.005  
while start < t\_flight:  
intervals.append(start)  
start = start + interval  
return intervals  
  
def update\_position(i, circle, intervals, u, theta):  
  
t = intervals[i]  
x = u\*math.cos(theta)\*t  
y = u\*math.sin(theta)\*t - 0.5\*g\*t\*t  
circle.center = x, y  
return circle,  
  
def create\_animation(u, theta):  
  
intervals = get\_intervals(u, theta)  
  
xmin = 0  
xmax = u\*math.cos(theta)\*intervals[-1]  
ymin = 0  
t\_max = u\*math.sin(theta)/g  
➊ ymax = u\*math.sin(theta)\*t\_max - 0.5\*g\*t\_max\*\*2  
fig = plt.gcf()  
➋ ax = plt.axes(xlim=(xmin, xmax), ylim=(ymin, ymax))  
  
circle = plt.Circle((xmin, ymin), 1.0)  
ax.add\_patch(circle)  
  
➌ anim = animation.FuncAnimation(fig, update\_position,  
fargs=(circle, intervals, u, theta),  
frames=len(intervals), interval=1,  
repeat=False)  
  
plt.title('Projectile Motion')  
plt.xlabel('X')  
plt.ylabel('Y')  
plt.show()  
  
if \_\_name\_\_ == '\_\_main\_\_':  
try:  
u = float(input('Enter the initial velocity (m/s): '))  
theta = float(input('Enter the angle of projection (degrees): '))  
except ValueError:  
print('You entered an invalid input')  
else:  
theta = math.radians(theta)  
create\_animation(u, theta)

La función create\_animation() acepta dos argumentos: u y theta. Estos argumentos corresponden a la velocidad inicial y al ángulo de proyección*(θ*), que se suministraron como entrada al programa. La función get\_intervals() se utiliza para encontrar los intervalos de tiempo en los que calcular las *coordenadas* *x* e *y*. Esta función se implementa haciendo uso de la misma lógica que utilizamos en el [Capítulo 2](ch02.html#ch02), cuando implementamos una función independiente, frange(), para ayudarnos.

Para establecer los límites de los ejes de la animación, necesitaremos encontrar los valores mínimo y máximo de *x* e *y*. El valor mínimo de cada uno es 0, que es el valor inicial de cada uno. El valor máximo de la coordenada *x* es el valor de la coordenada al final del vuelo de la pelota, que es el último intervalo de tiempo de la lista intervals. El valor máximo de la coordenada *y* es cuando la bola está en su punto más alto, es decir, en ➊, donde calculamos ese punto mediante la fórmula

image

Una vez que tenemos los valores, creamos los ejes en ➋, pasando los límites de eje adecuados. En las dos sentencias siguientes, creamos una representación de la bola y la añadimos al objeto Axes de la figura creando un círculo de radio 1.0 en (xmin, ymin)-las coordenadas mínimas de los ejes *x* e *y*, respectivamente.

A continuación, creamos el objeto FuncAnimation ➌, suministrándole el objeto figura actual y los siguientes argumentos:

update\_position Esta función cambiará el centro del círculo en cada fotograma. La idea aquí es que se crea un nuevo fotograma por cada intervalo de tiempo, por lo que fijamos el número de fotogramas al tamaño de los intervalos de tiempo (consulta la descripción de frames en esta lista). Calculamos las *coordenadas* *x e* *y* de la bola en el instante de tiempo en el iintervalo de tiempo, y fijamos el centro del círculo a estos valores.

fargs La función update\_position() necesita acceder a la lista de intervalos de tiempo, intervalos, velocidad inicial y theta, que se especifican mediante este argumento de palabra clave.

frames Como dibujaremos un fotograma por intervalo de tiempo, fijamos el número de fotogramas al tamaño de la lista intervals.

repeat Como ya comentamos en el primer ejemplo de animación, la animación se repite indefinidamente por defecto. No queremos que eso ocurra en este caso, así que fijamos esta palabra clave en False.

Cuando ejecutes el programa, te pedirá las entradas iniciales y luego creará la animación, como se muestra en la [Figura 6-4](ch06.html#ch6fig4).



*Figura 6-4: Animación de la trayectoria de un proyectil*

[anterior](ch06_1.html)[Subtema 2 de 5: (Ver todo)](ch06.html)[siguiente](ch06_3.html)