Capítulo 2: Visualizar datos con gráficos

### **Trazar con fórmulas**

Hasta ahora, hemos estado trazando puntos en nuestros gráficos basándonos en medidas científicas observadas. En esas gráficas, ya teníamos todos nuestros valores para *x* e *y* establecidos. Por ejemplo, las temperaturas y fechas registradas ya estaban a nuestra disposición en el momento en que quisimos crear el gráfico de la ciudad de Nueva York, mostrando cómo variaba la temperatura a lo largo de meses o años. En esta sección, vamos a crear gráficos a partir de fórmulas matemáticas.

#### ***Ley de Gravitación Universal de Newton***

Según la ley de gravitación universal de Newton, un cuerpo de masa m1 atrae a otro cuerpo de masam2 con una fuerza *F* según la fórmula

image

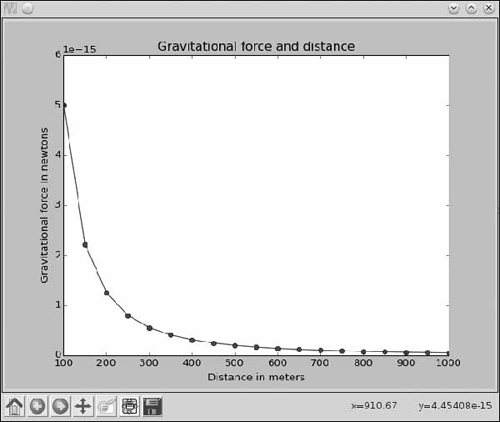
donde *r* es la distancia entre los dos cuerpos y *G* es la constante gravitatoria. Queremos ver qué ocurre con la fuerza a medida que aumenta la distancia entre los dos cuerpos.

Tomemos las masas de dos cuerpos: la masa del primer cuerpo*(*m1) es de 0,5 kg, y la masa del segundo cuerpo*(*m2) es de 1,5 kg. El valor de la constante gravitatoria es 6,674 × 10-11 Nm2 kg-2. Ahora estamos preparados para calcular la fuerza gravitatoria entre estos dos cuerpos a 19 distancias diferentes: 100 m, 150 m, 200 m, 250 m, 300 m, y así hasta 1000 m. El siguiente programa realiza estos cálculos y también dibuja la gráfica:

'''  
The relationship between gravitational force and  
distance between two bodies  
'''  
  
import matplotlib.pyplot as plt  
  
# Draw the graph  
def draw\_graph(x, y):  
plt.plot(x, y, marker='o')  
plt.xlabel('Distance in meters')  
  
plt.ylabel('Gravitational force in newtons')  
plt.title('Gravitational force and distance')  
plt.show()  
  
def generate\_F\_r():  
# Generate values for r  
➊ r = range(100, 1001, 50)  
# Empty list to store the calculated values of F  
F = []  
  
# Constant, G  
G = 6.674\*(10\*\*-11)  
# Two masses  
m1 = 0.5  
m2 = 1.5  
  
# Calculate force and add it to the list, F  
➋ for dist in r:  
force = G\*(m1\*m2)/(dist\*\*2)  
F.append(force)  
  
# Call the draw\_graph function  
➌ draw\_graph(r, F)  
  
if \_\_name\_\_=='\_\_main\_\_':  
generate\_F\_r()

La función generate\_F\_r() realiza la mayor parte del trabajo en el programa anterior. En ➊, utilizamos la función range() para crear una lista denominada r con distintos valores para la distancia, utilizando un valor de paso de 50. El valor final se especifica como 1001 porque queremos que también se incluya 1000. A continuación, creamos una lista vacía (F), donde almacenaremos la fuerza gravitatoria correspondiente a cada una de estas distancias. A continuación, creamos etiquetas referidas a la constante gravitatoria (G) y a las dos masas (m1 y m2). Utilizando un bucle for ➋, calculamos entonces la fuerza en cada uno de los valores de la lista de distancias (r). Utilizamos una etiqueta (force) para referirnos a la fuerza calculada y añadirla a la lista (F). Por último, llamamos a la función draw\_graph() en ➌ con la lista de distancias y la lista de las fuerzas calculadas. El *eje x* del gráfico muestra la fuerza, y el *eje y* muestra la distancia. El gráfico se muestra en la [Figura 2-12](ch02.html#ch2fig12).

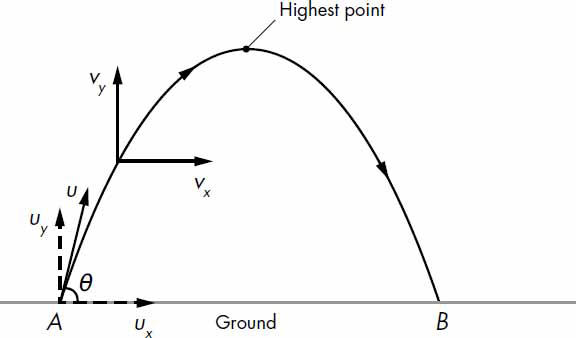
A medida que aumenta la distancia (r), disminuye la fuerza gravitatoria. Con este tipo de relación, decimos que la fuerza gravitatoria es *inversamente proporcional* a la distancia entre los dos cuerpos. Además, ten en cuenta que cuando cambia el valor de una de las dos variables, la otra variable no cambiará necesariamente en la misma proporción. A esto lo llamamos *relación no lineal*. Como resultado, acabamos teniendo una línea curva en la gráfica en lugar de una recta.



*Figura 2-12: Visualización de la relación entre la fuerza gravitatoria y la distancia al cuadrado*

#### ***Movimiento de proyectiles***

Ahora vamos a representar gráficamente algo que te resultará familiar de la vida cotidiana. Si lanzas una pelota a través de un campo, seguirá una trayectoria como la que se muestra en la [Figura 2](ch02.html#ch2fig13)-13.



*Figura 2-13: Movimiento de una pelota lanzada en* el punto A -con un*ángulo (*θ) *y una velocidad (*U)*- y que cae al suelo en el punto* B

En la figura, la pelota es lanzada desde el punto *A* y cae en el punto *B*. Este tipo de movimiento se denomina movimiento de *proyectil*. Nuestro objetivo aquí es utilizar las ecuaciones del movimiento de proyectil para representar gráficamente la trayectoria de un cuerpo, mostrando la posición de la pelota desde el punto en que es lanzada hasta que vuelve a tocar el suelo.

Cuando lanzas la pelota, ésta tiene una velocidad inicial y la dirección de esa velocidad crea un determinado ángulo con el suelo. Llamemos a la velocidad inicial *u* y al ángulo que forma con el suelo *θ* (theta), como se muestra en la [Figura 2-13](ch02.html#ch2fig13). La bola tiene dos componentes de velocidad: una en la dirección *x*, calculada por *ux* = *u* *cosθ*, y otra en la dirección *y*, donde *uy* = *u* *sinθ*.

A medida que la bola se mueve, su velocidad cambia, y representaremos ese cambio de velocidad mediante *v*: la componente horizontal es *vx* y la componente vertical es *vy*. Para simplificar, supongamos que la componente horizontal*(vx*) no cambia durante el movimiento del cuerpo, mientras que la componente vertical*(vy*) disminuye debido a la fuerza de la gravedad según la ecuación *vy* = *uy* *- gt*. En esta ecuación, *g* es la aceleración gravitatoria y *t* es el tiempo en el que se mide la velocidad. Como *uy* = *u* *sinθ*, podemos sustituir para obtener

*vy* = *u* *sinθ* - *gt*.

Como la componente horizontal de la velocidad permanece constante, la distancia horizontal recorrida*(Sx*) viene dada por *Sx* = *u*(*cosθ)t*. Sin embargo, la componente vertical de la velocidad cambia, y la distancia vertical recorrida viene dada por la fórmula

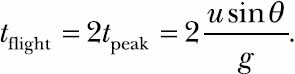
image

En otras palabras, *Sx* y *Sy* nos dan las *coordenadas* *x* e *y* de la bola en un momento dado de su vuelo. Utilizaremos estas ecuaciones cuando escribamos un programa para dibujar la trayectoria. Cuando utilicemos estas ecuaciones, el tiempo*(t*) se expresará en segundos, la velocidad se expresará en m/s, el ángulo de proyección*(θ*) se expresará en grados y la aceleración gravitatoria*(g*) se expresará en m/s2.

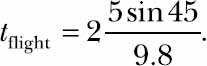
Sin embargo, antes de escribir nuestro programa, tendremos que averiguar cuánto tiempo estará la pelota en vuelo antes de tocar el suelo, para saber cuándo nuestro programa debe dejar de trazar la trayectoria de la pelota. Para ello, primero averiguaremos cuánto tarda la pelota en alcanzar su punto más alto. La pelota alcanza su punto más alto cuando la componente vertical de la velocidad*(vy*) es 0, es decir, cuando *vy* = *u* sen *θ* - *gt* = 0. Por tanto, buscamos el valor *t* mediante la fórmula

image

Llamaremos a este tiempo t\_peak. Tras alcanzar su punto más alto, la pelota tocará el suelo después de estar en el aire durante otros t\_peak segundos, por lo que el tiempo total de vuelo (t\_flight) de la pelota es



Tomemos una pelota lanzada con una velocidad inicial*(u*) de 5 m/s en un ángulo*(θ*) de 45 grados. Para calcular el tiempo total de vuelo, sustituimos *u* = 5, *θ* = 45 y *g* = 9,8 en la ecuación que vimos anteriormente:



En este caso, el tiempo de vuelo de la pelota resulta ser de 0,72154 segundos (redondeado a cinco decimales). La pelota estará en el aire durante este periodo de tiempo, así que para dibujar la trayectoria, calcularemos sus *coordenadas* *x* e y a intervalos regulares durante este periodo de tiempo. ¿Con qué frecuencia debemos calcular las coordenadas? Idealmente, con la mayor frecuencia posible. En este capítulo, calcularemos las coordenadas cada 0,001 segundos.

##### **Generación de números en coma flotante equidistantes**

Hemos utilizado la función range() para generar números enteros igualmente espaciados; es decir, si quisiéramos una lista de números enteros entre 1 y 10 con cada número entero separado por 1, utilizaríamos range(1, 10). Si quisiéramos un valor de paso diferente, podríamos especificarlo a la función rango como tercer argumento. Por desgracia, no existe una función incorporada de este tipo para los números de coma flotante. Así que, por ejemplo, no hay ninguna función que nos permita crear una lista de los números de 0 a 0,72 con dos números consecutivos separados por 0,001. Podemos utilizar un bucle while como el siguiente para crear nuestra propia función para esto:

'''  
Generate equally spaced floating point  
numbers between two given values  
'''  
  
def frange(start, final, increment):  
  
numbers = []  
➊ while start < final:  
➋ numbers.append(start)  
start = start + increment  
  
return numbers

Hemos definido una función frange() (rango "punto flotante") que recibe tres parámetros: start y final se refieren a los puntos inicial y final del rango de números, y increment se refiere a la diferencia entre dos números consecutivos. Inicializamos un bucle while en ➊, que continúa la ejecución mientras el número apuntado por start sea menor que el valor de final. Almacenamos el número apuntado por start en la lista numbers ➋ y luego añadimos el valor que hemos introducido como increment en cada iteración del bucle. Por último, devolvemos la lista numbers.

Utilizaremos esta función para generar instantes de tiempo igualmente espaciados en el programa de dibujo de trayectorias que se describe a continuación.

##### **Dibujar la trayectoria**

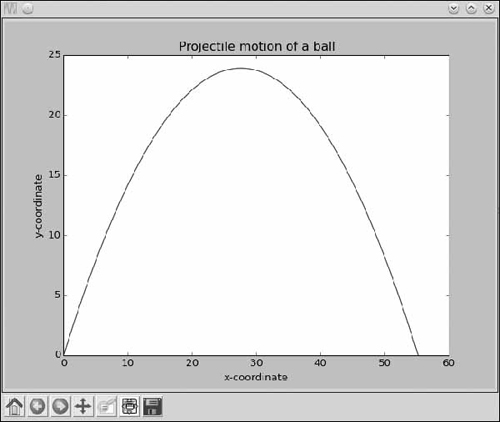
El siguiente programa dibuja la trayectoria de una pelota lanzada con una velocidad y un ángulo determinados, ambos suministrados como entrada al programa:

'''  
Draw the trajectory of a body in projectile motion  
'''  
  
from matplotlib import pyplot as plt  
import math  
  
def draw\_graph(x, y):  
plt.plot(x, y)  
plt.xlabel('x-coordinate')  
plt.ylabel('y-coordinate')  
plt.title('Projectile motion of a ball')  
  
def frange(start, final, interval):  
  
numbers = []  
while start < final:  
numbers.append(start)  
start = start + interval  
  
return numbers  
  
def draw\_trajectory(u, theta):  
  
➊ theta = math.radians(theta)  
g = 9.8  
  
# Time of flight  
➋ t\_flight = 2\*u\*math.sin(theta)/g  
# Find time intervals  
intervals = frange(0, t\_flight, 0.001)  
  
# List of x and y coordinates  
x = []  
y = []  
➌ for t in intervals:  
x.append(u\*math.cos(theta)\*t)  
y.append(u\*math.sin(theta)\*t - 0.5\*g\*t\*t)  
  
draw\_graph(x, y)  
  
if \_\_name\_\_ == '\_\_main\_\_':  
➍ 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:  
draw\_trajectory(u, theta)  
plt.show()

En este programa, utilizamos las funciones radians(), cos(), y sin() definidas en el módulo math de la biblioteca estándar, por lo que importamos dicho módulo al principio. La función draw\_trajectory() acepta dos argumentos, u y theta, correspondientes a la velocidad y al ángulo con el que se lanza la bola. Las funciones seno y coseno del módulo math esperan que el ángulo se proporcione en radianes, así que en ➊, convertimos el ángulo (theta) de grados a radianes utilizando la función math.radians(). A continuación, creamos una etiqueta (g) para referirnos al valor de la aceleración debida a la gravedad, 9,8 m/s2. En ➋, calculamos el tiempo de vuelo y luego llamamos a la función frange() con los valores de start, final, y increment fijados en 0, t\_flight, y 0,001, respectivamente. A continuación, calculamos las coordenadas *x* e *y* de la trayectoria en cada uno de los instantes de tiempo y las almacenamos en dos listas distintas, x y y ➌. Para calcular estas coordenadas, utilizamos las fórmulas para las distancias *Sx* y *Sy* que hemos discutido antes.

Por último, llamamos a la función draw\_graph() con las coordenadas *x* e *y* para dibujar la trayectoria. Observa que la función draw\_graph() no llama a la función show() (veremos por qué en el siguiente programa). Utilizamos un bloque try...except ➍ para informar de un mensaje de error en caso de que el usuario introduzca una entrada no válida. La entrada válida para este programa es cualquier número entero o de coma flotante. Cuando ejecutas el programa, te pide estos valores como entrada y luego dibuja la trayectoria (ver [Figura 2](ch02.html#ch2fig14)-14):

Enter the initial velocity (m/s): 25  
Enter the angle of projection (degrees): 60



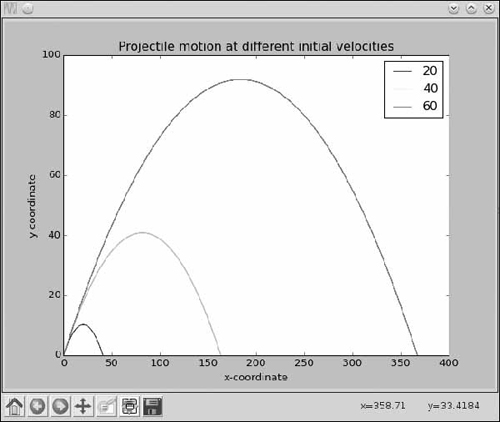
*Figura 2-14: Trayectoria de una pelota lanzada con una velocidad de 25 m/s y un ángulo de 60 grados*

##### **Comparación de la trayectoria con distintas velocidades iniciales**

El programa anterior te permite realizar experimentos interesantes. Por ejemplo, ¿cómo sería la trayectoria de tres pelotas lanzadas a distintas velocidades pero con el mismo ángulo inicial? Para representar gráficamente tres trayectorias a la vez, podemos sustituir el bloque de código main de nuestro programa anterior por el siguiente:

if \_\_name\_\_ == '\_\_main\_\_':  
  
# List of three different initial velocities  
➊ u\_list = [20, 40, 60]  
theta = 45  
for u in u\_list:  
draw\_trajectory(u, theta)  
  
# Add a legend and show the graph  
➋ plt.legend(['20', '40', '60'])  
plt.show()

Aquí, en lugar de pedir al usuario del programa que introduzca la velocidad y el ángulo de proyección, creamos una lista (u\_list) con las velocidades 20, 40 y 60 en ➊ y fijamos el ángulo de proyección en 45 grados (utilizando la etiqueta theta). A continuación, llamamos a la función draw\_trajectory() con cada uno de los tres valores de u\_list utilizando el mismo valor para theta, que calcula la lista de *coordenadas* *x* e *y* y llama a la función draw\_graph(). Cuando llamamos a la función show(), las tres gráficas se muestran en el mismo gráfico. Como ahora tenemos una gráfica con múltiples trazados, añadimos una leyenda a la gráfica en ➋ antes de llamar a show() para mostrar la velocidad de cada línea. Cuando ejecutes el programa anterior, verás la gráfica que se muestra en la [Figura 2-15](ch02.html#ch2fig15).



*Figura 2-15: La trayectoria de una pelota lanzada en un ángulo de 60 grados, con una velocidad de 20, 40 y 60 m/s*

[anterior](ch02_4.html)[Subtema 5 de 7: (Ver todo)](ch02.html)[siguiente](ch02_6.html)