Capítulo 7: Resolución de problemas de cálculo

### **Encontrar el máximo global mediante el ascenso del gradiente**

A veces sólo nos interesa encontrar el máximo global de una función en lugar de todos los máximos y mínimos locales y globales. Por ejemplo, podemos querer descubrir el ángulo de proyección para el que una bola recorrerá la máxima distancia horizontal. Vamos a aprender un nuevo enfoque más práctico para resolver un problema de este tipo. Este enfoque hace uso únicamente de la primera derivada, por lo que sólo es aplicable a funciones para las que pueda calcularse la primera derivada.

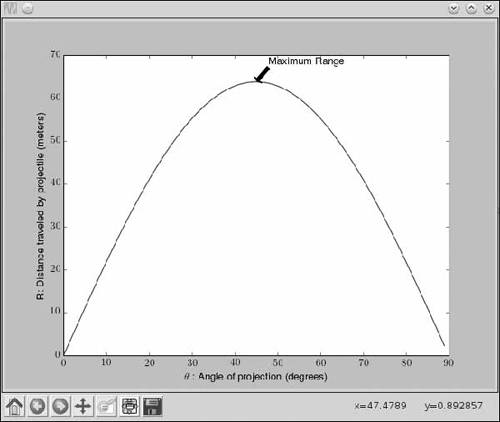
Este método se denomina *método* del gradiente ascendente, que es un enfoque iterativo para encontrar el máximo global. Como el método del ascenso gradiente implica muchos cálculos, es perfecto para resolverlo mediante programación y no a mano. Vamos a probarlo utilizando el problema de ejemplo de encontrar el ángulo de proyección. En el [Capítulo 2](ch02.html#ch02), dedujimos la expresión

image

para calcular el tiempo de vuelo de un cuerpo en movimiento de proyectil que es lanzado con una velocidad *u* a un ángulo *θ*. El *alcance* de un proyectil, *R*, es la distancia horizontal total recorrida por el proyectil y viene dada por el producto de *ux* *×*tflight. Aquí, *ux* es la componente horizontal de la velocidad inicial y es igual a *u* *cosθ*. Sustituyendo las fórmulas de *ux* y tflight, obtenemos la expresión

image

El gráfico de [la Figura 7-4](ch07.html#ch7fig4) muestra valores de *θ* entre 0 y 90 grados y el alcance correspondiente (distancia recorrida) para cada ángulo. En el gráfico podemos ver que el alcance máximo se obtiene cuando el ángulo de proyección es de unos 45 grados. Ahora aprenderemos a utilizar el método del gradiente ascendente para hallar numéricamente este valor de *θ*.



*Figura 7-4: Alcance de un proyectil lanzado con una velocidad inicial de 25 m/s con distintos ángulos de proyección*

El método de ascenso por gradiente es un método iterativo: empezamos con un valor inicial de θ *-digamos*, 0,001, o θold = 0,001- y nos acercamos gradualmente al valor de *θ* que corresponde al alcance máximo[(Figura 7-5](ch07.html#ch7fig5)). El paso que nos acerca es la ecuación

image

donde *λ* es el *tamaño del paso* y

image

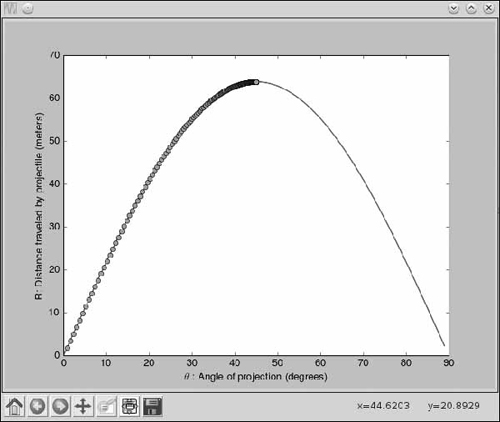
es la derivada de *R* respecto a *θ*. Una vez fijado θold = 0,001, hacemos lo siguiente

1. Calcula θnuevo utilizando la ecuación anterior.

2. Si la diferencia absoluta θnew - θold es mayor que un valor, *ε*, fijamos θold = θnew y volvemos al paso 1. En caso contrario, vamos al paso 3.

3. θnew es un valor aproximado de *θ* para el que *R* tiene el valor máximo.

El valor de *épsilon(ε*) determina cuándo decidimos detener la iteración del algoritmo. Se trata en "[El papel del tamaño del paso y](ch07.html#ch07lev2sec09)épsilon" en [la página 197](ch07.html#page_197).



*Figura 7-5: El método de ascenso gradiente nos lleva iterativamente hacia el punto máximo de la función.*

La siguiente función grad\_ascent() implementa el algoritmo de ascenso gradiente. El parámetro x0 es el valor inicial de la variable en el que comenzar la iteración, f1x es la derivada de la función cuyo máximo queremos encontrar, y x es el objeto Symbol correspondiente a la variable de la función.

'''  
Use gradient ascent to find the angle at which the projectile  
has maximum range for a fixed velocity, 25 m/s  
'''  
  
import math  
from sympy import Derivative, Symbol, sin  
  
def grad\_ascent(x0, f1x, x):  
➊ epsilon = 1e-6  
➋ step\_size = 1e-4  
➌ x\_old = x0  
➍ x\_new = x\_old + step\_size\*f1x.subs({x:x\_old}).evalf()  
➎ while abs(x\_old - x\_new) > epsilon:  
x\_old = x\_new  
x\_new = x\_old + step\_size\*f1x.subs({x:x\_old}).evalf()  
  
return x\_new  
  
➏ def find\_max\_theta(R, theta):  
# Calculate the first derivative  
R1theta = Derivative(R, theta).doit()  
theta0 = 1e-3  
theta\_max = grad\_ascent(theta0, R1theta, theta)  
➐ return theta\_max  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
g = 9.8  
# Assume initial velocity  
u = 25  
# Expression for range  
theta = Symbol('theta')  
➑ R = u\*\*2\*sin(2\*theta)/g  
  
➒ theta\_max = find\_max\_theta(R, theta)  
print('Theta: {0}'.format(math.degrees(theta\_max)))  
print('Maximum Range: {0}'.format(R.subs({theta:theta\_max})))

Establecemos el valor épsilon en 1e-6 y el tamaño del paso en 1e-4 en ➊ y ➋, respectivamente. El valor épsilon debe ser siempre un valor positivo muy pequeño cercano a 0, y el tamaño del paso debe elegirse de forma que la variable se incremente en pequeñas cantidades en cada iteración del algoritmo. La elección del valor de épsilon y del tamaño del paso se trata con un poco más de detalle en "[El papel del tamaño del paso y de épsilon](ch07.html#ch07lev2sec09)" en [la página 197](ch07.html#page_197).

Ajustamos x\_old a x0 en ➌ y calculamos x\_new por primera vez en ➍. Utilizamos el método subs() para sustituir el valor de x\_old en lugar de la variable y luego utilizamos evalf() para calcular el valor numérico. Si la diferencia absoluta abs(x\_old – x\_new) es mayor que epsilon, el bucle while en ➎ sigue ejecutándose, y seguimos actualizando el valor de x\_old y x\_new según los pasos 1 y 2 del algoritmo de ascenso gradiente. Una vez que salimos del bucle -es decir, de abs(x\_old – x\_new) > epsilon- devolvemos x\_new, el valor de la variable correspondiente al valor máximo de la función.

Empezamos a definir la función find\_max\_theta() en ➏. En esta función, calculamos la derivada de primer orden de R; creamos una etiqueta, theta0, y la establecemos en 1e-3; y llamamos a la función grad\_ascent() con estos dos valores como argumentos, además de un tercer argumento, el objeto símbolo theta. Una vez obtenido el valor de *θ* correspondiente al valor máximo de la función (theta\_max), lo devolvemos en ➐.

Por último, creamos la expresión que representa el rango horizontal en ➑, habiendo establecido la velocidad inicial, u = 25, y el objeto símbolo theta correspondiente al ángulo *θ*. A continuación, llamamos a la función find\_max\_theta() con R y theta en ➒.

Cuando ejecutes este programa, deberías ver la siguiente salida:

Theta: 44.99999978475661  
Maximum Range: 63.7755102040816

El valor de *θ* se imprime en grados y resulta ser próximo a 45 grados, como era de esperar. Si cambias la velocidad inicial por otros valores, verás que el ángulo de proyección en el que se alcanza el alcance máximo es siempre próximo a 45 grados.

#### ***Un programa genérico para el ascenso por gradiente***

Podemos modificar ligeramente el programa anterior para hacer un programa genérico de ascenso por gradiente:

'''  
Use gradient ascent to find the maximum value of a  
single-variable function  
'''  
  
from sympy import Derivative, Symbol, sympify  
  
def grad\_ascent(x0, f1x, x):  
epsilon = 1e-6  
step\_size = 1e-4  
x\_old = x0  
x\_new = x\_old + step\_size\*f1x.subs({x:x\_old}).evalf()  
while abs(x\_old - x\_new) > epsilon:  
x\_old = x\_new  
x\_new = x\_old + step\_size\*f1x.subs({x:x\_old}).evalf()  
  
return x\_new  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
f = input('Enter a function in one variable: ')  
var = input('Enter the variable to differentiate with respect to: ')  
var0 = float(input('Enter the initial value of the variable: '))  
try:  
f = sympify(f)  
except SympifyError:  
print('Invalid function entered')  
else:  
➊ var = Symbol(var)  
➋ d = Derivative(f, var).doit()  
➌ var\_max = grad\_ascent(var0, d, var)  
print('{0}: {1}'.format(var.name, var\_max))  
print('Maximum value: {0}'.format(f.subs({var:var\_max})))

La función grad\_ascent() sigue siendo la misma. Ahora, sin embargo, el programa pide al usuario que introduzca la función, la variable de la función y el valor inicial de la variable, donde comenzará el ascenso gradiente. Una vez que estamos seguros de que SymPy puede reconocer la entrada del usuario, creamos un objeto Símbolo correspondiente a la variable en ➊, hallamos la primera derivada con respecto a ella en ➋, y llamamos a la función grad\_ascent() con estos tres argumentos. El valor máximo se obtiene en ➌.

Aquí tienes un ejemplo de ejecución:

Enter a function in one variable: 25\*25\*sin(2\*theta)/9.8  
Enter the variable to differentiate with respect to: theta  
Enter the initial value of the variable: 0.001  
theta: 0.785360029379083  
Maximum value: 63.7755100185965

La entrada de la función es la misma que en nuestra primera implementación del gradiente ascendente, y el valor de *θ* se imprime en radianes.

Aquí tienes otra ejecución del programa, que encontrará el valor máximo de *cosy*:

Enter a function in one variable: cos(y)  
Enter the variable to differentiate with respect to: y  
Enter the initial value of the variable: 0.01  
y: 0.00999900001666658  
Maximum value: 0.999950010415832

El programa también funciona correctamente para una función como cos(y) + k, donde k es una constante:

Enter a function in one variable: cos(y) + k  
Enter the variable to differentiate with respect to: y  
Enter the initial value of the variable: 0.01  
y: 0.00999900001666658  
Maximum value: k + 0.999950010415832

Sin embargo, una función como cos(ky) no funcionará porque su derivada de primer orden, kcos(ky), aún contiene k, y SymPy no sabe nada sobre su valor. Por lo tanto, SymPy no puede realizar un paso clave en el algoritmo de ascenso gradiente, a saber, la comparación abs(x\_old - x\_new) > epsilon.

#### ***Advertencia sobre el valor inicial***

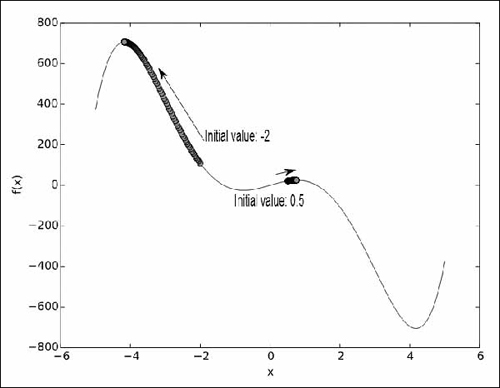
El valor inicial de la variable a partir de la cual iniciamos la iteración del método de ascenso gradiente desempeña un papel muy importante en el algoritmo. Consideremos la función x5 - 30x3 + *50x*, que hemos utilizado como ejemplo en [la Figura 7-3](ch07.html#ch7fig3). Vamos a encontrar el máximo utilizando nuestro programa genérico de ascenso por gradiente:

Enter a function in one variable: x\*\*5 - 30\*x\*\*3 + 50\*x  
Enter the variable to differentiate with respect to: x  
Enter the initial value of the variable: -2  
x: -4.17445116397103  
Maximum value: 705.959460322318

El algoritmo de ascenso por gradiente se detiene cuando encuentra el *pico más cercano*, que no siempre es el máximo global. En este ejemplo, cuando parte del valor inicial -2, se detiene en el pico que también corresponde al máximo global (aproximadamente 706) en el dominio considerado. Para comprobarlo mejor, probemos con otro valor inicial:

Enter a function in one variable: x\*\*5 - 30\*x\*\*3 + 50\*x  
Enter the variable to differentiate with respect to: x  
Enter the initial value of the variable: 0.5  
x: 0.757452532565767  
Maximum value: 25.0846622605419

En este caso, el pico más cercano en el que se detiene el algoritmo de ascenso gradiente no es el verdadero máximo global de la función. [La figura 7-6](ch07.html#ch7fig6) muestra el resultado del algoritmo de ascenso por gradiente en ambos casos.



*Figura 7-6: Resultados del algoritmo de ascenso por gradiente con distintos valores iniciales. El ascenso por gradiente siempre nos lleva al pico más cercano.*

Por tanto, al utilizar este método, el valor inicial debe elegirse con cuidado. Algunas variaciones del algoritmo intentan resolver esta limitación.

#### ***El papel del tamaño del paso y el épsilon***

En el algoritmo de ascenso gradiente, el siguiente valor de la variable se calcula mediante la ecuación

image

donde *λ* es el tamaño del *paso*. El tamaño del paso determina la distancia del siguiente paso. Debe ser pequeño para evitar *pasar por encima de* un pico. Es decir, si el valor actual de *x* está cerca del valor que corresponde al valor máximo de la función, el siguiente paso no debería ir más allá del pico. En ese caso, el algoritmo no tendrá éxito. Por otra parte, los valores muy pequeños tardarán más en calcularse. Hemos utilizado un tamaño de paso fijo de 10-3, pero puede que éste *no* sea el valor más adecuado para todas las funciones.

El valor de épsilon*(ε*) que determina cuándo decidimos detener la iteración del algoritmo debe ser un valor lo suficientemente pequeño como para que estemos convencidos de que el valor de *x* no está cambiando. Esperamos que la primera derivada, *f′(x*), sea 0 en el punto máximo, y lo ideal es que la diferencia absoluta |θnew - θold| sea 0 (véase el paso 2 del algoritmo de ascenso gradiente en [la página 192](ch07.html#page_192)). Sin embargo, debido a imprecisiones numéricas, puede que no obtengamos exactamente una diferencia de 0; por ello, el valor de épsilon se elige para que sea un valor cercano a 0, que, a efectos prácticos, nos diría que el valor de *x* ya no está cambiando. He utilizado 10-6 como épsilon para todas las funciones. Este valor, aunque suficientemente pequeño y adecuado para las funciones que tienen solución para *f′(x*) = 0, como sin(x), puede no ser el valor adecuado para otras funciones. Por lo tanto, es una buena idea verificar el valor máximo al final para asegurarse de que es correcto y, si es necesario, ajustar el valor de epsilon en consecuencia.

El paso 2 del algoritmo de ascenso gradiente también implica que, para que el algoritmo termine, la ecuación *f′(x*) = 0 debe tener solución, lo que no ocurre con una función como *ex* o log*(x*). Por tanto, si proporcionas una de estas funciones como entrada al programa anterior, el programa no te dará una solución y seguirá ejecutándose. Podemos hacer que el programa de ascenso gradiente sea más útil para estos casos incorporando una comprobación para saber si *f′(x*) = 0 tiene solución. Aquí tienes el programa modificado:

'''  
Use gradient ascent to find the maximum value of a  
single-variable function. This also checks for the existence  
of a solution for the equation f'(x)=0.  
'''  
  
from sympy import Derivative, Symbol, sympify, solve  
  
def grad\_ascent(x0, f1x, x):  
# Check if f1x=0 has a solution  
➊ if not solve(f1x):  
print('Cannot continue, solution for {0}=0 does not exist'.format(f1x))  
return  
epsilon = 1e-6  
step\_size = 1e-4  
x\_old = x0  
x\_new = x\_old + step\_size\*f1x.subs({x:x\_old}).evalf()  
while abs(x\_old - x\_new) > epsilon:  
x\_old = x\_new  
x\_new = x\_old + step\_size\*f1x.subs({x:x\_old}).evalf()  
  
return x\_new  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
f = input('Enter a function in one variable: ')  
var = input('Enter the variable to differentiate with respect to: ')  
var0 = float(input('Enter the initial value of the variable: '))  
try:  
f = sympify(f)  
except SympifyError:  
print('Invalid function entered')  
else:  
var = Symbol(var)  
d = Derivative(f, var).doit()  
var\_max = grad\_ascent(var0, d, var)  
➋ if var\_max:  
print('{0}: {1}'.format(var.name, var\_max))  
print('Maximum value: {0}'.format(f.subs({var:var\_max})))

En esta modificación de la función grad\_ascent(), llamamos a la función solve() de SymPy en ➊ para determinar si la ecuación *f′*(*x*) = 0, aquí f1x, tiene solución. Si no es así, imprimimos un mensaje y volvemos. Otra modificación aparece en el bloque \_\_main\_\_ en ➋. Comprobamos si la función grad\_ascent() ha devuelto correctamente un resultado; si es así, procedemos a imprimir el valor máximo de la función y el valor correspondiente de la variable.

Estos cambios permiten al programa manejar funciones como log*(x*) y *ex*:

Enter a function in one variable: log(x)  
Enter the variable to differentiate with respect to: x  
Enter the initial value of the variable: 0.1  
Cannot continue, solution for 1/x=0 does not exist

Verás lo mismo para *ex*.

**ALGORITMO DE DESCENSO DE GRADIENTE**

El algoritmo inverso del algoritmo de ascenso por gradiente es el algoritmo de *descenso* por gradiente, que es un método para encontrar el valor mínimo de una función. Es similar al algoritmo de ascenso por gradiente, pero en lugar de "subir" por la función, "bajamos". El Desafío nº 2 de la [página 205](ch07.html#page_205) analiza la diferencia entre estos dos algoritmos y te da la oportunidad de poner en práctica el inverso.

[anterior](ch07_6.html)[Subtema 7 de 11: (Ver todo)](ch07.html)[siguiente](ch07_8.html)