06 - Bisection Search

Newton-Raphson

El algoritmo de aproximación más utilizado suele atribuirse a Isaac Newton. Normalmente se denomina método de Newton, pero a veces se conoce como método de Newton-Raphson. Se puede utilizar para encontrar las raíces reales de muchas funciones, pero sólo lo veremos en el contexto de la búsqueda de las raíces reales de un polinomio con una variable. La generalización a polinomios con múltiples variables es sencilla tanto matemática como algorítmicamente.

Un polinomio con una variable (por convención, escribimos la variable como x) es o bien 0 o la suma de un número finito de términos distintos de cero, por ejemplo, $3x^2 + 2x + 3$. Cada término, por ejemplo, $2x + 3$, es la suma de un número finito de términos distintos de cero. Cada término, por ejemplo, $3x^2$, consiste en una constante (el coeficiente del término, 3 en este caso) multiplicada por la variable (x en este caso) elevada a un exponente entero no negativo (2 en este caso). El exponente de un término se denomina grado de ese término. El grado de un polinomio es el mayor grado de cualquier término. Algunos ejemplos son 3 (grado 0), $2,5x + 12$ (grado 1) y $3x^2$ (grado 2).

Si p es un polinomio y r un número real, escribiremos p(r) para representar el valor del polinomio cuando x = r. Una raíz del polinomio p es una solución de la ecuación p = 0, es decir, un r tal que p(r) = 0. Así, por ejemplo, el problema de encontrar una aproximación a la raíz cuadrada de 24 puede formularse como encontrar un x tal que $x^2 - 24$ se aproxime a 0.

Newton demostró un teorema que implica que si un valor, llamémoslo conjetura, es una aproximación a una raíz de un polinomio, entonces $conjetura - p(conjetura)/p'(conjetura)$, donde p' es la primera derivada de p, es una aproximación mejor que conjetura.

La primera derivada de una función f(x) expresa cómo cambia el valor de f(x) con respecto a los cambios en x. Por ejemplo, la primera derivada de una constante es 0, porque el valor de una constante no cambia. Para cualquier término $c * x^p$, la primera derivada de ese término es $c * p * x^{p-1}$. Así, la primera derivada de un polinomio de la forma

$$c1 * x^p + c2 * x^{p-1} + \ldots + cx + k$$

es

$$c1 * p * x^{p-1} + c2 * (p-1)x^{p-2} + \ldots + c$$

Para encontrar la raíz cuadrada de un número, digamos k, tenemos que encontrar un valor x tal que $x^2 - k = 0$. La primera derivada de este polinomio es simplemente $2x$. Por lo tanto, sabemos que podemos mejorar la conjetura actual eligiendo como siguiente $conjetura - (conjetura2 - k)/2 * conjetura$. El siguiente código ilustra cómo utilizar este método para encontrar rápidamente una aproximación a la raíz cuadrada.

```
# Newton Raphson para raiz cuadrada
# Busca x tal que x ** 2 - 24 este entre epsilon 0
epsilon = 0.01
guess = k / 2

while abs(guess ** 2 - k) >= epsilon:
    guess = guess - (((guess ** 2) - k) / (2 * guess))

print("La raiz cuadrada de", k, "es cerca a", guess)
```

Ejercicio manual

Añade algún código a la implementación de Newton-Raphson que lleve la cuenta del número de iteraciones utilizadas para encontrar la raíz. Usa ese código como parte de un programa que compare la eficiencia de Newton-Raphson y la búsqueda por bisección. (Deberías descubrir que Newton-Raphson es mucho más eficiente).

Mi aproximación al ejercicio:

Con el método Newton-Raphson

```
k = int(input("Ingresa un numero"))
epsilon = 0.01
guess = k / 2.0
num_iter = 0

while abs(guess ** 2 - k) >= epsilon:
    guess = guess - ((guess ** 2 - k) / (2 * guess))
    num_iter += 1

print(guess, num_iter)
```

Con la búsqueda binaria

```
k = int(input("Ingresa un numero"))
```

```python
epsilon = 0.01
low = 0.0
high = max(1.0, k)
guess = (high + low) / 2
num_iter = 0

while abs(guess ** 2 - k) >= epsilon:
    if guess ** 2 < k:
        low = guess
    else:
        high = guess
    guess = (high + low) / 2.0
    num_iter += 1

print(guess, num_iter)
```

Para comparar la eficiencia utilizo un `k = 24` lo cual me da como resultado:

```
[Newton-Raphson]  Raíz de 24 ≈ 4.898..., Iteraciones: 5
[Bisección]       Raíz de 24 ≈ 4.898..., Iteraciones: 13
```

Ejercicio manual de la lectura

Suponga que se le da un número entero 0 ≤ N ≤ 1000. Escribe un trozo de código Python que utilice la búsqueda por bisección para adivinar N. El código imprime dos líneas:

- `count`: con cuántas conjeturas se necesitaron para encontrar N, y
- `answer`: con el valor de N.

Pistas: Si el valor medio está exactamente entre dos enteros, elige el más pequeño.

Mi aproximación

```python
N = int(input("Ingresa un entero"))

low = 0
high = 1000
count = 0

while True:
    guess = (high + low) // 2
    count += 1

    if guess == N:
```

```
            print("count:", count)
            print("answer:", guess)
            break
        elif guess < N:
            low = guess + 1
        else:
            high = guess - 1
```

En cada intento se compara con el valor medio del rango actual. Si el valor medio está justo entre dos números (por ejemplo, `(5 + 6) / 2 = 5.5`), usamos `// 2` para quedarnos con el **menor**.
