Método de Newton-Raphson

El método de Newton (o de Newton-Raphson) es un algoritmo de búsqueda de las raíces (o ceros) de una función de valor real, el cual produce aproximaciones sucesivas, cada vez mejores a dichas raíces.

La versión simple del método se aplica a una función f sobre una variable real x, la derivada f' de la función y una estimación inicial x_0 para una raíz de la función. A partir de dichas hipótesis, si estimación inicial es suficientemente buena, i.e., cercana a la raíz, entonces el valor x_1 definido por la siguiente expresión:

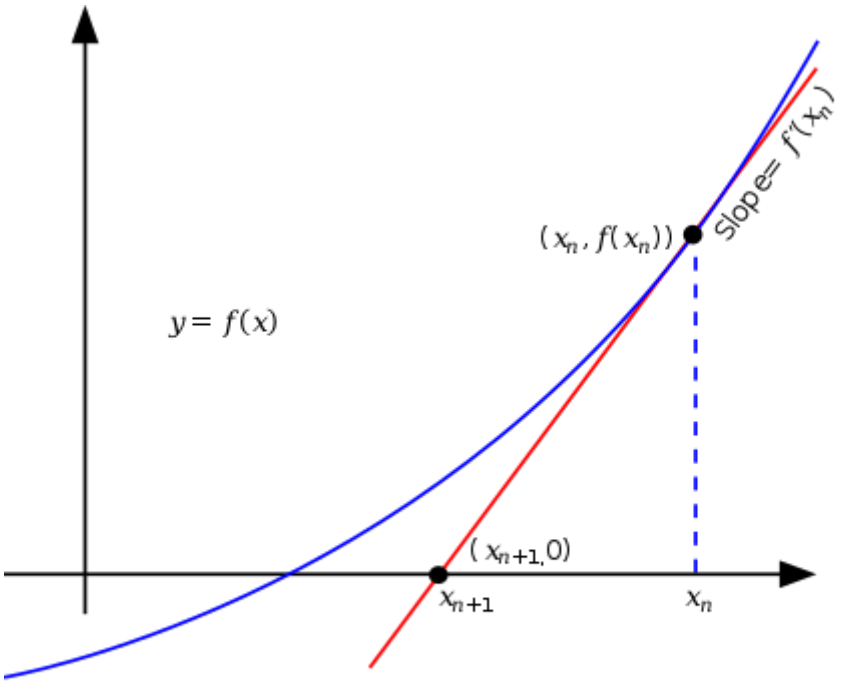
$$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

es una mejor aproximación de la raíz que x_0.

Geométricamente, $(x_1, 0)$ es la intersección del eje x con la tangente del gráfico de f en el punto $(x_0, f(x_0))$, por lo que, si consideramos la recta tangente de f en dicho punto como una aproximación de la función f, entonces x_1 es la raíz de dicha recta y, por tanto, de la aproximación de f.

El proceso se repite hasta alcanzar un valor suficientemente preciso. El número de dígitos correctos se duplica aproximadamente con cada paso.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



El proceso comienza con un valor inicial arbitrario x_0, cuanto más cercano a la raíz, mejor. Pero, en ausencia de cualquier estimación de la raíz, aplicando un método de "adivinar y verificar" puede reducir las posibilidades a un intervalo razonablemente pequeño. El método generalmente converge, siempre que esta suposición inicial sea lo suficientemente cercana al cero desconocido y que $f'(x_0) \neq 0$

Se asume que la función tiene un "buen comportamiento", es decir, que

- está definida para todos los valores del intervalo que vamos a tratar,
- a cada punto del intervalo, la función tiene un solo valor,
- es continua y
- sus derivadas son definidas y continuas.

Se puede demostrar que el método de Newton-Raphson tiene convergencia cuadrática: si α es una raíz, entonces:

$$|x_{k+1} - \alpha| \leq C|x_k - \alpha|^2$$

para una cierta constante C. Esto significa que si, en algún momento el error es menor o igual a 0.1, a cada nueva iteración el número de decimales exactos se duplica aproximadamente. En la práctica puede servir para hacer una estimación aproximada del error:

El error relativo entre dos aproximaciones sucesivas viene dado por:

$$E = \frac{|x_{k+1} - x_k|}{|x_{k+1}|}$$

Con lo cual se toma el error relativo como si la última aproximación fuera el valor exacto. Se detiene el proceso iterativo cuando este error relativo es aproximadamente menor que una cantidad fijada previamente.

El método de Newton es una técnica poderosa, ya que su convergencia es cuadrática, i.e., el número de dígitos precisos se duplica aproximadamente en cada paso. Sin embargo, el método requiere que la derivada se pueda calcular directamente de forma analítica, la cual puede no ser fácil de obtener o ser costosa de evaluar.

En estas situaciones, puede ser apropiado aproximar la derivada usando la pendiente de una línea que pasa por dos puntos cercanos en la función. Esta aproximación se corresponde al método de la secante, cuya convergencia es más lenta que la del método de Newton.

Consideremos el problema de encontrar un número positivo x tal que $\cos(x) = x^3$, para lo cual podemos transformarlo en el problema de encontrar el cero de $f(x) = \cos(x) - x^3$.

Sabemos que $f'(x) = -\operatorname{sen}(x) - 3x^2$. Ya que $cos(x) \leq 1$ para todo x y $x^3 > 1$ para $x > 1$, luego podemos deducir que el cero se encuentra entre 0 y 1.

Comenzaremos probando con el valor inicial $x_0 = 0.5$

In [3]:

```
from math import cos, sin

def f(x):
    return cos(x) - x**3

def f1(x):
    return -sin(x)-3*x**2

x0 = 0.5
x1 = x0 - (f(x0)/f1(x0))
x1
```

Out[3]:

```
1.1121416370972725
```

In [4]:

```
x2 = x1 - (f(x1)/f1(x1))
x2
```

Out[4]:

```
0.9096726937368068
```

In [5]:

```
x3 = x2 - (f(x2)/f1(x2))
x3
```

Out[5]:

```
0.8672638182088165
```

In [6]:

```
x4 = x3 - (f(x3)/f1(x3))
x4
```

Out[6]:

```
0.8654771352982646
```

In [7]:

```
x5 = x4 - (f(x4)/f1(x4))
x5
```

Out[7]:

```
0.8654740331109566
```

Como podemos ver, los 5 primeros dígitos decimales comienzan a repetirse.

Vamos a utilizar un segmento de código para calcular los primeros N valores.

In [9]:

```
# Observe que el valor x0 asignado previamente a 0.5 no lo habíamos cambiado
n = 10
for i in range(n):
    print('i, x(i)', i, x0)
    x1 = x0 - (f(x0)/f1(x0))
    x0 = x1
```

```
i, x(i) 0 0.5
i, x(i) 1 1.1121416370972725
i, x(i) 2 0.9096726937368068
i, x(i) 3 0.8672638182088165
i, x(i) 4 0.8654771352982646
i, x(i) 5 0.8654740331109566
i, x(i) 6 0.8654740331016144
i, x(i) 7 0.8654740331016144
i, x(i) 8 0.8654740331016144
i, x(i) 9 0.8654740331016144
```

In [12]:

```
# Hagamos la impresión más estética
# Observe que el valor x0 asignado previamente a 0.5 no lo habíamos cambiado
n = 10
print('Iteración\tAproximación')
for i in range(n):
    print('\t', i, '\t', x0)
    x1 = x0 - (f(x0)/f1(x0))
    x0 = x1
```

```
Iteración       Aproximación
         0       0.8654740331016144
         1       0.8654740331016144
         2       0.8654740331016144
         3       0.8654740331016144
         4       0.8654740331016144
         5       0.8654740331016144
         6       0.8654740331016144
         7       0.8654740331016144
         8       0.8654740331016144
         9       0.8654740331016144
```

In [13]:

```
# El código debe ser general. ¿qué pasaría si la derivada se anula?
# Nos daría un error al dividir por cero
# ¿por qué da cero la derivada? porque es un punto estacionario

n = 10
print('Iteración\tAproximación')
for i in range(n):
    print('\t', i, '\t', x0)
    y1 = f1(x0)
    if y1 == 0:
        # interrumpir el ciclo/bucle
        break
    x1 = x0 - (f(x0)/y1)
    x0 = x1
```

```
Iteración       Aproximación
         0       0.8654740331016144
         1       0.8654740331016144
         2       0.8654740331016144
         3       0.8654740331016144
         4       0.8654740331016144
         5       0.8654740331016144
         6       0.8654740331016144
         7       0.8654740331016144
         8       0.8654740331016144
         9       0.8654740331016144
```

En general, no es conveniente preguntar si el valor es igual a cero debido al rango limitado de valores posibles del ordenador. Por ello, resulta más conveniente comparar con valor que consideremos suficientemente pequeño. En nuestro caso, pudieramos tomar 10^{-6} por ejemplo.

¿Cuál es el mínimo valor positivo que acepta Python? Se obtiene mediante la librería del sistema (sys) de la siguiente forma:

In [15]:

```
import sys

sys.float_info.min
```

Out[15]:

```
2.2250738585072014e-308
```

In [16]:

```
# De esta manera, obtenemos todas las especificaciones de los valores flotantes

sys.float_info
```

Out[16]:

```
sys.float_info(max=1.7976931348623157e+308, max_exp=1024, max_10_exp=308,
min=2.2250738585072014e-308, min_exp=-1021, min_10_exp=-307, dig=15, mant_
dig=53, epsilon=2.220446049250313e-16, radix=2, rounds=1)
```

Si la primera derivada no se comporta bien en la vecindad de una raíz en particular, el método puede divergir de esa raíz. Un ejemplo de una función con una raíz, para la cual la derivada no se comporta bien en la vecindad de la raíz, es:

$$f(x) = |x|^a, \quad 0 < a < \frac{1}{2}$$

Si se encuentra un punto estacionario de la función, la derivada es cero y el método terminará debido a división por cero.

Bibliografía

- Método de Newton (https://es.wikipedia.org/wiki/M%C3%A9todo_de_Newton)
- Newton's method (https://en.wikipedia.org/wiki/Newton%27s_method)
- Stationary point (https://en.wikipedia.org/wiki/Stationary_point)
- The Newton-Raphson Method (https://personal.math.ubc.ca/~anstee/math104/newtonmethod.pdf)
- Lecture 1: Numerical Solution of Non-linear equations (https://homepages.see.leeds.ac.uk/~eargah/EARS1160/Lecture1.PDF)
- What exactly does it mean for a function to be "well-behaved"? (https://math.stackexchange.com/questions/206/what-exactly-does-it-mean-for-a-function-to-be-well-behaved)
- Differentiable function (https://en.wikipedia.org/wiki/Differentiable_function#continuously_differentiable)
- Newton's Method (https://patrickwalls.github.io/mathematicalpython/root-finding/newton/)
- Program for Newton Raphson Method (https://www.geeksforgeeks.org/program-for-newton-raphson-method/)
- Python Program Newton Raphson (NR) Method (with Output) (https://www.codesansar.com/numerical-methods/newton-raphson-method-python-program.htm)