Cálculo de raíces de ecuaciones de una variable

Bisección. Punto fijo. Newton-Raphson.

Manuel Carlevaro

Departamento de Ingeniería Mecánica

Grupo de Materiales Granulares - UTN FRLP

manuel.carlevaro@gmail.com

Cálculo Avanzado • 2025

Problema:

$$f(x) = 0$$

$$f(x) = g(x) \implies h(x) = f(x) - g(x) = 0$$

Teorema : Valores intermedios.

Sea $f : [a, b] \to \mathbb{R}$ *una función* ***continua*** *en* $[a, b]$ *tal que* $f(a) < f(b)$*. Entonces:* $\forall u \in (f(a), f(b))$ *existe* $c \in [a, b]$ *tal que* $f(c) = u$*.*

Problema:

$$f(x) = 0$$
$$f(x) = g(x) \implies h(x) = f(x) - g(x) = 0$$

Teorema : Valores intermedios.

Sea $f : [a, b] \to \mathbb{R}$ *una función* ***continua*** *en* $[a, b]$ *tal que* $f(a) < f(b)$*. Entonces:* $\forall u \in (f(a), f(b))$ *existe* $c \in [a, b]$ *tal que* $f(c) = u$*.*

Estrategia general:

- Mostrar que existe al menos una solución (x^*)
- Aislar una raíz: $D \subset \mathbb{R}$, $x^* \in D$ y $f(x) \neq 0 \; \forall x \in D \setminus \{x^*\}$
- Iterar

Problema:

$$f(x) = 0$$
$$f(x) = g(x) \implies h(x) = f(x) - g(x) = 0$$

Teorema : Valores intermedios.

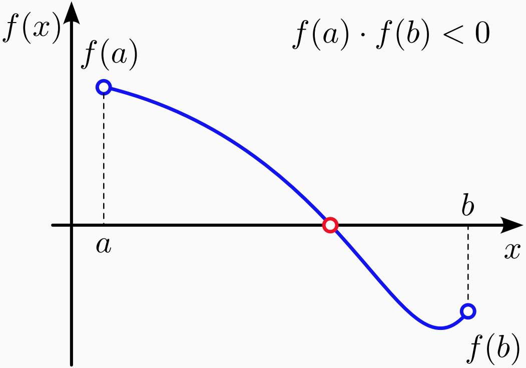
*Sea $f : [a, b] \to \mathbb{R}$ una función **continua** en $[a, b]$ tal que $f(a) < f(b)$. Entonces: $\forall u \in (f(a), f(b))$ existe $c \in [a, b]$ tal que $f(c) = u$.*

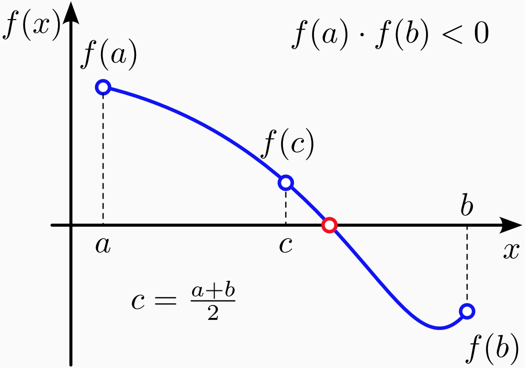
Estrategia general:

- Mostrar que existe al menos una solución (x^*)
- Aislar una raíz: $D \subset \mathbb{R}$, $x^* \in D$ y $f(x) \neq 0 \; \forall x \in D \setminus \{x^*\}$
- Iterar

Métodos:

- *Bracketing* (cerrados)
  - Bisección
  - Posición falsa (*regula falsi*)
  - ITP
- Interpolación
  - Secante
  - Muller
- Iterativos (abiertos)
  - Punto fijo
  - Newton-Raphson
  - Secante
  - Broyden
- Combinación de cerrados y abiertos
  - Brent
  - Ridders

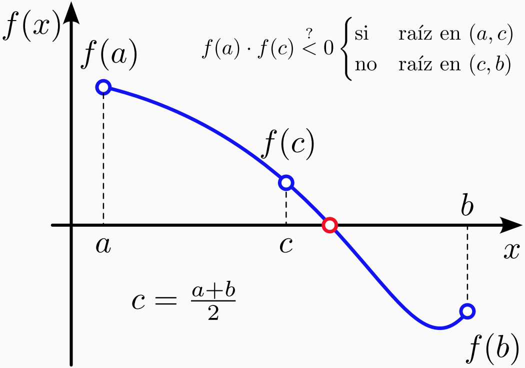

$f(x)$
$f(a)$
$f(a) \cdot f(b) < 0$
b
a
x
$f(b)$



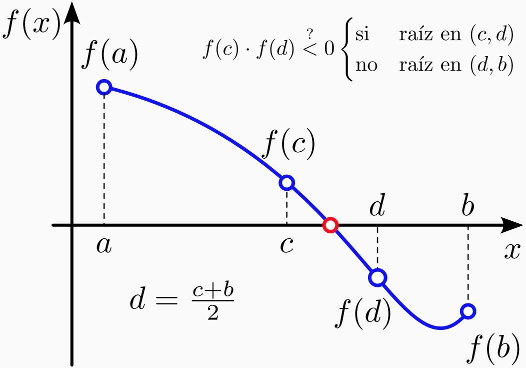
$$f(a) \cdot f(b) < 0$$

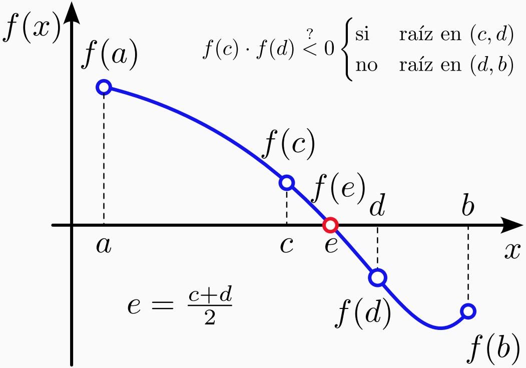
$$c = \frac{a+b}{2}$$

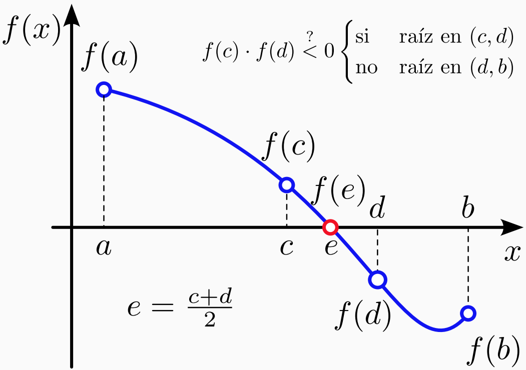
Método de bisección



Método de bisección




$f(x)$
$f(a)$
$f(c) \cdot f(d) \overset{?}{<} 0 \begin{cases} \text{si} & \text{raíz en } (c,d) \\ \text{no} & \text{raíz en } (d,b) \end{cases}$
$f(c)$
$f(e)$
d
b
a
c
e
x
$e = \frac{c+d}{2}$
$f(d)$
$f(b)$



Teorema : Convergencia y error .

Sea $[a,b]$ *el intervalo inicial, donde* $f(a) \cdot f(b) < 0$.
Defina una raíz aproximada como

$$x_n = \frac{a_{n-1} + b_{n-1}}{2}$$

Entonces existe una raíz x^* *tal que*

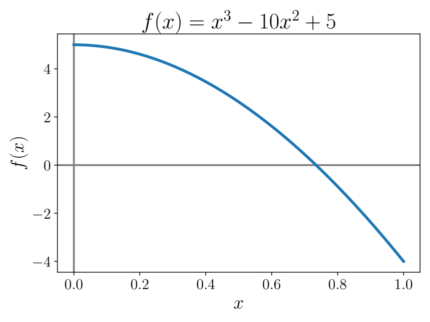
$$|x^* - x_n| \leq \frac{b-a}{2^n}$$

Además, para obtener una precisión de $|x^* - x_n| \leq \epsilon$ *es necesario iterar* n *veces con*

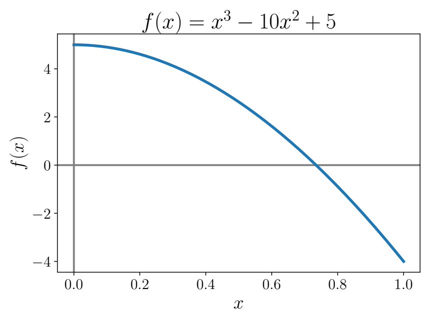
$$n \geq \frac{\log(b-a) - \log \epsilon}{\log 2}$$

Código

```python
#!/usr/bin/env python3

def biseccion(f, x1, x2, switch=0, eps=1.0e-9):
    """
    raiz = biseccion(f, x1, x2, switch=0, tol=1.0e-9)
    Encuentra una raíz de f(x) = 0 por el método de
    bisección. La raíz debe estar acotada en (x1, x2).
    Haciendo switch = 1 devuelve raiz = None
    si f(x) aumenta como resultado de la bisección.
    """
    from math import log, ceil
    f1 = f(x1)
    if abs(f1) < eps: return x1
    f2 = f(x2)
    if abs(f2) < eps: return x2
    if f1 * f2 > 0:
        print("Error: la raíz no está acotada.")
        quit()
    n = ceil(log(abs(x2 - x1)/eps) / log(2.0))
    for i in range(n):
        x3 = 0.5 * (x1 + x2)
        f3 = f(x3)
        if switch and (abs(f3) > abs(f1)) \
                and (abs(f3) > abs(f2)):
            return None
        if abs(f3) < eps: return x3
        if f2 * f3 < 0.0:
            x1 = x3
            f1 = f3
        else:
            x2 = x3
            f2 = f3
    return (x1 + x2) / 2.0

def f(x):
    return x**3 - 10 * x**2 + 5

raiz = biseccion(f, 0.0, 1.0)
print(f"La raiz de f(x) = 0 es {raiz}")
```




```
$ ./biseccion.py
La raiz de f(x) = 0 es 0.7346035079099238
```




```
$ ./biseccion.py
La raiz de f(x) = 0 es 0.7346035079099238
```

SciPy: optimize

```python
#!/usr/bin/env python3

from scipy import optimize

def f(x):
    return x**3 - 10 * x**2 + 5

raiz = optimize.bisect(f, 0.0, 1.0)
print(f"La raiz de f(x) = 0 es {raiz}")
```

```
$ ./scipy-biseccion.py
La raiz de f(x) = 0 es 0.7346035077880515
```

Punto fijo

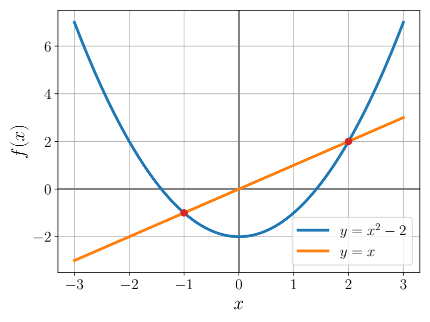
p es un **punto fijo** de g si $g(p) = p$.

Si g tiene un punto fijo en p, entonces:

$$f(x) = x - g(x)$$

tiene un cero en p.

Ejemplo: $g(x) = x^2 - 2 \rightarrow p = \{-1, 2\}$.

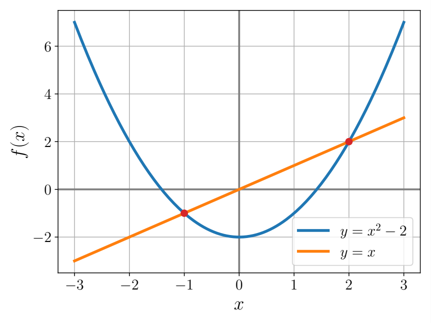


Punto fijo

p es un **punto fijo** de g si $g(p) = p$.
Si g tiene un punto fijo en p, entonces:

$$f(x) = x - g(x)$$

tiene un cero en p.
Ejemplo: $g(x) = x^2 - 2 \to p = \{-1, 2\}$.



Teorema : Existencia y unicidad.

I) *Si $g \in \mathcal{C}[a,b]$ y $g(x) \in [a,b] \, \forall x \in [a,b]$, entonces g tiene por lo menos un punto fijo en $[a,b]$.*

II) *Si, además, $g'(x)$ existe en $[a,b]$ y hay una constante positiva $k < 1$ con*

$$|g'(x)| \leq k, \ \forall x \in [a,b]$$

entonces existe exactamente un punto fijo en $[a,b]$.

Iteración de punto fijo:

- Aproximación inicial p_0
- $\{p_n\}_{n=0}^{\infty}$ con $p_n = g(p_{n-1}), \ n \geq 1$
- Si la sucesión converge a p y g es continua:

$$p = \lim_{n\to\infty} p_n = \lim_{n\to\infty} g(p_{n-1}) = g\left(\lim_{n\to\infty} p_{n-1}\right) = g(p)$$

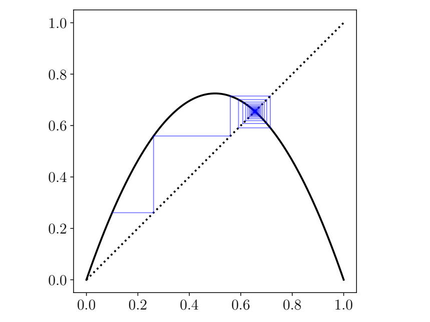
Código

```python
#!/usr/bin/env python3

def punto_fijo(g, x0, max_n=200, eps=1.0e-5):
    """
    p = punto_fijo(g, x0, max_n=200, eps=1.0e-5)
    Itera la relación x = g(x0), x0 <- x
    hasta agotar el número de iteraciones (en ese
    caso devuelve None), o hasta que se produzca
    convergencia con la tolerancia eps, en la que
    devuelve el punto fijo.
    """
    for k in range(max_n):
        x = g(x0)
        delta_x = x - x0
        print(f"{k:3d} {x:1.16f} {delta_x:1.16f}")
        if abs(delta_x / x) < eps:
            break
        x0 = x
    else:
        x = None
    return x
```

```python
def g(x):
    """
    Mapeo logístico x_{n+1} = r x_n (1 - x_n)
    """
    r = 2.9  # Un punto fijo
    # r = 3.1  # Atractor de dos puntos
    # r = 4.0  # Región caótica
    return r * x * (1 - x)

pfijo = punto_fijo(g, 0.1)
print(f"La raiz de g(x) = 0 es {pfijo}")
```

```
$ ./pfijo.py
0 0.2610000000000000 0.1610000000000000
1 0.5593491000000000 0.2983491000000000
2 0.7147852845546510 0.1554361845546509
... ...
94 0.6551757611798358 0.0000070667498989
95 0.6551694011125498 -0.0000063600672859
La raiz de g(x) = 0 es 0.6551694011125498
```

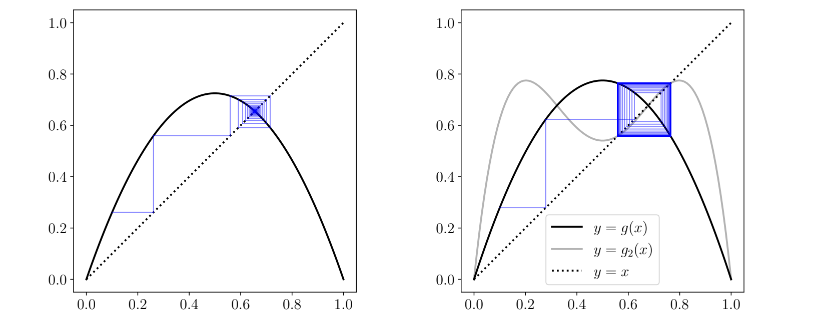
Gráfico *coweb*



$$r = 2.9$$

$$p = g(p) = \frac{19}{29}$$

$$g'(p) = -\frac{9}{10}$$



$$r = 2.9$$

$$p = g(p) = \frac{19}{29}$$

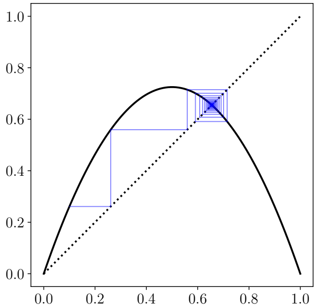
$$g'(p) = -\frac{9}{10}$$

$$r = 3.1$$

$$p = g(p) = \frac{21}{31}$$

$$g'(p) = -\frac{11}{10}$$

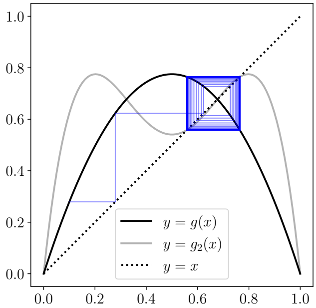
Gráfico *coweb*



$$r = 2.9$$

$$p = g(p) = \frac{19}{29}$$

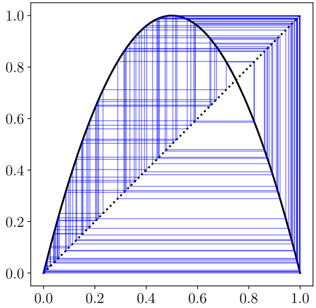
$$g'(p) = -\frac{9}{10}$$



$$r = 3.1$$

$$p = g(p) = \frac{21}{31}$$

$$g'(p) = -\frac{11}{10}$$

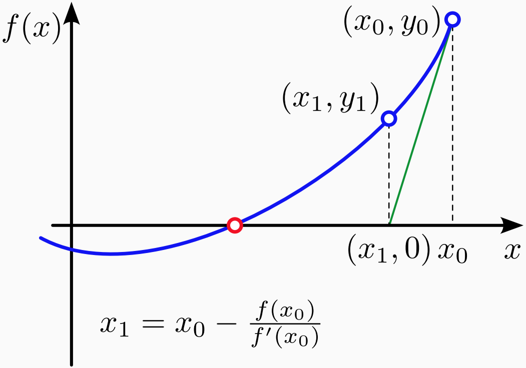


$$r = 4.0$$

$$p = g(p) = \frac{3}{4}$$

$$g'(p) = -2$$

Método de Newton-Raphson

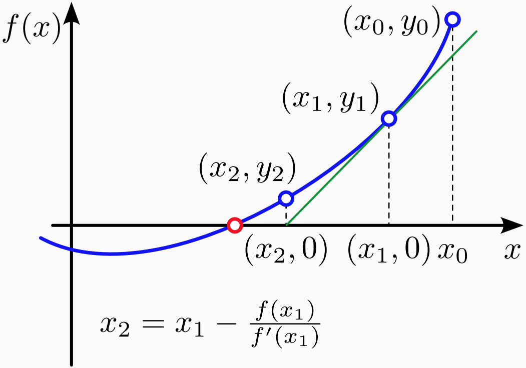


$$\frac{y - y_0}{x - x_0} = f'(x_0) \Rightarrow y = f(x_0) + f'(x_0)(x - x_0)$$

Haciendo $y = 0$:

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \mapsto x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

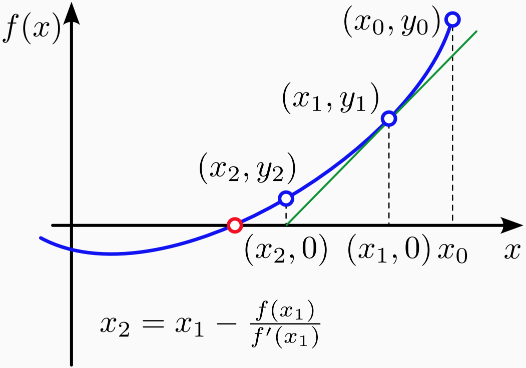
Método de Newton-Raphson



$$\frac{y - y_0}{x - x_0} = f'(x_0) \Rightarrow y = f(x_0) + f'(x_0)(x - x_0)$$

Haciendo $y = 0$:

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \mapsto x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$



$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)}$$

$$\frac{y - y_0}{x - x_0} = f'(x_0) \Rightarrow y = f(x_0) + f'(x_0)(x - x_0)$$

Haciendo $y = 0$:

$$x = x_0 - \frac{f(x_0)}{f'(x_0)} \mapsto x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Serie de Taylor ($x_n \approx x*$):

$$f(x) = f(x_n) + (x - x_n)f'(x_n) + \frac{1}{2}(x - x_n)^2 f''(\xi)$$

$\xi_n \in (x, x_n)$ Hacemos $f(x) = 0$:

$$x = x_n - \frac{f(x_n)}{f'(x_n)} - \frac{1}{2}(x - x_n)^2 \frac{f''(\xi_n)}{f'(x_n)}$$

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

Código:

```python
#!/usr/bin/env python3

from scipy import optimize

def NR(f, df, x, n=30, eps=1.0e-9):
    """
    raiz, iter = NR(f, df, x, n=30, tol=1.0e-9)
    Encuentra una raíz de f(x) = 0 por el método de
    Newton-Rahpson. Si se alcanza el máximo número
    de iteraciones n, devuelve None
    """
    for i in range(n):
        try:
            dx = -f(x) / df(x)
        except ZeroDivisionError:
            print("No hay convergencia.")
            quit()
        x += dx
        if abs(dx)< eps:
            return x, i
    print("Límite de iteraciones alcanzado.")
    return None, i

def f(x):
    return x**3 - 10 * x**2 + 5

def df(x):
    return 3 * x**2 - 20 * x

raiz, iteraciones = NR(f, df, 1.0)
print(f"La raiz de f(x) = 0 es {raiz}.")
print(f"Iteraciones: {iteraciones}")

raiz_2 = optimize.newton(f, 1.0, fprime=df)
print(f"La raiz 2 de f(x) = 0 es {raiz_2}.")
print(f"Diferencia: {abs(raiz_2 - raiz)}")
```

```
$ ./newton-raphson.py
La raiz de f(x) = 0 es 0.7346035077893033.
Iteraciones: 4
La raiz 2 de f(x) = 0 es 0.7346035077893033.
Diferencia: 0.0
```

Convergencia: Serie de Taylor alrededor de α, ($f(\alpha) = 0$):

$$f(\alpha) = f(x_n) + f'(x_n)(\alpha - x_n) + R_1 \tag{1}$$

Forma de Lagrange de R_1:

$$R_1 = \frac{1}{2!} f''(\xi_n)(\alpha -_n)^2$$

donde ξ_n está entre α y x_n. Como $f(\alpha) = 0$:

$$0 = f(x_n) + f'(x_n)(\alpha - x_n) + \frac{1}{2!} f''(\xi_n)(\alpha -_n)^2 \tag{2}$$

Dividiendo (2) por $f'(x_n)$ y reacomodando:

$$\frac{f(x_n)}{f'(x_n)} + (\alpha - x_n) = -\frac{f''(\xi_n')}{2f'(x_n)}(\alpha - x_n)^2 \tag{3}$$

$$\text{NR:} \quad x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{4}$$

resulta:

$$\underbrace{\alpha - x_{n+1}}_{\epsilon_{n+1}} = -\frac{f''\xi_n)}{2f'(x_n)}(\underbrace{\alpha - x_n}_{\epsilon_n})^2$$

$$|\epsilon_{n+1}| = \frac{|f''(x_n)|}{2|f'(x_n)|} \cdot \epsilon_n^2 \tag{5}$$

Convergencia al menos **cuadrática** si:

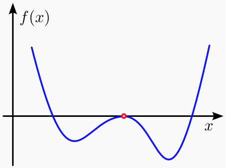
- $f'(x) \neq 0, \ \forall x \in I \equiv [\alpha - |\epsilon_0|, \alpha + |\epsilon_0|]$
- $f''(x) \in \mathcal{C}, \ \forall x \in I$
- $M|\epsilon_0| < 1$ con

$$M = \frac{1}{2}\left(\sup_{x \in I} |f''(x)|\right)\left(\sup_{x \in I} \frac{1}{|f'(x)|}\right)$$

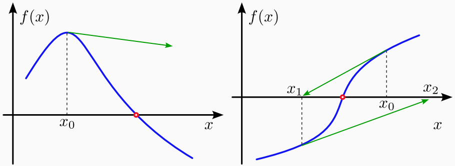
$$\boxed{|\epsilon_{n+1}| \leq M \cdot \epsilon_n^2}$$

Problemas:

- Raíz doble:

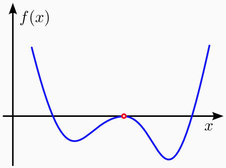


- Divergencia:

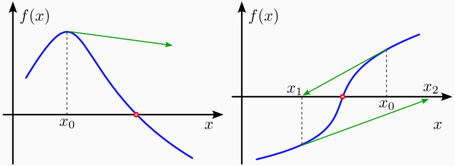


Problemas:

- Raíz doble:



- Divergencia:



Scipy.optimize

```python
#!/usr/bin/env python3

from scipy import optimize

def f(x):
    return x**3 - 10 * x**2 + 5

def df(x):
    return 3 * x**2 - 20 * x

metodos = ['bisect', 'newton', 'brentq', 'toms748']

for m in metodos:
    sol = optimize.root_scalar(f, x0=1.0,
            bracket=[0.01, 2], fprime=df, method=m)
    print(f"Método: {m:7s}, raiz = {sol.root}")
    print(sol)
```

```
$ ./root_scalar.py
Método: bisect , raiz = 0.7346035077910437
Método: newton , raiz = 0.7346035077893033
Método: brentq , raiz = 0.7346035077893016
Método: toms748, raiz = 0.7346035077893033
```

Lecturas recomendadas

- R.L. Burden, D.J. Faires y A.M. Burden. ***Análisis numérico.*** 10.ª ed. Mexico: Cengage Learning, 2017. Capítulo 2.
- J.F. Epperson. ***An Introduction to Numerical Methods and Analysis.*** 2.ª ed. Hoboken, United States: John Wiley & Sons, 2013. Capítulo 3.
- **kiusalaas2005**. Capítulo 4.
- E. Kreyszig, H. Kreyszig y E.J. Norminton. ***Advanced Engineering Mathematics.*** Hoboken, USA: John Wiley & Sons, Inc, 2011. Capítulo 19.