

---

A small, stylized icon of a pen or pencil tip, pointing upwards and to the right, positioned at the end of a horizontal line.

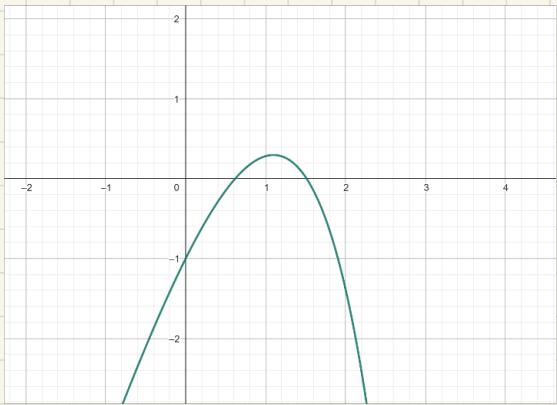
2.1 Realice 3 iteraciones del método de bisección en papel sobre las siguientes funciones. Luego, cree una función en *python* que tome como entrada  $f$  (cualquier función  $f(x)$ ),  $a$  y  $b$  (intervalo inicial),  $N$  (número máximo de iteraciones),  $y$  tol (tolerancia, que será la cota superior del error). Encuentre aproximaciones a las raíces tomando la tolerancia  $\text{tol} = 10^{-5}$  y  $N = 1000$ . Puede elegir tomar el error en la iteración  $i$  como

$$\epsilon = |p_i - p_{i-1}|, \quad \epsilon = \frac{|p_i - p_{i-1}|}{|p_i|}, \quad \text{o} \quad \epsilon = |f(p_i)|.$$

- a)  $3x - e^x = 0$  con  $x \in [1, 2]$ ,
- b)  $2x + 3 \cos x - e^x = 0$  con  $x \in [1, 2]$ ,
- c)  $x^2 - 4x + 4 - \ln x = 0$  con  $x \in [1, 2]$  y  $x \in [2, 4]$ ,
- d)  $x + 1 - 2 \sin \pi x = 0$  con  $x \in [0, 0.5]$  y  $x \in [0.5, 1]$ .

2)  $f(x) = 3x - e^x$

$$x \in [1, 2]$$



$$a = 1 \rightarrow f(a) = 3 - e^1 \quad \text{(circle)}$$

$$b = 2 \rightarrow f(b) = 6 - e^2 \quad \text{(circle)}$$

It<sub>1</sub>:

$$p = \frac{2+1}{2} = 1.5 \rightarrow f(1.5) = \oplus$$

$\Rightarrow$  Cambiamos  $a$  por  $p$

It<sub>2</sub>:

$$p = \frac{1.5+2}{2} = 1.75 \rightarrow f(1.75) = \ominus$$

It<sub>3</sub>:

$$P = \frac{1,5 + 1,75}{2} = 1,625 \rightarrow f(1,625) = \textcolor{blue}{-}$$

⇒ La raíz está entre 1,5 y 1,625

b)

$$2x + 3\cos(x) - e^x \quad x \in [1, 2]$$

$$a=1 \rightarrow f(a) = 0,9$$

$$b=2 \rightarrow f(b) = -4,63$$

$$P_1 = 1,5 \rightarrow f(1,5) = -1,26$$

$$P_2 = \frac{1+1,5}{2} = 1,25 \rightarrow f(1,25) = -0,04$$

$$P_3 = \frac{1+1,25}{2} = 1,125 \rightarrow f(1,125) = 0,46$$

$$\Rightarrow x_0 \in (1; 1,25)$$

c)  $f(x) = x^2 - 4x - \ln(x) + 4$

$x \in [1, 2]$ $f(1) = 1$ $f(2) = -0,69$ $P_1 = 1,5 \rightarrow f(1,5) = -0,15$ $P_2 = 1,25 \rightarrow f(1,25) = 0,33$ $P_3 = 1,375 \rightarrow f(1,375) = 0,07$	$x \in [2, 4]$ $f(2) = -0,69$ $f(4) = 2,6$ $P_1 = 3 \rightarrow f(3) = -0,09$ $P_2 = 3,25 \rightarrow f(3,25) = 0,99$ $P_3 = 3,25 \rightarrow f(3,25) = 0,58$
--	--

$$x_0 \in (1,375 ; 1,5)$$

d)  $f(x) = x + 1 - 2 \sin(\pi x)$

$x \in [0; 0,5]$ $f(0) = 1$ $f(0,5) = -0,5$ $P_1 = 0,25 \rightarrow f(0,25) = -0,16$ $P_2 = 0,125 \rightarrow f(0,125) = 0,33$ $P_3 = 0,1875 \rightarrow f(0,1875) = 0,07$	$x \in [0,5 ; 1]$ $f(0,5) = -0,5$ $f(1) = 2$ $P_1 = 0,75 \rightarrow f(0,75) = 0,33$ $P_2 = 0,625 \rightarrow f(0,625) = -0,22$ $P_3 = 0,6875 \rightarrow f(P_3) = 0,02$
---	---

$$x_0 \in (0,1875 ; 0,25)$$

$$x_1 \in (0,625 ; 0,6875)$$

# La parte de python:

```
def biseccion(f, a, b, tol=10**-5, max_iter=1000):
    for _ in range(max_iter):
        p = (a + b) / 2
        if abs(f(p)) < tol or f(p) == 0:
            return p
        if f(a) * f(p) < 0:
            b = p
        else:
            a = p
    return (a + b) / 2

def funcA(x):
    """a)  $3x - e^x = 0$ """
    return 3*x - np.exp(x)

def funcB(x):
    """b)  $2x + 3\cos(x) - e^x = 0$ """
    return 2*x + 3*np.cos(x) - np.exp(x)

def funcC(x):
    """c)  $x^2 - 4x + 4 - \ln(x) = 0$ """
    return x**2 - 4*x + 4 - np.log(x)

def funcD(x):
    """d)  $x + 1 - 2\sin(\pi x) = 0$ """
    return x + 1 - 2*np.sin(np.pi*x)

funcs = [funcA, funcB, funcC, funcC, funcD, funcD]
intervalos = [(1, 2), (1, 2), (1, 2), (2, 4), (0, 0.5), (0.5, 1)]

for func, intervalo in zip(funcs, intervalos):
    root = biseccion(func, intervalo[0], intervalo[1])
    print(f"Raíz de {func.__name__} entre {intervalo}: {root} aprox")
```

2.2 Sea  $f(x) = (x+2)(x+1)^2x(x-1)^3(x-2)$ . ¿A qué cero de  $f$  converge el método de la bisección cuando se aplica en los siguientes intervalos?

- a)  $[-1.5, 2.5]$ ,
- b)  $[-0.5, 2.4]$ ,
- c)  $[-0.5, 3]$ ,
- d)  $[-3, -0.5]$ .

```
def biseccion(f, a, b, tol=10**-5, max_iter=1000):
    for _ in range(max_iter):
        p = (a + b) / 2
        if abs(f(p)) < tol or f(p) == 0:
            return p
        if f(a) * f(p) < 0:
            b = p
        else:
            a = p
    return (a + b) / 2

inters = [(-1.5, 2.5), (-0.5, 2.4), (-0.5, 3), (-3, -0.5)]
func = lambda x: (x + 2) * (x + 1)**2 * x * (x - 1)**3 * (x - 2)

for intervalo in inters:
    root = biseccion(func, intervalo[0], intervalo[1])

    #redondeo porque todas las raices son enteras
    print(f"En el intervalo: {intervalo} el metodo converge a la raiz {round(root)} aprox")
```

```
En el intervalo: (-1.5, 2.5) el metodo converge a la raiz 0 aprox
En el intervalo: (-0.5, 2.4) el metodo converge a la raiz 0 aprox
En el intervalo: (-0.5, 3) el metodo converge a la raiz 2 aprox
En el intervalo: (-3, -0.5) el metodo converge a la raiz -2 aprox
```

2.3 Una partícula cae desde el reposo por un plano inclinado cuyo ángulo  $\theta$  cambia a una tasa constante

$$\frac{d\theta}{dt} = \omega < 0.$$

La distancia desde el punto de partida de la partícula a tiempo  $t$  está dado por la siguiente fórmula

$$x(t) = -\frac{g}{2\omega^2} \left( \frac{e^{\omega t} - e^{-\omega t}}{2} - \sin \omega t \right).$$

Suponga que la partícula se movió 0,5 m en 1 s y asuma que  $g = 9,8 \text{ m s}^{-2}$ . Encuentre una aproximación de  $\omega$  con error menor a  $10^{-5}$  (elegir uno de los 3 criterios).

Partimos de

$$x(t) = -\frac{g}{2\omega^2} \left( \frac{e^{\omega t} - e^{-\omega t}}{2} - \sin(\omega t) \right)$$

Reemplazamos  $g$  y analizamos  $\partial x$   
llamamos  $F(\omega)$

$$\Rightarrow \partial x = 0,5 = -\frac{9,8}{2\omega^2} \left( \frac{e^\omega - e^{-\omega}}{2} - \sin(\omega) \right) + \frac{9,8}{2\omega^2} \left( \frac{1-1}{2} - 0 \right)$$

$$\Rightarrow F(\omega) = -\frac{9,8}{2\omega^2} \left( \frac{e^\omega - e^{-\omega}}{2} - \sin(\omega) \right) - 0,5$$

buscamos raíces

```
def biseccion(f, a, b, tol=10**-5, max_iter=1000):
    for _ in range(max_iter):
        p = (a + b) / 2
        if abs(f(p)) < tol or f(p) == 0:
            return p
        if f(a) * f(p) < 0:
            b = p
        else:
            a = p
    return (a + b) / 2

func = lambda x: - (9.8 / (2 * x**2)) * ((np.exp(x) - np.exp(-x)) / 2 - np.sin(x)) - 0.5

# Evaluamos la funcion cerca del 0
print(f"f(-10^-5) = {func(-10**-5)}")

#buscamos un x tal que f(x) * f(-10^-5) < 0 para definir el b de la bisección
for i in range(-1, -101, -1):
    if func(i) > 0:
        print(f"w se aproxima a: {biseccion(func, -10**-5, i)}")
        break
print(f"f({i}) = {func(i)}")
```

```
f(-10^-5) = -0.49998590320674147
w se aproxima a: -0.3061135063171387
```

2.4 Sea  $f \in C[a, b]$  y  $f(a)f(b) < 0$ . Muestre que el método de la bisección genera una secuencia  $\{p_n\}_{n=1}^{\infty}$  que aproxima a  $p$  a cero con

$$n \geq 1 \quad |p_n - p| \leq \frac{b-a}{2^n},$$

cuando  $n \geq 1$ . Muestre que esta cota de error que converge linealmente a 0.

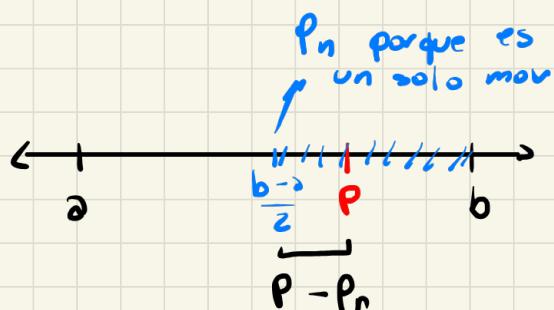
$$|p_n - p| \leq \frac{b-a}{2^n}$$

$p_n \rightarrow$  Val. aproximado

$p \rightarrow$  Val. exacto

Vamos para la primera approx  $n=1$

$$\Rightarrow \underbrace{|p_1 - p|}_{\text{dif entre approx y val (error)}} \leq \frac{b-a}{2^1}$$



Notamos que como  $p \in (a, b)$ , claramente tiene que estar a menos de la mitad de  $b-a$  respecto al centro

Si seguimos con  $n \geq 1$  solo vamos a estar acotando más a la mitad cada intervalo

- corte de  $n=1 \rightarrow 2^1$  fragmentos
- cortes de  $n=2 \rightarrow 2^2$  fragmentos
- cortes de  $n=3 \rightarrow 2^3$  fragmentos

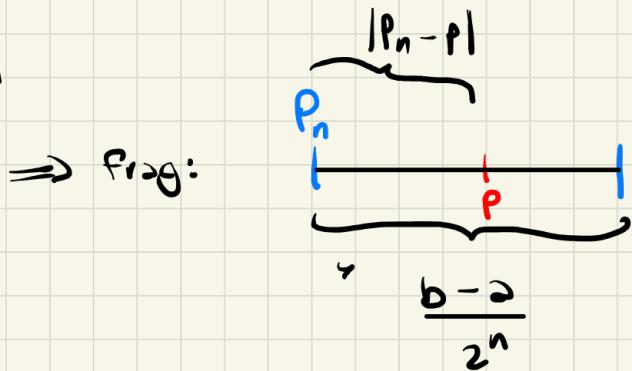


Lo único que vamos a estar haciendo en biseción es elegir que fragmento de  $\frac{b-a}{2^n}$

Vemos que en nuestra recta,  $p$  siempre va a caer dentro de el fragmento que estemos analizando



y como uno de los extremos del fragmento elegido es  $p_n$



Por lo que confirmamos que

$$|p_n - p| \leq \frac{b-a}{2^n} \quad \Rightarrow \text{cuando } n \rightarrow \infty$$

$\Rightarrow \liminf |p_n - p| \leq 0$

$$\frac{b-a}{2^n} \rightarrow 0$$

2.5 Encuentre una cota al número de iteraciones necesarias para lograr una aproximación con error menor a  $10^{-3}$  a la solución de  $x^3 = -x + 4$  en el intervalo  $[1, 4]$ . Encuentre la aproximación.

Tenemos  $x^3 = -x + 4$

→ llamamos  $f(x) = x^3 + x - 4 \rightarrow$  buscamos raíces

Buscamos que el error  $|P_n - p| < 10^{-3}$

Por la fórmula del 2.4

$$|P_n - p| \leq \frac{b-a}{2^n}$$

Por lo que buscamos  $n$  tal que

$$\frac{4-1}{2^n} < 10^{-3}$$

$$2^n > 3000$$

$$n > \log_2(3000)$$

$$n > 11,59$$

Por lo que se necesitan al menos 12 iteraciones

2.6 Sea  $f(x) = (x-1)^{10}$ ,  $p=1$  y  $p_n = 1 + 1/n$ . Muestre que  $|f(p_n)| < 10^{-3}$  para cualquier  $n > 1$  pero que para tener  $|p - p_n| < 10^{-3}$  se necesita  $n > 1000$ .

$$f(x) = (x-1)^{10} \quad p=1 \quad p_n = 1 + \frac{1}{n}$$

$$|f(p_n)| = \left(1 + \frac{1}{n} - 1\right)^{10} < 10^{-3}$$

$$\frac{1}{n^{10}} < \frac{1}{10^3}$$

$$n^{10} > 1000$$

Para  $n > 1$  ( $n \geq 2$ )

$$2^{10} > 1000$$

$$1024 > 1000 \quad \checkmark$$

De acá:  $n^{10} < (n+1)^{10} \quad \forall n : n > 1$

$\Rightarrow$  se cumple el enunciado

Pasemos al  $|p_n - p| < 10^{-3}$

$$\left|1 + \frac{1}{n} - 1\right| < 10^{-3}$$

$$\frac{1}{n} < \frac{1}{1000}$$

Está claro ya

$n > 1000$

2.7 Sea  $f(x) = x^3 - 2x + 1$ . Para resolver  $f(x) = 0$  se proponen los siguientes cuatro problemas de punto fijo. Verifique que los puntos fijos de las funciones  $g(x)$  se corresponden con raíces de  $f(x)$ . Escriba una función de *python* que tome como entrada  $g$  (cualquier función  $g(x)$ ),  $p_0$  (valor inicial),  $N$  (número máximo de iteraciones),  $y$  tol (tolerancia). Encuentre aproximaciones a las raíces tomando la tolerancia  $\text{tol} = 10^{-8}$  y  $N = 1000$ .

- a)  $g(x) = \frac{1}{2}(x^3 + 1)$ ,  $p_0 = \frac{1}{2}$ ,
- b)  $g(x) = \frac{2}{x} - \frac{1}{x^2}$ ,  $p_0 = \frac{1}{2}$ ,
- c)  $g(x) = \sqrt{2 - \frac{1}{x}}$ ,  $p_0 = \frac{1}{2}$ ,
- d)  $g(x) = -\sqrt[3]{1 - 2x}$ ,  $p_0 = \frac{1}{2}$ ,

```
def puntoFijo(g, p0, N, tol):
    for _ in range(N):
        p1 = g(p0)
        if abs(p1 - p0) < tol:
            return p1
        p0 = p1
    return p0

def gA(x):
    return 0.5 * (x**3 + 1)

def gB(x):
    if x == 0:
        print("gB trata de dividir por 0. El metodo de punto fijo falla")
        return x
    return 2/x - 1/(x**2)

def gC(x):
    if x == 0:
        print("gC trata de dividir por 0. El metodo de punto fijo falla")
        return x
    return np.sqrt(2 - 1/x)

def gD(x):
    return -(1 - 2*x)**(1/3)

funcionesG = [gA, gB, gC, gD]
puntosP0 = [0.5, 0.5, 0.5, 0.5]

print("Funciones y puntos iniciales:")
for func, p0 in zip(funcionesG, puntosP0):
    pf = puntoFijo(func, p0, 1000, 10**-8)
    print(f"Punto fijo de {func.__name__}: {pf}")
    print()
```

Funciones y puntos iniciales:  
 Punto fijo de gA: 0.6180339810585875  
 gB trata de dividir por 0. El metodo de punto fijo falla  
 Punto fijo de gB: 0.0  
 gC trata de dividir por 0. El metodo de punto fijo falla  
 Punto fijo de gC: 0.0  
 Punto fijo de gD: -1.61803398786939

En b y c la función trató de dividir por 0 en las primeras iteraciones por lo que no llegó a calcular un punto fijo

2.8 Para las funciones de punto fijo del ejercicio anterior, encuentre cuáles cumplen las condiciones de convergencia de punto fijo y en qué intervalo.

Buscamos  $(a, b)$  tal que

$$\begin{cases} |g'(x)| < 1 \\ g(x) \in (a, b) \end{cases}$$

a)  $g(x) = \frac{1}{2}(x^3 + 1)$

$$g'(x) = \frac{3}{2}x^2$$

$$g''(x) = 3x$$

$$y \quad p \in (a, b)$$

Tomamos  $(0, 5 : 0, 8)$

que contiene al  $0,618$

$\Rightarrow$  vemos que para intervalos positivos es convexa y creciente en todo su dom

Por lo que  $g'(i) < g'(j) \quad \forall i, j$  tal que  $i < j$

$$y \quad |g(0,8)| = 0,96 < 1 \quad \checkmark$$

Ahora, como  $g(x)$  es estrictamente creciente

$$\Rightarrow g(0,5) = 0,56$$

$$g(0,8) = 0,76$$

$$\Rightarrow g(I) \subset I$$

$$\text{con } I = (0,5 ; 0,8)$$

b) , c) no encontramos p

d)  $g(x) = -\sqrt[3]{1-2x}$

tomamos  $(-1,5; -1,8)$

$$g'(x) = \frac{2}{3}(1-2x)^{-\frac{2}{3}}$$

que contiene al  $-1,62$

$$g''(x) = \frac{8}{9}(1-2x)^{-\frac{5}{3}}$$

$\forall x$  negativo,  $g(x)$  es convexo creciente

$$\Leftrightarrow \begin{aligned} g(-1,5) &= -1,59 \\ g(-1,8) &= -1,66 \\ g'(-1,8) &= 0,24 \end{aligned} \quad \left. \begin{array}{l} \rightarrow g(\mathbb{I}) \subset \mathbb{I} \\ \mathbb{I} = (-1,5; -1,8) \end{array} \right\} |g'(x)| < 1 \quad \forall x \in \mathbb{I}$$

2.9 a) Demuestre el teorema de punto fijo (Burden Faires 2.4).

b) Demuestre que si  $g$  satisface las hipótesis del teorema de punto fijo, entonces las cotas de error de la iteración de punto fijo son

$$|p_n - p| \leq k^n \max\{p_0 - a, b - p_0\},$$

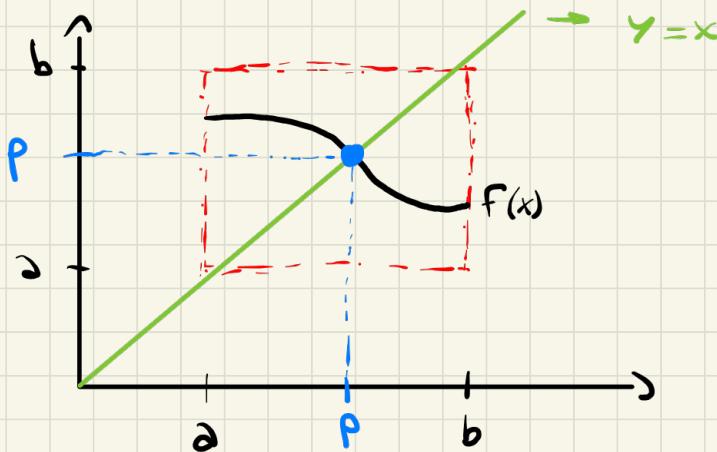
y

$$|p_n - p| \leq \frac{k^n}{1-k} |p_1 - p_0|, \quad \text{para todo } n \geq 1.$$

a) El teorema dice que para una  $f(x)$  continua y derivable en un intervalo  $(a, b)$  tal que

$$- f(x) \in (a, b) \quad \forall x : x \in (a, b)$$

$$- |f'(x)| < 1 \quad \forall x : x \in (a, b)$$



$\Rightarrow \exists_p$  tal que

$$f(p) = p$$

$$\circ f(p) = \text{Id}$$

Demo:

Si  $f(a) = a$  o  $f(b) = b \Rightarrow$  punto fijo en los extremos

Ahora bien, planteamos una

$h(x) = f(x) - x$  la cual cumple que

$$\text{si } h(p) = 0 \Rightarrow f(p) = p$$

$\Rightarrow$  notemos que si  $h(a)h(b) < 0$  implica que

$f(a) > a$  y  $f(b) < b$

O

$f(a) < a$  y  $f(b) > b$

y por el teorema de Bolzano

$\exists c$  tal que  $h(c) = 0$

$\Rightarrow f(c) = c \rightarrow \underline{\text{Punto fijo}}$

b)

Tenemos una  $g(x)$  que satisface el Teo pf

$$|P_n - p| < K^n \cdot \max(P_0 - a, b - P_0)$$

$$|P_n - p| < \frac{K^n}{1 - K} |P_1 - P_0| \quad n \geq 1$$

Partimos de

$$|P_n - p| = |g(P_{n-1}) - g(p)|$$

Ahora, sabemos por valor medio que si tenemos

$$\text{un } p, q \in (a, b) \Rightarrow \exists c \text{ tal que } g'(c) = \frac{g(p) - g(q)}{p - q}$$

$$\Rightarrow g(p) - g(q) = g'(c)(p - q)$$

Usando  $P_{n-1}$  y  $p$  como  $p$  y  $q$

$\text{y } g'(c) < K \text{ por def } < 1$

$$\Rightarrow |g(P_{n-1}) - g(p)| = \overbrace{|g'(c)|}^{< K} |P_{n-1} - p|$$

$$\Rightarrow |P_n - p| < K |P_{n-1} - p|$$

Siguiendo el patrón

$$|P_{n-1} - p| < \kappa |P_{n-2} - p|$$

Por lo que

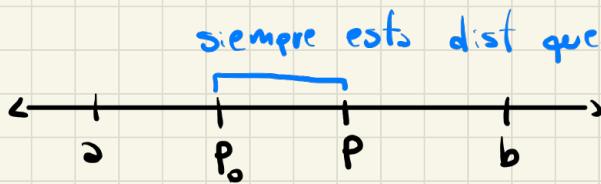
$$|P_n - p| < \kappa^2 |P_{n-2} - p| \text{ y si iteramos } n \text{ veces}$$

nos queda

$$|P_n - p| < \kappa^n |P_0 - p|$$

$\kappa$  es un num entre 0,1 cualquier que usamos para hacer ver que efectivamente es menor

Sabiendo esto veamos que como  $p \in (a, b)$



siempre este dist quedo menor a los de  $P_0$  y algun extremo

$$\Rightarrow \text{como } |P_0 - p| < \max(P_0 - a, b - P_0)$$

Entonces

$$|P_n - p| < \kappa^n |P_0 - p| < \kappa^n \max(P_0 - a, b - P_0)$$

Para la segunda desigualdad partimos de

$$|P_n - P| < \kappa |P_{n-1} - P| \quad y \text{ analizamos por}$$

$$\Rightarrow \underbrace{|P_{j+1} - P_j|}_{\text{dif de la iter}} < \kappa \underbrace{|P_{j-1} - P_j|}_{\text{dif de lo anterior}} \quad \begin{array}{l} \text{iteraciones como} \\ \text{se comporta} \end{array}$$

Inicialmente:

$$|P_1 - P_0| \rightarrow \text{primera iteración } (j=0)$$

$$|P_2 - P_1| < \kappa |P_1 - P_0| \rightarrow \text{segunda } (j=1)$$

$$|P_3 - P_2| < \kappa |P_2 - P_1| < \kappa^2 |P_1 - P_0| \quad (j=2)$$

$\Rightarrow$  En general cada iteración individual es:

$$|P_{j+1} - P_j| < \kappa^j |P_1 - P_0|$$

y sumando todas las iteraciones ( $j \rightarrow \infty$ )

$$\sum_{j=0}^{\infty} |P_{j+1} - P_j| < \sum_{j=0}^{\infty} \kappa^j |P_1 - P_0| \quad \begin{array}{l} \text{no depende de } j \\ \text{converge a } \frac{1}{1-\kappa} \end{array}$$

Ahora desigualdad triangular (prop)  $|a+b| \leq |a| + |b|$

$$\left| \sum_{j=0}^{\infty} (P_{j+1} - P_j) \right| \leq \sum_{j=0}^{\infty} |P_{j+1} - P_j| < \sum_{j=0}^{\infty} k^j |P_1 - P_0|$$



$$\left| (P_1 - P_0) + (P_2 - P_1) + (P_3 - P_2) + \dots + (P_j - P_{j-1}) + (P_{j+1} - P_j) \right|$$

se van cancelando todo y queda  $|P - P_0|$  p por inf tiende a

$$\Rightarrow \underbrace{|P - P_0|}_{\text{arrow}} \leq \frac{1}{1-k} |P_1 - P_0|$$

y reemplazando en  $|P_n - P| < k^n |P - P_0|$

llegamos a  $|P_n - P| < \frac{k^n}{1-k} |P_1 - P_0|$

2.10 Los siguientes cuatro métodos son propuestos para computar  $21^{1/3}$ . Asumiendo  $p_0 = 1$ , ordénelos de mayor a menor con respecto a su velocidad aparente de convergencia.

$$a) p_n = \frac{20p_{n-1} + 21/p_{n-1}^2}{21}$$

$$b) p_n = p_{n-1} - \frac{p_{n-1}^3 - 21}{3p_{n-1}^2}$$

$$c) p_n = p_{n-1} - \frac{p_{n-1}^4 - 21p_{n-1}}{p_{n-1}^2 - 21}$$

$$d) p_n = \left( \frac{21}{p_{n-1}} \right)^{1/2}$$

```
def gA(x):
    return (20 * x + 21 * x**2) / 21

def gB(x):
    return x - (x**3 - 21)/3 * x**2

def gC(x):
    return x - (x**4 - 21 * x) / (x**2 - 21)

def gD(x):
    return (21 / x)**(1/2)

def puntoFijo(g, p0, N, tol):
    iters = 0
    for _ in range(N):
        p1 = g(p0)
        iters += 1
        if abs(p1 - p0) < tol:
            return p1, iters
        p0 = p1
    return p0, iters

funcs = [gA, gB, gC, gD]
for func in funcs:
    pf, iters = puntoFijo(func, 1, 1, 10**-5)
    print(f"Raíz encontrada por {func.__name__}: {pf} en {iters} iteraciones")
    print(f"Error: {abs(pf**2 - 21**(1/3))}")
```

```
Raíz encontrada por gA: 1.9523809523809523 en 1 iteraciones
Error: 1.052867206838834
Raíz encontrada por gB: 7.6666666666666667 en 1 iteraciones
Error: 56.018853601396664
Raíz encontrada por gC: 0.0 en 1 iteraciones
Error: 2.7589241763811203
Raíz encontrada por gD: 4.58257569495584 en 1 iteraciones
Error: 18.24107582361888
```

Por lo primer iteración,  
aparentemente se approxima  
en este orden:

a) - c) - d) - b)

Pero cuando analizamos algunas iters mas:

```
Raíz encontrada por gA: 1.3730619861265334e+101 en 9 iteraciones
Error: 1.8852992177457406e+202
Raíz encontrada por gB: nan en 9 iteraciones
Error: nan
Raíz encontrada por gC: 0.0 en 2 iteraciones
Error: 2.7589241763811203
Raíz encontrada por gD: 2.764398093397935 en 9 iteraciones
Error: 4.882972642401018
```

Vemos que b) diverge,  
a) se está yendo al  
carajo, c) nunca

Funcionó y d) va a converger

2. Utilice un método de iteración de punto fijo para encontrar la solución de las siguientes ecuaciones con una precisión de  $10^{-2}$ .

- $x^3 - x - 1 = 0$ , intervalo de interés:  $[1, 2]$  y  $p_0 = 1$ .
- $e^{-x} - x = 0$ , intervalo de interés:  $[0, 1]$  y  $p_0 = 1$ .
- $\sin(\ln(x)) - (x^3 - x^2) = 0$ , intervalo de interés:  $[0, 1]$  y  $p_0 = 0, 1$ .

a)  $x^3 - x - 1 = 0$  en  $[0, 1 ; 1]$  y  $p_0 = 0, 1$

notamos que

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

Por lo que buscamos las raíces de  $f(x)$

tal que  $x = g(x)$  con  $g(x) = x^3 - 1$

Pero como  $g'(x) = 3x^2$

La cual es  $> 1$   
cerca de  $p=1$

$\Rightarrow$  no cumplimos el criterio de la derivada para punto fijo

Por lo que reescribimos  $g(x)$

$$\Rightarrow x^3 - x - 1 = 0 \rightarrow x = \sqrt[3]{x+1}$$

Por lo que probamos buscando otra  $g(x)$

$$x^3 - x - 1 = 0 \Rightarrow x = \sqrt[3]{x+1}$$

$\Rightarrow$  vemos que  $g'(x) = \frac{(x+1)^{-\frac{2}{3}}}{3} < 1$

ahora cumple el criterio de convergencia

Al final pongo el código con los resultados

c)  $\sin(\ln(x)) - x^3 + x^2 = 0$

$$x = \sqrt[3]{\sin(\ln(x)) + x^2} \quad \boxed{g(x)}$$

Pero  $g(0,1) = \sqrt[3]{\sin(\ln(0,1)) + 0,01}$  DA NEGATIVO!!

$$x = \sqrt{x^3 - \sin(\ln(x))} \quad \boxed{g(x)}$$

Pero  $g'(x) = \frac{3x^2 - \frac{\cos(\ln(x))}{x}}{2\sqrt{x^3 - \sin(\ln(x))}} \Rightarrow |g'(1)| = \frac{3 - \frac{\cos(0)}{1}}{2\sqrt{1 - \sin(0)}} = 1$

No cumple  $|g'(x)| < 1$

Última opción

$$x = e^{\arcsen(x^3 - x^2)} \quad g(x)$$

Pero  $g'(x) = e^{\arcsen(x^3 - x^2)} \left( \frac{3x^2 - 2x}{\sqrt{1 - (x^3 - x^2)^2}} \right)$

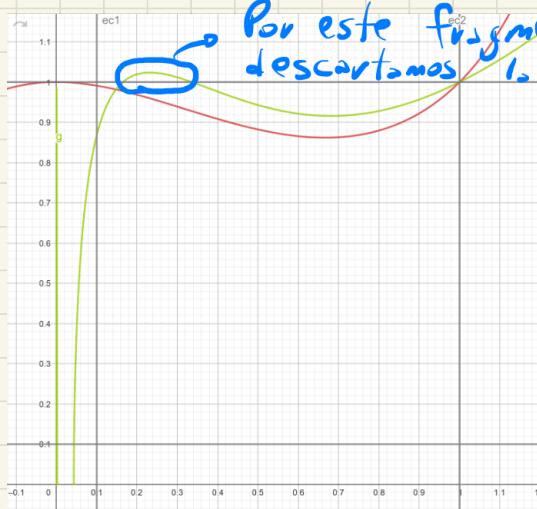
$$\Rightarrow g'(1) = \underbrace{e^{\arcsen(0)}}_{=1} \left( \frac{\frac{3-2}{\sqrt{1-0}}}{=1} \right)$$

Tampoco vale que  $|g'(x)| < 1$

En estos casos que no encontramos una buena  $g(x)$  podemos conformarnos si  $|g'(x)| = 1$  aunque sea poco óptimo

$$e^{\arcsen(x^3 - x^2)}$$

$$\sqrt{x^3 - \sin(\ln(x))}$$



Ahora sí, para el c llegamos que la mejor

$$g(x) = \arcsen(x^3 - x^2)$$

Código:

```
def puntoFijo(g, p0, N, tol):
    for _ in range(N):
        p1 = g(p0)
        if abs(p1 - p0) < tol:
            return p1
        p0 = p1
    return p0

def gA(x):
    return (x + 1)**(1/3)

def gB(x):
    return np.exp(-x)

def gC(x):
    return np.exp(math.asin(x**3 - x**2))

funcs = [gA, gB, gC]
puntosP0 = [1, 1, 0.1]

for func, p0 in zip(funcs, puntosP0):
    pf = puntoFijo(func, p0, 10000, 10**-2)
    print(f"Punto fijo de {func.__name__}: {pf}")
```

```
Punto fijo de gA: 1.324268744551578
Punto fijo de gB: 0.5648793473910495
Punto fijo de gC: 0.9912385863085208
```

2.12 Muestre que  $g(x) = 2^{-x}$  tiene un único punto fijo en  $[1/3, 1]$ . Utilice iteración de punto fijo para encontrarlo con una precisión de  $10^{-4}$ . Encuentre una cota al número de iteraciones necesarias para llegar a esta tolerancia partiendo de  $p_0 = 1$ . y compárela con el que obtuvo empíricamente.

$$g(x) = 2^{-x}$$

$$g'(x) = -\ln(2) 2^{-x}$$

$$\max(|g'(x)|) = 0,55015 \quad \text{en } x = \frac{1}{3}$$

Usando que  $\underline{\text{err}}$

$$|P_n - P| < k^n \max(P_0 - a, b - P_0)$$

$$k^n \max\left(1 - \frac{1}{3}, 1 - 1\right) < 10^{-4}$$

$$n \geq |g'(x)|^{-1} \frac{2}{3} < 10^{-4}$$

$$\Rightarrow \text{usamos el máximo de } (0,55015)^n < \frac{3}{20000}$$

$$\log_{0,55015}\left(\frac{3}{20000}\right) < n$$

$$n > 14,73$$

Necesitas mas de 15 iteraciones

```
def puntoFijo(g, p0, N, tol):
    iter = 0
    for _ in range(N):
        p1 = g(p0)
        if abs(p1 - p0) < tol:
            return p1, iter
        p0 = p1
        iter += 1
    return p0, iter

def f(x):
    return 2**(-x)

result = puntoFijo(f, 0, 1000, 10**-4)
print(f"Punto fijo en: {result[0]}. Iteraciones: {result[1]}")
```

Punto fijo en: 0.6412052524498624. Iteraciones: 12

En la teoría llegamos a que se necesitan 15  
iters, empíricamente, 12

2.13 Encuentre un valor aproximado iterando 3 veces para  $\sqrt[3]{25}$  utilizando el método de la bisección y el método de iteración de punto fijo. Compare los resultados. Busque el número de iteraciones necesarias para tener una precisión de  $10^{-4}$ .

Probamos buscar  $\sqrt[3]{25} = x$   
 $\Rightarrow$  llamamos  $f(x) = x^3 - 25$

Ahora, para punto fijo:

$$g(x) = x - F(x)$$

$$g(x) = x - x^3 + 25$$

Chequemos:

$$|g'(x)| = |1 - 3x^2| < 1$$

$$\begin{aligned} -1 &< 1 - 3x^2 < 1 \\ 2 &> 3x^2 > 0 \end{aligned} \quad \left. \right\} x \in (0, \sqrt{\frac{2}{3}})$$

Como no cumple que  $|g'(x)| < 1 \quad \forall x$  que nos interesan

$$\Rightarrow \text{vemos } 2 > 3x^2 > 0$$

Cabo truco

$$g(x) = x \pm \alpha f(x)$$

siempre vale

La idea es que  $x \in (0, b)$  con  $b > \sqrt[3]{25}$

Tomemos  $b = 4$  (por ver en el gráfico)

$$\Rightarrow 3\alpha x^2 < 2 \rightarrow \sqrt{\frac{2}{3\alpha}} = 4$$

$$\alpha = \frac{1}{24}$$

$$\Rightarrow \text{nueva } f(x) = \frac{1}{24}(x^3 - 25)$$

$$g(x) = x - \frac{1}{24}(x^3 - 25)$$

```
def f(x):
    return (1/24) * (x**3 - 25)

def g(x):
    return x - (1/24) * (x**3 - 25)

def puntoFijo(g, p0, N, tol):
    iters = 0
    for _ in range(N):
        p1 = g(p0)
        if abs(p1 - p0) < tol:
            return p1, iters
        p0 = p1
        iters += 1
    return p0, iters

def biseccion(f, a, b, tol=10**-4, max_iter=1000):
    iters = 0
    for _ in range(max_iter):
        p = (a + b) / 2
        iters += 1
        if abs(f(p)) < tol or f(p) == 0:
            return p, iters
        if f(a) * f(p) < 0:
            b = p
        else:
            a = p
    return (a + b) / 2, iters

print("Con 3 iteraciones:")
print(f"Bisección entre 0 y 4: {biseccion(f, 0, 4, 10**-4, 3)}")
print(f"Punto fijo de g con p0 0: {puntoFijo(g, 0, 3, 10**-4)}")
print()
print("Buscando tolerancia 10^-4:")
bisecc = biseccion(f, 0, 4, 10**-4)
pf = puntoFijo(g, 0, 1000, 10**-4)
print(f"Bisección entre 0 y 4: {bisecc[0]} en {bisecc[1]} iteraciones")
print(f"Punto fijo de g con p0 0: {pf[0]} en {pf[1]} iteraciones")
```

Con 3 iteraciones:  
Bisección entre 0 y 4: 2.75  
Punto fijo de g con p0 0: 2.726122199459186  
  
Buscando tolerancia 10^-4:  
Bisección entre 0 y 4: 2.924072265625 en 14 iteraciones  
Punto fijo de g con p0 0: 2.9240159050138743 en 5 iteraciones

2.14 Sea  $g \in C^1[a, b]$  y  $p \in (a, b)$  con  $g(p) = p$  y  $|g'(p)| > 1$ . Muestre que existe un  $\delta > 0$  tal que si  $0 < |p_0 - p| < \delta$ , entonces  $|p_0 - p| < |p_1 - p|$ . Es decir, no importa que tan cerca esté  $p_0$  de  $p$ , la siguiente iteración  $p_1$  siempre estará más lejos, por lo que el método de iteración de punto fijo no converge si  $p_0 \neq p$ .

Partamos de que

$$|P_n - P| < \frac{\kappa^n}{1-\kappa} |P_1 - P_0| \quad \text{con } \kappa > |g'(x)|$$

Con  $n=1$

$$\begin{aligned} |P_1 - P| &< \frac{\kappa}{1-\kappa} |P_1 - P_0| \\ \Rightarrow |P_1 - P| &< -\kappa |P_1 - P_0| \quad \text{ABS!} \end{aligned}$$

$$|P_n - P| = |g(P_{n-1}) - g(P)| = |g'(c)| |P_{n-1} - P|$$

$$\Leftrightarrow \exists c \text{ tal que } g'(c) = \frac{g(P_n) - g(P)}{P_n - P} > 1$$

$\Rightarrow$  llegamos a que

$$|P_n - P| = |g'(c)| |P_{n-1} - P| \text{ por lo que } |g'(c)| > 1$$

$$|P_n - P| > |P_{n-1} - P|$$

2.15 Encuentre una función  $g$  definida en  $[0, 1]$  que no cumple ninguna de las hipótesis del teorema de punto fijo pero igual tiene un único punto fijo en  $[0, 1]$ .

Aca jugó chat descaradamente

Planteamos

$$g(x) = x + \sqrt{x}$$

No cumple que  $g(x) \in [0, 1] \quad \forall x : x \in [0, 1]$

$$g(0,5) = 0,5 + \sqrt{0,5} = 1,207 \notin [0, 1]$$

Ademas no es  $C^1[0, 1]$

$$g'(x) = 1 + \frac{1}{2\sqrt{x}} \quad y \quad g'(0) \text{ explota todo}$$

Por ultimo

$$|g'(x)| = \left| 1 + \frac{1}{2\sqrt{x}} \right| > 1 \text{ siempre}$$

(es  $1 + \text{algo positivo} \neq 0$ )

2.16 Utilice el método de Newton para encontrar soluciones con los siguientes problemas

- $e^x + 2^{-x} + 2 \cos x - 6 = 0$ , con  $x \in [1, 2]$ ,
- $\ln(x-1) + \cos(x-1) = 0$ , con  $x \in [1, 3, 2]$ ,
- $(x-2)^2 - \ln x = 0$ , con  $x \in [1, 2]$ .

Primero resuelva un par de iteraciones a mano, partiendo de algún  $p_0$  dentro de los intervalos, y luego implemente una función de *python* para converger con una tolerancia de  $10^{-5}$ .

b)  $f(x) = \ln(x-1) + \cos(x-1) = 0 \quad x \in [1, 3, 2]$

$$f'(x) = \frac{1}{x-1} + \sin(x-1)$$

tomamos  $x_0 = 1,5$

Porque esto cerca

c)  $f(x) = e^x + 2^{-x} + 2 \cos(x) - 6 = 0 \quad x \in [1, 2]$

$$f'(x) = e^x - \ln(2) 2^{-x} - 2 \sin(x)$$

c)  $F(x) = (x-2)^2 - \ln(x) = 0 \quad x \in [1, 2]$

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

Mas oya va a hacer

la parte a mano

```
def Newton(f, df, x0, tol, max_iter):
    for _ in range(max_iter):
        x1 = x0 - f(x0) / df(x0)
        if abs(x1 - x0) < tol:
            return x1
        x0 = x1
    return x0

def fA(x):
    return np.exp(x) + 2**(-x) + 2*np.cos(x) - 6

def dfA(x):
    return np.exp(x) - 2**(-x) * np.log(2) - 2*np.sin(x)

def fB(x):
    return np.log(x - 1) + np.cos(x - 1)

def dfB(x):
    return 1/(x - 1) - np.sin(x - 1)

def fC(x):
    return (x - 2)**2 - np.log(x)

def dfC(x):
    return 2*(x - 2) - 1/x

fs = [fA, fB, fC]
dfs = [dfA, dfB, dfC]

for i in fs:
    print(f"Raiz de {i.__name__}: {Newton(i, dfs[fs.index(i)], 1.5, 10**-5, 1000)}")
```

Raiz de fA: 1.829383601933849

Raiz de fB: 1.3977484759580519

Raiz de fC: 1.4123911720238844

2.17 La ecuación  $x^2 - 10 \cos x = 0$  tiene dos soluciones  $\pm 1,3793646$ . Utilice el método de Newton para aproximar las soluciones con los siguientes  $p_0$

- a)  $p_0 = -100,$
- b)  $p_0 = -50,$
- c)  $p_0 = -25,$
- d)  $p_0 = 25,$
- e)  $p_0 = 50,$
- f)  $p_0 = 100.$

```
def Newton(f, df, x0, tol, max_iter):  
    for _ in range(max_iter):  
        x1 = x0 - f(x0) / df(x0)  
        if abs(x1 - x0) < tol:  
            return x1  
        x0 = x1  
    return x0  
  
def f(x):  
    return x**2 - 10*np.cos(x)  
  
def df(x):  
    return 2*x + 10*np.sin(x)  
  
p0 = [-100, -50, -25, 25, 50, 100]  
  
for i in p0:  
    root = Newton(f, df, i, 10**-5, 1000)  
    print(f"Raíz encontrada con p0={i}: {root}")
```

Raíz encontrada con  $p_0=-100: -1.3793645942270283$   
Raíz encontrada con  $p_0=-50: -1.3793645942220307$   
Raíz encontrada con  $p_0=-25: 1.3793645942242991$   
Raíz encontrada con  $p_0=25: -1.3793645942242991$   
Raíz encontrada con  $p_0=50: 1.3793645942220307$   
Raíz encontrada con  $p_0=100: 1.3793645942270283$

2.18 La función  $f(x) = \ln(x^2 + 1) - e^{0.4x} \cos \pi x$  tiene un número infinito de ceros.

- Determine, con precisión de  $10^{-6}$ , el único cero negativo.
- Determine, con precisión de  $10^{-6}$ , los cuatro ceros positivos más pequeños.
- Determine una aproximación inicial razonable para encontrar el  $n$ -ésimo cero positivo más pequeño.
- Utilice el resultado del inciso anterior para encontrar el 25avo cero positivo más pequeño.

Aca están hechos a) y b)

```
def f(x):
    return np.log(x**2 + 1) - np.exp(0.4*x) * np.cos(np.pi*x)

# se la re complicaba copilot jaaja
def df(x):
    term1 = (2*x) / (x**2 + 1)
    exp_term = np.exp(0.4*x)
    term2 = exp_term * (0.4 * np.cos(np.pi*x) - np.pi * np.sin(np.pi*x))
    return term1 - term2

def Newton(f, df, x0, tol, max_iter):
    for _ in range(max_iter):
        x1 = x0 - f(x0) / df(x0)
        if abs(x1 - x0) < tol:
            return x1
        x0 = x1
    return x0

def inSeen(x):
    for i in seen:
        if abs(i - x) < 10**-6:
            return True
    return False

print(f"a) Cero negativo, arranco en -2 por probar (-1 no funcio): {Newton(f, df, -2, 10**-6, 1000)}")
print("b)")
seen = []
for i in range(4):
    last = i
    root = -1
    for j in range(last, 1000):
        root = Newton(f, df, j/2, 10**-6, 1000)
        if root > 0 and not inSeen(root):
            last = j
            seen.append(root)
            print(f"    Cero positivo, con p0 = {j/2} en: {root}")
            break
    root = -1
```

a) Cero negativo, arranco en -2 por probar (-1 no funcio): -0.4341430472857288

b)

```
Cero positivo, con p0 = 0.0 en: 0.4506567478900095
Cero positivo, con p0 = 1.5 en: 1.7447380533683496
Cero positivo, con p0 = 2.5 en: 2.2383197950741383
Cero positivo, con p0 = 3.5 en: 3.709041201375952
```

Para el c. n. Agustín Orozco (el de lo tutto)  
supo hacerlo. Ésto recomendó el chat:

Sea

$$f(x) = \ln(x^2 + 1) - e^{0.4x} \cos(\pi x), \quad f'(x) = \frac{2x}{x^2 + 1} - 0.4e^{0.4x} \cos(\pi x) + \pi e^{0.4x} \sin(\pi x).$$

Los ceros positivos quedan muy cerca de los semienteros. Para el  $n$ -ésimo (con  $n = 1$  el más chico), usá como semilla de Newton:

$$x_n^{(0)} = n - \frac{1}{2}$$

Si querés una semilla aún mejor (una "semilla de Newton" de primer orden obtenida linealizando en el semientero), tomá:

$$x_n^{(0)} = \left(n - \frac{1}{2}\right) + \frac{(-1)^n \ln\left((n - \frac{1}{2})^2 + 1\right)}{\pi e^{0.4(n - \frac{1}{2})}}$$

(Para  $n$  impar la corrección es hacia la izquierda; para  $n$  par, hacia la derecha.)

Luego aplicás Newton-Raphson:

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

Add del chat y resultado del d)

```
# Idea del chat para el c
def chatIdeaP0(n):
    p0 = n - 0.5
    return p0 + ((-1)**n) * np.log(p0**2 + 1) / (np.pi * np.exp(0.4 * p0))

print("d)")
print(f" Cero positivo numero 25: {Newton(f, df, chatIdeaP0(25), 10**-6, 1000)}")
```

Cero positivo numero 25: 24.499887047445956

2.19 Utilice el método de Newton para resolver la ecuación

$$0 = \frac{1}{2} + \frac{1}{4}x^2 - x \sin x - \frac{1}{2} \cos 2x,$$

con  $p_0 = \pi/2$ . ¿Algo le resultado llamativo? Repita con  $p_0 = 5\pi$  y  $p_0 = 10\pi$ .

*nop*

$$f(x) = \frac{1}{2} + \frac{1}{4}x^2 - x \sin(x) - \frac{1}{2} \cos(2x)$$

$$f'(x) = \frac{1}{2}x + \sin(2x) - \sin(x) - x \cos(x)$$

```
def f(x):
    return 1/2 + (x**2)/4 - x * np.sin(x) - (1/2) * np.cos(2*x)

def df(x):
    return 1/2 * x + np.sin(2*x) - np.sin(x) - x * np.cos(x)

def Newton(f, df, x0, tol, max_iter):
    for _ in range(max_iter):
        x1 = x0 - f(x0) / df(x0)
        if abs(x1 - x0) < tol:
            return x1
        x0 = x1
    return x0

p0 = [np.pi/2, 5 * np.pi, 10 * np.pi]

for i in p0:
    print(f"Raiz con p0 = {i}: {Newton(f, df, i, 10**-5, 1000)}")
```

Raiz con  $p_0 = \pi/2$ : 1.8954884189769137  
Raiz con  $p_0 = 5\pi$ : 1.8954890013853471  
Raiz con  $p_0 = 10\pi$ : 9.172687397047352e-06

## 2.20 La ecuación de anualidades ordinaria

$$A = \frac{P}{i} [1 - (1 + i)^{-n}],$$

se utiliza para calcular cuanto se debe pagar de una hipoteca en un período fijo de tiempo.  $A$  es el monto de la hipoteca,  $P$  es el monto de cada pago,  $i$  es la tasa de interés por período por los  $n$  períodos de pago. Suponga que es necesaria una hipoteca a 30 años con un valor de \$135000 y que el prestatante puede pagar como máximo \$1000 por mes. ¿Cuál es el interés máximo que puede pagar el prestatante?

$$135000 = \frac{1000}{i} \left( 1 - (1 + i)^{-30 \cdot 12} \right)$$

$$f(x) = \frac{1000}{x} \left( 1 - (1 + x)^{-360} \right) - 135000$$

$$f'(x) = -\frac{1000}{x^2} \left( 1 - (1 + x)^{-360} \right) + \frac{360000}{x} (1 + x)^{-361}$$

```
def f(x):
    return 1000/x * (1 - (1 + x)**-360) - 135000

def df(x):
    return -1000/(x**2) * (1 - (1 + x)**-360) + 1000/x * 360 * (1 + x)**-361

def Newton(f, df, x0, tol, max_iter):
    for _ in range(max_iter):
        x1 = x0 - f(x0) / df(x0)
        if abs(x1 - x0) < tol:
            return x1
        x0 = x1
    return x0

#hay que usar un p0 muy chico porque se va al carajo newton por la pendiente
print(Newton(f, df, 0.00005, 10**-5, 1000))
```

La sol a la ec: 0.006749917140648157

eso es el interés mensual maximo

2.21 La ecuación de iteración para el método de la secante se puede escribir de la siguiente forma

$$p_n = \frac{f(p_{n-1})p_{n-2} - f(p_{n-2})p_{n-1}}{f(p_{n-1}) - f(p_{n-2})}.$$

Explique por qué, en general, es probable que esta ecuación sea menos precisa que la vista en clase.

Partiendo de Newton donde

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{F'(p_{n-1})}$$

Podemos reescribir

$$F'(p_{n-1}) = \frac{f(p_{n-1}) - f(p_{n-2})}{p_{n-1} - p_{n-2}}$$

reemplazando

$$p_n = p_{n-1} - \frac{f(p_{n-1})(p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})}$$

$$p_n = \frac{p_{n-1}(f(p_{n-1}) - f(p_{n-2})) - f(p_{n-1})(p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})}$$

$$p_n = \frac{\cancel{p_{n-1} + f(p_{n-1})} - p_{n-1} f(p_{n-2}) - \cancel{f(p_{n-1}) p_{n-1} + f(p_{n-1}) p_{n-2}}}{f(p_{n-1}) - f(p_{n-2})}$$

$$P_n = \frac{f(P_{n-1}) P_{n-2} - P_{n-1} f(P_{n-2})}{F(P_{n-1}) - f(P_{n-2})}$$

⇒ Sirve porque viene de desarrollar la derivada como la pendiente entre los anteriores dos puntos por lo que ya no se necesita derivar  $f$

2.22 Derive la siguiente formula de error para el método de Newton

$$|p - p_{n+1}| \leq \frac{M}{2|f'(p_n)|} |p - p_n|^2,$$

asumiendo que las hipótesis del teorema visto en clase se cumplen, que  $|f'(p_n)| \neq 0$  y que  $M = \max |f''(x)|$ . Ayuda: utilice la derivación del método de Newton a partir de la serie de Taylor de  $f(p)$  vista en clase.

Taylor b.  $f(p)$

$$F(p) = f(p_n) + (p - p_n)f'(p_n) + \underline{O((p_n - p)^2)}$$

$$\text{err dc Taylor} = \frac{f''(c)(p_n - p)^2}{2}$$

Si igualamos  $f(p) = 0$  ya que  $p$  es raíz

$$\underbrace{f'(p_n)(p - p_n)}_{= -f(p_n) - \frac{f''(c)(p_n - p)^2}{2}} = -f(p_n) - \frac{f''(c)(p_n - p)^2}{2}$$

$$p = p_n - \frac{f(p_n)}{f'(p_n)} - \frac{f''(c)(p_n - p)^2}{2 f'(p_n)}$$

$\hookrightarrow p_1$

$$\Rightarrow |p - p_{n+1}| = \left| -\frac{f''(c)(p_n - p)^2}{2 f'(p_n)} \right|$$

$$|p - p_{n+1}| = \frac{M}{2|f'(p_n)|} |p_n - p|^2$$

2.23 Dada  $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$  el método Newton-Raphson generalizado consiste en realizar la iteración vectorial

$$x_{k+1} = x_k - (DF|_{x_k})^{-1} F(x_k),$$

donde  $x_k \in \mathbb{R}^n$  y  $(DF|_{x_k})^{-1}$  es la inversa de la matriz diferencial de  $F$  evaluada en  $x_k$ . Usar Newton-Raphson generalizado para hallar una raíz de la función

$$F(x_1, x_2) = \begin{bmatrix} x_1 + x_2 - 3 \\ x_1^2 + x_2^2 - 9 \end{bmatrix}.$$

Tenemos

$$f(x, y) = \begin{pmatrix} x + y - 3 \\ x^2 + y^2 - 9 \end{pmatrix}$$

Buscamos aprox  $x, y$  tal que  $f(x, y) = 0$

Sacabidón:

$$DF(x, y) = \begin{pmatrix} 1 & 1 \\ 2x & 2y \end{pmatrix}$$

$$DF^{-1} = \frac{1}{2y - 2x} \begin{pmatrix} 2y & -1 \\ -2x & 1 \end{pmatrix}$$

Planteamos una nueva forma de representar el err

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

# Utilizamos Newton:

$$x_{k+1} = x_k - Df^{-1}(x_k) F(x_k) \quad x_k \in \mathbb{R}^n$$

Código:

```
def F(X):
    x, y = X
    return np.array([x + y - 3, x**2 + y**2 - 9])

def DF(X, inverse = False):
    x, y = X
    if inverse:
        matrix = 1/(2.0*(y - x)) * np.array([[2*y, -1], [-2*x, 1]])
    else:
        matrix = np.array([[1, 1], [2*x, 2*y]])
    return matrix

def newton(F, DF, X0, N, tol, metodo):
    k = 0
    XK = X0
    Xlist = [X0]
    err = tol + 1

    if metodo == 'Directo':
        while err > tol and k < N:
            XK = XK - DF(XK, inverse=True) @ F(XK)
            Xlist.append(XK)
            k += 1
    elif metodo == 'Ingenioso':
        while err > tol and k < N:
            XK = np.linalg.solve(DF(XK), -F(XK) + DF(XK) @ XK)
            Xlist.append(XK)
            k += 1

    return XK

print("Raíces método directo: {newton(F, DF, np.array([2, 1]), 1000, 10**-5, 'Directo')}")
print("Raíces método ingenioso: {newton(F, DF, np.array([0.5, 1]), 1000, 10**-5, 'Ingenioso')}")
```

```
Raíces método directo: [3.0000000e+00 3.60158409e-17]
Raíces método ingenioso: [-4.4408921e-16 3.0000000e+00]
```

2.24 Muestre que las siguientes secuencias convergen sublinealmente a  $p = 0$ . ¿Qué tan grande debe ser  $n$  para que  $|p_n - p| \leq 5 \times 10^{-2}$ ?

a)  $p_n = \frac{1}{n}, n \geq 1,$

b)  $p_n = \frac{1}{n^2}, n \geq 1.$

Convergencia lineal:

Dado un  $p_n$  y un  $p$ , se calcula

} by chat

$$\lim_{n \rightarrow \infty} \frac{|p_{n+1} - p|}{|p_n - p|} = q$$

- Lineal si  $\lim_{n \rightarrow \infty} \frac{|p_{n+1} - p|}{|p_n - p|} = q$  con  $0 < q < 1$ .
- Superlineal si ese límite es 0.
- Sublineal si ese límite es 1.

$$\Rightarrow \text{Ambos casos} \left\{ \begin{array}{l} \lim_{n \rightarrow \infty} \frac{\frac{1}{n+1}}{\frac{1}{n}} \rightarrow 1 \\ \text{(L'Hopital y} \\ \text{huevadas que medir} \\ \text{pds) hacer} \end{array} \right. \quad \left. \begin{array}{l} \lim_{n \rightarrow \infty} \frac{\frac{1}{(n+1)^2}}{\frac{1}{n^2}} \rightarrow 1 \end{array} \right.$$

Ambos sublineales con  $p=0$  ✓

Ahora buscamos  $|P_n - \bar{P}| < 0,5 \times 10^{-2}$

$\Rightarrow$  a)  $\frac{1}{n} < 0,05 \rightarrow n > 20 \rightarrow$  necesitamos al menos 20 o 21 iters

b)  $\frac{1}{n^2} < 0,05 \rightarrow n^2 > 20$   
 $n > \sqrt{20} \rightarrow$  al menos 5 iters  
 $n > 4,47$

2.25 Se puede demostrar que si  $\{p_n\}_{n=1}^{\infty}$  es una secuencia dada por el método de la secante que converge a  $p$ , la solución de  $f(x) = 0$ , entonces existe una constante  $C$  tal que

$$|p_{n+1} - p| \approx C|p_n - p||p_{n-1} - p|.$$

Asumo que  $\{p_n\}$  converge con orden  $\alpha$ , es decir, que

$$\lim_{n \rightarrow \infty} \frac{|p_{n+1} - p|}{|p_n - p|^{\alpha}} = \lambda,$$

con  $\lambda > 0$  una constante, utilice esta información para mostrar que  $\alpha = \frac{1+\sqrt{5}}{2}$ . Ayuda: la suposición sobre el orden de convergencia quiere decir que puede tomar  $|p_{n+1} - p| \approx \lambda|p_n - p|^{\alpha}$ .

Arrancamos con

$$\lim_{n \rightarrow \infty} \frac{|p_{n+1} - p|}{|p_n - p|^{\alpha}} = \lambda$$

en error en n

$\Rightarrow$  entendemos que si

$$e_{n+1} = \lambda e_n^{\alpha} \Rightarrow e_n = \lambda e_{n-1}^{\alpha}$$

Por lo que

$e_{n+1} = \lambda^{\alpha+1} e_{n-1}^{\alpha^2}$

Reemplazamos en  $\underline{e_{n+1}} \approx C \underline{e_n} \underline{e_{n-1}}$

$$\lambda^{\alpha+1} e_{n-1}^{\alpha^2} \approx C \lambda^{\alpha} e_{n-1}^{\alpha} e_{n-1}$$

Si tomamos  $C$  como  $\lambda^\alpha$  ya que es una cte positiva

$$\Rightarrow \cancel{\lambda^{\alpha+1}} e_{n-1}^{\alpha^2} \approx \cancel{\lambda^\alpha} \lambda e_{n-1}^{\alpha+1}$$

Igualo  $\alpha$

$$\alpha^2 \approx \alpha + 1 <$$

$$\alpha = \frac{1 + \sqrt{5}}{2}$$

$$\cancel{\alpha = \frac{1}{2} - \frac{\sqrt{5}}{2}} < 0$$