UNIVERSIDAD DISTRITAL
FRANCISCO JOSÉ DE CALDAS

# Taller: Métodos para resolver ecuaciones lineales


Néstor Heli Aponte Ávila

Métodos Númericos

Cód. 20182167052

2022 - I


31 de mayo de 2022

## 1. Ejercicios

1. Complete los datos e indique que tipo de error se presenta en cada situación.

   *a)* $\dfrac{\sin(\frac{\pi}{4}+0.00001)-\sin(\frac{\pi}{4})}{0.00001} = \dfrac{0.70711385222-0.70710678119}{0.00001} \approx (0.7071032456451575{+}0j)$

```python

# Language = Python
from cmath import pi,sin

print(sin(pi/4+0.00001))
print(sin(pi/4))

print((sin(pi/4+0.00001)-sin(pi/4))/0.00001)

```

   - $\sin(\frac{\pi}{4}{+}0.00001) = (0.7071138522190039{+}0j) \approx 0.70711385222 \longrightarrow$ Redondeo
   - $\sin(\frac{\pi}{4}) = (0.7071067811865475{+}0j) \approx 0.70710678119 \longrightarrow$ Redondeo

   *b)* $\dfrac{\ln(2+0.00005)-\ln(2)}{0.00005} = \dfrac{0.69317218025-0.69314718056}{0.00005} \approx (0.49999375010267855{+}0j)$

```python

# Language = Python
from cmath import log

print(log(2+0.00005))
print(log(2))

print((log(2+0.00005)-log(2))/0.00005)

```

   - $\ln(2{+}0.00005) = (0.6931721802474504{+}0j) \approx 0.69317218025 \longrightarrow$ Redondeo
   - $\ln(2) = (0.6931471805599453{+}0j) \approx 0.69314718056 \longrightarrow$ Redondeo

2. Sea $f(x) = (x+2)(x+1)x(x-1)^3(x-2)$. ¿A cuál cero de $f$ converge el método de bisección en los siguientes intervalos?

   *a)* $[-3, 2.5]$

```python

bisection(-3,2.5,lambda x: (x+2)*(x+1)*x*((x-1)**3)*(x-2),0.00001)

```

```
[yoriichinara@MANJARO Functions]$ /bin/python bisection.py
Vector de puntos medios:  [-0.25, 1.125, 1.8125, 2.15625, 1.984375, 2.0703125, 2.02734375, 2.005
859375, 1.9951171875, 2.00048828125, 1.997802734375, 1.9991455078125, 1.99981689453125, 2.000152
587890625, 1.9999847412109375, 2.0000686645507812, 2.0000267028808594, 2.0000057220458984, 1.999
995231628418, 2.000000476837158]
Iteraciones: 20
c:  2.000000476837158
f(c):  1.1444114079512432e-05
Error:  0.000005245208740234375
[yoriichinara@MANJARO Functions]$
```

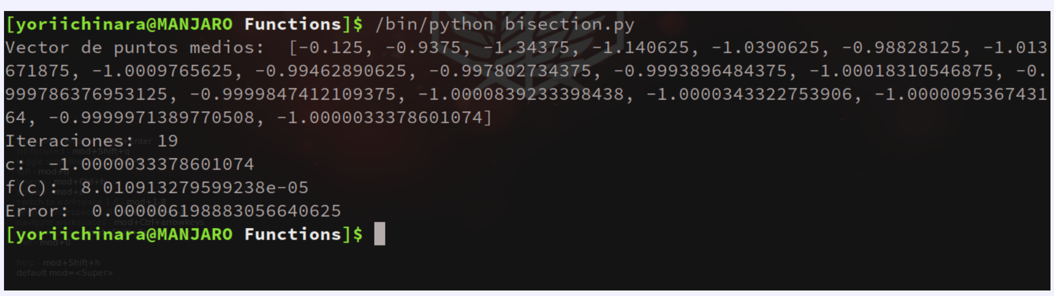
*b*) $[-2.5, 3]$

```python
bisection(-2.5,3,lambda x: (x+2)*(x+1)*x*((x-1)**3)*(x-2),0.00001)
```

```
[yoriichinara@MANJARO Functions]$ /bin/python bisection.py
Vector de puntos medios:  [0.25, -1.125, -1.8125, -2.15625, -1.984375, -2.0703125, -2.02734375,
-2.005859375, -1.9951171875, -2.00048828125, -1.997802734375, -1.9991455078125, -1.9998168945312
5, -2.000152587890625, -1.9999847412109375, -2.0000686645507812, -2.0000267028808594, -2.0000057
220458984, -1.999995231628418, -2.000000476837158]
Iteraciones: 20
c:  -2.000000476837158
f(c):  -0.0001029969612319075
Error:  0.000005245208740234375
[yoriichinara@MANJARO Functions]$
```

*c*) $[-1.75, 1.5]$

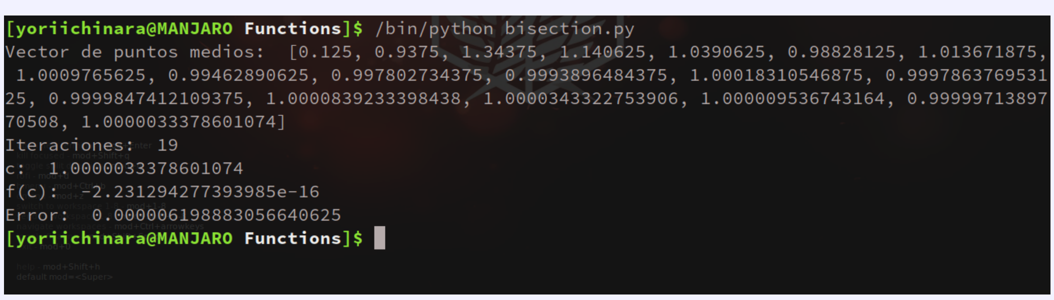
```python
bisection(-1.75,1.5,lambda x: (x+2)*(x+1)*x*((x-1)**3)*(x-2),0.00001)
```



```
[yoriichinara@MANJARO Functions]$ /bin/python bisection.py
Vector de puntos medios:  [-0.125, -0.9375, -1.34375, -1.140625, -1.0390625, -0.98828125, -1.013
671875, -1.0009765625, -0.99462890625, -0.997802734375, -0.9993896484375, -1.00018310546875, -0.
999786376953125, -0.9999847412109375, -1.0000839233398438, -1.0000343322753906, -1.0000095367431
64, -0.9999971389770508, -1.0000033378601074]
Iteraciones: 19
c:  -1.0000033378601074
f(c):  8.010913279599238e-05
Error:  0.000006198883056640625
[yoriichinara@MANJARO Functions]$
```

*d*) $[-1.5, 1.75]$

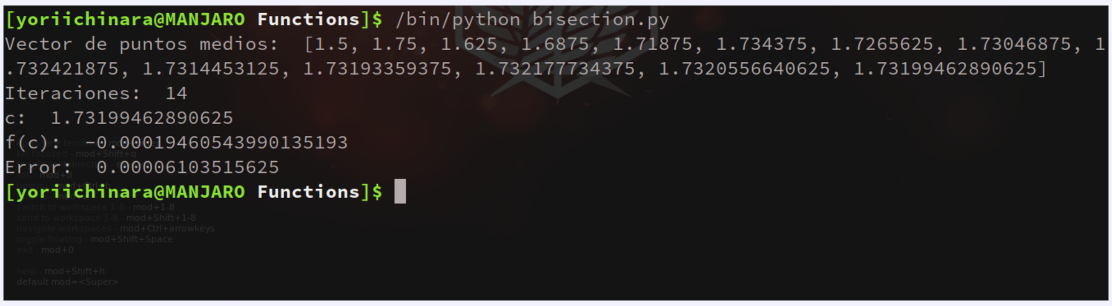
```python
bisection(-1.5,1.75,lambda x: (x+2)*(x+1)*x*((x-1)**3)*(x-2),0.00001)
```



```
[yoriichinara@MANJARO Functions]$ /bin/python bisection.py
Vector de puntos medios:  [0.125, 0.9375, 1.34375, 1.140625, 1.0390625, 0.98828125, 1.013671875,
 1.0009765625, 0.99462890625, 0.997802734375, 0.9993896484375, 1.00018310546875, 0.9997863769531
25, 0.9999847412109375, 1.0000839233398438, 1.0000343322753906, 1.000009536743164, 0.99999713897
70508, 1.0000033378601074]
Iteraciones: 19
c:  1.0000033378601074
f(c):  -2.231294277393985e-16
Error:  0.000006198883056640625
[yoriichinara@MANJARO Functions]$
```

3. Encuentre una aproximación a $\sqrt{3}$ correcta con una exactitud de $10^{-4}$ por medio del algoritmo de la bisección. (Considere $f(x) = x^2 - 3$).

```python
bisection(1,2,lambda x: x**2-3,0.0001)
```




4. Sea $\{P_n\}$ la sucesión definida por $p_n = \sum_{k=1}^{n} \frac{1}{k}$. Demuestre que $p_n$ diverge aun cuando

$$\lim_{n\to\infty} p_n - p_{n-1} = 0$$

*Demostración.*

- $\lim\limits_{n\to\infty} p_n - p_{n-1} = \lim\limits_{n\to\infty} \sum_{k=1}^{n} \frac{1}{k} - \sum_{k=1}^{n-1} \frac{1}{k} = \lim\limits_{n\to\infty} \frac{1}{n} = 0$
- Monótonia

$$\sum_{k=1}^{n+1} \frac{1}{k} = \sum_{k=1}^{n} \frac{1}{k} + \frac{1}{n+1}$$
$$p_{n+1} = p_n + \frac{1}{n+1}$$
$$p_n < p_{n+1}$$

- Sin embargo la sucesión no es acotada, para ello considerese la subsucesión $\{S_n\}$ como...

$$s_n = \sum_{k=1}^{2^n} \frac{1}{k}$$

Notesé que cada término de esa sucesión cumple que

$$\begin{aligned} s_n = \sum_{k=1}^{2^n} \frac{1}{k} &= 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8} + \cdots + \frac{1}{2^n} \\ &\geq 1 + \frac{1}{2} + \left(\frac{1}{4} + \frac{1}{4}\right) + \left(\frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8}\right) + \cdots + \left(\frac{1}{2^n} + \cdots + \frac{1}{2^n}\right) \\ &\geq 1 + 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 4 \cdot \frac{1}{8} + \cdots + 2^{n-1} \cdot \frac{1}{2^n} \\ &\geq 1 + \left(\frac{1}{2} + \frac{1}{2} + \frac{1}{2} + \cdots + \frac{1}{2}\right) \\ &\geq 1 + n \cdot \frac{1}{2} = 1 + \frac{n}{2} \end{aligned}$$

De allí que $\lim\limits_{n\to\infty} s_n \geq \lim\limits_{n\to\infty} 1 + \frac{n}{2} = \infty$, por lo cual $\{S_n\}$ diverge al infinito positivo y consiguientemente también lo hará su sucesión madre.

$$\therefore \lim_{n\to\infty} \{P_n\} = \infty$$

$\square$

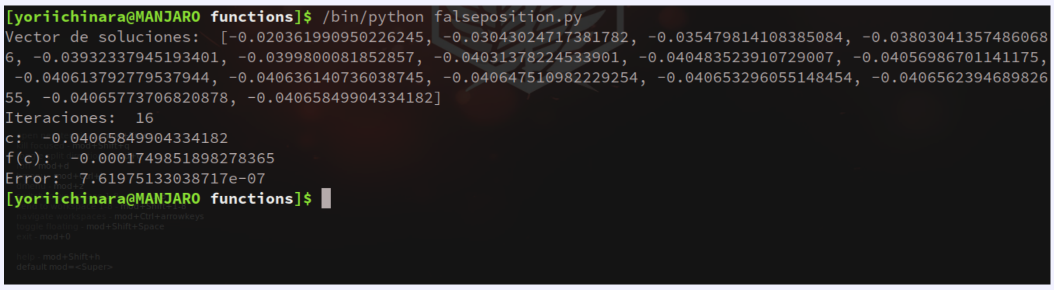
5. El polinomio $f(x) = 230x^4 + 18x^3 + 9x^2 - 221x - 9$ tiene dos ceros reales, uno en $[-1, 0]$ y el otro en $[0, 1]$. Trate de aproximar uno de estos ceros con una exactitud de $10^{-6}$ por medio de:

- El método de la posición falsa
- El método de la secante
- El método de Newton

Utilice los extremos de cada intervalo como aproximaciones iniciales en los dos primeros métodos y los intervalos como semilla para el último método.
Ponga en una sola tabla las tres sucesiones obtenidas, y el número de iteraciones que se realizaron en cada caso. Escriba una conclusión sobre la eficacia de los tres métodos para este ejemplo.

- Método de la posición falsa.

```

falseposition(-1,0,lambda x: 230*x**4+18*x**3+9*x**2-221*x-9,10**(-6))

```

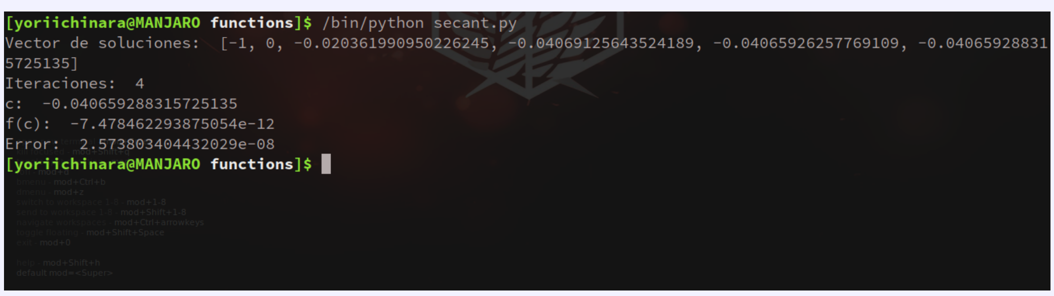



- Método de Newton-Rhapson.

```python

newrap(0,lambda x: 230*x**4+18*x**3+9*x**2-221*x-9,lambda x: 920*x**3+54*x**2+18*x
    -221,10**(-6))

```


- Método de la Secante.

```python

secant(-1,0,lambda x: 230*x**4+18*x**3+9*x**2-221*x-9,10**(-6))

```




| It | falseposition() | newrap() | secant() |
|---|---|---|---|
| 1 | -0.020361990950226245 | -0.04072398190045249 | -0.020361990950226245 |
| 2 | -0.03043024717381782 | -0.0406592884873237 | -0.04069125643524189 |
| 3 | -0.035479814108385084 | -0.04065928831575886 | -0.04065926257769109 |
| 4 | -0.038030413574860686 | | -0.040659288315725135 |
| 5 | -0.03932337945193401 | | |
| 6 | -0.0399800081852857 | | |
| 7 | -0.04031378224533901 | | |
| 8 | -0.040483523910729007 | | |
| 9 | -0.04056986701141175 | | |
| 10 | -0.040613792779537944 | | |
| 11 | -0.040636140736038745 | | |
| 12 | -0.040647510982229254 | | |
| 13 | -0.040653296055148454 | | |
| 14 | -0.040656239468982655 | | |
| 15 | -0.04065773706820878 | | |
| 16 | -0.04065849904334182 | | |

Cuadro 1: Comparativa convergencia

**Reflexión.** El método de la posición falsa es bueno conocerlo, saber como funciona pero llevado a la practica debería ser nuestra última opción, ya que resulta bastante evidente que no da la talla de ninguna manera a los otros dos. En cuanto a estos otros, son mucho mas efectivos y no se sacan tanta diferencia como uno pensaría, en este caso la derivada no era nada del otro mundo pero en un hipotetico donde si lo sea, el método de la secante se presenta como una opción muy viable.

## 2. Programas en Python

```python
# Metodo de biseccion de Bolzano

# Librerias necesarias
import numpy as np

# Colorsitos para las salidas por consola
class bcolors:
    OK = '\033[92m' #GREEN
    WARNING = '\033[93m' #YELLOW
    FAIL = '\033[91m' #RED
    RESET = '\033[0m' #RESET COLOR

# [a,b] Es el intervalo inicial con el que trabajo
# fx: Expresion matematica de mi funcion (requiere anteponer lambda x: ...)
# fx(x0): evalua el polinomio en x0
# tol: error que tolera
# error: dimension del intervalo que contiene la raiz (error)
# c: la mejor aproximacion que tengo a mi raiz
def bisection(a,b,fx,tol):

    error = abs(b-a) # Estimacion del error
    S = [] # Vector donde recojo todas mis soluciones 'c'
    contador = 0 # Contador para el numero de iteraciones

    while tol < error: # Mientras el error no cumpla con la tolerancia
        fa = fx(a)
        fb = fx(b)

        if 0 < np.sign(fa)*np.sign(fb):
            print(bcolors.WARNING + "No hay condiciones")
            break

        c = (a+b)/2
        S.append(c)
        fc = fx(c)

        if np.sign(fa)*np.sign(fc)<0: # Analisis de signos
            b = c
        else:
            a = c

        error = abs(b-a)
        contador += 1

    print(bcolors.RESET + 'Vector de puntos medios: ',S,
        '\nIteraciones: ',contador,
        '\nc: ', c,
        '\nf(c): ',fx(c),
        '\nError: ', error
    )

bisection(1,2,lambda x: x**2-3,0.0001) # Ejemplo
```

Listing 1: Método de Bisección

```python
# Metodo de la posicion falsa

# Librerias necesarias
import numpy as np

# Colorsitos para las salidas por consola
class bcolors:
    OK = '\033[92m' #GREEN
    WARNING = '\033[93m' #YELLOW
    FAIL = '\033[91m' #RED
    RESET = '\033[0m' #RESET COLOR

# [a,b] Es el intervalo inicial con el que trabajo
# fx: Expresion matematica de mi funcion (requiere anteponer lambda x: ...)
# fx(x0): evalua el polinomio en x0
# tol: error que tolera
# error: dimension del intervalo que se recorta
# c: corte de la recta y mejor aproximacion al cero
def falseposition(a,b,fx,tol):

    error = abs(b-a)
    S = [] # Vector que recoge mis soluciones
    contador = 0 # Contador de iteraciones

    while tol < error: # Mientras la tolerancia sea menor que el error
        fa = fx(a)
        fb = fx(b)

        if 0 < np.sign(fa)*np.sign(fb):
            print(bcolors.WARNING + "No hay condiciones")
            break

        c = b - (fb*(a-b)/(fa - fb)) # Formula
        S.append(c)
```

```
        fc = fx(c)

        if np.sign(fa)*np.sign(fc) < 0: # Analisis de signos
            error = abs(b-c) # El error se mide en el intervalo que estoy recortando
            b = c
        else:
            error = abs(c-a)
            a = c

        contador += 1

    print(bcolors.RESET + 'Vector de soluciones: ',S,
        '\nIteraciones: ',contador,
        '\nc: ', c,
        '\nf(c): ',fx(c),
        '\nError: ', error
    )

falseposition(-1,0,lambda x: 230*x**4+18*x**3+9*x**2-221*x-9,10**(-6))
```

Listing 2: Método de la posición falsa

```
# Metodo de Newton-Rhapson

# Librerias necesarias
import numpy as np

# Colorsitos para las salidas por consola
class bcolors:
    OK = '\033[92m' #GREEN
    WARNING = '\033[93m' #YELLOW
    FAIL = '\033[91m' #RED
    RESET = '\033[0m' #RESET COLOR

# a0: Punto inicial
# ai: Siguiente punto
# fx: Expresion matematica de mi funcion (requiere anteponer lambda x: ...)
# dfx: Derivada de fx
# fx(x0): evalua el polinomio en x0
# tol: error que tolera
# error: distancia a la raiz (error)
# c: solucion
# f(c): funcion evaluada en la solucion encontrada
def newrap(a0,fx,dfx,tol):
    S = [a0] # Vector que recoge las soluciones
    error = 2*tol # Un error inicial para darle condiciones al ciclo while
    contador = 0 # Para contar el numero de iteraciones

    while tol < error:
        a0 = S[-1]
        ai = a0 - (fx(a0)/dfx(a0))
        S.append(ai)
        error = abs(ai - a0)
        contador += 1

    c = S[-1]

    print(bcolors.RESET + 'Vector de soluciones: ',S,
        '\nIteraciones: ',contador,
        '\nc: ', c,
        '\nf(c): ',fx(c),
        '\nError: ', error
    )

newrap(0,lambda x: 230*x**4+18*x**3+9*x**2-221*x-9,lambda x: 920*x**3+54*x**2+18*x
    -221,10**(-6))
```

Listing 3: Método de Newton-Rhapson

```
# Metodo de la Secante

# Librerias necesarias
import numpy as np

# Colorsitos para las salidas por consola
class bcolors:
    OK = '\033[92m' #GREEN
    WARNING = '\033[93m' #YELLOW
    FAIL = '\033[91m' #RED
    RESET = '\033[0m' #RESET COLOR

# a1,a0: Valores iniciales
# ai: Siguiente punto
# fx: Expresion matematica de mi funcion (requiere anteponer lambda x: ...)
# fx(x0): evalua el polinomio en x0
# tol: error que tolera
# error: distancia a la raiz (error)
# c: solucion
# f(c): funcion evaluada en la solucion encontrada
def secant(a0,a1,fx,tol):
    S = [a0,a1] # Vector que recoge las soluciones
```

```python
    error = 2*tol # Un error inicial para darle condiciones al ciclo while
    contador = 0 # Para contar el numero de iteraciones

    while tol < error:
        a0 = S[-2]
        a1 = S[-1]
        ai = a1 - ((fx(a1)*(a1-a0))/(fx(a1)-fx(a0)))
        S.append(ai)
        error = abs(ai - a1)
        contador += 1

    c = S[-1]

    print(bcolors.RESET + 'Vector de soluciones: ',S,
        '\nIteraciones: ',contador,
        '\nc: ', c,
        '\nf(c): ',fx(c),
        '\nError: ', error
    )

secant(-1,0,lambda x: 230*x**4+18*x**3+9*x**2-221*x-9,10**(-6))
```

Listing 4: Método de la Secante

# Referencias


[1] Jhon H. Mathews - Kurtis D. Fink, *Métodos Númericos con MATLAB*, Pretince Hall, (2000)

[2] ESPOL, *Métodos Númericos - Curso con Python*, Guayaquil - Ecuador, (link)