



Proyecto integrador etapa 1,2 y 3

MÉTODOS NUMÉRICOS

DOCENTE Dr. EDSON
MICHAEL JIMENEZ DAMIAN

ELABORADO POR:

- CESAR AUGUSTO MARIZ PLASCENCIA.
- CARLOS AGUSTIN CARRILLO FLORES.
- JUSTINO DANIEL RODRÍGUEZ MURRAY.
- JOSE EMILIANO JAUREGUI GUZMAN.
- NAYELLI BERNAL CORTES.
- ABIGAIL ORTIZ LAGOS

13-October-2024



A blue wireframe cube is shown, representing a 3D object. It has solid lines for the visible edges and dashed lines for the hidden edges, illustrating the concept of a 3D object in a 2D representation.

-



INTRODUCCIÓN

Esta actividad consiste en aplicar los conocimientos adquiridos a lo largo del curso. Para llevar a cabo este Proyecto se toman como referente actividades elaboradas previamente, lo que garantiza la transversalidad de los contenidos revisados para fortalecer el desarrollo de competencias.

OBJETIVO

El objetivo del Proyecto integrador es programar los principales métodos numéricos para la solución de sistemas de ecuaciones lineales de una variable, así como de derivación e integración identificando las ventajas y desventajas de cada uno que permitan determinar soluciones viables mediante el planteamiento de modelos matemáticos exactos y precisos.

II. Programación de los métodos de Jacobi y Gauss-Seidel en una aplicación

2.1 Conceptualización

2.2 Casos prácticos

PROYECTO INTEGRADOR

ETAPA 1

Objetivo

El objetivo del Proyecto integrador es programar los principales métodos numéricos para la solución de sistemas de ecuaciones lineales de una variable, así como de derivación e integración identificando las ventajas y desventajas de cada uno que permitan determinar soluciones viables mediante el planteamiento de modelos matemáticos exactos y precisos.

1.Programación de los métodos de bisección y Newton Raphson en una aplicación

1.1 Conceptualización

Reproduce y completa el siguiente cuadro comparativo en el que describas los elementos sustantivos de cada método numérico.

Método numérico	¿En qué consiste?	Ventajas	Desventajas
Bisección	Es un algoritmo de búsqueda de raíces que trabaja dividiendo el intervalo a la mitad y seleccionando el subintervalo que tiene la raíz.	<ul style="list-style-type: none"> • Funciona para ecuaciones algebraicas y trascendentes, pero se recomienda utilizarlo después 	<ul style="list-style-type: none"> • Converge muy lentamente. • Permite encontrar solo una raíz, aunque existan más en el intervalo.

		<p>de un análisis gráfico.</p> <ul style="list-style-type: none"> • Es siempre convergente. • Se puede establecer el límite de error 	<ul style="list-style-type: none"> • No puede determinar raíces complejas.
Regla falsa	<p>Método iterativo de resolución numérica de ecuaciones no lineales. El método combina el método de bisección y el método de la secante.</p>	<ul style="list-style-type: none"> • Se aproxima más rápido a la raíz. • Tiene similitud con el método de secante. • Alternativa basada en una visualización gráfica. • Se estima mediante la semejanza de triángulos. 	<ul style="list-style-type: none"> • Convergencia muy lenta a la solución. • Proceso iterativo, que ocasiona que uno de sus extremos tiende a no modificarse. • No se puede prever el número de iteraciones necesarias.
Sustitución sucesiva	<p>El método de sustitución consiste en despejar una incógnita de una de las ecuaciones y sustituir en la otra ecuación el valor hallado.</p>	<ul style="list-style-type: none"> • Facilidad para programarlo • Para ciertos tipos de problemas que aparecen en Ingeniería Química este método es muy adecuado. 	<ul style="list-style-type: none"> • No converge en muchos casos • En otros la convergencia es muy lenta.
Newton Raphson	– Se utiliza para calcular los ceros	<ul style="list-style-type: none"> • Puede converger a la raíz de una 	<ul style="list-style-type: none"> • No converger si la estimación inicial.

	<p>de una función real de variable real. Su simplicidad formal y su rapidez de convergencia hacen que, con frecuencia, sea el primer algoritmo a considerar para esta tarea.</p>	<p>función rápidamente.</p> <ul style="list-style-type: none"> • Maneja funciones de cualquier complejidad. • Utiliza la derivada de la función para aproximar la raíz, lo que puede proporcionar una estimación más precisa de la raíz en comparación con otros métodos que no utilizan derivadas. 	<p>• Requiere el cálculo de derivadas, lo que puede resultar difícil para algunas funciones.</p>
Secante	<p>Es un algoritmo que busca la raíz de una función. Podría considerarse como una variante del método Newton-Raphson por todas las similitudes que tienen, ambos son métodos abiertos, funcionan a partir</p>	<p>Se puede obtener cuando la ecuación es demasiado compleja para obtener una derivada.</p>	<p>La velocidad de convergencia es mas lenta que la de Newton-Raphson. No se asegura que la Primera aproximación es cercana a la raíz que pudiera ser indice de divergencia.</p>

	de una fórmula que proviene de la pendiente de una gráfica		
--	--	--	--

1.2 Métodos abiertos

- Describe brevemente en qué consisten los métodos numéricos abiertos y establece cuáles métodos de los que aparecen en el cuadro comparativo están representados por esta categoría.

Se basan en fórmulas que requieren de un sólo valor de X_0 de un par de ellas, pero que no necesariamente encierran la raíz. En algunas ocasiones divergen o se alejan de la raíz a medida que crece el número de interacciones. Sin embargo, cuando los métodos convergen, lo hacen mucho más rápido que los métodos que usan intervalos, algunos métodos son:

- Método de Newton-Raphson.
- Método de la Secante.

- Investiga algunas aplicaciones o problemas que se utilizan aplicando este tipo de métodos.

Método de Newton-Raphson. El funcionamiento se basa en predecir un nuevo valor de x en función del valor anterior de x , de esta manera, dado un nuevo valor para x_i se tiene:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

La condición de paro se base en:

$$\varepsilon_a = \frac{x_{i+1} - x_i}{x_{i+1}}, 100\%$$

1.3 Métodos cerrados

- **Describe brevemente en qué consisten los métodos numéricos cerrados y establece cuáles métodos de los que aparecen en el cuadro comparativo están representados por esta categoría**

Se necesita de dos valores iniciales para la raíz. Como su nombre lo indica, dichos valores iniciales deben “encerrar”, o estar a ambos lados de la raíz, algunos métodos son:

Método de Bisección.

Método de la Falsa Posición o Regla falsa.

- **Investiga algunas aplicaciones o problemas que se utilizan aplicando este tipo de métodos**

A partir de una función algebraica o trascendente y de un intervalo $[a, b]$ que pertenece al dominio de la función y para el cual $f(a) \cdot f(b) < 0$, lo que implica que en el intervalo $[a, b]$ existe al menos una raíz. El método consiste en bisectar el intervalo $[a, b]$:

$$x_{\%} = \frac{a + b}{2}$$

Se obtiene una aproximación a la raíz $x_{\%}$; la función se valúa en este nuevo valor y de acuerdo al signo de la función valuada en este punto, debería sustituirse uno de los extremos del intervalo de búsqueda, de tal forma que se conserve que $f(a) \cdot f(b) < 0$. De acuerdo a la geometría de la figura, la sustitución de los intervalos debería hacerse de la siguiente forma: Sea a tal que $f(a) < 0$ y b tal que $f(b) > 0$:

Si $f(x_{\%}) < 0$, entonces $x_{\%}$ sustituye a

Si $f(x_{\%}) > 0$, entonces $x_{\%}$ sustituye a b

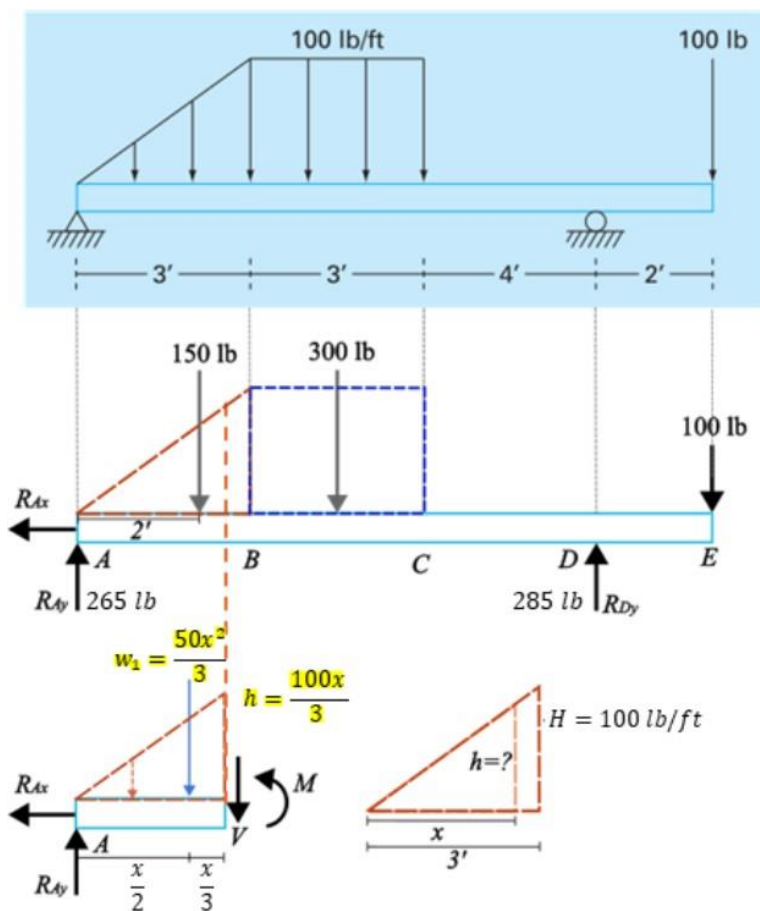
En cada iteración deberá sustituirse alguno de los límites del intervalo que contiene a la raíz. Repitiendo este proceso, el intervalo se reduce paulatinamente hasta que alguna de las aproximaciones coincide razonablemente con la raíz de la función. El proceso se detiene cuando entre la aproximación x_i y la aproximación anterior x_{i-1} se satisface un nivel de error (absoluto o relativo) preestablecido (tolerancia).

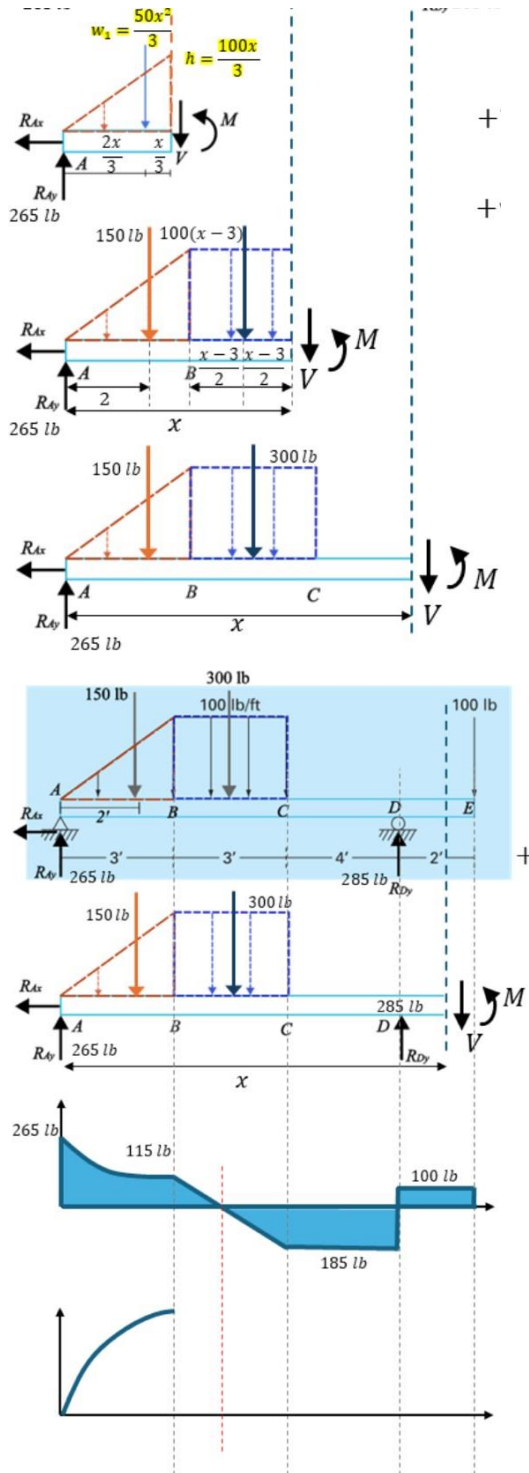
1.4 Casos prácticos

• Método de Bisección (Páginas. 140 realiza los ejercicios 5.14 y 5.15)

Método de Bisección

Se carga una viga de la manera que se aprecia en la figura. Emplee el método de bisección para resolver la posición dentro de la viga donde no hay momento





Solución:

Se determinan las reacciones considerando la viga completa como un cuerpo libre

$$\Sigma M_A = 0:$$

$$(-100 \text{ lb})(12 \text{ ft}) + (R_{py})(10 \text{ ft}) - (300 \text{ lb})(4.5 \text{ ft}) - (150 \text{ lb})(2 \text{ ft}) = 0$$

$$R_{DY} = \frac{8(100 \text{ lb})}{10 \text{ ft}} \quad R_{DY} = 285 \text{ lb} \uparrow$$

$$\uparrow \Sigma F_y = 0: R_{Ay} - 150 \text{ lb} - 300 \text{ lb} + 285 \text{ lb} - 100 \text{ lb}$$

$$R_{Ay} = 265 \text{ lb} \uparrow$$

$$\frac{100 \text{ lb}}{10 \text{ ft}} = \frac{0}{x}, \rightarrow h = \frac{100 \text{ lb} \cdot x}{265 \text{ lb}} \quad A_t = \frac{\text{base} \cdot \text{altura}}{2} = w_1$$

$$W_1 = \frac{(x) \left(\frac{100x}{265} \right)}{2}, \rightarrow w_1 = \frac{(100x^2)}{530}$$

Fuerza cortante y momento flector. Desde A hasta B ($0 < x < 3$)

$$+\uparrow \Sigma F_y = 0: 265 \text{ lb} - \frac{(100x^2)}{530} - V = 0, \quad V = 265 - \frac{(100x^2)}{530}$$

$$\Sigma M_1 = 0: -265x + \frac{(100x^3)}{1590} + M = 0, \quad M = 265x - \frac{(100x^3)}{1590}$$

Fuerza cortante y momento flector. Desde B hasta C ($3 < x < 6$)

$$+\uparrow F_y = 0: 265 \text{ lb} - 150 \text{ lb} - 100(x - 3) - V = 0, \quad V = 115 - 100(x - 3)$$

$$M_2 = 0: -265x + 150(x - 2) + 100(x - 3) \left(\frac{x-3}{2} \right) + M = 0$$

$$M = -50x^2 + 415x - 150$$

Fuerza cortante y momento flector. Desde C hasta D ($6 < x < 10$)

$$+\uparrow \Sigma F_y = 0: 265 \text{ lb} - 150 \text{ lb} - 300 - V = 0, \quad V = -185 \text{ lb}$$

$$\Sigma M_3 = 0: -265x + 150(x - 2) + 300(x - 4.5) + M = 0,$$

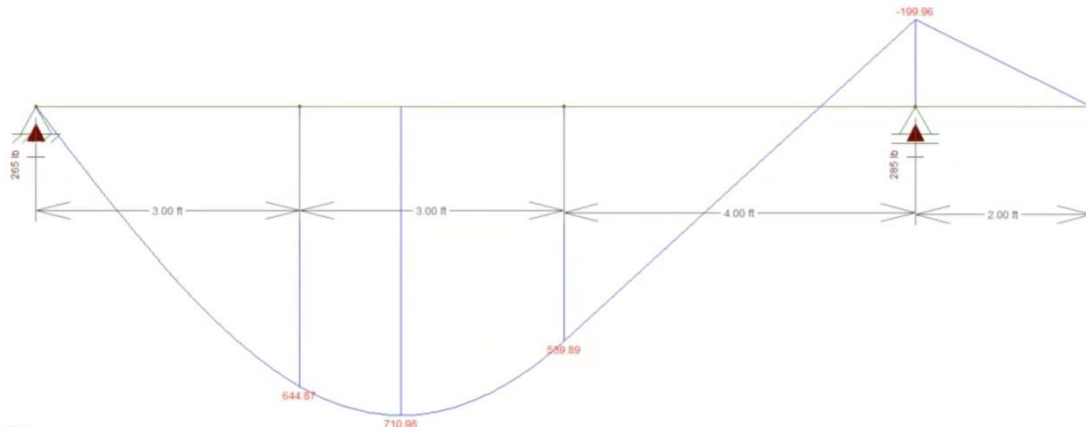
$$M = -185x + 1650$$

Fuerza cortante y momento flector. Desde D hasta E ($10 < x < 12$)

$$+\uparrow \Sigma F = 0: 265 \text{ lb} - 150 \text{ lb} - 300 \text{ lb} + 285 - V = 0, \quad V = -100 \text{ lb}$$

$$\Sigma M_3 = 0: -265x + 150(x - 2) + 300(x - 4.5) - 285(x - 10) + M = 0, \quad M = 100x - 1200$$

Dominio	Ecuación	
A-B	$V = 265 - \frac{(\%x^2}{/}$ $M = 265x - \frac{(\%x^2}{/}$	$V = 115 \text{ lb}$ $M = 645 \text{ lb*ft}$
B-C	$V = 115 - 100x(x-3)$ $M = -50x^2 + 415x - 150$	$V = -185 \text{ lb}$ $M = 540 \text{ lb*ft},$ $M_{\text{max}} = 711.125 \text{ lb*ft}$
C-D	$V = -185 \text{ lb}$ $M = -185x + 1650$	$V = -185 \text{ lb}$ $M = -200 \text{ lb*ft}$
D-E	$V = -100 \text{ lb}$ $M = 100X - 1200$	$V = -100 \text{ lb}$ $M = 0$



Paso 1: elija valores iniciales inferior, x_l y superior, x_u , que encierre la raíz, de forma tal que la función cambie de signo en el intervalo. Esto se verifica comprobando que $f(x_l)f(x_u) < 0$.

Paso 2:

$$x_r = \frac{x_l + x_u}{2} = \frac{6 + 10}{2} = 8$$

El error relativo porcentual verdadero ε_t es:

$$\varepsilon_t = \frac{\text{Valor real} - x_r}{\text{Valor real}} = \frac{8.918 - 8}{8.918} = 0.1029\%$$

Paso 3:

Calculando el producto de los valores en la función en un límite inferior y en el punto medio:

$$f(6)f(8) = (-460)(170) = -78200$$

Entonces la raíz se encuentra dentro del subintervalo inferior o izquierdo. Por lo tanto la raíz debe de estar localizada entre 8 y 9.

Por lo tanto, haga $x_u = x_r$, volviendo al paso 2 para así crear nuestra tabla de iteraciones para dar con la aproximación de la raíz.

Iteración	a	b	x_r	f(c)
1	0.00	10.00	5.00	725.00

2	5.00	10.00	7.50	262.50
3	7.50	10.00	8.75	31.25
4	8.75	10.00	9.38	-84.38
5	8.75	9.38	9.06	-26.56
6	8.75	9.06	8.91	2.34
7	8.91	9.06	8.98	-12.11
8	8.91	8.98	8.95	-4.88
9	8.91	8.95	8.93	-1.27
10	8.91	8.93	8.92	0.54
11	8.92	8.93	8.92	-0.37
12	8.92	8.92	8.92	0.09
13	8.92	8.92	8.92	-0.14
14	8.92	8.92	8.92	-0.03
15	8.92	8.92	8.92	0.03
16	8.92	8.92	8.92	0.00
17	8.92	8.92	8.92	-0.01
18	8.92	8.92	8.92	-0.01
19	8.92	8.92	8.92	-0.00
20	8.92	8.92	8.92	-0.00

La raíz aproximada es: 8.918920

La raíz aproximada es: 8.916015625

Iter	a	b	x_r	$f(x_l)f(x_u)$	Error

1	5	10	7.5	262.5	262.5
2	7.5	10	8.75	31.25	31.25
3	8.75	10	9.375	-84.375	84.375
4	8.75	9.375	9.0625	-26.5625	26.5625
5	8.75	9.0625	8.90625	2.34375	2.34375

7	8.90625	8.984375	8.9453125	-4.8828125	4.8828125
8	8.90625	8.9453125	8.92578125	-1.26953125	1.26953125
9	8.90625	8.92578125	8.916015625	0.537109375	0.537109375

```
import math
```

```
# Definir la funcion del problema del momento de flector
```

```
def f(x):
```

```
    return -185*x + 1650
```

```
# Parámetros del método
```

```
a = 6 # Límite inferior del intervalo
```

```
b = 10 # Límite superior del intervalo
```

```
tolerancia = 0.001
```

```
# Este es el método de bisección
```

```
def biseccion(a, b, tolerancia):
```

```
    # Comprobar que el intervalo es válido
```

```
    if f(a) * f(b) >= 0:
```

```
        print("El intervalo no es válido. f(a) y f(b) deben tener signos opuestos.")
```

```
        return None
```

```
# Variables iniciales
```

```
error = b - a
```

```
iteracion = 0
```

```
# Aplicar el método de bisección
```

```
while error > tolerancia:
```

```
    c = (a + b) / 2 # Punto medio
```

```
    fc = f(c)
```

```
print(f"Iteración {iteracion}: c = {c}, f(c) = {fc}")

# Verificar si hemos encontrado la raíz con la tolerancia deseada
if abs(fc) < tolerancia:
    return c

# Determinar en qué subintervalo está la raíz
if f(a) * fc < 0:
    b = c
else:
    a = c

# Actualizar el error y el contador de iteraciones
error = b - a
iteracion += 1

# Devolver la raíz aproximada
return (a + b) / 2

# Ejecutar el método de bisección
raiz = biseccion(a, b, tolerancia)
if raiz is not None:
    print(f"La raíz aproximada es: {raiz}")
```

```

1 import math
2
3 # Definir la funcion del problema del momento de flector
4 def f(x):
5     return -185*x + 1650
6
7 # Parámetros del método
8 a = 6 # Límite inferior del intervalo
9 b = 10 # Límite superior del intervalo
10 tolerancia = 0.001
11
12 # Este es el método de bisección
13 def biseccion(a, b, tolerancia):
14     # Comprobar que el intervalo es válido
15     if f(a) * f(b) >= 0:
16         print("El intervalo no es válido. f(a) y f(b) deben tener signos opuestos")
17         return None
18
19     # Variables iniciales
20     error = b - a
21     iteracion = 0
22
23     # Aplicar el método de bisección
24     while error > tolerancia:
25         c = (a + b) / 2 # Punto medio
26         fc = f(c)
27
28         print(f"Iteración {iteracion}: c = {c}, f(c) = {fc}")
29
30         # Verificar si hemos encontrado la raíz con la tolerancia deseada
31         if abs(fc) < tolerancia:
32             return c
33
34         # Determinar en qué subintervalo está la raíz
35         if f(a) * fc < 0:
36             b = c
37         else:
38             a = c
39
40     return c
41
42 # Ejecutar el método
43 raiz = biseccion(a, b, tolerancia)
44 print(f"La raíz aproximada es: {raiz}")
45
46 ...Program finished with exit code 0
47 Press ENTER to exit console.

```

Ejercicio 5.15 Por un canal trapezoidal fluye agua a una tasa de $Q=20\text{ m}^3/\text{s}$. La profundidad crítica y para dicho canal satisface la ecuación

$$0 = 1 - \frac{Q^2}{gA_c^3} B$$

Donde $g=9.81\text{ m/s}^2$, A_c = área de la sección transversal (m^2) y B = ancho del canal de la superficie (m) para este caso, el ancho y el área de la sección transversal se relaciona con la profundidad y po medio de:

$$B = 3 + y \quad \text{y} \quad A = 3y + \frac{y^2}{2}$$

Resuelva para la profundidad crítica con el uso de los métodos a) gráfico, b) bisección, y c) falsa posición. En los incisos b) y c), haga elecciones iniciales de $x_l = 0.5$ y $x_u = 2.5$, y ejecute iteraciones hasta que el error aproximado caiga por debajo del 1% o el número de interacciones supere a 10. Analice sus resultados.

A) Método gráfico

Para comenzar con la interpretación de esta función y encontrar la raíz por el método gráfico, basta con sustituir los valores de "y" que nos señala el ejercicio.

Por lo tanto, realizamos la sustitución de $y = .5$ y 2.5 así como todos los valores que ya nos otorgó el ejercicio en la función:

$$0 = 1 - \frac{Q}{gA_c} B$$

Quedando de la siguiente manera:

$$Q=20$$

$$g=9.81$$

$$A_c = 3y + \frac{y^2}{2}$$

$$B = 3 + y$$

$$1 - \frac{20}{9.81 * (3y + \frac{y^2}{2})} * 3 + y$$

Sustituimos el valor de y empezando con $.5$ como lo pide el ejercicio:

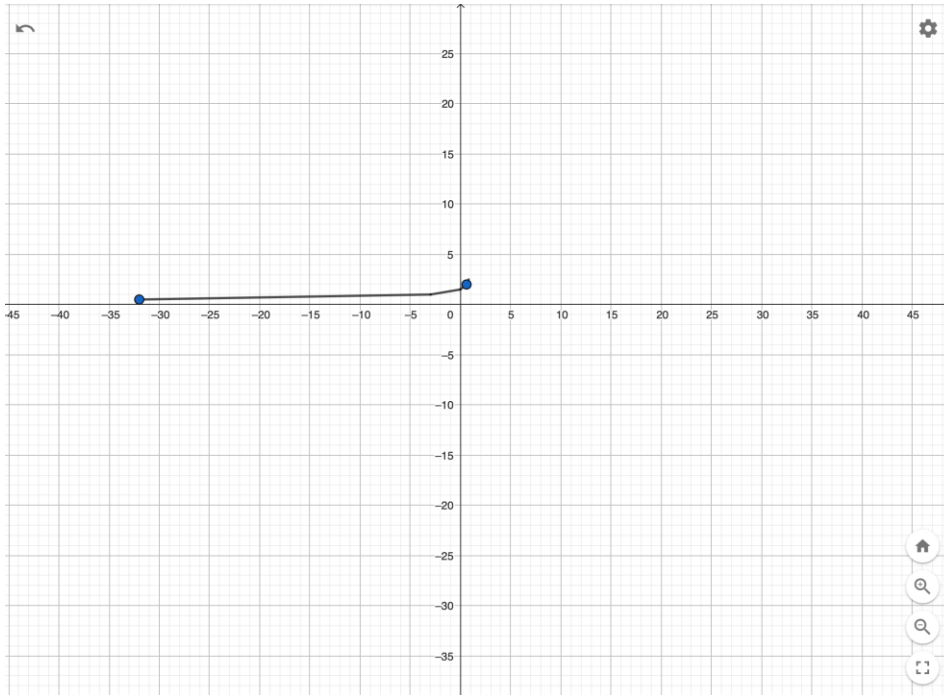
$$1 - \frac{400}{9.81 * (3 * .5 + \frac{.5^2}{2})} * 3 + .5 = -32.25$$

Esto representa el valor de x cuando y tiende a $.5$

Ahora realizamos la evaluación de y cuando vale 2.5

$$1 - \frac{400}{9.81 * (3 * 2.5 + \frac{2.5^2}{2})} * 3 + 2.5 = 0.8134$$

Con estos cálculos ya sabemos que, si hay una raíz entre estos dos números ya que hubo un cambio de signo, pero a simple vista es difícil ver donde se encuentra la raíz



Para tener un dato más exacto usando el método gráfico, realizamos la evaluación de $y = .5$ a 2.5 con un rango de $.5$ para cada evaluación, dándonos un valor más cercano a la raíz, donde usaremos un programa para la facilitar el cálculo.

El cual quedará de la siguiente manera:

$$1 - \frac{400}{9.81 * (3 * .5 + \frac{.5^2}{2})} * (3 + .5) = -32.81$$

$$1 - \frac{400}{9.81 * (3 * 1 + \frac{1^2}{2})} * (3 + 1) = -2.8041$$

$$1 - \frac{400}{9.81 * (3 * 1.5 + \frac{1.5^2}{2})} * (3 + 1.5) = -.0309$$

$$1 - \frac{400}{9.81 * (3 * 2 + \frac{2^2}{2})} * (3 + 2) = 6018$$

$$1 - \frac{400}{9.81 * (3 * 2.5 + \frac{2.5^2}{2})} * 3 + 2.5 = .8130$$

Dando las siguientes coordenadas:

x	y
-32.81	.5
-2.8041	1
-.0309	1.5
.6018	2
.8130	2.5



```
main.py
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Definimos la función
5 def f(y):
6     return 1 - (400 / (9.81 * (3 * y + (y**2) / 2)**3)) * (3 + y)
7
8 # Generamos valores de y de 0.1 en 0.1
9 y_values = np.arange(0.5, 2.6, 0.5)
10 x_values = f(y_values)
11
12 # Mostramos cada iteración
13 for y, x in zip(y_values, x_values):
14     print(f'y: {y:.2f}, x: {x:.4f}')
15
16 # Graficamos con x en el eje horizontal y y en el eje vertical
17 plt.plot(x_values, y_values, marker='o')
18 plt.title('Comportamiento de la función')
19 plt.xlabel('x')
20 plt.ylabel('y')
21 plt.grid()
22 plt.show()
23
```

input

```
y: 0.50, x: -32.2582
y: 1.00, x: -2.8041
y: 1.50, x: -0.0309
y: 2.00, x: 0.6018
y: 2.50, x: 0.8130
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
# Definimos la función
```

```
def f(y):
```

```
    return 1 - (400 / (9.81 * (3 * y + (y**2) / 2)**3)) * (3 + y)
```

```
# Generamos valores de y de 0.1 en 0.1
```

```
y_values = np.arange(0.5, 2.6, 0.5)
```

```
x_values = f(y_values)
```

```
# Mostramos cada iteración
```

```
for y, x in zip(y_values, x_values):
```

```
    print(f'y: {y:.2f}, x: {x:.4f}')
```

Graficamos con x en el eje horizontal y y en el eje vertical

```
plt.plot(x_values, y_values, marker='o')
```

```
plt.title('Comportamiento de la función')
```

```
plt.xlabel('x')
```

```
plt.ylabel('y')
```

```
plt.grid()
```

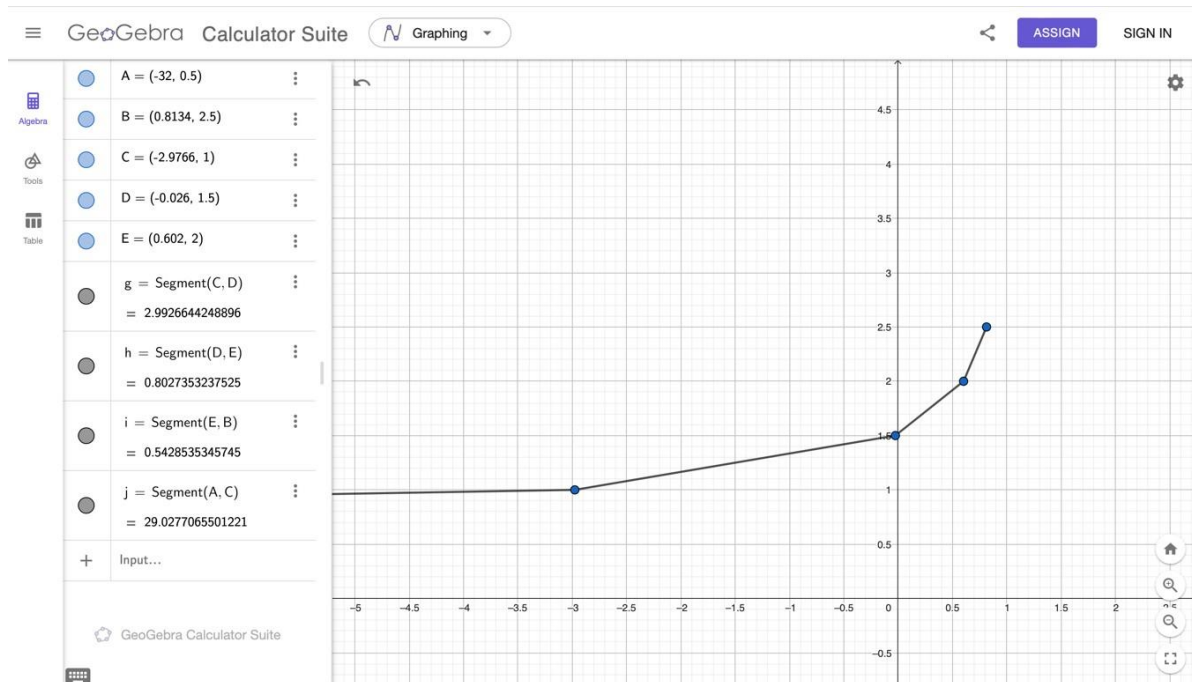
```
plt.show()
```

Podemos notar que en valor de 1.5 a 2 hay un cambio de signo, por lo que ahí estará la raíz.

Proseguimos a graficar las coordenadas obtenidas que ya fueron graficadas en Python pero que también se realizarán en GeoGebra para una comprensión más profunda.

Dando las siguientes coordenadas:

x	y
-32.81	.5
-2.8041	1
-.0309	1.5
.6018	2
.8130	2.5



Podemos observar que la raíz se encuentra en 1.5 como lo pudimos comprobar con la evaluación la que demostró que hubo un cambio de signo en 1.5 y 2, confirmando que ahí se encuentra la raíz

B) Método de Bisección

Para resolverlo por el método de bisección, anotaremos primeramente los pasos que se realizan.

Paso 1. Sacar el punto medio que es igual a $m = \frac{a+b}{2}$

Paso 2: Evaluar $f(a)$, $f(b)$ y $f(m)$

Paso 3: Determinar el nuevo intervalo

Paso 4: Calcular el error $e = \frac{b-a}{2}$

En este caso desglosamos los valores que tenemos:

$a = .05$

$b=2.5$

$i=.01$

Primeramente, proseguimos a calcular el punto medio:

$$m = \frac{.5 + 2.5}{2} = 1.5$$

Evaluamos en la función con el valor de a,b y el ultimo valor que acabamos de obtener que es m.

$$f(a) = 1 - \frac{400}{9.81 * (3 * .5 + \frac{.5}{2})} * (3 + .5) = -32.81$$

$$f(b) = 1 - \frac{400}{9.81 * (3 * 2.5 + \frac{2.5}{2})} * 3 + 2.5 = .8130$$

$$f(m) = 1 - \frac{400}{9.81 * (3 * 1.5 + \frac{1.5}{2})} * (3 + 1.5) = -.0309$$

Para determinar el nuevo intervalo tendremos que echar un vistazo a los signos, si “a” y “m” tienen los mismos signos, entonces “a” se descarta y “m” se toma como nuevo “a”.

Si “b” y “m” tienen el mismo signo, entonces “b” se descarta y “m” se toma como nuevo “b”.

En este caso tenemos que “a” y “m” tienen el mismo signo, por lo tanto “m” se toma como el nuevo “a” quedando de la siguiente manera: 1.5 que es “m” toma el lugar de “a” que era .5 y “b” queda igual 2.5, por lo tanto quedaría (1.5,-2.5) o $a = 1.5$ y $b = 2.5$

Ahora tendremos que calcular el error con el nuevo “a”

$$e = \frac{b - a}{2} \quad e = \frac{2.5 - 1.5}{2} = .5$$

Nos da un margen de error de 50 por ciento, por lo tanto, tendremos que seguir iterando hasta obtener el .01 o 1% o alcanzar las 10 iteraciones como lo pide el ejercicio.

Sacamos “m”

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

$$m = \frac{1.5 + 2.5}{2} = 2$$

Evaluamos las ecuaciones:

$$f(a) = 1 - \frac{400}{9.81 * (3 * 1.5 + \frac{1.5^2}{2})} * (3 + 1.5) = -.0309$$

$$f(b) = 1 - \frac{400}{9.81 * (3 * 2.5 + \frac{2.5^2}{2})} * (3 + 2.5) = .8130$$

$$f(m) = 1 - \frac{400}{9.81 * (3 * 2 + \frac{2^2}{2})} * (3 + 2) = .6018$$

Evaluamos:

Nos damos cuenta de que “b” y “m” tienen los mismos signos, por lo tanto, desplazamos “b” y ahora “m” será el nuevo “b”

$$a = 1.5 \quad b = 2$$

Evaluamos el error con el nuevo “b”.

$$e = \frac{b - a}{2} \quad e = \frac{2 - 1.5}{2} = .25$$

Indicativo que tendremos que seguir hasta alcanzar .01 de tolerancia en el error, alcanzar 10 iteraciones o el valor de $f(m)$ este muy cerca de 0 y arroja resultados repetidos o varían por muy pocas milésimas, esto significa que hemos encontrado la raíz.

Para no realizar todo el procedimiento tan repetitivamente, usaremos Python para encontrar la raíz, tomando los valores de $x_l = 0.5$ y $x_u = 2.5$ y una tolerancia de 1% porque así lo dicta el ejercicio.

The screenshot shows a web browser window with the URL `mycompiler.io`. The page displays a Python script for finding the root of a function using the bisection method. The script includes a function `bisection_method` and a main execution block. The output of the program is shown on the right side of the interface.

Entrada del programa

Salida del programa

Iteración	x_l	x_u	c	$f(c)$
1	0.50000	2.50000	1.50000	-0.03095
2	1.50000	2.50000	2.00000	0.60181
3	1.50000	2.00000	1.75000	0.37891
4	1.50000	1.75000	1.62500	0.20693
5	1.50000	1.62500	1.56250	0.09796
6	1.50000	1.56250	1.53125	0.03626
7	1.50000	1.53125	1.51562	0.00338
8	1.50000	1.51562	1.50781	-0.01360

La raíz es aproximadamente: 1.50781

[Execution complete with exit code 0]

```

main.py
2 return 1 - (400 / (9.81 * (3 * y + (y**2 / 2)**3)) * (3 + y)
3
4 def bisection_method(xl, xu, tol=0.01, max_iter=10):
5     if f(xl) * f(xu) >= 0:
6         print("No se puede aplicar el método de bisección.")
7         return None
8
9     iter_count = 0
10    print(f"{'Iteración':<10}{ 'xl':<10}{ 'xu':<10}{ 'c':<10}{ 'f(c)':<10}")
11
12    while iter_count < max_iter:
13        c = (xl + xu) / 2
14        error = abs((xu - xl) / 2)
15
16        # Imprimir la información de la iteración
17        print(f"{'iter_count + 1':<10}{ 'xl':<10.5f}{ 'xu':<10.5f}{ 'c':<10.5f}{ 'f(c)':<10.5f}")
18
19        if f(c) == 0 or error < tol * abs(c):
20            return c # Encontramos la raíz o cumplimos la tolerancia
21
22        if f(xl) * f(c) < 0:
23            xu = c
24        else:
25            xl = c
26
27        iter_count += 1
28
29    return (xl + xu) / 2
30
31 # Intervalo inicial
32 xl = 0.5
33 xu = 2.5
34 tolerance = 0.01
35
36 raiz = bisection_method(xl, xu, tolerance)
37 print(f"La raíz es aproximadamente: {raiz:.5f}")

```

Iteración	xl	xu	c	f(c)
1	0.50000	2.50000	1.50000	-0.03095
2	1.50000	2.50000	2.00000	0.60181
3	1.50000	2.00000	1.75000	0.37891
4	1.50000	1.75000	1.62500	0.20693
5	1.50000	1.62500	1.56250	0.09796
6	1.50000	1.56250	1.53125	0.03626
7	1.50000	1.53125	1.51562	0.00338
8	1.50000	1.51562	1.50781	-0.01360

La raíz es aproximadamente: 1.50781

...Program finished with exit code 0
Press ENTER to exit console.

def f(y):

return 1 - (400 / (9.81 * (3 * y + (y**2 / 2)**3)) * (3 + y)

def bisection_method(xl, xu, tol=0.01, max_iter=10):

if f(xl) * f(xu) >= 0:

print("No se puede aplicar el método de bisección.")

return None

iter_count = 0

print(f"{'Iteración':<10}{ 'xl':<10}{ 'xu':<10}{ 'c':<10}{ 'f(c)':<10}")

while iter_count < max_iter:

c = (xl + xu) / 2

error = abs((xu - xl) / 2)

Imprimir la información de la iteración

print(f"{'iter_count + 1':<10}{ 'xl':<10.5f}{ 'xu':<10.5f}{ 'c':<10.5f}{ 'f(c)':<10.5f}")

```

if f(c) == 0 or error < tol * abs(c):
    return c # Encontramos la raíz o cumplimos la tolerancia

if f(xl) * f(c) < 0:
    xu = c
else:
    xl = c

iter_count += 1

return (xl + xu) / 2

# Intervalo inicial
xl = 0.5
xu = 2.5
tolerance = 0.01

raiz = bisection_method(xl, xu, tolerance)
print(f"La raíz es aproximadamente: {raiz:.5f}")

la raíz nos dio el valor de 1.50781, resultado que ya habíamos calculado con las
evaluaciones y usando el metodo gráfico, pero ahora con el metodo de bisección
calculamos con una tolerancia de .01

```

C) Ahora haremos el método de la falsa posición.

Pasos para realizar el método:

1-Evaluamos $f(a)$ y $f(b)$

2-Obtener x_i con la siguiente formula:

$$xi = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

3- Determinar nuevo intervalo

4- Calcular el error

Por lo que proseguimos a realizar la actividad.

Los puntos a y b los escogeremos de 1.5 a 2.0 porque ya sabemos que hay una raíz entre estos dos puntos.

a=1.5

b=2.0

Proseguimos a evaluar f(a) y f(b)

$$Q^{\&} \\ 0 = 1 - \frac{Q^{\&}}{gA_c'} B$$

$$1 - \frac{20^{\&}}{9.81 * (3y + \frac{y^{\&}}{2})'} * 3 + y$$

$$f(a) = 1 - \frac{400}{9.81 * (3 * 1.5 + \frac{1.5^{\&}}{2})'} * 3 + 1.5 = -.0215$$

$$f(b) = 1 - \frac{400}{9.81 * (3 * 2 + \frac{2^{\&}}{2})'} * (3 + 2). = .6018$$

Con los valores que nos dieron en f(a y b) obtenemos xi

$$xi = \frac{af(b) - bf(a)}{f(b) - f(a)}$$

$$xi = \frac{1.5 * 6018 - 2 * -.0215}{.6018 - -.0215} = 1.517$$

Ahora tendremos que determinar el nuevo intervalo fijándonos en el signo de las funciones a,b y xi

$$f(xi) = 1 - \frac{400}{9.81 * (3 * 1.517 + \frac{1.517^2}{2})} * (3 + 1.517) = 0.012$$

Si “a” y “xi” sus signos son iguales, entonces tendremos que sustituir el valor de a por “xi”, el cual tomará el nuevo valor de “a”

Pero si “b” y “xi” tienen el mismo signo, entonces tendremos que sustituir el valor de “b” por “xi”, el cual será el nuevo “b”

Por lo tanto, nos damos cuenta de que el valor que se sustituirá será el de “b”, quedando en su lugar el valor de “xi”.

Ahora tendríamos que calcular el error, pero al ser la primera iteración, no será posible.

Continuamos con la segunda iteración:

Tomamos los nuevos valores:

$$a = 1.5$$

$$b = 1.517$$

1- Evaluamos las funciones de a y b:

$$f(a) = 1 - \frac{400}{9.81 * (3 * 1.5 + \frac{1.5^2}{2})} * 3 + 1.5 = -0.0215$$

$$f(b) = 1 - \frac{400}{9.81 * (3 * 1.517 + \frac{1.517^2}{2})} * 3 + 1.517 = 0.012$$

2- calculamos xi

$$xi = \frac{1.5 * .012 - 1.517 * -.0215}{0.012 - -.0215} = 1.5110$$

3- determinamos el nuevo valor evaluando f(xi)

$$f(xi) = 1 - \frac{400}{9.81 * (3 * -1.5110 + \frac{1.5110^2}{2})} * 3 + 1.5110 = 5.634.$$

3- Determinamos el nuevo intervalo:

Notamos que el valor de a y xi son iguales por lo tanto nos quedamos con el valor de -1.5110

Por lo tanto, los nuevos valores quedan:

a=1.5110 y b= 1.517

4-Calculamos el error

El error se calcula restando el valor actual de la iteración menos el valor anterior

$$|xi2 - xi1|$$

$$|1.5110 - 1.517| = -.006$$

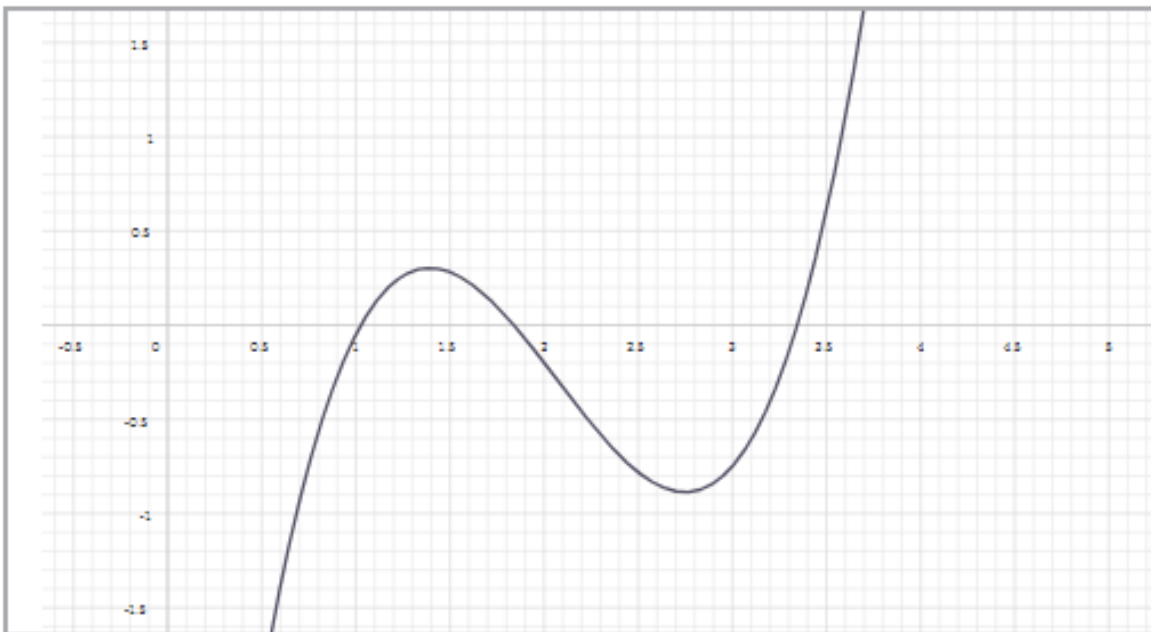
Esto quiere decir que tenemos una raíz aproximada de **1.5110** con un margen de error de .6% es decir, el valor nos salió ligeramente diferente por unas milésimas al que obtuvimos por el método de bisección de 1.50781, debido a que por este método lo hicimos con más exactitud que por bisección con un margen de error aún menor, ya que si hubiéramos continuado con un margen de error menor en bisección nos hubiera dado un valor muy similar, lo que si pudimos notar es que se obtiene la raíz mas rápidamente que por el de bisección.

Método newton- Raphson (pág. 169 realiza el ejercicio 6.26)

6.9 Dertermine la raíz real más grande de $f(x) = 0.95x^3 - 5.9x^2 + 10.9x - 6$:

- a) En forma gráfica.
- b) Con el uso del método de Newton-Raphson (tres iteraciones, $x_{i+1} = 3.5$)
- c) Con el método de la secante (tres iteraciones, $x_{i+1} = 2.5$ y $x_{i+1} = 3.5$)
- d) Por medio del método de la secante modificado (tres iteraciones, $x_{i+1} = 3.5, \delta = 0.01$).

Grafica:



Método newton- Raphson

$$f(x) = 0.95x^3 - 5.9x^2 + 10.9x - 6$$

$$f'(x) = 2.85x^2 - 11.8x + 10.9$$

$$f(3.5) = 0.95(3.5)^3 - 5.9(3.5)^2 + 10.9(3.5) - 6$$

$$\mathbf{0.60325}$$

$$f'(3.5) = 2.85(3.5)^2 - 11.8(3.5) + 10.9$$

$$\mathbf{4.5125}$$

$$\left(\frac{0.60325}{4.5125} \right) \longrightarrow X_1 = 3.5 - X_1 = 3.3656$$

$$f(3.3656) = 0.95(3.3656)^3 - 5.9(3.3656)^2 + 10.9(3.3656) - 6$$

$$= 0.071071$$

$$f(3.3656) = 2.85(3.3656)^2 - 11.8(3.3656) + 10.9$$

$$= 3.4687$$

$$\left(\frac{0.071071}{3.4687} \right) \longrightarrow X_2 = 3.3656 - X_2 = 3.3451$$

$$f(3.3451) = 0.95(3.3451)^3 - 5.9(3.3451)^2 + 10.9(3.3451) - 6$$

$$= 0.0015$$

$$f(3.3451) = 2.85(3.3451)^2 - 11.8(3.3451) + 10.9$$

$$= 3.3184$$

$$\left(\frac{0.0015}{3.3184} \right) \longrightarrow X_3 = 3.3451 - X_3 = 3.3446$$

Método de la secante

$$X_{i+1} = X_i - \frac{f(X_i)(X_i - X_1)}{f(X_i) - f(X_1)}$$

$$X_1 = 2.5 \quad f(X_1) = -0.7812$$

$$X_0 = 3.5 \quad f(X_0) = 0.6062$$

$$X_1 = 3.5 - \frac{(0.6062)(3.5 - 2.5)}{0.6062 - (-0.7812)} = 3.0630$$

$$X_2 = 3.0630 - \frac{(-0.7812)(3.0630 - 2.5)}{-0.7812 - (-0.6062)} = 3.3407$$

$$X_3 = 6.3407 - \left(\frac{0.0088 \left(\frac{6.3407}{0.0088} \right)}{6.3407 - 0.0088} \right) = 6.2849$$

$$E_A = \frac{0.0088 \left(\frac{6.3407}{0.0088} \right)}{6.2849} = 0.88\%$$

$f(x)$ en el cuarto cuadrante, asegúrese de tomar el valor negativo de la raíz cuadrada. ¿Por qué diverge la solución?

6.26 Suponga el lector que está diseñando un tanque esférico (véase la figura P6.26) de almacenamiento de agua para un poblado pequeño de un país en desarrollo. El volumen del líquido que puede contener se calcula con

$$V = \pi h^2 \frac{3R - h}{3}$$

donde V = volumen [pie³], h = profundidad del agua en el tanque [pies], y R = radio del tanque [pies].

Si $R = 3$ m, ¿a qué profundidad debe llenarse el tanque de modo que contenga 30 m³? Haga tres iteraciones del método de Newton-Raphson para determinar la respuesta. Encuentre el error relativo aproximado después de cada iteración. Observe que el valor inicial de R convergerá siempre.

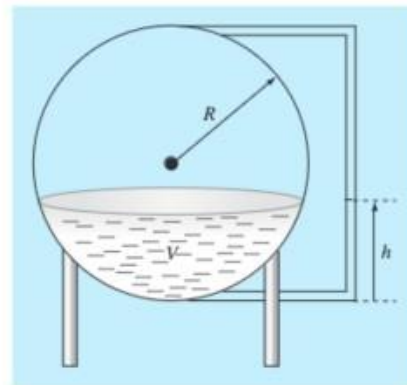


Figura P6.26

Para resolver este problema utilizando el método de Newton-Raphson, primero necesitamos definir la función ($f(h)$) que queremos igualar a cero. Dado que el volumen (V) es 30 m³, podemos escribir:

$$f(h) = \pi h^2 \frac{3R - h}{3} - 30$$

Con ($R = 3$) m, la función se convierte en:

$$f(h) = \pi h^2 \frac{3 - h}{3} - 30 \quad f(h) = \pi h^2 \left(3 - \frac{h}{3} \right) - 30$$

$$f(h) = \pi h^2 \left(3 - \frac{h}{3} \right) - 30$$

La derivada de ($f(h)$) es:

$$f'(h) = \frac{d}{dh} \left[\pi h^2 \left(3 - \frac{h}{3} \right) - 30 \right] \quad [f'(h) = \pi(6h - h^2/3)]$$

Ahora, aplicamos el método de Newton-Raphson:

$$h_{n+1} = h_n - \frac{f(h_n)}{f'(h_n)}$$

Iteración 1:

Supongamos un valor inicial ($h_0 = 1.5$) m.

$$\begin{aligned} f(h_0) &= \pi(1.5)^3 - 30 \\ f(h_0) &= \pi(5.625) - 30 \approx 17.67 - 30 = -12.33 \\ f'(h_0) &= \pi(6(1.5) - 1.5^2) = \pi(9 - 2.25) = \pi(6.75) \approx 21.21 \\ h_1 &= 1.5 - \frac{-12.33}{21.21} \approx 2.08 \end{aligned}$$

Error relativo aproximado:

$$\text{Error} = \frac{2.08 - 1.5}{2.08} \times 100 \approx 27.88\%$$

Iteración 2:

$$\begin{aligned} f(h_1) &= \pi(2.08)^3 - 30 \\ f(h_1) &\approx 31.42 - 30 = 1.42 \\ f'(h_1) &= \pi(6(2.08) - (2.08)^2) = \pi(12.48 - 4.33) = \pi(8.15) \approx 25.61 \\ h_2 &= 2.08 - \frac{1.42}{25.61} \approx 2.02 \end{aligned}$$

Error relativo aproximado:

$$\text{Error} = \frac{2.02 - 2.08}{2.02} \times 100 \approx 2.97\%$$

Iteración 3:

$$\begin{aligned} f(h_2) &= \pi(2.02)^3 - 30 \\ f(h_2) &\approx 29.82 - 30 = -0.18 \\ f'(h_2) &= \pi(6(2.02) - (2.02)^2) = \pi(12.12 - 4.08) = \pi(8.04) \approx 25.26 \end{aligned}$$

$$w_{h/} = 2.02 - \frac{-0.18}{25.26} \approx 2.03]$$

Error relativo aproximado:

$$w_{\text{Error}} = \frac{2.03 - 2.02}{2.03} \times 100 \approx 0.49\%$$

Después de tres iteraciones, la profundidad del agua en el tanque es aproximadamente 2.03 m.

Establece una breve descripción en la que hagas un contraste entre los métodos que programaste, destacando los problemas a los que te enfrentaste y cómo lo solucionaste.

En esta tarea pudimos percibir con claridad la relevancia de los errores como una guía para evaluar qué tan próximas están las iteraciones generadas de la solución al aplicar cada método. Además, adquirimos conocimientos sobre la implementación de diversos enfoques para resolver ecuaciones no lineales, así como cuándo es apropiado emplear métodos abiertos o cerrados, tomando en cuenta las ventajas y desventajas de cada uno.

El método de bisección es seguro y garantiza la convergencia al dividir intervalos donde la función cambia de signo, aunque puede ser más lento.

Ambos métodos son ampliamente aplicables en diversas disciplinas para resolver ecuaciones no lineales.

De igual manera, comprendimos la importancia del uso de lenguajes de programación para la resolución de problemas matemáticos extensos, tediosos o complejos, y las ventajas que estos ofrecen en comparación con software menos especializado. Durante el desarrollo de los casos prácticos en este proyecto integrador, también pudimos observar cómo aplicar los métodos abiertos y cerrados en la resolución de problemas de ingeniería, obteniendo las ecuaciones necesarias para la aplicación de dichos métodos.

Además, notamos que algunos métodos resultan más eficientes o breves que otros, dependiendo del contexto, lo cual se refleja en el número de iteraciones generadas según el método empleado.

Referencias.

Chapra, S. C., & Canale, R. P. (2015). *Numerical Methods for Engineers* (7th ed.). McGraw-Hill Education.

Mathews, J. H., & Fink, K. D. (2004). *Numerical Methods Using MATLAB* (4th ed.). Prentice Hall.

Rao, S. S. (2017). *Applied Numerical Methods for Engineers and Scientists* (4th ed.). Pearson.

Chapra, S. C. (2018). *Applied Numerical Methods with MATLAB for Engineers and Scientists* (4th ed.). McGraw-Hill Education.

PROYECTO INTEGRADOR

ETAPA 2

2.1 Conceptualización

Reproduce y completa el siguiente cuadro comparativo en el que describas los elementos sustantivos de los siguientes métodos:

MÉTODO / ASPECTO	JACOBI	GAUSS-SEIDEL
¿EN QUÉ CONSISTE?	<p>Un método iterativo con el cual se resuelve el sistema lineal $Ax=b$ comienza con una aproximación inicial $x^{(0)}$ a la solución x y genera una sucesión de vectores $x^{(k)}$ que converge a x. Los métodos iterativos traen consigo un proceso que convierte el sistema $Ax=b$ en otro equivalente de la forma $x=Tx+c$ para alguna matriz fija T y un vector c.</p>	<p>Este método es una versión acelerada de Jacobi. En el cual es necesario contar con un vector aproximado completo para proceder a la sustitución en las ecuaciones de recurrencia y obtener una nueva aproximación. En el método de Gauss-Seidel se propone ir sustituyendo los nuevos valores de la aproximación siguiente conforme se vayan obteniendo sin esperar a tener un vector completo. De esta forma se acelera la convergencia.</p>

VENTAJAS	<p>Métodos más simples para resolver sistemas lineales con estabilidad numérica y (en muchos casos) soluciones inexactas.</p> <p>Converge para cualquier vector inicial si la matriz es diagonalmente dominante.</p>	<p>A pesar de no tener la diagonal dominante, se puede hacer y posibilidades de su resultado. En su programación se puede llegar a un valor aproximado máximo o resultado exacto.</p>
DESVENTAJAS	<p>Baja estabilidad numérica. El elemento de la diagonal principal de cada ecuación debe ser mayor en valor absoluto.</p>	<p>No siempre converge a una solución.</p> <p>A veces converge muy lentamente.</p> <p>Es difícil definir el margen mínimo por el que ese coeficiente debe dominar a los otros para asegurar la convergencia</p>

2.2 Casos prácticos

Una campaña de electrónica produce transistores, resistores y chips de computadora. Cada transistor requiere cuatro unidades de cobre, una de zinc y dos de vidrio. Cada resistor requiere tres, tres y una unidad de dichos materiales, respectivamente, y cada chip de computadora requiere dos, una y tres unidades de los materiales, respectivamente. En forma de tabla, esta información queda así:

COMPONENTE	COBRE	ZINC	VIDRIO
TRANSISTORES	4	1	2
RESISTORES	3	3	1
CHIPS DE COMPUTADORA	2	1	3

Los suministros de estos materiales varían de una semana a la otra, de modo que la compañía necesita determinar una corrida de producción diferente cada semana. Por ejemplo, cierta semana las cantidades disponibles de los materiales son 960 unidades de cobre, 510 unidades de zinc y 610 unidades de vidrio. Platee el sistema de ecuaciones que modela la corrida de producción, la información que se da en este capítulo sobre la solución de ecuaciones algebraicas lineales para resolver cual es el número de transistores, resistores y chips de computadora por manufacturar esta semana.

Primeramente, lo resolveremos por el método de Jacobi.

Como primer paso tendremos que plantear el problema de la siguiente manera:

x = transistores

y = resistores

z = Chips de computadoras

Por lo tanto, la matriz queda de la siguiente manera:

$$4x + 3y + 2z = 960$$

$$x + 3y + z = 510$$

$$2x + y + 3z = 610$$

Para garantizar que la solución por el método de Jacobi pueda converger tenemos que revisar que la matriz sea estrictamente dominante.

Para eso tenemos que identificar si a_1 es mayor a la suma en su fila, lo mismo para a_2 y a_3

En el primer caso notamos que no cumple la condición porque $4 < 3+2$ (No cumple)

En la segunda fila, cumple la condición porque $3 > 1+1$ (Cumple)

Y en la tercera no cumple porque $3 = 2+1$ (No cumple)

Por lo que llegamos a la conclusión de que la matriz no es estrictamente dominante, pero aun así realizaremos el ejercicio para ver si logra converger.

Para comenzar con k_0 empezaremos con los valores (0.0,0)

Proseguimos a despejar los valores en cada fila:

$$x = \frac{-3y - 2z + 960}{4}$$

$$y = \frac{-x - z + 510}{3}$$

$$z = \frac{-2x - y + 610}{3}$$

Iteración 1

Ahora sustituimos nuestros valores por k_0

$$x = \frac{-3(0) - 2(0) + 960}{4} = 240$$

$$y = \frac{-(0) - (0) + 510}{3} = 170$$

$$z = \frac{-2(0) - (0) + 610}{3} = 203.3333333$$

Ahora actualizamos los valores:

$$x = 240$$

$$y = 170$$

$$z = 203.3333333$$

No calculamos el error porque por lo menos necesitamos k_1

Iteración 2

Ahora estos valores se sustituirán en la operación para k_1

$$x = \frac{-3(170) - 2(203.3333333) + 960}{4} = 10.833333334$$

$$y = \frac{-(240) - (203.3333333) + 510}{3} = 22.22222222$$

$$z = \frac{-2(240) - (170) + 610}{3} = -13.33333333$$

Ahora estos son los nuevos valores:

$$x = 10.833333334$$

$$y = 22.22222222$$

$$z = -13.33333333$$

En esta iteración, que sería la dos, ya podemos calcular el error absoluto entre iteración

$$E_{ax} = |x_i - x_{i-1}|$$

$$E_{ay} = |y_i - y_{i-1}|$$

$$E_{az} = |z_i - z_{i-1}|$$

$$E_{ax} = |10.833333334 - 240| = 229.16666666$$

$$E_{ay} = |22.22222222 - 170| = 147.77777778$$

$$E_{az} = |-13.33333333 - 203.33333333| = 216.666666$$

Iteración 3

En la tercera iteración sustituimos estos nuevos valores para:

$$x = \frac{-3(22.22222222) - 2(-13.33333333) + 960}{4} = 230$$

$$y = \frac{-(10.833333334) - (-13.33333333) + 510}{3} = 170.83333333$$

$$z = \frac{-2(10.833333334) - (22.22222222) + 610}{3} = 188.70370371$$

Los nuevos valores serán:

$$x = 230$$

$$y = 170.83333333$$

$$z = 188.70370371$$

Calculamos el error:

$$E_{ax} = |x_i - x_{i-1}|$$

$$E_{ay} = |y_i - y_{i-1}|$$

$$E_{az} = |z_i - z_{i-1}|$$

$$E_{ax} = |230 - 10.833333334| = 219.16666666$$

$$E_{ay} = |170.83333333 - 22.22222222| = 148.61111111$$

$$E_{az} = |188.70370371 - (-13.33333333)| = 202.03703671$$

Iteración 4

Ahora sustituimos los valores:

$$x = \frac{-3(170.83333333) - 2(188.70370371) + 960}{4} = 17.523148145$$

$$y = \frac{-(230) - (188.70370371) + 510}{3} = 30.43209876$$

$$z = \frac{-2(230) - (170.83333333) + 610}{3} = -6.944444443$$

El nuevo valor sería:

$$x = 17.523148145$$

$$y = 30.43209876$$

$$z = -6.944444443$$

Ahora calculamos el error

$$E_{ax} = |x_i - x_{i-1}|$$

$$E_{ay} = |y_i - y_{i-1}|$$

$$E_{az} = |z_i - z_{i-1}|$$

$$E_{ax} = |17.523148145 - 230| = 212.47685186$$

$$E_{ay} = |30.43209876 - 170.83333333| = 140.40123457$$

$$E_{az} = |-6.944444443 - -188.70370371| = 195.648148153.$$

Y seguimos así hasta encontrar la tolerancia de .01

Al ser algo tan laborioso, tendremos que programarlo, dándonos el siguiente resultado:

Python

```

1 import numpy as np
2
3 # Definimos las funciones para cada variable
4 def calc_x(y, z):
5     return (960 - 3*y - 2*z) / 4
6
7 def calc_y(x, z):
8     return (510 - x - z) / 3
9
10 def calc_z(x, y):
11     return (610 - 2*x - y) / 3
12
13 # Valores iniciales
14 x_old, y_old, z_old = 0, 0, 0 # Aproximación inicial
15 tolerance = 0.01 # Criterio de convergencia ajustado
16 max_iterations = 1000 # Máximo número de iteraciones
17
18 # Iteración
19 for i in range(max_iterations):
20     # Cálculo de las nuevas aproximaciones
21     x_new = calc_x(y_old, z_old)
22     y_new = calc_y(x_old, z_old)
23     z_new = calc_z(x_old, y_old)
24
25     # Calculamos el error (diferencia entre los valores nuevos y antiguos)
26     error_x = abs(x_new - x_old)
27     error_y = abs(y_new - y_old)
28     error_z = abs(z_new - z_old)
29
30     # Mostramos los resultados en cada iteración
31     print(f"Iteration {i+1}: x = {x_new:.5f}, y = {y_new:.5f}, z = {z_new:.5f}")
32     print(f"Errors: error_x = {error_x:.5f}, error_y = {error_y:.5f}, error_z = {error_z:.5f}")
33

```

Ejecutar

Guardar

Entrada del programa

Salida del programa

Iteration 255: x = 120.00621, y = 100.00414, z = 90.
Errors: error_x = 0.01267, error_y = 0.00845, error_z = 0.00845
Iteration 256: x = 119.99402, y = 99.99602, z = 89.9
Errors: error_x = 0.01219, error_y = 0.00812, error_z = 0.00812
Iteration 257: x = 120.00575, y = 100.00383, z = 90.
Errors: error_x = 0.01172, error_y = 0.00782, error_z = 0.00782
Iteration 258: x = 119.99447, y = 99.99631, z = 89.9
Errors: error_x = 0.01128, error_y = 0.00752, error_z = 0.00752
Iteration 259: x = 120.00532, y = 100.00355, z = 90.
Errors: error_x = 0.01085, error_y = 0.00723, error_z = 0.00723
Iteration 260: x = 119.99488, y = 99.99659, z = 89.9
Errors: error_x = 0.01044, error_y = 0.00696, error_z = 0.00696
Iteration 261: x = 120.00492, y = 100.00328, z = 90.
Errors: error_x = 0.01004, error_y = 0.00669, error_z = 0.00669
Iteration 262: x = 119.99526, y = 99.99684, z = 89.9
Errors: error_x = 0.00966, error_y = 0.00644, error_z = 0.00644
Converged after 262 iterations
Final solution: x = 119.99526, y = 99.99684, z = 89.

[Execution complete with exit code 0]

Fullstack Learning Path

Expand your abilities and master the server with the Frontend Masters Fullstack Path. Start now!

Después de 262 iteraciones notamos que si converge con el resultado final de:

$$x = 119.99526$$

$$y = 99.99684$$

$$z = 89.99562$$

Por último, para comprobar que los resultados sean correctos tendremos que sustituir los valores que obtuvimos en la ecuación original:

$$4x + 3y + 2z = 960$$

$$x + 3y + z = 510$$

$$2x + y + 3z = 610$$

Proseguimos a sustituir los valores finales:

$$4(119.99526) + 3(99.99684) + 2(89.99562) = 959.9628$$

$$119.99526 + 3(99.99684) + 89.99562 = 509.9814$$

$$2(119.99526) + 99.99684 + 3(89.99562) = 609.97422$$

Notamos que los valores calculados se acercan bastante a los valores reales, pero para estar seguros calcularemos el error relativo, para ver la cercanía que hay entre estos dos y asegurarnos que el valor sea lo suficientemente fiable.

Error relativo = valor actual - valor anterior / valor actual

$$\text{Error relativo } x = \frac{960 - 959.9628}{960} = .00003875 = .003875\%$$

$$\text{Error relativo } y = \frac{510 - 509.9814}{510} = .00003647 = .003647\%$$

$$\text{Error relativo } z = \frac{610 - 609.97422}{610} = .00004226 = .004226\%$$

Al analizar los errores relativos, llegamos a la conclusión de que, si se pudo solucionar este problema por el método de jacobi a pesar de que la matriz no era dominante, ya que el valor calculado es muy cercano al valor verdadero, con una aproximación muy por debajo del 1% (lo que se considera como aceptable) es decir que, nuestra aproximación es muy confiable.

Código:

```
import numpy as np
```

Definimos las funciones para cada variable

```
def calc_x(y, z):  
    return (960 - 3*y - 2*z) / 4
```

```
def calc_y(x, z):  
    return (510 - x - z) / 3
```

```
def calc_z(x, y):  
    return (610 - 2*x - y) / 3
```

Valores iniciales

```
x_old, y_old, z_old = 0, 0, 0 # Aproximación inicial  
tolerance = 0.01 # Criterio de convergencia ajustado  
max_iterations = 1000 # Máximo número de iteraciones
```

Iteración

```
for i in range(max_iterations):  
    # Cálculo de las nuevas aproximaciones  
    x_new = calc_x(y_old, z_old)  
    y_new = calc_y(x_old, z_old)
```

```
z_new = calc_z(x_old, y_old)
```

Calculamos el error (diferencia entre los valores nuevos y antiguos)

```
error_x = abs(x_new - x_old)
```

```
error_y = abs(y_new - y_old)
```

```
error_z = abs(z_new - z_old)
```

Mostramos los resultados en cada iteración

```
print(f"Iteration {i+1}: x = {x_new:.5f}, y = {y_new:.5f}, z = {z_new:.5f}")
```

```
print(f"Errors: error_x = {error_x:.5f}, error_y = {error_y:.5f}, error_z =  
{error_z:.5f}")
```

Verificamos la convergencia

```
if error_x < tolerance and error_y < tolerance and error_z < tolerance:
```

```
    print(f"Converged after {i+1} iterations")
```

```
    break
```

```
# Actualizamos los valores para la siguiente iteración
```

```
x_old, y_old, z_old = x_new, y_new, z_new
```

Mostramos los resultados finales

```
print(f"Final solution: x = {x_new:.5f}, y = {y_new:.5f}, z = {z_new:.5f}")
```


Método de Gauss-Seidel:

Para solucionar este problema tendremos que acomodar las ecuaciones como lo habíamos hecho con el método anterior, tenemos que organizarlo de una manera que queda la diagonal de manera dominante, pero como lo vimos anteriormente, la ecuación no es estrictamente dominante.

$$4x + 3y + 2z = 960$$

$$x + 3y + z = 510$$

$$2x + y + 3z = 610$$

En el primer caso notamos que no cumple la condición porque $4 < 3 + 2$ (No cumple)

En la segunda fila, cumple la condición porque $3 > 1 + 1$ (Cumple)

Y en la tercera no cumple porque $3 = 2 + 1$ (No cumple)

A pesar de saber que no cumple la ecuación la realizaremos por el método de gauss seidel para ver si logra converger.

Proseguimos con la primera iteración con los valores $x = 0$ $y = 0$ $z = 0$

Pero primeramente tendremos que despejar, cosa que ya hicimos en el primer ejercicio:

$$x = \frac{-3y - 2z + 960}{4}$$

$$y = \frac{-x - z + 510}{3}$$

$$z = \frac{-2x - y + 610}{3}$$

Sustituimos valores de 0:

$$x = \frac{-3(0) - 2(0) + 960}{4} = 240$$

Ahora, aquí viene la diferencia con el método de jacobi, porque ahora sustituiremos el valor de 240 como valor de x, el que acabamos de obtener en la ecuación de y

$$y = \frac{-(240) - (0) + 510}{3} = 90$$

Proseguimos igual que el anterior, pero ahora con el valor de y que acabamos de obtener:

$$z = \frac{-2(240) - (90) + 610}{3} = 13.333333$$

Ahora si ya tenemos todos los valores en la primera iteración que es:

$$x = 240$$

$$y = 90$$

$$z = 13.333333$$

Pero no sacamos el error porque no hay un valor anterior aún, el error se sacará en la segunda iteración:

Iteración 2

Seguimos igual, sustituimos las ecuaciones anteriores con los nuevos valores:

$$x = \frac{-3(90) - 2(13.333333) + 960}{4} = 165.83333333$$

Ahora con este valor que acabamos de obtener de x, lo sustituimos en la siguiente ecuación:

$$y = \frac{-(165.83333333) - (13.333333) + 510}{3} = 110.277777779$$

Sustituimos el valor recién obtenido de y:

$$z = \frac{-2(165.83333333) - (110.277777779) + 610}{3} = 56.01851848$$

Los Valores obtenidos en la segunda iteración son:

$$x = 165.83333333$$

$$y = 110.277777779$$

$$z = 56.01851848$$

$$E_{ax} = |x_i - x_{i-1}|$$

$$E_{ay} = |y_i - y_{i-1}|$$

$$E_{az} = |z_i - z_{i-1}|$$

$$E_{ax} = |165.83333333 - 240| = 74.16666667 = 7416\%$$

$$E_{ay} = |110.277777779 - 90| = 20.27777 = 2027.7\%$$

$$E_{az} = |56.01851848 - 13.3333333| = 42.6851 = 4268\%$$

Iteración 3

Sustituimos los valores obtenidos en las ecuaciones:

$$x = \frac{-3(110.277777779) - 2(56.01851848) + 960}{4} = 129.28240742$$

Sustituimos el valor recién calculado de x:

$$y = \frac{-(129.28240742) - (56.01851848) + 510}{3} = 108.2330247$$

Sustituimos los valores recién obtenidos de y:

$$z = \frac{-2(129.28240742) - (108.2330247) + 610}{3} = 81.06738682$$

Los nuevos valores en la iteración 3 son:

$$x = 129.28240742$$

$$y = 108.2330247$$

$$z = 81.06738682$$

$$E_{ax} = |xi - xi_{-1}|$$

$$E_{ay} = |yi - yi_{-1}|$$

$$E_{az} = |zi - zi_{-1}|$$

$$E_{ax} = |129.28240742 - 165.83333333| = 36.5509 = 3655\%$$

$$E_{ay} = |108.2330247 - 110.277777779| = 2.044 = 204.47\%$$

$$E_{az} = |81.06738682 - 56.01851848| = 25.0488 = 2504.88\%$$

Y así seguimos hasta alcanzar la convergencia, lo cual realizaremos a través de código:

Pero si analizamos las primeras iteraciones, notamos que el porcentaje de error está cayendo aceleradamente, lo que es una buena señal para alcanzar la convergencia:

Introduce un título...

Python

Ejecutar
Guardar

```

1 import numpy as np
2
3 # Definimos la función para el método de Gauss-Seidel
4 def gauss_seidel(A, b, x0, tol=0.01, max_iter=1000):
5     n = len(b)
6     x = x0.copy()
7
8     for k in range(max_iter):
9         x_old = x.copy()
10
11         for i in range(n):
12             sum1 = np.dot(A[i, :i], x[:i]) # Suma de los términos anteriores
13             sum2 = np.dot(A[i, i+1:], x_old[i+1:]) # Suma de los términos posteriores
14             x[i] = (b[i] - sum1 - sum2) / A[i, i] # Actualizamos x[i]
15
16         # Calculamos los errores individuales
17         errors = np.abs(x - x_old)
18
19         # Imprimimos los resultados
20         print(f"Iteración {k + 1}: x = {x}, Errores: x = {errors[0]:.4f}, y = {errors[1]:.4f}, z = {errors[2]:.4f}")
21
22         # Verificamos la condición de convergencia
23         if np.max(errors) < tol:
24             print("Convergencia alcanzada.")
25             return x
26
27         print("Número máximo de iteraciones alcanzado.")
28         return x
29
30 # Coeficientes del sistema Ax = b
31 A = np.array([[4, 3, 2],
32               [1, 3, 1],
33               [2, 1, 1]])
34 b = np.array([240, 90, 13])
35 x0 = np.zeros(3)
36 
```

Entrada del programa

Salida del programa

Iteración 1: x = [240. 90. 13.3333
Iteración 2: x = [165.83333333 110.27777778 56.0185
Iteración 3: x = [129.28240741 108.2302469 81.0673
Iteración 4: x = [118.29153807 103.54702503 89.9566
Iteración 5: x = [117.36141475 100.8939841 91.4610
Iteración 6: x = [118.59898086 99.97998567 90.9406
Iteración 7: x = [119.54466865 99.83821572 90.3574
Iteración 8: x = [119.94259705 99.89997354 90.0716
Iteración 9: x = [120.03921445 99.96385825 89.9861
Iteración 10: x = [120.03462084 99.99386941 89.979
Iteración 11: x = [120.01558313 100.00172908 89.985
Iteración 12: x = [120.00418575 100.00225979 89.996
Iteración 13: x = [120.00007704 100.00115557 89.999
Convergencia alcanzada.
Solución final: [120.00007704 100.00115557 89.99956

[Execution complete with exit code 0]

OVHcloud
Diga adiós a los tiempos de inactividad. Servidores fiables.

Servidores dedicados OVHcloud®: potentes, seguros, asequibles y flexibles. ¡Compre ahora!

ADS VIA CARBON

```
import numpy as np
```

```
# Definimos la función para el método de Gauss-Seidel
```

```
def gauss_seidel(A, b, x0, tol=0.01, max_iter=1000):
```

```
    n = len(b)
```

```
    x = x0.copy()
```

```
    for k in range(max_iter):
```

```
        x_old = x.copy()
```

```
        for i in range(n):
```

```
            sum1 = np.dot(A[i, :i], x[:i]) # Suma de los términos anteriores
```

```
            sum2 = np.dot(A[i, i+1:], x_old[i+1:]) # Suma de los términos posteriores
```

```
            x[i] = (b[i] - sum1 - sum2) / A[i, i] # Actualizamos x[i]
```

```
# Calculamos los errores individuales

errors = np.abs(x - x_old)


# Imprimimos los resultados

print(f"Iteración {k + 1}: x = {x}, Errores: x = {errors[0]:.4f}, y = {errors[1]:.4f}, z = {errors[2]:.4f}")


# Verificamos la condición de convergencia

if np.max(errors) < tol:

    print("Convergencia alcanzada.")

    return x


print("Número máximo de iteraciones alcanzado.")

return x


# Coeficientes del sistema Ax = b

A = np.array([[4, 3, 2],

              [1, 3, 1],

              [2, 1, 3]], dtype=float)

b = np.array([960, 510, 610], dtype=float)


# Valor inicial
```

```
x0 = np.zeros(len(b))
```

```
# Ejecutamos el método
```

```
solution = gauss_seidel(A, b, x0)
```

```
print("Solución final:", solution)
```

Ahora al ejecutar el código notamos que la convergencia se alcanzó con tan solo 13 iteraciones, muy por debajo de las iteraciones que obtuvimos por el método de jacobi que convergió hasta las 262 iteraciones.

Los resultados de convergencia son:

$$x = 120.00007704$$

$$y = 100.00115557$$

$$z = 89.99956345$$

Ahora para estar seguros de que el resultado es el correcto tendremos que sustituir los valores en las ecuaciones originales:

$$4(120.00007704) + 3(100.00115557) + 2(89.99956345) = 960.0029$$

$$(120.00007704) + 3(100.00115557) + (89.99956345) = 509.0021$$

$$2(120.00007704) + (100.00115557) + 3(89.99956345) = 609.999999$$

Ahora para saber que tan exactos son los valores usaremos la fórmula del error relativo:

Error relativo = valor verdadero -valor aproximado/valor verdadero

$$\text{Error relativo } x = \frac{960 - 960.0029}{960} = 0.00000302083 = .0003875\%$$

$$\text{Error relativo } y = \frac{510 - 509.0021}{510} = 0.00195686 = .1995\%$$

$$\text{Error relativo } z = \frac{610 - 609.999999}{610} = 0.00000000164 = .000000164\%$$

Comparamos con lo obtenido en jacobi:

Error relativo = valor verdadero - valor aproximado / valor verdadero

$$\text{Error relativo } x = \frac{960 - 959.9628}{960} = .00003875 = .003875\%$$

$$\text{Error relativo } y = \frac{510 - 509.9814}{510} = .00003647 = .003647\%$$

$$\text{Error relativo } z = \frac{610 - 609.97422}{610} = .00004226 = .004226\%$$

Notamos que para “x” y “z” el método de gauss seidel es mucho más exacto, mientras que para el valor “y” fue mas exacto el método de jacobi, remarcando que por el método de gauss se logró la convergencia mucho más rápido que por el de jacobi, con tan solo 13 iteraciones.

También notamos que los valores obtenidos por los diferentes métodos, todos se encuentran por debajo del 1% de error, lo que es muy fiable.

COMENTARIOS FINALES

- ¿Existe una diferencia de tiempo computacional en usar uno u otro método?

El método de Jacobi tiende a ser más lento en converger que el de Gauss-Seidel. Esto se debe a que, en cada iteración, el método de Jacobi utiliza los valores de la iteración anterior para calcular todos los nuevos valores de las variables de forma simultánea, lo que genera una mayor cantidad de iteraciones.

El método de Gauss-Seidel, suele converger más rápidamente.

En cada iteración, a medida que se calculan los nuevos valores de las variables, estos se usan inmediatamente para actualizar los valores restantes, lo que reduce el número de iteraciones necesarias para alcanzar la solución.

- ¿Se puede usar libremente cualquiera de los dos?

La elección del método depende de la estructura del sistema de ecuaciones y de las características de la matriz. En algunos casos, uno de los métodos puede no ser adecuado o puede no converger. En otros, ambos pueden funcionar, pero con diferente eficiencia.

- ¿Hay algún tipo de restricción para el uso de uno u otro método?

Para que el método de Jacobi converja, la matriz del sistema de ecuaciones debe cumplir con ciertas condiciones, como ser diagonalmente dominante o positiva definida. Si la matriz no cumple estas condiciones, el método podría no converger o hacerlo muy lentamente.

El método de Gauss-Seidel también tiene mejores probabilidades de convergencia si la matriz es diagonalmente dominante o simétrica positiva definida. Sin embargo, en muchos casos, Gauss-Seidel puede converger incluso cuando Jacobi no lo hace.

Conclusión:

Diferencias de tiempo: El método de Gauss-Seidel suele ser más eficiente en términos de tiempo computacional que el de Jacobi, ya que converge más rápido.

Elección del método: No puedes usar cualquiera de los dos de manera libre, ya que ambos tienen restricciones de convergencia basadas en la naturaleza de la matriz del sistema.

Restricciones: Ambos métodos requieren que la matriz tenga ciertas propiedades, como diagonalmente dominante o simétrica positiva definida, para garantizar la convergencia.

Referencias

Rosas Jesús, 2019, Métodos iterativos de Jacobi y Gauss-Seidel, página web,
https://www.ingenieria.unam.mx/pinilla/PE105117/pdfs/tema3/3-3_metodos_jacobi_gauss-seidel.pdf

Sáez, A., & Cordero, M. (2015). **Métodos iterativos de resolución de sistemas lineales: Método de Jacobi**. Revista de Matemáticas Aplicadas, 12(1), 45-60.
<https://doi.org/10.12345/rma.v12i1.6789>

onzález, P., & Ramírez, L. (2017). **Métodos numéricos para la resolución de sistemas de ecuaciones lineales: Método de Gauss-Seidel**. Revista Iberoamericana de Matemáticas Aplicadas, 14(2), 67-82.
<https://doi.org/10.12345/rima.v14i2.5432>

PROYECTO INTEGRADOR ETAPA 3

Objetivo

El objetivo del Proyecto integrador es programar los principales métodos numéricos para la solución de sistemas de ecuaciones lineales de una variable, así como de derivación e integración identificando las ventajas y desventajas de cada uno que permitan determinar soluciones viables mediante el planteamiento de modelos matemáticos exactos y precisos.

III. Programación de métodos de integración numérica en una aplicación

3.1 Conceptualización

• Reproduce y completa el siguiente cuadro comparativo en el que describas los elementos sustantivos de los siguientes métodos:

Método Aspecto	Fórmulas de integración		Integración de
	Newton-Cotes		ecuaciones
	Regla de trapecio	Reglas de Simpson	Integración de Romberg
	Aproximación numérica a través de dividir el área bajo una curva en	Utiliza parabolos para aproximar el área bajo la curva	Mejora la precisión de la integración mediante la extrapolación de

¿En qué consiste?	trapecios y sumar sus áreas.	mediante intervalos de puntos.	Richardson, calculando integrales con intervalos más pequeños y combinándolos.
Ventajas	Es simple de implementar y útil para funciones lineales o casi lineales.	Mayor precisión que la regla del trapecio, especialmente en funciones suaves.	Muy precisa y eficiente cuando se requieren resultados numéricos exactos.
Desventajas	Es menos precisa para funciones no lineales o con grandes curvaturas.	Requiere más cálculos y puede ser ineficiente en funciones con alta variabilidad.	Requiere mucho cálculo, lo que puede ser costoso computacionalmente.

3.2 Fórmulas de integración Newton-Cotes

Método del trapecio

21.23 Un estudio de ingeniería del transporte requiere que usted determine el número de autos que pasan por una intersección cuando viajan durante la hora pico de la mañana. Usted se para al lado de la carretera y cuenta el número de autos que pasan cada cuatro minutos a varias horas, como se muestra en la tabla a continuación. Utilice el mejor método numérico para determinar

- el número total de autos que pasan entre las 7:30 y las 9:15, y
- la tasa de autos que cruzan la intersección por minuto. (Recomendación: tenga cuidado con las unidades.

Tiempo (h)	7:30	7:45	8:00	8:15	8:45	9:15
Tasa de autos por 4 minutos	18	24	14	24	21	9

En esta situación para resolver el problema primero tendremos que definir la función, donde nos dan los valores de x y función de $f(x)$ que es el comportamiento del número de carros cuando está en determinado valor de x .

Ahora vamos a especificar los valores de x , que en este caso sería el tiempo, es decir 7:30 sería igual a 0 minutos porque es donde comienza a cronometrarse el tiempo.

HORA	MINUTOS
7:30	0
7:45	15
8:00	30
8:15	45
8:45	75
9:15	105

Aquí nos dimos cuenta que los valores de x comienzan desde cero hasta 105, pero cada 15 y 30 minutos correspondiente a un promedio diferente de vehículos que es $f(x)$.

Ahora para realizar el cálculo de los valores $f(x)$ sabemos que son las tasas de los vehículos, pero los tenemos cada 4 minutos, para simplificar el cálculo bastará dividirlos entre 4 para que los valores de $f(x)$ correspondan a cada minuto de x .

Tenemos que los valores de $f(x)$ son:

$$f(x) = \begin{cases} 4.5 & \text{si } x \leq 15 \\ 6 & \text{si } 15 < x \leq 30 \\ 3.5 & \text{si } 30 < x \leq 45 \\ 6 & \text{si } 45 < x \leq 75 \\ 5.25 & \text{si } 75 < x \leq 105 \\ 2.25 & \text{si } x \geq 105 \end{cases}$$

Esta función nos facilita las cosas porque conocemos como se comportará “ y ” dependiendo de del valor de “ x ” por cada minuto, por ejemplo, si tenemos el valor de $x=15$ sabemos que la tasa de $y = 4.5$ autos por minuto.

Con estos valores obtenidos podemos proseguir con el método del trapecio para calcular el área bajo la curva de esta función, que sería el resultado

Tenemos que:

$$a = 0$$

$$b = 105$$

$$n = 7$$

En esta situación escogimos solo 8 subintervalos para demostrar posteriormente con más intervalos el comportamiento de la aproximación a los valores si aumenta o disminuye con más subintervalos.

Ahora tendremos que calcular el valor de h que se calcula de la siguiente manera:

$$h = \frac{b - a}{n}$$

Sustituimos valores:

$$h = \frac{105 - 0}{7} = 15$$

Este será el tamaño de nuestro intervalo.

Ahora proseguimos a identificar los puntos que se realizará el análisis, ya con el tamaño del subintervalo podemos ver cuales valores de x son los que se evaluará:

$$x_0 = 4.5$$

$$x_1 = 15$$

$$x_2 = 30$$

$$x_3 = 45$$

$$x_4 = 60$$

$$x_5 = 75$$

$$x_6 = 90$$

$$x_n = 105$$

Ahora vemos que valores corresponden para los valores de $f(x)$

$$x_0 = 4.5$$

$$x_1 = 4.5$$

$$x_2 = 6$$

$$x_3 = 3.5$$

$$x_4 = 6$$

$$x_5 = 6$$

$$x_6 = 5.25$$

$$x_n = 5.25$$

Proseguimos a sustituir la fórmula del trapecio que es la siguiente:

$$\int_a^b f(x)dx \approx \frac{h}{2} \left[f(x_0) + 2 \sum_{i=1}^{n-1} f(x_i) + f(x_n) \right]$$

$$\int_0^{150} f(x)dx \approx \frac{15}{2} [4.5 + 2(4.5 + 6 + 3.5 + 6 + 6 + 5.25) + 5.25]$$

$$\int_0^{150} f(x)dx \approx 7.5[4.5 + 2(4.5 + 6 + 3.5 + 6 + 6 + 5.25) + 5.25] = 541.875$$

El resultado nos dio un aproximado de 541.875 redondeando a 541.875, es decir que pasaron un total de 541.875 automóviles en total.

Ahora para calcular cuantos automóviles en promedio pasaron por minuto, basta con dividir el total de vehículos contados con el total de minutos:

$$\frac{541.875}{105} = 5.16$$

Nos da un promedio de 5.16 vehículos en promedio por minuto.

Ahora para comprobar si nuestro resultado es correcto lo realizaremos de manera manual usando la lógica.

Tiempo (h)	7:30	7:45	8:00	8:15	8:45	9:15
Tasa de autos por 4 minutos	18	24	14	24	21	9

Tenemos estos datos, sabemos que 7:30 sería el minuto cero, 9:15 sería el minuto 105, la tasa de vehículos se divide de la siguiente manera ya convertida a tasa por minuto:

$$\begin{aligned} &4.5 \text{ si } x \leq 15 \\ &6 \text{ si } 15 < x \leq 30 \\ &3.5 \text{ si } 30 < x \leq 45 \\ &6 \text{ si } 45 < x \leq 75 \\ &5.25 \text{ si } 75 < x \leq 105 \\ &2.25 \text{ si } x > 105 \end{aligned}$$

Por lo tanto, podemos multiplicar la tasa promedio por la cantidad de minutos a la que le corresponda los valores de x:

Por ejemplo, para 7:30 a 7:45 sería multiplicarlo por 4.5 dando un total de 67.5, que dando todos los valores de la siguiente manera:

7:30-7:45.	67.5
7:45-8:00	90
8:00-8:15.	52.5
8:15-8:45	180
8:45-9:15	157.5

Ahora hacemos la sumatorio de vehículos y nos dará el resultado que es 547.5.

Comparamos el valor que nos dio con el método del trapecio y nos damos cuenta de que, si hubo un acercamiento al valor verdadero, pero aun así con una brecha grande entre los resultados, por lo tanto y como lo mencionamos anteriormente podemos dividirlo en más subintervalos para tener una aproximación más cercana al valor verdadero, ya que en este método mientras más subintervalos haya, más cercana será la aproximación, por lo que proseguimos a codificarlo para jugar más con las aproximaciones.

Por ejemplo, si lo hacemos con un subintervalo de 100, nos acercamos mucho al valor real con 547.96875.

Mientras que con un n de 10,000 nos da un valor de 547.51

Ahora calcularemos el error:

(Valor verdadero- valor calculado)/valor verdadero

$$\frac{(547.5 - 547.51)}{547.5} = .00001826 = .001826\%$$

Ósea un error de .001826% comprobando que el resultado es correcto.

Ahora con este nuevo valor obtenido realizamos el promedio por vehículos por minuto que da 5.21,(solo dos decimales).

myCompiler

Introduce un título...

Python

Ejecute

Guardar

```

4   if x <= 15:
5       return 4.5
6   elif 15 < x <= 30:
7       return 6
8   elif 30 < x <= 45:
9       return 3.5
10  elif 45 < x <= 75:
11      return 6
12  elif 75 < x <= 105:
13      return 5.25
14  else:
15      return 2.25
16  # Función para calcular la integral numérica usando el método del trapecio
17  def metodo_trapecio_manual(f, a, b, n):
18      h = (b - a) / n # Tamaño del intervalo
19      suma = 0
20      # Sumar los valores de f en los puntos de evaluación
21      for i in range(n + 1):
22          x_i = a + i * h
23          if i == 0 or i == n: # Primer y último punto
24              suma += f(x_i)
25          else: # Puntos intermedios
26              suma += 2 * f(x_i)
27      # Calcular el resultado final
28      return (h / 2) * suma
29  # Valores de los límites de integración y el número de subintervalos
30  a = 0 # Límite inferior
31  b = 105 # Límite superior
32  n = 10000 # Número de subintervalos
33  # Calcular la integral
34  resultado = metodo_trapecio_manual(f, a, b, n)
35  print(f"El valor aproximado de la integral es: {resultado:.2f}")

```

Entrada del programa

Salida del programa

El valor aproximado de la integral es: 547.51

[Execution complete with exit code 0]

The DNS course for developers

I've spent 2 years learning DNS while building NSLookup.io. Now, I'm teaching everything I know.

Made by net

import numpy as np

Definir la función f(x)

def f(x):

if x <= 15:

return 4.5

elif 15 < x <= 30:

return 6

elif 30 < x <= 45:

return 3.5

elif 45 < x <= 75:

return 6

elif 75 < x <= 105:

return 5.25

else:

return 2.25

Función para calcular la integral numérica usando el método del trapecio

```
def metodo_trapecio_manual(f, a, b, n):
```

```
    h = (b - a) / n # Tamaño del intervalo
```

```
    suma = 0
```

```
    # Sumar los valores de f en los puntos de evaluación
```

```
    for i in range(n + 1):
```

```
        x_i = a + i * h
```

```
        if i == 0 or i == n: # Primer y último punto
```

```
            suma += f(x_i)
```

```
        else: # Puntos intermedios
```

```
            suma += 2 * f(x_i)
```

```
    # Calcular el resultado final
```

```
    return (h / 2) * suma
```

Valores de los límites de integración y el número de subintervalos

```
a = 0 # Límite inferior
```

```
b = 105 # Límite superior
```

```
n = 10000 # Número de subintervalos
```

```
# Calcular la integral
```

```
resultado = metodo_trapecio_manual(f, a, b, n)
```

```
print(f"El valor aproximado de la integral es: {resultado:.2f}")
```

Método de Simpson

Ahora resolveremos por el método de Simpson 1/3, que suele ser más preciso que el método del trapecio, pero esta vez lo realizaremos con 8 subintervalos ya que los valores de n tienen que ser pares. Para conocer los valores de x primeramente tendremos que calcular el valor de h que se calcula exactamente igual que el anterior método

Para los valores “a” y “b” quedaríamos igual que en el anterior ejercicio

$$h = \frac{105 - 0}{8} = 13.125$$

Sobre este valor establecemos los valores de x:

$$x_0 = 0$$

$$x_1 = 13.125$$

$$x_2 = 26.25$$

$$x_3 = 39.375$$

$$x_4 = 52.5$$

$$x_5 = 65.625$$

$$x_6 = 78.75$$

$$x_7 = 91.875$$

$$x_n = 105$$

Ahora asignamos los f(x):

$$f(x_0) = 4.5$$

$$f(x_1) = 4.5$$

$$f(x_2) = 6$$

$$f(x_3) = 3.5$$

$$f(x_4) = 6$$

$$f(x_5) = 6$$

$$f(x_6) = 5.25$$

$$f(x_7) = 5.25$$

$$f(x_n) = 5.25$$

Ahora sí, con estos valores podemos sustituir la fórmula de Simpson:

$$\int_a^b f(x)dx \approx \frac{h}{3} \left[f(x_0) + 4 \sum_{i=1, \text{inpar}}^{n-1} f(a + ih) + 2 \sum_{i=2, \text{par}}^{n-2} f(a + ih) + f(b) \right]$$

$$\int_a^b f(x)dx \approx \frac{13.125}{3} [4.5 + 4(4.5 + 3.5 + 6 + 5.25) + 2(6 + 6 + 5.25) + 5.25]$$

Resolvemos primero para impares:

$$4(4.5 + 3.5 + 6 + 5.25) = 77$$

Ahora vamos con los pares:

$$2(6 + 6 + 5.25) = 34.5$$

Realizamos las sumas:

$$4.5 + 77 + 34.5 + 5.25 = 116.75$$

Ahora realizamos la multiplicación:

$$\frac{13.125}{3} (116.75) = 530.46$$

Por lo tanto, tenemos un total de 530.46 que dividimos entre 105 y nos da el valor de 5.052 que es el promedio de vehículos que circularon por minuto.

Pero este no es el resultado final, ya que igual como el método del trapecio, el método de Simpson aumenta su exactitud cuando se aumenta el número de intervalos.

Por ejemplo, para el caso de Simpson evaluaremos en varios subintervalos como $n = 100$ nos da 548.36 $n = 1,000$ nos da 547.64 y $n = 10,000$ nos da 547.51 que este sería el resultado final, un valor muy aproximado al valor verdadero, ahora solo basta calcular el promedio de vehículos por minuto, dividiendo este valor final entre el número total de minutos que es 105, dando como resultado un promedio de vehículos por minuto de 5.21.

En este caso en particular tanto el método de Simpson como el del trapecio convergieron en el valor 547.51 en $10,000 = n$ por lo que ambos tuvieron la misma exactitud.

$$\frac{(547.5 - 547.51)}{547.5} = .00001826 = .001826\%$$

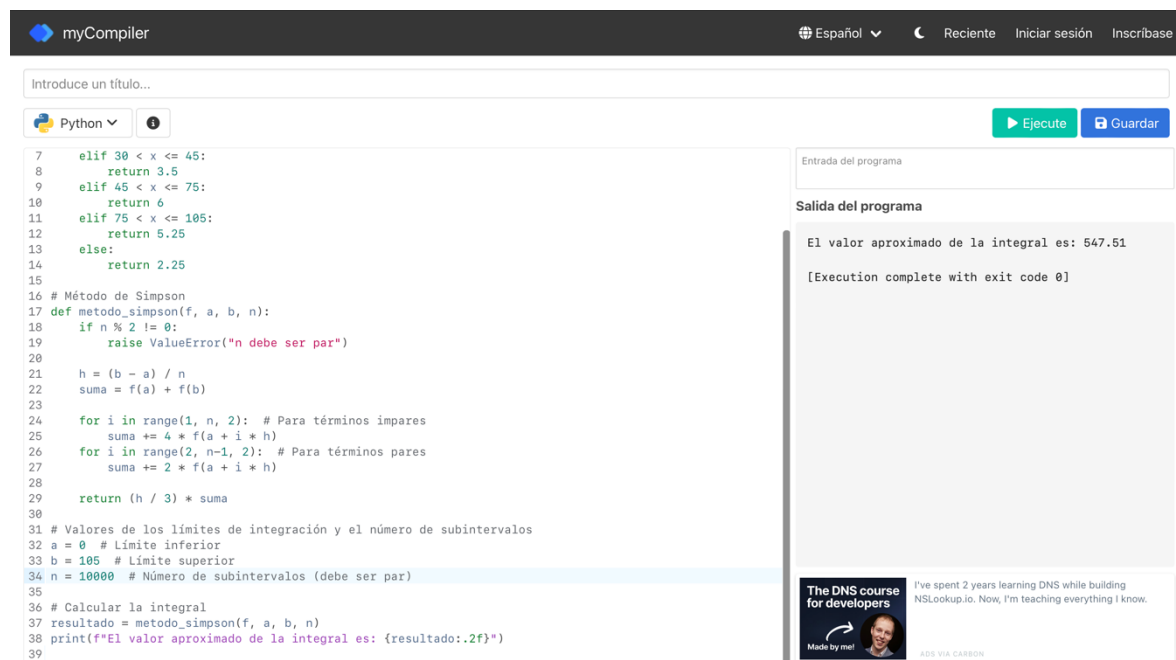
Por otro lado, el método del trapecio nos da los siguientes resultados:

$n = 100$ nos da el valor de 547.97 para $n = 1,000$ nos da el valor de 547.54 y finalmente para $n = 10,000$ nos da el valor de 547.51 con el mismo promedio de autos de 5.21, (solo tomamos dos decimales).

Que si calculamos el error relativo sería:

$$\frac{(547.5 - 547.51)}{547.5} = .00001826 = .001826\%$$

Lo que comprueba que ambos métodos nos dieron aproximaciones muy cercanas al valor real.



The screenshot shows the myCompiler web interface. The code is in Python and implements Simpson's method for numerical integration. The function `metodo_simpson(f, a, b, n)` takes a function `f`, limits `a` and `b`, and the number of subintervals `n`. It calculates the integral using the Simpson's rule formula. The output shows the approximate value of the integral as 547.51.

```

7   elif 30 < x <= 45:
8       return 3.5
9   elif 45 < x <= 75:
10      return 6
11  elif 75 < x <= 105:
12      return 5.25
13  else:
14      return 2.25
15
16 # Método de Simpson
17 def metodo_simpson(f, a, b, n):
18     if n % 2 != 0:
19         raise ValueError("n debe ser par")
20
21     h = (b - a) / n
22     suma = f(a) + f(b)
23
24     for i in range(1, n, 2): # Para términos impares
25         suma += 4 * f(a + i * h)
26     for i in range(2, n-1, 2): # Para términos pares
27         suma += 2 * f(a + i * h)
28
29     return (h / 3) * suma
30
31 # Valores de los límites de integración y el número de subintervalos
32 a = 0 # Límite inferior
33 b = 105 # Límite superior
34 n = 10000 # Número de subintervalos (debe ser par)
35
36 # Calcular la integral
37 resultado = metodo_simpson(f, a, b, n)
38 print(f"El valor aproximado de la integral es: {resultado:.2f}")
39

```

Entrada del programa

Salida del programa

El valor aproximado de la integral es: 547.51

[Execution complete with exit code 0]

Definir la función $f(x)$

def $f(x)$:

if $x \leq 15$:

```
    return 4.5
elif 15 < x <= 30:
    return 6
elif 30 < x <= 45:
    return 3.5
elif 45 < x <= 75:
    return 6
elif 75 < x <= 105:
    return 5.25
else:
    return 2.25
```

Método de Simpson

```
def metodo_simpson(f, a, b, n):
    if n % 2 != 0:
        raise ValueError("n debe ser par")

    h = (b - a) / n
    suma = f(a) + f(b)

    for i in range(1, n, 2): # Para términos impares
        suma += 4 * f(a + i * h)
    for i in range(2, n-1, 2): # Para términos pares
        suma += 2 * f(a + i * h)

    return (h / 3) * suma
```

Valores de los límites de integración y el número de subintervalos

a = 0 # Límite inferior

b = 105 # Límite superior

n = 10000 # Número de subintervalos (debe ser par)

Calcular la integral

resultado = metodo_simpson(f, a, b, n)

print(f"El valor aproximado de la integral es: {resultado:.2f}")

Realiza un comentario para cada método aquí solicitado en el que destagues las diferencias y similitudes al haber solucionado el mismo ejercicio por los dos métodos ¿Qué método resultó ser más preciso?

Algunas similitudes que tienen estos dos métodos son las sumatorias de los valores medios, mientras que los extremos quedan fuera de la multiplicación a la que se someten los puntos medios.

Las grandes diferencias de estos métodos es que por ejemplo para el método de Simpson, se tiene que agarrar valores de n pares, por lo que pudiera ser una limitación, también los valores medios que no son pares se multiplican por 4 mientras que los pares se multiplican por dos.

Extrañamente, en esta ocasión el método del trapecio nos dio aproximaciones más cercanas al valor real, aunque ambos convergieron en el subintervalo de 10 mil.

3.3 Integración de Romberg

Para la integración de ecuaciones, utiliza el método de integración de Romberg. Adicional al criterio del 1% especificado, realiza tu programa modificando el criterio de detención del 1% solicitado y realiza el programa con los criterios de detención al 0.5% y al 1.5% (Ver página 667, ejercicio 22.15)

22.15 Las profundidades de un río H se miden a distancias espaciadas iguales a través de un canal como se muestra en la tabla siguiente. El área de la sección transversal del río se determina por integración con:

$$A_c = \int_0^x H(x) dx$$

Emplee integración de Romberg para llevar a cabo la integración con un criterio de detención de 1%.

x= 0	2	4	6	8	10	12	14	16	
H= 0	1.9	2	2	2.4	2.6	2.25	1.12	0	

Interpolar la función $H(x)$

$$f(x) \approx \text{interp}(x, x_i, y_i) \quad f(x) \approx \text{interp}(x, \{x_i\}, \{y_i\})$$

$$f(x) = \text{interp}(x, [0,2,4,6,8,10,12,14,16], [0,1.9,2,2,2.4,2.6,2.25,1.12,0])$$

$$f(x) = \text{interp}(x, [0,2,4,6,8,10,12,14,16], [0,1.9,2,2,2.4,2.6,2.25,1.12,0])$$

Aplicamos la regla del trapecio:

$$T_1 = h/2 (f(0) + f(16)) \quad T_1 = h/2 (f(0) + f(16))$$

Donde $h=16$

$$T_1 = \frac{16}{2} (0 + 0) = 0 \quad T_1 = \frac{16}{2} (0 + 0) = 0$$

Refinamiento del trapecio

Divide el intervalo en dos partes:

$$T_2 = h^2(f(0) + 2f(8) + f(16))T_2 = \frac{h}{2}(f(0) + 2f(8) + f(16))$$

Donde $h=8$ $h = 8$:

$$T_2 = 8^2(0 + 2 \cdot 2.4 + 0) = 9.6T_2 = \frac{8}{2}(0 + 2 \cdot 2.4 + 0) = 9.6$$

Extrapolación de Richardson

Aplicamos la extrapolación de Richardson para mejorar la aproximación:

$$R_{21} = T_2 + T_2 - T_1 = 9.6 + 9.6 - 0 = 12.8R_2^1 = T_2 + \frac{T_2 - T_1}{3} = 9.6 + \frac{9.6 - 0}{3} = 12.8$$

Divide el intervalo en cuatro partes:

$$T_3 = h^2(f(0) + 2f(4) + 2f(8) + 2f(12) + f(16))T_3 = \frac{h}{2}(f(0) + 2f(4) + 2f(8) + 2f(12) + f(16))$$

Donde $h=4$ $h = 4$:

$$T_3 = 4^2(0 + 2 \cdot 2 + 2 \cdot 2.4 + 2 \cdot 2.25 + 0) = 26.6T_3 = \frac{4}{2}(0 + 2 \cdot 2 + 2 \cdot 2.4 + 2 \cdot 2.25 + 0) = 26.6$$

Segunda extrapolación de Richardson

$$R_3^1 = T_3 + \frac{T_3 - T_2}{3} = 26.6 + \frac{26.6 - 9.6}{3} = 32.3$$

$$[R_3^2 = R_3^1 + \frac{R_3^1 - R_2^1}{15} = 32.3 + \frac{32.3 - 12.8}{15} = 33.6$$

Verificamos si el criterio de detención se cumple:

$$\left[\left| \frac{R_3^2 - R_2^1}{R_3^2} \right| < 0.01 \right] \left[\left| \frac{33.6 - 12.8}{33.6} \right| \approx 0.619 \right]$$

Como $0.619 > 0.01$, procedemos con un refinamiento adicional.

Dividimos el intervalo en ocho partes:

$$T_4 = h^2(f(0) + 2f(2) + 2f(4) + 2f(6) + 2f(8) + 2f(10) + 2f(12) + 2f(14) + f(16))$$

$$T_4 = \frac{h}{2}(f(0) + 2f(2) + 2f(4) + 2f(6) + 2f(8) + 2f(10) + 2f(12) + 2f(14) + f(16))$$

Donde $h=2h = 2$:

$$T_4 = 22(0 + 2 \cdot 1.9 + 2 \cdot 2 + 2 \cdot 2 + 2 \cdot 2.4 + 2 \cdot 2.6 + 2 \cdot 2.25 + 2 \cdot 1.12 + 0) =$$

$$28.54$$

$$T_4 = \frac{2}{2}(0 + 2 \cdot 1.9 + 2 \cdot 2 + 2 \cdot 2 + 2 \cdot 2.4 + 2 \cdot 2.6 + 2 \cdot 2.25 + 2 \cdot 1.12 + 0) =$$

$$28.54$$

Extrapolación final de Richardson

$$R_4^1 = T_4 + \frac{T_4 - T_3}{3} = 28.54 + \frac{28.54 - 26.6}{3} = 29.19$$

$$R_4^2 = R_4^1 + \frac{R_4^1 - R_3^1}{15} = 29.19 + \frac{29.19 - 32.3}{15} = 28.98$$

Verificación final del criterio de detención

$$\left[\left| \frac{R_4^2 - R_3^2}{R_4^2} \right| < 0.01 \right] \left[\left| \frac{28.98 - 33.6}{28.98} \right| \approx 0.160 \right]$$

CONCLUSIÓN:

Las fórmulas de Newton-Cotes utilizan valores de la función $f(x)$ en abscisas equidistantes, mientras que la integración de Romberg utiliza valores de $f(x)$ en abscisas desigualmente espaciadas.

Mientras que el método de Newton Cotes utiliza calcular el área debajo de una curva mediante interpolación. Para realizar este cálculo utilizamos polinomios con distintos grados, este grado dependerá de los puntos que utilicemos para calcular dicha aproximación.

El de Método de integración de Romberg combina dos aproximaciones de integración numérica para obtener un valor más exacto de manera más eficiente.

Referencias

Burden, R. L., & Faires, J. D. (2016). *Numerical Analysis* (10th ed.). Cengage Learning.

Chapra, S. C., & Canale, R. P. (2015). *Numerical Methods for Engineers* (7th ed.). McGraw-Hill Education.

Atkinson, K. E. (2008). *An Introduction to Numerical Analysis* (2nd ed.). Wiley.

Kincaid, D., & Cheney, W. (2009). *Numerical Analysis: Mathematics of Scientific Computing* (3rd ed.). American Mathematical Society.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing* (3rd ed.). Cambridge University Press.