

UNIVERSIDAD NACIONAL DEL ALTIPLANO
FACULTAD DE ING. ESTADÍSTICA E INFORMÁTICA
ESCUELA PROFESIONAL DE ING. ESTADÍSTICA E INFORMÁTICA



PROGRAMACIÓN NUMÉRICA

iDOCENTE:

Dr. TORRES CRUZ FRED

iALUMNOS:

- AQUINO SANDOVAL JEAN CARLOS
- PARICAHUA PARI CLIDE NEIL
- QUENAYA LOZA LUIS ANGEL
- TICONA MIRAMIRA ROBERTO ANGEL

iSECCIÓN:

”IV SEMESTRE - A”

PUNO - PERÚ

2025

Introducción

La programación numérica es una rama fundamental dentro de las ciencias computacionales y aplicadas, cuyo objetivo es proporcionar métodos y herramientas para resolver problemas matemáticos mediante el uso de algoritmos y programas informáticos. A través de ella, es posible aproximar soluciones a ecuaciones que no pueden resolverse de forma analítica, optimizar funciones complejas, y analizar sistemas dinámicos en diversas áreas del conocimiento.

El propósito de este libro es introducir al lector en los principios y aplicaciones prácticas de la programación numérica, utilizando lenguajes de programación de propósito científico como `Python` y `R`. A lo largo de los capítulos, se desarrollan los fundamentos teóricos y computacionales necesarios para comprender los métodos numéricos más empleados en ingeniería, física, economía y nutrición, entre otros campos.

En la primera unidad, se abordarán los conceptos básicos de la programación numérica, la definición de funciones matemáticas y sus restricciones, así como los métodos clásicos para el encontrar raíces de ecuaciones no lineales, tales como el método de bisección, el método de Newton-Raphson y el método de la secante. Estos métodos serán implementados paso a paso en código, permitiendo observar su comportamiento, eficiencia y convergencia.

En la segunda unidad, se explorarán métodos más avanzados, entre ellos el gradiente descendente y sus aplicaciones en optimización, la interpolación polinómica como técnica para aproximar funciones a partir de datos discretos, y la diferenciación numérica, que permite estimar derivadas de funciones cuando no se dispone de una expresión analítica. Cada tema será acompañado de ejemplos prácticos y ejercicios que ayudarán a reforzar la comprensión teórica mediante la experimentación computacional.

Además, se presentarán recomendaciones sobre buenas prácticas en la escritura de código científico, el uso eficiente de librerías numéricas, y la validación de resultados mediante análisis de error. El enfoque de este libro combina la precisión matemática con la aplicación práctica, fomentando el desarrollo de habilidades analíticas y computacionales en el lector.

En conjunto, este material busca no solo enseñar los fundamentos de la programación numérica, sino también promover un pensamiento crítico y estructurado al enfrentar problemas reales que requieren soluciones computacionales.

Índice general

Introducción	3
Índice de Figuras	9
I Unidad I	11
1. Programación Numérica	13
2. Funciones	15
2.1. Funciones a nuestro alrededor	15
2.2. Definición de función	15
2.3. Cuatro formas de representar una función	16
2.4. Gráficas de funciones	17
2.4.1. Gráficas de funciones por localización de puntos	17
2.4.2. La prueba de la recta vertical	18
2.5. Aplicación	19
3. Restricciones	21
3.1. Ejemplo	21
4. Método de Newton Raphson	25
4.1. Fundamento teórico	25
4.2. Condiciones de convergencia	25
4.3. Criterio de parada	26
4.4. Algoritmo del método de Newton-Raphson	26
4.5. Aplicación del método en Python	26
5. Método de Bisección	31

5.1. Idea fundamental del método	31
5.2. Criterios de parada	31
5.3. Ventajas del método	32
5.4. Desventajas	32
5.5. Aplicación del método en Python	32
6. Método de la Secante	37
6.1. Fundamento teórico	37
6.2. Interpretación geométrica	37
6.3. Convergencia	38
6.4. Ventajas y limitaciones	38
6.5. Aplicación del método en Python	38
7. Método de Punto Fijo	43
7.1. Definición y formulación básica	43
7.2. Condición de convergencia	43
7.3. Orden de convergencia	44
7.4. Errores y criterios de parada	44
7.5. Ventajas del método	44
7.6. Desventajas	45
7.7. Interpretación gráfica	45
7.8. Conclusión	45
7.9. Aplicación del método en Python	46
8. Método de Regula Falsi	51
8.1. Idea fundamental del método	51
8.2. Actualización del intervalo	51
8.3. Convergencia	52
8.4. Criterios de parada	52

ÍNDICE GENERAL	7
8.5. Ventajas y desventajas	52
8.6. Resumen conceptual	53
8.7. Aplicación del método en Python	53
II Unidad II	57
9. Gradiente de una función	59
10. Diferenciación Numérica	61
11. Interpolación	63
Conclusiones	65
Bibliografía	67

Índice de figuras

2.1. Diagrama de flechas de f	16
2.2. Cuatro formas de representar una función	17
2.3. La altura de la gráfica arriba del punto x es el valor de $f(x)$	18
2.4. Funciones lineal y constante	18
2.5. Prueba de la recta vertical	19
2.6. Graficando funciones con Python	20
3.1. Aplicación de restricciones en Python	22
4.1. Método de Newthon-Rapshon en Python	29
5.1. Método de Bisección en Python	36
6.1. Método de la Secante en Python	41
7.1. Método de Punto Fijo en Python	49
8.1. Método de Regula Falsi en Python	56

Parte I

Unidad I

1 Programación Numérica

La programación numérica es una disciplina que combina las matemáticas aplicadas y la informática con el objetivo de resolver problemas cuantitativos mediante métodos computacionales. Se centra en el diseño, análisis e implementación de algoritmos que permiten obtener soluciones aproximadas a ecuaciones, sistemas y modelos que, en la mayoría de los casos, no pueden resolverse de forma analítica (Burden & Faires, 2016; Chapra & Canale, 2015).

A diferencia de la programación convencional, que busca desarrollar aplicaciones funcionales o sistemas de información, la programación numérica se orienta a la resolución eficiente y precisa de problemas matemáticos. Entre sus principales aplicaciones se encuentran la simulación de fenómenos físicos y biológicos, la modelización económica, la ingeniería de datos, la inteligencia artificial y el análisis estadístico en ciencias de la salud (Press et al., 2007).

El objetivo fundamental de esta área es transformar problemas continuos en representaciones discretas que puedan ser tratadas por un computador. De esta forma, se logra aproximar soluciones a problemas de optimización, integración, derivación, interpolación, ajuste de curvas y resolución de ecuaciones diferenciales.

En términos prácticos, la programación numérica permite al investigador o profesional:

- Resolver ecuaciones no lineales mediante métodos iterativos.
- Aproximar derivadas e integrales de funciones cuando no se dispone de una forma analítica.
- Interpolar o ajustar funciones a datos experimentales.
- Optimizar funciones de una o varias variables bajo restricciones.
- Analizar errores y estimar la estabilidad numérica de los métodos empleados.

Actualmente, lenguajes como **Python**, **R**, **MATLAB** y **Julia** ofrecen bibliotecas especializadas que facilitan el desarrollo de algoritmos numéricos de alto rendimiento. Estos entornos han hecho posible que la programación numérica sea una herramienta accesible y poderosa para la investigación científica, la ingeniería y la docencia (Chapra & Canale, 2015).

En síntesis, la programación numérica constituye una base esencial para la solución computacional de problemas científicos y técnicos, integrando el razonamiento matemático con la capacidad de cómputo moderna.

2 Funciones

2.1 Funciones a nuestro alrededor

En casi todos los fenómenos físicos observamos que una cantidad depende de otra. Por ejemplo, la estatura de una persona depende de su edad, la temperatura de la fecha, el costo de enviar un paquete por correo depende de su peso. Usamos el término función para describir esta dependencia de una cantidad con respecto a otra. Esto es, decimos lo siguiente: (Stewart et al., 2001)

- La estatura es una función de la edad.
- La temperatura es una función de la fecha.
- El costo de enviar un paquete por correo depende de su peso.

2.2 Definición de función

Una función f es una regla que asigna a cada elemento x de un conjunto A exactamente un elemento, llamado $f(x)$, de un conjunto B .

Para hablar de una función, es necesario darle un nombre. Usaremos letras como f, g, h, \dots para representar funciones. Por ejemplo, podemos usar la letra f para representar una regla como sigue:

" f " es la regla "elevar al cuadrado el número"

cuando escribimos $f(2)$ queremos decir "aplicar la regla f al número 2". La aplicación de la regla da $f(2) = 2^2 = 4$. Del mismo modo, $f(3) = 3^2 = 9$, $f(4) = 4^2 = 16$, y en general $f(x) = x^2$. (Stewart et al., 2001)

Por lo general consideramos funciones para las cuales los conjuntos A y B son conjuntos de número reales. El símbolo $f(x)$ se lee "f de x" o "f en x" y se denomina valor de f en x , o la imagen de x bajo f . El conjunto A recibe el nombre de dominio de la función. El rango de f es el conjunto de todos los valores posibles de $f(x)$ cuando x varía en todo el dominio. El símbolo que representa un número arbitrario del dominio de una función f se llama variable independiente. El símbolo que representa un número en el rango de f se llama variable dependiente. Por tanto, si escribimos $y = f(x)$, entonces x es la variable independiente y y es la variable dependiente. (Stewart et al., 2001)

Es útil considerar una función como una [máquina](#). Si x está en el dominio de la función f , entonces cuando x entra a la máquina, es aceptada como entrada y la máquina produce una salida $f(x)$ de acuerdo con la regla de la función. Así, podemos considerar el dominio como el conjunto de todas las posibles entradas y el rango como el conjunto de todas las posibles salidas. (Stewart et al., [2001](#))

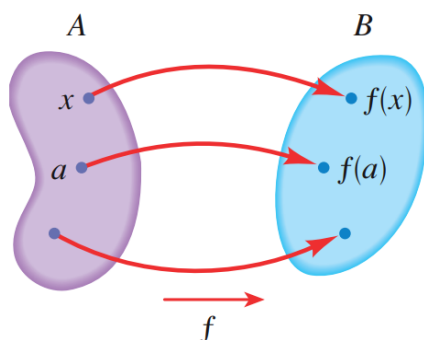


Figura 2.1: Diagrama de flechas de f

2.3 Cuatro formas de representar una función

Para entender mejor lo que es una función, podemos describir una función específica en las siguientes cuatro formas: (Stewart et al., [2016](#))

- verbalmente (por descripción en palabras)
- algebraicamente (por una fórmula explícita)
- visualmente (por una gráfica)
- numéricamente (por una tabla de valores)

Una función individual puede estar representada en las cuatro formas, y con frecuencia es útil pasar de una representación a otra para adquirir más conocimientos sobre la función. No obstante, ciertas funciones se describen en forma más natural por medio de un método que por los otros. Un ejemplo de una descripción verbal es la siguiente regla para convertir entre escalas de temperatura: (Stewart et al., [2016](#))

”Para hallar el equivalente Fahrenheit de una temperatura Celsius, multiplicar por $\frac{9}{5}$ la temperatura Celsius y luego sumar 32”

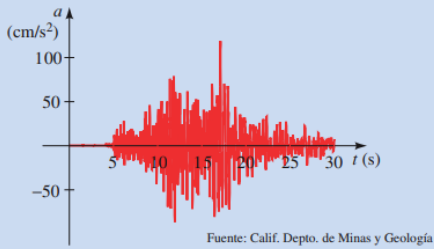
Verbal Con palabras: $P(t)$ es la “población del mundo en el instante t ” Relación de la población P y el tiempo t	Algebraica Por medio de una fórmula: $A(r) = \pi r^2$ Área de un círculo														
Visual Por medio de una gráfica:  Aceleración vertical durante un terremoto	Numérica Por medio de una tabla de valores: <table><tr><th>w (onzas)</th><th>$C(w)$ (dólares)</th></tr><tr><td>$0 < w \leq 1$</td><td>0.37</td></tr><tr><td>$1 < w \leq 2$</td><td>0.60</td></tr><tr><td>$2 < w \leq 3$</td><td>0.83</td></tr><tr><td>$3 < w \leq 4$</td><td>1.06</td></tr><tr><td>$4 < w \leq 5$</td><td>1.29</td></tr><tr><td>\vdots</td><td>\vdots</td></tr></table> Costo de enviar una carta por correo de primera clase	w (onzas)	$C(w)$ (dólares)	$0 < w \leq 1$	0.37	$1 < w \leq 2$	0.60	$2 < w \leq 3$	0.83	$3 < w \leq 4$	1.06	$4 < w \leq 5$	1.29	\vdots	\vdots
w (onzas)	$C(w)$ (dólares)														
$0 < w \leq 1$	0.37														
$1 < w \leq 2$	0.60														
$2 < w \leq 3$	0.83														
$3 < w \leq 4$	1.06														
$4 < w \leq 5$	1.29														
\vdots	\vdots														

Figura 2.2: Cuatro formas de representar una función

2.4 Gráficas de funciones

2.4.1 Gráficas de funciones por localización de puntos

Para graficar una función f localizamos los puntos $(x, f(x))$ en un plano de coordenadas. En otras palabras, localizamos los puntos (x, y) cuya coordenada x es una entrada y cuya coordenada y es la correspondiente salida de la función. (Stewart et al., 2016)

Si f es una función con dominio A , entonces la gráfica de f es el conjunto de pares ordenados

$$(x, f(x)) | x \in A$$

localizados en un plano de coordendas. En otras palabras, la gráfica de f es el conjunto de todos los puntos (x, y) tales que $y = f(x)$; esto es, la gráfica de f es la gráfica de la ecuación $y = f(x)$.

La gráfica de una función f da un retrato del comportamiento o “historia de la vida” de la función. Podemos leer el valor de $f(x)$ a partir de la gráfica como la altura de la gráfica arriba del punto x . (Stewart et al., 2016)

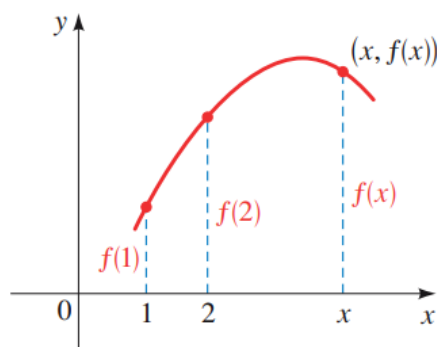


Figura 2.3: La altura de la gráfica arriba del punto x es el valor de $f(x)$

Una función f de la forma $f(x) = mx + b$ se denomina función lineal porque su gráfica es la gráfica de la ecuación $y = mx + b$, que representa una recta con pendiente m y punto de intersección b en y . Un caso especial de una función lineal se presenta cuando la pendiente es $m = 0$. La función $f(x) = b$, donde b es un número determinado, recibe el nombre de *funcinconstante* porque todos sus valores son el mismo número, es decir, b . Su gráfica es la recta horizontal $y = b$. (Stewart et al., 2016)

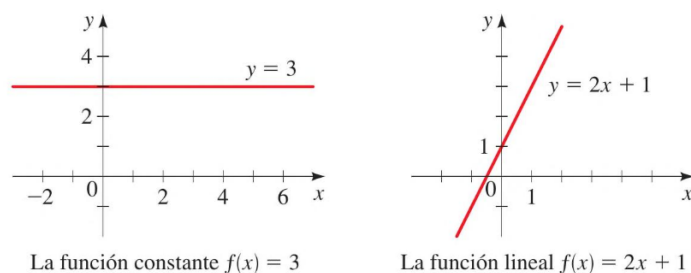


Figura 2.4: Funciones lineal y constante

2.4.2 La prueba de la recta vertical

La gráfica de una función es una curva en el plano xy . Pero surge la pregunta. ¿Cuáles curvas del plano xy son gráficas de funciones? ESTo se contesta por medio de la prueba siguiente. (Stewart et al., 2016)

Una curva en el plano de coordenadas es la gráfica de una función si y sólo si ninguna recta vertical cruza la curva más de una vez.

Podemos ver la Figura 2.5 para entender por qué la Prueba de la Recta Vertical es verdadera. Si cada recta vertical $x = a$ cruza la curva sólo una vez en (a, b) , entonces exactamente un valor funcional está definido por $f(a) = b$. Pero si una recta $x = a$ cruza la curva dos veces, en (a, b) y en (a, c) , entonces la curva no puede representar una función porque una función no puede asignar dos valores diferentes a a .

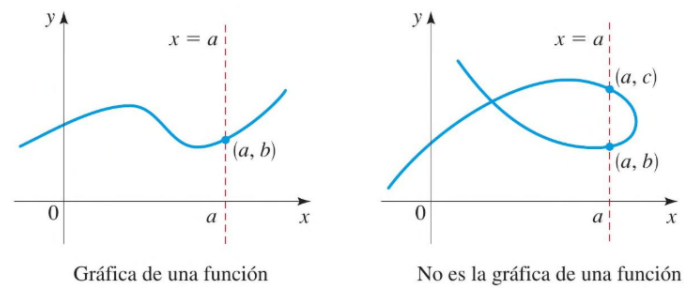


Figura 2.5: Prueba de la recta vertical

2.5 Aplicación

Se presentará un código en Python que sea capaz de graficar funciones, según los datos de entrada que se pidan.

```

1 import sympy as sp
2 import numpy as np
3 import matplotlib.pyplot as plt
4
5 # === Ingreso de la función por el usuario ===
6 expr_str = input("Ingrese la función en términos de x (ejemplo: sin(x), x**2 + 3*x
7                  ↪ - 5, exp(-x)*cos(x)): ")
8
9 # === Definición de variable simbólica ===
10 x = sp.Symbol('x')
11
12 # === Conversión del texto a expresión simbólica ===
13 try:
14     expr = sp.sympify(expr_str)
15 except sp.SympifyError:
16     print("Error: la función ingresada no es válida.")
17     exit()
18
19 # === Creación de función numérica evaluable ===
20 f = sp.lambdify(x, expr, modules=['numpy'])
21
22 # === Intervalo de graficación ===
23 x_vals = np.linspace(-10, 10, 400)
24 y_vals = f(x_vals)
25
26 # === Graficar ===
27 plt.figure(figsize=(7,5))
28 plt.plot(x_vals, y_vals, label=f"$f(x) = {sp.latex(expr)}$", color='navy')
29 plt.title("Gráfica de la función ingresada", fontsize=13)
30 plt.xlabel("x")
31 plt.ylabel("f(x)")
32 plt.grid(True, linestyle='--', alpha=0.6)
33 plt.axhline(0, color='black', linewidth=1)
34 plt.axvline(0, color='black', linewidth=1)
35 plt.legend()

```

```
35 plt.show()
```

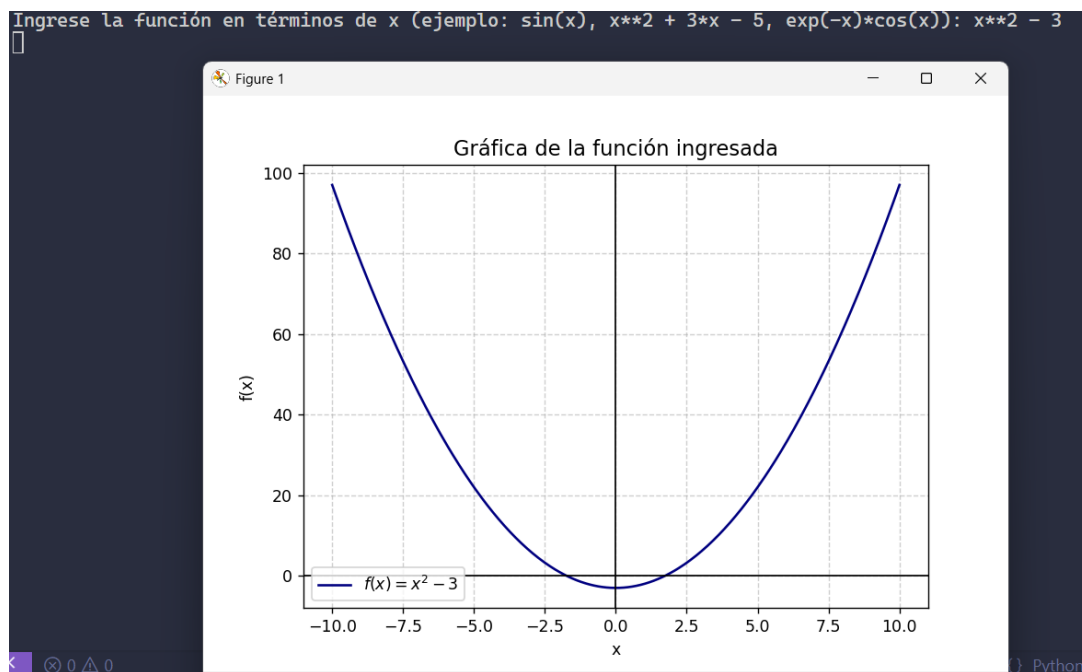


Figura 2.6: Graficando funciones con Python

3 Restricciones

Las restricciones son condiciones que limitan el conjunto de valores posibles que pueden tomar las variables en un problema matemático. Estas limitaciones definen el espacio factible o región factible, dentro del cual se busca una o más soluciones que cumplan con determinados criterios. Las restricciones pueden ser igualdades (por ejemplo, $x + y = 10$) o desigualdades (por ejemplo, $x \geq 0$), y desempeñan un papel fundamental en los problemas de optimización, programación lineal y análisis numérico. (Chapra & Canale, 2015)

Un sistema de ecuaciones consiste en un conjunto de ecuaciones que comparten las mismas variables. Resolverlo implica encontrar los valores que satisfacen todas las ecuaciones simultáneamente. Estos sistemas pueden clasificarse en lineales y no lineales, y sus métodos de resolución varían desde técnicas algebraicas clásicas (como la sustitución o igualación) hasta procedimientos numéricos avanzados (como el método de Gauss-Seidel o Newton-Raphson). (Chapra & Canale, 2015)

En el contexto de la programación numérica, las restricciones se representan y manipulan mediante código, permitiendo no solo resolver el sistema de ecuaciones sino también visualizar gráficamente la región factible y el punto óptimo. Esta representación es especialmente útil para comprender la interacción entre las ecuaciones y las limitaciones impuestas, facilitando el análisis y la toma de decisiones en problemas aplicados de ingeniería, economía y ciencias computacionales. (Chapra & Canale, 2015)

3.1 Ejemplo

Un desarrollador tiene 15 horas semanales para dedicar al desarrollo de software de front-end (x) y back-end (y). Además:

- Debe dedicar al menos 5 horas al desarrollo de front-end para cumplir con los entregables del cliente.
- El tiempo total no puede exceder 15 horas por restricciones de tiempo del sprint.

Formule las restricciones, represéntelas gráficamente e identifique las combinaciones posibles de tiempo a invertir en cada actividad.

Variables

- x = horas dedicadas al front-end
- y = horas dedicadas al back-end

Restricciones

1. Debe dedicar al menos 5 horas al mes.

$$\blacksquare x \geq 5$$

2. El tiempo total no puede exceder a 15 horas.

$$\blacksquare x + y \leq 15$$

Gráfico generado con python



Figura 3.1: Aplicación de restricciones en Python

Interpretación del gráfico

En la representación gráfica se observa la región factible determinada por las restricciones planteadas:

- $x \geq 5$: el desarrollador debe dedicar al menos 5 horas al front-end. Esta condición aparece como una línea vertical en $x = 5$ y la región válida se encuentra a la derecha de ella.
- $y \geq 0$: el tiempo dedicado al back-end no puede ser negativo, por lo que la región se limita a la parte superior del eje x .
- $x + y \leq 15$: la suma de horas asignadas a front-end y back-end no puede superar las 15 horas semanales. Gráficamente, corresponde a la semirrecta bajo la línea $x + y = 15$.

La intersección de estas tres restricciones genera una región triangular factible delimitada por los puntos (5, 0), (15, 0) y (5, 10). Esto significa que cualquier combinación de horas ubicada dentro o sobre este triángulo cumple con las condiciones del problema. Por ejemplo, el desarrollador podría dedicar:

- 5 horas a front-end y 10 horas a back-end,
- 10 horas a front-end y 5 horas a back-end,
- o bien 15 horas únicamente a front-end.

En conclusión, la región sombreada representa todas las combinaciones posibles de tiempo de trabajo entre front-end y back-end que respetan tanto el mínimo requerido en front-end como la restricción máxima de 15 horas semanales.

Código en Python

```

1      # Función para preparar expresiones lineales
2  def preparar_expresion(expr: str) -> str:
3      expr = expr.replace(" ", "")          # quitar espacios
4      expr = expr.replace("^", "**")        # potencia
5      expr = expr.replace("-x", "-1*x")    # caso -x
6      expr = expr.replace("+x", "+1*x")    # caso +x
7      if expr.startswith("x"):             # si empieza con x
8          expr = "1*" + expr
9      expr = expr.replace("x", "*x")       # poner multiplicación
10     expr = expr.replace("**x", "*x")     # corregir si se duplicó
11     return expr
12
13     # Restricciones:
14     # 1) x = 5 (vertical)
15     # 2) x + y = 15 -> y = -x + 15
16     func2 = preparar_expresion("-x+15")
17
18     # Rango de la gráfica
19     xmin, xmax = -5, 20
20     ymin, ymax = -5, 20
21
22     # Recorremos el plano
23     for y in range(ymax, ymin - 1, -1):

```

```
24 linea = ""
25 for x in range(xmin, xmax + 1):
26     # Recta 1:  $x = 5$ 
27     cond1 = (x == 5)
28
29     # Recta 2:  $y = -x + 15$ 
30     try:
31         y2 = eval(func2)
32     except:
33         y2 = None
34     cond2 = (y2 is not None and abs(y - y2) < 0.5)
35
36     # Región factible:  $x \geq 5$ ,  $y \geq 0$ ,  $x+y \leq 15$ 
37     region = (x >= 5 and y >= 0 and x + y <= 15)
38
39     # Qué dibujar
40     if cond1 and cond2:
41         linea += "#"
42     elif cond1:
43         linea += "*"
44     elif cond2:
45         linea += "o"
46     elif x == 0 and y == 0:
47         linea += "+"
48     elif x == 0:
49         linea += "|"
50     elif y == 0:
51         linea += "-"
52     elif region:
53         linea += "."
54     else:
55         linea += " "
56     print(linea)
57
58     # Leyenda
59     print("\nLeyenda del gráfico:")
60     print(" * =  $x = 5$ ")
61     print(" o =  $x + y = 15$ ")
62     print(" # = Intersección")
63     print(" . = Región factible")
64     print(" | = Eje Y")
65     print(" - = Eje X")
66     print(" + = Origen (0,0)")
```


4 Método de Newton Raphson

El método de Newton-Raphson es una técnica iterativa utilizada para encontrar raíces de ecuaciones no lineales de la forma:

$$f(x) = 0$$

Es uno de los métodos más eficaces y ampliamente utilizados debido a su rapidez de convergencia cuando se cumplen las condiciones necesarias. Fue propuesto originalmente por Isaac Newton y posteriormente generalizado por Joseph Raphson (Chapra & Canale, 2015)

4.1 Fundamento teórico

La idea básica del método consiste en aproximar la función $f(x)$ mediante su expansión de Taylor alrededor de un punto x_i y despreciar los términos de orden superior:

$$f(x) \approx f(x_i) + f'(x_i)(x - x_i)$$

Para hallar la raíz, se hace $f(x) = 0$, y despejando x se obtiene la fórmula iterativa:

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

Esta expresión permite calcular una mejor aproximación de la raíz en cada iteración. El proceso continúa hasta que el valor de x_{i+1} converge a la raíz real con una tolerancia previamente establecida (Chapra & Canale, 2015)

4.2 Condiciones de convergencia

El método de Newton-Raphson presenta convergencia cuadrática, es decir, el error disminuye aproximadamente al cuadrado en cada iteración, siempre que se cumplan las siguientes condiciones (Sullivan, 2004):

- La función $f(x)$ es continua y derivable en un intervalo que contiene la raíz buscada.
- La derivada $f'(x)$ no se anula en el entorno de la raíz.
- La estimación inicial x_0 está suficientemente cerca de la raíz real.

Sin embargo, si $f'(x_i)$ se aproxima a cero o si la estimación inicial está muy alejada, el método puede divergir o generar oscilaciones (Burden & Faires, 2016).

4.3 Criterio de parada

El proceso iterativo se detiene cuando se cumple alguna de las siguientes condiciones (Chapra & Canale, 2015):

- $|f(x_{i+1})| < \varepsilon$
- $|x_{i+1} - x_i| < \varepsilon$

donde ε es la tolerancia o el error máximo permitido.

4.4 Algoritmo del método de Newton-Raphson

1. Elegir una estimación inicial x_0 .

2. Calcular $f(x_0)$ y $f'(x_0)$.

3. Evaluar:

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

4. Repetir el proceso:

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

hasta que se cumpla el criterio de convergencia.

4.5 Aplicación del método en Python

El objetivo fue desarrollar en el lenguaje de programación Python un programa que permita al usuario ingresar una función $f(x)$ y graficarla en un intervalo definido. Esta etapa inicial tiene como propósito ayudar al usuario a identificar visualmente las posibles raíces y decidir si desea aplicar el método de Newton-Raphson.

En caso afirmativo, el programa solicita un valor inicial x_1 basado en la observación de la gráfica y ejecuta el algoritmo iterativo de Newton-Raphson hasta que la diferencia entre dos aproximaciones sucesivas sea menor a una tolerancia predefinida. Finalmente, el programa muestra en pantalla la raíz aproximada encontrada y el número de iteraciones necesarias para alcanzarla.

Este enfoque combina la interpretación gráfica con el análisis numérico, promoviendo una comprensión más completa del comportamiento de la función y de la eficacia del método iterativo.

Entrada

Una cadena de texto que representa una función matemática.

$$f(x) = x^3 - x - 1$$

Salida

- Gráfica para evaluar si realizar o no el método.
- La raíz encontrada.
- Número de iteraciones realizadas hasta encontrar la raíz.

Código

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # === 1. Ingreso de la función ===
5 func_str = input("Ingrese la función f(x): ") # Ejemplo: x**3 - x - 1
6
7 # Definimos la función y su derivada (usando derivada numérica)
8 def f(x):
9     return eval(func_str)
10
11 def f_prime(x, h=1e-6): # derivada numérica
12     return (f(x + h) - f(x - h)) / (2 * h)
13
14 # === 2. Graficar la función ===
15 xmin = float(input("Ingrese el valor mínimo de x: "))
16 xmax = float(input("Ingrese el valor máximo de x: "))
17
18 x = np.linspace(xmin, xmax, 400)
19 y = f(x)
20
21 plt.figure(figsize=(8, 5))
22 plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
23 plt.axhline(0, color='black', linestyle='--')
24 plt.axvline(0, color='black', linestyle='--')
25 plt.title("Gráfico de la función ingresada")
26 plt.xlabel("x")
27 plt.ylabel("f(x)")
28 plt.legend()
29 plt.grid(True)
30 plt.show()
31
32 # === 3. Pregunta si desea aplicar Newton-Raphson ===
33 op = input("¿Desea encontrar una raíz con el método de Newton-Raphson? (s/n): ").
34     ↪ lower()
35
36 if op == "s":
37     # === 4. Ingreso de punto inicial ===

```

```
37 x0 = float(input("Basado en la gráfica, ingrese el valor inicial x1: "))
38
39 # Parámetros del método
40 tol = 1e-6
41 max_iter = 100
42
43 # Iteraciones
44 for i in range(1, max_iter + 1):
45     fx = f(x0)
46     fpx = f_prime(x0)
47
48     if fpx == 0:
49         print(f"La derivada es cero en x = {x0}. El método no puede continuar.")
50         break
51
52     x1 = x0 - fx / fpx
53
54     # Verificar convergencia
55     if abs(x1 - x0) < tol:
56         print(f"\n Raíz aproximada encontrada: {x1:.6f}")
57         print(f"Iteraciones realizadas: {i}")
58         break
59
60     x0 = x1
61 else:
62     print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")
63
64 else:
65     print("No se aplicó el método de Newton-Raphson.")
```

Ejecución

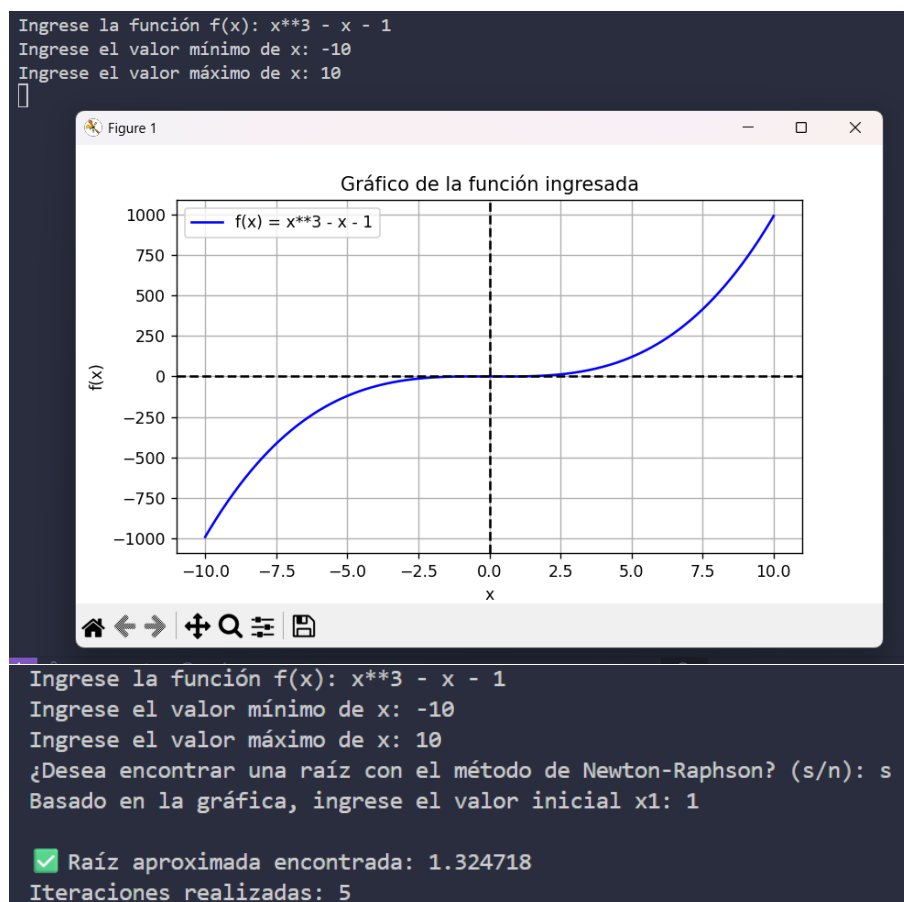


Figura 4.1: Método de Newton-Raphson en Python

5 Método de Bisección

El método de bisección es uno de los algoritmos más simples, robustos y confiables para la localización de raíces de ecuaciones no lineales de la forma

$$f(x) = 0.$$

Su fundamento teórico se basa en el Teorema del Valor Intermedio, el cual garantiza la existencia de una raíz siempre que la función sea continua en un intervalo cerrado $[a, b]$ y que se cumpla la condición

$$f(a) f(b) < 0.$$

Esto indica que en el intervalo ocurre un cambio de signo y, por tanto, debe existir al menos una raíz en su interior (Burden & Faires, 2016).

5.1 Idea fundamental del método

El método consiste en dividir sucesivamente el intervalo $[a, b]$ por su punto medio. Si llamamos

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

entonces evaluamos $f(c)$ y determinamos en qué subintervalo persiste el cambio de signo:

- Si $f(a) f(c) < 0$, la raíz se encuentra en $[a, c]$.
- Si $f(c) f(b) < 0$, la raíz se encuentra en $[c, b]$.

Este proceso se repite recursivamente, produciendo intervalos cada vez más pequeños. Debido a su naturaleza, el método siempre converge cuando la función es continua y se inicia con un intervalo válido, aunque la convergencia es relativamente lenta (Chapra & Canale, 2015).

5.2 Criterios de parada

El método se detiene cuando se cumple alguna de las siguientes condiciones:

1. El ancho del intervalo es menor que una tolerancia establecida:

$$|b - a| < \varepsilon.$$

2. El valor funcional es cercano a cero:

$$|f(c)| < \delta.$$

3. Se alcanza un número máximo de iteraciones:

$$n \geq n_{\text{máx}}.$$

El error después de n iteraciones está acotado por

$$|x - c_n| \leq \frac{b - a}{2^n},$$

lo cual permite estimar de manera precisa el número de pasos necesarios para obtener una tolerancia deseada (Süli & Mayers, 2003).

5.3 Ventajas del método

El método de bisección posee varias ventajas importantes:

- Garantiza convergencia siempre que exista un cambio de signo en el intervalo.
- Es simple de implementar.
- Permite una estimación directa del error en cada iteración.
- Es estable numéricamente y no requiere derivadas.

Estas características lo convierten en un método ideal para iniciar procesos de localización de raíces o para validar aproximaciones obtenidas mediante métodos más rápidos pero menos robustos, como Newton-Raphson o la secante (Press et al., 2007).

5.4 Desventajas

A pesar de su robustez, el método de bisección presenta algunas limitaciones:

- La convergencia es lineal y relativamente lenta en comparación con otros métodos.
- Requiere conocer un intervalo donde la función cambie de signo.
- No obtiene raíces múltiples o raíces donde la función no cambia de signo.

Aun así, su combinación de simplicidad y fiabilidad lo convierte en un método fundamental dentro de los algoritmos de análisis numérico (Burden & Faires, 2016).

5.5 Aplicación del método en Python

Se desarrolló en Python un programa que permite al usuario ingresar una función matemática y visualizar su gráfico en un intervalo definido. Esta etapa inicial facilita la elección del intervalo $[a, b]$ donde la función cambia de signo.

Una vez seleccionado el intervalo, el programa ejecuta el método de bisección iterativamente, mostrando en pantalla una tabla con los valores de a , b , c , $f(c)$ y el error de cada iteración. El proceso finaliza cuando se cumple la condición de tolerancia o se alcanza el número máximo de iteraciones. Finalmente, se presenta la raíz aproximada encontrada.

Este enfoque combina el análisis gráfico con la interpretación numérica, permitiendo comprender mejor el comportamiento de la función y la estabilidad del método.

Entrada

Una cadena de texto que representa una función matemática.

$$f(x) = x^3 - x - 1$$

Salida

- Gráfica para evaluar el intervalo de búsqueda.
- Tabla con las iteraciones del método.
- Raíz aproximada y número de iteraciones realizadas.

Restricciones

El método requiere que:

- $f(x)$ sea continua en el intervalo $[a, b]$.
- Se cumpla la condición $f(a) \cdot f(b) < 0$.

Código

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # === 1. Ingreso de la función ===
5 func_str = input("Ingrese la función f(x): ") # Ejemplo: x**3 - x - 1
6
7 # Definimos la función
8 def f(x):
9     return eval(func_str, {"np": np, "x": x})
10
11 # === 2. Graficar la función ===
12 xmin = float(input("Ingrese el valor mínimo de x: "))
13 xmax = float(input("Ingrese el valor máximo de x: "))
14
15 x = np.linspace(xmin, xmax, 400)
```

```

16 y = f(x)
17
18 plt.figure(figsize=(8, 5))
19 plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
20 plt.axhline(0, color='black', linestyle='--')
21 plt.axvline(0, color='black', linestyle='--')
22 plt.title("Gráfico de la función ingresada")
23 plt.xlabel("x")
24 plt.ylabel("f(x)")
25 plt.legend()
26 plt.grid(True)
27 plt.show()
28
29 # === 3. Pregunta si desea aplicar el método de Bisección ===
30 op = input("¿Desea encontrar una raíz con el método de Bisección? (s/n): ").lower()
31
32 if op == "s":
33     # === 4. Ingreso del intervalo inicial ===
34     a = float(input("Ingrese el extremo izquierdo del intervalo (a): "))
35     b = float(input("Ingrese el extremo derecho del intervalo (b): "))
36
37     # Verificación del cambio de signo
38     if f(a) * f(b) > 0:
39         print("\n No hay cambio de signo en el intervalo [a, b]. Intente con otro intervalo  

40             ↪ .")
41     else:
42         # Parámetros del método
43         tol = 1e-6
44         max_iter = 100
45
46         print("\nIteración |      a      |      b      |      c      |      f(c)      |  

47             ↪ Error")
48         print("-----")
49
50         for i in range(1, max_iter + 1):
51             c = (a + b) / 2
52             fc = f(c)
53             error = abs(b - a) / 2
54
55             print(f"{i:9d} | {a:10.6f} | {b:10.6f} | {c:10.6f} | {fc:10.6f} | {error:10.6f}")
56
57             if abs(fc) < tol or error < tol:
58                 print(f"\n Raíz aproximada encontrada: {c:.6f}")
59                 print(f"Iteraciones realizadas: {i}")
60                 break
61
62             if f(a) * fc < 0:
63                 b = c
64             else:
65                 a = c
66
67         print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")

```

```
66 else:  
67 print("No se aplicó el método de Bisección.")
```

Ejecución

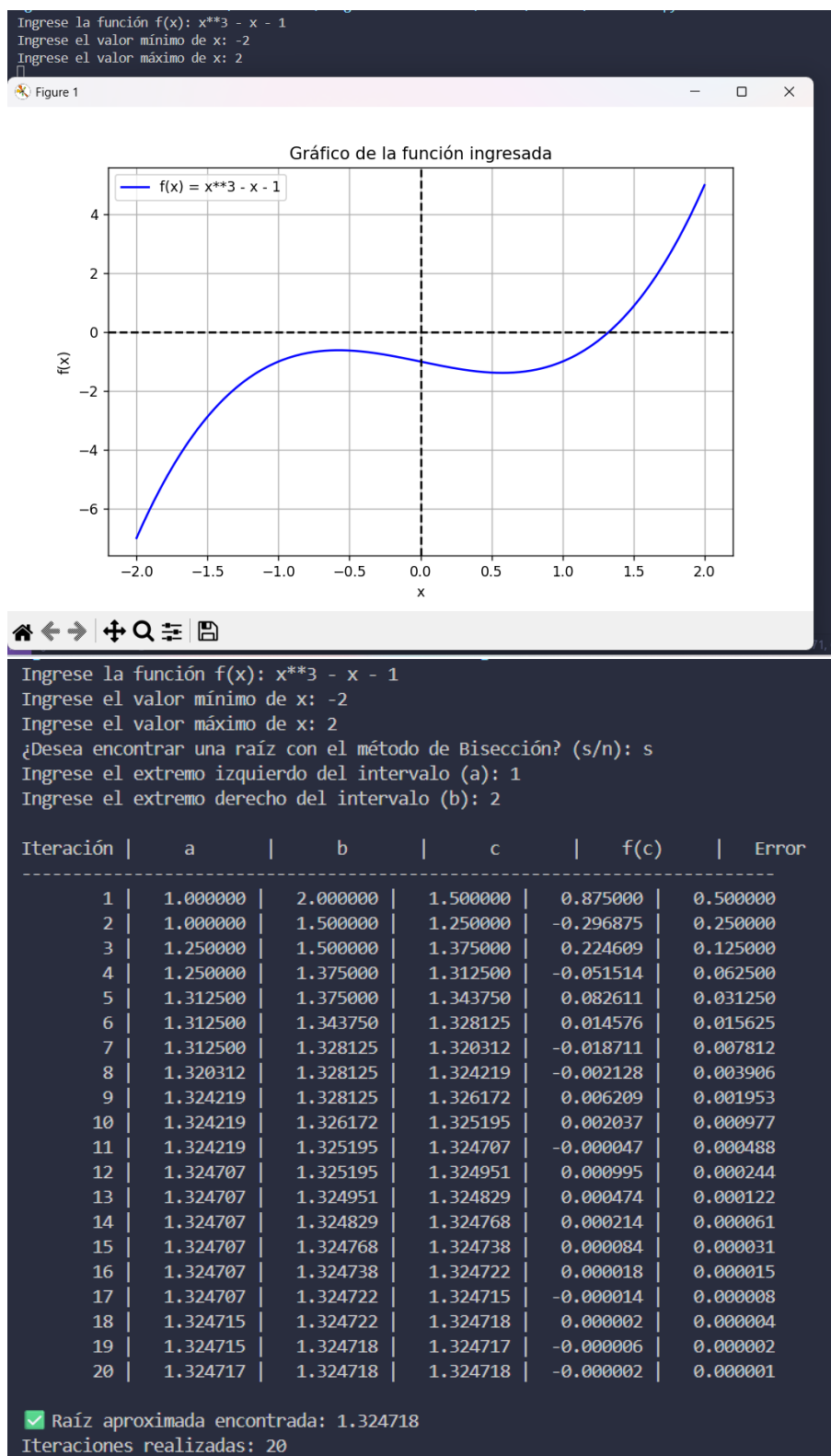


Figura 5.1: Método de Bisección en Python

6 Método de la Secante

El método de la secante es una técnica numérica ampliamente utilizada para la resolución de ecuaciones no lineales, especialmente en contextos donde la derivada de la función no está disponible o resulta difícil de calcular. A diferencia del método de Newton–Raphson, que requiere la evaluación explícita de la derivada, la secante se basa en una aproximación numérica de la misma utilizando dos puntos consecutivos de la función (Burden & Faires, 2011; Chapra & Canale, 2015).

6.1 Fundamento teórico

El método surge de la idea de aproximar la derivada de la función mediante un cociente incremental, reemplazando la derivada verdadera por:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}.$$

Sustituyendo esta expresión en la fórmula de Newton–Raphson se obtiene la iteración del método de la secante:

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

La deducción formal y el análisis matemático del método pueden consultarse en textos clásicos de análisis numérico (Burden & Faires, 2011), así como en manuales modernos de métodos computacionales (Atkinson, 2009; Chapra & Canale, 2015).

6.2 Interpretación geométrica

Geométricamente, el método consiste en trazar la recta secante que pasa por los puntos $(x_{n-1}, f(x_{n-1}))$ y $(x_n, f(x_n))$, y encontrar su intersección con el eje x . Esta construcción geométrica proporciona una aproximación sucesiva a la raíz de la ecuación $f(x) = 0$ (Sullivan, 2004).

La secante generada en cada iteración reemplaza el uso de la tangente del método de Newton, permitiendo que el método avance sin necesidad de calcular derivadas. Esta propiedad convierte a la secante en un algoritmo eficiente en problemas aplicados donde la evaluación de $f'(x)$ puede ser costosa o inestable (Cheney & Kincaid, 2009).

6.3 Convergencia

El método posee una convergencia superlineal, cuya razón de convergencia es aproximadamente 1.618, el número áureo. Aunque es menos rápido que Newton–Raphson, suele ser más eficiente cuando la derivada no está disponible o cuando su cálculo es propenso a errores numéricos (Atkinson, 2009). Sin embargo, su éxito requiere seleccionar dos valores iniciales adecuados para evitar divisiones entre valores de función muy pequeños o iteraciones divergentes (Burden & Faires, 2011).

6.4 Ventajas y limitaciones

Entre sus principales ventajas destacan:

- No requiere el cálculo de la derivada de la función.
- Tiende a ser más estable que Newton–Raphson cuando la derivada es difícil de evaluar.
- Posee una tasa de convergencia superior al método de bisección.

Sus principales limitaciones incluyen:

- No garantiza convergencia global.
- Requiere dos valores iniciales definidos.
- Puede divergir si los valores iniciales no son adecuados o si la función presenta discontinuidades o múltiples raíces cercanas.

Para un análisis detallado de sus propiedades teóricas y aplicaciones prácticas pueden consultarse referencias especializadas en análisis numérico (Atkinson, 2009; Burden & Faires, 2011; Chapra & Canale, 2015; Cheney & Kincaid, 2009).

6.5 Aplicación del método en Python

Se desarrolló un programa en Python que permite al usuario ingresar una función $f(x)$ y graficarla en un intervalo definido. Esta etapa inicial facilita la identificación visual de posibles raíces y la selección de los valores iniciales x_0 y x_1 .

Luego, el programa aplica el método de la secante iterativamente hasta alcanzar una tolerancia establecida o un número máximo de iteraciones. En cada iteración, se muestran los valores de x_0 , x_1 , $f(x_0)$, $f(x_1)$, la nueva aproximación x_2 y el error absoluto. Finalmente, se imprime la raíz aproximada encontrada y el número de iteraciones necesarias.

Este procedimiento combina la exploración gráfica con el razonamiento numérico, fomentando una comprensión más completa de la convergencia y del comportamiento de la función.

Entrada

Una cadena de texto que representa una función matemática.

$$f(x) = x^3 - x - 1$$

Salida

- Gráfica para visualizar la función y elegir los puntos iniciales.
- Tabla con los valores de cada iteración.
- Raíz aproximada y número de iteraciones necesarias.

Restricciones

- $f(x)$ debe ser continua en el intervalo analizado.
- Los valores iniciales x_0 y x_1 deben ser distintos y cercanos a la raíz.

Código

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # === 1. Ingreso de la función ===
5 func_str = input("Ingrese la función f(x): ") # Ejemplo: x**3 - x - 1
6
7 # Definimos la función
8 def f(x):
9     return eval(func_str, {"np": np, "x": x})
10
11 # === 2. Graficar la función ===
12 xmin = float(input("Ingrese el valor mínimo de x: "))
13 xmax = float(input("Ingrese el valor máximo de x: "))
14
15 x = np.linspace(xmin, xmax, 400)
16 y = f(x)
17
18 plt.figure(figsize=(8, 5))
19 plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
20 plt.axhline(0, color='black', linestyle='--')
21 plt.axvline(0, color='black', linestyle='--')
22 plt.title("Gráfico de la función ingresada")
23 plt.xlabel("x")
24 plt.ylabel("f(x)")
25 plt.legend()

```

```

26 plt.grid(True)
27 plt.show()
28
29 # === 3. Pregunta si desea aplicar el método de la secante ===
30 op = input("¿Desea encontrar una raíz con el método de la Secante? (s/n): ").lower
    ↪ ()
31
32 if op == "s":
33 # === 4. Ingreso de puntos iniciales ===
34 x0 = float(input("Ingrese el primer valor inicial x0: "))
35 x1 = float(input("Ingrese el segundo valor inicial x1: "))
36
37 # Parámetros del método
38 tol = 1e-6
39 max_iter = 100
40
41 print("\nIteración |      x0      |      x1      |      f(x0)      |      f(x1)      |
    ↪      x2      |      Error")
42 print("-----")
43
44 for i in range(1, max_iter + 1):
45 f0 = f(x0)
46 f1 = f(x1)
47
48 if f1 - f0 == 0:
49 print(f"\n División por cero en la iteración {i}. El método no puede continuar.")
50 break
51
52 x2 = x1 - f1 * (x1 - x0) / (f1 - f0)
53 error = abs(x2 - x1)
54
55 print(f"{i:9d} | {x0:10.6f} | {x1:10.6f} | {f0:12.6f} | {f1:12.6f} | {x2:10.6f} | {
    ↪ error:10.6f}")
56
57 if error < tol:
58 print(f"\n Raíz aproximada encontrada: {x2:.6f}")
59 print(f"Iteraciones realizadas: {i}")
60 break
61
62 x0, x1 = x1, x2
63
64 else:
65 print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")
66 else:
67 print("No se aplicó el método de la Secante.")

```

Ejecución

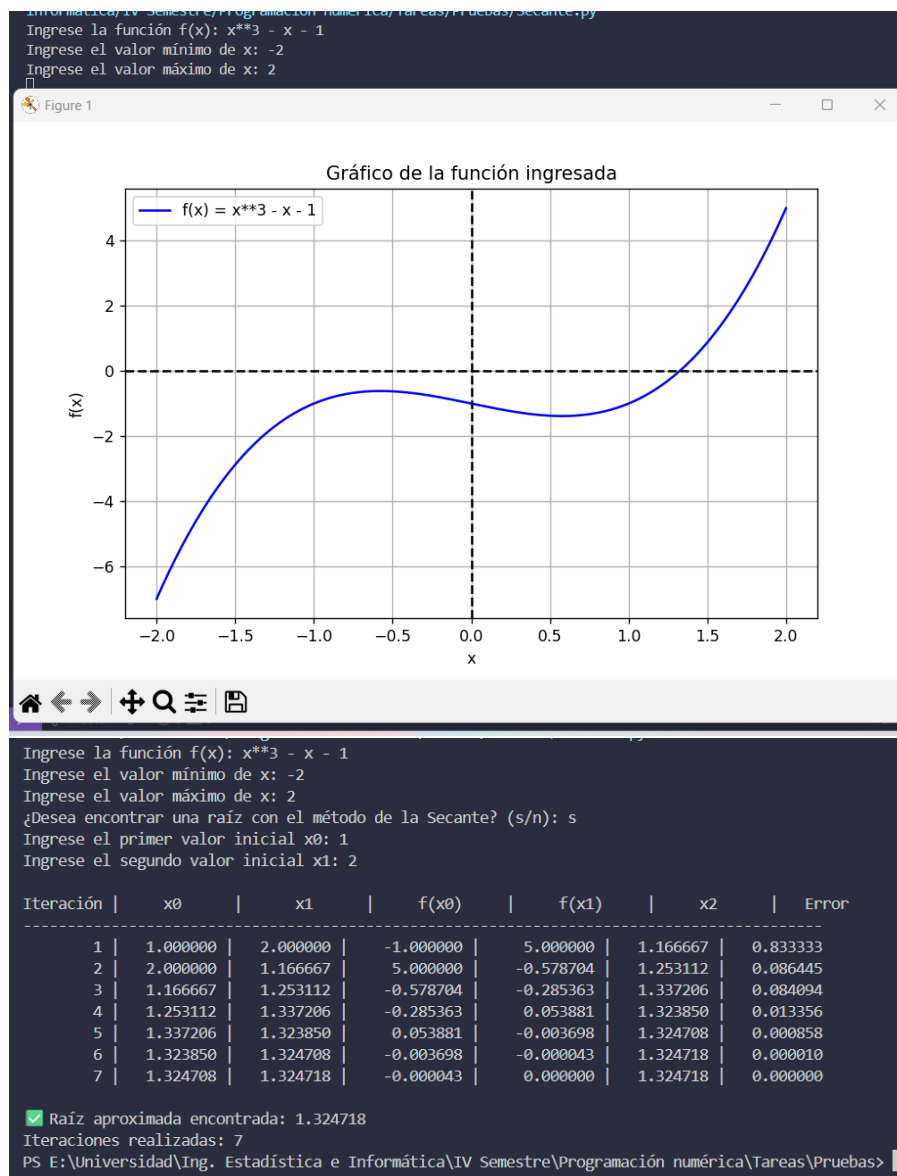


Figura 6.1: Método de la Secante en Python

7 Método de Punto Fijo

El método de punto fijo es uno de los procedimientos fundamentales para resolver ecuaciones no lineales de la forma

$$f(x) = 0.$$

A diferencia de otros métodos basados directamente en la función $f(x)$, el método de punto fijo se construye a partir de una transformación adecuada de la ecuación original en una función equivalente

$$x = g(x),$$

de modo que la raíz buscada es un punto fijo de la función g , es decir, un valor r tal que $g(r) = r$ (Burden & Faires, 2016).

La ecuación original puede expresarse de diversas maneras para obtener distintas funciones de iteración $g(x)$. La elección de esta función es crucial para garantizar la convergencia y determinar la eficiencia del método.

7.1 Definición y formulación básica

Dado un valor inicial x_0 , la aproximación iterativa se construye mediante el proceso repetitivo:

$$x_{n+1} = g(x_n), \quad n = 0, 1, 2, \dots$$

Si esta sucesión converge a un valor r , entonces

$$\lim_{n \rightarrow \infty} x_n = r \quad \Rightarrow \quad r = g(r),$$

lo que confirma que r es un punto fijo de g y, por ende, una solución de la ecuación original (Chapra & Canale, 2015).

7.2 Condición de convergencia

El comportamiento del método depende de las propiedades de la función $g(x)$. Uno de los resultados más importantes para garantizar convergencia es el siguiente:

Criterio de convergencia. Supongamos que la función g es continua en un intervalo I y que además su derivada $g'(x)$ es continua en el mismo intervalo. Si existe una constante $0 < k < 1$ tal que

$$|g'(x)| \leq k \quad \text{para todo } x \in I,$$

entonces el método de punto fijo es convergente para cualquier elección de $x_0 \in I$, y la sucesión generada converge a un único punto fijo en dicho intervalo (Burden & Faires, 2011).

Este criterio implica que la derivada de g controla la “fuerza” con la que la iteración se acerca al punto fijo. En términos prácticos:

$$|g'(r)| < 1 \quad \Rightarrow \quad \text{convergencia local,}$$

$$|g'(r)| > 1 \quad \Rightarrow \quad \text{divergencia.}$$

Esto hace evidente que la elección de la forma $g(x)$ no es arbitraria. Algunas transformaciones del tipo $x = g(x)$ producen convergencia, otras generan oscilaciones y otras divergencia total (Cheney & Kincaid, 2009).

7.3 Orden de convergencia

El método de punto fijo tiene convergencia lineal bajo las condiciones mencionadas. Esto significa que existe una constante C tal que

$$|x_{n+1} - r| \approx C |x_n - r|,$$

lo cual sugiere una velocidad de aproximación más lenta frente a métodos como Newton-Raphson, cuya convergencia es cuadrática (Atkinson, 2009). Sin embargo, su simplicidad lo convierte en una herramienta útil para la enseñanza y para establecer intervalos iniciales seguros para otros métodos más rápidos.

7.4 Errores y criterios de parada

El error en la iteración puede estimarse mediante:

$$|x_{n+1} - x_n| < \varepsilon \quad \text{o bien} \quad |g(x_n) - x_n| < \varepsilon,$$

donde ε es una tolerancia escogida. Otra forma consiste en fijar un número máximo de iteraciones si la función converge lentamente o presenta oscilaciones (Press et al., 2007).

7.5 Ventajas del método

El método de punto fijo presenta varias virtudes importantes:

- Su implementación es extremadamente sencilla.
- No requiere derivadas como Newton-Raphson.
- Es útil para analizar transformaciones funcionales equivalentes.
- Proporciona un marco general para estudiar muchos métodos iterativos.

- Permite visualizar la convergencia mediante diagramas cobweb (trampas de telaraña), muy utilizados en la enseñanza (Anton et al., 2012).

Además, su estructura simple hace posible analizar la estabilidad de los puntos fijos en modelos matemáticos aplicados, como ecuaciones diferenciales, sistemas dinámicos y procesos iterativos en ingeniería (Süli & Mayers, 2003).

7.6 Desventajas

A pesar de su utilidad conceptual y didáctica, el método presenta ciertas limitaciones:

- La convergencia depende fuertemente de la elección de $g(x)$.
- Incluso con una buena elección, la convergencia es lineal, por lo que puede ser lenta.
- Si $|g'(x)| \geq 1$, la iteración diverge.
- No siempre es sencillo encontrar una transformación apropiada.

En aplicaciones industriales o científicas, suele preferirse utilizar punto fijo como método preliminar para garantizar la existencia de una raíz antes de usar algoritmos más robustos o rápidos, como Newton-Raphson, la secante o métodos híbridos (Chapra & Canale, 2015).

7.7 Interpretación gráfica

Gráficamente, el método consiste en trazar la curva $y = g(x)$ junto con la recta $y = x$. El punto donde ambas se intersectan es el punto fijo. La iteración sigue los pasos:

1. Comenzar en $(x_0, g(x_0))$.
2. Proyectar verticalmente hacia la curva.
3. Proyectar horizontalmente hacia la recta $y = x$.

Este proceso se repite hasta alcanzar una convergencia visual. Este tipo de representación, conocida como diagrama cobweb, es muy útil para estudiar estabilidad y para reforzar la intuición del método (Riley et al., 2006).

7.8 Conclusión

El método de punto fijo constituye una herramienta esencial en el análisis numérico. Aunque no es el método más rápido ni el más eficiente, su simplicidad, su fundamento teórico sólido y su relación con otros métodos iterativos lo hacen indispensable en la formación matemática y en el estudio formal de la convergencia de algoritmos numéricos.

Su aplicación correcta depende de una adecuada elección de la función $g(x)$ y de un análisis cuidadoso de las condiciones de convergencia. Esto permite utilizarlo como punto de partida en el estudio y aplicación de métodos más avanzados para la resolución de ecuaciones no lineales (Burden & Faires, 2016).

7.9 Aplicación del método en Python

El objetivo fue desarrollar un programa en Python que permita ingresar una función $f(x)$ y su forma iterativa $g(x)$, graficar la función y luego aplicar el método de Punto Fijo si el usuario lo desea. El programa calcula sucesivamente los valores de $x_{n+1} = g(x_n)$ hasta alcanzar una precisión deseada y muestra los resultados obtenidos.

Entrada

- Una cadena de texto que representa la función original $f(x)$.
- La forma iterativa correspondiente $g(x)$.
- El intervalo para graficar.
- El valor inicial x_0 .

$$\begin{aligned}f(x) &= \cos(x) - x \\g(x) &= \cos(x)\end{aligned}$$

Salida

- Gráfica de la función $f(x)$.
- La raíz aproximada encontrada.
- Número de iteraciones realizadas hasta cumplir la tolerancia.

Restricciones

- El método requiere que $|g'(x)| < 1$ en el entorno de la raíz para garantizar la convergencia.
- La función debe ser continua y evaluable en todo el intervalo seleccionado.

Código

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # === 1. Ingreso de la función y su forma iterativa g(x) ===
5 print("=== MÉTODO DEL PUNTO FIJO ===")
6 print("Recuerde que f(x) = 0 se reescribe como x = g(x)")
7 print("Ejemplo: Si f(x) = cos(x) - x, entonces g(x) = cos(x)\n")
8
9 func_str = input("Ingrese la función original f(x): ") # Ejemplo: np.cos(x) - x
10 g_str = input("Ingrese la función iterativa g(x): ") # Ejemplo: np.cos(x)
11
12 # Definición de funciones
13 def f(x):
14     return eval(func_str, {"np": np, "x": x})
15
16 def g(x):
17     return eval(g_str, {"np": np, "x": x})
18
19 # === 2. Graficar la función f(x) ===
20 xmin = float(input("Ingrese el valor mínimo de x: "))
21 xmax = float(input("Ingrese el valor máximo de x: "))
22
23 x = np.linspace(xmin, xmax, 400)
24 y = f(x)
25
26 plt.figure(figsize=(8, 5))
27 plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
28 plt.axhline(0, color='black', linestyle='--')
29 plt.axvline(0, color='black', linestyle='--')
30 plt.title("Gráfico de la función ingresada")
31 plt.xlabel("x")
32 plt.ylabel("f(x)")
33 plt.legend()
34 plt.grid(True)
35 plt.show()
36
37 # === 3. Pregunta si desea aplicar el método de punto fijo ===
38 op = input("¿Desea aplicar el método de Punto Fijo? (s/n): ").lower()
39
40 if op == "s":
41     # === 4. Ingreso de valor inicial ===
42     x0 = float(input("Ingrese el valor inicial x0: "))
43
44     tol = 1e-6
45     max_iter = 100
46
47     print("\nIteración |      x0      |      g(x0)      |      f(x0)      |      Error")
48     print("-----")
49
50     for i in range(1, max_iter + 1):
51         x1 = g(x0)

```

```
52 error = abs(x1 - x0)
53
54 print(f"{i:9d} | {x0:10.6f} | {x1:12.6f} | {f(x1):12.6f} | {error:10.6f}")
55
56 if error < tol:
57     print(f"\n Raíz aproximada encontrada: {x1:.6f}")
58     print(f"Iteraciones realizadas: {i}")
59     break
60
61 x0 = x1
62 else:
63     print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")
64 else:
65     print("No se aplicó el método de Punto Fijo.")
```

Ejecución

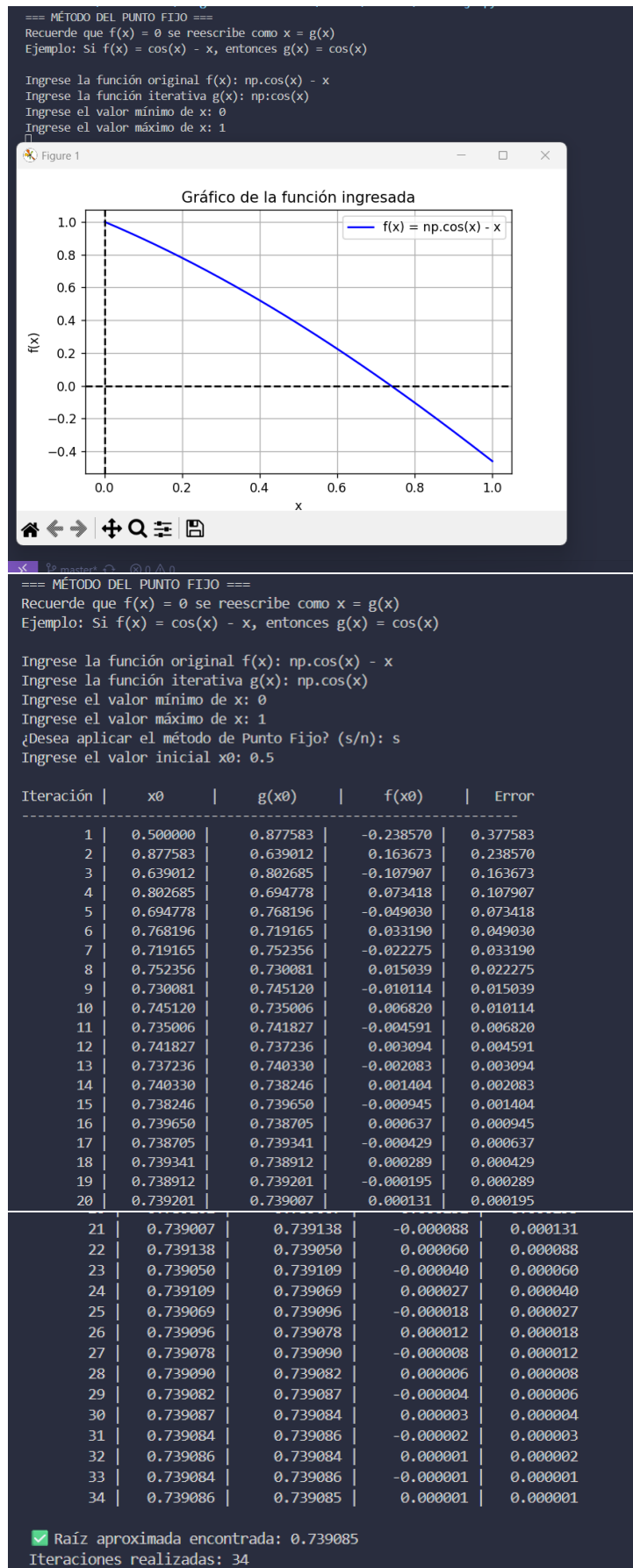


Figura 7.1: Método de Punto Fijo en Python

8 Método de Regula Falsi

El método de regula falsi, también conocido como método de la falsa posición, es una técnica clásica para la solución numérica de ecuaciones no lineales de la forma

$$f(x) = 0.$$

Este método combina aspectos del método de la bisección y del método de la secante, manteniendo siempre un intervalo que contiene la raíz mientras utiliza una aproximación lineal similar a la secante para generar nuevas aproximaciones. Su origen se remonta a prácticas matemáticas antiguas, pero su formalización moderna aparece descrita en textos fundamentales de análisis numérico (Burden & Faires, 2011; Chapra & Canale, 2015; Cheney & Kincaid, 2009).

8.1 Idea fundamental del método

Sea un intervalo cerrado $[a, b]$ tal que la función f sea continua en él y satisfaga la condición

$$f(a) f(b) < 0,$$

lo cual garantiza, mediante el Teorema del Valor Intermedio, la existencia de al menos una raíz en dicho intervalo. A diferencia del método de la bisección, que toma el punto medio como nueva aproximación, el método de regula falsi utiliza la recta secante que une los puntos $(a, f(a))$ y $(b, f(b))$ para obtener una aproximación más refinada a la raíz (Burden & Faires, 2016; Chapra & Canale, 2015).

El punto donde esta recta corta al eje x se calcula mediante la fórmula:

$$x_r = b - \frac{f(b)(a - b)}{f(a) - f(b)}.$$

Este x_r se denomina falsa posición y representa la aproximación actual de la raíz.

8.2 Actualización del intervalo

Una vez calculado x_r , se evalúa $f(x_r)$ y se conserva el intervalo que garantice el cambio de signo. Es decir:

$$\begin{cases} a = x_r & \text{si } f(a) f(x_r) < 0, \\ b = x_r & \text{si } f(a) f(x_r) > 0. \end{cases}$$

Este proceso asegura que la raíz permanece dentro del intervalo, proporcionando la robustez característica del método de bisección, pero con un paso adicional guiado por la pendiente de

la secante, lo que permite avanzar de manera más eficiente en muchas situaciones (Atkinson, 2009; Süli & Mayers, 2003).

8.3 Convergencia

El método de regula falsi es linealmente convergente, aunque en algunos casos su velocidad puede ser lenta. Esto ocurre especialmente cuando uno de los extremos del intervalo permanece fijo durante muchas iteraciones, lo cual puede suceder si la función presenta curvaturas pronunciadas o cambios de pendiente que provocan que la secante tienda a acercarse preferentemente a uno de los extremos (Burden & Faires, 2011; Press et al., 2007).

A pesar de ello, su principal ventaja es que **nunca pierde la raíz**, ya que siempre mantiene el intervalo acotado con un cambio de signo, lo que lo hace más seguro que métodos que no usan acotamiento, como el método de la secante clásico.

8.4 Criterios de parada

Los criterios más empleados para finalizar el proceso iterativo son:

- Error relativo entre iteraciones:

$$\left| \frac{x_r^{(k)} - x_r^{(k-1)}}{x_r^{(k)}} \right| < \varepsilon.$$

- Evaluación de la función:

$$|f(x_r)| < \varepsilon.$$

- Máximo número de iteraciones: Se detiene el proceso cuando se supera un número preestablecido de iteraciones.

Estos criterios permiten controlar la precisión deseada y asegurar que la aproximación obtenida es suficientemente cercana a la raíz (Burden & Faires, 2016; Chapra & Canale, 2015).

8.5 Ventajas y desventajas

Ventajas:

- Conserva la propiedad de acotamiento del método de la bisección.
- Puede converger más rápido que la bisección en muchas funciones.
- No requiere cálculo de derivadas.

Desventajas:

- La convergencia puede ser muy lenta si uno de los extremos del intervalo queda “atrapado”.
- Usualmente tiene una convergencia lineal, más lenta que la del método de la secante o Newton-Raphson.

8.6 Resumen conceptual

El método de regla falsi representa un compromiso entre estabilidad y velocidad: mantiene la seguridad del acotamiento, pero intenta mejorar la rapidez usando una aproximación lineal. Por ello, es especialmente útil cuando se requiere una garantía estricta de que la raíz no se perderá, pero se desea evitar la lentitud del método de la bisección tradicional (Burden & Faires, 2011; Chapra & Canale, 2015; Cheney & Kincaid, 2009).

8.7 Aplicación del método en Python

El objetivo fue desarrollar un programa en Python que permita al usuario ingresar una función $f(x)$, graficarla en un intervalo definido y aplicar el método de Regula Falsi si así lo desea. El programa realiza iteraciones hasta que se cumpla un criterio de tolerancia, mostrando en cada paso los valores de a , b , c , $f(c)$ y el error estimado.

Entrada

- Una cadena de texto que representa la función matemática $f(x)$.
- Los valores iniciales a y b que delimitan el intervalo de búsqueda.
- El intervalo para graficar la función.

$$f(x) = x^3 - x - 1$$

Salida

- Gráfica de la función $f(x)$ en el intervalo ingresado.
- La raíz aproximada encontrada.
- Número de iteraciones realizadas hasta cumplir la tolerancia.

Restricciones

- La función debe ser continua en el intervalo $[a, b]$.
- Debe cumplirse la condición de cambio de signo $f(a)f(b) < 0$.

Código

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # === 1. Ingreso de la función ===
5 print("=== MÉTODO DE REGULA FALSI (FALSA POSICIÓN) ===")
6 func_str = input("Ingrese la función f(x): ") # Ejemplo: x**3 - x - 1
7
8 # Definición de la función
9 def f(x):
10 return eval(func_str, {"np": np, "x": x})
11
12 # === 2. Graficar la función ===
13 xmin = float(input("Ingrese el valor mínimo de x: "))
14 xmax = float(input("Ingrese el valor máximo de x: "))
15
16 x = np.linspace(xmin, xmax, 400)
17 y = f(x)
18
19 plt.figure(figsize=(8, 5))
20 plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
21 plt.axhline(0, color='black', linestyle='--')
22 plt.axvline(0, color='black', linestyle='--')
23 plt.title("Gráfico de la función ingresada")
24 plt.xlabel("x")
25 plt.ylabel("f(x)")
26 plt.legend()
27 plt.grid(True)
28 plt.show()
29
30 # === 3. Pregunta si desea aplicar el método de Regula Falsi ===
31 op = input("¿Desea aplicar el método de Regula Falsi? (s/n): ").lower()
32
33 if op == "s":
34 # === 4. Ingreso de los extremos del intervalo ===
35 a = float(input("Ingrese el valor de a (extremo izquierdo): "))
36 b = float(input("Ingrese el valor de b (extremo derecho): "))
37
38 # Comprobación del cambio de signo
39 if f(a) * f(b) > 0:
40 print(" La función no cambia de signo en el intervalo. No se puede aplicar el
    ↪ método.")
41 else:
42 tol = 1e-6
43 max_iter = 100
44
45 print("\nIteración |      a      |      b      |      c      |      f(c)      |
    ↪ Error")
46 print("-----")
47
48 c_old = a
49 for i in range(1, max_iter + 1):

```

```
50 # Fórmula de Regula Falsi
51 c = b - (f(b) * (a - b)) / (f(a) - f(b))
52 error = abs(c - c_old)
53 c_old = c
54
55 print(f"{i:9d} | {a:10.6f} | {b:10.6f} | {c:10.6f} | {f(c):10.6f} | {error:10.6f}")
56
57 if abs(f(c)) < tol or error < tol:
58     print(f"\n Raíz aproximada encontrada: {c:.6f}")
59     print(f"Iteraciones realizadas: {i}")
60     break
61
62 # Actualización de intervalos
63 if f(a) * f(c) < 0:
64     b = c
65 else:
66     a = c
67 else:
68     print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")
69 else:
70     print("No se aplicó el método de Regula Falsi.")
```

Ejecución

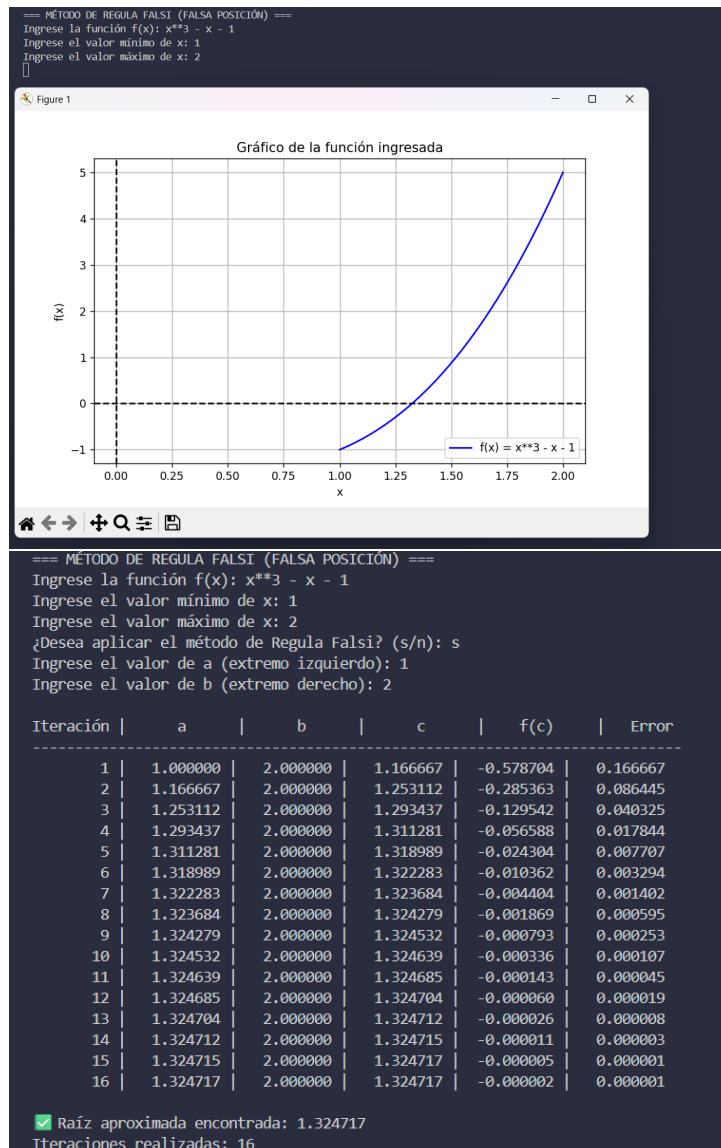


Figura 8.1: Método de Regula Falsi en Python

Parte II

Unidad II

9 Gradiente de una función

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

10 Diferenciación Numérica

11 Interpolación

Conclusiones

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetur adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Bibliografía

Bibliografía

- Anton, H., Bivens, I., & Davis, S. (2012). *Cálculo* (10.^a ed.). Wiley.
- Atkinson, K. (2009). *An Introduction to Numerical Analysis* (2nd). Wiley.
- Burden, R. L., & Faires, J. D. (2011). *Análisis Numérico* (9.^a ed.). Cengage Learning.
- Burden, R. L., & Faires, J. D. (2016). *Análisis numérico* (10.^a ed.). Cengage Learning.
- Chapra, S. C., & Canale, R. P. (2015). *Métodos numéricos para ingenieros* (7.^a ed.). McGraw-Hill Education.
- Cheney, W., & Kincaid, D. (2009). *Numerical Mathematics and Computing* (6th). Brooks/Cole.
- Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing* (3.^a ed.). Cambridge University Press.
- Riley, K. F., Hobson, M. P., & Bence, S. J. (2006). *Mathematical Methods for Physics and Engineering* (3.^a ed.). Cambridge University Press.
- Stewart, J., Redlin, L., & Watson, S. (2001). *Precálculo: Matemáticas para el Cálculo* (5.^a ed.). Thomson.
- Stewart, J., Redlin, L., & Watson, S. (2016). *Precálculo: Matemáticas para el Cálculo* (7.^a ed.). Cengage Learning.
- Süli, E., & Mayers, D. F. (2003). *An Introduction to Numerical Analysis*. Cambridge University Press.
- Sullivan, F. (2004). *Numerical Methods for Engineers*. Prentice Hall.