UNIVERSIDAD NACIONAL DEL ALTIPLANO

FACULTAD DE ING. ESTADÍSTICA E INFORMÁTICA

ESCUELA PROFESIONAL DE ING. ESTADÍSTICA E INFORMÁTICA

PROGRAMACIÓN NUMÉRICA

DOCENTE:

Dr. TORRES CRUZ FRED

ALUMNOS:

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

SECCIÓN:

"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

Índice de figuras

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 $f, g, h, ...$ 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)

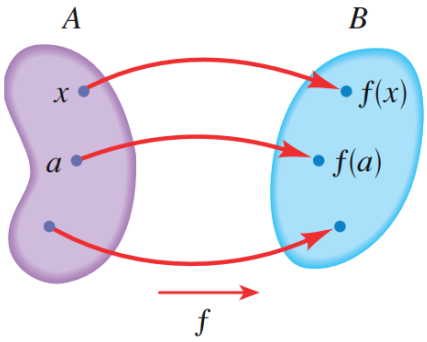


Figura 2.1: Diagrama de flechas de f

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 entres 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

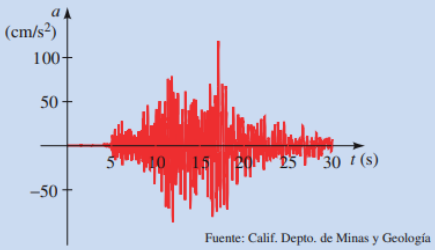
Por medio de una fórmula:

$$A(r) = \pi r^2$$

Área de un círculo

Visual

Por medio de una gráfica:



Aceleración vertical durante un terremoto

Numérica

Por medio de una tabla de valores:

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
⋮	⋮

Costo de enviar una carta por correo de primera clase

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)

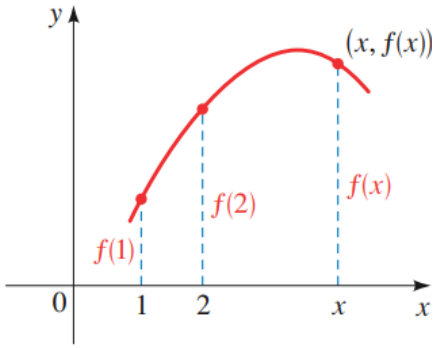


Figura 2.3: La altura de la gráfica arriba del punto x es el valor de $f(x)$

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)

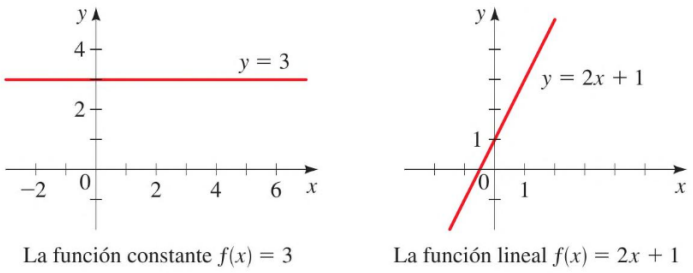


Figura 2.4: Funciones lineal y constante

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.

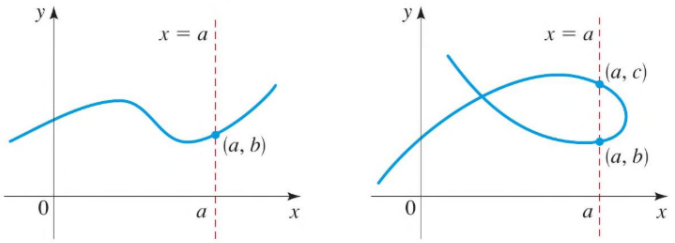


Figura 2.5: Prueba de la recta vertical

2.5 Aplicación

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

```python
import sympy as sp
import numpy as np
import matplotlib.pyplot as plt

# === Ingreso de la función por el usuario ===
expr_str = input("Ingrese la función en términos de x (ejemplo: sin(x), x**2 + 3*x - 5, exp(-x)*cos(x)): ")

# === Definición de variable simbólica ===
x = sp.Symbol('x')

# === Conversión del texto a expresión simbólica ===
try:
expr = sp.sympify(expr_str)
except sp.SympifyError:
print("Error: la función ingresada no es válida.")
exit()

# === Creación de función numérica evaluable ===
f = sp.lambdify(x, expr, modules=['numpy'])

# === Intervalo de graficación ===
x_vals = np.linspace(-10, 10, 400)
y_vals = f(x_vals)

# === Graficar ===
plt.figure(figsize=(7,5))
plt.plot(x_vals, y_vals, label=f"$f(x) = {sp.latex(expr)}$", color='navy')
plt.title("Gráfica de la función ingresada", fontsize=13)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.grid(True, linestyle='--', alpha=0.6)
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.legend()
```

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

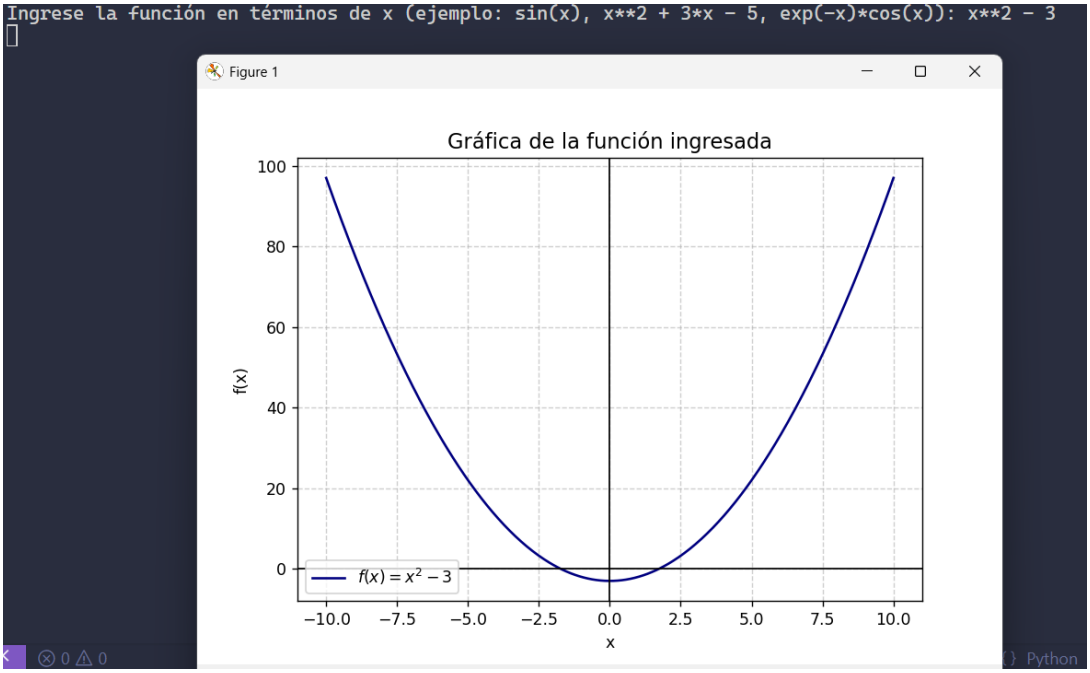


Figura 2.6: Graficando funciones con Python

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.
   - $x \geq 5$
2. El tiempo total no puede exceder a 15 horas.
   - $x + y \leq 15$

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

```
    # Función para preparar expresiones lineales
def preparar_expresion(expr: str) -> str:
expr = expr.replace(" ", "")           # quitar espacios
expr = expr.replace("^", "**")         # potencia
expr = expr.replace("-x", "-1*x")      # caso -x
expr = expr.replace("+x", "+1*x")      # caso +x
if expr.startswith("x"):               # si empieza con x
expr = "1*" + expr
expr = expr.replace("x", "*x")         # poner multiplicación
expr = expr.replace("**x", "*x")       # corregir si se duplicó
return expr

# Restricciones:
# 1) x = 5  (vertical)
# 2) x + y = 15  ->  y = -x + 15
func2 = preparar_expresion("-x+15")

# Rango de la gráfica
xmin, xmax = -5, 20
ymin, ymax = -5, 20

# Recorremos el plano
for y in range(ymax, ymin - 1, -1):
```

```
linea = ""
for x in range(xmin, xmax + 1):
# Recta 1: x = 5
cond1 = (x == 5)

# Recta 2: y = -x + 15
try:
y2 = eval(func2)
except:
y2 = None
cond2 = (y2 is not None and abs(y - y2) < 0.5)

# Región factible: x>=5, y>=0, x+y<=15
region = (x >= 5 and y >= 0 and x + y <= 15)

# Qué dibujar
if cond1 and cond2:
linea += "#"
elif cond1:
linea += "*"
elif cond2:
linea += "o"
elif x == 0 and y == 0:
linea += "+"
elif x == 0:
linea += "|"
elif y == 0:
linea += "-"
elif region:
linea += "."
else:
linea += " "
print(linea)

# Leyenda
print("\nLeyenda del gráfico:")
print("  * = x = 5")
print("  o = x + y = 15")
print("  # = Intersección")
print("  . = Región factible")
print("  | = Eje Y")
print("  - = Eje X")
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

```
import numpy as np
import matplotlib.pyplot as plt

# === 1. Ingreso de la función ===
func_str = input("Ingrese la función f(x): ")  # Ejemplo: x**3 - x - 1

# Definimos la función y su derivada (usando derivada numérica)
def f(x):
return eval(func_str)

def f_prime(x, h=1e-6):  # derivada numérica
return (f(x + h) - f(x - h)) / (2 * h)

# === 2. Graficar la función ===
xmin = float(input("Ingrese el valor mínimo de x: "))
xmax = float(input("Ingrese el valor máximo de x: "))

x = np.linspace(xmin, xmax, 400)
y = f(x)

plt.figure(figsize=(8, 5))
plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.title("Gráfico de la función ingresada")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()

# === 3. Pregunta si desea aplicar Newton-Raphson ===
op = input("¿Desea encontrar una raíz con el método de Newton-Raphson? (s/n): ").
    ↪ lower()

if op == "s":
# === 4. Ingreso de punto inicial ===
```

```
x0 = float(input("Basado en la gráfica, ingrese el valor inicial x1: "))

# Parámetros del método
tol = 1e-6
max_iter = 100

# Iteraciones
for i in range(1, max_iter + 1):
fx = f(x0)
fpx = f_prime(x0)

if fpx == 0:
print(f"La derivada es cero en x = {x0}. El método no puede continuar.")
break

x1 = x0 - fx / fpx

# Verificar convergencia
if abs(x1 - x0) < tol:
print(f"\n Raíz aproximada encontrada: {x1:.6f}")
print(f"Iteraciones realizadas: {i}")
break

x0 = x1
else:
print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")

else:
print("No se aplicó el método de Newton-Raphson.")
```

Ejecución

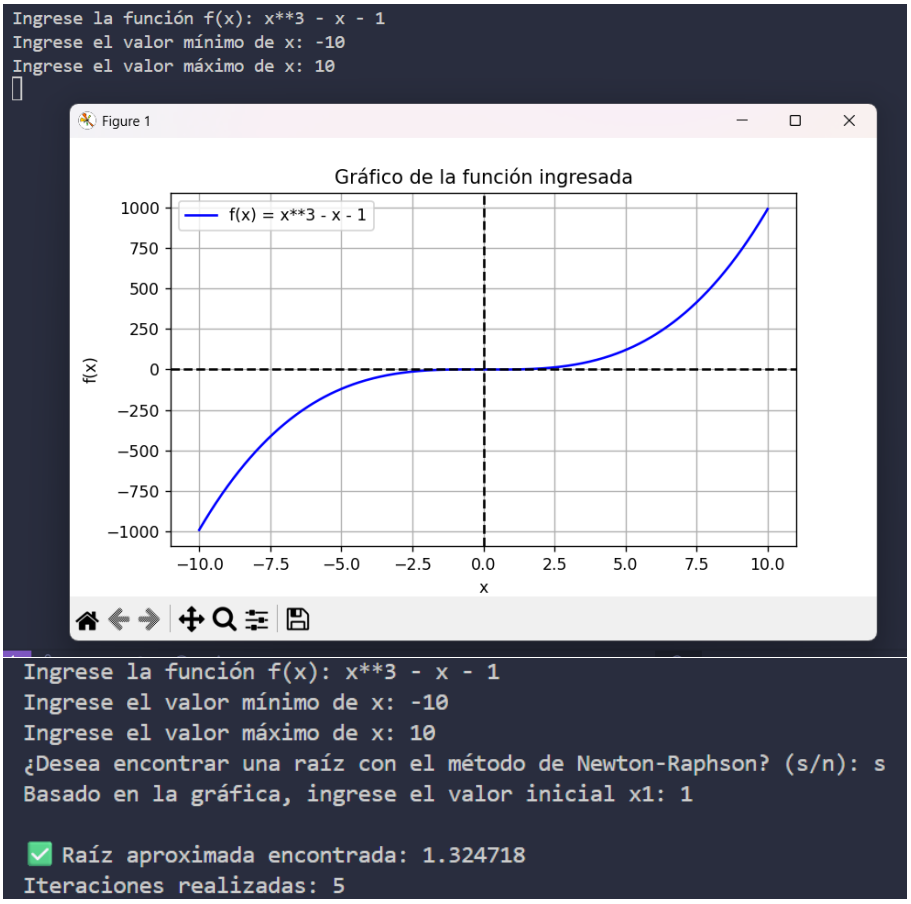


Figura 4.1: Método de Newthon-Rapshon en Python

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).

5.0.1 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).

5.0.2 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).

5.0.3 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).

5.0.4 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).

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

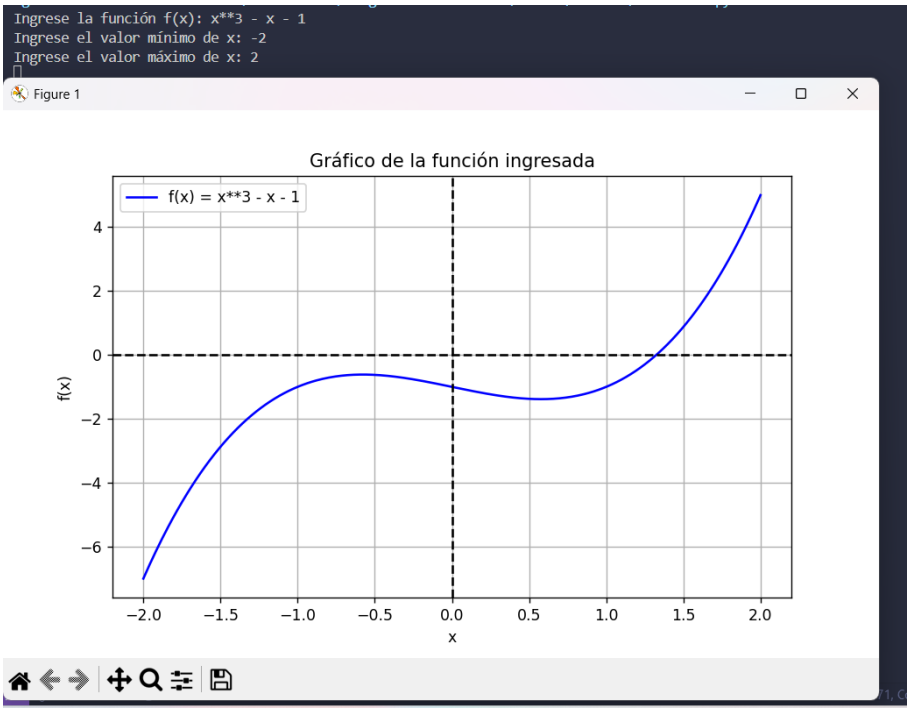
Código

```
import numpy as np
import matplotlib.pyplot as plt

# === 1. Ingreso de la función ===
func_str = input("Ingrese la función f(x): ")  # Ejemplo: x**3 - x - 1

# Definimos la función
def f(x):
return eval(func_str, {"np": np, "x": x})

# === 2. Graficar la función ===
xmin = float(input("Ingrese el valor mínimo de x: "))
xmax = float(input("Ingrese el valor máximo de x: "))

x = np.linspace(xmin, xmax, 400)
```

```
y = f(x)

plt.figure(figsize=(8, 5))
plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.title("Gráfico de la función ingresada")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()

# === 3. Pregunta si desea aplicar el método de Bisección ===
op = input("¿Desea encontrar una raíz con el método de Bisección? (s/n): ").lower()

if op == "s":
# === 4. Ingreso del intervalo inicial ===
a = float(input("Ingrese el extremo izquierdo del intervalo (a): "))
b = float(input("Ingrese el extremo derecho del intervalo (b): "))

# Verificación del cambio de signo
if f(a) * f(b) > 0:
print("\n No hay cambio de signo en el intervalo [a, b]. Intente con otro intervalo
    ↪ .")
else:
# Parámetros del método
tol = 1e-6
max_iter = 100

print("\nIteración |     a       |      b       |      c       |    f(c)     |
    ↪ Error")
print("-------------------------------------------------------------------")

for i in range(1, max_iter + 1):
c = (a + b) / 2
fc = f(c)
error = abs(b - a) / 2

print(f"{i:9d} | {a:10.6f} | {b:10.6f} | {c:10.6f} | {fc:10.6f} | {error:10.6f}")

if abs(fc) < tol or error < tol:
print(f"\n Raíz aproximada encontrada: {c:.6f}")
print(f"Iteraciones realizadas: {i}")
break

if f(a) * fc < 0:
b = c
else:
a = c
else:
print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")
```

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

Ejecución




```
Ingrese la función f(x): x**3 - x - 1
Ingrese el valor mínimo de x: -2
Ingrese el valor máximo de x: 2
¿Desea encontrar una raíz con el método de Bisección? (s/n): s
Ingrese el extremo izquierdo del intervalo (a): 1
Ingrese el extremo derecho del intervalo (b): 2

Iteración |     a       |     b       |     c       |    f(c)     |   Error
-------------------------------------------------------------------------------
        1 |   1.000000  |   2.000000  |   1.500000  |   0.875000  |   0.500000
        2 |   1.000000  |   1.500000  |   1.250000  |  -0.296875  |   0.250000
        3 |   1.250000  |   1.500000  |   1.375000  |   0.224609  |   0.125000
        4 |   1.250000  |   1.375000  |   1.312500  |  -0.051514  |   0.062500
        5 |   1.312500  |   1.375000  |   1.343750  |   0.082611  |   0.031250
        6 |   1.312500  |   1.343750  |   1.328125  |   0.014576  |   0.015625
        7 |   1.312500  |   1.328125  |   1.320312  |  -0.018711  |   0.007812
        8 |   1.320312  |   1.328125  |   1.324219  |  -0.002128  |   0.003906
        9 |   1.324219  |   1.328125  |   1.326172  |   0.006209  |   0.001953
       10 |   1.324219  |   1.326172  |   1.325195  |   0.002037  |   0.000977
       11 |   1.324219  |   1.325195  |   1.324707  |  -0.000047  |   0.000488
       12 |   1.324707  |   1.325195  |   1.324951  |   0.000995  |   0.000244
       13 |   1.324707  |   1.324951  |   1.324829  |   0.000474  |   0.000122
       14 |   1.324707  |   1.324829  |   1.324768  |   0.000214  |   0.000061
       15 |   1.324707  |   1.324768  |   1.324738  |   0.000084  |   0.000031
       16 |   1.324707  |   1.324738  |   1.324722  |   0.000018  |   0.000015
       17 |   1.324707  |   1.324722  |   1.324715  |  -0.000014  |   0.000008
       18 |   1.324715  |   1.324722  |   1.324718  |   0.000002  |   0.000004
       19 |   1.324715  |   1.324718  |   1.324717  |  -0.000006  |   0.000002
       20 |   1.324717  |   1.324718  |   1.324718  |  -0.000002  |   0.000001

✅ Raíz aproximada encontrada: 1.324718
Iteraciones realizadas: 20
```

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).

6.0.1 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).

6.0.2 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).

6.0.3 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).

6.0.4 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).

6.1 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

```
import numpy as np
import matplotlib.pyplot as plt

# === 1. Ingreso de la función ===
func_str = input("Ingrese la función f(x): ")  # Ejemplo: x**3 - x - 1

# Definimos la función
def f(x):
return eval(func_str, {"np": np, "x": x})

# === 2. Graficar la función ===
xmin = float(input("Ingrese el valor mínimo de x: "))
xmax = float(input("Ingrese el valor máximo de x: "))

x = np.linspace(xmin, xmax, 400)
y = f(x)

plt.figure(figsize=(8, 5))
plt.plot(x, y, label=f"f(x) = {func_str}", color='blue')
plt.axhline(0, color='black', linestyle='--')
plt.axvline(0, color='black', linestyle='--')
plt.title("Gráfico de la función ingresada")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.show()

# === 3. Pregunta si desea aplicar el método de la secante ===
```

```
op = input("¿Desea encontrar una raíz con el método de la Secante? (s/n): ").lower()

if op == "s":
# === 4. Ingreso de puntos iniciales ===
x0 = float(input("Ingrese el primer valor inicial x0: "))
x1 = float(input("Ingrese el segundo valor inicial x1: "))

# Parámetros del método
tol = 1e-6
max_iter = 100

print("\nIteración |     x0      |      x1      |     f(x0)     |     f(x1)     |     x2      |   Error")
print("-------------------------------------------------------------------------------")

for i in range(1, max_iter + 1):
f0 = f(x0)
f1 = f(x1)

if f1 - f0 == 0:
print(f"\n División por cero en la iteración {i}. El método no puede continuar.")
break

x2 = x1 - f1 * (x1 - x0) / (f1 - f0)
error = abs(x2 - x1)

print(f"{i:9d} | {x0:10.6f} | {x1:10.6f} | {f0:12.6f} | {f1:12.6f} | {x2:10.6f} | {error:10.6f}")

if error < tol:
print(f"\n Raíz aproximada encontrada: {x2:.6f}")
print(f"Iteraciones realizadas: {i}")
break

x0, x1 = x1, x2

else:
print("\n No se alcanzó la convergencia después de", max_iter, "iteraciones.")
else:
print("No se aplicó el método de la Secante.")
```

Ejecución

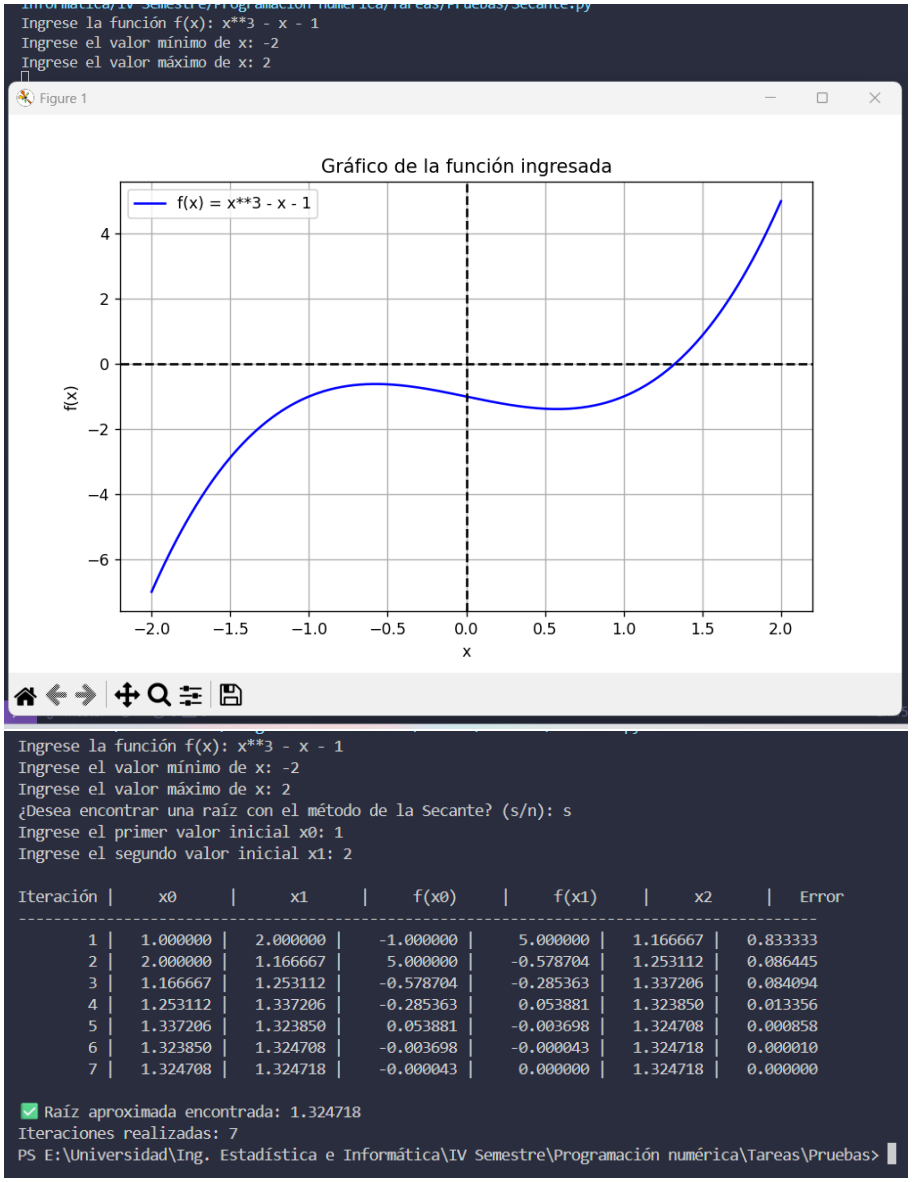


Figura 6.1: Método de la Secante en Python

7 Método de Punto Fijo

8 Método de Regula Falsi

Parte II

Unidad II

9 Gradiente de una función

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer 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, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer 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, consectetuer 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

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.

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.