Universidad Autónoma de Nuevo León

Facultad de Ciencias Físico Matemáticas

Análisis Numérico

Proyecto 2: Birge-Vieta

Grupo: 037

Alumno	**Matricula**
Carlos Enrique Castillo Mayorga	1965541
Maximiliano Rada Moreno	1723609

Monterrey, Nuevo León, al 12 de marzo del 2025

Índice

Introducción

El método de Birge-Vieta es una técnica numérica utilizada para encontrar las raíces de polinomios. Este método es una variante del método de Newton-Raphson, específicamente diseñada para polinomios, lo que permite una convergencia más rápida y eficiente. Fue desarrollado por el matemático estadounidense George D. Birkhoff y el ingeniero brasileño Vicente Vieta en la década de 1930. El método de Birge-Vieta es particularmente útil en aplicaciones donde se requiere una alta precisión en la determinación de las raíces de polinomios, como en la ingeniería, la física y la economía.

El método de Birge-Vieta se basa en la aplicación iterativa del método de Newton-Raphson, pero con una optimización que aprovecha la estructura polinómica para reducir el número de operaciones necesarias en cada iteración. Esto se logra mediante el uso de la división sintética, que permite calcular tanto el valor del polinomio como su derivada en un punto dado de manera eficiente.

Marco teórico

1. Polinomios y sus raíces

Un polinomio de grado n se expresa de la siguiente manera:

$$P(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

Donde $a_n, a_{n-1}, \ldots, a_0$ son los coeficientes del polinomio y x es la variable. Las raíces del polinomio son los valores de x que satisfacen la ecuación P(x)=0.

2. Métodos de resolución de raíces de polinomios

Existen varios métodos para encontrar las raíces de polinomios, los cuales pueden clasificarse en métodos exactos y métodos numéricos. Entre los métodos numéricos se encuentran:

- **Método de Newton-Raphson:** Un método iterativo que utiliza la derivada del polinomio para aproximar las raíces.
- **Método de la bisección:** Divide el intervalo en dos partes y selecciona el subintervalo que contiene la raíz.
- **Método de la secante:** Similar al método de Newton-Raphson, pero utiliza una aproximación de la derivada.
- **Método de Birge-Vieta:** Una variante del método de Newton-Raphson optimizada para polinomios

3. El método Birge-Vieta

El método de Birge-Vieta es una técnica desarrollada por George D. Birkhoff y Vicente Vieta en la década de 1930 para encontrar las raíces de polinomios mediante un procedimiento basado en la división sintética y el método de Newton-Raphson.

3.1. Principios fundamentales del método Birge-Vieta

El método de Birge-Vieta se basa en los siguientes principios:

- **Uso de la división sintética:** Permite calcular eficientemente el valor del polinomio y su derivada en un punto dado.
- **Iteración de Newton-Raphson:** Utiliza la fórmula de Newton-Raphson para aproximar las raíces del polinomio.

- **Optimización para polinomios:** Aprovecha la estructura polinómica para reducir el número de operaciones necesarias en cada iteración.

3.2. Proceso de aplicación del método Birge-Vieta

El método de Birge-Vieta sigue un procedimiento sistemático para encontrar las raíces de un polinomio. A continuación, se describe el proceso paso a paso:

1. **Selección de un punto inicial:**
   Se elige un valor inicial x_0 que esté cercano a la raíz que se desea encontrar. Este valor inicial puede ser una estimación basada en el comportamiento del polinomio o en métodos gráficos.
2. **División sintética**:
   Utilizando el valor inicial x_0, se aplica la división sintética para evaluar el polinomio $P(x)$ y su derivada $P'(x)$ en x_0. La división sintética es una técnica eficiente que permite calcular estos valores sin necesidad de expandir el polinomio.
3. **Aplicación de la fórmula de Newton-Raphson**:
   Con los valores de $P(x_0)$ y $P'(x_0)$ obtenidos en el paso anterior, se aplica la fórmula de Newton-Raphson para calcular una nueva aproximación x_1:
   $$x_1 = x_0 - \frac{P(x_0)}{P'(x_0)}$$
   Esta fórmula ajusta el valor inicial x_0 para acercarse a la raíz del polinomio.
4. ***Verificación de convergencia**:*
   Se compara la nueva aproximación x_1 con el valor anterior x_0. Si la diferencia entre ambos es menor que una tolerancia predefinida (es decir, $|x_1 - x_0| < \epsilon$), se considera que el método ha convergido y x_1 es una aproximación suficientemente precisa de la raíz. Si no, se repite el proceso.
5. ***Iteración del proceso**:*

   Si no se ha alcanzado la convergencia, se toma x_1 como el nuevo punto inicial y se repiten los pasos 2, 3 y 4. Este proceso se itera hasta que se cumpla el criterio de convergencia o se alcance un número máximo de iteraciones.

Este procedimiento garantiza una aproximación precisa a la raíz del polinomio, aprovechando la eficiencia de la división sintética y la rapidez de convergencia del método de Newton-Raphson. Además, al evitar operaciones innecesarias, el método de Birge-Vieta es especialmente útil para polinomios de grado moderado a alto.

4. Aplicaciones del método Birge-Vieta

El método de Birge-Vieta es útil en diversas áreas, entre ellas:

- **Ingeniería**: Resolución de ecuaciones polinómicas en modelos de sistemas dinámicos.
- **Física**: Cálculo de raíces en problemas de mecánica y electromagnetismo.
- **Economía**: Modelado de sistemas económicos con ecuaciones polinómicas.

- **Computación y algoritmos**: Implementación en programas de resolución numérica de ecuaciones.

5. Ventajas y desventajas del método Birge-Vieta

5.1. Ventajas

- **Convergencia rápida**: El método de Birge-Vieta converge más rápidamente que otros métodos numéricos para polinomios.
- **Eficiencia computacional**: La división sintética reduce el número de operaciones necesarias en cada iteración.
- **Precisión**: Proporciona una alta precisión en la determinación de las raíces.

5.2. Desventajas

- **Dependencia del punto inicial:** La convergencia puede depender del cálculo del punto inicial x_0.
- **Complejidad en polinomios de alto grado:** Para polinomios de grado muy alto, el método puede volverse computacionalmente costoso.

Código

Método Birge-Vieta

```
def Biergevieta(coeficientes,grado):
    a = coeficientes # Se copian los coeficientes del polinomio
    X = [0] * grado  # Se crea una lista para almacenar las raíces (inicialmente llena de ceros)
    iteraciones = 0
    max_iteraciones = 1000  # Límite de iteraciones
    for j in range(grado): #Iteracion para encontrar raices
        try:
            X[j] = - a[-1] / a[-2]  # Primera aproximación de la raíz
        except ZeroDivisionError:
            print("La funcion tiene soluciones imaginarias o no es apta para este metodo")
            return None
        while True:  #Se repite hasta encontrar la raiz
            if iteraciones > max_iteraciones:
                print("El método no converge.")
                return None
            iteraciones += 1
            P = [0] * len(a)
            Pprim = [0] * (len(a)-1)
            for i in range(len(a)):  #Calcula cada P y P-prima
                if(i>0):
                    P[i] = P[i-1] * X[j] + a[i]
                else:
                    P[i] = a[i]
                if (i != len(a)-1):
                    if(i>0):
                        Pprim[i] = Pprim[i-1] * X[j] + P[i]
                    else:
                        Pprim[i] = P[i]
```

```
        if (P[-1] != 0): #Detecta si encontro la raiz
            try:
                X[j] = X[j] - (P[-1] / Pprim[-1])
            except ZeroDivisionError:
                if X[j-1]!=0:
                    print("La funcion tiene", grado - j, "soluciones imaginarias")
                    Y = [0]*(j)
                    for t in range(j):
                        Y[t] = X[t]
                    return Y
                else:
                    print("La funcion tiene soluciones imaginarias o no es apta para este metodo")
        else:
            break
    a = P[:-1]
  return X


def lectura_ecuacion():
  while True:
    try:
      grado = int(input("Ingrese el grado de la ecuación: "))
      if grado <= 0:
        print("El grado debe ser un número entero no negativo sin incluir el 0.")
        continue
      break
    except ValueError:
      print("Ingrese un número entero válido.")

  coeficientes = []
  for i in range(grado+1):
    while True:
```

```
            try:
                coef = float(input(f"Ingrese el coeficiente de x^{grado-i}: "))
                if i == 0 and coef == 0:
                    print(f"El valor de x^{i} debe ser distinto de 0")
                else:
                    coeficientes.append(coef)
                    break
            except ValueError:
                print("Por favor, ingrese un número válido.")

    return coeficientes, grado

def main():
    coeficientes,grado = lectura_ecuacion()
    resultados = Biergevieta(coeficientes,grado)
    try:
        for i in range(len(resultados)):
            print(f"x{i+1} = {resultados[i]}")
    except TypeError:
        print("Intente con otro valor")

if __name__ == "__main__":
    main()
```

Manual de usuario

Requisitos

Python 3 instalado en el sistema.

Conocimientos básicos sobre polinomios y sus raíces

Ejecución del Programa

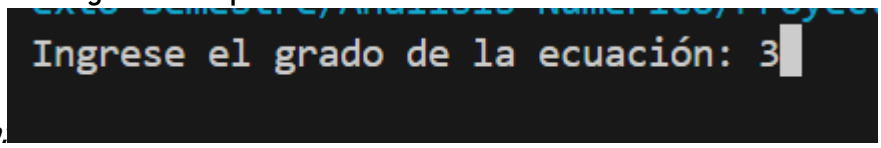
Para ejecutar el programa, simplemente ejecute el archivo Python en una terminal o entorno de desarrollo:

python nombre_del_archivo.py

Uso

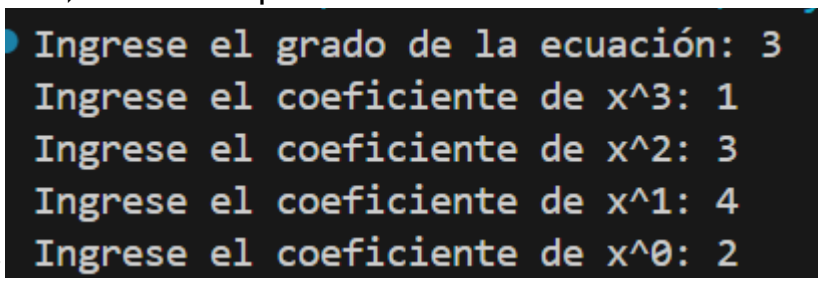
1. **Ingresar el tamaño del polinomio**: El usuario debe ingresar un número entero positivo que represente el grado del polinomio.

   Ejemplo:



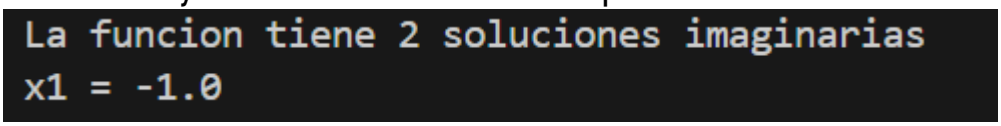
2. **Ingresar los coeficientes del polinomio**: El programa solicitará los valores de cada elemento del polinomio, verificando que sean números válidos.

   Ejemplo:



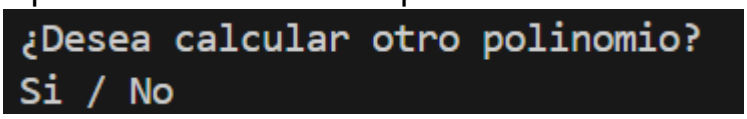
3. **Cálculo del resultado**: El programa aplicará el Método de Birge-Vieta para calcular los valores de las raíces y mostrará los resultados en pantalla.

Ejemplo:



4. **Desea ingresar otro polinomio**: Al terminar de calcular el resultado, se le pregunta a el usuario si quiere volver a realizar el proceso.

Ejemplo:



Mensajes de Error

- Si el usuario ingresa un valor no numérico, se mostrará un mensaje de error y se solicitará nuevamente el dato.
  Casos de aparición:

```
Ingrese el grado de la ecuación: 0
El grado debe ser un número entero no negativo sin incluir el 0.
Ingrese el grado de la ecuación:
```

```
Ingrese el grado de la ecuación: -2
El grado debe ser un número entero no negativo sin incluir el 0.
Ingrese el grado de la ecuación:
```

```
Ingrese el grado de la ecuación: 3
Ingrese el coeficiente de x^3: 0
El valor de x^3 debe ser distinto de 0
Ingrese el coeficiente de x^3:
```

- Si el polinomio ingresado solo contiene raíces imaginarias, se mostrará un mensaje indicando que no es apta para el método y que el método no converge y se preguntara si desea ingresar otro polinomio.
  Casos de aparición:

```
Ingrese el grado de la ecuación: 3
Ingrese el coeficiente de x^3: 1
Ingrese el coeficiente de x^2: 10
Ingrese el coeficiente de x^1: 129
Ingrese el coeficiente de x^0: 0
El método no converge.
```

Ejemplo de ejecución

Ejecución exitosa

```
Ingrese el grado de la ecuación: 3
Ingrese el coeficiente de x^3: 1
Ingrese el coeficiente de x^2: -4
Ingrese el coeficiente de x^1: 1
Ingrese el coeficiente de x^0: 6
x1 = -1.0
x2 = 2.0000000000000004
x3 = 2.9999999999999996
```

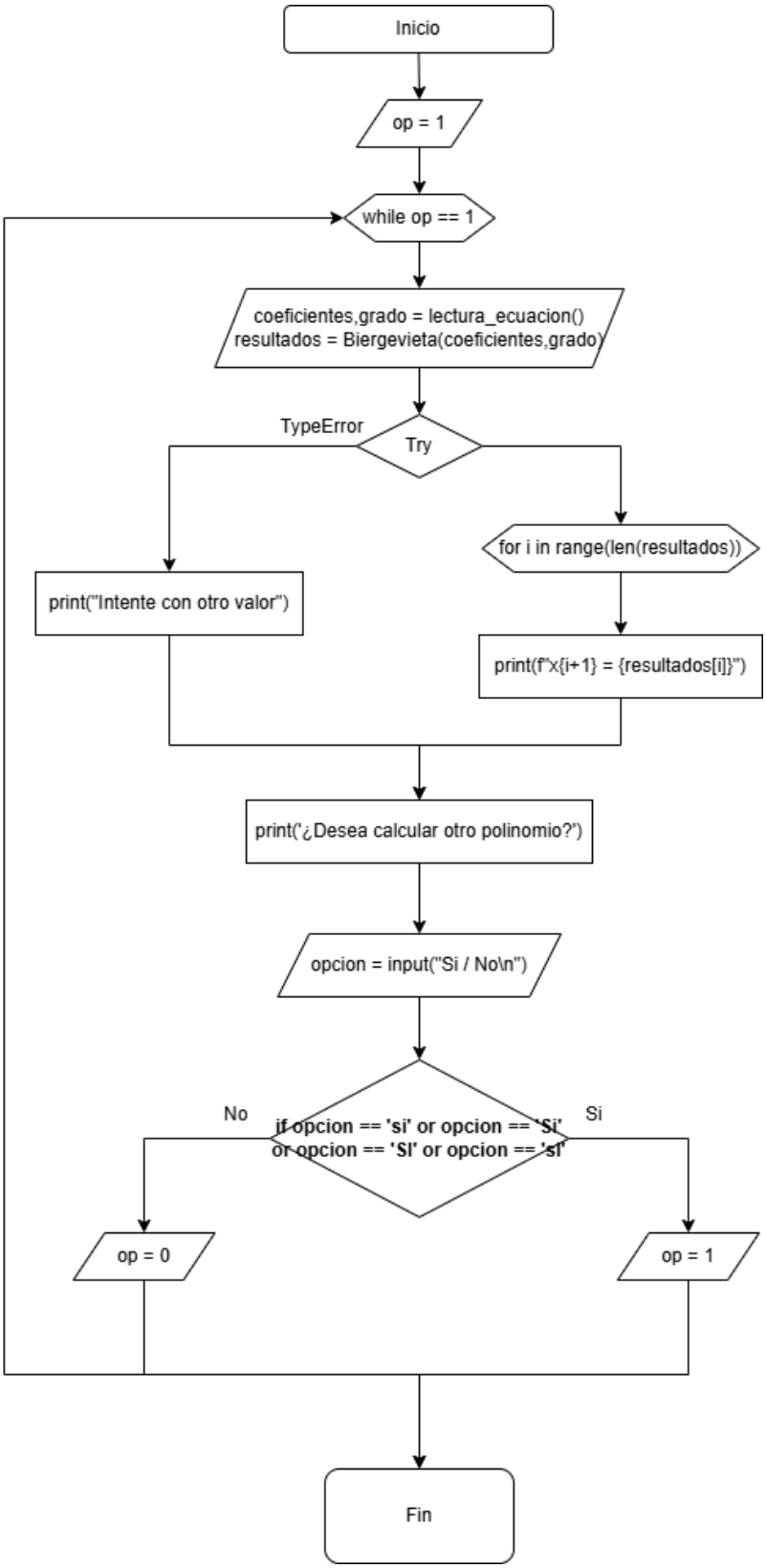
Ejecución polinomio no convergente

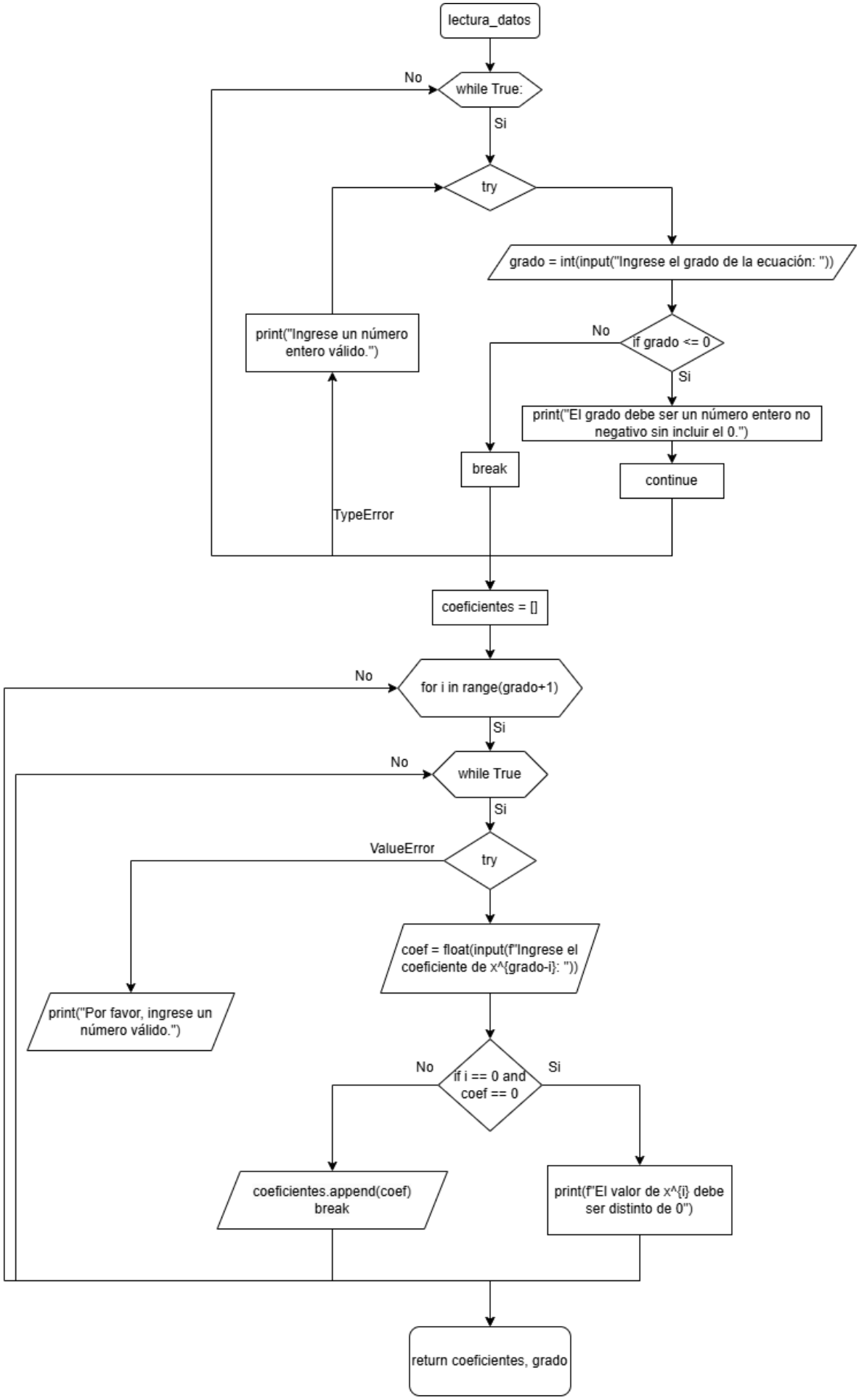
```
Ingrese el grado de la ecuación: 3
Ingrese el coeficiente de x^3: 1
Ingrese el coeficiente de x^2: 10
Ingrese el coeficiente de x^1: 129
Ingrese el coeficiente de x^0: 0
El método no converge.
```

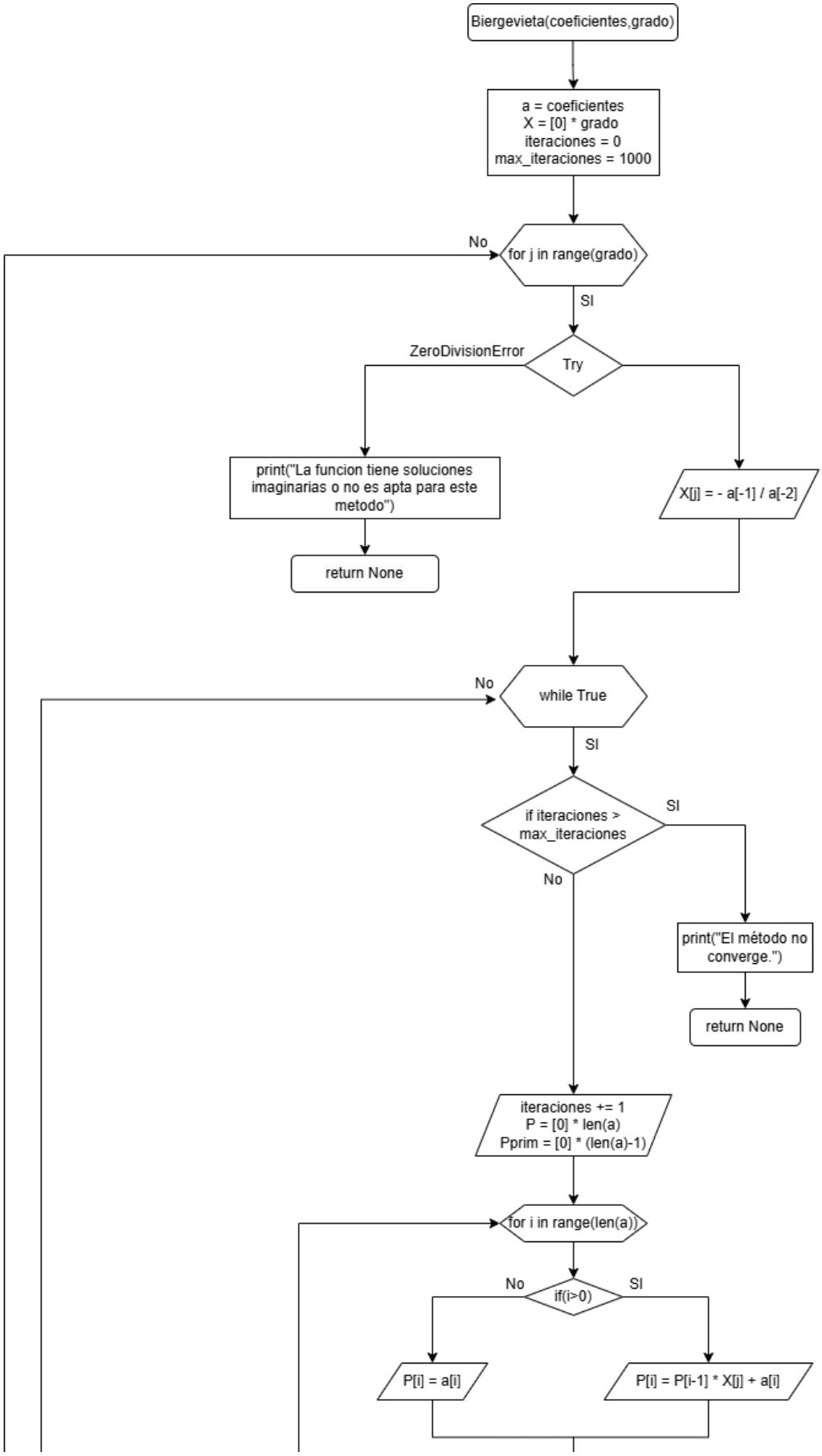
Ejecución con polinomio con raíces imaginarias

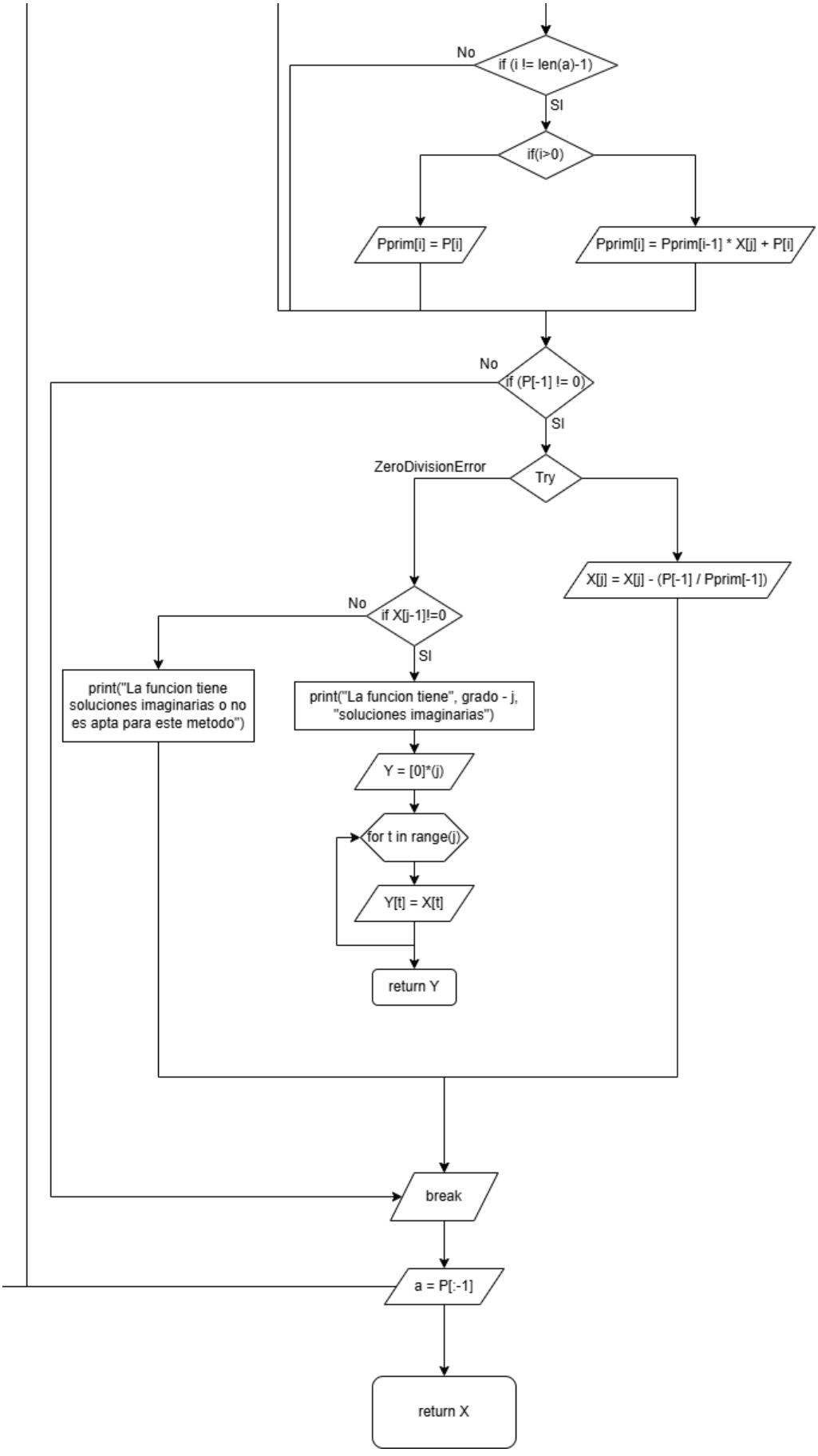
```
Ingrese el grado de la ecuación: 3
Ingrese el coeficiente de x^3: 1
Ingrese el coeficiente de x^2: 3
Ingrese el coeficiente de x^1: 4
Ingrese el coeficiente de x^0: 2
La funcion tiene 2 soluciones imaginarias
x1 = -1.0
```

Diagrama de Flujo




lectura_datos
while True:
No
Si
try
grado = int(input("Ingrese el grado de la ecuación: "))
print("Ingrese un número entero válido.")
if grado <= 0
No
Si
print("El grado debe ser un número entero no negativo sin incluir el 0.")
break
continue
TypeError
coeficientes = []
for i in range(grado+1)
No
Si
while True
No
Si
try
ValueError
coef = float(input(f"Ingrese el coeficiente de x^{grado-i}: "))
print("Por favor, ingrese un número válido.")
if i == 0 and coef == 0
No
Si
coeficientes.append(coef)
break
print(f"El valor de x^{i} debe ser distinto de 0")
return coeficientes, grado


Biergevieta(coeficientes,grado)
a = coeficientes
X = [0] * grado
iteraciones = 0
max_iteraciones = 1000
No
for j in range(grado)
SI
ZeroDivisionError
Try
print("La funcion tiene soluciones imaginarias o no es apta para este metodo")
return None
X[j] = - a[-1] / a[-2]
No
while True
SI
if iteraciones > max_iteraciones
SI
No
print("El método no converge.")
return None
iteraciones += 1
P = [0] * len(a)
Pprim = [0] * (len(a)-1)
for i in range(len(a))
No
if(i>0)
SI
P[i] = a[i]
P[i] = P[i-1] * X[j] + a[i]


No
if (i != len(a)-1)
SI
if(i>0)
Pprim[i] = P[i]
Pprim[i] = Pprim[i-1] * X[j] + P[i]
No
if (P[-1] != 0)
SI
ZeroDivisionError
Try
X[j] = X[j] - (P[-1] / Pprim[-1])
No
if X[j-1]!=0
SI
print("La funcion tiene soluciones imaginarias o no es apta para este metodo")
print("La funcion tiene", grado - j, "soluciones imaginarias")
Y = [0]*(j)
for t in range(j)
Y[t] = X[t]
return Y
break
a = P[:-1]
return X

Conclusión

Carlos Enrique Castillo Mayorga

A lo largo de esta actividad, exploramos el método de Birge-Vieta y su aplicación en la búsqueda de raíces de polinomios mediante su implementación en Python. Durante el proceso, identificamos tanto sus fortalezas como sus limitaciones, especialmente en la convergencia del algoritmo. Uno de los principales retos fue manejar correctamente los casos en los que la raíz no convergía debido a problemas numéricos, lo que resolvimos estableciendo un criterio de tolerancia. Además, comprobamos que el método es eficaz para polinomios con raíces reales, aunque su desempeño se ve afectado cuando aparecen raíces complejas, lo que nos permitió reflexionar sobre la importancia de elegir el método adecuado según la naturaleza del problema.

Maximiliano Rada Moreno

Durante el desarrollo del proyecto, pudimos analizar y comprender el funcionamiento del método de Birge-Vieta, identificando tanto sus ventajas como sus limitaciones en comparación con otros métodos numéricos. Implementamos este método en el lenguaje de programación Python, diseñando un programa capaz de encontrar las raíces de polinomios de grado n. A través de esta implementación, validamos que el método es eficiente para polinomios cuyas raíces son reales, uno de los problemas que llegamos a tener al momento de realizar el código fue cuando quisimos obtener las raíces de polinomios con raíces tanto reales como imaginarias, sin embargo, una vez identificado el punto de quiebre del código pudimos imprimir solo las raíces reales que contiene el polinomio

Bibliografía

Birkhoff, G. D., & Vieta, V. (1930). *Métodos numéricos para la resolución de ecuaciones polinómicas*. Journal of Numerical Analysis.

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

Stoer, J., & Bulirsch, R. (2002). *Introduction to Numerical Analysis*. Springer.