Trabajo Práctico Nº 3

Raíces de Polinomios

|  |  |  |
| --- | --- | --- |
| **Integrantes** | | |
| **Apellido y Nombre** | **LU** | **Carrera** |
| Alarcón, Pablo Guillermo | 216094 – 217312 | LAS – TUP |
| Mamaní, Cristian Alfredo | 217333 | TUP |
| Venticola, Leandro | 218521 | TUP |
| Saldaño, Sebastián | 215166 | LAS |
| Tapia, Rubén | 213701 - 218120 | LAS - TUP |

**Índice de Contenidos**

[1 Enunciado del Problema 2](#_Toc427003425)

[2 Diseño Modular 2](#_Toc427003426)

[3 Casos de Pruebas 2](#_Toc427003427)

[3.1 Casos de Pruebas Unitarias 2](#_Toc427003428)

[3.2 Casos de Pruebas de Integración 2](#_Toc427003429)

[4 Pruebas 2](#_Toc427003430)

[4.1 Pruebas Unitarias 2](#_Toc427003431)

[4.2 Pruebas de Integración 2](#_Toc427003432)

[5 Conclusión 3](#_Toc427003433)



# 1 Enunciado del Problema

* Realizar un programa que incorpore procedimientos para:
  1. Calcular el valor de un polinomio para un determinado punto.
  2. Dividir un polinomio de grado n por otro de grado 1.
  3. Dividir un polinomio de grado n por otro de la forma
  4. Determine las posibles raíces enteras.
  5. Determine las posibles raíces racionales.
  6. Determine las cotas de las raíces positivas y negativas por distintos métodos.
  7. Encontrar raíces reales por el método de Newton para polinomios.
* Implemente el Método de Bairstow.
* Diseñe casos de pruebas y realice un informe sobre el funcionamiento de los programas

# 2 Diseño Modular

El Programa consta principalmente de 3 clases:

**1) Cls\_Polin:**

|  |  |  |
| --- | --- | --- |
| **Atributo o Propiedad** | **Tipo** | **Descripción** |
| +Coef | Cls\_vector | Almacena los coeficientes del polinomio |
| +Raices | Cls\_vector | Guarda las raíces reales e imaginarias obtenidas por algún método |
| +Masc | Integer | Guarda la cantidad de decimales a mostrar |
| +A0 | Boolean | Determina si debe mostrar desde el coeficiente con menor grado o con el mayor |

|  |  |  |
| --- | --- | --- |
| **Metodo** | **Resultado** | **Descripción** |
| +Crear(Grado=3) | Constructor | Crea el polinomio con Grado dado, por defecto grado=3 |
| +Grado() | Integer | Retorna el grado del polinomio |
| +Clon() | Cls\_Polin | Retorna un objeto de Cls\_Polin con el mismo estado |
| +Invertir\_Coef | Vacío | Invierte los coeficientes |
| +Coef\_To\_string() | String | Retorna el string representativo del polinomio |
| +evaluar(x) | extended | Retorna el valor del polinomio en el punto “x” dado |
| +derivada() | Cls\_Polin | Devuelve la primer derivada en un objeto de Cls\_Polin |
| +ruffini(divisor,cociente,resto) | Boolean | Retorna “true” si es posible hacer esto. Divisor: polinomio lineal; guarda en cociente y resto los valores obtenidos mediante ruffini |
| +hornerCuadratico (divisor, cociente, resto) | Boolean | Retorna “true” si es posible hacer esto. Divisor: polinomio cuadratico; guarda en cociente y resto los valores obtenidos mediante hornerCuadratico |
| +newton(x0) | extended | Retorna la raíz del polinomio obtenida mediante newton para polinomios con el valor inicial “x0”. Si el algoritmo se dispara retorna la constante INFINITO = 9.999.999 |
| +posiblesRaicesRacionales() | Cls\_vector | Retorna un vector con las posibles raíces racionales |
| +posiblesRaicesEnteras() | Cls\_Vector | Retorna un vector con las posibles raíces enteras |
| +Lagrange() | Cls\_Vector | Retorna un vector con 4 valores que son las cotas en formato [cotaInfNeg, cotaSupNeg, cotaInfPos, cotaSupPos]. Si no tiene cotas del lado positivo o negativo retorna ceros |
| +Laguerre() | Cls\_Vector | Retorna un vector con 4 valores que son las cotas en formato [cotaInfNeg, cotaSupNeg, cotaInfPos, cotaSupPos]. Si no tiene cotas del lado positivo o negativo retorna ceros |
| +cotasNewton() | Cls\_Vector | Retorna un vector con 4 valores que son las cotas en formato [cotaInfNeg, cotaSupNeg, cotaInfPos, cotaSupPos]. Si no tiene cotas del lado positivo o negativo retorna ceros |
| +Sturm(cotas) | Cls\_Vector | Devuelve nuevas cotas a partir de las cotas de Cls\_vector enviadas en el parámetro |
| +Bairstow(error, r0,s0) | Vacío | Calcula las raíces reales e imaginarias y las guarda en el atributo raíces de Cls\_Matriz |

**2) Cls\_Vector**

|  |  |  |
| --- | --- | --- |
| **Atributo o Propiedad** | **Tipo** | **Descripción** |
| -xCell | Array of extended | Guarda los valores en un vector dinamico |
| -ordenN | Integer | Guarda el orden del vector = cantidad -1 |
| +cells[i] | Property | Escritura y lectura del elemento en la posición “i” de xCell |
| +N | Property | Escritura y lectura de ordenN |

|  |  |  |
| --- | --- | --- |
| **Métodos** | **Retorno** | **Descripción** |
| +Crear(Grado=) |  |  |
| +norma() | extended | Retorna la norma euclidea del vector |
| +xEscalar(k) | Vacío | Multiplica los elementos del vector por un escalar “k” |
| +intercambiar(i,j) | Vacío | Intercambia los elementos de las posiciones “i” y “j” |
| +eliminarX(i) | Vacío | Elimina el elemento en la posición “i” |
| +insertarX(x,i) | Vacío | Inserta el elemento “x” en la posición “i” |
| +subVector(pos,cant) | Cls\_Vector | Retorna un objeto de la clase Vector de cantidad “cant” desde la posición “pos” |
| +suma(VectorA, VectorB) | Vacío | Suma elemento a elemento los vectores A y B y los guarda en sí mismo |
| +suma(vectorB) | Vacío | Suma elemento a elemento con el vectorB |
| +ToString() | String | Retorna el string representante del vector |

**Cls\_Matriz**

|  |  |  |
| --- | --- | --- |
| **Atributo o Propiedad** | **Tipo** | **Descripción** |
| -xCell | Array of array of extended | Guarda los valores de la matriz |
| -NF - | Integer | Cantidad de filas |
| -NC | Integer | Cantidad de columnas |
| +Cells[i,j] | Property | Lectura y Escritura del elemento “i,j” de xCell |
| +NumF | Property | Lectura y escritura de NF |
| +NumC | Property | Lectura y escritura de NC |

|  |  |  |
| --- | --- | --- |
| **Métodos** | **Retorno** | **Descripción** |
| +Crear(filas,columnas) | Constructor | Construye una matriz con cantidad de filas y columnas dadas |
| +Limpia(k = 0) | Vacío | Llena la matriz con el escalar dado, por defecto k=0 |
| +eliminaFila(i) | Vacío | Elimina la fila i-ésima |
| +eliminaColumna(i) | Vacío | Elimina la columna i-ésima |
| +insertarFila(vector, fila) | Vacío | Inserta un vector en la fila indicada |
| +insertarColumna( vector, columna) | Vacío | Inserta un vector en la columna dada |
| +sonIguales(MatrizA) | Boolean | Compara elemento a elemento con la matrizA |
| +esCuadrada() | Boolean | Retorna true si la matriz es cuadrada |
| +det() | Extended | Retorna el determinante de la matriz |
| +noEsSingular() | Boolean | Retorna true si la matriz tiene inversa |
| +esSimetrica() | Boolean | Retorna True si la matriz es simétrica |
| +esAntiSimetrica() | Boolean | Retorna True si es anti Simétrica |
| +esEstrictamenteDiagonalmenteDominante | Boolean | Retorna True si la matriz es estrictamente diagonalmente dominante |
| +xEscalar(k) | Vacío | Multiplica los elementos de la matriz por el escalar “k” |
| +fila\_xEscalar(fil,k) | Vacío | Multiplica la fila “fil” por el escalar “k” |
| +columna\_xEscalar( col,k) | Vacío | Multiplica la columna “col” por el escalar “k” |

# 3 Casos de Pruebas

Abreviaturas:

DE: Datos de Entrada.

DS: Datos de Salida.

## 3.1 Casos de Pruebas Unitarias

Módulo de PosiblesRaicesEnteras:

Caso 1:

DE: P(x)=-x^2+0x+1

DS: 1,-1

Caso 2:

DE: P(x)=x^2+1+0

DS: Ninguna Raiz

Caso 3:

DE: P(x)=x^2+1+2.5

DS: 1,-1,2,-2

Caso 4:

DE: P(x)=x^2+x-3

DS: 1,-1,3,-3

Módulo de Posibles Raices Racionales:

Caso 1:

DE: P(x)=-x^2+x+2

DS: Ninguna Raiz

Caso 2:

DE: P(x)=-x^2+x+0

DS: Ninguna Raiz

Caso 3:

DE: P(x)=x^2+x+3

DS: Ninguna Raiz

Caso 4:

DE: P(x)=3x^2+x+1

DS: ⅓ , -⅓

Caso 5:

DE: P(x)=2x^2+x-1

DS: ½ , -½

Módulo de Lagrangue:

Caso 1:

DE: P(x)=4x^4+0x^3-5x^2+0x+1

DS:

Cota Superior Positiva: 2.118033989

Cota Inferior Positiva: 0.3090169944

Cota Superior Negativa: -2.118033989

Cota Inferior Negativa: -0.3090169944

Caso 2:

DE: P(x)=4x^4-8x^3+3x^2+2x-1

DS:

Cota Superior Positiva: 3

Cota Inferior Positiva: 0.2

Cota Superior Negativa: -0.3

Cota Inferior Negativa: -1.8

Módulo de Laguerre:

Caso 1:

DE: P(x)=4x^4+0x^3-5x^2+0x+1

DS: Cota Superior Positiva: 1.5

Cota Inferior Positiva: 0.4

Cota Superior Negativa: -0.4

Cota Inferior Negativa: -1.5

Caso 2:

DE: P(x)=4x^4-8x^3+3x^2+2x-1

DS:

Cota Superior Positiva: 2

Cota Inferior Positiva: 0.3

Cota Superior Negativa: -0.3

Cota Inferior Negativa: -0.6

Módulo de división de un polinomio por uno de grado 1.

Caso 1

DE: Dividendo= ; Divisor= ;

DS: Cociente= ; Resto= -15/18;

Caso 2

DE: Dividendo= ; Divisor= ;

DS: Cociente= 0,5; Resto= 1;

Caso 3

DE: Dividendo= 3; Cociente= ;

DS: El algoritmo no puede realizar esa operación.

Módulo de división por un polinomio de la forma

Caso 1

DE: Dividendo= ; Divisor=

DS: Cociente= ; Resto=

Caso 2

DE: Dividendo= ; Divisor=

DS: Cociente= ; Resto=

Caso 3

DE: Dividendo= ; Divisor=

DS: El algoritmo no puede realizar la operación.

Módulo de obtención de cotas por Newton

Caso 1

DE: Polinomio= ; Dato Extra: Raices= {-1,1,3};

DS: Intervalo Negativo= [-2,0]; Intervalo Positivo= [0,4];

Caso 2

DE: Polinomio= ; Dato Extra: Sin raíces Reales;

DS: Sin cotas;

Caso 3

DE: Polinomio= ; Dato Extra: Sin raíces Negativas;

DS: Sin cotas Negativas; Intervalo Positivo [4; 5];

Obtención de Raíz por Newton

Caso 1

DE: P(x) =

DS: “no tiene raíces reales”

Caso 2

DE: P(x) =

DS: raíz = -1

Módulo de Sturm:

Caso 1:

DE: P(x)=4x^4+0x^3-5x^2+0x+1

DS: Intervalo de Raíces Negativas: [-0.5,-0.49] ; [-1.01,-1]

Intervalo de Raíces Positivas: [0.49,0.5] ; [1,1.01]

Bairstow

Caso de prueba 1

P(x)=x⁴+5x³+15x²+5x-26

Error=0,001 r0=-1,01 r0=2,01

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 5 | 15 | 5 | -26 |
| -1,01 | X | -1,01 | -4,0299 | -13,109901 | X |
| 2,01 | X | X | 2,01 | 5,9898 | 26,090001 |
|  | 1 | 3,99 | 12,9801 | -0,090001 | 0,090001 |
| -1,01 | X | -1,01 | -3,0098 | -12,100103 | X |
| 2,01 | X | X | 2,01 | 5,9898 | 24,080403 |
|  | 1 | 2,98 | 11,9803 | -6,200304 | 24,170404 |

(11,9803 2,98 | 0,090001 )

(-6,200304 11,9803 |-0,090001 )

Δs=19,5551449/122,004494=0,120707422

Δr=-0,269707117/11,9803=0,120707422

r1=-1,01-0,022512551=-1,032512551

s1=2,01-0,120707422=1,889292578

|-0,0222512551 | =0,016988995

|-1,0325125510 |

|0,120707422 | = 0,063890274

|1,889292578 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 5 | 15 | 5 | -26 |
| -1,032512551 | X | -1,032512551 | -4,096480587 | -13,20873894 | X |
| 1,889292578 | X | X | 1,889292578 | -7,495744591 | 2416936474 |
|  | 1 | 3,967487449 | 12,79281199 | -0,7129943490 | -1,83063526 |
| -1,032512551 | X | -1,032512551 | -3,030398419 | -12,03053284 | X |
| 1,889292578 | X | X | 1,889292578 | 5,545026291 | 22,01348195 |
|  | 1 | 2,934974898 | 11,65170615 | -7,198500898 | 20,18284669 |

(11,65170615 2,934974898 | 0,712994349 )

(-7,198500898 11,65170615 | 1,83063526 )

Δs=1,83063526/156,8896756=0,1011668296

Δr=0,712994349/11,6517015

r1=-1,026687551

s1=1,900960874

|-0,0015825 | =0,005673585887

|-1,026687551 |

|0,011668296 | = 0,00613810424

|1,900960874 |

P(x)=(x²+3,967487449x+12,79281199).(x²+1,026687551x-1,900960874)

R1:(-3,967487449-sqrt(3,967487449²-(4\*12,79281199)))/2=(-3,967487449-sqrt(-37,499833452))/2= -3,967487449/2-6,123710758i/2=-1,983743725–3,061855379i

R2:(-3,967487449+sqrt(3,967487449²-(4\*12,79281199)))/2=-1,983743725+3,061855379i

R3:(-1,026687551-sqrt(1,026687551²-(4\*(-1,900960874))))/2=(-1,026687551-2,942436205)/2= -3,969123756/2=-1,984561878

R4:(-1,026687551+sqrt(1,026687551²-(4\*(-1,900960874))))/2=(-1,026687551+2,942436205)/2= 0,957874327

## 3.2 Casos de Pruebas de Integración

# 4 Pruebas

Abreviaturas:

DE: Datos de Entrada.

DS: Datos de Salida.

## 4.1 Pruebas Unitarias

De los casos de pruebas detallados anteriormente en el punto 3.1 los siguientes algoritmos reaccionaron de la siguiente manera:

Módulo de PosiblesRaicesEnteras:

Caso 1:

DE: P(x)=-x^2+0x+1

DS: 1,-1

Caso 2:

DE: P(x)=x^2+1+0

DS: 0.0

Caso 3:

DE: P(x)=x^2+1+2.5

DS: 1,-1,2,-2

Caso 4:

DE: P(x)=x^2+x-3

DS: 1,-1,3,-3

Módulo de Posibles Raices Racionales:

Caso 1:

DE: P(x)=-x^2+x+2

DS: 0.0

Caso 2:

DE: P(x)=-x^2+x+0

DS: 0.0

Caso 3:

DE: P(x)=x^2+x+3

DS: 0.0

Caso 4:

DE: P(x)=3x^2+x+1

DS: 0.33 , -0.33

Caso 5:

DE: P(x)=2x^2+x-1

DS: 0.50 , -0.50

Módulo de Lagrangue:

Caso 1:

DE: P(x)=4x^4+0x^3-5x^2+0x+1

DS:

Cota Superior Positiva: 2.1180339887498949E+000

Cota Inferior Positiva: 3.0901699437494740E-001

Cota Superior Negativa: -2.1180339887498949E+000

Cota Inferior Negativa: -3.0901699437494740E-001

Caso 2:

DE: P(x)=4x^4-8x^3+3x^2+2x-1

DS:

Cota Superior Positiva: 3.0000000000000000E-000

Cota Inferior Positiva: 4.1421356237309509E-001

Cota Superior Negativa: -3.3333333333333333E+001

Cota Inferior Negativa: -1.7937005259840997E+000

Módulo de Laguerre:

Caso 1:

DE: P(x)=4x^4+0x^3-5x^2+0x+1

DS:

Cota Superior Positiva: 1.5

Cota Inferior Positiva: 0.4

Cota Superior Negativa: -0.4

Cota Inferior Negativa: -1.5

Caso 2:

DE: P(x)=4x^4-8x^3+3x^2+2x-1

DS:

Cota Superior Positiva: 2.0000000000000000E+000

Cota Inferior Positiva: 5.000000000000000E-001

Cota Superior Negativa: -2.9999999999999999E+001

Cota Inferior Negativa: -5.9999999999999999E+001

Módulo de división de un polinomio por uno de grado 1.

**Polinomio.ruffini**(divisor, cociente, resto):bool

Caso 1

Datos Esperados:

DS: Cociente= ; Resto= -15/18;

Datos Retornados:

Cociente= ; Resto= -0.833333333; ruffini= true;

Caso 2

Datos Esperados:

Cociente= 1/2; Resto= 1;

Datos Retornados: Cociente= 0.5; Resto= 1; ruffini= true;

Caso 3

Datos Retornados:

Ruffini= false;

Módulo de división por un polinomio de la forma

**Polinomio.HornerCuadratico**(divisor, cociente, resto) bool

Caso 1

Datps Esperados:

Cociente= ; Resto=

Datos Retornados:

Cociente= ; Resto= ; hornerCuadratico= true

Caso 2

Datos Esperados:

Cociente= ; Resto=

Datos Retornados:

Cociente= ; Resto= ; honerCuadratico=true;

Caso 3

Datos Retornados:

HornerCuadratico= false;

Módulo de obtención de cotas por Newton

**Polinomio.cotasNewton**():vector;

Cotas=[infNeg,supPos,infPos,supPos]

Caso 1

Raices= {-1,1,3}

Dato Retornado:

Cotas=[-1.2; -0.83; 0.83; 3]

Caso 2

Sin raíces Reales;

DS: cotas[0;0;0;0];

Caso 3

Datos esperados:

Sin cotas Negativas; Intervalo Positivo= [4; 5];

Datos retornados:

Cotas= [0; 0; 4; 5]

Módulo de Obtención de Raíz por Newton

Caso 1

Dato Esperado: “no tiene raíces reales”

Dato Retornado: raíz = 9999999 (Constante INFINITO)

Caso 2

Dato Esperado: raíz = -1

Caso de prueba 2.1

DE: x0 = 1; error= 0.0001

Dato Esperado: “el método no converge”

Dato Retornado: 9999999 (Constante INFINITO)

Caso de prueba 2.2

DE: x0 = -2; error= 0.0001

Dato Esperado: raíz = -1;

Dato Retornado: raíz = -1

Módulo de Sturm:

Caso 1:

DE: P(x)=4x^4+0x^3-5x^2+0x+1

DS:

Intervalo de Raíces Negativas: [-0.80,-0.40] ; [-1.20,-0.80]

Intervalo de Raíces Positivas: [0.40,0.80] ; [0.80,1.20]

## 4.2 Pruebas de Integración

# 5 Conclusión

Alarcon,Pablo: *Me pareció que no nos pudimos organizar para este programa de la mejor manera y tuvimos mucho tiempo sin contactarnos.Tuve una forma de realizar los métodos que incomodo a mis compañeros, y aprendí que no debo hacer para la próxima.También pude aprender y profundizar más conceptos de los métodos que me tocaron realizar.*

Mamaní, Cristian: *Creo que al igual que varios de mis compañeros empecé a ganar experiencia en el uso de Git y GitHub; son herramientas muy potentes y muy reconocidas en el ámbito de los progamadores, nos van a servir para el mundo laboral.*