Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

Método de la ingeniería

Fase 1: Identificación del problema

La empresa especializada en desarrollo de soluciones de nube y locales Oracle desea lanzar una nueva funcionalidad dentro de Java, la cual es *encontrar las raíces de un polinomio*, implementando al menos 2 algoritmos que solucionen esa tarea, debido a que los estudiantes de ingeniería y desarrolladores necesitan de estas funciones básicas para sus programas.

- **Problema:**
  - Dado un polinomio de grado ***n***

$$\boldsymbol{P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0 \mid 0 \leq n \leq 10}$$

  - Encontrar todos los valores de ***x*** que hagan a ***P(x) = 0***

$$\boldsymbol{P(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x^1 + a_0 x^0 = 0}$$

- **Requerimientos Funcionales:**

Nombre	**R1**
Resumen	Encontrar las raíces reales de un polinomio de grado n
Entrada	Polinomio
Salida	Raíces reales

Nombre	**R2**
Resumen	Encontrar las raíces imaginarias de un polinomio de grado n
Entrada	Polinomio
Salida	Raíces imaginarias

Nombre	**R3**
Resumen	General polinomio aleatorio de grado 10
Entrada	Ninguna
Salida	Polinomio de grado 10

Nombre	**R4**
Resumen	Generar un polinomio
Entrada	• n -> Grado máximo del polinomio • An -> Constantes del polinomio
Salida	Polinomio generado

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

- **Requerimiento No Funcionales**

Nombre	**RNF1**
Resumen	La complejidad temporal de los algoritmos para encontrar las raíces de un polinomio no debe ser de O(2^n)

Nombre	**RNF2**
Resumen	La complejidad espacial de los algoritmos para encontrar las raíces de un polinomio no debe ser de O(n^2)

Nombre	**RNF3**
Resumen	La interfaz gráfica de usuario que le permita ingresar el polinomio que desea resolver.

Nombre	**RNF4**
Resumen	La interfaz gráfica que le permita generar aleatoriamente polinomios hasta de grado 10, los cuales puedan servir de entrada a los algoritmos que ustedes proponen.

Nombre	**RNF5**
Resumen	La interfaz gráfica que le permita ver al usuario las raíces del polinomio que fue entregado como entrada y qué método fue utilizado para encontrarlas.

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

Fase 2: Recopilación de información

Definiciones:

Referencias

educativo, P. (8 de Febrero de 2019). *Portal Educativo*. Obtenido de https://www.portaleducativo.net/primero-medio/46/factorizacion

Kurosch, A. .. (1 de Noviembre de 2014). *ecured*. Obtenido de https://www.ecured.cu/Teorema_fundamental_del_%C3%A1lgebra

Tutors, V. (8 de Febrero de 2019). *varsityTutors*. Obtenido de https://www.varsitytutors.com/hotmath/hotmath_help/spanish/topics/descartes-rule-of-signs

Polinomio:
Un polinomio es una expresión algebraica constituida por la suma o la resta de una serie de términos constantes o variables, los cuales deben ser finitos y pueden tomar exponentes de valores definidos, tomando su exponente se considera el grado del polinomio.

Raíces de un polinomio:
Se dice que un valor x = a es raíz de un polinomio P(x), cuando al sustituir dicho valor en el polinomio, el resultado es 0; es decir, cuando P(a) = 0

Función polinómica:
Las funciones polinómicas son, como su nombre lo dice, funciones que constan de un polinomio.

$$f(x) = ax^n + bx^{n-1} + \ldots\ldots + jx + h$$

en donde n es un entero positivo, llamado, grado del polinomio. Resulta evidente, que el coeficiente del grado mayor, no puede ser cero, o sea, a tiene que ser diferente de cero, para que el grado del polinomio se n. Cualquiera de los otros coeficientes puede ser cero.

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

Factorización:

Factorizar una expresión algebraica (o suma de términos algebraicos), es el procedimiento que permite escribir como multiplicación dicha expresión.

Los factores o divisores de una expresión algebraica son los términos, ya sean números y/o letras, que multiplicados entre sí dan como producto la primera expresión.

$$\mathrm{A}(\mathrm{A}+\mathrm{B}) = \mathrm{A}^2 + \mathrm{B}$$

(educativo, 2019)

Teorema fundamental del algebra:

En Matemáticas y más específicamente Álgebra superior, Análisis Matemático, Geometría y funciones de variable compleja, es un teorema que plantea que todo polinomio no constante de una variable tiene al menos una raíz.

Del presente se deriva que todo polinomio p(x) de una variable no constante tiene la misma cantidad de raíces reales o complejas que su grado n, resultado teórico que es vital para el cálculo matemático.

Sea el polinomio de grado n (n>0) de una variable:

- $p(x)=a_0+a_1x+a_2x^2+...+a_nx^n$.

Existe un número r tal que p(r)=0 o lo que es lo mismo, pero expresado como una factorización:

- $p(x)=(x-r)(b_0+b_1x+...+b_{n-1}x^{n-1})$

  (Kurosch, 2014)

Regla de los signos de descartes:

La regla de los signos de Descartes nos ayuda a identificar el número posible de raíces reales de un polinomio p(x) sin graficar o resolverlas realmente. Dese cuenta por favor que esta regla no proporciona el número exacto de raíces del polinomio ni identifica las raíces del polinomio.

La regla establece que el número posible de las raíces positivas de un polinomio es igual al número de cambios de signo en los coeficientes de los términos o menor que los cambios de signo por un múltiplo de 2.
(Tutors, 2019)

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

Fase 3: Búsqueda de soluciones creativas

Para la solución de este problema necesitamos enfocarnos en los algoritmos necesario para hallar las raíces de un polinomio, para esto vamos a generar las ideas usando conocimientos aprendidos en diferentes cursos de cálculo, sobre hallar las raíces de un polinomio, y así poder buscar las alternativas que mejor se adapten a la solución de problema.

Alternativa 1: Fuerza bruta con el método de Horner.

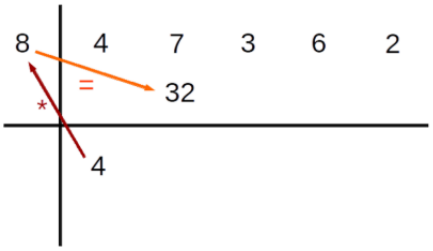
Esta solución se basa en usar el método de Horner para probar valores particulares de x hasta que P(x) = 0. Aunque el calcular la solución de un polinomio para un determinado valor de x es tarea sencilla, el método de Horner reduce drásticamente la cantidad de operaciones para llegar a ese resultado, esto lo hace un excelente método para buscar las raíces del polinomio probando distintos valores de x.

Para explicar cómo funciona el método de Horner vamos a tomar el siguiente Polinomio:

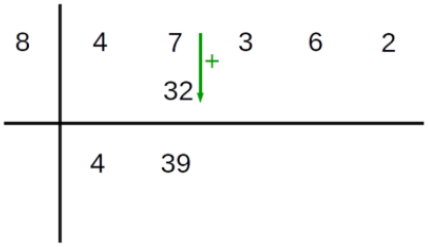
$$4x^4 + 7x^3 + 3x^2 + 6x + 2$$

Y lo vamos a evaluar en x = 8, esto debería dar 20210

Colocamos los coeficientes del polinomio en una tabla junto con el valor de x que quiere evaluarse. Bajamos el primer coeficiente y lo multiplicamos por el valor de x colocando el resultado debajo del siguiente coeficiente en la tabla

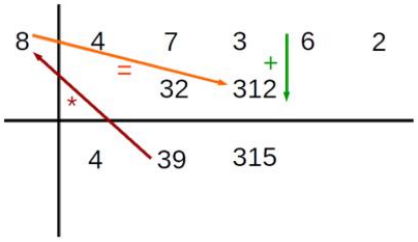


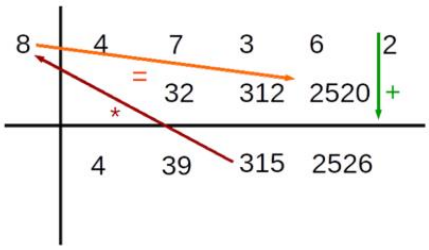
Sumamos los dos valores obteniendo un nuevo resultado parcial



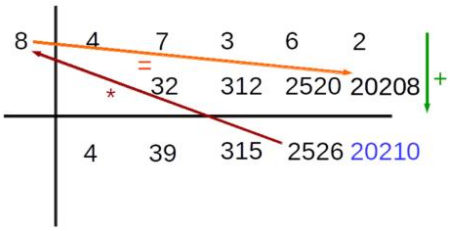
Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

Repetimos la operación para cada coeficiente





Al llegar al último coeficiente obtenemos el resultado final



Alternativa 2: Método de Newton para polinomios.

Este método es un algoritmo se usa para encontrar las raíces de un polinomio de cualquiera grado. Este método parte de una aproximación inicial para para la raíz x_0 y obtiene una aproximación mejor x_1, la cual se obtiene a partir de la fórmula:

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

El método de Newton consiste en una lineación del polinomio, es decir, *f* se reemplaza por una recta tal que contiene al punto (x_0,$f(x_0)$) y cuya pendiente coincide con la derivada de la función en el punto, $f'(x_0)$. La nueva aproximación a la raíz, x_1, se obtiene de la intersección de la función linear con el eje *X* de ordenadas.

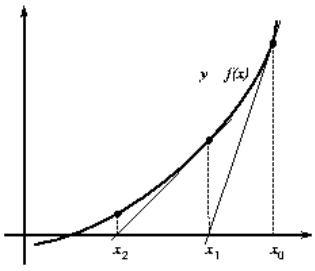
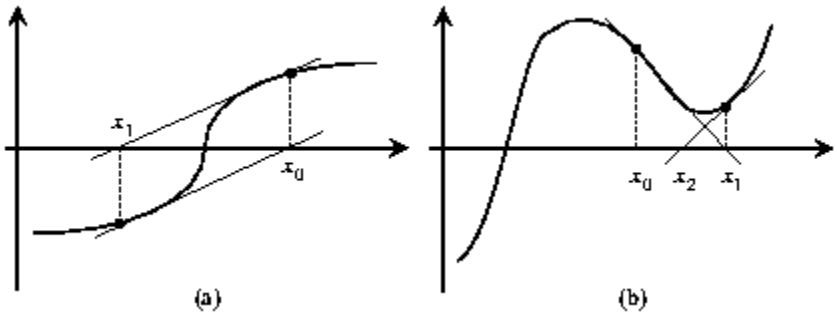


Figure: Interpretación geométrica del método de Newton.

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

El método de Newton es muy rápido y eficiente ya que la convergencia es de tipo cuadrático (el número de cifras significativas se duplica en cada iteración). Sin embargo, la convergencia depende en gran medida de la forma que adopta la función en las proximidades del punto de iteración. En la siguiente figura se muestran 2 situaciones en las que el método no es capaz de alcanzar la convergencia o converge a un punto, el cual no es una raíz.



Alternativa 3: Fórmula cuadrática.

La fórmula cuadrática es una manera segura de resolver polinomios de la forma

$$P(x) = ax^2 + bx + c$$

Colocando los valores a, b y c en la siguiente formula, podremos obtener los valores de x para los cuales $P(x) = 0$.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Alternativa 4: Método de Bairstow.

El método de Lin Bairstow es un método iterativo, basado en el método de Müller y de Newton Raphson. Dado un polinomio $f_n(x)$ se encuentran dos factores, un polinomio cuadrático $f_2 = x^2 - rx - s$ y $f_{n-2}(x)$. El procedimiento general para el método de Lin Bairstow es:

1. Dado $f_n(x)$ y r_0 y s_0
2. Utilizando el método de NR calculamos $f_2(x) = x^2 - r_0x - s_0$ y $f_{n-2}(x)$, tal que, el residuo de $f_n(x)/ f_2(x)$ sea igual a cero.
3. Se determinan las raíces $f_2(x)$, utilizando la formula general.
4. Se calcula $f_{n-2}(x)= f_n(x)/ f_2(x)$.
5. Hacemos $f_n(x)= f_{n-2}(x)$
6. Si el grado del polinomio es mayor que tres regresamos al paso 2
7. Si no terminamos

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

La principal diferencia de este método, respecto a otros, es que permite calcular todas las raíces de un polinomio (reales e imaginarias).

Alternativa 5: Método de bisección

Este método consiste en obtener una aproximación muy cercana a la raíz a partir de un intervalo inicial (a, b), en el cual hay un cambio de signo en la función, es decir: f(a)f(b)<0. Se obtiene el punto medio a partir de esta ecuación:

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

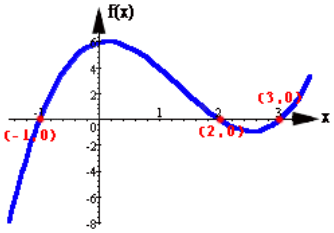
x_m es la nueva aproximación a la raíz, y se vuelve a tomar un intervalo, pero ahora más pequeño, considerando que siga existiendo un cambio de signo en la función. El método termina cuando se cumple con alguna condición de paro, en este programa la condición es la tolerancia:

$$|x_{i+1} - x_i| \leq \varepsilon$$

Este es un método "de encierro", para aplicarlo se debe contar con un intervalo inicial, en donde $f(a) * f(b) < 0$. Este método requiere de menos pasos en un programa, sin embargo, converge más lentamente que el de Newton-Raphson.

Alternativa 6: Método Grafico

Este es un método simple para obtener una aproximación de las raíces del polinomio. El método consiste en graficar la función y observar donde se corta con el eje x, para el cual ofrece una aproximación inicial de las raíces.



Como podemos ver en la imagen las raíces de este polinomio serian:

$$x = -1, x = 2 \; y \; x = 3.$$

El mayor inconveniente de este método es su poca precisión y exactitud. Sin embargo, en la actualidad se puede rápidamente graficar funciones con un alto grado de realismo.

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

Alternativa 7: Método de la regla falsa

Este método es muy similar a la alternativa 5, ambos son métodos acotados. Es un método matemático que de forma algorítmica busca las soluciones aproximadas de una ecuación en un determinado intervalo.

Para hacer este método se tiene que seguir los siguientes pasos:

1. Selección de un intervalo [a, b] donde hallan un cero.
2. Calcular un punto de intersección como nuevo punto.

$$\frac{f(b)-f(a)}{b-a}=\frac{f(b)}{b-c} \qquad c=b-\frac{f(b)(b-a)}{f(b)-f(a)}$$

3. Comprobar si hay un cambio de signo en [a, c] o en [c, b].
4. Si el producto es cero entonces es una raíz.
5. Sino volver al punto 2.

Con cada iteración, se obtiene un resultado muy aproximado, más no exacto.

Fase 4: Transición de la formulación de ideas a los diseños preliminares

- **Descarte de ideas no factibles**

Se descartaron las siguientes opciones de la búsqueda de soluciones creativas debido a:

Alternativa 1	Esta alternativa se descartó porque pese a que el método de Horner para resolver un polinomio para un determinado valor de x es muy efectivo, el ir probando diferentes valores de x hasta que P(x) = 0 podría tener una complejidad temporal de O(n!) o nunca acabar.
Alternativa 3	Esta alternativa se descartó debido a que la formula cuadrática sólo funciona para polinomios de grado 2, esto no resultaría eficaz porque el usuario puede ingresar polinomios de mayor grado.
Alternativa 5	Esta alternativa se descartó debido a que este algoritmo es mucho más lento que el método de Newton además da aproximaciones menos precisas de las raíces del polinomio.
Alternativa 6	Esta alternativa se descartó debido a que los valores de x para los cuales P(x) = 0, son demasiados imprecisos, además el graficar la función para obtener las raíces resultaría muy complicado y costoso en cuanto al tiempo.

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

- **Diseños preliminares**

Para los diseños preliminares hemos decidido hacer el pseudocódigo de los 3 algoritmos más importantes para así podernos dar una idea de cómo implementarlos en código. Además, aprovecharemos el pseudocódigo para sacar la complejidad espacial y temporal de cada algoritmo.

Método 1:

#	bairstow(a[], r0, s0, re[], im[])	C.E	C.T
1	n = a.length, iter =0;	2	2
2	b[] = new double[n], c[] = new double[n]	2*n	2
3	roots[] = new double[n]	n	1
4	ea1 = 1, ea2 = 1, T = 0.00001	3	3
5	r=r0, s=s0,det, ds, dr	5	4
6	MaxIter = 100, i	2	2
7	for(iter=0; iter< MaxIter && n>3; iter++)	1	n+1
8	do		n+Log(n)
9	Division_Derivada(a, b, c, r, s, n)	6	(n-1)+Log(n)
10	det = c[2]*c[2] - c[3]*c[1]	1	Log(n)-1
11	if(det!=0)		Log(n)-1
12	dr = (-b[1]*c[2] + b[0]*c[3])/det	1	Log(n)-1
13	ds = (-b[0]*c[2] + b[1]*c[1])/det	1	Log(n)-1
14	r = r+dr	1	Log(n)-1
15	s = s+ds	1	Log(n)-1
16	if(r!=0)		Log(n)-1
17	ea1 = Math.abs(dr/r)*100.0	1	Log(n)-1
18	if(s!=0)		Log(n)-1
19	ea2 = Math.abs(ds/s)*100.0	1	Log(n)-1
20	Else		0
21	r = 5*r+1	1	0
22	s = s+1	1	0
23	iter = 0	1	0
24	while ((ea1 > T) && (ea2 > T))		n+Log(n)
25	raices(r, s, re, im, n)	5	6
26	n = n-2	1	1
27	for(i=0; i<n; i++)		n+1
28	a[i] = b[i+2]	1	n
29	if (n < 4) break		1
30	if(n==3)		1
31	r = -a[1]/a[2]	1	1
32	s = -a[0]/a[2]	1	1
33	raices(r, s, re, im, n)		6
34	else		0
35	re[n-1] = -a[0]/a[1]	1	0

36	im[n-1] = 0	1	0
37	for(i=1; i<re.length; i++)		n
38	roots[i]=re[i]		n-1

Método 2:

#	**evaluate (x0)**	**C.E**	**C.T**
1	y = a[n-1]	n	1
2	z = a[n-1]	n	1
3	for(int j = n-2; j>0; j--)	1	n-1
4	y = x0*y + a[j]		n-2
5	z = x0*z + y		n-1
6	y = x0*y + a[0]		1

Método 3:

#	**solveRoots ()**	**C.E**	**C.T**
1	Horner h=new Horner(a)	1	1
2	double f=1, d	2	2
3	while(f>tol)		n
4	h.evaluate(p)	2n+1	(3n-1)+(n-1)
5	f = h.val()		(1)+(n-1)
6	d = h.der()		(1)+(n-1)
7	p = p - f/d		n-1
8	return p		1

Fase 5: Evaluación y selección de la mejor solución.

Actualmente tenemos tres alternativas de solución, las cuales resuelven nuestro problema de buscar raíces, como sólo necesitamos dos alternativas, definiremos una serie de criterios que nos ayudaran a escoger las dos mejores en base al número de puntos.

Criterios:

- **Criterio A: Precisión en la solución.**
  Este criterio se basa en que tan exacta es la raíz dada por la alternativa respeto a la real.
  - Raíz exacta: 3 Puntos
  - Raíz aproximada precisa: 2 Puntos
  - Raíz aproximada imprecisa: 1 Punto

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

- **Criterio B: Raíces Imaginarias.**
  Este criterio se basa si la alternativa puede dar una solución imaginaria a las raíces del polinomio.
  - Número Imaginario: 3 Puntos
  - Número real: 2 Puntos

- **Criterio C: Complejidad Temporal.**
  Este criterio se basa en que tan rápido la alternativa puede dar respuesta al problema.
  - Constante: 6 Puntos
  - Logarítmica: 5 Puntos
  - Lineal: 4 Puntos
  - Polinomial: 3 Puntos
  - Exponencial: 2 Puntos
  - Factorial: 1 Punto

- **Criterio D: Complejidad Espacial.**
  Este criterio se basa en cuanta memoria necesita la alternativa para dar una respuesta al problema.
  - Constante: 6 Puntos
  - Logarítmica: 5 Puntos
  - Lineal: 4 Puntos
  - Polinomial: 3 Puntos
  - Exponencial: 2 Puntos
  - Factorial: 1 Punto

- **Criterio E: Numero de raíces.**
  Este criterio se basa en cuantas raíces del polinomio da la alternativa.
  - Todas: 3 Puntos
  - Algunas: 2 Puntos
  - Una: 1 Punto

Evaluación:

# Alternativa	Criterio A	Criterio B	Criterio C	Criterio D	Criterio E	Total Puntos
Alternativa 2	2	2	4	4	1	13
Alternativa 4	2	3	5	4	3	17
Alternativa 7	1	2	3	4	1	11

Con base en los resultados obtenidos vamos a descartar la alternativa 7 e implementar las alternativas 2 y 4 para la solución de nuestro problema.

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

Paso 6: Preparación de informe y especificaciones.

- **Análisis de complejidad temporal**

  - ✓ **Método 1:**
    T(n) = $7n + 13\log_2 n + 6$
    T(n) = $An + B\log_2 n + C$
    Por medio del T(n) anterior podemos concluir que la notación asintótica para la complejidad temporal del método 1 es: $O(n)$.

  - ✓ **Método 2:**
    T(n) = $3n - 1$
    T(n) = $An + B$
    Por medio del T(n) anterior podemos concluir que la notación asintótica para la complejidad temporal del método 2 es: $O(n)$.

  - ✓ **Método 3:**
    T(n) = $8n + 1$
    T(n) = $An - B$
    Por medio del T(n) anterior podemos concluir que la notación asintótica para la complejidad temporal del método 3 es: $O(n)$.

- **Análisis de complejidad espacial**

  - ✓ **Método 1:**
    T(n) = $3n + 28$
    T(n) = $An + B$
    Por medio del T(n) anterior podemos concluir que la notación asintótica para la complejidad espacial del método 1 es: $O(n)$.

  - ✓ **Método 2:**
    T(n) = $2n + 1$
    T(n) = $An + B$
    Por medio del T(n) anterior podemos concluir que la notación asintótica para la complejidad espacial del método 2 es: $O(n)$.

  - ✓ **Método 3:**
    T(n) = $3n + 4$
    T(n) = $An + B$
    Por medio del T(n) anterior podemos concluir que la notación asintótica para la complejidad espacial del método 3 es: $O(n)$.

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

Diseño de pruebas unitarias

- **bairstow**

Objetivo: Probar que el método bairstow() funcione correctamente para diferentes casos de prueba				
Clase: BairstowMethod		**Metodo:** bairstow		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
1	Dado un polinomio guardado en un arreglo, el método se encarga de calcular sus respectivas raíces (caso base)	scenarioOne()	thePoly = new Polynomial(new double[] {6,5,1})	El algoritmo devuelve las raíces del polinomio dado: -3,-2

Objetivo: Probar que el método bairstow() funcione correctamente para diferentes casos de prueba				
Clase: BairstowMethod		**Metodo: bairstow**		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
2	Dado un polinomio guardado en un arreglo, el método se encarga de calcular sus respectivas raíces (caso limite)	scenarioTwo()	thePoly = new Polynomial(new double[] {1,-3,-4,12})	El algoritmo devuelve las raíces del polinomio dado: 0.5, -0.5, 0.3

Objetivo: Probar que el método bairstow() funcione correctamente para diferentes casos de prueba				
Clase: BairstowMethod		**Metodo: bairstow**		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
3	Dado un polinomio guardado en un arreglo, el método se encarga de calcular sus respectivas raíces (caso interesante)	scenarioThree()	thePoly = new Polynomial(new double[] {3,12})	El algoritmo devuelve las raíces del polinomio dado: -0.25

- **solveRoots**

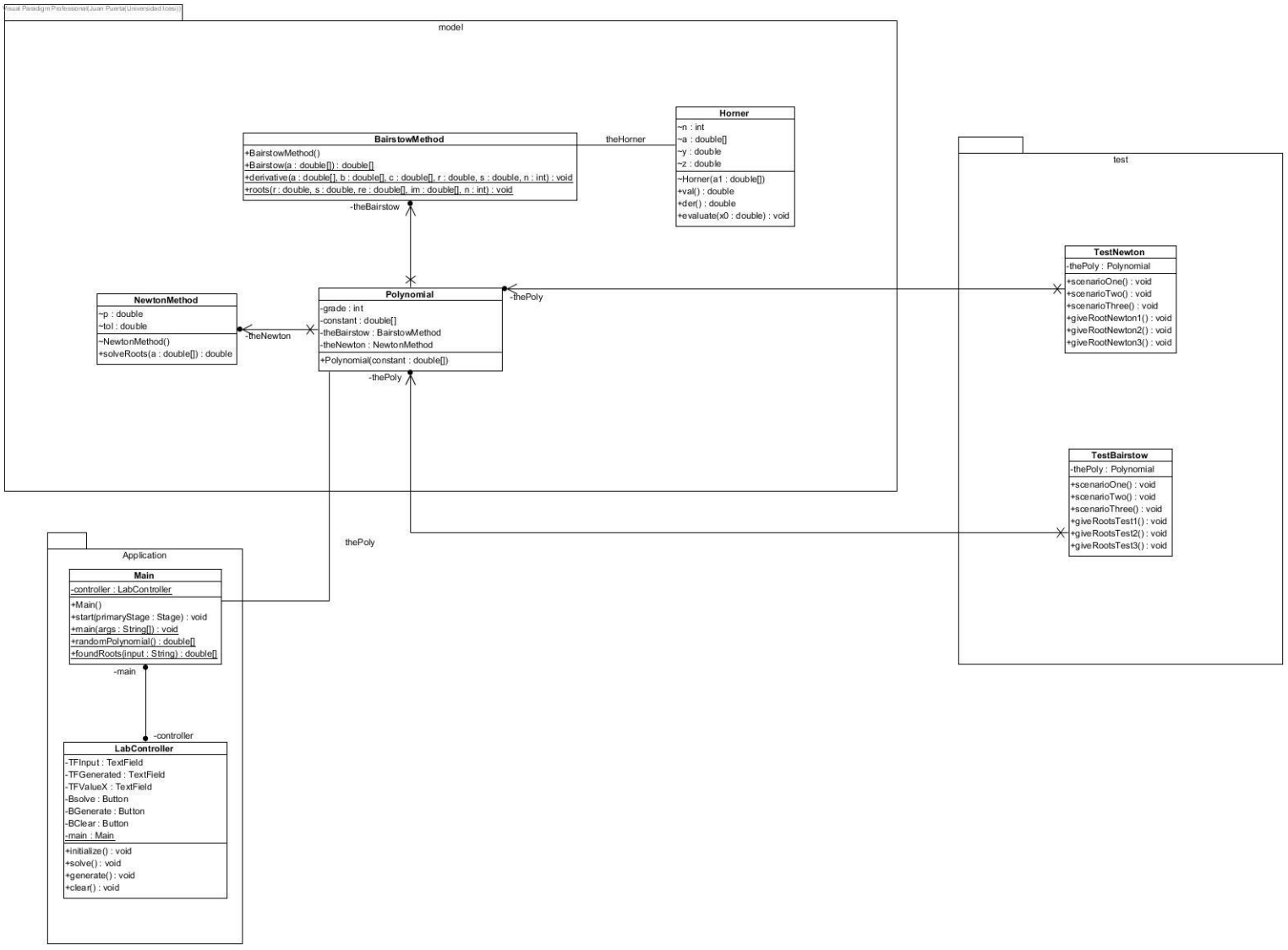
Objetivo: Probar que el método solveRoots() funcione correctamente para diferentes tipos de pruebas				
Clase: NewtonMethod		**Metodo:** solveRoots		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
1	Dado un polinomio de grado 4, el método debe de calcular una de sus raíces (caso base)	scenarioOne()	thePoly = new Polynomial(new double[] {1,-5,6,2})	El algoritmo devuelve la raíz del polinomio, el cual debe de tener un valor de (-5.54)

Objetivo: Probar que el método solveRoots() funcione correctamente para diferentes tipos de pruebas				
Clase: NewtonMethod		**Metodo:** solveRoots		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
2	Dado un polinomio de grado 10, el método debe de calcular una de sus raíces (caso limite)	scenarioThree()	thePoly = new Polynomial(new double[] {2,-5,6,7,2,3,6,1,6,4})	El algoritmo devuelve la raíz del polinomio, el cual debe de tener un valor de (-1.86)

Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

Objetivo: Probar que el método solveRoots() funcione correctamente para diferentes tipos de pruebas				
Clase: NewtonMethod		**Metodo:** solveRoots		
Caso #	**Descripción:**	**Escenario**	**Valores de entrada**	**Resultado**
3	Dado un polinomio de grado 6, el método debe de calcular una de sus raíces (caso interesante)	scenarioTwo()	thePoly = new Polynomial(new double[] {2,-5,6,7,2,3})	El algoritmo devuelve la raíz del polinomio, el cual debe de tener un valor de (-1.63)

- Diagrama de Clases



Fabian David Portilla
Juan Sebastián Puerta
Juan Camilo Castillo

Fase 7: Implementación

La implementación de la solución se encuentra en el siguiente repositorio de github:

https://github.com/Juan-Puerta/Lab1-AED