Método de la ingeniería.

1. Oracle necesita 2 métodos para encontrar las raíces complejas y reales de un polinomio que máximo va a ser de grado 10 con exponentes enteros y coeficientes racionales.

-Se nos pide un informe donde analicemos las posibles soluciones y seleccionemos las más pertinentes.

-Dado que Oracle es una empresa conocida por su excelencia, podemos inferir que para ellos es importante que dentro de los criterios de para seleccionar la solución este la eficiencia temporal y espacial.

-Oracle quiere visualizar la funcionalidad de los algoritmos desde una interfaz gráfica.

|  |  |
| --- | --- |
| Nombre | Ingresar el polinomio que desea resolver |
| Resumen | El usuario puede proponer los coeficientes del polinomio al que le quiere hallar las raíces. |
| Entradas | Una serie de números racionales para representar los coeficientes del polinomio a resolver. |
| Resultados | Ninguna. |

|  |  |
| --- | --- |
| Nombre | Generar aleatoriamente polinomios |
| Resumen | El usuario puede generar un polinomio con coeficientes escogidos al azar. |
| Entradas | Ninguna. |
| Resultados | Una serie de números aleatorios enteros para representar los coeficientes del polinomio a resolver. |

|  |  |
| --- | --- |
| Nombre | Visualizar las raíces del polinomio |
| Resumen | Las posibles raíces del polinomio son desplegadas en la interfaz. |
| Entradas | Ninguna. |
| Resultados | Una lista de números reales representando las raíces del polinomio. |

|  |  |
| --- | --- |
| Nombre | Resolver el polinomio con 2 algoritmos diferentes |
| Resumen | Se le permite al usuario escoger que algoritmo quiere usar para resolver el problema. |
| Entradas | El algoritmo elegido para realizar la operación. |
| Resultados | Una lista de números reales representando las raíces del polinomio. |

2. Un polinomio del latín polynomium, y este del griego, πολυς polys ‘muchos’, ​ es una expresión algebraica constituida por una suma finita de productos entre variables (valores no determinados o desconocidos) y constantes (números fijos llamados coeficientes), o bien una sola variable. Las variables pueden tener exponentes de valores definidos naturales incluido el cero y cuyo valor máximo se conocerá como grado del polinomio.

Las raíces de un polinomio son números tales que hacen que un polinomio valga cero.

El teorema fundamental del cálculo dicta que, si P(x) es un polinomio de grado *n*entonces P(x) tendrá exactamente *n* raíces, donde alguna de ellas pueda que este repetida.

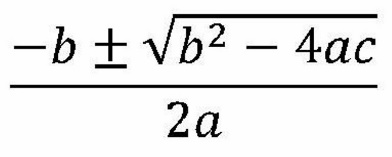
Existen algunos casos de polinomios donde se presentan raíces que son evidentes, es decir, que podemos averiguar simplemente mediante la observación y evitando hacer cálculos demasiado trabajosos.

De manera más general, si *r* es la raíz de un polinomio P(x) de grado *n*, entonces podemos escribir a P(x) como:

Donde Q(x) es un polinomio de grado *n* -1.

3.

¿Qué hacer cuando la solución no es tan evidente?

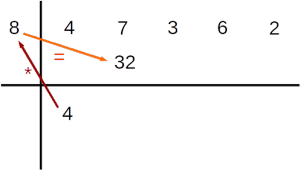
**Opción 1 \La fórmula de Bhaskara**, que te permitirá averiguar fácilmente cuales son las raíces racionales de un polinomio de grado 2.

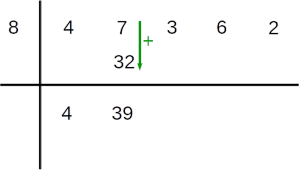
**Opción 2 \ El método de Ruffini** para polinomios nos permite dividir un polinomio de grado 2 o mayor con el fin de obtener uno de un grado menor, para seguir en la búsqueda de las raíces que aún no se conozcan. Sirve mucho en el caso de polinomios de tercer grado, ya que si logramos descifrar alguna de sus raíces podremos obtener uno de grado 2 y finalmente mediante la fórmula de Bhaskara, averiguar las dos raíces restantes.

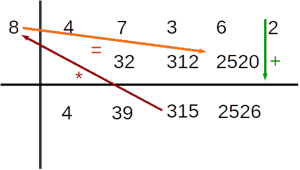
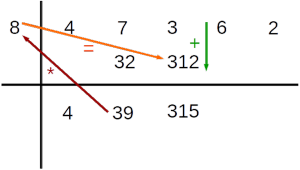
Los siguientes métodos se ven apoyados en gran medida por **método de Horner** que presentaba una nueva manera más rápida de calcular el valor de un polinomio y de manera análoga su convergencia a 0:

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

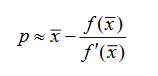


Sumamos los dos valores obteniendo un nuevo resultado parcial

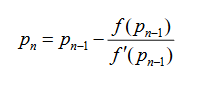
Repetimos la operación para cada coeficiente

Al llegar al último coeficiente obtenemos el resultado final.

Podemos ver en este ejemplo como se ha reducido la cantidad de multiplicaciones necesarias ahorrando al sistema 6 operaciones extras, puede representar un avance significativo en la eficiencia de los algoritmos siguientes.

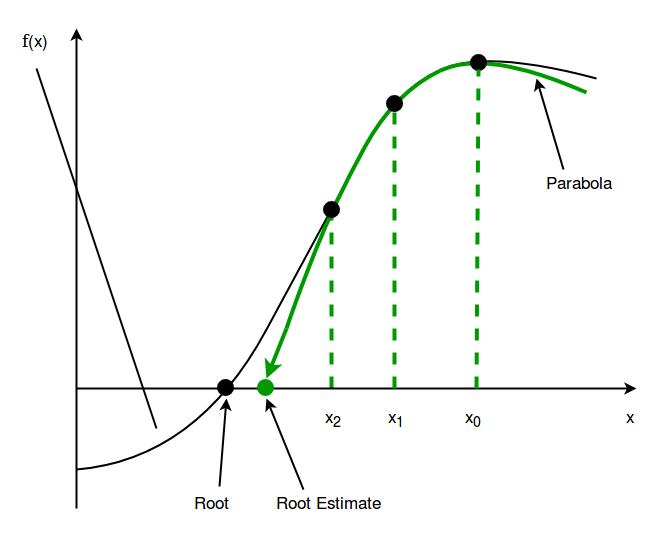
**Opción 3 \ El método de Newton** para hallar las raíces de la ecuación f(x) = 0, es el más conocido, y a menudo, el más efectivo.

El método de Newton consiste en tomar una aproximación inicial, x, y a continuación obtener una aproximación más refinada mediante la fórmula de arriba. Es decir, se trata de acercarnos a la raíz p por medio de la fórmula recursiva:

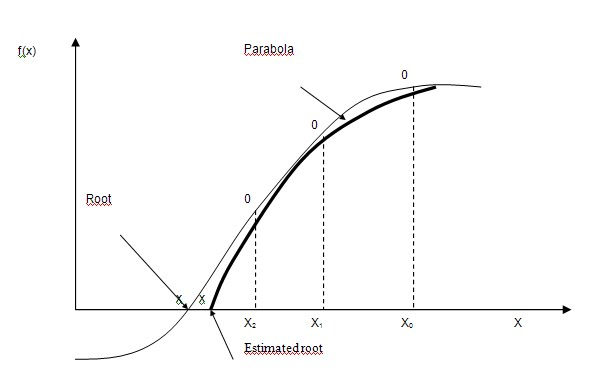


**Opción 4 \ El Método Muller** es un algoritmo de búsqueda de raíces para encontrar la raíz de una ecuación de la forma, f (x) = 0. Fue descubierto por David E. Muller en 1956.

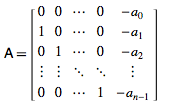
Comienza con tres supuestos iniciales de la raíz, y luego construye una parábola a través de estos tres puntos, y toma la intersección del eje x con la parábola como la siguiente aproximación. Este proceso continúa hasta que se encuentra una raíz con el nivel de precisión deseado.

Aunque es más lento que el método de Newton - Raphson, que tiene una tasa de convergencia de 2, pero supera uno de los mayores inconvenientes del método de Newton-Raphson, es decir, el cálculo de la derivada en cada paso.

**Opción 5 \** **El enfoque de Bairstows** es usar el método de Newton para ajustar los coeficientes u y v en la ecuación cuadrática x^2 + ux + v hasta que sus raíces también sean las raíces del polinomio resuelto. Las raíces de la ecuación cuadrática pueden entonces determinarse, y el polinomio puede dividirse por la ecuación cuadrática para eliminar esas raíces. Este proceso se itera entonces hasta que el polinomio se vuelve cuadrático o lineal, y se han determinado todas las raíces.



**Opción 6 \**Podemos ir un poco más allá y representar el polinomio como una matriz, a esta matriz se le llamara **matriz compañera**:



El vector:

Representa un vector propio de la matriz de valor propio t, cuando t es una raíz del polinomio característico *p*(*t*), utilizando esta propiedad podemos encontrar las raíces tanto imaginarias como complejas de esta matriz.

Más específicamente podemos usar la **descomposición en valores singulares** para factorizar **A**, esto es posible gracias a nociones del algebra lineal tales como matrices ortogonales, valores singulares y bases ortonormales con todos conceptos podemos crear una ecuación en la que podemos encontrar los valores buscados:

**Opción 7\** También podríamos implementar un **algoritmo genético** para que aprenda a solucionar las raíces de un polinomio. Son llamados así porque se inspiran en la evolución biológica y su base genético-molecular.

Estos algoritmos hacen evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados.

4. -Basada en la investigación anterior podemos determinar que tratar de crear un método para hallar raíces resulta más complejo de lo que se quisiera admitir, aunque tratar de hacer una búsqueda completa de las raíces sería un ejercicio interesante, no resultaría particularmente útil, dado que la complejidad temporal podría (y sería lo más probable) ser exponencial. Incluso creando restricciones a partir de las opciones que nos plantea Ruffini (**Opción 2**) con la división del coeficiente de mayor grado sobre el de menor grado, para determinar las posibles raíces el proceso no sería remotamente tan eficiente como el de las siguientes opciones a considerar.

La **Opción 1** resulta aún más inoportuna dado qué no se acerca siquiera a los requerimientos mínimos que debe cumplir el algoritmo (qué se encuentran en el punto 1).

**Opción 3** resulta más oportuna pero aun corta en lo que requiere el cliente dado que el método de newton solo es capaz de dar 1 raíz del polinomio, aunque nos tomáramos la molestia de crear un algoritmo para hacer división sintética, se seguiría quedando corto en funcionalidad dado que no es capaz de hallar raíces complejas. Pero uno de los peores problemas es que requiere de un valor inicial para realizar los cálculos y este determina gran parte de la efectividad del método.

**Opción 4** se acerca un poco más a lo requerido por el problema ya que permite hallar raíces complejas, pero quedándose corto en los mismos aspectos que la **Opción 3,** depende ahora de 3 valores iniciales que aunque pueden mejorar la convergencia, quedan igual de cortos; y además solo halla la 1 raíz a la vez.

**Opción 5** es el primer algoritmo de la lista que cumple con los requerimientos mínimos del algoritmo (punto 1), es capaz de calcular todas las raíces incluyendo complejas y las calcula todas. El único inconveniente es que la convergencia y la precisión que no siempre son las mejores.

**Opción 6** es el algoritmo que ejecuta de manera más limpia y estable la funcionalidad requerida, cumple con todos los requerimientos de la **opción 5** con una buena precisión y siempre asegurando la convergencia. El problema es que es implementado partiendo de una librería, sin la anterior herramienta implementar dicha solución sería una tarea bastante complicada por todos los conceptos y operaciones que se deberían implementar patas matrices.

**Opción 7** representa una solución por fuera de la experticia de los programadores (y de los fines del curso) así, aunque sería capaz de cumplir todos los requerimientos debe ser eliminada**.**

-Codificar los métodos de Newton, Muller o Bairstows, este ejercicio resulta más conveniente dado que estos procedimientos están dados en sí mismos como algoritmos iterativos en los que se busca a partir de un proceso matemático tratar de aproximar los valores de las raíces.

5.

Criterio 1 – Eficiencia temporal:

[3]- Logarítmica

[2]- Lineal

[1]- Mayor o igual a cuadrática

Criterio 2 – Eficiencia espacial:

[2]- Menor que cuadrática

[1]- Mayor o igual a cuadrática

Criterio 3 – Precisión:

[1]-Inexacta

[3]- Precisa

Criterio 4 – Convergencia:

[1] – Algunas veces

[2] – Casi siempre

[3] - Siempre

Criterio 5 – Facilidad de evaluación:

[2] – Operaciones básicas de java

[1] – Implementación de librerías externas

\*Evaluación:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Criterio 1 | Criterio 2 | Criterio 3 | Criterio 4 | Criterio 5 | Total |
| Opción 3 | 2 | 2 | 1 | 1 | 2 | 8 |
| Opción 4 | 2 | 2 | 1 | 1 | 2 | 8 |
| Opción 5 | 2 | 2 | 3 | 2 | 2 | 11 |
| Opción 6 | 2 | 1 | 3 | 3 | 1 | 10 |

Así pues, utilizaremos los métodos de Bairstows y el de la matriz compañera para realizar el cálculo de las raíces de un polinomio.

Diseño de casos de prueba:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Clase | Método | Escenario | Valores de entrada | Resultado |
| Polynomial | solveBairstow | Crear un polinomio cuyos coeficientes sean:  2,-2,2,-2,2,-2  Donde se el grado del mayor exponente empieza en la izquierda. | Ninguno. | \* El conjugado  0.5 + 0.866i  \* El conjugado  -0.5 + 0.866i  \* El entero 1 |
| Polynomial | solveBairstow | Crear un polinomio cuyos coeficientes sean:  0,2,0,7,0  Donde se el grado del mayor exponente empieza en la izquierda. | Ninguno. | Lanza la excepción InvalidFormatException.java |
| Polynomial | solveBairstow | Crear un polinomio cuyos coeficientes sean:  0.5,-5,0.2,-2  Donde se el grado del mayor exponente empieza en la izquierda. | Ninguno. | \*El entero 10  \*El conjugado  1.658\*10^-18 + 0.632i |
| Polynomial | solveBairstow | Crear un polinomio cuyos coeficientes sean:  4,””,3  Donde se el grado del mayor exponente empieza en la izquierda. | Ninguno. | Lanza la excepción InvalidFormatException.java |
| Polynomial | findRoots | Crear un polinomio cuyos coeficientes sean:  2,-2,2,-2,0,0  Donde se el grado del mayor exponente empieza en la izquierda. | Ninguno. | \*El entero con multiplicidad 2:  0  \* El conjugado  4.73\*10^-16 + 1i  \* El entero 1 |
| Polynomial | findRoots | Crear un polinomio cuyos coeficientes sean:  0,2,0,7,0  Donde se el grado del mayor exponente empieza en la izquierda. | Ninguno. | Lanza la excepción InvalidFormatException.java |
| Polynomial | findRoots | Crear un polinomio cuyos coeficientes sean:  1,-1.5,-5.5,3,0  Donde se el grado del mayor exponente empieza en la izquierda. | Ninguno. | \*Los Enteros:  3  -2  0.5  0 |
| Polynomial | findRoots | Crear un polinomio cuyos coeficientes sean:  4,””,3  Donde se el grado del mayor exponente empieza en la izquierda. | Ninguno. | Lanza la excepción InvalidFormatException.java |

Bibliografía:

<http://tutorial.math.lamar.edu/Classes/Alg/ZeroesOfPolynomials.aspx>

<https://www.geeksforgeeks.org/program-muller-method/>

https://youtu.be/dmsosSGVxgs

<https://prezi.com/kdjcizosymv7/metodo-de-muller/>

<https://www.geeksforgeeks.org/program-muller-method/>

<https://github.com/ergenekonyigit/Numerical-Analysis-Examples>

<https://www.youtube.com/watch?v=dmsosSGVxgs&feature=youtu.be>

<https://es.wikipedia.org/wiki/Descomposici%C3%B3n_en_valores_singulares>

<http://mathworld.wolfram.com/CompanionMatrix.html>

<https://github.com/lessthanoptimal/ejml/blob/v0.31/examples/src/org/ejml/example/PolynomialRootFinder.java>

<https://ejml.org/wiki/index.php?title=Example_Polynomial_Roots>

<https://github.com/lessthanoptimal/ejml>