MÉTODO DE LA INGENIERÍA

Fase 1:Identificación del problema

Necesidades y síntomas:

- Los ingenieros de la empresa ORACLE requieren resolver polinomios de grado 10.
- No está implementado entre el grupo de ingenieros un algoritmo que resuelva los polinomios de grado 10.
- La solución a este problema de garantizar una eficiencia sobresaliente.
- La solución del problema deberá ser entregado con eficacia y interactiva para que los usuarios puedan usarlo con facilidad y sin inconvenientes.

Definición del problema:

- La empresa ORACLE requiere de un algoritmo que permita encontrar las raíces de un polinomio de hasta grado 10.

Requerimientos Funcionales :

- R1 Ingresar Polinomio al sistema:
- R1.1 ingresar coeficientes: el programa está en capacidad de recibir cualquier tipo de dato numérico sea positivo o negativo en la interfaz. Como entradas se necesita un dato numérico que el usuario ingresará al sistema. como resultado el sistema tomará los datos como parámetros para hacer las operaciones del polinomio.

- R1.2 Seleccionar grado de exponente: el sistema facilita la posibilidad de elegir hasta exponente 10 las variables que el usuario elija poner en pantalla. Como entradas no habrán y como resultado el sistema mostrará el polinomio construido por el usuario

- R2 Generar Polinomio Aleatorio: el sistema está en capacidad de generar un polinomio aleatoriamente y mostrarlo en pantalla.

- R3 Encontrar Raíces del Polinomio: el sistema está en capacidad de resolver polinomios de hasta grado 10 y mostrar sus raíces en pantalla.

- R4 Mostrar Método: El sistema muestra el método usado en pantalla después de resolver un polinomio.

Fase 2: Recopilación de información:

Conceptos básicos para poder entender una solución posible del problema.

- **Los polinomios:** están constituidos por un conjunto finito de variables (llamadas incógnitas) y constantes (llamadas coeficientes), con las operaciones aritméticas de suma, resta y multiplicación, así como también exponentes enteros positivos. Pueden ser de una o de varias variables.

- **Grado de un polinomio:**Se define el grado de un monomio como el exponente de su variable. El grado de un polinomio es el del monomio de mayor grado, y se denota por Gra(p)

- **Raíz de un Polinomio:** Las raíces de un polinomio son números tales que hacen que un polinomio valga cero. Podemos decir también que las raíces enteras de un polinomio de coeficientes enteros serán divisores del término independiente.

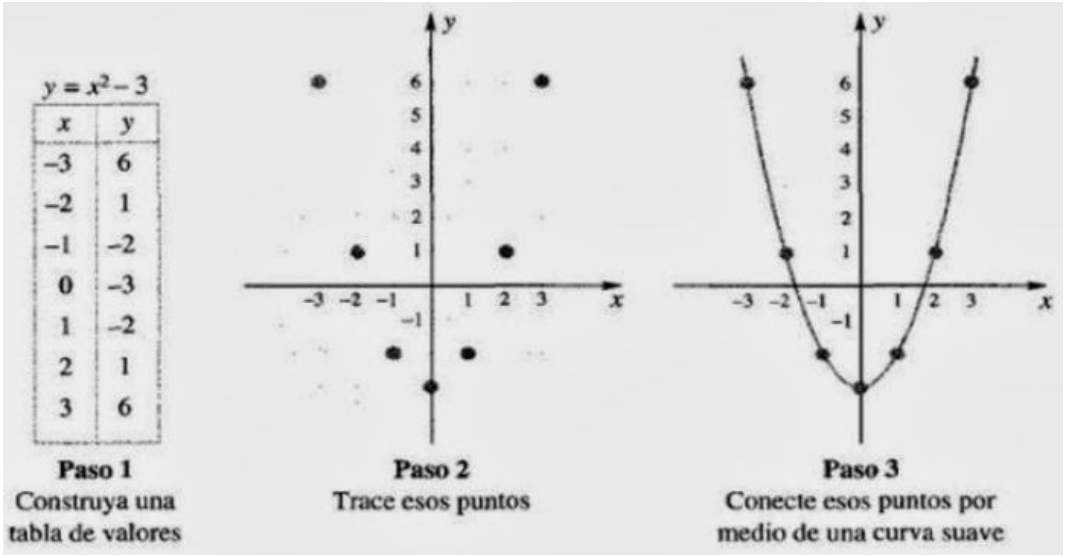
- **coeficiente:** es el número antes de la variable elevada a un exponente

Fuente: https://definicion.de/

Fase 3:Búsqueda de soluciones creativas.

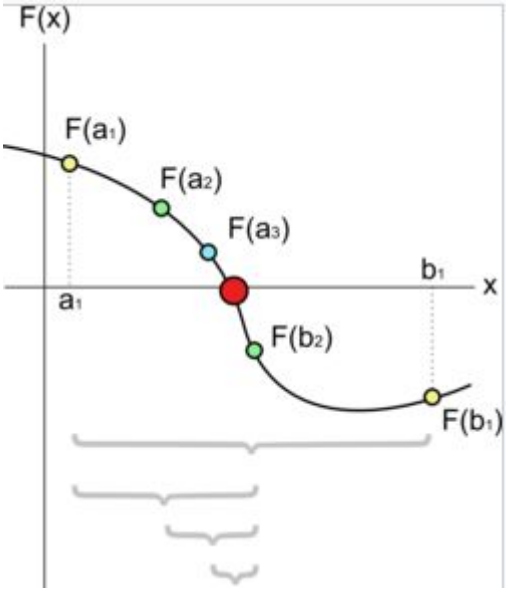
Alternativa 1 Graficar la función.

Una forma de encontrar las raíces de un polinomio es gráficamente, realizando la tabulación y encontrando los valores que corten con el eje x.



Alternativa 2. Bisección

Este método se basa en el teorema del valor medio (Bolzano): Si f es continua en el intervalo [a,b], tal que f(a)f(b) tienen signos opuestos (es decir f(a)f(b)< 0) entonces existe (al menos) un número p pertenece a [a,b] tal que f (p)=0 El método de bisección requiere dividir varias veces a la mitad los subintervalos de [a b,] y, en cada paso, localizar la mitad que contenga p. Para simplificar supongamos que existe una única raíz en [a b] , y construyamos las siguientes sucesiones:{an},{ bn} {pn }, con , a1=a , b1=b,p1=a1+b1/2 Si f (p1) =0 tome p=p1 . Caso contrario f(p1) tiene el mismo signo de f(a1) o de f(b1). Si f (p1)f(a1)<0 entonces p pertenece [a1,p1] En este caso tomamos a2=a1,b2=p1 .Repetimos el procedimiento en el intervalo [a2,b2]. Caso contrario,p pertenece [p1,b1] y a2=p1, b2=b1 y repetimos el procedimiento en el intervalo [a2.b2].



Alternativa 3. Newton Raphson

Este método de resolución numérica busca un cero de la función **f(x)** por aproximaciones sucesivas a partir de un valor inicial x_0. El valor sucesivo x_{n+1} es la abscisa del punto en que la tangente a la gráfica de **f(x)** en x_n corta al eje **Ox**.

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Alternativa 4. Método de la secante

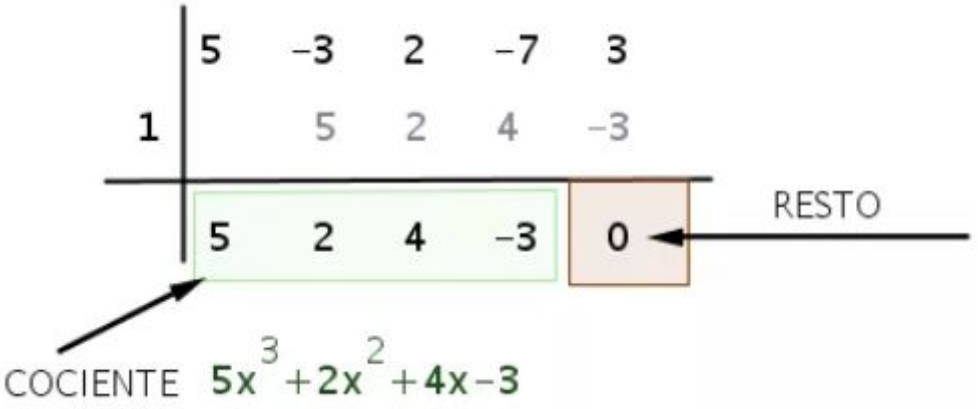
El principal inconveniente del método de Newton estriba en que requiere conocer el valor de la primera derivada de la función en el punto. Sin embargo, la forma funcional de *f*(*x*) dificulta en ocasiones el cálculo de la derivada. En estos casos es más útil emplear el método de la secante.

El método de la secante parte de dos puntos (y no sólo uno como el método de Newton) y estima la tangente (es decir, la pendiente de la recta) por una aproximación de acuerdo con la expresión:

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

Alternativa 5. Método de Ruffini

El método o regla de Ruffini es un método que nos permite dividir un polinomio entre un binomio y además permite localizar las raíces de un polinomio para factorizarlo en binomios. En otras palabras esta técnica **posibilita dividir o descomponer un polinomio algebraico de grado n, en un binomio algebraico, y luego en otro polinomio algebraico de grado n-1.** Y para que esto sea posible se necesita saber o conocer por lo menos una de las raíces del polinomio único, con el propósito de que la separación sea exacta.



Alternativa 6. Método de Graeffe

Este método fue descubierto independientemente por Graeffe (o Gräffe), Dandelin, y Lobachevsky. La idea general es, tomar un polinomio $P_1(x)P1(x)$ y crearnos un polinomio $P_2(x)P2(x)$ cuyas raíces sean los cuadrados de las raíces de $P_1(x)P1(x)$. Es decir, si las raíces de $P_1(x)P1(x)$ son $r_1,r_2,\cdots,r_nr1,r2,\cdots,rn$ entonces las raíces de $P_2(x)P2(x)$ serán $r_{21},r_{22},\cdots,r_{2n}r12,r22,\cdots,rn2$. Repetimos el procedimiento numerosas veces, así llegamos a un polinomio $P_m(x)Pm(x)$ cuyas raíces son $r_{2^m1},r_{2^m2},\cdots,r_{2^mn}r12m,r22m,\cdots,rn2m$. Este procedimiento hace que las raíces que antes estaban juntas, se separen muchísimo, y nos permite usar las fórmulas de Vieta (o relaciones entre coeficientes y raíces) para aproximarlas.

Fase 4:Transición de las Ideas a los Diseños Preliminares.

En este paso descartamos las ideas que no nos parecían apropiadas para resolver este problema, iniciamos descartando la idea de Graficar la función punto por punto debido a que era necesario conocer un intervalo apropiado para empezar graficar y conocer los ceros de la función, además no solucionaría el problema de encontrar las raíces complejas

El método de la bisección, Newton Raphson y el método de la secante fueron descartados debido a que solo nos daban una solución implementando todas las iteraciones , y ya que el cliente nos pide hallar todas las raíces de un polinomio n, los descartamos. Además estos métodos nos piden encontrar un intervalo cercano a la raíz.

La revisión de todas las alternativas nos conduce a lo siguiente:

Alternativa 1: Método de Ruffini.

- El método solo da las soluciones reales del polinomio
- El método reduce los polinomios hasta el punto de usar la ecuación cuadrática

Alternativa 2: Método de Graeffe

- El método se complica cuando las raíces del polinomio original tienen magnitud parecida.

Fase 5:Evaluación y selección de la mejor solución.

Criterios.

- Criterio A. Precisión de la Solución. La alternativa entrega una solución:
    - [2] Exacta
    - [1] Aproximada
- Criterio B Eficiencia. Se prefiere una solución con mayor eficiencia.
    - [4] Constante
    - [3] Mayor a constante
    - [2] Logaritmica
    - [1]Lineal
- Criterio C Completitud. Se prefiere una solución que encuentre todas las soluciones tanto reales como imaginarias
    - [3]Todas
    - [2] más de una si las hay.
    - [1]Sólo una o ninguna

- Criterio D. Facilidad en implementación algorítmica:
- [2] Compatible con las operaciones aritméticas básicas de un equipo de cómputo moderno

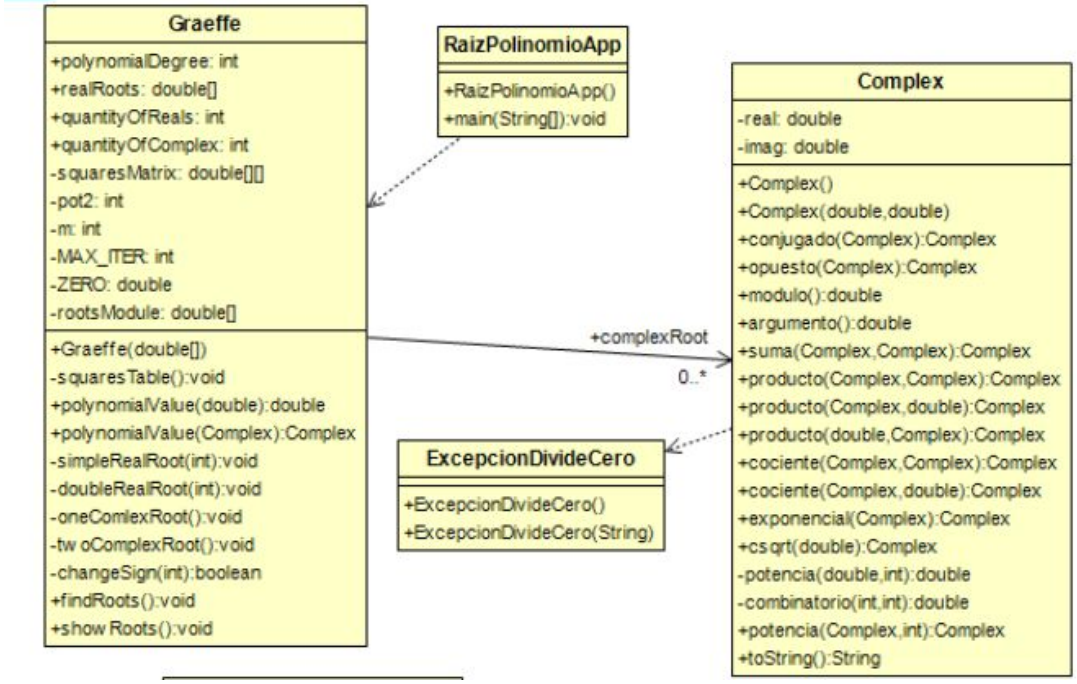
- [1] No compatible completamente con las operaciones aritméticas básicas de un equipo de cómputo moderno

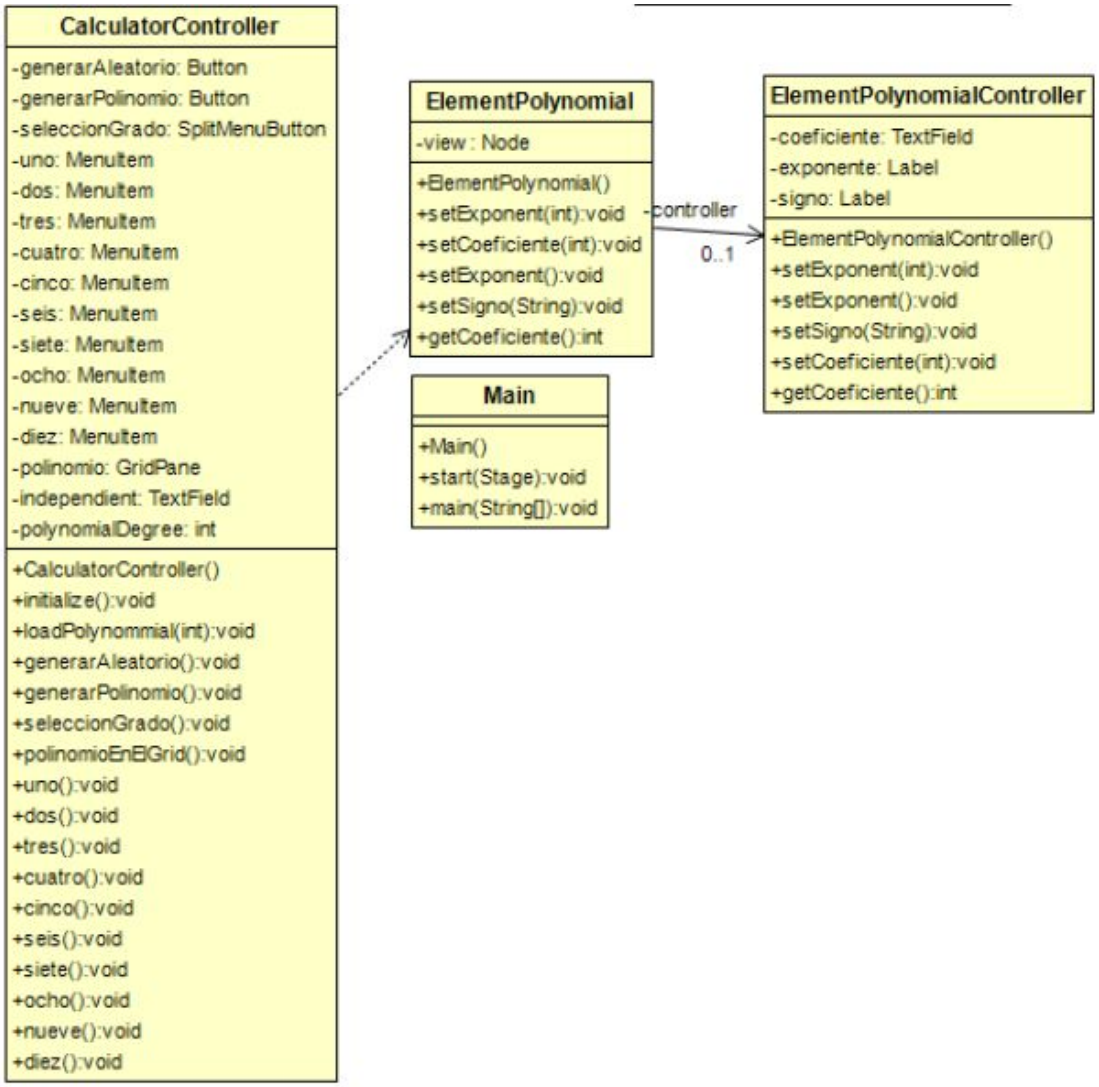
MÉTODO	CRITERIO A	CRITERIO B	CRITERIO C	CRITERIO D	TOTAL
RUFFINI	2	3	2	2	9
GRAEFFE	2	3	3	2	10

De acuerdo a la evaluación de criterios se debería de elegir la alternativa de Graeffe pues obtuvo mayor educación.

Fase 6 :

Diagrama UML





Especificación del problema: Raíces de un polinomio de grado n.
Entradas: coeficiente del P(n)= ax^n+bx^n-1+....+ c, Solo es necesario conocer los valores a b c…..n
Salida: Rn, R1, R2…. Rn tal que P(R1),P(R2)….P(Rn) igual a 0 R1 R2…. Rn pueden ser números reales o complejos

Consideraciones : se deben tener en cuenta los siguientes casos para calcular el valor de las raíces

1) El polinomio tiene todas sus raíces imaginarias.
2) El polinomio tiene todas las raíces reales.
3) El polinomio tiene al menos una raíz real y las demás imaginarias.
4) El polinomio tiene al menos una raíz imaginaria y las demás reales.s

Complejidad espacial
public Graeffe(double[] coef)

Tipo	Nombre	Tamaño

Entrada	coef	**n**
Auxiliar	realRoots	**n**
Auxiliar	squareMatrix	**nxm**
Auxiliar	polynomialDegree	**1**
Auxiliar	MAX_ITER	**1**
Auxiliar	j	**1**
Auxiliar	m	**1**
Auxiliar	quantityOfReals	**1**
Auxiliar	quantityOfComplex	**1**
		=n(2+m)+6

complejidad espacial = O(n,m)

private void squaresTable()

Tipo	**Nombre**	**Tamaño**
Auxiliar	k	1
Auxiliar	l	1
Auxiliar	signo	1
Auxiliar	i	1
Auxiliar	squareMatrix	nxm
Auxiliar	m	1
Auxiliar	polynomialDegree	1
Auxiliar	i	1
Auxiliar	i	1
Auxiliar	s	1
Auxiliar	MAX_ITER	1
Auxiliar	post	1
Auxiliar	i	1
		=nxm+11

complejidad espacial = O(n,m)

COMPLEJIDAD TEMPORAL

Metodo graeffe()

algoritmo	**Complejidad**
polynomialDegree=coef.length-1;	1
realRoots=new double[polynomialDegree];	m
squaresMatrix= new double[MAX_ITER][polynomialDegree+1]	nxm

for(int j=0; j<polynomialDegree+1; j++){
squaresMatrix[0][j]=coef[j];
}
	m+2
	m

for(int m=1; m<MAX_ITER; m++){
for(int j=0; j<polynomialDegree+1; j++){
squaresMatrix[m][j]=0.0;
}
}
	n+1
	n
	n-1

quantityOfReals=0; quantityOfComplex=0;	1 1

- T(n,m)=1+m+nm+m+2+n+1+n+n-1+1+1
- T(n,m)=m(2+n)+3n+4
- Complejidad O(n,m)

Método squaresTable()

int k,l, signo;	3
for(int i=1; i<polynomialDegree+1; i++){ squaresMatrix[0][i]/=squaresMatrix[0][0];	n+1 n

} 	
squaresMatrix[0][0]=1.0;	1

do{
 	
for(int i=0; i<polynomialDegree+1; i++){
 	
squaresMatrix[m][i]=squaresMatrix[m-1][i]*squaresMatrix[m-1][i];
if(Double.isInfinite(squaresMatrix[m][i])){
break pajuera;
}
}
//dobles productos
for(int i=1; i<polynomialDegree; i++){
for(int s=1; s<polynomialDegree/2+1; s++){
k=i-s; l=i+s;
if((k<0)||(l>polynomialDegree)) {
break;
}
signo=(s%2==0)? +1: -1;
 	
squaresMatrix[m][i]+=signo*2*squaresMatrix[m-1][k]*squaresMatrix[m-1][l];
if(Double.isInfinite(squaresMatrix[m][i])){
break pajuera;
}
}
}
m++;
}while(m<MAX_ITER);
	
	n+1
	
	n
	n
	
	n+1
	(n+1)((n/2)+1) Techo
	(n+1)(n/2)
	(n+1)(n/2)
	(n+1)(n/2)
	
	(n+1)(n/2)
	
	(n+1)(n/2)
	(n+1)(n/2)
	
	n
	n+1

m--;
pot2=1;
for(int i=1; i<=m; i++){
pot2*=2;
}
}
	1
	1
	n+1
	n

Metodo PolynomialValue()

double y=0.0;	1
double[] pot_x=new double[polynomialDegree+1];	1
pot_x[0]=1.0	1

for(int i=1; i<polynomialDegree+1; i++){
pot_x[i]=pot_x[i-1]*x;
}
	n+1
	n

for(int i=0; i<polynomialDegree+1; i++){
y+=squaresMatrix[0][i]*pot_x[polynomialDegree-i];
}
	n+1
	n

return y;	1

Diseño de pruebas unitarias automáticas.

P1. Se espera que al pasar el parámetro coef, al mostrar la primera fila de la matriz squareMatrix sea igual al arreglo de doubles.

Clase	**Método**	**Escenario**	**Valores Entrada**	**Resultado**
Graeffe	**Graeffe(double [] coef)**	coef={10,9,7,6, 5,1,8}	coef[]	**la primera fila de la matriz es igual al arreglo**
	Graeffe(double [] coef)	coef={10,9,7,6, 5,1,8}	coef[]	**los valores de la matriz square Matrix son cero desde la segunda fila**
	revisarCoordenada(String coordenada)	coordenada=1-1 valido=true dificultad=FACIL(Se espera que lance la excepción cuando válido es falso)	String coordenada	**La coordenada lleva a una casilla destapada**

Links para usar: https://www.youtube.com/watch?v=QCf57lw_GT0
https://www.youtube.com/watch?v=YOHtlzPmfzE
https://www.youtube.com/watch?v=SPSf8tGe2Oc
https://www.youtube.com/watch?v=Sv6l3jB30yg
http://galia.fc.uaslp.mx/~medellin/classesPN.htm

michael
https://juncotic.com/metodo-de-horner-algoritmos-antiguos/
https://www.youtube.com/watch?v=klo77kr_VGg&t=129s
https://www.youtube.com/watch?v=klo77kr_VGg&t=129s

para la fase 6:

https://mynmblog.wordpress.com/tag/raices-de-polinomios/