# Método de la Ingeniería

**Aplicación para la Solución de un Problema**

**Miguel Angel Romero Rosas y Jonathan Arenas**

# Contexto Problemático

Una empresa de fabricación de microprocesadores está evaluando la posibilidad de implementar varios algoritmos de ordenamiento como instrucciones básicas de su próximo coprocesador matemático. la empresa ha decidido implementar tres (3) algoritmos diferentes de ordenamiento que permitan ordenar, muy rápidamente, números enteros de tamaño arbitrariamente grande y números en formato de coma flotante de cualquier tamaño.

# Desarrollo de la Solución

# Paso 1. Identificación del Problema

## Identificación de necesidades y síntomas

* Una empresa de fabricación de microprocesadores desea implementar tres algoritmos de ordenamientos como una operación nativa de su próximo coprocesador.
* La solución al problema debe garantizar un ordenamiento muy rápidamente de números enteros de tamaño arbitrariamente grande y números en formato de coma flotante de cualquier tamaño.
* La solución del problema debe de ser eficiente ya que esto evitaría que el procesador principal tenga que realizar estas tareas de cómputo intensivo.
* El coprocesador matemático que se quiere implementar debe acelerar el rendimiento del Sistema por el hecho de esta descarga de trabajo en el procesador principal.
* Una empresa de fabricación de microprocesadores desea implementar tres algoritmos de ordenamientos como una operación nativa de su próximo coprocesador.
* Sus anteriores líneas de coprocesadores no implementaban o no soportaban varios algoritmos de ordenamiento.
* los algoritmos deben permitir ordenar muy rápidamente.
* La empresa desea un software para probar los algoritmos qué serán implementados en el hardware.
* No disponen de un software que permita evaluar la eficiencia y eficacia de los algoritmos en el hardware
* La empresa desea poder ingresar los valores que desean ordenar.
* La empresa desea que el programa genere aleatoriamente los valores.
* La empresa desea tener control sobre la cantidad total de valores generados y el intervalo en el cual se generarán los mismos.
* La empresa, además, requiere poder elegir entre números repetidos o ausencia de estos en los números generados.
* La generación aleatoria debe permitir elegir entre unas configuraciones de los valores en la secuencia.
* La empresa requiere que el software ordene los valores ingresados utilizando el algoritmo apropiado de acuerdo con el tipo de número a ordenar y el intervalo de números generados.
* Finalmente, La empresa requiere que la interfaz gráfica muestre el tiempo que se demoró el algoritmo en ordenar las entradas.

## Definición del Problema

Una empresa que fabrica microprocesadores desea incluir en su próximo coprocesador la funcionalidad de ordenar entradas numéricas grandes, ya sean enteros o de coma flotante. Para esto, requieren incluir tres algoritmos de ordenamiento en el hardware que tengan un buen rendimiento dependiendo el tipo de número y la magnitud de números de la entrada. Además, Desean un software que permita evaluar estos algoritmos en el hardware. Así, la empresa requiere que el software lea y genere entradas numéricas enteras y de coma flotante. Para el caso de la generación, el software, precisa brindar control sobre el número total de datos, el intervalo de generación, existencia de números repetidos y elección entre posibles configuraciones de los valores en la secuencia. Finalmente, ordenar la serie de datos y mostrarlos junto al tiempo empleado.

## **Paso 2. Requerimientos Funcionales:**

* La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda ingresar los valores que se desean ordenar.
* La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda configurar el tipo de dato a generar.
* La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda configurar la cantidad total de números a generar.
* La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda configurar el intervalo en el cual se generarán los números.
* La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda decidir la ausencia o existencia de números repetidos.
* La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda elegir entre las siguientes configuraciones de los valores en la secuencia: (a)valores ya ordenados (b)valores ordenados inversamente (c)valores en orden completamente aleatorio (d)valores desordenados en un % dado por el usuario.
* La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda generar aleatoriamente valores.
* la interfaz gráfica debe dar al modelo del mundo los valores que se desean generar junto con la configuración deseada para la serie de datos de salida.
* El modelo del mundo debe generar la serie de datos con las especificaciones brindadas por la interfaz del usuario.
* el modelo del mundo debe brindar a la interfaz del usuario la serie de datos generados.
* La interfaz gráfica debe ofrecer una opción mediante la cual el usuario pueda ordenar los valores.
* La interfaz gráfica debe dar al modelo del mundo los valores que se desean ordenar.
* El modelo del mundo debe brindar al programa 3 algoritmos de ordenamiento para entradas grandes enteras y de coma flotante.
* El modelo del mundo debe elegir un algoritmo mediante el cual, dependiendo de la entrada, se llevará a cabo la operación de ordenar.
* El modelo del mundo debe brindar a la interfaz gráfica los valores de la secuencia.
* El modelo del mundo debe brindar a la interfaz gráfica el tiempo empleado en el proceso de ordenamiento.
* La interfaz gráfica debe mostrar al usuario la serie de datos ordenados.
* La interfaz gráfica debe mostrar al usuario el tiempo empleado en el proceso de ordenar.

## 

# Paso 3. Recopilación de Información

## Definiciones

**Números en formato coma flotante:**

es una forma de notación científica usada en los microprocesadores con la cual se pueden representar números racionales extremadamente grandes y pequeños de una manera muy eficiente y compacta, y con la que se pueden realizar operaciones aritméticas.

Sin embargo, hay que tener en cuenta que no se puede evaluar si un coma flotante es mayor, o menor, que otro de la misma forma que se haría con un número entero. Pues, podemos tener A= 1,0 y B=1,0 pero puede que B= 1,00000001.

Cada dato float cuesta 4 bytes.

También, tienen un estándar que convierte los datos de coma flotante en bytes, se llama IEEE 754.

Para comprobar números flotantes no se evalúa si son iguales. Por el contrario, se busca evaluar si su diferencia es muy pequeña. El margen de error que compara esta diferencia normalmente se llama épsilon. En su forma más simple:

if(Math.abs(a-b) < 0.00001)

Pero como épsilon puede ser, o muy grande para floats pequeños o muy pequeño para floats muy grandes, es necesario ver si el error relativo es menor que épsilon:

if(Math.abs((a-b)/b) < 0.00001)

Hay casos especiales donde esto falla:

* Cuando tanto a como b son cero. 0.0/0.0 es NaN, lo que provoca una excepción en algunas plataformas o devuelve falso para todas las comparaciones.
* Cuando solo b es cero, la división devuelve infinito, lo que también puede causar una excepción, o es mayor que épsilon incluso cuando a es más pequeño.
* Devuelve «falso» cuando tanto a como b son muy pequeños, pero a ambos lados del cero, incluso cuando son los números no nulos más pequeños.
* Además, el resultado no es conmutativo (nearlyEquals(a,b) no es siempre lo mismo que nearlyEquals(b,a)). Para solucionar estos problemas, el código tiene que ser mucho más complejo, así que necesitamos meterlo en una función:
* public static boolean nearlyEqual(float a, float b, float epsilon)
* {
* final float absA = Math.abs(a);
* final float absB = Math.abs(b);
* final float diff = Math.abs(a - b);
* if (a == b) { // Atajo, maneja los infinitos
* return true;
* } else if (a \* b == 0) { // a o b o ambos son cero
* // El error relativo no es importante aquí
* return diff < (epsilon \* epsilon);
* } else { // Usar el error relativo
* return diff / (absA + absB) < epsilon;
* }
* }

Este método [pasa las pruebas](http://puntoflotante.org/errors/NearlyEqualsTest.java) para muchos casos especiales importantes, pero como puedes ver, utiliza cierta lógica no trivial. En particular, tiene que utilizar una definición totalmente distinta del margen de error cuando a o b son cero, porque la definición clásica del error relativo es inútil en esos casos.

Hay algunos casos en los que el método de arriba todavía produce resultados inesperados (concretamente, es mucho más estricto cuando un valor es casi cero que cuando es exactamente cero), y algunas de esas pruebas para las que fue desarrollado probablemente especifica un comportamiento que no es apropiado para algunas aplicaciones.

## Comparando valores de punto flotante como enteros

Hay una alternativa a aplicar toda esta complejidad conceptual a una tarea aparentemente tan sencilla: en lugar de comparar a y b como [números reales](http://es.wikipedia.org/wiki/N%C3%BAmero_real), podemos concebirlos como pasos discretos y definir el margen de error como el número máximo de valores de punto flotante posibles entre esos dos números.

Esto es conceptualmente muy evidente y fácil y tiene la ventaja de que escala implícitamente el margen de error relativo con la magnitud de los valores. Técnicamente es un poco más complejo, pero no mucho más de lo que puedas pensar, porque los números de punto flotante del IEEE 754 están diseñados para mantener su orden cuando sus secuencias de bits se interpretan como enteros.

Sin embargo, este método requiere que el lenguaje de programación soporte conversión entre valores de punto flotante y secuencias de bits enteras.

http://puntoflotante.org/errors/comparison/

**Coprocesador:** es un microprocesador de un ordenador utilizado como suplemento de las funciones del procesador principal (la CPU). Las operaciones ejecutadas por uno de estos coprocesadores pueden ser operaciones de aritmética en coma flotante, procesamiento gráfico, procesamiento de señales, procesado de texto, criptografía, etc.

# Operaciones con aritmética:

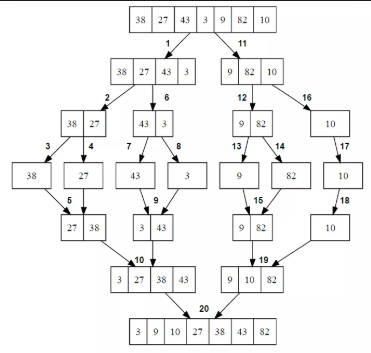
son: suma, resta, multiplicación, división, potenciación, división entera. Los operadores lógicos son: "y", "o", "no" y "o exclusivo".

**Numero entero:**

es un elemento del conjunto numérico que contiene los números naturales N ={1,2,3,4….}, sus opuestos y el cero.1​ Los enteros negativos, como −1 o −3 (se leen «menos uno», «menos tres», etc.), son menores que cero y todos los enteros positivos.

Alternativas:

**Alternativa 1: Método Merge Sort:**

 El algoritmo de ordenación por combinación, Merge Sort, se basa en la técnica Divide y Vencerás, ordena recursivamente un conjunto de elementos dividiéndolo en dos, ordenando cada una de estas partes en forma independiente y combinando los dos resultados. Este a su vez **recibe como entrada un arreglo de números enteros** denominado v, lo parte utilizando el **método copyOfRange** de la clase en java, se llama recursivamente con cada una de las dos partes como argumento y, una vez terminada la ordenación de dichas partes, invoca al proceso de combinación de las dos respuestas implementado en el método combinar, el cual recibe como entrada el arreglo original y las dos mitades de este previamente ordenadas que serán combinadas en el arreglo original. Este método es muy eficiente, comparado con otros métodos de ordenación, **pero solo es para números enteros.**

**Alternativa 2: Método Quick Sort:**

A close up of a clock

Description generated with low confidence

El objetivo de este algoritmo es dividir recursivamente el vector en partes iguales, indicando un elemento de inicio, fin y un pivote, que nos permitirá segmentar nuestra lista. Una vez dividida, lo que hace, es dejar todos los mayores que el pivote a su derecha y todos los menores a su izq. Al finalizar el algoritmo, nuestros elementos están ordenados. Esto lo que nos va a permitir en el programa es ordenar de una manera más eficiente y eficaz los números que lleguen como entrada al algoritmo.

**Alternativa 3:**

# Alternativa 2:

Otra opción es usar Math.floor() para obtener la parte entera del numero y restársela al numero y si es 0 pues entonces el número es entero y sino pues obviamente no.

function esEntero(numero){

if (numero - Math.floor(numero) == 0) {

alert ("Es un numero entero");

} else {

alert ("Es un numero decimal");

}

}

**Alternativa 3:**

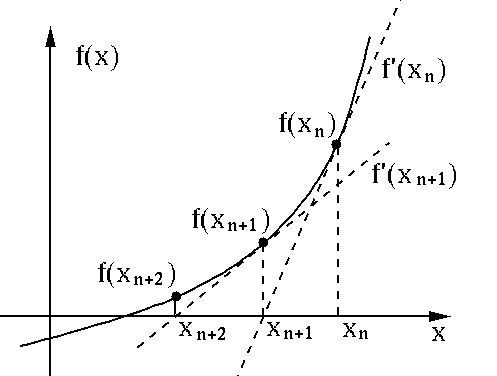
**Paso 4. Transición de las Ideas a los Diseños Preliminares** :

Fuente:

# https://es.wikipedia.org

Para este paso, aunque podemos pensar en soluciones propias, y como resulta que el problema es uno clásico en las matemáticas, buscamos en textos especializados diversas estrategias a través de los cuales puedan ser encontradas las raíces de una ecuación cuadrática. Los métodos encontrados son los siguientes:

## Alternativa 1. Método de Newton­Raphson

Es un método general para encontrar aproximaciones de las raíces de una función real.

Sea *f* : [*a*,*b*] → *R* una función derivable definida en el intervalo real [a,b]. Empezamos con un valor inicial *x*0 y definimos para cada número natural n: *xn*+1 = *xn* − *ff*′((*xxnn*)) . Donde *f*′ denota la derivada de f con respecto a x.

2

## Alternativa 2. Normalización y Factorización

La ecuación cuadrática *ax*[[1]](#footnote-1) + *bx* + *c* = 0 puede ser normalizada, de tal manera que se reduzca a la ecuación *x*2 + *b*′*x* + *c*′ = 0 . Es decir, una ecuación cuadrática donde *a* = 1 : *b*′ = *ba* y *c*′ = *ac* .

Luego la expresión puede descompuesta en factores de la forma (*x* − *r*1) y (*x* − *r*2) tal que *ax*2 + *bx* + *c* = *x*2 + *b*′*x* + *c*′ = (*x* − *r*1)(*x* − *r*2) = 0 .

Esta factorización se consigue encontrando *r*1 + *r*2 = *b*′ al tiempo que *r*1 \* *r*2 = *c*′.

## Alternativa 3. Fórmula de la Ecuación Cuadrática

Utilizar la fórmula de la ecuación cuadrática: *r* = −*b*±√2*ba*2−4*ac*

**Paso 4. Transición de las Ideas a los Diseños Preliminares**

Lo​ primero que hacemos en este paso es descartar las ideas que no son factibles. En este sentido ​***descartamos******la***

***Alternativa 2 (Normalización y Factorización)*** debido a la dificultad que supone encontrar *r*1 y *r*2 que satisfagan *r*1 + *r*2 = *b*′ al tiempo que *r*1 \* *r*2 = *c*′, ya que resolver este par de ecuaciones de forma analítica nos pone un

problema similar al problema original de encontrar las raíces de una función cuadrática.

La revisión cuidadosa de las otras alternativas nos conduce a lo siguiente:

*Alternativa 1. Método de Newton­Raphson*.

­ Es un método abierto en el que no está garantizada su convergencia global.

­ El método puede no converger si la función presenta múltiples puntos de inflexión y pendientes pronunciadas cerca de la raíz. Sin embargo, no es el caso de la función cuadrática por lo que esta característica no sería una desventaja.

­ El método implica iterar desde un primer valor aproximado hasta llegar a un valor tan cercano como un valor de error mínimo indicado sea alcanzado.

­ El método requiere de un valor inicial que debe estar cerca de la raíz para aumentar la probabilidad y rapidez de convergencia.

## Alternativa 3. Fórmula de la Ecuación Cuadrática

­ No es posible calcular las raíces de cualquier función cuadrática directamente utilizando las operaciones aritméticas simples de un equipo de cómputo, porque el cálculo del discriminante *b*2 − 4*ac* puede llevar a valores negativos, lo cual produce números imaginarios al requerirse el cálculo de la raíz cuadrada del discriminante en la fórmula, en casos que éste sea menor a 0.

# Paso 5. Evaluación y Selección de la Mejor Solución

## Criterios

Deben definirse los criterios que permitirán evaluar las alternativas de solución y con base en este resultado elegir la solución que mejor satisface las necesidades del problema planteado. Los criterios que escogimos en este caso son los que enumeramos a continuación. Al lado de cada uno se ha establecido un valor numérico con el objetivo de establecer un peso que indique cuáles de los valores posibles de cada criterio tienen más peso (i.e., son más deseables).

­ *Criterio A.* Precisión de la solución. La alternativa entrega una solución:

­ [2] Exacta (se prefiere una solución exacta)

­ [1] Aproximada

­ *Criterio B.* Eficiencia. Se prefiere una solución con mejor eficiencia que las otras consideradas. La eficiencia puede ser:

­ [4] Constante

­ [3] Mayor a constante

­ [2] Logarítimca

­ [1] Líneal

­ *Criterio C.* Completitud. Se prefiere una solución que encuentre todas las soluciones. Cuántas soluciones entrega:

­ [3] Todas

­ [2] Mas de una si las hay, aunque no todas

­ [1] Solo 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

## Evaluación

Evaluando los criterios anteriores en las alternativas que se mantienen, obtenemos la siguiente tabla:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Criterio A** | **Criterio B** | **Criterio C** | **Criterio D** | **Total** |
| Alternativa 1. Método de  Newton­Raphson | Aproximada 1 | Mayor a constante 3 | Solo una 1 | Compatible 2 | 7 |
| Alternativa 2. Fórmula de la Ecuación Cuadrática | Exacta 2 | Constante 4 | Todas  3 | No compatible  (directamente) 1 | 10 |

## Selección

De acuerdo con la evaluación anterior se debe seleccionar la Alternativa 2, ya que obtuvo la mayor puntuación de acuerdo con los criterios definidos. Se debe tener en cuenta que hay que hacer un manejo adecuado del criterio en el cual la alternativa fue peor evaluada que la otra alternativa.

# Paso 6. Preparación de Informes y Especificaciones

*Especificación del Problema* (en términos de entrada/salida)

*Problema:* Raíces de una función cuadrática

*Entradas:* coeficientes de la función *f*(*x*) = *ax*2 + *bx* + *c* solo es necesario conocer los valores de *a* , *b* y *c*.

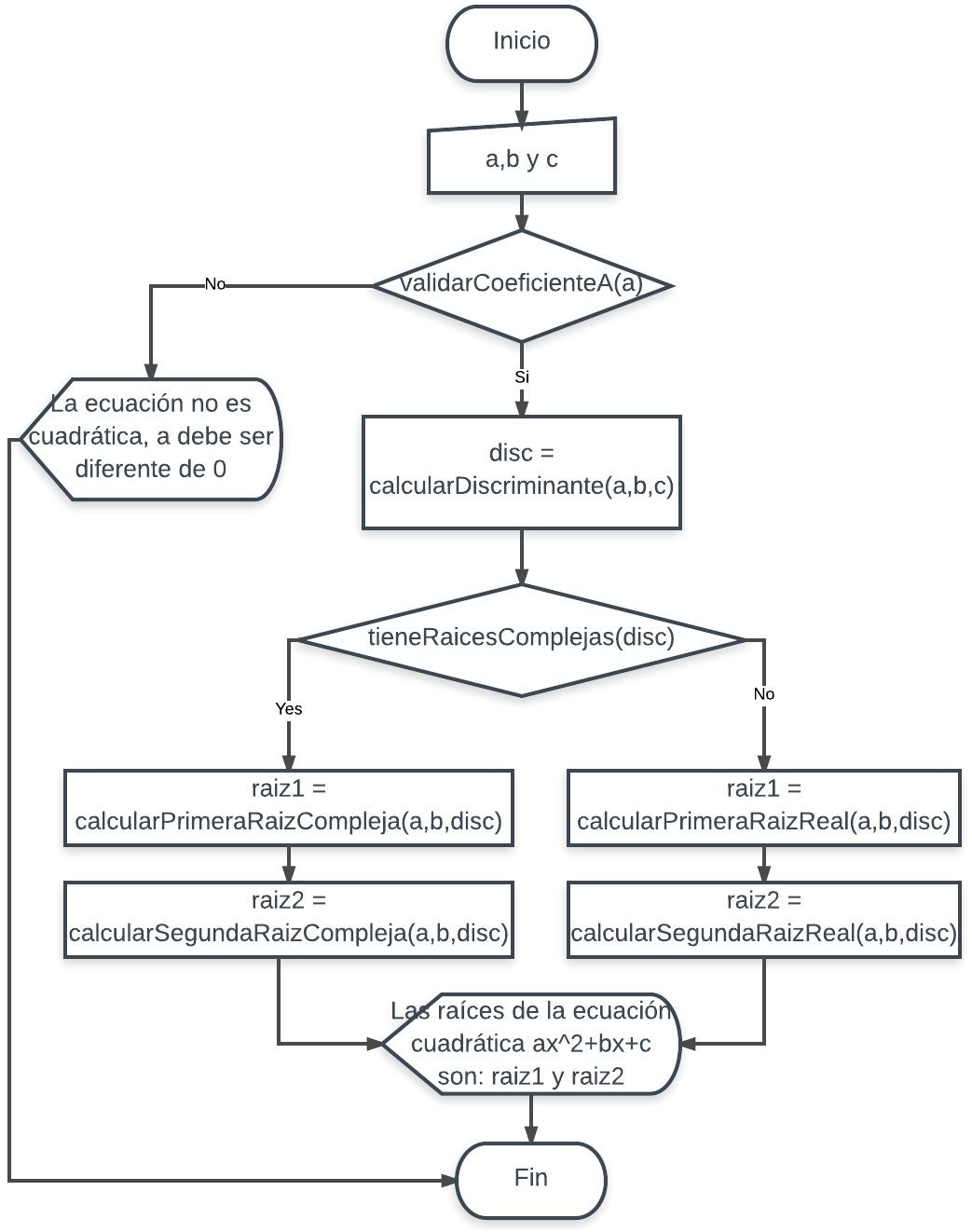
*Salida:* *r*1 y *r*2 tal que *ar*12 + *br*1 + *c* = 0 y *ar*22 + *br*2 + *c* = 0 . *r*1 y *r*2 pueden ser números reales o complejos.

## Consideraciones

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

1. La función tiene dos soluciones reales, que ocurre cuando la parábola descrita por la función cuadrática corta el eje X en dos puntos diferentes. En este caso el discriminante es positivo.
2. La función tiene una solución real, que ocurre cuando la parábola es tangente al eje X. Aquí el discriminante es cero.
3. No hay solución real. La parábola no corta el eje X. El discriminante es menor que cero, y por tanto la función puede tener una o dos soluciones complejas.

## Diagrama de Flujo del Algoritmo Pseudocódigo del Algoritmo



if(!validarCoeficienteA())​

imprimir “No es una ecuación cuadrática”; else disc ← calcularDiscriminante(a, b, c); if(tieneRaicesComplejas(disc)){ raiz1 ← calcularPrimeraRaizCompleja(a, b, disc); raiz2 ← calcularSegundaRaizCompleja(a, b, disc); else raiz1 ← ""+calcularPrimeraRaizReal(a, b, disc); raiz2 ← ""+calcularSegundaRaizReal(a, b, disc);

imprimir "\nLas raíces de la ecuación cuadrática:\n"

+ a +"x^2 + "+b+"x + "+c+" = 0, son:\n"

+ "Raíz 1: "+raiz1+"\n"

+ "Raíz 2: "+raiz2+"\n";

# Paso 7. Implementación del Diseño

Implementación en un Lenguaje de Programación.

Lista de Tareas a implementar:

a. ​ Validar los coeficientes​

b. ​ Calcular el discriminante​

c. ​ Verificar si hay raíces complejas​

d. ​ Calcular la primera raíz real​

e. ​ Calcular la segunda raíz real​

f. ​ Calcular la primera raíz compleja​

g. Calcular la segunda raíz compleja

# Construcción

|  |  |
| --- | --- |
| **Especificación de Subrutinas**  Validar los coeficientes | Escritura del código en un Lenguaje de Programación  ­Java en este caso­ |

|  |  |
| --- | --- |
| Nombre: | validarCoeficienteA |
| Descripción: | Valida que el coeficiente ​*a* sea diferente de 0 |
| Entrada: | ­ ​*a*: double, es el coeficiente de la incognita al cuadrado |
| Retorno: | boolean, true si el coeficiente es válido y false si no lo es |

public boolean validarCoeficienteA(double a){ if(a == 0){

return false;

}else{

return true;

}

}

Calcular el discriminante

|  |  |
| --- | --- |
| Nombre: | calcularDiscriminante |
| Descripción: | Calcula el discriminante de la ecuación cuadrática con base en los coeficientes |
| Entrada: | ­ a: double, coeficiente de x^2  ­ b: double, coeficiente de x  ­ c: double, valor constante |
| Retorno: | double, el discriminante ya  calculado |

public double calcularDiscriminante(double a,

double b, double c){

double disc;

disc = b\*b – 4\*a\*c;

return disc;

}

Verificar si hay raíces complejas

|  |  |
| --- | --- |
| Nombre: | sonRaicesComplejas |
| Descripción: | Verifica el discriminante para  determinar si las raíces son complejas o no |
| Entrada: | ­ discrim: double, es el discriminante de la ecuación |
| Retorno: | boolean, true si las raíces son complejas y false si son reales |

public boolean sonRaicesComplejas(

double discrim){

if(discrim<0){

return true;

}else{

return false;

} }

Calcular la primera raíz real

|  |  |
| --- | --- |
| Nombre: | calcularSolucionRealUno |
| Descripción: | Calcula la primera raíz real |
| Entrada: | ­ a: double, el coeficiente de la incognita al cuadrado  ­ b: double, el coeficiente de la incognita  ­ d: double, el discriminante |
| Retorno: | double, el valor de la primera raíz real |

public double calcularSolucionRealUno(double a

double b, double d){

double raizUno;

raizUno = (­b + Math.sqrt(d))/(2\*a);

return raizUno;

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Calcular la segunda raíz real   |  |  | | --- | --- | | Nombre: | calcularSolucionRealDos | | Descripción: | Calcula la segunda raíz real | | Entrada: | ­ a: double, el coeficiente de la incognita al cuadrado  ­ b: double, el coeficiente de la incognita  ­ d: double, el discriminante | | Retorno: | double, el valor de la segunda raíz real |   public double calcularSolucionRealDos(double a  double b, double d){  double raizDos;    raizDos = (­b ­ Math.sqrt(d))/(2\*a);    return raizDos;  }   |  |  | | --- | --- | |  |  | | Calcular la primera raíz compleja | public String calcularSolucionComlejaDos(double a |  |  |  | | --- | --- | | Nombre: | calcularSolucionComplejaUno | | Descripción: | Calcula la primera raíz compleja | | Entrada: | ­ a: double, el coeficiente de la incognita al cuadrado  ­ b: double, el coeficiente de la incognita  ­ d: double, el discriminante | | Retorno: | String, el número complejo que representa la primera solución compleja |   double b, double d){  String solComplejaDos;  double pReal, pImag;    pReal = ­b/(2\*a);  pImag = Math.sqrt(­d)/(2\*a);    solComplejaDos = pReal+” ­ ”+pImag+”i”;    return solComplejaDos; } |

}

Calcular la segunda raíz compleja

|  |  |
| --- | --- |
| Nombre: | calcularSolucionComplejaDos |
| Descripción: | Calcula la segunda raíz compleja |
| Entrada: | ­ a: double, el coeficiente de la incognita al cuadrado  ­ b: double, el coeficiente de la incognita  ­ d: double, el discriminante |
| Retorno: | String, el número complejo que representa la segunda solución compleja |

public String calcularSolucionComlejaUno(double a

double b, double d){

String solComplejaUno;

double pReal, pImag;

pReal = ­b/(2\*a);

pImag = Math.sqrt(­d)/(2\*a);

solComplejaUno = pReal+” + ”+pImag+”i”;

return solComplejaUno;

}

public String encontrarRaicesFuncionCuadratica(double a, double b, double c){ if(!validarCoeficienteA()){

return “No es una ecuación cuadrática”;

}

double disc;

String raiz1, raiz2;

//Se invoca el método calcularDiscriminante

disc = calcularDiscriminante(a, b, c);

//Se verifica cúal es la naturaleza de las raíces y se calculanCofe

if(tieneRaicesComplejas(disc)){

raiz1 = calcularPrimeraRaizCompleja(a, b, disc);

raiz2 = calcularSegundaRaizCompleja(a, b, disc);

}else{

raiz1 = ""+calcularPrimeraRaizReal(a, b, disc);

raiz2 = ""+calcularSegundaRaizReal(a, b, disc);

}

//Se retorna el resultado

return "\nLas raíces de la ecuación cuadrática:\n"

+ a +"x^2 + "+b+"x + "+c+" = 0, son:\n"

+ "Raíz 1: "+raiz1+"\n"

+ "Raíz 2: "+raiz2+"\n";

}

# Paso 3. Búsqueda de Soluciones Creativas

1. ​http://fourier.eng.hmc.edu/e176/lectures/NM/node20.html [↑](#footnote-ref-1)