Juan D. Castillo, Mateo Carvajal, Héctor C. Alzate

Sistema de resolución de ecuaciones cuadráticas con interceptos

Abstract
Algoritmo para encontrar la intersección de dos ecuaciones cuadráticas dentro de un rango definido usando Assembler para intel x86 embebido en C++, y graficador de ecuaciones usando Java.

I. Introducción

La práctica consistió en desarrollar un programa que permitiese desarrollar el siguiente procedimiento:

1. Ingresar parámetros para dos ecuaciones cuadráticas.
2. Ingresar un dominio en el cual hallar interceptos.
3. Analizar las ecuaciones y determinar en que puntos se encuentra una intercepción.
4. Graficar las ecuaciones ingresadas.

Este trabajo fue desarrollado utilizando Assembler embebido en c++, con las especificaciones del procesador intel x86, además del uso de Java2D para desarrollar las gráficas recibiendo los argumentos por línea de comandos.

II. Problema

A. Definición

“Una ecuación de segundo grado o ecuación cuadrática es una ecuación que tiene la forma de una suma de términos cuyo grado máximo es dos, es decir, una ecuación cuadrática puede ser representada por un polinomio de segundo grado o polinomio cuadrático. La expresión canónica general de una ecuación cuadrática es:

$ax2+bx+c=0$, para $a\neq0$

donde x representa la variable y a, b y c son constantes; a es un coeficiente cuadrático (distinto de 0), b el coeficiente lineal y c es el término independiente. Este polinomio se puede representar mediante una gráfica de una función cuadrática o parábola. Esta representación gráfica es útil, porque la intersección de esta gráfica con el eje horizontal coinciden con las soluciones de la ecuación (y dado que pueden existir dos, una o ninguna intersección, esos pueden ser los números de soluciones de la ecuación).”

El intercepto entre 2 ecuaciones se encontrará en el punto en el que para un mismo valor de x en el dominio ambas ecuaciones tengan el mismo valor de y.

B. Condiciones

- El número ingresado como “mayor” debe ser mayor que el “menor”.
- Los coeficientes “a” y “d” deben ser diferentes de 0.
- El punto de intercepción debe estar en un x, entero.

III. Algoritmo

A. Solución

Nuestro programa realiza el siguiente proceso para solucionar el problema:

1. Se leen los parámetros de ambas ecuaciones.
2. Se leen los parámetros del intervalo.
3. Se itera sobre valores enteros del intervalo y se comparan los valores de y encontrados.
4. En caso de que sean iguales se imprime el punto de la intersección.
5. Cuando se termina de recorrer el intervalo, se llama a un programa gráficador haciendo una simulación a un llamado por línea de comandos.
6. Se pasan los puntos de ambas ecuaciones.

B. Pseudocódigo

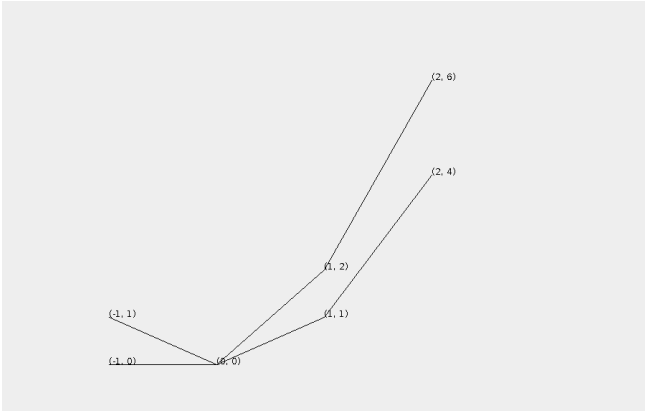
```
Begin
   read a, b, c;//eq1
   read d, e, f; //eq2
   read m, n;   //range, r E Z

   cont := 0;   //interceptions

   for (x:=m; x<=n; x++) do
      y1 := a*x*x + b*x + c;
      y2 := d*x*x + e*x + f;

      if (y1 = y2) then
         cont++;
         print(x, y1);
         draw(x, y1, true);    //different color
      else
         draw(x, y1, false);
         draw(x, y2, false);
      endif
   endfor

   if (cont = 0) then
      print("No interceptions found");
   endif
End
```

IV. Pruebas

Se desarrolló un programa en Java que recibe los mismos parámetros utilizados por el otro programa por la línea de comandos, para poder comparar los resultados y las gráficas.

V. Gráficas

Para desarrollar las gráficas reutilizamos el componente que desarrollamos para las pruebas, generando un .jar diferente, a continuación una muestra de como se ven las gráficas.



VI. Dificultades y soluciones

1. Desconocimiento de la nueva especificación para assembler, nos apoyamos de documentación en línea y los ejemplos desarrollados por el profesor.
2. Realización de la gráfica, reutilizamos el módulo de gráficas que ya habíamos implementado para el módulo de pruebas.
3. Dificultad verificando los resultados, se desarrolló un programa simple en Java para realizar comparaciones.
4. Coordinación en el desarrollo del proyecto, utilizamos un repositorio centralizado y nos asignamos componentes del proyecto.
5. El manejo de la pila, se solucionó a través de la revisión extensiva de código y el manejo de versiones.

VII. Conclusiones

1. Las instrucciones utilizadas por el SimuProc no corresponden exactamente a las usadas por el procesador intel.
2. El lenguaje Assembler es complejo de manejar, ya que se han de cuidar muchos aspectos de la programación que damos por hecho en otros lenguajes.
3. El compilador gcc convierte las instrucciones del lenguaje C/C++ en instrucciones de assembler.
4. El manejo de la pila es crucial para el desarrollo de prácticamete cualquier problema.
5. Se pueden integrar diferentes lenguajes de programación a través de diferentes ejecutables y el uso de llamados simulados a la línea de comandos.