Yojhan Steven García Peña

Programación Evolutiva

Práctica 3 – Convocatoria extraordinaria

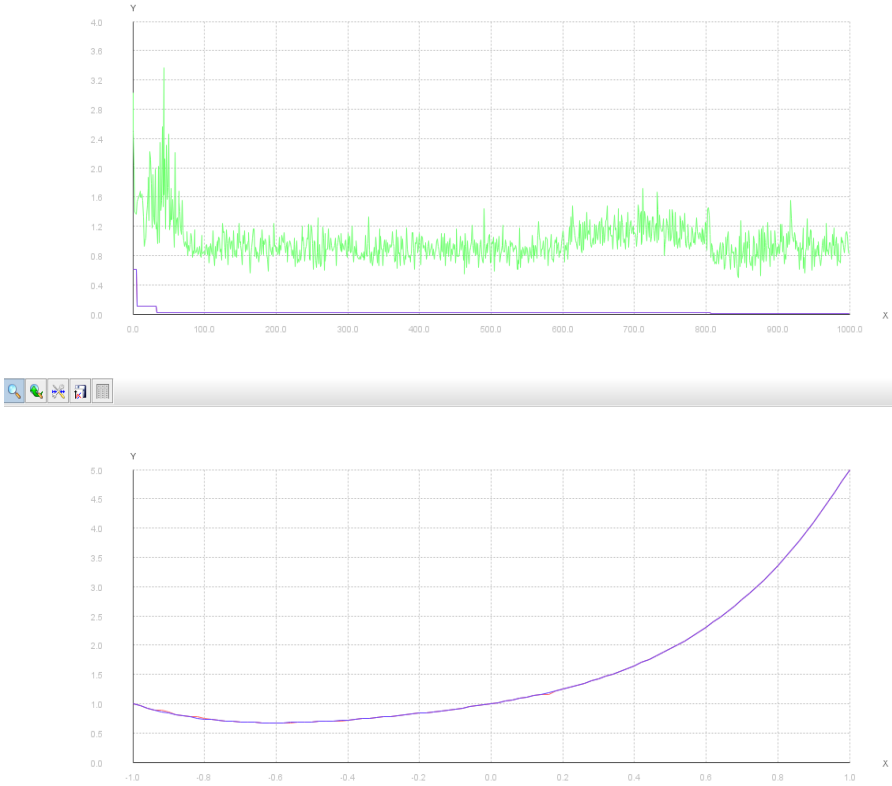
1. Gráficas

Problema:

Regresión simbólica de la función

$$f(x) = x^4 + x^3 + x^2 + x + 1$$

Resultados:



Error: 0

Resultado: (((-1-1) +-1) -((-1-(((x*(((x*x) +1) *((-1*-1) +x)))+2)+2))--1))

En la gráfica superior se ve la evolución del fitness y en la inferior la representación de ambas curvar (aunque estén superpuestas una encima de otra).

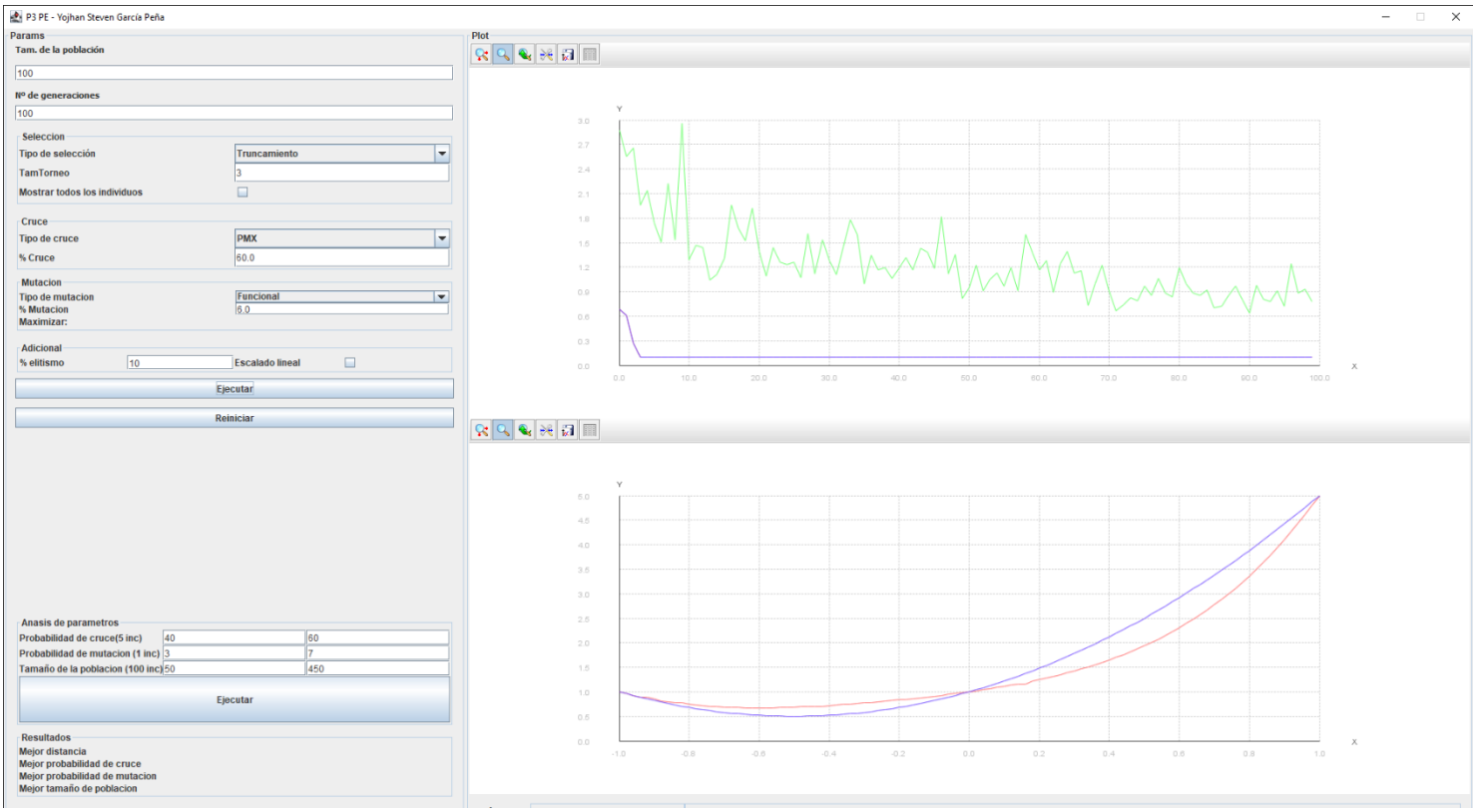
Se puede ver defectos visuales y puntos donde no coinciden, pero eso es debido al redondeo y a solo representar 100 puntos de la función. A lo largo de distintas ejecuciones el programa ha conseguido varias veces replicar la función original con distintas variaciones del resultado.

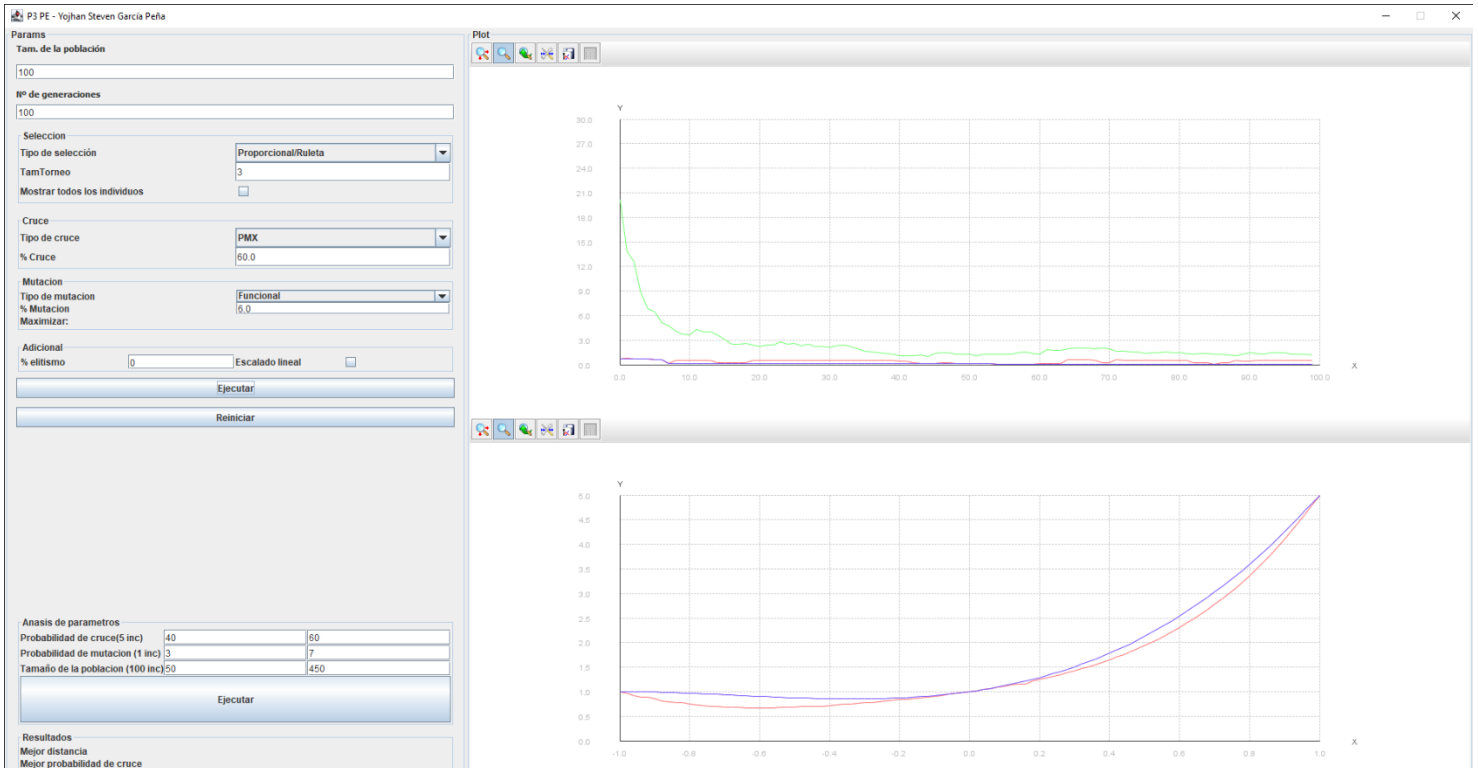
En el siguiente enlace se pueden ver ambas gráficas representadas con mejor detalle:

https://www.desmos.com/calculator/oa1jkzeyrs

Este es uno de los mejores resultados que he obtenido, y se ha logrado con una población de 350 individuos y a lo largo de 1.000 generaciones. En este caso he utilizado la selección por truncamiento, cruce de ramas con 55% de probabilidad de cruce y mutación por intersección con 6% de probabilidad de cruce. El algoritmo tenía aplicado un 10% de elitismo, activado escalado lineal y control de bloating con N = 2.

1.2 Otras gráficas con otros parámetros





2 Implementación

Esta práctica ha sido hecha sobre la práctica 2, cambiando todas las cosas necesarias para su adaptación, por lo que gran parte del código es igual a la práctica anterior.

El primer cambio que tuve que hacer fue sobre el tipo de genes, eliminando la implementación anterior y haciendo una en las que los genes son representados por árboles.

Para manejar el fitness, me he creado una clase estática auxiliar llamada 'CalcularFitness'. Esa clase contiene el array de los 100 puntos calculados. También contiene métodos auxiliares para consultar el dataset y calcular el fitness de algún individuo.

Realmente no utilizo 100 puntos, sino que utilizo 101 para incluir también el punto cuando la x = 1 en los puntos guardados.

```
public static double ComputeIndex(int idx) {

    double x = (idx - 50) / 50.0;

    double v = Math.pow(x, 4) + Math.pow(x, 3) + Math.pow(x, 2) + x + 1;

    return v;
}

public static double EvaluateDataset(double value) {


    if(value < -1 || value > 1) return 0;

    int idx = (int)((value + 1) * 50);

    return dataset[idx];
}
```

Se ve la implementación del método para rellenar el dataset y el método para consultarlo

De entre los cambios que había que hacer, lo más complejo fue la correcta implementación del árbol y sus funcionalidades. Implementar los métodos de cruce y mutación fue bastante trivial.

Para almacenar el valor de los nodos del árbol he creado una clase valor, la cual puede tomar un valor de función o un valor de terminal. Ambos son representados mediante enumerados.

Para la inicialización de los individuos he utilizado la inicialización creciente (Grow initialization).

```java
public static double CalculateFitness(Nodo raiz) {

    double error = 0;

    double incremento = 2.0 / (numeroPuntos - 1);

    for(int i = 0; i < numeroPuntos; i++) {

        double x = -1 + incremento * i;

        double fitnessReal = EvaluateDataset(x);

        double fitnessCalculado = CalcularFitnessDeRama(raiz, x);

        double dif = fitnessReal - fitnessCalculado;

        error += (dif * dif) / numeroPuntos;
    }


    return error;
}
```

Esta es la forma en la que se calcula la función de fitness. Recibe como parámetro el nodo raíz del árbol. El fitness es la media de la diferencia al cuadrado en cada uno de los 100 puntos.

```java
public static double CalcularFitnessDeRama(Nodo rama, double x) {


    if(rama.GetValor().esFuncion())
    {

        switch(rama.GetValor().funcion) {
        case add:
            return CalcularFitnessDeRama(rama.izquierdo, x) + CalcularFitnessDeRama(rama.derecho, x);
        case mult:

            return CalcularFitnessDeRama(rama.izquierdo, x) * CalcularFitnessDeRama(rama.derecho, x);
        case sub:

            return CalcularFitnessDeRama(rama.izquierdo, x) - CalcularFitnessDeRama(rama.derecho, x);
        }

    }
    else {


        switch(rama.GetValor().terminal) {
        case Cero:
            return 0;
        case Dos:
            return 2;
        case MenosDos:
            return -2;
        case MenosUno:
            return -1;
        case Uno:
            return 1;
        case x:
            return x;
        }
    }

    return 0;
}
```

Esta es la función dónde se calcula el fitness de cada rama.

He añadido como control de bloating el método de Tarpeian y se puede activar y desactivar desde la interfaz al igual que controlar el valor de la N.

He añadido un recuadro donde se muestran el valor que tenían los distintos parámetros con los que se ha encontrado la mejor solución hasta el momento.

Por último, como el valor obtenido al ejecutar el algoritmo puede ser muy aleatorio (con la misma configuración con la que se consigue el mejor resultado también se pueden conseguir resultados pésimos) he añadido a la interfaz la posibilidad de controlar una variable de repeticiones. Cuando se ejecuta el algoritmo se ejecutará tantas veces como se haya marcado y representará el mejor resultado obtenido.

3 Conclusiones y curiosidades

Me sorprende haber podido conseguir una función que sea exactamente igual que la función original, pues pensé que lo que conseguiría sería una aproximación más o menos buena. Pero ya no solo es eso lo que me ha sorprendido, sino que el hecho que al representar ambas funciones en todo su dominio podemos ver que coinciden en todo momento y no solo en el intervalo en el que trabajábamos.

Esta práctica no ha sido especialmente complicada pero sí que ha sido una de las que más me ha gustado porque al ser la más visual (por el hecho de comparar ambas gráficas) me ha llamado mucho más la atención ver como poco a poco iba mejorando el algoritmo.

4 Reparto de tareas

Todo lo implementado en la práctica ha sido hecho por mí.

5 Parte B

Lo más complicado de la parte B en mi opinión ha sido entender lo que se pide. He tenido que estar leyendo el temario unas cuantas veces hasta que he podido entender con exactitud lo que se pedía. Una vez entendido lo que se pide, el resto de la práctica es bastante trivial y lo he podido hacer utilizando es esqueleto de la práctica 3 y utilizando código de prácticas anteriores.

Como diferencia significativa con respecto a la parte A, a parte de la implementación del árbol, es la forma en la que se calcula el fitness. He utilizado de nuevo la clase ‘CalcularFitness’ pero he tenido que crear otra clase auxiliar llamada ‘DecodificationData’.

```java
public static double CalculateFitness(double x, int[] codones, DecodificationData data, boolean esFuncion) {

    if(!data.isValid())
    {
        //System.out.println("Error: se ha sobrepasado el numero de wraps permitidos");

        //Si

        return 0;
    }

    int current = codones[data.Pop()];


    if (esFuncion) {

        Funcion f = GetFuncion(current);



        boolean isFunction = SiguienteEsFuncion(codones, data.Pop());

        double a = CalculateFitness(x, codones, data, isFunction);

        isFunction = SiguienteEsFuncion(codones, data.Pop());

        double b = CalculateFitness(x, codones, data, isFunction);


        switch (f) {
        case Add:
            return a + b;
        case Mult:
            return a * b;
        case Sub:
            return a - b;
        }

    }
```

Esta es la forma en la que se calcula el fitness en la parte B.

La gramática que he utilizado para desarrollar la práctica sería la siguiente:

N = {<Func>, <Op>, <Exp>}

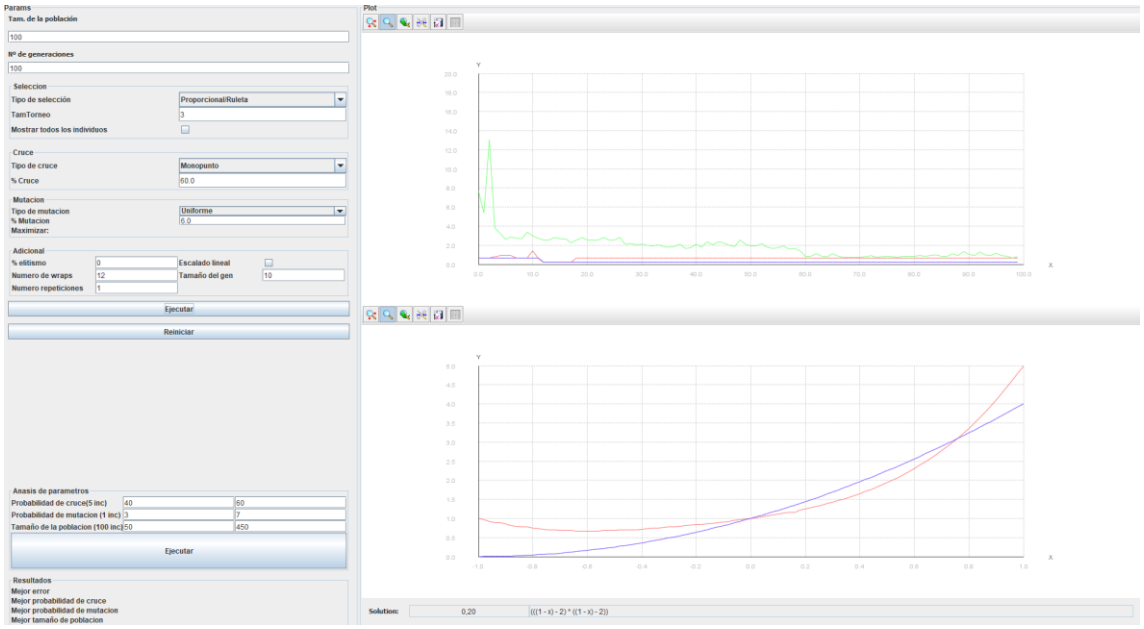
T = {x, -2, -1, 0, 1, 2}

S: <Func>

Func = <Exp> <Op> <Epx>

Op = + | * | -

Exp = x | -2 | -1 | 0 | 1 | 2



No he conseguido resultados igual de buenos con la gramática evolutiva que utilizando un árbol.

Todo lo implementado en la parte B ha sido hecho por mí.

Esta memoria es bastante peor porque no he podido dedicarle mucho tiempo debido a mi lesión, ya que apenas podía ponerme en el ordenador y cuando lo hacía no podía ser en periodos de más de 10 minutos debido al dolor y el mareo.