Primer Proyecto Programado

Curso:
Lenguajes de Programación

Symbolic Regression

Profesor:
Eddy Ramírez Jiménez

Estudiantes:
Alejandro Cerdas
Kener Castillo

May 23, 2024

1 Resumen ejecutivo

El proyecto consiste en que, dado unos puntos, se genere una función que se aproxime a la función generadora de dichos puntos, mediante programación genética. El documento consiste en una introducción para describir el problema, un marco teórico con la teoría necesaria para el proyecto, un análisis de los resultados, unas conclusiones con un análisis de todo el proyecto, los aprendizajes del equipo y una bibliografía donde se puede encontrar las referencias utilizadas en el proyecto.

El IDE usado por el equipo para la realización fue Visual Studio Code, ideal para el trabajo remoto entre miembros del grupo de trabajo, además de su compatibilidad con un gran número de lenguajes, entre los que se encuentra el lenguaje utilizado en este proyecto: Racket. Racket es un lenguaje funcional de la familia de Scheme y Lisp. La creación de Racket se le atribuye a PLT tras una serie de versiones de diferentes programas. Racket es ideal para el proyecto gracias a la facilidad para usar bibliotecas, macros, módulos, recursión e hilos. En el caso del proyecto, resulto muy útil gracias al fácil manejo de árboles que ofrece. Para la realización del proyecto se utilizó 3 bibliotecas de Racket: Plot para el despliegue de las gráficas de las funciones y puntos. Otra biblioteca utilizada fue racket/gui, la cual otorgo la posibilidad de realizar frames y canvas para el despliegue y actualización de las gráficas y finalmente se utilizó racket/futures para el uso de hilos.

El área de la computación que abarca lo estudiado en este proyecto se llama symbolic regression. Este último es un tipo de análisis de regresión que busca el espacio de todas las expresiones matemáticas para encontrar el modelo que mejor aproxima un conjunto de datos dado. El espacio de búsqueda se da sobre las funciones y variables del conjunto definido, buscando maximizar la precisión y simpleza.

La técnica usada para resolver el problema fue programación genética, por lo que se definió un árbol de expresiones matemáticas como individuo y un bosque como la población. El fitness es determinado mediante la suma del cuadrado de las diferencias a cada punto. Para la selección se eligen n individuos y se saca el mejor de estos. Para el cruce, se obtiene una rama aleatoria de cada padre y se coloca de forma aleatoria en el otro. Y como se diseñaron las siguientes mutaciones: eliminar una rama aleatoria, agregar una rama aleatoria en una posición aleatoria o cambiar operaciones aleatorias en los nodos del árbol. Cada una de estas tiene una probabilidad para aplicarse y tratan de aumentar la diversidad y corregir los árboles. Mediante paralelismo se crearon unas cuantas poblaciones independientes. Cada P generaciones se realiza un proceso de migración que consiste en el intercambio de T individuos entre tres poblaciones. Por último, se utiliza elitismo para conservar el mejor individuo que se encuentre y mediante la herramienta gráfica de Racket, Plot 3d, se puede observar a este en cada generación.

Se realizó pruebas con distintas funciones y se obtuvieron resultados positivos con expresiones no muy complejas. Sin embargo, no se logró aproximar el resto de funciones y el algoritmo presentó dificultades para trabajar con operaciones como exponenciación y logaritmos.

2 Introducción

El proyecto consiste en que, dado unos puntos, se genere una función que se aproxime a la función generadora de dichos puntos, mediante programación genética. Se sabe que todas las operaciones que forman parte de la función son binarias como: $a+b$, $a-b$, a/b , $a*b$, $\log_a(b)$, a^b, su dominio es $\mathbb{R}^2$ y su codominio $\mathbb{R}$. El proyecto se llevó a cabo en Racket, debido a sus características mencionadas en el marco teórico.
Este documento se presenta la solución al problema antes mencionado usando symbolic regression. El documento consta de un marco teórico, donde se describe el lenguaje, el IDE y se explica la teoría necesaria para realizar el proyecto. Además, se presenta una sección de análisis de los resultados, donde hay una serie de pruebas de la solución y análisis de dichas pruebas. También, se presenta una conclusión con un análisis del proyecto en general. Finalmente, se tiene una sección de aprendizajes, donde se presenta lo aprendido por cada miembro del equipo, y una bibliografía.

3 Marco teórico

El IDE usado para el desarrollo del proyecto fue **Visual Studio Code**. VSC es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Es un editor de código gratuito y de código abierto. Lanzado el 18 de noviembre de 2015, Visual Studio Code fue construido sobre el framework Electron. Es compatible con varios lenguajes de programación y un conjunto de características que pueden o no estar disponibles para un lenguaje dado [1]. La principal razón de usar este IDE, es su facilidad de trabajo remoto entre todos los miembros del grupo de trabajo.

El lenguaje de programación fue Racket. Se trata de un lenguaje de programación funcional familia de Lisp y Scheme diseñado con el objetivo de permitir crear nuevos lenguajes o dialectos. En enero de 1994 PLT, fundada por Matthias Felleisen, creó MrED como la primera máquina virtual para Racket. Más tarde desarrollaron DrScheme con PLT Scheme como principal lenguaje de desarrollo y en los siguientes años se fueron introduciendo nuevas características que lo consolidaron como entorno de desarrollo. El 7 de junio de 2010, con la salida de la versión 5.0, se le cambió el nombre a PLT Scheme por Racket, así como el nombre DrScheme a DrRacket . Racket resulta un lenguaje altamente flexible, incluso sin el uso de dialectos, características como el uso de bibliotecas, macros, módulos, recursión en cola y su sintaxis le permite ser usado para todo tipo de tareas [2]. Para symbolic regression resulta especialmente útil, ya que facilita el uso de árboles y la creación, así como la evaluación, de funciones en tiempo de ejecución.

En la solución se implementaron las siguientes bibliotecas de Racket:

- La biblioteca Plot proporciona una interfaz flexible para producir casi cualquier tipo de gráfico. Incluye diagramas de dispersión, diagramas de líneas, diagramas de contorno, histogramas y superficies e isosuperficies 3D [3].
- La biblioteca racket/gui proporciona todos los enlaces de clases, interfaces y procedimientos definidos en este manual, además de los enlaces de racket/draw y file/resource [4].
- La biblioteca racket/futures da acceso a las funciones Future y Touch, que brindan acceso al paralelismo admitido por el hardware y el sistema operativo [6].

Symbolic regression es un tipo de análisis de regresión que busca el espacio de todas las expresiones matemáticas para encontrar el modelo que mejor aproxima un conjunto de datos dado [7]. Análisis de regresión en el modelado estadístico es un conjunto de procesos estadísticos para estimar la relación entre una variable dependiente, usualmente el resultado de una función, y una o más variables independientes, como los parámetros de una función. La forma más común de análisis de regresión es regresión lineal [5].

Espacio de búsqueda se realiza sobre todas las expresiones matemáticas formadas por funciones y variables de un conjunto definido, buscando maximizar la precisión y la simpleza. En el caso de este proyecto, a partir de un conjunto de puntos dado se buscaba encontrar la función que cumpliera con estas características.

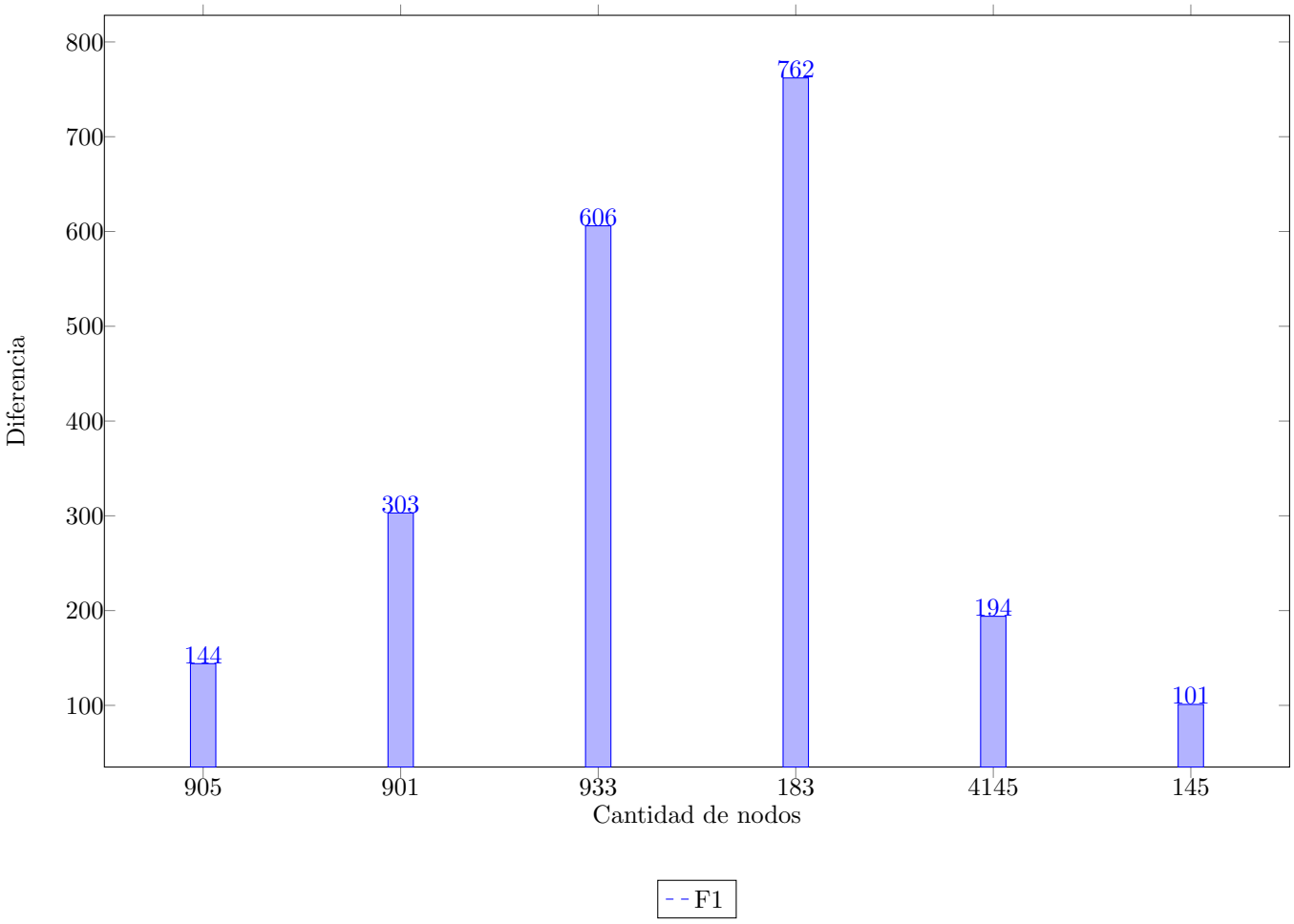
La solución empleada consiste en un algoritmo genético, el cual es una serie de pasos que simulan un proceso de selección natural, propuesta por Darwin, y los cuadros de Punnett. Es decir, la idea es elegir a una población de individuos y, mediante una función de selección, realizar cruces entre individuos con el fin de crear otra generación. Además, dentro de cada cruce debe existir la posibilidad de mutar al individuo. Se debe seleccionar un algoritmo para la selección, cruce y mutación apto para el problema a resolver. Durante cada generación de individuos, se mide a cada individuo para saber que tan cerca está de la respuesta deseada, a esta función se le conoce como fitness. Gracias al fitness podemos conocer al elite. El elite es el individuo con mayor fitness (más cercano a la respuesta) y debe estar presente siempre dentro de la población en cada generación. Al tener todas estas funciones y si son ideales para el problema, el algoritmo genético estaría listo. En versiones más avanzadas de los algoritmos genéticos se pueden encontrar a programas con más de una población y con una función de migración. La forma de trabajar varias poblaciones es mediante hilos y la función de migración busca intercambiar individuos entre poblaciones. Las demás funciones permanecen de la misma forma mencionada.
Para el proyecto, como individuos se seleccionó un árbol y como población se seleccionó un bosque. Como conjunto de poblaciones, se utilizó un conjunto de bosques. La función de fitness definió evaluando todo el árbol con todos los puntos dados y se sumó el cuadrado de las diferencias con cada punto. La función de elite, reconoce al elite entre el conjunto de poblaciones. Para la selección se utiliza la forma de torneo, que consiste en la selección aleatoria de n posibles padres y seleccionar los mejores dos. Para el cruce, se obtiene una rama aleatoria de cada padre y se coloca en una posición aleatoria del otro. En el caso de la mutación se tiene 3 diferentes tipos de algoritmos, una para eliminar nodos aleatorios en un árbol, otra para agregar ramas aleatorias en posiciones aleatorias en un árbol y la última, consiste en cambiar la operación de nodos aleatorios en un árbol. Finalmente, como función de migración, se obtienen una serie de individuos de 3 poblaciones y se intercambian esos individuos.

4 Análisis de los resultados

4.1 F1

Table 1: F1

fitness	cantNodos	generaciones
144	905	2000
303	901	3500
606	933	3500
762	183	3500
194	4145	20000
101	145	20000


800
700
600
500
400
300
200
100
Diferencia
762
606
303
194
144
101
905
901
933
183
4145
145
Cantidad de nodos
F1

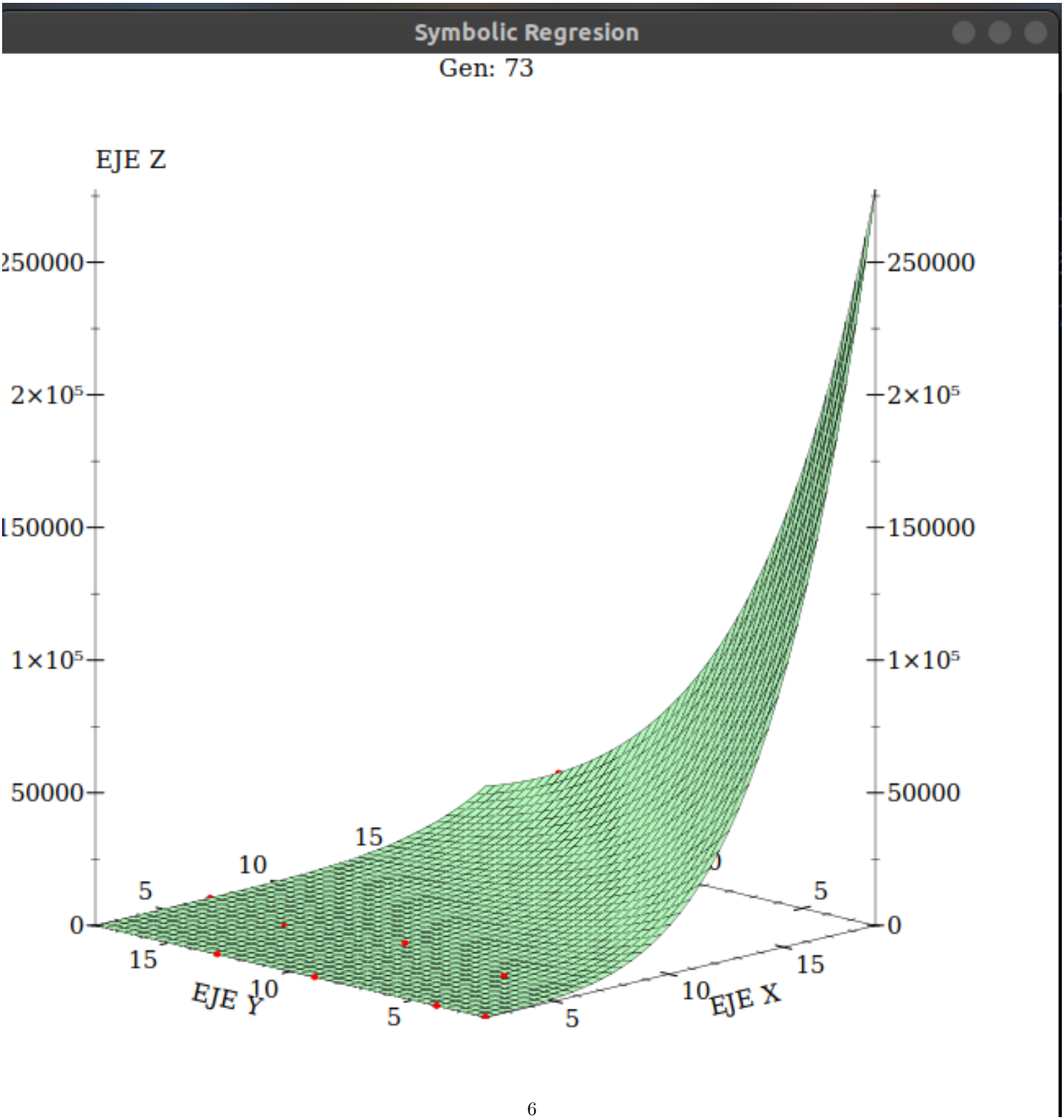
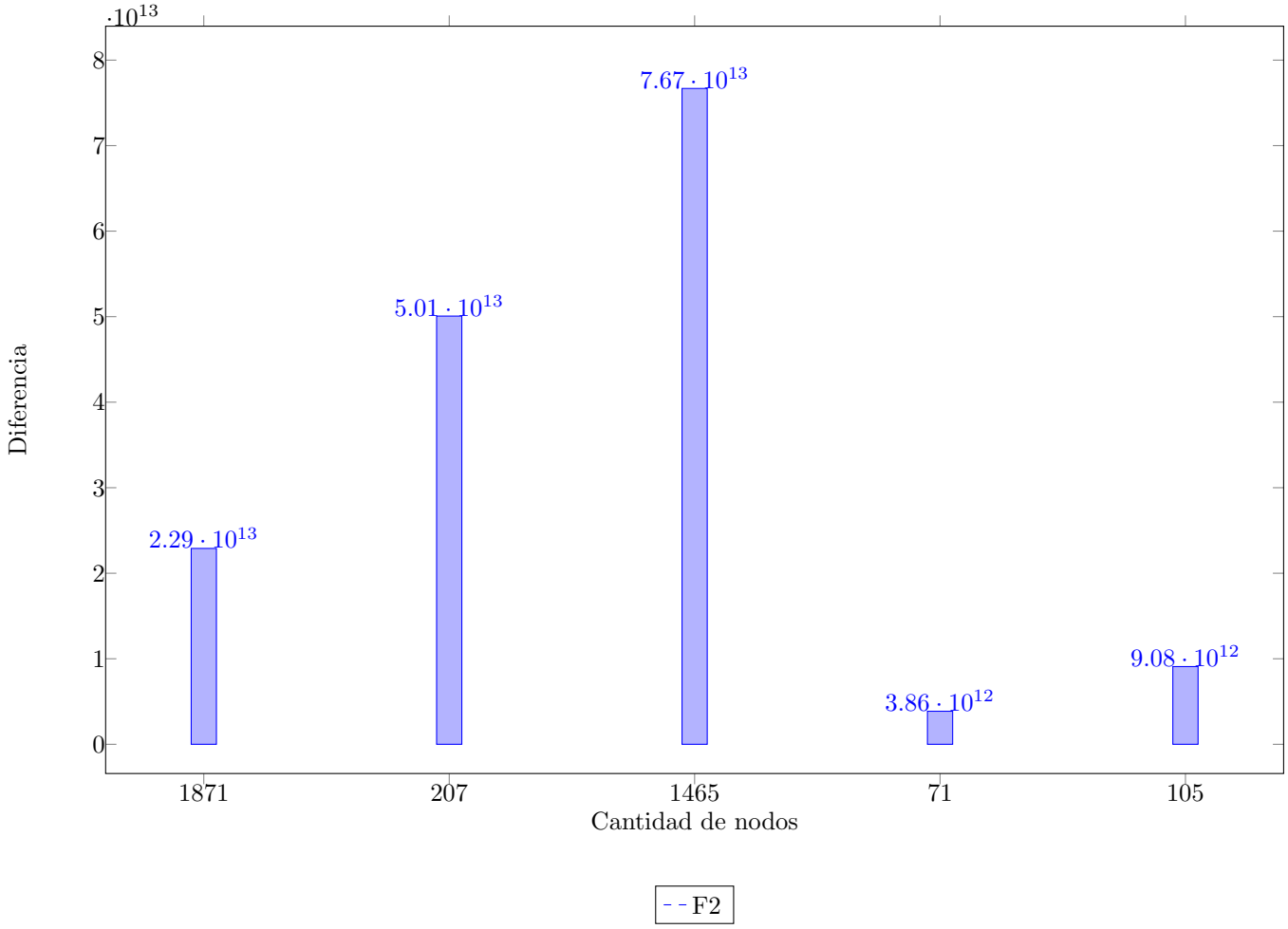


Figure 1: Gráfica de f1

4.2 F2

Table 2: F2

fitness	cantNodos	generaciones
22900939416129	1871	2000
50068916361812	207	1000
76683910852828	1465	1000
3864213561068	71	1000
9084196575269	105	2000


$\cdot 10^{13}$
8
7
6
5
4
3
2
1
0
Diferencia
$2.29 \cdot 10^{13}$
$5.01 \cdot 10^{13}$
$7.67 \cdot 10^{13}$
$3.86 \cdot 10^{12}$
$9.08 \cdot 10^{12}$
1871
207
1465
71
105
Cantidad de nodos
F2

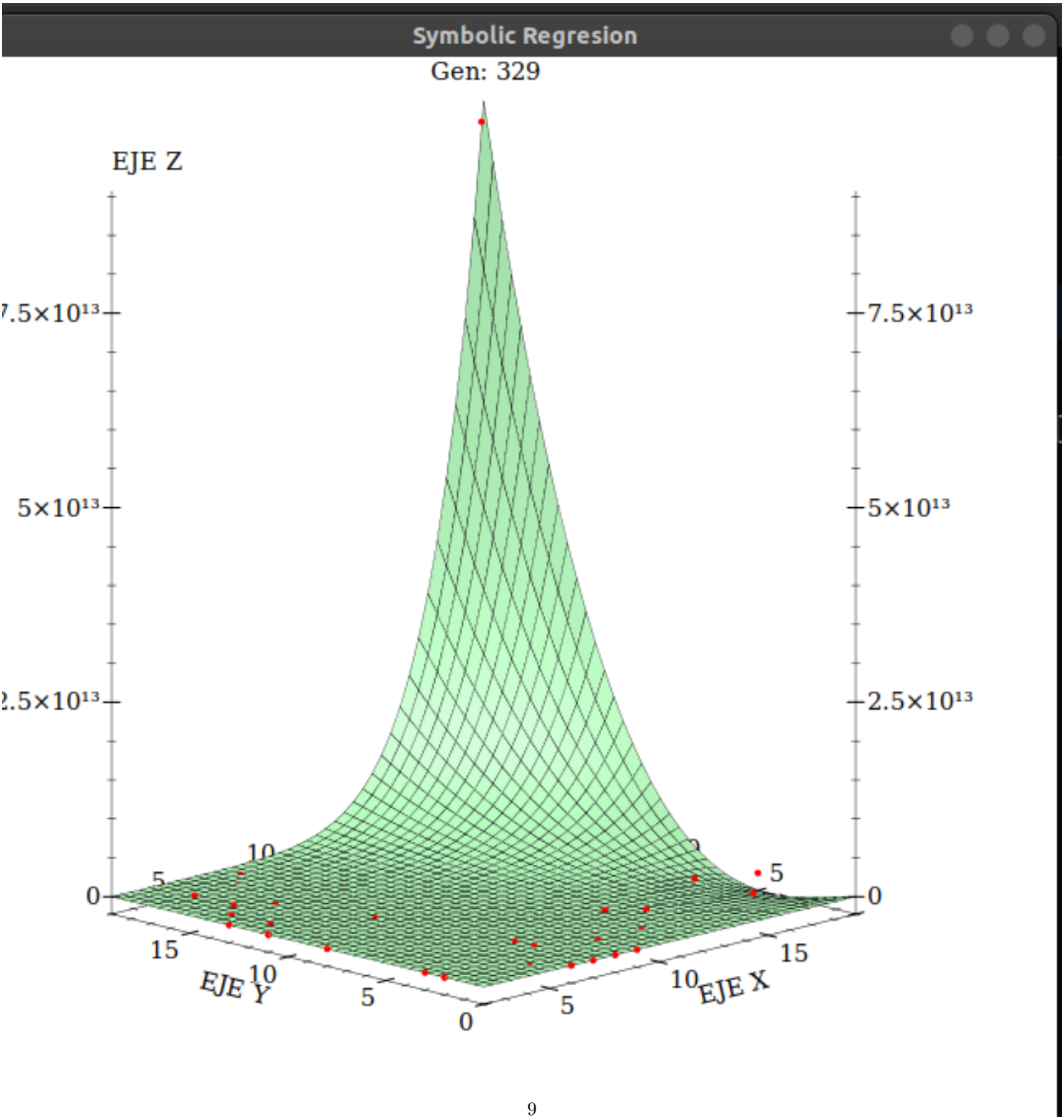
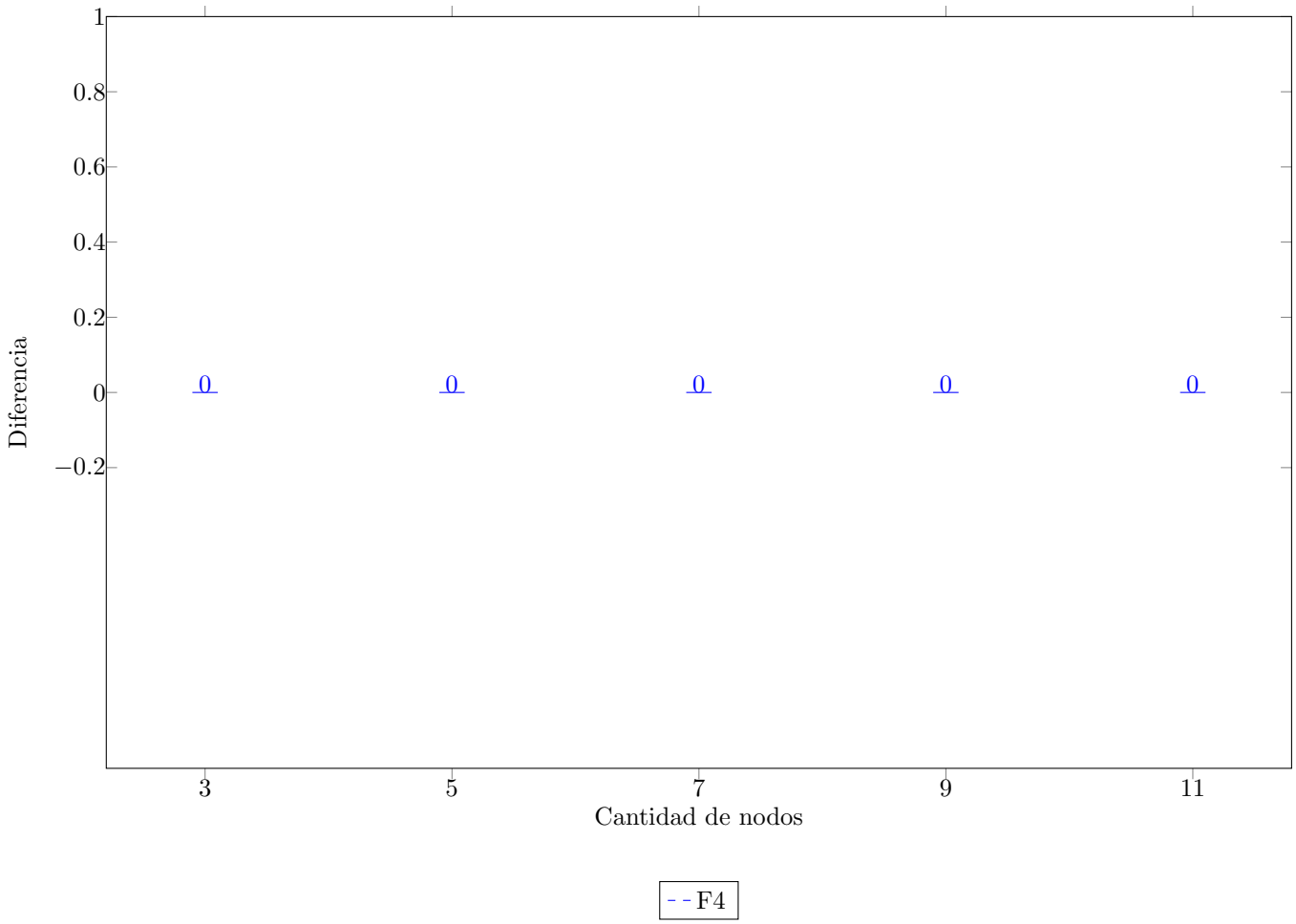


Figure 2: Gráfica de f2

4.3 F4

Prueba	Nodos	Distancia	Generaciones
1	7	0	1
2	5	0	1
3	3	0	1
4	9	0	1
5	11	0	3


1
0.8
0.6
0.4
0.2
0
−0.2
Diferencia
0
0
0
0
0
3
5
7
9
11
Cantidad de nodos
F4

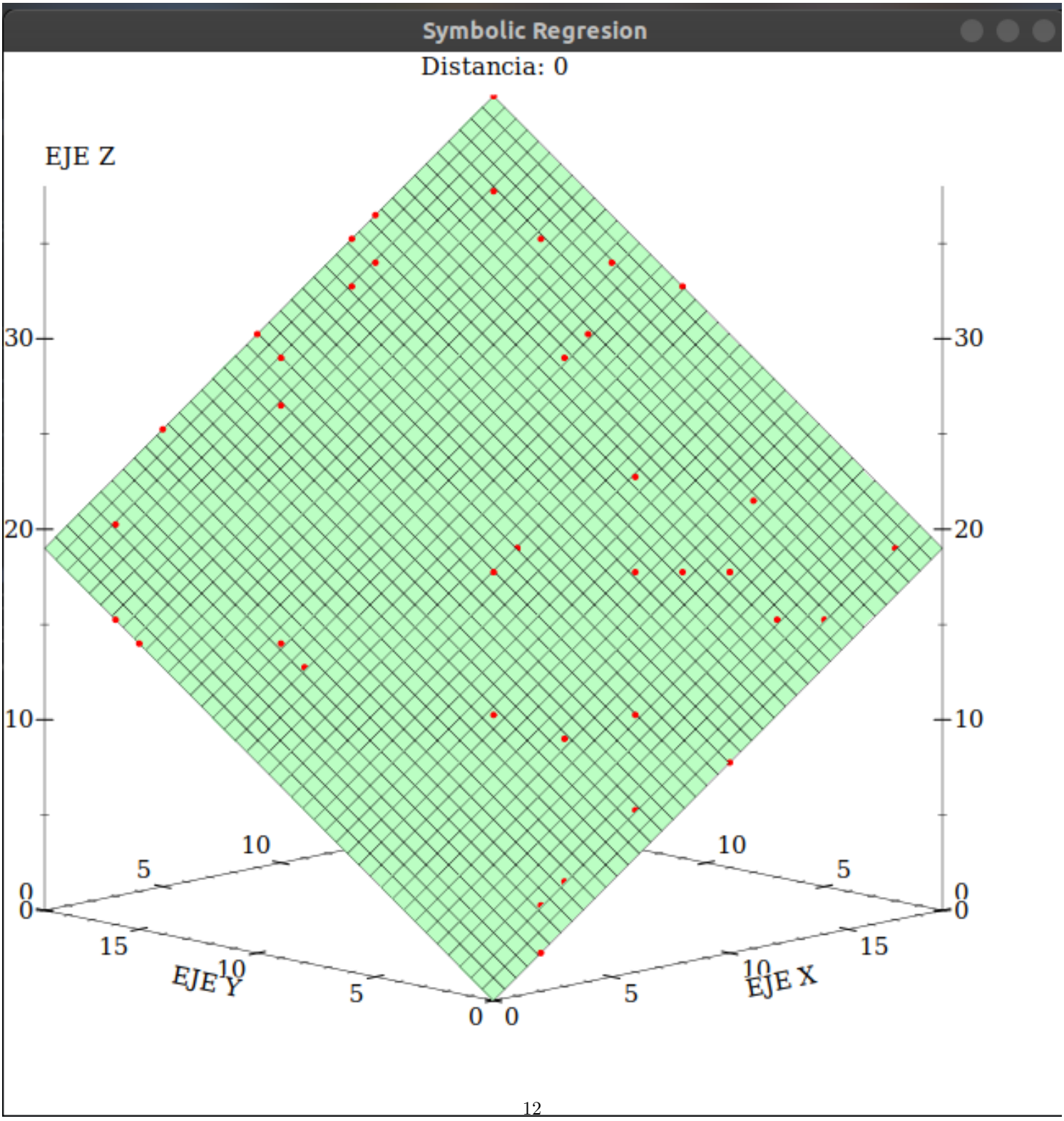
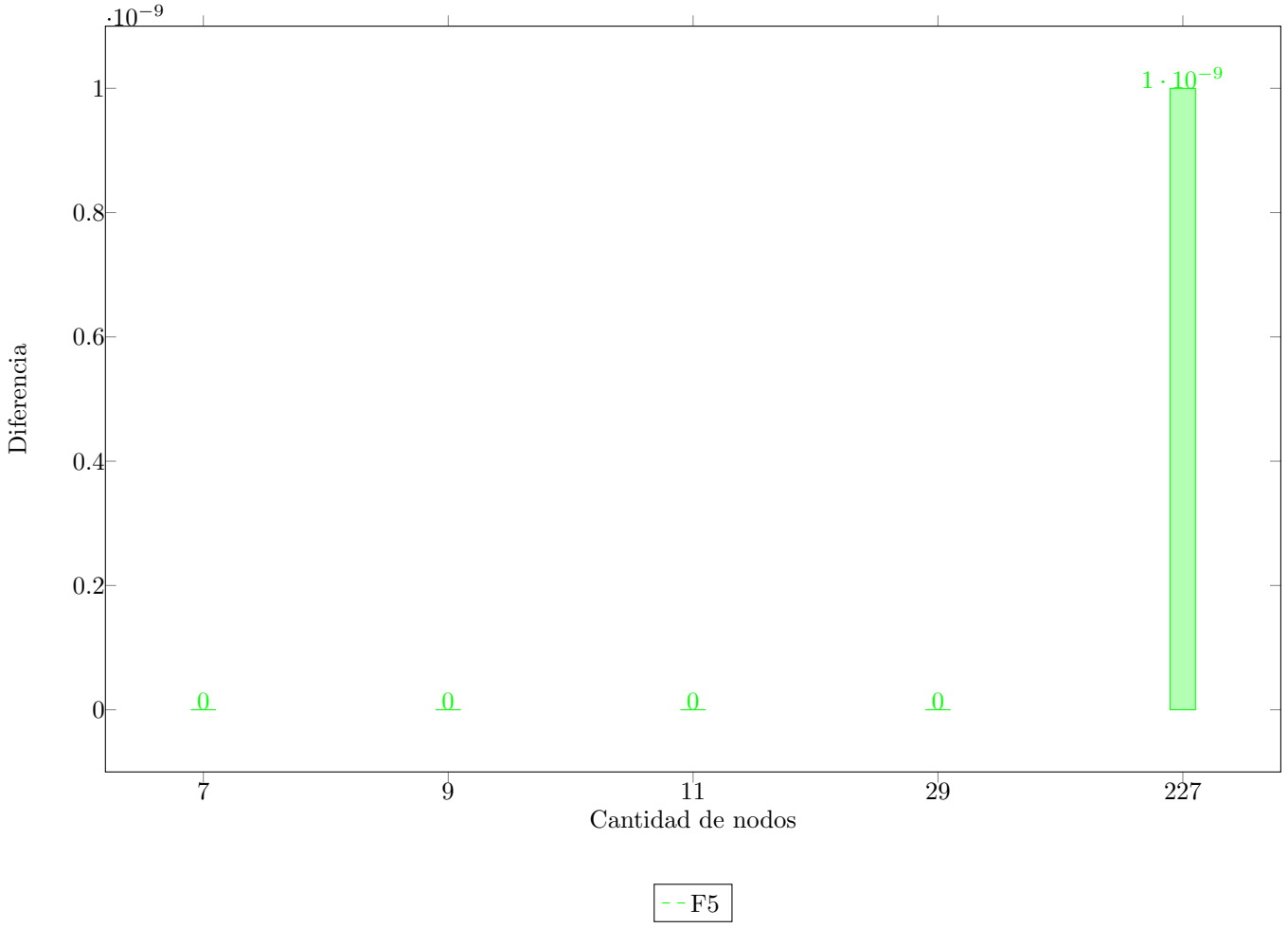


Figure 3: Gráfica de f4

4.4 F5

Prueba	Nodos	Distancia	Generaciones
1	227	0.000001	1000
2	7	0	4
3	29	0	8
4	9	0	4
5	11	0	456


$\cdot 10^{-9}$
1
0.8
0.6
0.4
0.2
0
Diferencia
$1 \cdot 10^{-9}$
0
0
0
0
7
9
11
29
227
Cantidad de nodos
F5

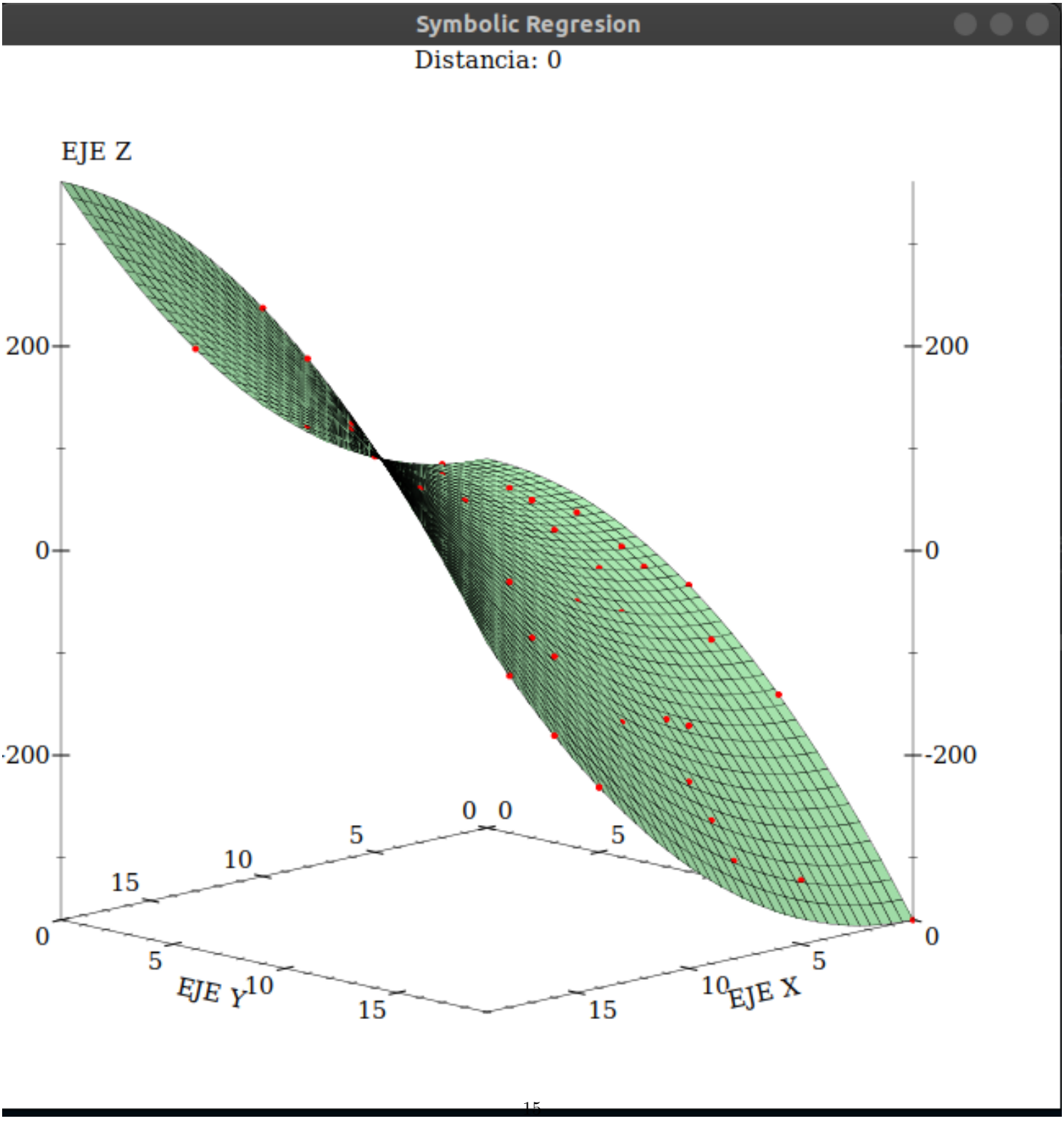
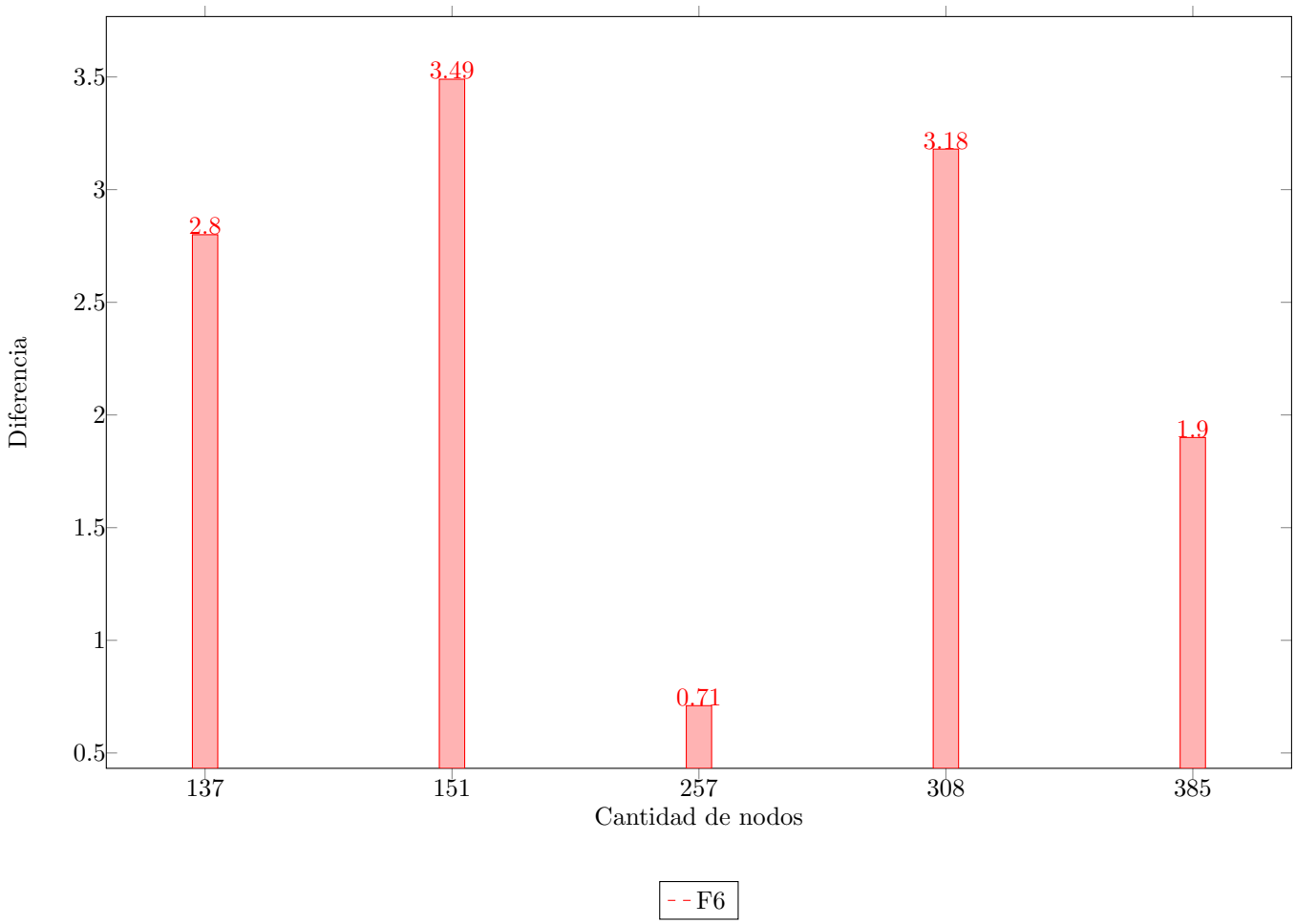


Figure 4: Gráfica de f5

4.5 F6

Prueba	Nodos	Distancia	Generaciones
1	257	0.71	1000
2	137	2.80	1000
3	151	3.49	1000
4	385	1.9	1000
5	308	318	1000


Diferencia
3.5
3
2.5
2
1.5
1
0.5
2.8
3.49
0.71
3.18
1.9
137
151
257
308
385
Cantidad de nodos
F6

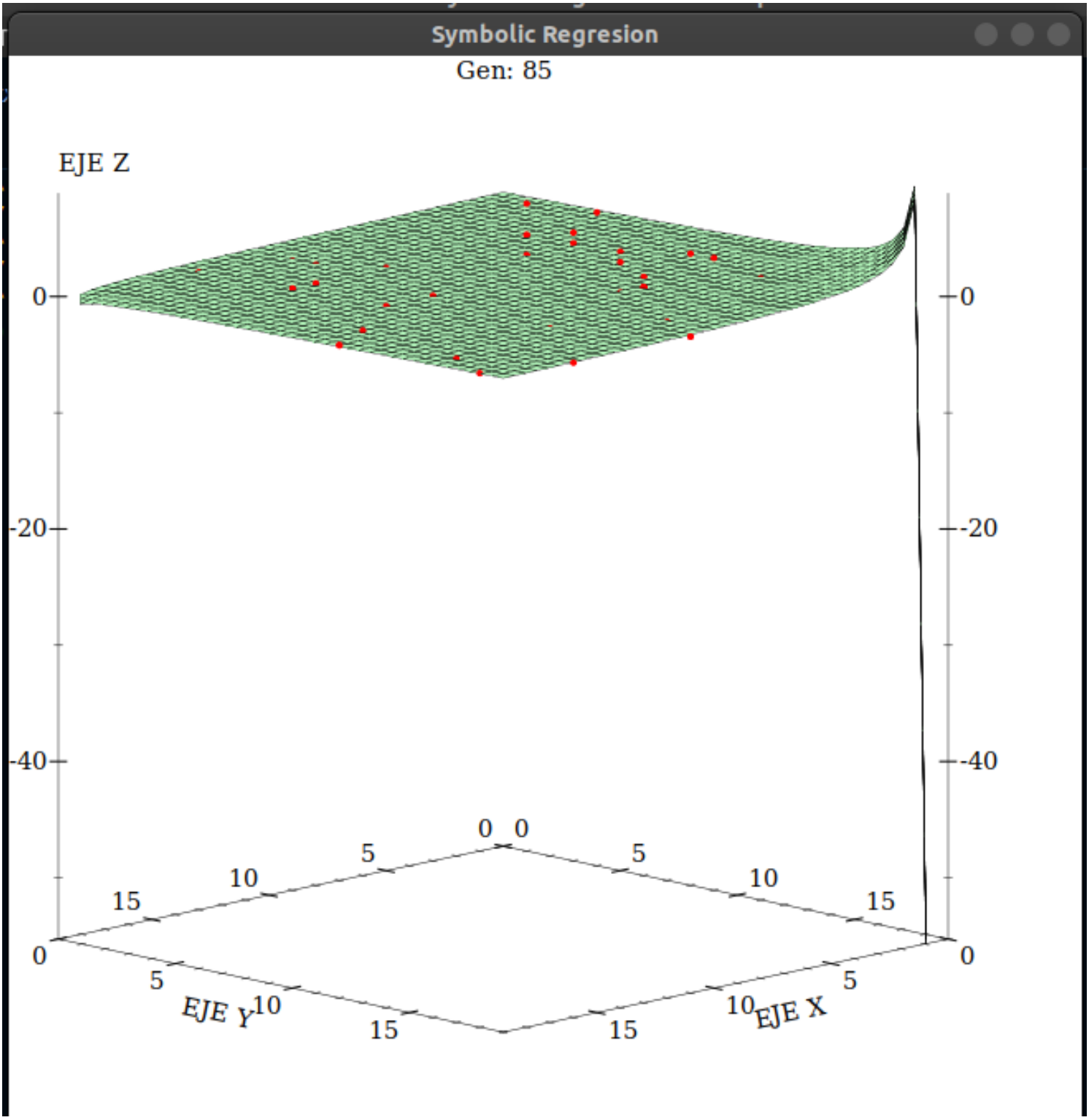


Figure 5: Gráfica de f6

5 Conclusiones

En f1, el programa presenta una variación en los resultados, tanto en la cantidad de nodos del árbol como en la distancia total. Dependiendo qué tan bien le vaya logra alcanzar un mínimo relativamente bueno de 101 y un máximo de 762, por lo que presenta algo de dificultad para aproximar la función en comparación con f4, f5 y f6. De todas la pruebas realizadas se mantuvo a una distancia total entre 100 y 1000 aproximadamente. Algo a destacar es que presentó un tiempo ejecución bastante rápido, por lo que se pudo probar hasta 20000 generaciones.

F2 resultó ser una de las funciones con mayor dificultad para aproximar. El programa no logró aproximarla y se mantuvo constantemente a una distancia bastante alta. Se presentaron árboles bastante grandes como muchos de los intentos por parte del algoritmo por aproximar la función, así como un aumento considerable del tiempo de ejecución al incrementar la cantidad de generaciones.

En f4, el programa acierta con gran rapidez la función y se obtiene 0 de fitness. Lo que varía son el número de nodos, pero no tiene ningún efecto en esta función. Le toma muy pocas generaciones encontrar una solución y usualmente resultaron expresiones simples y con pocos nodos. La función a encontrar era bastante simple, por lo que no sorprenden muchos estos resultados.

En f5, el programa tiene a acertar la función generadora en un bajo número de generaciones. Sin embargo, cuando no se logra acercar a la función en pocas generaciones, el programa obtiene una función con fitness 0 después de una serie de generaciones o se aproxima casi llegando a 0. Además, se puede observar que entre menos nodos tenga el árbol, f5 es más probable que resulte 0. En general se obtienen resultados buenos y no le toma mucho tiempo al programa en correr una cantidad grande de generaciones.

En f6, los individuos tienden a acercarse más a 0, si tienen una mayor cantidad de nodos. Además, se puede que al ser un logaritmo, al programa le cuesta acertar la función con un fitness de 0, aunque sí logra aproximarla exitosamente y en una cantidad de generaciones no muy grande.

En general, el algoritmo presentó una mayor facilidad y tendencia para aproximar las funciones mediante operaciones básicas como la suma, resta y multiplicación, lo que resultaba en ocasiones en árboles grandes pero una rápida ejecución. Por otro lado, presentó gran dificultad con operaciones como logaritmo y exponenciación por la alta probabilidad de indefinirse y de crecer drasticamente en el caso de la exponenciación, consecuentemente aparecieron con menos frecuencia en los árboles y aumentaban el tiempo y uso de memoria al ejecutarlas con números muy grandes.

6 Aprendizajes

6.1 Alejandro Cerdas

Aprendí conceptos generales de números complejos y symbolic regression. Además, mejore en el uso de Racket y Swindle. Ahora conozco el concepto de simulated annealing y la diferencia entre concurrencia y paralelismo.

6.2 Kener Castillo

En esta ocasión descubrí conceptos nuevos y técnicas como simulated annealing, programación genética y symbolic regression que se relacionan con el área de machine learning e inteligencia artificial, lo que me resultó muy interesante. Por otro lado, pude familiarizarme y reforzar mis conocimientos con un lenguaje funcional como Racket y enfrentarme a un problema totalmente diferente de los que había resuelto con algoritmos genéticos.

7 Bibliografía

[1] "Visual Studio Code," Wikipedia, Nov. 17, 2020. https://es.wikipedia.org/wiki/Visual_Studio_Code.

[2] C. de, "lenguaje de programación," Wikipedia.org, May 17, 2014. https://es.wikipedia.org/wiki/Racket_(lenguaje_de_programaci%C3%B3n) (accessed May 23, 2024).

[3] "Plot: Graph Plotting," docs.racket-lang.org. https://docs.racket-lang.org/plot/index.html (accessed May 23, 2024).

[4] "The Racket Graphical Interface Toolkit," docs.racket-lang.org. https://docs.racket-lang.org/gui/

[5] Wikipedia Contributors, "Regression analysis," Wikipedia, Mar. 22, 2019. https://en.wikipedia.org/wiki/Regression

[6] "11.4 Futures," docs.racket-lang.org. https://docs.racket-lang.org/reference/futures.html (accessed May 23, 2024).

[7] "Symbolic regression," Wikipedia, Dec. 27, 2022. https://en.wikipedia.org/wiki/Symbolic_regression