Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

ENCONTRANDO FUNCIONES DE REGRESIÓN UTILIZANDO ALGORITMOS GENÉTICOS

Luis Mauricio Déleon Barquero
Asesorado por el Ing. Estuardo Zapeta

Guatemala, octubre de 2018

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

ENCONTRANDO FUNCIONES DE REGRESIÓN UTILIZANDO ALGORITMOS GENÉTICOS

TRABAJO DE GRADUACIÓN

PRESENTADO A LA JUNTA DIRECTIVA DE LA
FACULTAD DE INGENIERÍA
POR

LUIS MAURICIO DÉLEON BARQUERO
ASESORADO POR EL MSC ING. ESTUARDO ZAPETA

AL CONFERÍRSELE EL TÍTULO DE

INGENIERO EN CIENCIAS Y SISTEMAS

GUATEMALA, OCTUBRE DE 2018

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

FACULTAD DE INGENIERÍA

NÓMINA DE JUNTA DIRECTIVA

DECANO	Ing. Pedro Antonio Aguilar Polanco
VOCAL I	Ing. Angel Roberto Sic García
VOCAL II	Ing. Pablo Christian de León Rodríguez
VOCAL III	Ing. José Milton de León Bran
VOCAL IV	Br. Oscar Humberto Galicia Nuñez
VOCAL V	Br. Carlos Enrique Gomez Donis
SECRETARIA	Inga. Lesbia Magalí Herrera López

TRIBUNAL QUE PRACTICÓ EL EXAMEN GENERAL PRIVADO

DECANO	Ing. Pedro Antonio Aguilar Polanco
EXAMINADOR	Ing. Oscar Alejandro Paz Campos
EXAMINADOR	Ing. Everest Darwin Medinilla Rodríguez
EXAMINADOR	Ing. Edgar Estuardo Santos Sutuj
SECRETARIO	Ing. Pablo Christian de León Rodríguez (a.i.)

HONORABLE TRIBUNAL EXAMINADOR

En cumplimiento con los preceptos que establece la ley de la Universidad de San Carlos de Guatemala, presento a su consideración mi trabajo de graduación titulado:

ENCONTRANDO FUNCIONES DE REGRESIÓN UTILIZANDO ALGORITMOS GENÉTICOS

Tema que me fuera asignado por la Dirección de la Escuela de Ingeniería en Ciencias y Sistemas, con fecha 31 de julio de 2018.

Luis Mauricio Déleon Barquero

Guatemala 14 de septiembre de 2018

Ing. Carlos Azurdia
Escuela de Sistemas
Facultad de Ingeniería USAC.

Respetable Ing. Carlos Azurdia:

Por medio de la presente hago de su conocimiento que apruebo el trabajo de tesis titulado "**Encontrando Funciones de Regresión utilizando Algoritmos Genéticos**" del estudiante Luis Mauricio Déleon Barquero que se identifica con carné No. 200614739 de la Facultad de Ingeniería, USAC, de la Carrera de Ingeniería en Ciencias y Sistemas, y que lo he acompañando en el desarrollo de la misma, durante el segundo semestre del año 2018.

Sin otro particular, me es grato suscribirme.

Ing. Estuardo Zapeta
Ingeniería en Ciencias y Sistemas
Colegiado 12767

F:_____

MSC Ing. Estuardo Zapeta
Catedrático de Maestria
Facultad de Ingeniería, USAC
estuardo.zapeta@gmail.com
4215-7406

Universidad San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 3 de octubre de 2018

Ingeniero
Marlon Antonio Pérez Türk
Director de la Escuela de Ingeniería
En Ciencias y Sistemas

Respetable Ingeniero Pérez:

Por este medio hago de su conocimiento que he revisado el trabajo de graduación del estudiante **LUIS MAURICIO DÉLEON BARQUERO** con carné **200614739 y CUI 2760 31083 0101** titulado **"ENCONTRANDO FUNCIONES DE REGRESIÓN UTILIZANDO ALGORITMOS GENÉTICOS"** y a mi criterio el mismo cumple con los objetivos propuestos para su desarrollo, según el protocolo aprobado.

Al agradecer su atención a la presente, aprovecho la oportunidad para suscribirme,

Atentamente,

Ing. Carlos Alfredo Azurdia
Coordinador de Privados
y Revisión de Trabajos de Graduación

ESCUELA DE INGENIERÍA EN CIENCIAS Y SISTEMAS

UNIVERSIDAD DE SAN CARLOS
DE GUATEMALA

FACULTAD DE INGENIERÌA
ESCUELA DE INGENIERÍA EN
CIENCIAS Y SISTEMAS
TEL: 24767644

El Director de la Escuela de Ingeniería en Ciencias y Sistemas de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer el dictamen del asesor con el visto bueno del revisor y del Licenciado en Letras, del trabajo de graduación ***"ENCONTRANDO FUNCIONES DE REGRESIÓN UTILIZANDO ALGORITMOS GENÉTICOS",*** *realizado por el estudiante, LUIS MAURICIO DÉLEON BARQUERO aprueba el presente trabajo y solicita la autorización del mismo.*

"ID Y ENSEÑAD A TODOS"

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DIRECCION DE INGENIERIA EN CIENCIAS Y SISTEMAS

Ing. Marlon Antonio Pérez Turk
Director
Escuela de Ingeniería en Ciencias y Sistemas

Guatemala, 26 de octubre de 2018

Universidad de San Carlos
de Guatemala

Facultad de Ingeniería
Decanato

DTG. 431.2018

El Decano de la Facultad de Ingeniería de la Universidad de San Carlos de Guatemala, luego de conocer la aprobación por parte del Director de la Escuela de Ingeniería en Ciencias y Sistemas, al Trabajo de Graduación titulado: **"ENCONTRANDO FUNCIONES DE REGRESIÓN UTILIZANDO ALGORITMOS GENÉTICOS",** presentado por el estudiante universitario: **Luis Mauricio Déleon Barquero** y después de haber culminado las revisiones previas bajo la responsabilidad de las instancias correspondientes, autoriza la impresión del mismo.

IMPRÍMASE:

Ing. Pedro Antonio Aguilar Polanco
Decano

UNIVERSIDAD DE SAN CARLOS DE GUATEMALA
DECANO
FACULTAD DE INGENIERIA

Guatemala octubre de 2018.

/echm

ACTO QUE DEDICO A:

Dios	Por guiarme en tiempos de necesidad y darme la vida.
Mis padres	Enma Barquero y Carlos Alvarez por siempre apoyarme en mis estudios y amarme todo el tiempo.
Mi novia	Maria Ana Menzel por mantenerme enfocado y amarme tanto.
Mi hermano	Carlos Guillermo por ser una inspiración de dedicación.
Mis tíos	Hector Barquero y los Alvarez, Luis, Melina, Celina y Betty por estar presentes todo el tiempo y alimentarme tanto el cuerpo como el alma.
Mi abuela	Enma Barquero por siempre enseñarme algo nuevo.
Mis abuelos	En el cielo por mantener su presencia viva en mi corazón.
Mi familia	Por su apoyo incondicional.

AGRADECIMIENTOS A:

Universidad de San Carlos de Guatemala	Por ser mi *alma mater*, proveerme de un sin fin de experiencias y ser un área de libre pensamiento y expresión.
Facultad de Ingeniería	Por mantenerse actualizada y procurar excelencia en cada uno de sus alumnos.
Magister Ingeniero	Estuardo Zapeta por creer en mí y apoyarme.
Señor Carlos José	Por enseñarme que la práctica mantiene la mente afilada.
Mis compañeros de la Facultad	Por todas las horas que compartimos haciendo proyectos juntos.

ÍNDICE GENERAL

ÍNDICE DE ILUSTRACIONES

FIGURAS

TABLAS

LISTA DE SÍMBOLOS

Símbolo	Significado
e	Constante matemática 2,71828…
pi	Constante matemática 3,14159…
$\mathbf{X_n}$	Valores de una variable independiente
$\mathbf{x_n}$	Variable independiente
Y	Valores de la variable dependiente
y	Variable independiente

GLOSARIO

Algoritmo genético Algoritmo basado en la evolución y cruce de las especies.

Bitcoin Moneda virtual.

Cruce Combinación de dos genomas.

Fitness Aptitud, indicador de que tan adaptado esta un organismo.

Función de regresión En estadística o matemática es la función que explica la relación de una variable dependiente con otras variables independientes.

Generación Grupo de genomas independiente, base de la siguiente generación.

Genoma Secuencia de genes que describe una posible solución.

Implementación Creación de un programa.

Inteligencia artificial Rama de la computación que trata de solucionar problemas o hallar optimizaciones.

Interpolación	Calculo de un valor que no es resultado de una medición sino de una aproximación.
JavaScript	Lenguaje de programación.
Mutación	Cambios aleatorios al genoma.
Página web	Herramienta de internet que provee información o servicios.
Población	Conjunto de entidades.
Variable dependiente	En matemática es la variable cuyos valores dependen de los que tomen otras variables.
Variable independiente	En matemática es la variable cuyo valor no depende de otra variable.
WolframAlpha	Página web matemática.

RESUMEN

Aquí se darán a conocer los pasos para la creación de una herramienta que pueda generar una función de regresión, capaz de describir una variable dependiente en base a variables independientes utilizando algoritmos genéticos. Esta herramienta sería un avance que permitiría a gente que no tiene conocimiento de los modelos de regresión estadística, hacer aproximaciones e interpolaciones de manera sencilla. La página web se crearía en JavaScript para aprovechar el poder de procesamiento del usuario y estaría alojada indefinidamente en https://regresion.incodemode.com.

La página seria gratuita y de código abierto, con los siguientes parámetros de entrada: Cero o más listas de valores que incluyen las variables independientes y una lista de valores para la variable dependiente.

Como salidas, una función matemática que trate de explicar la relación de las variables; una o más gráficas que muestren la diferencia entre los valores ingresados y el resultado de la regresión, un desglose y una gráfica de los pasos más importantes del algoritmo genético.

Para comprobar la funcionalidad del sistema se utilizarán funciones conocidas y se comprobará la salida con la función inicial.

OBJETIVOS

General

Verificar que es factible encontrar funciones de regresión utilizando algoritmos genéticos.

Específicos

1. Comprobar que los algoritmos genéticos son una solución viable en el manejo de funciones matemáticas.

2. Crear una herramienta que genere funciones de regresión con base en los datos ingresados.

3. Facilitar el uso de la ciencia a personas ajenas al concepto de matemática de regresión.

4. Utilizar inteligencia artificial, específicamente algoritmos genéticos para extender las capacidades del usuario.

5. Permitir al usuario observar de manera gráfica los datos que está manejando y su aproximación por medio de regresión a una función matemática definida.

6. Dar a conocer al usuario que tanto se aproximan sus datos a la función de regresión.

INTRODUCCIÓN

La inteligencia artificial es una rama de la informática que ha venido a revolucionar el mundo, crea optimizaciones y soluciones a problemas que antes no se consideraban posibles, por ejemplo, el manejo de carros de manera autónoma.

Por otro lado se tiene el área de la regresión matemática, que se encarga de encontrar correlaciones entre datos, para predecir los resultados de valores que no se hayan obtenido por medio de la experimentación, a esto se le llama interpolación.

En este proyecto se pretenden combinar estas dos áreas de la ciencia, para poder crear regresiones por medio de algoritmos genéticos, esto facilitará el acceso a formas más complejas de regresión, a la gente que no tenga un conocimiento matemático tan profundo.

1. MARCO TEÓRICO

1.1. Inteligencia artificial

La inteligencia artificial es una rama de las ciencias de la computación que se dedica a generar algoritmos para que las máquinas parezcan inteligentes, “esta estudia agentes inteligentes, que es cualquier dispositivo que percibe su medio ambiente y toma acciones” [1] Esto lo hace para maximizar su probabilidad de alcanzar objetivos o solucionar problemas. Algunas de las capacidades que tratamos de imitar de los humanos en las máquinas son el aprendizaje y la solución de problemas.

Este documento se centrará solamente en la solución de problemas, específicamente con un algoritmo genético que no tiene conocimiento previo de estructuras matemáticas comunes como $ax+b$ y que no aprende, ni guarda memoria alguna de una ejecución a otra, aunque se busque exactamente la misma función matemática con las mismas variables de entrada al algoritmo.

1.2. Algoritmos genéticos

Es un tipo de algoritmo de búsqueda, del tipo evolutivo, dentro de la inteligencia artificial, este funciona imitando a la forma en la que la genética funciona en la naturaleza.

[1] RUSSELL, Stuart & NORVIG, Peter. *Artificial Intelligence A Modern Approach*. p. 2.

“La idea en estos sistemas es evolucionar una población de posibles soluciones utilizando operadores inspirados en la biología, como la mutación, el cruce de especies y la selección natural.”[2]

Los algoritmos genéticos se diferencian de otros algoritmos de búsqueda porque empiezan con una población de posibles soluciones aleatorias o semi-aleatorias, generalmente cada solución está definida por un genoma que la describe, luego estas soluciones o genomas se cruzan entre sí para crear nuevas soluciones, y luego se seleccionan las mejores soluciones dependiendo de su *fitness* o aptitud. Este proceso de selección y cruce se repite varias veces hasta llegar a una solución aceptable o un número de generaciones específico.

Existen varios tipos de selección y cruce, algunos involucran mutación para agregar la posibilidad de nuevas soluciones, a veces la selección se hace aleatoria y otras depende del *fitness* de cada posible solución, por ejemplo, los *fitness* más altos tienen mayor probabilidad de reproducirse.

Uno de los usos más relevantes es la creación de antenas utilizadas en misiones espaciales de la *NASA* con requerimientos específicos, por ejemplo, cierto tipo de radiación o frecuencia para lo cual no existían otras antenas adecuadas.

1.3. Regresión matemática

La necesidad de una regresión lineal surge cuando tenemos datos que parecen tener relación con otros, por ejemplo, la edad de las personas parece tener relación con su altura. Es definitivo que no es una relación exacta y que varía un poco de una persona a otra, pero en general observamos que la

[2] MITCHELL, Melanie. *An Introduction to Genetic Algorithms.* p. 2.

mayoría de gente va creciendo conforme a su edad y peso y llega a su altura máxima a los 21, luego de los 50 algunos pierden unos centímetros. Vemos una correlación bastante fuerte, y puede ser encontrada una función que se adecue a los datos por medio de una regresión matemática.

Dado que el tema de regresión es muy extenso se plantearon solo dos puntos de mayor importancia para esta tesis:

- La regresión tiene como resultado una función $f(x_1,x_2,..., x_n) + e = r$ siendo f la función con variables independientes x_0, x_1, hasta x_n, una variación e y una variable dependiente r. En el ejemplo, x_1 sería la edad de las personas, x_2 sería el peso y r seria la altura.

  Entonces se diría que la altura puede ser explicada por la edad y el peso de una persona, otra variable independiente podría ser la talla de cintura y para que fuera más exacto se dividiría a la gente según sexo o etnia, pero esto no es pertinente para esta investigación.

- La forma de obtener la función de regresión depende de la minimización de la sumatoria del cuadrado de las diferencias, por ejemplo, si la regresión f(X) es Y = X con valores (X_n, Y_n) (1 1) (2 2) (3 3,5) se tiene su sumatoria de diferencias como:

  $(Y_1 - f(X_1))^2 + (Y_2-f(X_2))^2 + (Y_3-f(X_3))^2$

  $= (1-f(1))^2 + (2-f(2))^2 + (3,5-f(3))^2$

  $= (1-1)^2+(2-2)^2+(3,5-3)^2$

  $= 0,25$

como en el tercer punto Y debiera ser 3 según la función de regresión, pero es 3,5, se tiene al final una sumatoria de diferencias al cuadrado de 0,25 para Y = X.

1.4. Explicación detallada de la implementación

Como se ha mencionado antes, se utilizarán algoritmos genéticos con genomas que representen cada una de las posibles regresiones. Al inicio el usuario debe de introducir uno o más grupos de valores reales y finitos que incluyen:

- Valores de las variables independientes X_n: El usuario puede introducir valores separados por coma de 0 o más variables independientes X_n. Si el usuario no ingresa ninguna variable independiente se creará una automáticamente $X_0 = \{0, 1, 2, \dots , X_{0m}\}$ con valores del 0 hasta m según cantidad de valores haya en la variable dependiente.
- Valores de la variable dependiente Y: Este grupo de valores es obligatorio.

Cuando se ejecuta el algoritmo se generan 100 posibles funciones de regresión o genomas que se obtienen de manera aleatoria, por medio de una función recursiva que tiene las siguientes posibilidades, siendo *RAND* una llamada más a la misma función aleatoria:

- Un operador *op* que puede ser suma, resta, multiplicación, división o potencia:

  RAND op RAND
- Logaritmo:

  log(*RAND* , *RAND*)
- Paréntesis:

  (*RAND*)
- Una función *F* que puede ser seno, coseno, tangente, seno inverso, coseno inverso, tangente inversa o raíz cuadrada:

F(*RAND*)

- Número real o entero seleccionado por una función aleatoria *power law*, que trata de mantener igual probabilidad, de obtener un numero de 1 a 10 como de 11 a 100 o 101 a 1 000 y así para cualquier exponente de 10.

  exp (random(log(max)-log(min)))*min*

  Siendo *min* 1, *max* el mayor número posible de javascript y *random* un numero aleatorio entre 0 y 1.
- Las constantes *pi*, *e, 0 o 1*.
- Cualquiera de las variables independientes x_n.

A estas posibles soluciones se les asigna el *fitness* como una sumatoria de las diferencias entre los valores provistos por el usuario y los valores obtenidos por la posible función de regresión $\sum(f(X_{0i}, X_{1i}, \dots, X_{ni})-Y_i)^2$ y se van insertando en la lista de población de manera ordenada, los que tengan menor *fitness* van de primero, y los de mayor *fitness* van de ultimo.

No todas las funciones generadas por el sistema son utilizadas, ya que cada posible solución debe pasar por un proceso que valida:

- Su longitud no debe ser mayor de 25 caracteres
- Todos los resultados de la función con todas las entradas de variables independientes X_n que ha dado el usuario sean finitos y reales.
- El *fitness* sea un numero finito y real, no imaginario.
- Se simplifica la función a manera de eliminar multiplicaciones por cero, multiplicaciones de números con números, multiplicaciones por uno y cualquier otra cosa que pueda incrementar la complejidad de la expresión matemática sin aportar a la solución final.
- Que la solución no se haya encontrado antes, que no esté repetida.

Este filtro ahorra procesamiento en funciones que desde un principio no son útiles para crear la función de regresión.

Luego de que se tiene una generación con una población de cien posibles soluciones, se pasa a crear la siguiente generación, esto se hace seleccionando las primeras cincuenta soluciones de la generación anterior y luego creando nuevas soluciones, a partir de los genomas de la generación anterior.

A las herramientas encargadas de crear nuevas soluciones, para la siguiente generación a partir de la generación anterior se les llama operaciones de selección, cruce y mutación.

La selección se refiere a la forma en la que se seleccionan los padres para el cruce. Esta se hace asignándole una oportunidad de selección a cada una de las posibles soluciones desde 0,2 hasta 1,2 siendo 1,2 el que tenga *fitness* 0 y 0,2 el que tenga el *fitness* más alto, luego se selecciona de manera aleatoria cada uno de los padres basado en su oportunidad de selección, así los que tengan una oportunidad de selección de 1,2 son más propensos a ser seleccionados que los que tienen 0,2. En otras palabras, mientras más cercano a 0 es el *fitness* ($\sum(f(X_{0i}, X_{1i}, \dots, X_{ni})-Y_i)^2$), de una posible solución hay más posibilidades de que esta sea seleccionada, para crear los genomas de la siguiente generación, a esto se le llama una selección elitista.

$$Oportunidad\ de\ selección = 1,2 - (fitness/maxFitness)$$

El cruce se da luego de seleccionar dos padres. La idea es combinar las dos posibles soluciones para que den como resultado una nueva, al cruce utilizado en esta tesis se le llama cruce de árbol en un punto. La forma más fácil

de entenderlo es que si cada expresión matemática fuese un árbol, al primer árbol se le corta una de sus ramas y luego se le corta una rama al segundo para pegársela al primero. La selección de que ramas se cortan es completamente aleatoria, con la misma probabilidad de suceder desde la raíz hasta las hojas de la expresión matemática.

Por ultimo la mutación, trata de incrementar o reducir el espacio de búsqueda, generando posibles soluciones que con el simple cruce no se encontrarían. Estas tratan de cambiar la expresión matemática de forma aleatoria luego de haber sido cruzada y esto sucede en el 10% de las posibles soluciones. La forma de lograrlo es reemplazando la rama que se iba a poner al primer padre con otra expresión matemática aleatoria como las generadas en la primera generación.

La expresión matemática resultante es tratada como una nueva posible solución, y debe pasar de nuevo el proceso de validación, para poder formar parte de la siguiente generación.

Este proceso de generar nuevas generaciones se repite un máximo de 10 000 veces o hasta alcanzar uno de los criterios de terminación:

- Una posible solución con *fitness* cero: esto indica que se ha encontrado una función de regresión que coincide perfectamente con los valores dados por el usuario.
- Más de 100 generaciones pasadas en las que el *fitness* de los primeros 5 genomas no haya cambiado: esto indica que el algoritmo ya se estancó y que sería más difícil que encuentre una mejor solución.

Al final, cuando concluye el algoritmo se dice que la solución con menor *fitness* de la última generación, es la solución más optima encontrada por el algoritmo. Esto no significa que sea una solución perfecta, y probablemente correr el algoritmo de nuevo de otra solución más o menos óptima. Esto se debe al carácter aleatorio del mismo y no debe ser tomado a la ligera.

A continuación, se presenta un ejemplo práctico, será muy simplificado para dar a entender el funcionamiento del algoritmo.

Si se tuvieran los grupos de valores:

- X_1 = {1 2 3 }
- Y = {1 2 3,5}

Se puede tener una población de posibles soluciones iniciales:

- $f_1(X) = X+3;$
- $f_2(X) = 2X$
- $f_3(X) = X^2$

Con sus respectivos *fitness*:

- $fitness_1 = (1-4)^2+(2-5)^2+(3,5-6)^2 = 24,25$
- $fitness_2 = 11,25$
- $fitness_3 = 34,25$

Y sus valores de oportunidad de selección (VOS) serian:

- $VOS_1 = 1,2 - (24,25 / 34,25) = 1,2 – 0,71 = 0,49$
- $VOS_2 = 0,87$
- $VOS_3 = 0,2$

Luego se elegiría la mitad con el menor *fitness* para la siguiente generación y no se copia el resto, dejando f_2 en la siguiente generación y haciendo cruces entre f_1 con f_2 y f_2 con f_3 de manera elitista.

Y en la siguiente corrida podría quedar luego de la selección, cruce, mutación y eliminación:

- $f_2(X) = 2X$
- $f_4(X) = 2X^2$
- $f_5(X) = X$

Con los respectivos *fitness*

- $fitness_2 = 11,25$
- $fitness_4 = 247,25$
- $fitness_5 = 0,25$

En este caso si el criterio de terminación fuera un fitness menor a 0,5 se podría dar por concluido el ejemplo y decir que f_5 ha sido la mejor función de regresión encontrada por esta ejecución del algoritmo.

1.5. JavaScript

JavaScript es un lenguaje de programación usado en *internet.* Se ha elegido este lenguaje porque corre en casi todos los navegadores que se utilizan a la fecha de la publicación de esta tesis. Este lenguaje se correrá del lado del cliente y se reducirá la carga de procesamiento sobre el servidor que aloje el programa. También por sus cualidades de funcionalidad asíncrona que evitará que el navegador del usuario se bloquee por bucles demasiado tardados. Aparte existe una gran variedad de librerías disponibles para el manejo de gráficas y manejo de la página web, que ayudará a enfocarse en la implementación de los algoritmos.

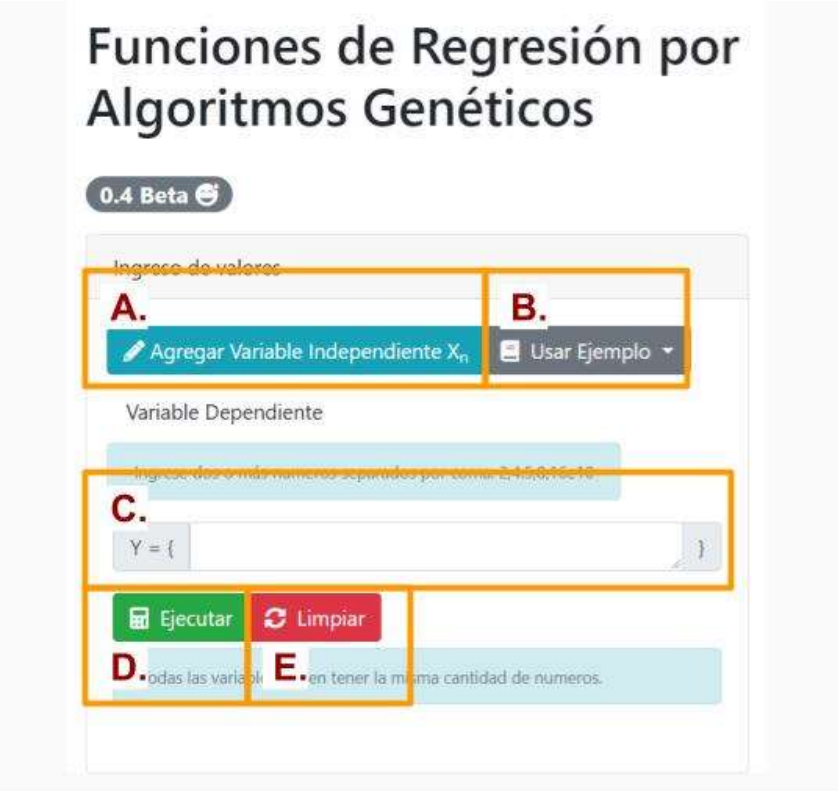
2. MARCO PRÁCTICO

2.1. Manual de usuario

A la fecha de la publicación de esta tesis, la página web se encuentra en: http://regresion.incodemode.com

El formulario de ingreso se ve así:

Figura 1. **Formulario de ingreso**

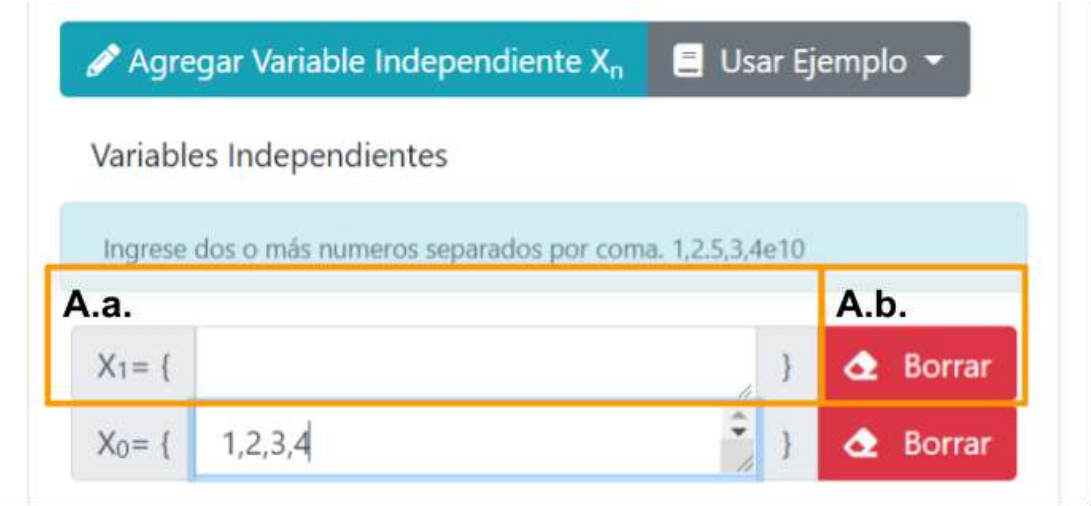


Fuente: elaboración propia, empleando captura de http://regresion.incodemode.com.

Sus partes son:

- A. Agregar una nueva variable independiente: despliega el área de variables independientes y coloca la variable independiente según sea necesario, se debe presionar tantas veces como variables independientes se tengan y las variables serán nombradas desde X_0 hasta X_n.

Figura 2. **Ingreso de variables independientes**



Fuente: elaboración propia, empleando captura de http://regresion.incodemode.com.

- A.a. En el campo de texto se pueden ingresar los valores del grupo X_n separados por coma. Todas las variables ingresadas deben tener la misma cantidad de valores.

- A.b. Las variables independientes X_n se pueden eliminar con el botón “Borrar”.

- B. Usar ejemplo: se pueden seleccionar distintos ejemplos que llenarán automáticamente las variables necesarias para ejecutarse. Esto eliminará cualquier cosa que se haya escrito antes en los campos del formulario, más no cancelará algún algoritmo que esté corriendo en ese momento.

Figura 3. **Usar ejemplo**

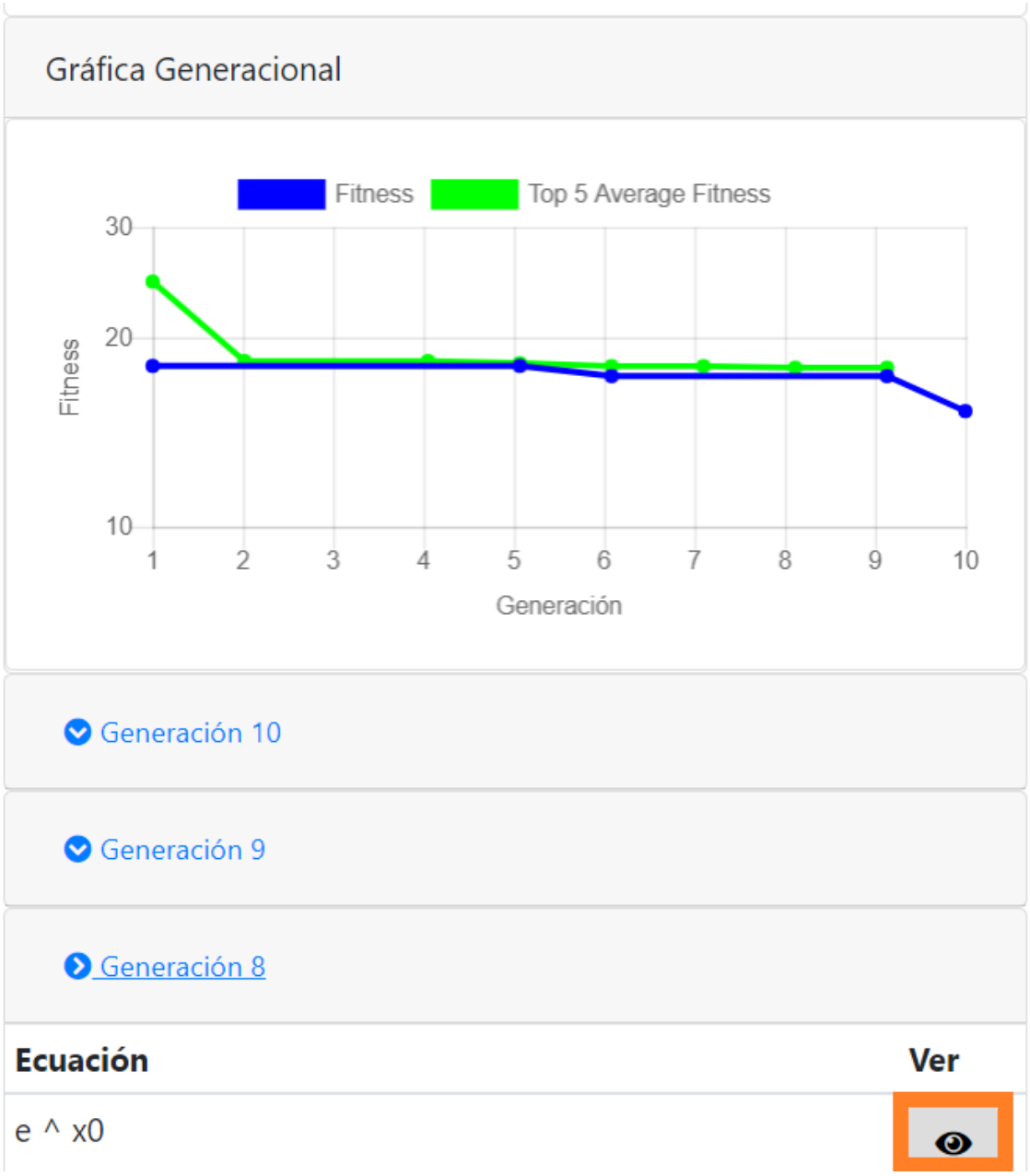
Fuente: elaboración propia, empleando capturas de http://regresion.incodemode.com.

En la figura 3 se puede observar cómo al seleccionar un ejemplo de la lista, se agregan las variables necesarias y se actualizan sus valores.

- C. Variable dependiente: los valores de Y se deben de ingresar separados por comas, y debe haber la misma cantidad de valores que en cualquier otra variable independiente.

- D. Ejecutar: corre el algoritmo genético que se ha declarado en el formulario. Luego de presionar el botón se mostrará una gráfica (Figura 4), que tiene el fitness mínimo de cada generación y la media de los primeros 5 en el eje vertical y en el eje horizontal tiene el número de generación. Más abajo se mostrará un acordeón con las últimas 100 generaciones y dentro de cada generación sus respectivos genomas ordenados por *fitness*.

Figura 4. **Gráfica generacional**

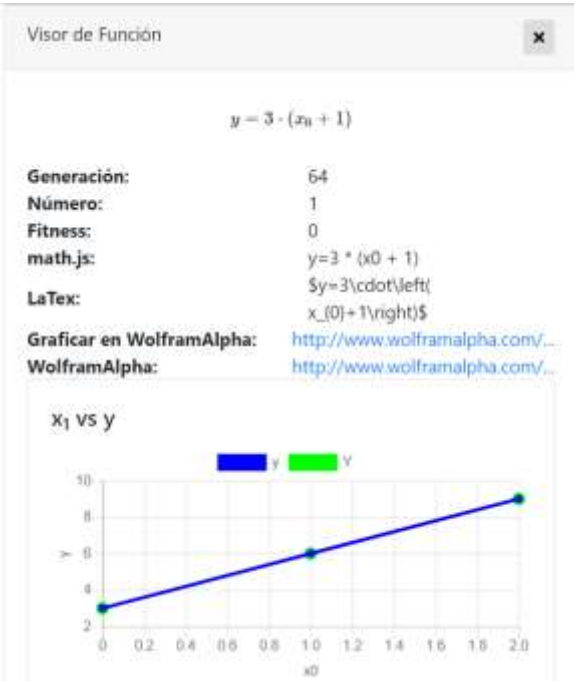


Fuente: elaboración propia, empleando captura de la página.
http://regresion.incodemode.com.

Al presionar el botón de ver, se verá más arriba la descripción del genoma que se desee y esta incluye:

- La ecuación con un diseño presentable
- El número de la generación en la que fue creada la función
- El número según el *fitness* en la generación en la que se decidió ver la función actual.
- La representación utilizada compatible con la librería *math.js.*
- El código en *LaTex* utilizado para dibujar la versión presentable de la ecuación.
- Un enlace a la gráfica que se puede generar en *WolframAlpha*.
- Un enlace a la información que pueda proveer *WolramAlpha*.
- La gráfica de la función.

Figura 5. **Visor de función**



Fuente: elaboración propia, empleando captura de la página. http://regresion.incodemode.com.

Las gráficas tienen una línea azul que indica el genoma que se está estudiando y los puntos verdes, son los valores originales que se metieron en el formulario. Se le puede dar *click* a cualquier punto de las gráficas cuando son varias variables independientes. para seleccionar el valor de X_{ij} en el que se van a mostrar el resto de gráficas.

- E. Limpiar formulario: actualiza la página limpiando por completo el formulario, y deteniendo cualquier algoritmo que se estuviera corriendo.

2.2. Explicación del código

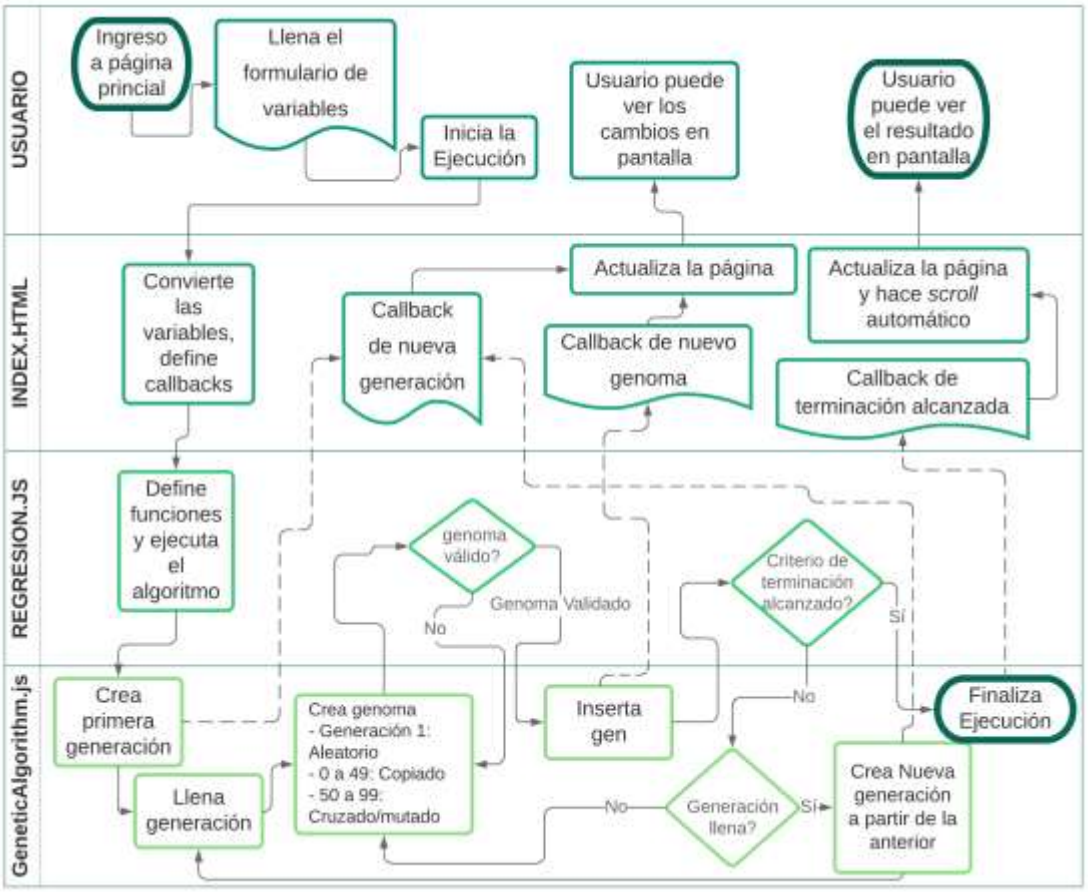
Este programa se basa en una sencilla implementación de algoritmos genéticos sobre *javascript*, algunos de los ciclos han sido convertidos a *setTimeout* con estados, para evitar que el explorador se bloquee mientras se está corriendo el algoritmo.

Toda la carga de procesamiento del algoritmo se lleva a cabo del lado del usuario y el servidor solo sirve los archivos *html, js, css* e imágenes. En ningún momento el servidor debe ejecutar ninguna parte de algoritmo.

Por ser implementado en JavaScript el programa se basa en llamadas a funciones que se pasan de una capa a otra, esto incluye cosas como la función de fitness, la del criterio de terminación y los *callbacks*. Los *callbacks* son funciones que se llaman desde dentro del código, estas tienen acceso al *DOM* y no están embebidas en el algoritmo sino que el programador las crea, estas funciones son llamadas cuando suceden cosas que pueden requerir la actualización de la página.

A continuación, se muestra un resumen de lo que sucede en cada una de las capas que se explicarán más adelante, para hacerlo más sencillo y entendible se han combinado los tres distintos pasos de “Crear genoma” en uno solo.

Figura 6. **Diagrama de flujo**



Fuente: elaboración propia, diagrama de flujo simplificado con las capas utilizadas.

Las tres distintas opciones de crear genoma se refieren a: La primera generación (*status = fillingInitialSetup*); Los primeros 50 genomas de las siguientes generaciones que solo se copian de la generación anterior y por

último el cruce y mutación para los últimos 50 genomas de las siguientes generaciones (*status* = *fillingGenerations*).

Se advierte al lector que el resto de esta explicación está orientado a programadores, para entender mejor esta parte se sugiere haber leído antes la explicación detallada de la implementación, y tener conocimientos básicos de programación.

Las tres capas principales son:

- El algoritmo genético: *public/assets/js/geneticAlgorithm.js*
- La implementación matemática: *public/assets/js/regresion.js*
- La parte visual y de control: *public/index.html*

2.2.1. El algoritmo genético

Esta capa del programa se encarga de hacer el trabajo pesado sin que el navegador se vea afectado y también llama a los *callbacks* que son los encargados de actualizar la página web en la capa visual*. Los únicos callbacks* que se tienen contemplados son:

- Cuando se inicia una nueva generación
  newGenerationStartedCallback(generation)
- Cuando se encuentra un genoma válido
  newGeneFoundCallback(geneObject)
- Cuando se llega al criterio de terminación
  finishCriteriaFoundCallback()

Los pasos a grandes rasgos del algoritmo son:

- Al principio se generan 100 genes aleatorios y validados para formar la primera generación.
- Luego en cada generación subsecuente:
    - Se dejan los mejores 50 genomas de la generación anterior
    - Se seleccionan y cruzan los 100 genomas de la generación anterior de modo elitista hasta tener un total de 100 genomas en la nueva generación.
    - Por cada genoma se verifica el criterio de terminación:
        - Si el mejor genoma tiene un *fitness* de 0 se detiene el algoritmo.
        - Si han pasado 100 generaciones sin cambio de fitness en las primeras 5 soluciones de cada generación se detiene el algoritmo.

El algoritmo se basa en la implementación (*regresión.js*), para tomar las decisiones de selección y cruce. Durante la fase de validación se espera que el genoma pueda cambiar, retornando *false* si es invalido y un objeto de genoma actualizado con su *fitness* cuando sí es válido.

La función *execute()*, es la que tiene el algoritmo principal y funciona llamando a *setTimeout(iterable,1),* para no bloquear el navegador, sus dos estados principales son:

- *“fillingInitialSetup"* Que llena la primera generación
- *“fillingGenerations" Que funciona para las siguientes generaciones*

El *callback* de nuevo genoma encontrado (newGeneFoundCallback), se llama desde la función de inserción (tryPushGene), solamente si el nuevo genoma resulta valido. La función de inserción (*tryPushGene)* trata de insertar el nuevo genoma a la generación actual, pero antes lo valida para verificar que es una expresión matemática válida y que sus valores son números reales y finitos para las variables dadas.

Cada vez que se inicia una nueva generación se llama un *callback (newGenerationStartedCallback) para que este actualice la página según sea necesario.*

En cada pasada del algoritmo se llama a la función que verifica el criterio de terminación (*finishCriteriaTest)*, si esta función devuelve verdadero entonces se llama al *callback de terminación (finishCriteriaFoundCallback), para actualizar la página y enseñarle al usuario el resultado.*

2.2.2. La implementación matemática

Se encarga de utilizar la librería *math.js* para implementar las siguientes funciones:

- *randomize(limit =2)*: es una función recursiva que genera un gen totalmente aleatorio, esta es la base para el resto del algoritmo genético, el parámetro *limit* se refiere a la profundidad máxima, cuando llega a 0 solo puede elegir entre variables y números o constantes. Por ejemplo, para crear una suma se hace:

```
var op1 = randomize(limit -1);
var op2 = randomize(limit -1);
```

return op1+operator+op2;

- *fitnessFunction(geneoma)*: recibe el parámetro del genoma en modo de texto y luego por medio de la librería matemática evalúa, para todos los posibles puntos (X_0, X_1, …, X_n) que ha proveído el usuario para sacar la sumatoria del cuadrado de las diferencias, el resultado es el fitness del genoma.

- *validation(code)*: por medio de la librería matemática hace lo siguiente:

    - Verifica que todos los puntos ingresados puedan ser evaluados en el genoma y den como resultado números reales y finitos.
    - Simplifica la ecuación para evitar resultados repetidos.
    - Verifica que el *fitness* del genoma también sea un número real y finito.
    - Por último, retorna el genoma simplificado o falso si no pasa las verificaciones.

- *equals(genoma1, genoma2)*: verifica si dos funciones son iguales

- *crossover(genoma1, genoma2)*: cruza de manera aleatoria los dos árboles de las ecuaciones en un solo punto, y tiene un 10% de probabilidad de crear una mutación que sobrescriba al extremo obtenido del segundo padre.

- *execute()*: inicializa las variables para el algoritmo genético con sus funciones y lo ejecuta.

- finishCriteriaTest(genomas, currentGeneration): retorna verdadero si y solo si:

    - El *fitness* del primer genoma es 0
    - No ha cambiado el *fitness* de los primeros 5 genomas en las ultimas 100 generaciones.

2.2.3. La parte visual y de control

Se encarga de la interfaz gráfica por medio de los siguientes componentes:

- *Bootstrap*: sirve para mostrar los controles del formulario y área de resultados, que sea vea bien en los celulares (*responsive*) y estiliza la interfaz gráfica.
- *jQuery*: para manejo del *DOM* y eventos de los controles.
- *Graph.js*: para generar las gráficas y sus eventos.
- *FontAwesome.css*: para los iconos.
- *MathJax*: muestra la función matemática encontrada de manera presentable.
- *math.js*: para simplificar y evaluar expresiones matemáticas, también convierte la expresión en *LaTex*, para ser visualizada con *MathJax*.

Esta área también implementa los tres *callbacks* del algoritmo genético e inicializa el código de regresion.js.

2.3. Soporte y solución de problemas

El programa corre en los siguientes navegadores:

- *Chrome*: este es el navegador preferido, sobre todo la última versión
- *Opera*: excepto con "*Data saving*" activado
- *Internet Explorer*: versión 10 o superior (corre más lento que en los otros navegadores).
- *Microsoft Edge.*
- *Firefox.*

Si su navegador no está en esta lista por favor descargue Chrome o Firefox.

Si su navegador se queda trabado por más de un par de minutos, es posible que necesite un procesador más rápido.

Si su navegador se bloquea y le indica que no tiene memoria suficiente para mostrar la página, debe cerrar otros programas que estén ejecutándose u obtener más memoria RAM.

2.4. Requerimientos mínimos

- Debe correr sobre uno de los navegadores descritos anteriormente
- 2 gigas de RAM como mínimo
- 2 núcleos para permitir trabajar mientras se corre el algoritmo tanto en móvil como desktop.
- Se recomienda tener SSD y no correr ningún otro proceso intensivo en el fondo.

2.5. Enlaces

- Código: *https://github.com/incodemode/regresion/tree/TesisV1*
- *Url* del proyecto: *http://regresion.incodemode.com/tesis*

2.6. Datos para pruebas

Los datos utilizados para hacer pruebas se encuentran en el archivo *public/assets/js/possibleVariables.js* y son:

- $y = x_0^2$
- $y = 3\ (x_0 + 1)$
- $y = (x_0 + 2\ )^3$
- $y = x_0 + x_1$
- $y = (x_0 + x_1)^2$
- $y = sqrt(x_0^2 + x_1^2)$
- El precio del *bitcoin* de enero de 2017 a junio de 2018.

Estos valores son accesibles solo por el código a través de la variable *possibleVariables,* ya que algunos valores son generados hasta que se corre el código, la forma más sencilla de verlos es ingresando a la página y ejecutar *console.log(possibleVariables),* en la consola de desarrollador (*F12*), desde *Chrome*.

2.7. Pruebas

La página *public/test.html* contiene el código para correr los posibles ejemplos 30 veces cada uno, siempre respetando la arquitectura de *callbacks* utilizada en la implementación. Cada vez que se llega al criterio de terminación se imprime en pantalla el nombre del ejemplo, la mejor solución encontrada, su

fitness y el número de generaciones antes de la terminación. Los resultados de la ejecución se encuentran en los anexos, pero en la tabla 1 está un resumen para una fácil interpretación de los resultados.

Tabla I. **Resumen de pruebas**

Prueba	**Generaciones promedio**	***Fitness* promedio**	***Fitness* mínimo**	**Resultado *Fitness* mínimo**
$3(x_0+1)$	66	0,00	0,00	$3 * (x_0 + 1)$
x_0+x_1	11	0,00	0,00	$x_1 + x_0$
x_0^2	8	0,00	0,00	x_0^2
$x_0^2+x_1^2$	462	0,21	0,00	$x_1^2 + x_0^2$
$(x_0+2)^3$	446	11,60	0,00	$((x_0 + 3)^2 + 3) * x_0 + 8$
$(x_0^2+x_1^2)^{0,5}$	698	0,43	0,01	$x_1/sqrt(x_0*(pi+e*x_0/x_1)/x_1)+x_0$
***Bitcoin* 2017**	764	2 811 766 000,00	1 480 797 171,00	$32*(2*sqrt(x_0)*sin(sqrt(x_0)+36)*pi+x_0)$

Fuente: elaboración propia, empleando tabla pivote de los resultados generados por la página http://regresion.incodemode.com/test.html.

CONCLUSIONES

1. El uso de algoritmos genéticos es un método viable para encontrar funciones de regresión simples, por ejemplo $(3x_0+1)$, x_0+x_1 o x_0^2, pero requiere de muchas generaciones y puede llegar a soluciones subóptimas con funciones más complejas como la distancia entre dos puntos, $(x_0+2)^3$ o el precio del *Bitcoin*.

2. El sistema sí encuentra una función de regresión específica con un resultado de *fitness* cercano a 0. Aunque esta no sea exactamente la esperada es matemáticamente equivalente para los datos dados.

3. El sistema se ha simplificado lo más posible, para que cualquiera pueda utilizarlo, y se han utilizado librerías que mejoran la usabilidad, como *bootstrap* y *graph.js*.

4. Este algoritmo puede encontrar una aproximación subóptima del precio del *Bitcoin* y puede ayudar a extender las capacidades de un usuario, sin conocimiento matemático previo.

5. Las gráficas utilizadas sí le permiten al usuario entender que tanto se aproxima la función de regresión a sus datos.

6. El usuario puede saber que tanto se aproxima la función de regresión a sus datos por medio del *fitness,* que se muestra en el resumen de la función y las gráficas proporcionadas.

RECOMENDACIONES

1. Los parámetros del algoritmo como el criterio de terminación y el número de genes deben de ser incrementados en el caso de buscar una función muy compleja, para evitar una convergencia temprana en una solución no óptima.

2. Este algoritmo podría verse beneficiado de una implementación multiprocesador que corra fuera del explorador, por ejemplo, para crear varios posibles genomas en paralelo para hacerlo más rápido.

3. Implementar, según los autores Ms. Shikha Malik y Mr. Sumit Wadhwa, *DGCA* (*Dynamic Genetic Clustering Algorithm*). Este tipo de implementación ha demostrado evitar la convergencia en soluciones subóptimas que es el mayor problema de esta tesis. *DGCA* es una variación de algoritmos genéticos en la que se agrupan los genomas según sus rasgos, para mantener varias especies conviviendo al mismo tiempo, evitando así que la convergencia prematura y sobrepoblación subsecuente de una única especie afecte la diversidad genética de las posibles soluciones.

BIBLIOGRAFÍA

1. Especificación de ECMAScript. [en línea]. <http://www.ecma-international.org/publications/standards/Ecma-262.htm>. [Consulta: 1 de agosto de 2018].

2. GESTAL, Marcos; RIVERO, Daniel; RABUÑAL, Juan Ramón; DORADO, Julián y PAZOS, Alejandro. *Introducción a los Algoritmos Genéticos y la Programación Genética*. España: Consorcio Editorial Galego, 2010. 76p.

3. HORNBY, Gregory S. y GLOBUS, Al. *Automated Antenna Design with Evolutionary Algorithms*. [en línea]. <http://alglobus.net/NASAwork/papers/Space2006Antenna.pdf>. [Consulta: 29 de agosto de 2018].

4. MALIK, Shikha & WADHWA, Sumit. *Preventing Premature Convergence in Genetic Algorithm Using DGCA and Elitist Technique*. [en línea]. <https://pdfs.semanticscholar.org/db76/78845db91e5cdabbd2767de11c1e85eae6ae.pdf>. [Consulta: 19 de agosto de 2018].

5. MITCHELL, Melanie. *An Introduction to Genetic Algorithms.* 1ra ed. Cambridge, England: MIT Press, 1998. 158p.

6. RUSSELL, Stuart J. y NORVIG, Peter. *Artificial Intelligence: A Modern Approach.* 1ra ed. Upper Saddle River, New Jersey, USA: Prentice Hall, 1995. 932p.

7. TUSELL, Fernando. *Análisis de Regresión. Introducción Teórica y Práctica basada en R*. Bilbao, España: 2011. [en línea]. <http://www.et.bs.ehu.es/~etptupaf/nuevo/ficheros/estad3/nreg1.pdf>. [Consulta: 12 de julio de 2018].

APÉNDICES

Apéndice 1. **Resultados de $3(x_0+1)$**

genes	fitness	generation
3 * (x0 + 1)	0	100
3 * (x0 + 1)	0	72
3 * (x0 + 1)	0	47
3 * (x0 + 1)	0	54
3 * (x0 + 1)	0	20
3 * (x0 + 1)	0	106
3 * (x0 + 1)	0	86
3 * (x0 + 1)	0	32
3 * (x0 + 1)	0	52
3 * (x0 + 1)	0	82
3 * (x0 + 1)	0	36
3 * (x0 + 1)	0	37
3 * (x0 + 1)	0	49
3 * (x0 + 1)	0	110
3 * (x0 + 1)	0	55
3 * (x0 + 1)	0	67
3 * (x0 + 1)	0	25
3 * (x0 + 1)	0	29
3 * (x0 + 1)	0	151
3 * (x0 + 1)	0	72
3 * (x0 + 1)	0	67
3 * (x0 + 1)	0	61
3 * (x0 + 1)	0	172
3 * (x0 + 1)	0	44
3 * (x0 + 1)	0	67
3 * (x0 + 1)	0	96
3 * (x0 + 1)	0	50
3 * (x0 + 1)	0	40
3 * (x0 + 1)	0	30
3 * (x0 + 1)	0	61

Fuente: elaboración propia, obtenida ejecutando la página http://regresion.incodemode.com/test.html. Consulta: septiembre de 2018.

Apéndice 2. **Resultados de x_0+x_1**

genes	fitness	generation
x1 + x0	0	24
x0 + x1	0	15
x1 + x0	0	19
x1 + x0	0	5
x1 + x0	0	19
x0 + x1	0	13
x1 + x0	0	10
x1 + x0	0	2
x0 + x1	0	6
x0 + x1	0	7
x0 + x1	0	10
x1 + x0	0	6
x1 + x0	0	11
x1 + x0	0	14
x1 + x0	0	18
x0 + x1	0	11
x0 + x1	0	5
x0 + x1	0	8
x1 + x0	0	12
x1 + x0	0	7
x0 + x1	0	14
x1 + x0	0	8
x0 + x1	0	5
x1 + x0	0	7
x1 + x0	0	22
x1 + x0	0	7
x0 + x1	0	13
x1 + x0	0	21
x1 + x0	0	16
x1 + x0	0	8

Fuente: elaboración propia, obtenida ejecutando la página http://regresion.incodemode.com/test.html. Consulta: septiembre de 2018.

Apéndice 3. **Resultados de x_0^2**

genes	fitness	generation
x0 ^ 2	0	6
x0 ^ 2	0	10
x0 ^ 2	0	7
x0 ^ 2	0	4
x0 ^ 2	0	4
x0 ^ 2	0	3
x0 ^ 2	0	6
x0 ^ 2	0	3
x0 ^ 2	0	7
x0 ^ 2	0	4
x0 ^ 2	0	17
x0 ^ 2	0	10
x0 ^ 2	0	16
x0 ^ 2	0	4
x0 ^ 2	0	3
x0 ^ 2	0	5
x0 ^ 2	0	7
x0 ^ 2	0	12
x0 ^ 2	0	6
x0 ^ 2	0	9
x0 ^ 2	0	8
x0 ^ 2	0	7
x0 ^ 2	0	7
x0 ^ 2	0	12
x0 ^ 2	0	10
x0 ^ 2	0	5
x0 ^ 2	0	6
x0 ^ 2	0	3
x0 ^ 2	0	12
x0 ^ 2	0	12

Fuente: elaboración propia, obtenida ejecutando la página http://regresion.incodemode.com/test.html. Consulta: septiembre de 2018.

Apéndice 4. **Resultados de $x_0^2 + x_1^2$**

Genes	**fitness**	**generation**
2 * x0 * x1 + x1 * atan(x1 / x0) ^ (e * x0) / e	0,1421183066	643
(2 * x0 + x1 / ((1 / x1) ^ 2 * x0 ^ e * e ^ e)) * x1	0,1629745631	1 224
x1 ^ 2 + x0 ^ 2	0,0000000000	56
x1 ^ 2 + x0 ^ 2	0,0000000000	27
2 * x1 * x0 + x1 * atan(x1 / (2 * x0)) ^ e	0,2279298051	803
sqrt(sqrt(2 * x1 + atan(x1 ^ x1))) * x1 * x0 + x1	0,1149651343	578
(x0 * atan(x0) + sqrt(x1 * atan(20 ^ x0))) * x1	0,0881542238	773
x1 ^ 2 + x0 ^ 2	0,0000000000	36
x0 ^ 2 + x1 ^ 2	0,0000000000	305
x1 * (2 * x0 - x0 * tan(pi) + 0,23298494159392624)	1,1403963210	250
2 * x1 * x0 + x1 * (1 - x0 / x1)	0,0000000000	522
x1 ^ 2 + x0 ^ 2	0,0000000000	51
x1 * (2 * x0 + cos(atan(x0)) * cos(e - 8))	0,8735846803	433
2*(x0*x1+log(x1 ^ (e–0,1425465430742778),5 917))	1,0933700760	616
x0 ^ 2 + x1 ^ 2	0,0000000000	31
x1 * (2 * x0 - atan(x0) + atan(x1) ^ atan(e + x1))	0,2572133768	564
(2 * x0 - sqrt(x0) + sqrt(x1)) * x1 + 1 / pi ^ pi	0,0332433948	687
x0 ^ 2 + x1 ^ 2	0,0000000000	71
2 * x1 * x0 + cos((x0 / x1) ^ (4 * pi) ^ 16) ^ 128	0,0000000000	552
x1 * (x0 * sqrt(e) + log(x1, e))	0,3860128793	330
2 * x0 * x1 - x0 + x1	0,0000000000	306
sqrt(1,5707963267948966 * sqrt(x1)) * x1 * x0 + x1	0,1732569388	566
x0 * x1 * sqrt((x1 + atan(atan(x1))) / x0 + e)	0,1152531820	404
x1 ^ 2 + x0 ^ 2	0,0000000000	40
x1 * (sqrt(sqrt(x1 * sqrt(pi * x0)) * x0) + x0)	0,1473099369	534
(acos(cos(x0)) + x0 + 1) * x1 - x0	0,0000000000	554
x1 * 2 / cos(sin(x1 / e) ^ x0 ^ x0 / x0) * x0	0,1706494041	712
x1 * (2 * x0 - cos(sqrt(sin(sin(x1) + x0) + e)))	0,4016096245	539
(sqrt(sqrt(x1 * sqrt(pi * x0)) * x0) + x0) * x1	0,1473099369	623
2 * x0 * x1 + x1 / ((2 ^ x0 + x0 ^ 2 + x0) / x1)	0,4750000000	1 032

Fuente: elaboración propia, obtenida ejecutando la página http://regresion.incodemode.com/test.html. Consulta: septiembre de 2018.

Apéndice 5. **Resultados de $(x_0^2 + x_1^2)^{0.5}$**

Genes	fitness	generation
(x0 + x1 + log(x0, sin(1 / (e + 2)) ^ e)) * 2 / e	0,07001987147	1 112
x0 / (cos(e ^ (4 - x1) + tan(cos(-x1))) + 2) + x1	0,23765830540	916
log(x1 ^ atan(pi), pi) ^ atan(1,4142135623730951 ^ x1) + x0	1,34935197800	449
sqrt(x0 + 0,1411200080598672) * sqrt(2 * x1)	0,09786423496	681
x1 ^ 2 / e / x0 ^ 1 + x0 / log(2, x0) / 81 + x0	0,09510107101	1 152
x1 / sqrt(x0 * (pi + e * x0 / x1) / x1) + x0	0,01393610600	610
x0 + (e * x1 - x0) / (acos(tan(e)) + 2)	0,05569339035	509
sqrt(2 * x1 * x0) + 1,5707963267948966 / e ^ e	0,09741743480	433
sqrt(2 * x0 + cos(atan(atan(x1) + 2))) * sqrt(x1)	0,09702898754	424
log(2 ^ x1 ^ atan(x1), 2 * x0 + sqrt(e ^ x0)) + x0	0,08471780953	653
atan(sqrt(cos(sqrt(cos(cos(pi)))))) * (x0 + x1)	0,03441621624	442
x0 / (x0 * 1,5707963267948966 / -(2 * x1) + pi) + x1	0,22549070280	762
atan(pi ^ (pi / atan(x0))) / 2 * x0 - x1 / pi + x1	0,04078585135	1 037
log(x1 ^ sqrt((cos(x0) + x1) / sqrt(x0)), e) + x0	0,86587603810	438
x0 + (x1 - cos(sin(x0) / atan(x1))) ^ sqrt(e) / x0	0,24358439860	1 267
sqrt(pi * x1) - atan(2 ^ x1 ^ 4) + sqrt(x1 * x0)	0,64844014350	731
x0 / (atan(2 * e + x0 / x1) + (x1 / x0) ^ 2) + x1	0,02353578176	740
atan((cos(3 * cos(x0)) + 2) ^ x0 + 2) * x1	2,19056098300	890
(atan(x1 / x0) / e + x0 / x1) * x0 / pi + x1	0,03484334216	801
x0 - cos(x1 / pi) * sin(sqrt(x0 / e)) + sqrt(x1)	0,94649862820	390
atan(atan(pi + 1,7320508075688772) + e + 3) * sqrt(x1 * x0)	0,14514080450	871
(x1 + sqrt(x1) / e) / e + sqrt(x0 * sqrt(x1 * x0))	0,09739709803	942
x0 / (cos(cos(2 / x1 + x1)) ^ 4 + 2) + x1	0,27340218400	433
(x1 / pi / x0 ^ 1 + 1) / (pi / x1) + x0	0,40013327490	253
x1 + (x0 * log(x0, x1 + sin(cos(x0 ^ e))) + 1) / e	0,23307095950	1 000
atan(sqrt(x0 + 4) * x0 / (x1 + 1)) * (x1 + 1)	0,82088932680	642
(atan(atan(e) ^ x0 / 2) + sqrt(x0)) * sqrt(x1)	0,13904776230	955
sqrt(log(x1 ^ (x1 - cos(x1) + -sqrt(x0)), e)) + x0	0,83783714850	485
log(x1 ^ (e - 1), pi) + x0	2,57510150800	172
x1 + atan(2 ^ (1 / (x1 / x0)) ^ e) / (e / x0)	0,02699241970	738

Fuente: elaboración propia, obtenida ejecutando la página http://regresion.incodemode.com/test.html. Consulta: septiembre de 2018.

Apéndice 6. **Resultados de $(x_0 + 2)^3$**

Genes	fitness	generation
((pi ^ 2 + 1) * x0 + 2 * pi) * x0 + e + 6	2,528138794	567
((x0 + 1) ^ e + atan(atan(x0)) + sin(x0) + 2) * e	1,161141630	579
((x0 + 3) ^ 2 + 3) * x0 + 8	0,000000000	171
-5 - e + -x0 + (x0 + e) ^ e + x0 ^ acos(cos(e))	3,687470481	630
(x0 + sqrt(x0) + sqrt(atan(pi))) ^ e + pi	15,257325130	355
(x0 + 2) ^ 3	0,000000000	42
(e + x0) ^ e - (e - x0 * (x0 - 1)) * pi	4,530761127	577
(e * (x0 + 1)) ^ 2 - 2 * x0 * cos(-(x0 * cos(x0)))	1,862482407	501
(sqrt(sqrt(x0 ^ x0) + 2) + pi + 2) * (x0 + 1) ^ 2	1,623329202	806
(pi - cos(atan(x0 ^ atan(tan(x0) + pi))) + x0) ^ e	3,440451231	1 232
(x0 + 2) ^ 3	0,000000000	573
(x0 + 2) ^ 3	0,000000000	443
sqrt(x0 + 2) ^ 6 + x0 ^ (e + x0) * tan(tan(pi))	1,42000E-28	646
sqrt(x0) * x0 * acos(cos(tan(pi) + e)) ^ pi	103,635045800	546
x0 * (cos(pi ^ x0) + 31,50685070984081) + x0 ^ x0	71,301265450	429
(x0 + 2) ^ 3	0,000000000	119
((x0 + 1) ^ e + atan(atan(x0)) + sin(x0) + 2) * e	1,161141630	790
((x0 + 1) ^ e + sin(x0) + 2) * e + sqrt(x0)	0,922800841	492
2 * pi + (x0 * pi + sqrt(sqrt(x0 + sqrt(e)))) ^ 2	2,720970216	566
(atan(atan(e)) * x0 + atan(pi)) ^ e * pi	18,199203690	407
(x0 + 2) ^ 3	0,000000000	62
(x0 + sqrt(e)) ^ e * sqrt(e + 1)	1,820428697	524
(cos(tan(2 ^ e)) * x0 + 1) ^ 2 * (2 * e + x0)	14,394907870	593
e * x0 * ((x0 + 1) * pi + e)	74,601461920	386
(x0 + 2) ^ 3	0,000000000	43
(x0 * e + atan(pi ^ pi ^ pi ^ 4)) * (x0 + 1) * pi	21,321659310	400
(x0 + 2) ^ 3	0,000000000	117
(x0 + e) ^ e - e + -(x0 + 2) + -3 + x0 ^ e	3,687470481	554
(x0 - cos(pi) + 1) ^ 3	0,000000000	140
(x0 + 2) ^ 3	0,000000000	100

Fuente: elaboración propia, obtenida ejecutando la página http://regresion.incodemode.com/test.html. Consulta: septiembre de 2018.

Apéndice 7. **Resultados del precio del *bitcoin* de 2017**

genes	fitness	generation
2,1071487177940904 * (e ^ e + x0 + 2 * pi) * e ^ e	2 870 906 195	714
(e + 1) * (x0 + pi * sqrt(pi) + 17) * e * pi	2 870 799 024	903
12 * e * (pi * (pi + 2) + x0 + 1)	2 873 749 080	258
pi ^ 3 * (e + x0 + sqrt(x0 + sin(x0) + 22) + 11)	2 866 827 480	1 065
3 * (e + 8) * (x0 + 20)	2 871 404 651	369
4 * (2 * pi + e ^ e + x0 + 1) * e * sqrt(pi * e)	2 870 797 998	553
(x0 * (sin(sin(x0 ^ 25)) / 8 + 1) + 25) * 3 ^ pi	2 776 455 643	873
31,43407822282072 * (x0 + e ^ pi + 1,5152978215491797)	2 871 257 583	543
32 * (x0 + 21) + tan(sqrt(2 * x0 + pi + 58))	2 832 428 737	838
pi ^ 3 * (x0 + sqrt(370,5707963267949 - x0) + 15)	2 863 808 125	1 032
3 * (e + 8) * (x0 + 20)	2 871 404 651	541
(2 * e) ^ 2 * (sqrt(3 * e + x0 + 22) * e + x0)	2 861 874 957	1 004
3 ^ pi * (sqrt(sqrt(25 * (x0 + 13))) + x0 + 16)	2 868 958 089	1 566
(pi ^ pi - 4) * (x0 - sin(pi ^ x0) + 18) + 2	2 868 820 637	609
7 * (pi + 1,4056476493802699) * (e ^ e + x0 + 7)	2 870 815 349	859
pi ^ 3 * (x0 + tan((x0 + 31) * x0) + 27)	2 833 987 686	1 042
pi ^ 3 * ((sqrt(x0) * x0) ^ (3 - e) + x0 + 19)	2 867 831 025	659
(e + 2) * (e + 4) * (x0 + e ^ pi + cos(x0))	2 870 796 773	753
32 * ((2 * e + 1) * e + x0 + sqrt(sqrt(x0)))	2 870 749 900	446
pi ^ 3 * (x0 + sqrt(x0 + 29) + tan(x0 ^ 13) + 12)	2 780 361 367	1 346
32 * (2 * sqrt(x0) * sin(sqrt(x0) + 36) * pi + x0)	1 480 797 171	946
sqrt(3 * pi) ^ 3 * (3 * (sqrt(x0) + 1) + x0)	2 862 272 695	398
(e ^ pi + x0 + sin(x0)) * 3 ^ pi - 4 * tan(x0 ^ 2)	2 800 201 960	1 134
3 ^ pi * (e ^ pi + cos(x0 ^ pi ^ e * e) + x0)	2 866 232 488	465
(sin(3 * 2 ^ x0) + x0 + pi ^ e) * pi ^ 3 + x0	2 866 437 210	737
32 * (sqrt(sqrt(sqrt(sqrt(e ^ e)))) + 3 ^ e + x0)	2 871 014 961	539
(pi * e + e ^ pi) * (x0 + cos(sqrt(x0) * pi) + 23)	2 860 723 165	690
pi ^ 3 * ((1 / pi) ^ pi + 1) * (pi ^ e + x0)	2 870 940 445	422
3 ^ pi * (3 ^ e + x0) + sqrt(2 * 3 ^ (e + 1) * x0)	2 869 310 013	1 064
32 * (3 * (e + 4) + x0 + 0,8414709848078965)	2 871 014 958	564

Fuente: elaboración propia, obtenida ejecutando la página http://regresion.incodemode.com/test.html. Consulta: septiembre de 2018.