Examen Final

November 28, 2025

1 Temas Selectos: Cómputo Evolutivo

- Entrega: 28 de noviembre
- Semestre: 2026-1
- Estudiante: Rodrigo S. Cortez Madrigal

1.1 ¿Qué es el genoma en los seres vivos?

El genoma es el conjunto completo de material genético de un organismo, que incluye todos sus genes y secuencias de ADN. Contiene la información necesaria para el desarrollo, funcionamiento y reproducción de un ser vivo.

1.2 ¿Cuáles son las principales fuentes de variación genética en los sistemas biológicos?

Las principales fuentes de variación genética en los sistemas biológicos son:

- Recombinación genética:
  - Durante la meiosis, los cromosomas homólogos pueden intercambiar segmentos de ADN, lo que genera nuevas combinaciones de genes.
  - En los elementos genéticos móviles, como los transposones, que pueden moverse dentro del genoma y causar variación genética.
- Mutaciones: Cambios en la secuencia del ADN que pueden suceder durante la replicación, reparación o recombinación. Las distinguimos por:
  - Espontáneas
    - Espontaneas: Ocurren de manera natural durante la replicación del ADN.
    - Inducidas: Causadas por factores externos como radiación o agentes químicos.
  - Tipo de célula afectada:
    - Somaticas (afectan células no reproductivas) o germinales (afectan células reproductivas y pueden ser heredadas).
    - Germinales: Afectan las células reproductivas y pueden ser heredadas por la descendencia.
  - Estructura del ADN:
    - Puntuales: Afectan una sola base del ADN (sustituciones, inserciones, eliminaciones).
    - Cromosómicas: Afectan grandes segmentos de cromosomas (eliminaciones, duplicaciones, inversiones, translocaciones).

1.3 ¿Cómo se define el genotipo, el fenotipo y cuales es la relación entre ambos en los sistemas biológicos?

El genotipo se refiere a la composición genética de un organismo, es decir, la información que posee en sus genes y puede heredarse. El fenotipo, por otro lado, es la manifestación observable de ese genotipo, incluyendo características físicas, bioquímicas y comportamentales.

La relación entre genotipo y fenotipo es que el genotipo determina el potencial genético de un organismo, mientras que el fenotipo es el resultado de la interacción entre ese genotipo y el ambiente en el que se desarrolla el organismo.

1.4 Cuál es la relación genotipo-fenotipo en los sistemas artificiales?

En algoritmos evolutivos la relación genotipo-fenotipo existe en codificaciones (genotipos) utilizadas para representar posibles soluciones (fenotipos) en un espacio de búsqueda.

Un claro ejemplo es la codificiación de los algoritmos genéticos, en donde un genotipo puede ser una cadena de bits o una estructura de datos que representa una potencial solución a un problema. Cuando el genotipo se decodifica, produce un fenotipo evaluable que puede ser una posible solución. Aquí el genotipo es clave ya que los operadores genéticos actúan en el espacio codificado para explorar el espacio de soluciones optimizandolo y obteniendo así mejores fenotipos.

1.5 ¿Qué es un gen?

Un gen es una unidad básica de información genética, compuesto por una secuencia específica de ADN que codifica la información necesaria para la síntesis de proteínas o ARN. Cada gen ocupa una posición específica en un cromosoma y puede existir en diferentes formas llamadas alelos, que contribuyen a la variabilidad genética dentro de una población.

El genoma esta organizado en cromosomas, que son estructuras formadas por ADN y proteínas. Un cromosoma tiene distintas copias de un gen y heredamos un cromosoma de cada padre, por lo que tenemos cuatro copias de cada gen a los que llamamos alelos.

1.6 ¿Qué es una mutación neutral en sistemas naturales y artificiales?

Una mutuación neutral es aquella que no tiene un efecto significativo en la aptitud o función del organismo o sistema en el que ocurre.

- En sistemas naturales una mutación neutral no afecta la aptitud del organismo, por lo que puede persistir en la población sin ser seleccionado y eliminado. Estas mutaciones pueden acumularse con el tiempo y contribuir a la variabilidad genética sin influir directamente en la adaptación del organismo.
- En sistemas artificiales una mutación neutral es aquella que no altera el fitness del individuo. Estas mutaciones pueden ocurrir en partes del código o en parámetros que no afectan la funcionalidad principal del individuo, permitiendo que el sistema mantenga su desempeño mientras se explora el espacio de soluciones. En algoritmos como Programación Genética, una mutación neutral puede ayudar a mantener la diversidad genética en la población sin comprometer la calidad de las soluciones generadas.

1.7 ¿Qué es un intrón?

Un intrón es una región (secuencia de nucleótidos) dentro de un gen que no codifica para proteínas y que es removida durante el proceso de maduración del ARN mensajero (ARNm) en organismos eucariotas.

Durante la transcripción del ADN a ARN, tanto los exones (secuencias codificantes) como los intrones son transcritos al ARN precursor. Posteriormente, en un proceso llamado *splicing*, los intrones son eliminados y los exones se unen para formar el ARNm maduro, que luego se traduce en una proteína. No obstante los intrones pueden tener un rol regulador y contribuir a la diversidad genética a través de mecanismos como el empalme alternativo.

En sistemas artificiales como en la Programación Genética, los intrones son representados como partes del código que no afectan la funcionalidad del programa, pero que pueden influir en la estructura y evolución del mismo. Un claro ejemplo es cuando cuando una rama del árbol de un programa no contribuye a la salida final, como en el siguiente ejemplo:

```
*
/ \
0 (x - 3)
```

1.8 Definir deriva génica.

La deriva génica es un mecanismo de evolución caracterizado por cambios aleatorios para un alelo en una población a lo largo del tiempo. A diferencia de la selección natural, que favorece alelos beneficiosos, la deriva génica ocurre debido a eventos fortuitos. Estos cambios pueden llevar a la pérdida o fijación de alelos en la población, independientemente de su impacto en la aptitud del organismo.

La deriva génica tiene un efecto significativo en poblaciones pequeñas, donde los efectos aleatorios pueden impactar prinunciadamente en la composición genética de la población. Este proceso puede reducir la diversidad genética y contribuir a la diferenciación entre poblaciones aisladas.

En sistemas artificiales, la deriva génica puede observarse en algoritmos evolutivos donde, debido a la selección aleatoria y la reproducción, ciertas soluciones pueden volverse predominantes o desaparecer sin que esto esté relacionado con su calidad o desempeño.

1.9 Definir el concepto de esquema en AG.

Un esquema es una plantilla o patrón que representa un subconjunto de soluciones dentro del espacio de búsqueda. Identifica un subconjunto de cadenas con similitudes en determinadas posiciones de la cadena.

Un esquema se define mediante una cadena de caracteres que puede incluir valores específicos y símbolos comodín (generalmente representados por $'*'$) que pueden tomar cualquier valor (don't care).

Por ejemplo, en una representación binaria, el esquema $'1*0*'$ representa todas las cadenas de longitud 4 que comienzan con '1', tienen '0' en la tercera posición y pueden tener cualquier valor en las posiciones segunda y cuarta. Los esquemas son importantes porque permiten analizar cómo ciertos patrones de genes (o características) se propagan a través de generaciones en una población, ayudando a entender la dinámica de la evolución en los algoritmos genéticos.

1.10 ¿Qué nos dice el teorema del esquema en AG y PG?

El teorema del esquema, propuesto por John Holland, establece que los esquemas con ciertas características tienden a proliferar en una población a lo largo de las generaciones en algoritmos genéticos (AG) y programación genética (PG).

La longitud definitoria de un esquema, denotada como $L(H)$, mide la distancia entre las posiciones fijas más externas de la plantilla. específicamente, el teorema indica que los esquemas con una longitud definitoria más corta son menos susceptibles a la interrupción por operadores genéticos como la cruza y la mutación. Como resultado, los esquemas cortos se consideran más robustos y tienen más probabilidades de sobrevivir y propagarse a la siguiente generación.

Esto significa que estos esquemas beneficiosos se propagan a través de la población, lo que contribuye a la mejora continua de las soluciones generadas por el algoritmo. El teorema del esquema proporciona una base teórica para entender cómo los algoritmos evolutivos exploran y explotan el espacio de soluciones.

1.11 Definir los conceptos de orden y longitud de un esquema en AG.

El orden de un esquema se refiere al número de posiciones específicas (no comodines) que contiene el esquema. Por ejemplo, en el esquema $'1*0*'$ el orden es 2, ya que hay dos posiciones específicas ('1' y '0'). La longitud de un esquema es la distancia entre la primera y la última posición específica en el esquema. En el esquema $'1*0*'$, la longitud es 3, ya que la primera posición específica está en la posición 1 y la última en la posición 3.

1.12 Definir los conceptos de orden y longitud de un esquema en PG.

En Programación Genética (PG):

- El orden de un esquema se refiere al número de nodos específicos (no comodines) que contiene el esquema dentro de un árbol de programa.
- Longitud fija: La longitud de un esquema es la distancia entre el nodo más alto y el nodo más bajo que contienen valores específicos dentro del árbol del programa.
- Longitud variable: La longitud variable de un esquema se refiere al número total de nodos (tanto específicos como comodines) que contiene el esquema dentro de un árbol de programa.
- Logitud: $ D(inst(h,H)„ H) = D_{fixed}(H) + D_{var}(inst(h,H)„ H) $

1.12.1 Dada la siguiente población binaria, determinar el número de cadenas que contienen el esquema:

```
H = [(011,2), (110,7)]
I1 = [100111000000]
I2 = [001100110011]
I3 = [101100110110]
I4 = [011011011011]
I5 = [110110110110]
```

[1]:
```
import re
```

```python
# Podemos usar regular expressions para contar las cadenas que contienen los␣
 ↪esquemas.

H = [('011',2), ('110',7)]

I1 = '100111000000'
I2 = '001100110011'
I3 = '101100110110'
I4 = '011011011011'
I5 = '110110110110'

def evaluar_esquema(H, individuo):
    resultados = []
    for esquema, posicion in H:
        idx = posicion - 1
        patron = esquema.replace('*', '.*')
        subcadena = individuo[idx:idx+len(esquema)]
        cumple = re.fullmatch(patron, subcadena) is not None
        resultados.append((esquema, posicion, cumple))
    return resultados

print('-'*40)
for individuo in [I1, I2, I3, I4, I5]:
    resultados = evaluar_esquema(H, individuo)
    print(f'Individuo: {individuo}')
    for esquema, posicion, cumple in resultados:
        if cumple:
            print(f'- Cumple el esquema {esquema} en la posición {posicion}')
        else:
            print(f'- NO cumple el esquema {esquema} en la posición {posicion}')
    print('-'*40)
```

```
----------------------------------------
Individuo: 100111000000
- NO cumple el esquema 011 en la posición 2
- NO cumple el esquema 110 en la posición 7
----------------------------------------
Individuo: 001100110011
- Cumple el esquema 011 en la posición 2
- Cumple el esquema 110 en la posición 7
----------------------------------------
Individuo: 101100110110
- Cumple el esquema 011 en la posición 2
- Cumple el esquema 110 en la posición 7
----------------------------------------
Individuo: 011011011011
- NO cumple el esquema 011 en la posición 2
- NO cumple el esquema 110 en la posición 7
```

```
----------------------------------------
Individuo: 110110110110
- NO cumple el esquema 011 en la posición 2
- Cumple el esquema 110 en la posición 7
----------------------------------------
```

1.13 Del problema anterior, si no se considera la posición.

- a) Cuántos individuos contienen el esquema?
- b) Cuántas instancias del esquema hay en la población?

[2]:
```python
H = [('011',2), ('110',7)]

# Contemos cada esquema en cada individuo.

def contar_(H, individuo):
    conteos = []
    for esquema, _ in H:
        patron = esquema.replace('*', '.*')
        matches = re.findall(patron, individuo)
        conteos.append((esquema, len(matches)))
    return conteos

for individuo in [I1, I2, I3, I4, I5]:
    conteos = contar_(H, individuo)
    print(f'Individuo: {individuo}')
    for esquema, conteo in conteos:
        print(f'- El esquema {esquema} aparece {conteo} veces')
    print('-'*40)
```

```
Individuo: 100111000000
- El esquema 011 aparece 1 veces
- El esquema 110 aparece 1 veces
----------------------------------------
Individuo: 001100110011
- El esquema 011 aparece 3 veces
- El esquema 110 aparece 2 veces
----------------------------------------
Individuo: 101100110110
- El esquema 011 aparece 3 veces
- El esquema 110 aparece 3 veces
----------------------------------------
Individuo: 011011011011
- El esquema 011 aparece 4 veces
- El esquema 110 aparece 3 veces
----------------------------------------
Individuo: 110110110110
- El esquema 011 aparece 3 veces
```

```
- El esquema 110 aparece 4 veces
--------------------------------------
```

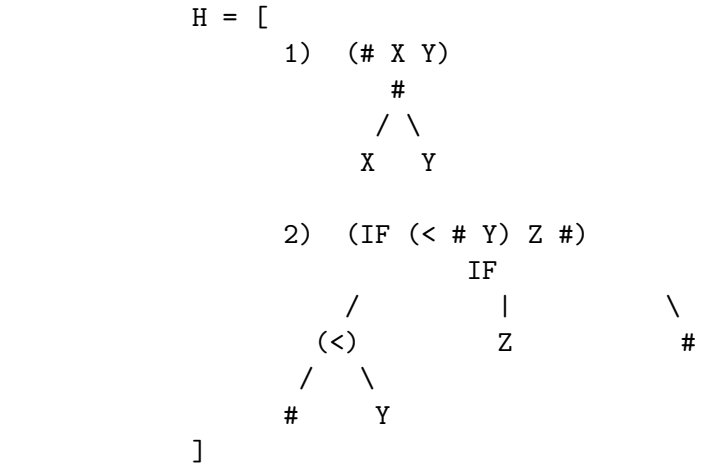
1.14 Definir el concepto de instanciación.

La instanciación es el proceso de asignar valores específicos a las variables de un esquema creando una instancia concreta del mismo. Especificamente en AG y PG, la instanciación implica tomar un esquema general con comodines o variables y reemplazarlos con valores específicos para generar un individuo particular que se pueda evaluar como una solución.

1.15 Definir orden y longitud fija del siguiente esquema: H = [(# X Y) (IF (< #Y) Z #)].

El esquema H = [(# X Y) (IF (< #Y) Z #)] tiene las siguientes características:

Dibujemos el grafo correspondiente al esquema:



- Orden: El orden del esquema depende del número de nodos específicos (no comodines) que contiene. En este caso, el orden es 6.
- Longitud fija: La longitud fija del esquema es 3, ya que hay dos niveles en el árbol del esquema: el nivel raíz y el nivel de los nodos hijos.

1.16 Dada la siguiente población formada por tres individuos, ¿Cuántas instancias se tienen del esquema anterior?

- $1 = (+(*X(IF(<YZ)ZCOS(Z)))(*XY))$
- $2 = (+(*(\%XY)(COS(\%XY)))(IF(<ZY)ZCOS(Z)))$
- $3 = (IF(<ZY)(*(+XY)Z)(IF(<COS(Z)Y)Z(*XY)))$

Para determinar cuántas instancias del esquema $H = [(\#XY)(IF(<\#Y)Z\#)]$ hay en la población dada, debemos buscar coincidencias en cada individuo.

- En el individuo 1, no hay ninguna instancia del esquema porque falta el segundo fragmento
- En el individuo 2, hay dos instancias. (# X Y) aparece dos veces (una directa y otra dentro de COS) con (IF (< Z Y) Z COS(Z)).
- En el individuo 3, hay dos instancias. (+ X Y) y (* X Y) en el else en (IF(< COS(Z) Y) Z (*X Y))

1.17 Definir la longitud para cada instancia de la población anterior.

- Para el individuo 1, no hay instancias del esquema, por lo que la longitud es 0.
- Para el individuo 2, hay dos instancias del esquema:
  - Primera instancia: (# X Y) dentro de COS. $D_{\text{var}} = 2 + 1 = 3$ Entonces la longitud total es $D = D_{\text{fixed}} + D_{\text{var}} = 3 + 3 = 6$.
  - Segunda instancia: (IF (< Z Y) Z COS(Z)). $D_{\text{var}} = 3 + 1 = 4$ Entoces la longitud total es $D = D_{\text{fixed}} + D_{\text{var}} = 3 + 4 = 7$.
- Para el individuo 3, hay dos instancias del esquema:
  - Primera instancia: (+ X Y) en el else en (IF(< COS(Z) Y) Z (*X Y)). $D_{\text{var}} = 2 + 1 = 3$ Entonces la longitud total es $D = D_{\text{fixed}} + D_{\text{var}} = 3 + 3 = 6$.
  - Segunda instancia: (* X Y) en el else en (IF(< COS(Z) Y) Z (*X Y)). $D_{\text{var}} = 1 + 0 = 1$ Entonces la longitud total es $D = D_{\text{fixed}} + D_{\text{var}} = 3 + 1 = 4$.

1.18 ¿Qué nos dice la teoría de la evolución neutral?

La teoría de la evolución neutral, propuesta por Motoo Kimura, sostiene que la mayoría de las variaciones genéticas en las poblaciones son el resultado de mutaciones neutrales que no afectan la aptitud del organismo.

Según esta teoría, la deriva genética (en lugar de la selección natural) es el principal motor de la evolución a nivel molecular. Esto implica que muchas mutaciones se fijan en una población simplemente por azar y no porque singinfiquen una ventaja adaptativa. Como corolario, implica que gran parte de la diversidad genética observada en las poblaciones puede explicarse sin recurrir a la selección natural.

Los individuos con mutaciones neutrales tienen la misma probabilidad de sobrevivir y reproducirse que aquellos sin dichas mutaciones, lo que lleva a cambios aleatorios en la frecuencia de los alelos en la población a lo largo del tiempo.

1.19 ¿Qué es el bloating y qué ocurre cuando se presenta?

El bloating es un fenómeno que ocurre en algoritmos evolutivos, sobre todo en GP, cuando los individuos tienden a crecer en tamaño sin una mejora correspondiente en su rendimiento o aptitud. Esto hace que los programas evolucionados sean innecesariamente complejos y difíciles de probar, lo que puede afectar negativamente la eficiencia del algoritmo.

Cuando se presenta el bloating, los recursos computacionales se desperdician en evaluar y manipular individuos grandes y complejos, lo que puede dificultar la convergencia hacia soluciones óptimas. Para mitigar el bloating, se pueden implementar técnicas como la penalización por tamaño o la limitación de la profundidad de los árboles de programa.

1.20 ¿Cuál es el tamaño del espacio de búsqueda para un AG binario, AG entero, AG real y PG?

El tamaño del espacio de búsqueda para los algoritmos genéticos (AG) y la programación genética (PG) varía en función de la representación y los parámetros utilizados.

Algoritmoa Genéticos:

- Binario: El tamaño del espacio es todas las combinaciones posibles de bits, es decir, 2^n, donde n es la longitud del genoma.
- Entero: El tamaño del espacio de búsqueda depende del rango de valores enteros dentro fuera de las restricciones y la longitud de la cadena. Si cada gen puede tomar valores entre a y b, y la longitud de la cadena es n, entonces el tamaño del espacio de búsqueda es $(b - a + 1)^n$.
- Real: El tamaño del espacio de búsqueda es en teoría infinito, porque los genes pueden tomar cualquier valor real dentro de un rango definido.

Programación Genética:

El tamaño del espacio de búsqueda en programación genética depende de la cantidad de funciones y terminales disponibles, así como de la profundidad máxima permitida para los árboles de programa. Si hay f funciones y t terminales, y la profundidad máxima es d, el tamaño del espacio de búsqueda es todas las combinaciones posibles de árboles que se pueden formar con esas funciones y terminales hasta la profundidad d.

1.20.1 ¿Cuáles los mecanismos de explotación y exploración en AG, PSO, ED y ACO.

AG (Algoritmos Genéticos):

- Explotación: Tiene que ver con la selección de los mejores individuos para reproducirse. Principalmente utilizando operadores como selección y cruza, pero también el elitismo para preservar las mejores soluciones.
- Exploración: Al introducir diversidad. Principalmente mediante mutaciones aleatorias, la cruza podria ser si consideramos que puede combinar individios muy aptos con otros menos aptos para explorar nuevas áreas del espacio de búsqueda. En general cualquier mecanismo que permita la introducción de nuevas variaciones en la población, preservando la diversidad genética y evitando la convergencia prematura.

PSO (Particle Swarm Optimization):

- Explotación: Las partículas ajustan sus posiciones en función del mejor personal y global.
- Exploración: Las partículas exploran nuevas áreas del espacio gracias a la influencia de la velocidad y la aleatoriedad en sus movimientos, precisamente con la inercia que permite que las partículas se desplacen más allá de sus mejores posiciones conocidas. En clase tambié mencionamos la idea de vecindarios locales que ayudan a tener como distintos cúmulos de partículas explorando diferentes regiones del espacio de búsqueda.

ED (Evolución Diferencial):

- Explotación: Utiliza la combinación de individuos existentes para generar nuevos, enfocándose en las mejores soluciones usando una selección más determinista.
- Exploración: Introduce variabilidad mediante la mutación y la recombinación.

ACO (Ant Colony Optimization):

- Explotación: Las hormigas siguen las rutas con mayor concentración de feromonas, reforzando así las soluciones exitosas.
- Exploración: Las hormigas exploran nuevas rutas al azar, lo que permite descubrir soluciones alternativas y evitar la convergencia prematura.

1.21 De acuerdo al equilibrio de Hardy-Weinberg, qué ocurre en una población binaria si:

- a) Pc = 100%, sin mutación, sin selección: La población permanecerá estable con las frecuencias alélicas constantes a lo largo del tiempo, ya que no hay fuerzas evolutivas actuando sobre ella. $p_{t+1} = p_t$ Se mantiene el equilibrio original (H–W) porque no cambian las frecuencias de 0 y 1.
- b) Pm = 100%, sin cruza, sin selección: Esto quiere decir que cada bit es *flippeado* en cada generación. $p_{t+1} = 1 - p_t$ La población entoces experimentará un cambio oscilatorio en las frecuencias alélicas, alternando entre la frecuencia original y su complemento en cada generación. Lo que claramente dice que no se alcanza un equilibrio estable.
- c) Pm = 50%, sin cruza, sin selección: Quiere decir que cada bit tiene .5 de probabilidad de *flippearse*. La población es forzada a un equilibrio de 50% ceros y 50% unos, pero la mutación empuja la frecuencia hacia el azar. Es un equilibrio, sí, pero no es el equilibrio de Hardy-Weinberg, ya que la mutación está alterando constantemente las frecuencias alélicas.

1.22 ¿Qué significa evolución y cómo se define evolución natural?

La evolución es el proceso mediante el cual las especies cambian a lo largo del tiempo debido a la acumulación de variaciones genéticas.

La evolución natural (biológica) se define como el mecanismo por el cual los organismos mejor adaptados a su entorno tienen una mayor probabilidad de sobrevivir y reproducirse, transmitiendo sus características favorables a las generaciones futuras. A pesar de que se entendía a la selección natural como el principal motor de la evolución, hoy en día se reconoce que la evolución es un proceso de múltiples componentes. Entre ellos la deriva genética, la mutación (relacionada por supuesto a la teoría neutral), la selección natural y el flujo génico. Permitiendo así en la diversidad biológica y la adaptación de las especies a sus entornos cambiantes.

1.23 De acuerdo a la serie de tiempo proporcionada y utilizando Programación Genética, encontrar un modelo que ajuste adecuadamente los datos. Proporcionar:

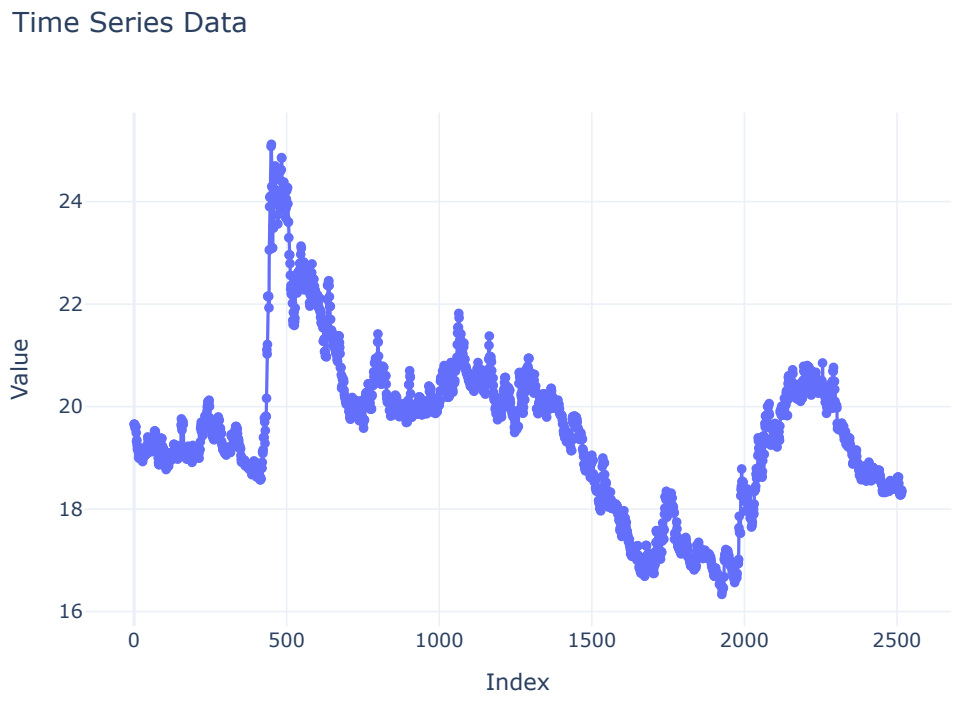
- Información del conjunto de primitivas (funciones y terminales).
- Parámetros utilizados en la ejecución del algoritmo.
- Gráfica comparativa de los datos reales v.s. los datos obtenidos por el mejor modelo de Programación Genética.
- Error cuadrático medio (MSE) del mejor modelo encontrado.
- El modelo utilizado para la gráfica comparativa.
- Pronosticos de los valores 2518, 2519 y 2520 de la serie de tiempo utilizando el mejor modelo encontrado.

[3]:
```
import math
import operator
import numpy as np
import pandas as pd
from deap import gp, base, creator, tools, algorithms
import plotly.graph_objs as go
```

```
import functools
import networkx as nx
import matplotlib.pyplot as plt

# Plotly render svgs
import plotly.io as pio
pio.renderers.default = "svg"
```

[4]:
```
clase = pd.read_csv('data/examen_datos.csv', header=None, index_col=False)
clase = clase.iloc[:, 0].copy()
clase = clase.astype(float)

time_series = clase.values
```

[5]:
```
# Plot time series data
fig = go.Figure()
fig.add_trace(go.Scatter(x=np.arange(len(time_series)), y=time_series,␣
 ↪mode='lines+markers', name='Data'))
fig.update_layout(title='Time Series Data', xaxis_title='Index',␣
 ↪yaxis_title='Value', template='plotly_white')
fig.show()
```



[6]:
```python
# Protected primitive functions

def protectedDiv(a, b):
    return a / b if abs(b) > 1e-9 else a

def protectedLog(a):
    a = float(a)
    return math.log(abs(a)) if abs(a) > 1e-9 else 0.0

def protectedSqrt(a):
    return math.sqrt(abs(a))

def protectedExp(a):
    # Establecer límites para evitar overflow
    a = max(min(a, 50), -50)
    return math.exp(a)

def protectedPow(a, b):
    try:
        if abs(a) > 1e6 or abs(b) > 8:
            return 1.0
        return float(pow(a, int(b)))
    except Exception:
        return 1.0

def evalSymbolic(individual, toolbox_sym, time_series):
    func = toolbox_sym.compile(expr=individual)
    try:
        errors = ((func(x) - y)**2 for x, y in enumerate(time_series))
        import math
        return (math.fsum(errors),)
    except (ValueError, OverflowError, ZeroDivisionError):
        return (1e10,)
```

[7]:
```python
# Funciones y terminales
pset_sym = gp.PrimitiveSet("MAIN", 1)  # x
pset_sym.renameArguments(ARG0='x')

# Operadores aritméticos
pset_sym.addPrimitive(operator.add, 2)
pset_sym.addPrimitive(operator.sub, 2)
pset_sym.addPrimitive(operator.mul, 2)
pset_sym.addPrimitive(protectedDiv, 2)

# Trig
pset_sym.addPrimitive(math.sin, 1)
pset_sym.addPrimitive(math.cos, 1)
```

```
pset_sym.addPrimitive(math.tan, 1)
pset_sym.addPrimitive(math.asin, 1)
pset_sym.addPrimitive(math.acos, 1)
pset_sym.addPrimitive(math.atan, 1)

# Log, Exp, Sqrt, Pow
pset_sym.addPrimitive(protectedLog, 1)
pset_sym.addPrimitive(protectedExp, 1)
pset_sym.addPrimitive(protectedSqrt, 1)
pset_sym.addPrimitive(protectedPow, 2)
#pset_sym.addPrimitive(abs, 1)

# Constantes
pset_sym.addTerminal(math.pi, name="pi")
pset_sym.addTerminal(math.e, name="e")
pset_sym.addEphemeralConstant("rand", functools.partial(np.random.uniform, -1,␣
 ↪1))

#  Clases Fitness / Individuo
try:
    creator.FitnessMinSymb
except AttributeError:
    creator.create("FitnessMinSymb", base.Fitness, weights=(-1.0,))
try:
    creator.IndividualSymb
except AttributeError:
    creator.create("IndividualSymb", gp.PrimitiveTree, fitness=creator.
 ↪FitnessMinSymb)

#  Toolbox
toolbox_sym = base.Toolbox()
toolbox_sym.register("expr", gp.genHalfAndHalf, pset=pset_sym, min_=1, max_=3)
toolbox_sym.register("individual", tools.initIterate, creator.IndividualSymb,␣
 ↪toolbox_sym.expr)
toolbox_sym.register("population", tools.initRepeat, list, toolbox_sym.
 ↪individual)
toolbox_sym.register("compile", gp.compile, pset=pset_sym)

# Main

toolbox_sym.register("evaluate", functools.partial(evalSymbolic,␣
 ↪toolbox_sym=toolbox_sym, time_series=time_series))
toolbox_sym.register("select", tools.selTournament, tournsize=5)
toolbox_sym.register("mate", gp.cxOnePoint)
toolbox_sym.register("expr_mut", gp.genHalfAndHalf, min_=0, max_=4)
toolbox_sym.register("mutate", gp.mutUniform, expr=toolbox_sym.expr_mut,␣
 ↪pset=pset_sym)
```

```
# Limitar altura de los árboles
toolbox_sym.decorate("mate", gp.staticLimit(key=operator.attrgetter("height"),␣
 ↪max_value=128))
toolbox_sym.decorate("mutate", gp.staticLimit(key=operator.
 ↪attrgetter("height"), max_value=128))
```

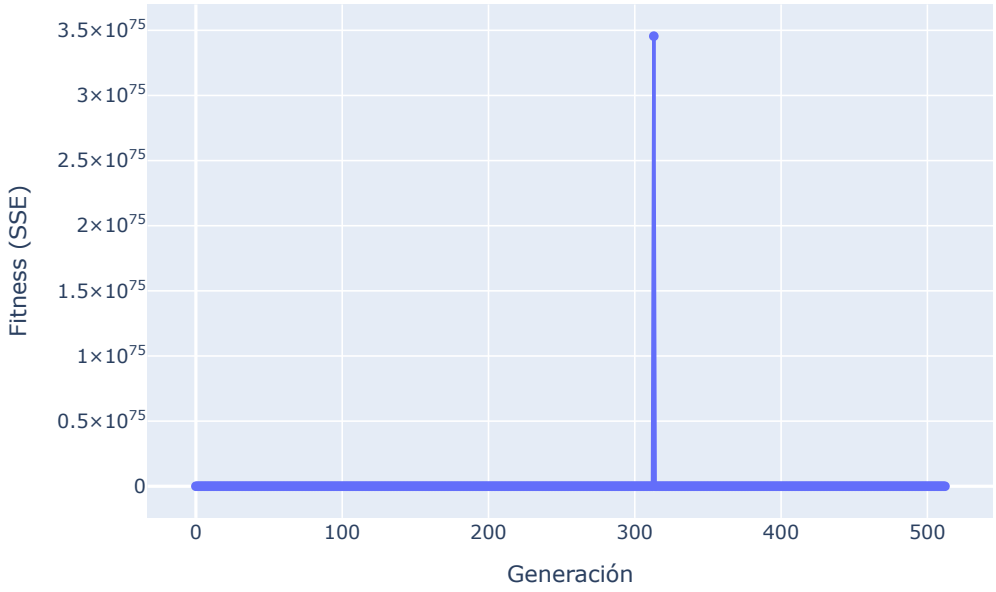
[ ]:
```
# Parámetros
POP_SIZE = 1024
N_GEN = 512
CXPB = 0.7 # Probabilidad de cruce
MUTPB = 0.3 # Probabilidad de mutación
np.random.seed(42)

# Estadísticas multi: fitness y tamaño
hof = tools.HallOfFame(1)

# Estadística sobre el primer valor del fitness
stats_first_fit = tools.Statistics(key=lambda ind: ind.fitness.values[0])
stats_fit = tools.Statistics(lambda ind: ind.fitness.values)
stats_size = tools.Statistics(len)

# MultiStatistics con fitness, tamaño y primer valor del fitness
mstats = tools.MultiStatistics(
    fitness=stats_fit,
    size=stats_size,
    first_fit=stats_first_fit
)

mstats.register("avg", np.mean)
mstats.register("std", np.std)
mstats.register("min", min)
mstats.register("max", max)

pop = toolbox_sym.population(n=POP_SIZE)

pop, log = algorithms.eaSimple(pop, toolbox_sym, cxpb=CXPB, mutpb=MUTPB,␣
 ↪ngen=N_GEN,
                               stats=mstats, halloffame=hof, verbose=__debug__)
```

[ ]:
```
main_df = pd.DataFrame(log)
for chapter in log.chapters:
    chapter_df = pd.DataFrame(log.chapters[chapter])
    chapter_df = chapter_df.add_prefix(f"{chapter}_")
    main_df = pd.concat([main_df, chapter_df], axis=1)

import pickle
```

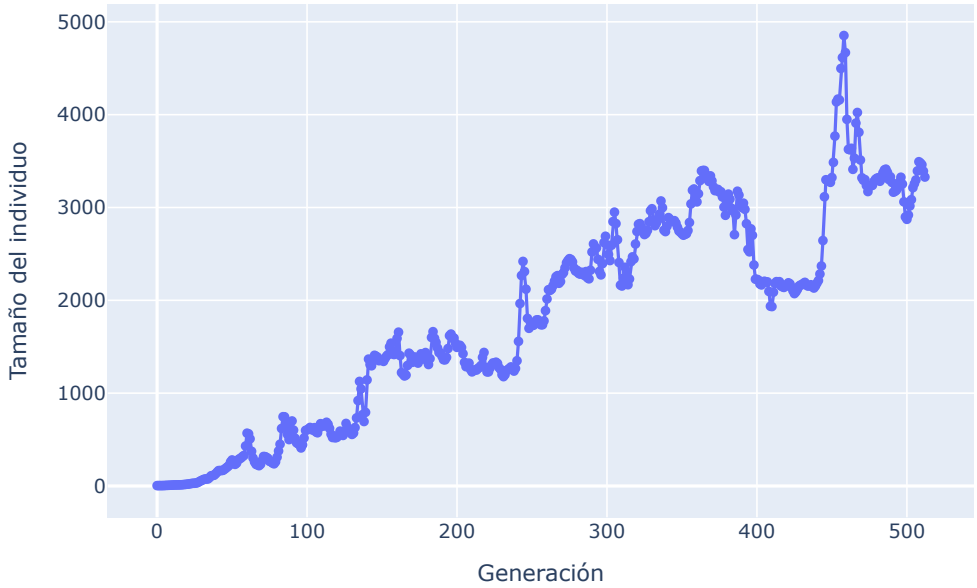
```
# Guardar el individuo (árbol)
with open('best_ind.pkl', 'wb') as f:
    pickle.dump(best, f)
```

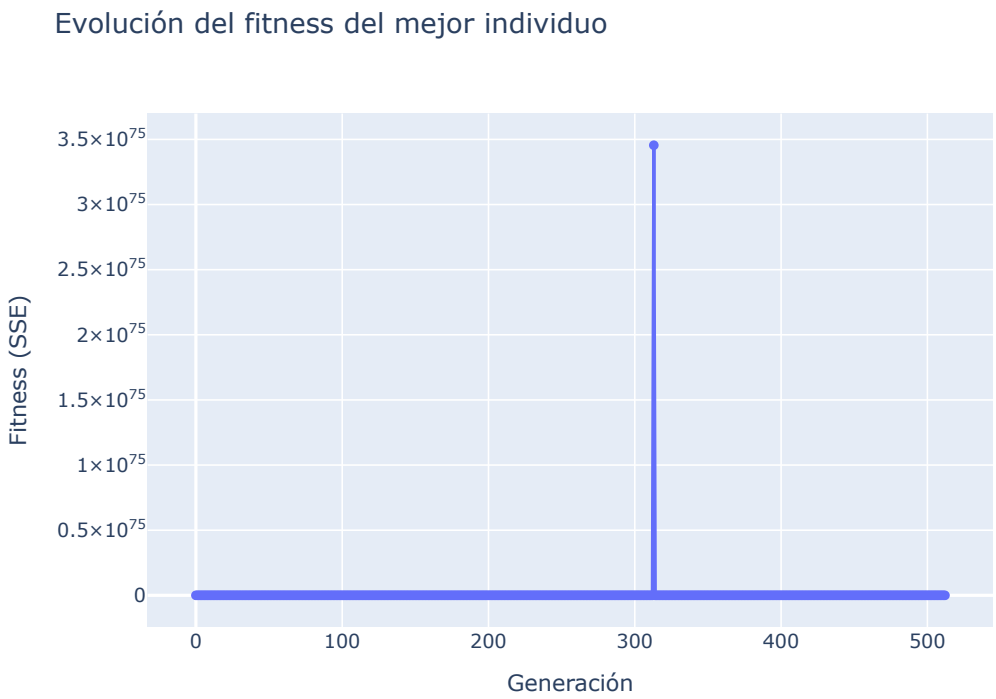
[11]:
```
fig = go.Figure()
fig.add_trace(go.Scatter(y=main_df['fitness_avg'], mode='lines+markers',␣
 ↪name='F1 Min'))
fig.update_layout(title='Evolución del fitness promedio',
                  xaxis_title='Generación',
                  yaxis_title='Fitness (SSE)')
fig.show()

fig = go.Figure()
fig.add_trace(go.Scatter(y=main_df['size_avg'], mode='lines+markers', name='F1␣
 ↪Min'))
fig.update_layout(title='Evolución del tamaño promedio',
                  xaxis_title='Generación',
                  yaxis_title='Tamaño del individuo')
fig.show()

fig = go.Figure()
fig.add_trace(go.Scatter(y=main_df['first_fit_avg'], mode='lines+markers',␣
 ↪name='F1 Min'))
fig.update_layout(title='Evolución del fitness del mejor individuo',
                  xaxis_title='Generación',
                  yaxis_title='Fitness (SSE)')
fig.show()
```

Evolución del fitness promedio



Evolución del tamaño promedio



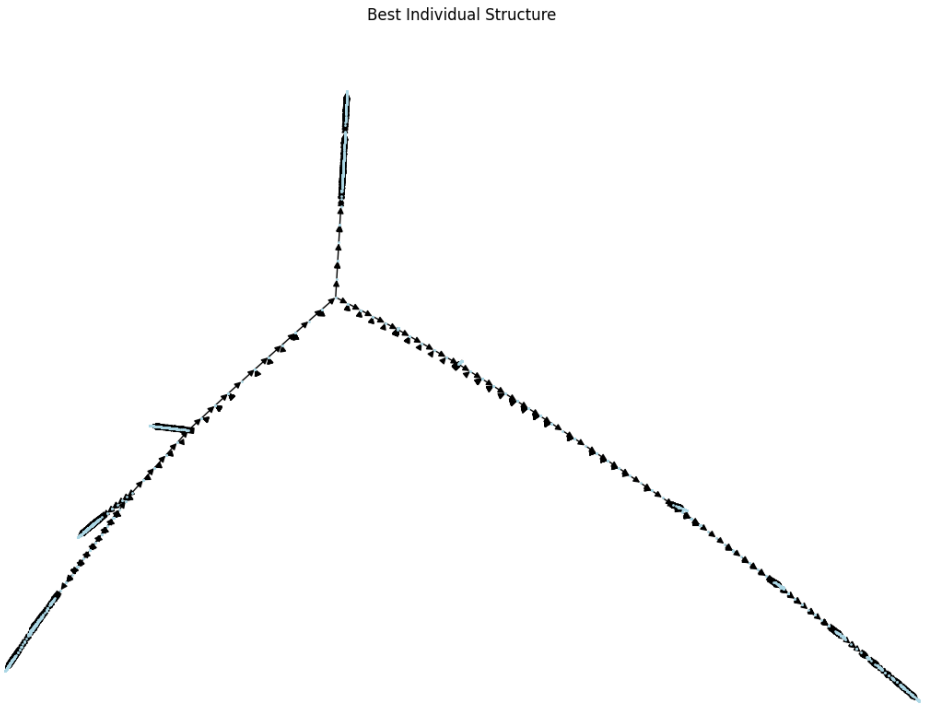


Mejor Solución Encontrada

[13]:
```
# Cargar el individuo y recompilar la función
with open('best_ind.pkl', 'rb') as f:
    best_loaded = pickle.load(f)
```

[14]:
```
# Obtener nodos del mejor individuo
nodes, edges, labels = gp.graph(best_loaded)

labels = {k: (f"{v:.2f}" if isinstance(v, float) else v) for k, v in labels.
 ↪items()}

G = nx.DiGraph()
G.add_nodes_from(nodes)
G.add_edges_from(edges)

pos = nx.spectral_layout(G)

plt.figure(figsize=(12, 8))
```

```
nx.draw(G, pos, with_labels=False, arrows=True, node_size=1,␣
 ↪node_color='lightblue', font_size=1)
#nx.draw_networkx_labels(G, pos, labels)
plt.title("Best Individual Structure")
plt.show()
```



[19]:
```
best_loaded = hof[0]
print("Mejor individuo:", best_loaded)
print("Altura:", best_loaded.height)
print("Error cuadrático (SSE):", best_loaded.fitness.values[0])


func_best = toolbox_sym.compile(expr=best_loaded)
```

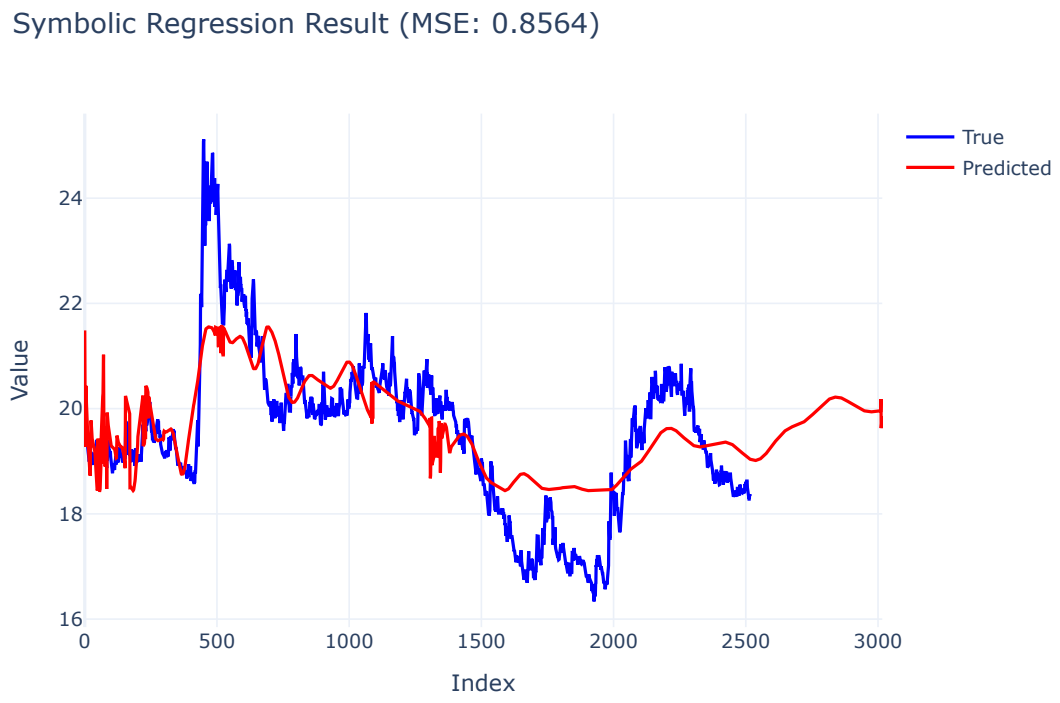
```
Mejor individuo: sub(add(tan(sin(add(add(cos(sin(pi)), mul(cos(mul(mul(pi, pi),
protectedExp(sub(protectedPow(protectedDiv(protectedPow(pi, pi),
add(protectedDiv(protectedPow(mul(cos(mul(cos(mul(mul(pi, pi),
protectedExp(sub(protectedPow(protectedDiv(sin(e),
add(protectedDiv(protectedPow(pi, pi), add(protectedSqrt(x), protectedExp(e))),
protectedDiv(sin(mul(pi, -0.007267505597527313)), mul(pi,
protectedExp(sub(mul(sin(0.17533810294652064), acos(sin(mul(tan(pi),
protectedPow(sub(mul(pi, -0.38829659409995165), protectedPow(pi, x)), x))))),
protectedDiv(e, protectedDiv(add(protectedSqrt(x), e),
```

```
protectedDiv(add(protectedSqrt(protectedLog(pi)), 0.6895664562337753),
mul(cos(mul(cos(mul(cos(mul(cos(add(protectedSqrt(x),
mul(protectedPow(mul(cos(add(protectedSqrt(mul(0.1974662247007397, pi)),
mul(sub(protectedDiv(tan(pi), protectedPow(sin(e), e)),
protectedDiv(protectedPow(0.6854238043211098, x), x)), -0.007267505597527313))),
protectedExp(add(add(mul(cos(e),
add(protectedSqrt(protectedPow(protectedLog(cos(mul(mul(pi, pi),
protectedExp(mul(pi, -0.007267505597527313))))),
protectedPow(0.6895664562337753, protectedSqrt(x)))), -0.3649005994848231)),
mul(protectedDiv(protectedPow(mul(cos(add(protectedSqrt(x), mul(pi,
-0.007267505597527313))), protectedExp(e)), pi),
add(protectedSqrt(add(protectedSqrt(x), tan(x))), tan(x))),
atan(protectedLog(protectedSqrt(x))))), e))), pi), -0.007267505597527313))),
protectedExp(sub(protectedPow(protectedDiv(protectedPow(pi, pi), x),
protectedDiv(add(protectedSqrt(x), 0.6895664562337753),
protectedPow(protectedLog(cos(mul(mul(pi, pi),
protectedExp(sub(protectedPow(protectedDiv(protectedPow(pi,
protectedSqrt(protectedDiv(add(-0.8926068981514255, e), e))), sub(x, pi)),
protectedDiv(add(protectedSqrt(x), 0.6895664562337753), mul(x,
atan(-0.45921342723094893)))),
protectedSqrt(protectedDiv(protectedPow(0.6854238043211098, x), x)))))),
protectedPow(0.6895664562337753, protectedSqrt(x))))), pi)))),
add(add(mul(cos(add(-0.5417706644866469, pi)), protectedExp(e)),
mul(add(protectedSqrt(mul(pi, mul(e, e))), mul(pi, -0.007267505597527313)),
atan(protectedLog(protectedLog(x))))), pi))), protectedExp(e))),
-0.007267505597527313))), mul(protectedSqrt(pi), protectedLog(e))))))))))),
protectedDiv(x, add(protectedSqrt(tan(e)), 0.6895664562337753)))),
protectedDiv(e, mul(sub(protectedDiv(tan(pi), protectedPow(sin(e), e)),
protectedDiv(protectedPow(0.6854238043211098, x), x)),
protectedExp(sub(cos(acos(cos(e))), protectedDiv(e, cos(mul(mul(atan(protectedLo
g(mul(protectedDiv(protectedPow(mul(cos(add(protectedSqrt(add(protectedSqrt(x),
protectedDiv(e, mul(pi, -0.007267505597527313)))), mul(protectedSqrt(pi), pi))),
protectedExp(sin(protectedSqrt(e)))), pi), add(protectedSqrt(x), tan(e))),
atan(atan(protectedLog(protectedExp(e)))))))), pi),
protectedDiv(protectedSqrt(atan(x)), pi))))))))), -0.007267505597527313),
-0.3649005994848231), pi))), pi))), mul(x, -0.3649005994848231))))),
mul(protectedDiv(protectedPow(mul(sin(e), protectedExp(e)), pi),
protectedPow(pi, pi)), protectedPow(mul(pi, sin(e)), pi))), pi),
protectedPow(pi, pi)), -0.007267505597527313)), pi))), e))), protectedExp(pi)),
pi)
Altura: 122
Error cuadrático (SSE): 2155.519540182032
```

[20]:
```
# Plot

mse = np.mean([(func_best(x) - y)**2 for x, y in enumerate(time_series)])
```

```
x1_vals = np.arange(len(time_series))
y1_vals = time_series

x2_vals = np.arange(len(time_series) + 500)
y2_vals = [func_best(x) for x in x2_vals]

fig = go.Figure()
fig.add_trace(go.Scatter(x=x1_vals, y=y1_vals, mode='lines', name='True',␣
 ↪line=dict(color='blue')))
fig.add_trace(go.Scatter(x=x2_vals, y=y2_vals, mode='lines', name='Predicted',␣
 ↪line=dict(color='red')))

fig.update_layout(title=f'Symbolic Regression Result (MSE: {mse:.4f})',
                  xaxis_title='Index',
                  yaxis_title='Value',
                  template='plotly_white')
fig.show()
```

```
/var/folders/4w/72wtbysj7kb189x1nn694p9h0000gn/T/ipykernel_30759/538667834.py:22
: RuntimeWarning:

divide by zero encountered in scalar power
```



[21]:
```python
# Predecir

to_predict = [2518, 2519, 2520]
for x in to_predict:
    print(f"x={x}, predicted={func_best(x)}")
```

```
x=2518, predicted=19.034887717107566
x=2519, predicted=19.03256869647048
x=2520, predicted=19.030379377737468
```

Mejor solución de todas mis corridas

[ ]:
```python
with open('best_big.pkl', 'rb') as f:
    best_loaded = pickle.load(f)

print("Mejor individuo:", best_loaded)
print("Altura:", best_loaded.height)
print("Error cuadrático (SSE):", best_loaded.fitness.values[0])

func_best = toolbox_sym.compile(expr=best_loaded)

to_predict = [2518, 2519, 2520]
for x in to_predict:
    print(f"x={x}, predicted={func_best(x)}")

# Plot

mse = np.mean([(func_best(x) - y)**2 for x, y in enumerate(time_series)])

x1_vals = np.arange(len(time_series))
y1_vals = time_series

x2_vals = np.arange(len(time_series) + 500)
y2_vals = [func_best(x) for x in x2_vals]

fig = go.Figure()
fig.add_trace(go.Scatter(x=x1_vals, y=y1_vals, mode='lines', name='True',␣
 ↪line=dict(color='blue')))
fig.add_trace(go.Scatter(x=x2_vals, y=y2_vals, mode='lines', name='Predicted',␣
 ↪line=dict(color='red')))

fig.update_layout(title=f'Symbolic Regression Result (MSE: {mse:.4f})',
                  xaxis_title='Index',
                  yaxis_title='Value',
                  template='plotly_white')
fig.show()
```

```
protectedSqrt(sub(mul(protectedExp(protectedDiv(protectedSqrt(e),
protectedPow(mul(protectedDiv(pi, e),
protectedDiv(tan(protectedExp(protectedDiv(61.14290513364358,
-34.3641034404776))), protectedDiv(-39.41685423336148, e))),
protectedSqrt(protectedExp(-20.398941409099436))))), protectedPow(pi,
sub(protectedPow(mul(-61.580470959729475, x), x), sin(pi)))), x)))))),
protectedExp(sub(protectedLog(x), protectedPow(x, 43.48689017565073)))))), e))),
add(x, sin(-46.287566509271485)))), x))), protectedPow(add(-39.41685423336148,
protectedLog(pi)), protectedSqrt(add(x, sin(protectedLog(mul(mul(pi,
-43.863356896805136), sub(mul(protectedExp(protectedDiv(protectedSqrt(e),
protectedPow(mul(protectedDiv(pi, e),
protectedDiv(tan(protectedExp(protectedDiv(61.14290513364358,
-34.3641034404776))), protectedDiv(-39.41685423336148, e))),
protectedSqrt(protectedExp(-20.398941409099436))))),
protectedPow(-34.3641034404776, protectedSqrt(mul(protectedDiv(add(e, e), e),
protectedPow(sub(mul(protectedExp(pi), e), x), protectedExp(x))))), x)))))))),
x)))))))), x))))))))))
Altura: 61
Error cuadrático (SSE): 1369.7717894520229
x=2518, predicted=18.509423010247883
x=2519, predicted=18.510192146181392
x=2520, predicted=18.510961348712286
```