28/05/2019

Trabajo Práctico 1.1: Algoritmo Genético Canónico con Elitismo

ALGORITMOS GENÉTICOS 2019

ANTONELLI (44852) – RECALDE (44704) – ROHN (41355)

UTN FRRO | Profesores: Díaz – Lombardo

Índice

ÍNDICE 1

INTRODUCCIÓN 2

CÓDIGO 3

EXPLICACIÓN DE FUNCIONAMIENTO..20

CONCLUSIÓN 22

Introducción

Implementamos algoritmos genéticos como una simulación computacional en la cual una población de las representaciones abstractas de las soluciones candidata (individuos) de un problema de optimización, evoluciona hacia mejores soluciones.

En este caso utilizamos *Python* y *C++* para la creación de un programa, mediante el uso de un ***algoritmo genético canónico***, que nos permite buscar el máximo de la función objetivo:

$$f(x) = \left(\frac{x}{coef}\right)^2 \text{ en el dominio } [0, 2^{30} - 1]$$

Donde: $coef = 2^{30} - 1$

Para lograr la implementación del algoritmo genético se tuvieron en cuenta los siguientes datos:

- Probabilidad de Corssover = 0,75
- Probabilidad de Mutación = 0,05
- Población inicial: 10 individuos
- Ciclos del programa: 20
- Método de Selección: Ruleta con el agregado de Elitismo
- Método de Crossover: 1 punto
- Método de Mutacion: invertida

Código

Python

- __init__

```python
from GeneticAlgorithmsElitism.Population import Population
from GeneticAlgorithmsElitism.Graphs import Graphic
# from GeneticAlgorithmsElitism.Chromosome import Chromosome
# import random


if __name__ == '__main__':
    # ImportantValues
    iterationLimit = 50  # 20,100,200  # Population Iterations
    initPopulationNum = 10  # Initial Population Size
    chromsomeSize = 30  # Chromosome Size
    crossoverProb = 0.75  # Probability of CrossOver
    mutationProb = 0.05  # Probability of Mutation

    # Initialize
    class Main(object):
        # First Population
        pob = Population(initPopulationNum, chromsomeSize, crossoverProb, mutationProb)
        graphicsData = {'averageOPs': [], 'minOPs': [], 'maxOPs': []}  # Dictionary for Graphics

        # Iterations
        for iterationCount in range(iterationLimit):
            print()
            averageOP, minOP, maxOP, elitChrom, secondElitChrom = pob.showPopulation(iterationCount)  # Show Actual Population and Return Data

            # Update Dictionary with important values
            graphicsData['averageOPs'].append(averageOP)
            graphicsData['minOPs'].append(minOP)
            graphicsData['maxOPs'].append(maxOP)

            # In the last iteration, the chromosomes population mustn't reproduce
            if iterationCount < iterationLimit - 1:
                pob.reproduce(elitChrom, secondElitChrom)  # Reproduction of Actual Generation

        # Graph Population's Evolution
        graph = Graphic(graphicsData, iterationLimit)
        graph.showPlots()

        # Last Reproduction Message
        print("Last Generation Reached Correctly")
        print("------------")
        print()
        print()

        # Final Tables
        print("TABLAS FINALES")
        print()
        print("Población ------------ Mínimo ------------ Máximo ------------ Promedio")
        for value in range(iterationLimit):
            print("Generación ", value, ":", graphicsData['minOPs'][value], "---", graphicsData['maxOPs'][value], "---",
                  graphicsData['averageOPs'][value])
```

- Chromosome

```python
import random


class Chromosome(object):
    # Class Attribute (Constant)
    # coef = random.randint(1, 230 - 1)
    coef = (2**30)-1  # (2^30)-1

    # Constructor / Instance Attributes
    def __init__(self, large, newBody):

        # Chromosome's Genes
        if newBody is None:
            self.body = []
            for _ in range(large):
                self.body.append(str((random.randint(0, 1))))
        else:
            self.body = newBody
        self.large = large
        # Initialize Objective Function Punctuation and Fitness
        self.setObjectivePunctuation()
        self.fitness = 0

    # Show All Genes of the Chromosome
    def getBody(self):
        return self.body

    def toBinInteger(self):
        str_bin_num = ''.join(str(i) for i in self.body)

        return int(str_bin_num)

    # Real Number to pass on Objective Function
    def getRealValue(self):
        num = ''.join(str(i) for i in self.body)
        return int(num, 2)  # Convert body to String and then to Binary Int

    def calcObjPunc(self):
        return (self.getRealValue() / self.getCoef()) ** 2

    def calcFitness(self, totalObj):
        # if totalObj == 0: totalObj = 1  # Prevent Division by Zero Error
        self.fitness = (self.getObjectivePunctuation() / totalObj)  # Update Fitness
        return self.fitness

    # Class Methods
    @classmethod
    def getCoef(cls):
        return cls.coef

    @classmethod
    def setCoef(cls, coeficient):
        cls.coef = coeficient

    # Getters and Setters
    def getLarge(self):
        return self.large

    def setLarge(self, large):
        self.large = large

    def getObjectivePunctuation(self):
        return self.objectivePunctuation

    def setObjectivePunctuation(self):
        self.objectivePunctuation = self.calcObjPunc()

    def getFitness(self):
        return self.fitness

    def setFitness(self, fitness):
        self.fitness = fitness

    def copy(self, another_crom, start, end):
        for i in range(start, end):
            self.body[i] = (another_crom.body[i])

    def mutate(self):
        mutPos = random.randint(0, self.getLarge()-1)

        if self.body[mutPos] == '0':  # If is a '0' then change to '1', and vice-versa
            self.body[mutPos] = '1'
        elif self.body[mutPos] == '1':
            self.body[mutPos] = '0'

        print("Mutated Chrom in position:", mutPos, ":", self.toBinInteger())
```

- Population

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from GeneticAlgorithmsElitism.Chromosome import Chromosome
import random


class Population(object):
    # Class Attributes
    population = []  # Initial Population (Array of Chromosomes)
    totalObjPunc = 0  # The Sum of All Objective Functions Punctuation
    totalFitness = 0  # The Sum of All Objective Values

    # Constructor / Instance Attributes
    def __init__(self, numChroms, chromSize, crossProb, mutProb):
        self.numChroms = numChroms
        self.chromSize = chromSize
        self.crossProb = crossProb
        self.mutProb = mutProb
        # print("Objective Function Coeficient:", Chromosome.getCoef())
        print("Start Algorithm")
        for _ in range(numChroms):
            oneChrom = Chromosome(chromSize, None)  # Initialization of Chromosomes
            self.addChrom(oneChrom)  # Add to Population

    # Show Actual Population and Stats
    def showPopulation(self, numIter):
        self.setTotalFitness(0)
        self.setTotalObjPunc(self.calcTotalObjPunc())
        large = self.getChromSize()
        averageObjPunc = self.getTotalObjPunc() / len(self.population)
        fitness = 0
        maxVal = 0
        secondMaxVal = 0
        minVal = 0
        maxChrom = 0
        secondMaxChrom = 0
        minChrom = 0
        print("Population ", (numIter + 1), ":")
        for i in range(len((self.population))):
            fitness = self.population[i].calcFitness(self.getTotalObjPunc())
            self.updateTotalFitness(fitness)
            if i == 0:
                maxVal = fitness
                minVal = fitness
            elif fitness > maxVal:
                maxVal = fitness
                maxChrom = i
            elif fitness < minVal:
                minVal = fitness
                minChrom = i
            elif (fitness > secondMaxVal) and (fitness < maxVal):
                secondMaxVal = fitness
                secondMaxChrom = i
            for j in range(large):
                print(self.population[i].getBody()[j], end='')
            print()
        fitness = self.getTotalFitnessAverage()
        print()
        print("Chromosome --- Value --- Objective Punctuation --- Fitness")
        print("Max Values: Chrom Nº", maxChrom, "with:", self.population[maxChrom].getRealValue(), "Val,",
              self.population[maxChrom].getObjectivePunctuation(), "OP,", round(maxVal, 4), "Fit")
        print("Second Max Values: Chrom Nº", secondMaxChrom, "with:", self.population[secondMaxChrom].getRealValue(), "Val,",
              self.population[secondMaxChrom].getObjectivePunctuation(), "OP,", round(secondMaxVal, 4), "Fit")
        print("Min Values: Chrom Nº", minChrom, "with:", self.population[minChrom].getRealValue(), "Val,",
              self.population[minChrom].getObjectivePunctuation(), "OP,", round(minVal, 4), "Fit")
        print("Average OP:", averageObjPunc, "--- Average Fitness:", fitness)  # round(fitness,6)
        print()
        # Return Important Data to use on Graphics
        elitChrom = self.population[maxChrom]
        secondEliChrom = self.population[secondMaxChrom]
        return (averageObjPunc, self.population[minChrom].getObjectivePunctuation(),
                self.population[maxChrom].getObjectivePunctuation(), elitChrom, secondEliChrom)

    # Calculate Total of Objective Functions Punctuation in the actual Generation
    def calcTotalObjPunc(self):
        acumObjPunc = 0
        for chromosome in self.population:
            acumObjPunc += chromosome.getObjectivePunctuation()  # Add Every Objective Function Punctuation
        #  self.setTotalObjPunc(acumulator)
        return acumObjPunc

    # Update Total Fitness
    @classmethod
    def updateTotalFitness(cls, fitness):
        cls.totalFitness += fitness

    # Add to Population
    def addChrom(self, Chrom):
        self.population.append(Chrom)

    # Reproduction
    def reproduce(self, elitChrom, secondElitChrom):
        parents = []  # List of Potential Parents
        newGeneration = []  # List of Children
```

```python
 print("Roulette Results: ", end='')

 # TO-DO:Send the second best Chromosome

 # Elitism
        parents.append(elitChrom)
        parents.append(secondElitChrom)
        self.addChildren(parents[0], parents[1], newGeneration)

        for _ in range(2, len(self.population), 2):
            for i in range(2):
                lastParent = self.roulette()  # Parents Selected by Roulette
                parents.append(self.population[lastParent])
        print()
        for i in range(2, len(parents), 2):
            father1 = parents[i]
            father2 = parents[i + 1]
            if self.crossPosibility():  # CrossOver Probability Evaluation
                son1, son2 = self.cross(father1, father2)  # CrossOver
                print("Successful CrossOver in reproduction:", (i + 2) / 2 - 1)  # Only Print
            else:
                son1, son2 = self.copy(father1, father2)  # Direct Assignation (Without CrossOver)
                print("CrossOver didn't happen in reproduction:", (i + 2) / 2 - 1)  # Only Print

            # Individual Mutation Probability Evaluation even when Crossover is not successful

            if self.mutationPosibility():
                son1.mutate()
            if self.mutationPosibility():
                son2.mutate()
            son1.setObjectivePunctuation()
            son2.setObjectivePunctuation()
            self.addChildren(son1, son2, newGeneration)
        print()
        print("Successful Elitism Applied")
        print()
        self.replacePopulation(newGeneration)
        self.setTotalFitness(0)

    # Genetic Operator (Roulette Method)
    def roulette(self):
        # Generator of a Bidimensional List (Fitness Range of Chromosomes)
        newRoulette = [[0] * 2 for _ in range(len(self.population))]
        acum = 0  # Acumulator of Relative Fitness from 0 to 1 (Fills Roulette)
        for i in range(len(self.population)):
            newRoulette[i][0] = acum  # Range Min: Last Acum Value
            acum += round(self.population[i].getFitness(), 6)  # Acum's Value From Zero
            newRoulette[i][1] = acum  # Range Max: New Acum Value
        ranNum = round(random.uniform(0, 1), 6)  # Random Number from 0.000000 to 0.999999
        # print("Random: ", ranNum)  # Only Print
        for i in range(len(newRoulette)):
            if newRoulette[i][0] < ranNum < newRoulette[i][1]:
                # Return Selected Chromosome if the Random Number Exists in its Range
                print(i, end=', ')
                return i

    def crossPosibility(self):  # CrossOver posibility evaluation
        if self.getCrossProb()*100 >= random.randint(1, 100):
            return True
        else:
            return False

    def cross(self, parent1, parent2):
        crom_size = parent1.getLarge()
        son1 = Chromosome(crom_size, None)
        son2 = Chromosome(crom_size, None)
        cut = random.randint(1, crom_size - 2)  # Random Cut Point (Except by zero or all genes)

        son1.copy(parent1, 0, cut)
        son1.copy(parent2, cut, crom_size)

        son2.copy(parent2, 0, cut)
        son2.copy(parent1, cut, crom_size)

        print()
        print("Son 1:", son1.toBinInteger())  # Only Print
        print("Son 2:", son2.toBinInteger())  # Only Print
        print("Cut Point on:", cut)  # Only Print
        return son1, son2

    def copy(self, chrom1, chrom2):
        # newGeneration.append(chrom1)
        # newGeneration.append(chrom2)
        son1 = chrom1
        son2 = chrom2
        print()
        print("Son 1 (Identical):", self.listToInt(chrom1.getBody()))  # Only Print
        print("Son 2 (Identical):", self.listToInt(chrom2.getBody()))  # Only Print
        return son1, son2

    def mutationPosibility(self):  # Mutation posibility evaluation
        if self.getMutProb() * 100 >= random.randint(1, 100):
            return True
        else:
            return False

    def addChildren(self, son1, son2, newGeneration):
        newGeneration.append(son1)
        newGeneration.append(son2)

    def replacePopulation(self, newGeneration):  # Replace All Population in every Iteration
        self.population = []
        for i in range(len(newGeneration)):
            self.population.append(newGeneration[i])

    def listToInt(self, arr):
        num = ''.join(str(i) for i in arr)
        return int(num)

    # Class Getters and Setters
    @classmethod
    def getTotalObjPunc(cls):
        return cls.totalObjPunc

    @classmethod
    def setTotalObjPunc(cls, total):
        cls.totalObjPunc = total

    @classmethod
    def getTotalFitness(cls):
        return cls.totalFitness

    @classmethod
    def setTotalFitness(cls, total):
            cls.totalFitness = total

    @classmethod
    def getTotalFitnessAverage(cls):
        return cls.totalFitness / len(cls.population)

    # Getters and Setters
    def getNumChroms(self):
        return self.numChroms

    def setNumChroms(self, numChroms):
        self.numChroms = numChroms

    def getChromSize(self):
        return self.chromSize

    def setChromSize(self, chromSize):
        self.chromSize = chromSize

    def getCrossProb(self):
        return self.crossProb

    def setCrossProb(self, crossProb):
        self.crossProb = crossProb

    def getMutProb(self):
        return self.mutProb

    def setMutProb(self, mutProb):
        self.mutProb = mutProb
```

- Graphs

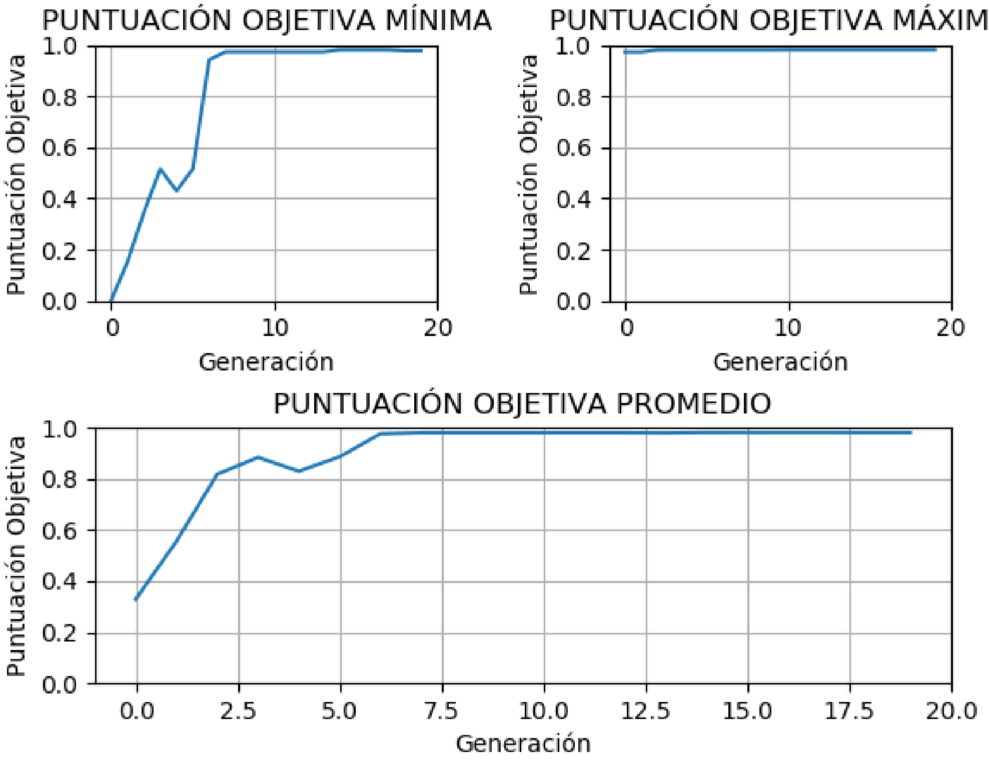
```
from matplotlib import pyplot
# import math
# import numpy as np

# pyplot.plot(x1,y1,'b-',x2,y2,'r-',x3,y3,'g-')
# pyplot.legend('value 1', 'value 2', 'value 3')
# pyplot.savefig('generations.png')  # Save Image --> Better Save Manually


class Graphic(object):
    def __init__(self, graphicsData, iterationLimit):
        self.averageOPs = graphicsData['averageOPs']
        self.minOPs = graphicsData['minOPs']
        self.maxOPs = graphicsData['maxOPs']
        self.generations = []
        for i in range(iterationLimit):
            self.generations.append(i)
        # pyplot.ion()

    # Show one Plot with all Graphs
    def showPlots(self):
        # with pyplot.style.context(('dark_background')):
        self.drawAll(self.minOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA MÍNIMA", 221)
        self.drawAll(self.maxOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA MÁXIMA", 222)
        self.drawAll(self.averageOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA PROMEDIO", 212)
        pyplot.show()  # Draw all the "Subplots"

    # Show one Graph in each Plot
    def showPlotsApart(self):
        self.draw(self.minOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA MÍNIMA")
        self.draw(self.maxOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA MÁXIMA")
        self.draw(self.averageOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA PROMEDIO")

    # Generate one Subplot for every Graph (and Show all Graphs)
    def drawAll(self, axisY, axisX, labY, labX, title, region):
        pyplot.subplot(region)
        pyplot.axis([-1, len(self.generations), 0, 1])
        pyplot.grid(True)
        pyplot.plot(axisX, axisY)
        pyplot.ylabel(labY)
        pyplot.xlabel(labX)
        pyplot.title(title)

    # Generate one plot for every Graph
    def draw(self, axisY, axisX, labY, labX, title):
        pyplot.axis([-1, len(self.generations), 0, 1])
        pyplot.grid(True)
        pyplot.plot(axisX, axisY)
        pyplot.ylabel(labY)
        pyplot.xlabel(labX)
        pyplot.title(title)
        pyplot.show()
```

Repositorio GitHub:
https://github.com/alereca/Genetic-Algorithms-Cpp
Derivadado de la anterior version sin Elitismo:
https://github.com/NicoCaptain/Genetic-Algorithms

Gráficas y Tablas

Python: Matplotlib

Algoritmo en 20 Generaciones

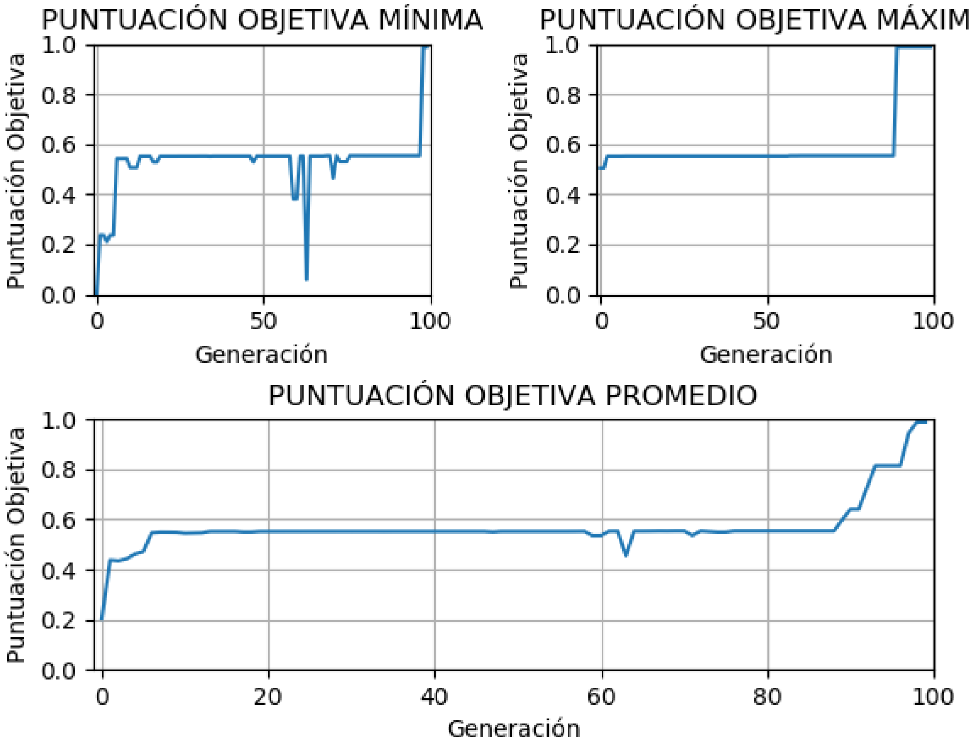


------------ ------------ ------------

Población	Mínimo	Máximo	Promedio
Generación 0	0.0023104775658062977	0.9727623800214141	0.32895423244688715
Generación 1	0.15408542872678546	0.9727624296231964	0.5557560916559194
Generación 2	0.3455672330784231	0.9809656152575349	0.8178600379784118
Generación 3	0.5152406818298353	0.9809656152575349	0.8829139380018397
Generación 4	0.42941962475843176	0.9809656152575349	0.8293972789891988
Generación 5	0.5152378112697693	0.9809696001004942	0.8853749516664491
Generación 6	0.9422600203759036	0.9809696001004942	0.975462406037855
Generación 7	0.9732432901353516	0.9809696001004942	0.9801536675106828
Generación 8	0.9732432901353516	0.9809696001004942	0.9801540659949787
Generación 9	0.9732432901353516	0.9809696001004942	0.9801540905312564
Generación 10	0.9732432901353516	0.9814494933917527	0.9802430693222701
Generación 11	0.9732470534558618	0.9814494933917527	0.9802914349834471
Generación 12	0.9732470534558618	0.9814494933917527	0.9802914349834468
Generación 13	0.9732432901353516	0.9814494933917527	0.979519559631267
Generación 14	0.9809658218788228	0.9814494933917527	0.9811124346213702
Generación 15	0.9809658218788228	0.9814494933917527	0.981160423212381

Generación 16	0.9809658144994906	0.9814494933917527	0.9812087888876256
Generación 17	0.9809658218788228	0.9814494933917527	0.9812091674477262
Generación 18	0.9771045120809053	0.9814494933917527	0.9808226571697174
Generación 19	0.9771045120809053	0.9814532725447468	0.9808710251522592

Algoritmo en 100 Generaciones



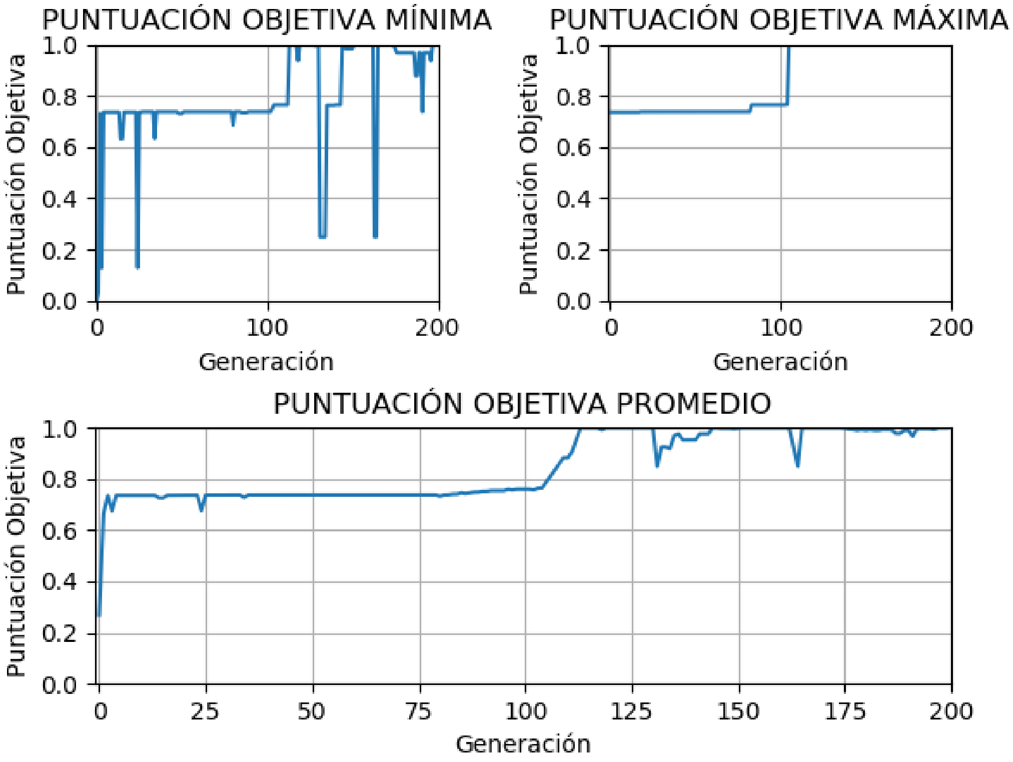
------------ ------------ ------------

Población	Mínimo	Máximo	Promedio
Generación 0	4.323769803803884e	0.20298874533747768	0.5046122896952413
Generación 1	0.237107484999612	0.5046122896952413	0.43725701928025557
Generación 2	0.237107484999612	0.5514327787323007	0.43419838290542084
Generación 3	0.21013748239738997	0.5514331535721718	0.44277767451082684
Generación 4	0.23690401269079542	0.5514331535721718	0.4619312875724873
Generación 5	0.23708426661017987	0.5514388190641157	0.47131378634387033
Generación 6	0.5430407654341587	0.5514388190641157	0.5463990925114002
Generación 7	0.5427653125735014	0.5517107969035288	0.5488728508298705
Generación 8	0.5430407654341587	0.5518921527160405	0.5487016773473826
Generación 9	0.5430407654341587	0.5518921527160405	0.548224519839485
Generación 10	0.5064378125175827	0.5518921527160405	0.5446276903757208
Generación 11	0.5064378125175827	0.5518921527160405	0.5454767710290808
Generación 12	0.5064378125175827	0.5518921527160405	0.5463165042586938
Generación 13	0.5515294708933395	0.5518921527160405	0.5516926732432066
Generación 14	0.5515294708933395	0.5518921527160405	0.5516745406421876
Generación 15	0.5515294266279329	0.5518921527160405	0.5517108043979171

Generación 16	0.5515294266279329	0.5518921527160405	0.5517289325723953
Generación 17	0.5289208419913568	0.5518921527160405	0.54943180592574
Generación 18	0.5288764539185794	0.5518921527160405	0.5494636353007325
Generación 19	0.5515294708933395	0.5518949866370797	0.5517836256008696
Generación 20	0.5515294708933395	0.5518949866370797	0.5517657763919546
Generación 21	0.5515323038830561	0.5518949866370797	0.5517839088998413
Generación 22	0.5515323038830561	0.5518949866370797	0.5517844785711492
Generación 23	0.5515323038830561	0.5519176582673271	0.5518094127543745
Generación 24	0.5515323038830561	0.5519176582673271	0.5517731415918722
Generación 25	0.5517107969035288	0.5519176582673271	0.5518323631666792
Generación 26	0.5517107969035288	0.5519176582673271	0.5518504987479304
Generación 27	0.5517107969035288	0.5519176582673271	0.5518660837740528
Generación 28	0.5517107969035288	0.5519176582673271	0.551884219355304
Generación 29	0.5517107969035288	0.5519176582673271	0.5518918710206899
Generación 30	0.5517107969035288	0.5519176582673271	0.5518893204655613
Generación 31	0.5518921527160405	0.5519176582673271	0.5519074560468125
Generación 32	0.5518921527160405	0.5519176582673271	0.5519074560468125
Generación 33	0.5518921527160405	0.5519176582673271	0.5519100066019412
Generación 34	0.5511923970674084	0.5519176582673271	0.5518371970840246
Generación 35	0.5518921527160405	0.5519176582673271	0.5519046220937591
Generación 36	0.5518921527160405	0.5519176582673271	0.5519017881407058
Generación 37	0.5518921527160405	0.5519176582673271	0.5519017881407058
Generación 38	0.5518921527160405	0.5519176582673271	0.5519063224492929
Generación 39	0.5518921527160405	0.5519176582673271	0.5519066235537459
Generación 40	0.5518921527160405	0.5519176582673271	0.5519029571279985
Generación 41	0.5518921527160405	0.5519176582673271	0.5519069600887174
Generación 42	0.5518921527160405	0.5519176582673271	0.5519069423763685
Generación 43	0.5518921527160405	0.5519176582673271	0.5519049586054476
Generación 44	0.5518923298358923	0.5519176582673271	0.5519097940414069
Generación 45	0.5518923298358923	0.5519176582673271	0.5519052619402698
Generación 46	0.5518923519758757	0.5519176582673271	0.5519083261491078
Generación 47	0.5289236163077511	0.5519176582673271	0.5496089219477387
Generación 48	0.5518949866370797	0.5519176582673271	0.5519128604697457
Generación 49	0.5518949866370797	0.5519176582673271	0.5519125969982139
Generación 50	0.5518949866370797	0.5519176582673271	0.551910325407091
Generación 51	0.5518949866370797	0.5519176582673271	0.5519143327900767
Generación 52	0.5518949866370797	0.5519176582673271	0.5519125925701157
Generación 53	0.5518949866370797	0.5519176582673271	0.5519128516135494
Generación 54	0.5518949423570075	0.5519176582673271	0.5519131150851723
Generación 55	0.5518949866370797	0.5519176804078185	0.5519153888902534
Generación 56	0.5518949866370797	0.5519176804078185	0.5519153933183516
Generación 57	0.5518949866370797	0.553369633348237	0.5520583258774214
Generación 58	0.5519176582673271	0.5533696388906358	0.5523532524543198
Generación 59	0.38181475596089165	0.5533696388906358	0.5351974807634206
Generación 60	0.38181475596089165	0.5537328974296948	0.5353790074472118
Generación 61	0.5519176582673271	0.5537328974296948	0.5527163002867933
Generación 62	0.5519176582673271	0.5537328974296948	0.5529341500532979
Generación 63	0.05948143980032688	0.5537328974296948	0.4542631115591546

Generación 64	0.5519176582673271	0.5537328974296948	0.552929608835569
Generación 65	0.5519176582673271	0.5537328974296948	0.5529296116015047
Generación 66	0.5519176582673271	0.5537329251507839	0.5531134014223198
Generación 67	0.5519176582673271	0.5537329251507839	0.5535513790576758
Generación 68	0.5519176582673271	0.5537329251507839	0.553369860685657
Generación 69	0.5535512671195486	0.5537329251507839	0.553714739942898
Generación 70	0.553551239403007	0.5537329251507839	0.5536965744174401
Generación 71	0.46462261091757007	0.5537329251507839	0.535892685412582
Generación 72	0.553551239403007	0.5537336348109034	0.5536967271606887
Generación 73	0.5307229250016897	0.5537336348109034	0.5513957304723099
Generación 74	0.5307229250016897	0.5537336348109034	0.5491311358013137
Generación 75	0.5307229250016897	0.5537336348109034	0.5491265947841553
Generación 76	0.553732919606566	0.5537336348109034	0.5537330659739642
Generación 77	0.5537329251507839	0.5537336348109034	0.5537330670828078
Generación 78	0.5537329251507839	0.5537336348109034	0.5537331380488196
Generación 79	0.5537329251507839	0.5537336348109034	0.5537330759535567
Generación 80	0.5537329251507839	0.5537336348109034	0.5537331557903177
Generación 81	0.5536421811100902	0.5537336348109034	0.5537241523522604
Generación 82	0.5537329251507839	0.5537336348109034	0.5537331646610667
Generación 83	0.5537329251507839	0.5537336348109034	0.5537333065930907
Generación 84	0.5537329251507839	0.5537336348109034	0.5537331114365528
Generación 85	0.5537329251507839	0.5537336348109034	0.5537331291780508
Generación 86	0.5537329251507839	0.5537336348109034	0.5537331824025647
Generación 87	0.5537329251507839	0.5537337235184504	0.5537331203073076
Generación 88	0.5537315945392856	0.5537337235184504	0.5537329783755338
Generación 89	0.5537315945392856	0.988299209324953	0.5971897575956042
Generación 90	0.5537315945392856	0.988299209324953	0.6406463150470092
Generación 91	0.553733013858274	0.9883001574037089	0.6406466227528014
Generación 92	0.553733013858274	0.9883001574037089	0.7275598827077684
Generación 93	0.5537337235184504	0.9883001574037089	0.8144733934931677
Generación 94	0.553733013858274	0.9883001574037089	0.8144733702108657
Generación 95	0.553733013858274	0.9883001574037089	0.8144733709515526
Generación 96	0.553733013858274	0.9883001574037089	0.8144734180757123
Generación 97	0.5537337235184504	0.9883001574037089	0.9448432295915563
Generación 98	0.988299209324953	0.9883001574037089	0.9882998729800819
Generación 99	0.988299209324953	0.9883001574037089	0.9882996833643309

Algoritmo en 200 Generaciones



------------ ------------ ------------

Población	Mínimo	Máximo	Promedio
Generación 0	0.0006013283477214634	0.7357318500752823	0.2673519297637418
Generación 1	0.031020847397699728	0.7357318500752823	0.6642748361592251
Generación 2	0.7308022818337874	0.7357318500752823	0.7347452820185507
Generación 3	0.12797676098673125	0.7357318500752823	0.6744637390305237
Generación 4	0.7357253059909556	0.7357318500752823	0.7357292324415516
Generación 5	0.7357253059909556	0.7357318500752823	0.7357285780331191
Generación 6	0.7357253059909556	0.7357318500752823	0.7357285780331191
Generación 7	0.7357253059909556	0.7357318500752823	0.7357279236246863
Generación 8	0.7357253059909556	0.7357318500752823	0.7357279236246864
Generación 9	0.7357253059909556	0.7357318500752823	0.7357279236246864
Generación 10	0.7353065451300187	0.7357318500752823	0.7356860475385927
Generación 11	0.7353065451300187	0.7357318500752823	0.7356867019470255
Generación 12	0.7357253059909556	0.7357318500752823	0.7357305412584171
Generación 13	0.7357253059909556	0.7357318500752823	0.7357305412584171
Generación 14	0.632419584056459	0.7357318500752823	0.7253993555569764
Generación 15	0.632419584056459	0.7357318500752823	0.7253987011485438
Generación 16	0.7357253059909556	0.7357318500752823	0.7357292733419935
Generación 17	0.7357253059909556	0.7357318500752823	0.7357293142424353
Generación 18	0.7357253059909556	0.7374080930625082	0.7358975520491488
Generación 19	0.7357253059909556	0.7374085025344063	0.7360651763946194
Generación 20	0.7357253059909556	0.7374085025344063	0.7360664852114847
Generación 21	0.7357253059909556	0.7374085025344063	0.7364017338089299

Generación 22	0.7357253059909556	0.7374085025344063	0.7365687847554091
Generación 23	0.7357253059909556	0.7374085025344063	0.7364018156101773
Generación 24	0.1286834028175566	0.7374085025344063	0.675865157553517
Generación 25	0.7357318500752823	0.7374085025344063	0.7369053225343186
Generación 26	0.7357318500752823	0.7374085025344063	0.7370730287274208
Generación 27	0.7374079906945514	0.7374085025344063	0.7374083694560413
Generación 28	0.7374079906945514	0.7374085025344063	0.7374084052848318
Generación 29	0.7374080930625082	0.7374085025344063	0.7374084564688171
Generación 30	0.7374084001664211	0.7374085025344063	0.7374084871792085
Generación 31	0.7373822453790404	0.7374085025344063	0.7374058614636719
Generación 32	0.7374084001664211	0.7374085025344063	0.73740847694241
Generación 33	0.7374084001664211	0.7374085025344063	0.7374084820608092
Generación 34	0.6339741364005332	0.7374085025344063	0.727065024973825
Generación 35	0.7374084001664211	0.7374085025344063	0.7374084513504137
Generación 36	0.7374084001664211	0.7374085025344063	0.7374084104032196
Generación 37	0.7373821941959581	0.7374085025344063	0.7374057898061733
Generación 38	0.7373821941959581	0.7374085025344063	0.737403169209127
Generación 39	0.7373756427761019	0.7374085025344063	0.7373999037068936
Generación 40	0.7373821430128777	0.7374085025344063	0.7374057949246638
Generación 41	0.7373821430128777	0.7374085025344063	0.7374058045216624
Generación 42	0.7373821430128777	0.7374085025344063	0.7374031685695096
Generación 43	0.7373821430128777	0.7374085025344063	0.7374057942848639
Generación 44	0.7374083937684222	0.7374085025344063	0.7374084200002182
Generación 45	0.7374083937684222	0.7374085025344063	0.7374084200002182
Generación 46	0.7374083937684222	0.7374101404231354	0.7374085940258897
Generación 47	0.7374083937684222	0.7374101404231354	0.7374087680515611
Generación 48	0.7307166032528238	0.7374101404231354	0.7367397623858727
Generación 49	0.7307166032528238	0.7374608135043308	0.7367448508073893
Generación 50	0.7306122588784075	0.7374608135043308	0.736739667940537
Generación 51	0.7374085025344063	0.7374608135043308	0.7374351493859873
Generación 52	0.7374085025344063	0.7374609158759541	0.7374353131752126
Generación 53	0.7374084001664211	0.7374625538228907	0.7374302765902329
Generación 54	0.7373035790785369	0.7374625538228907	0.737419303109005
Generación 55	0.7374085025344063	0.7375132287048857	0.7374458369893613
Generación 56	0.7374085025344063	0.7375132287048857	0.7374563196693124
Generación 57	0.7374085025344063	0.7375132287048857	0.737462052387548
Generación 58	0.7374085025344063	0.7375132287048857	0.7374622161822416
Generación 59	0.7374624514511539	0.7375132287048857	0.7374726580877685
Generación 60	0.7374624514511539	0.7375132287048857	0.7374726580877685
Generación 61	0.7373035790785369	0.7375132287048857	0.7374618383387064
Generación 62	0.7373035790785369	0.7375148667099164	0.7374300536328302
Generación 63	0.73730531921156	0.7375148667099164	0.737456647992848
Generación 64	0.7374608135043308	0.7375148667099164	0.7374776436866171
Generación 65	0.7374624514511539	0.7375148667099164	0.7374729037867247
Generación 66	0.7374624514511539	0.7375148667099164	0.7374678055859448
Generación 67	0.7374625538228907	0.7375148667099164	0.7374730573489947
Generación 68	0.7374625538228907	0.7375148667099164	0.7374848302904977
Generación 69	0.737305216850737	0.7376724308209627	0.7374913741828377

Generación 70	0.7374624514511539	0.7376724308209627	0.7375320165022823
Generación 71	0.7374625538228907	0.7376724308209627	0.737522874525603
Generación 72	0.7374625538228907	0.7376724308209627	0.7375058691789949
Generación 73	0.7374625538228907	0.7376724308209627	0.7375307778031319
Generación 74	0.7370959038327144	0.7376724308209627	0.7374888714371025
Generación 75	0.7374625538228907	0.7376724308209627	0.7375412827696488
Generación 76	0.7374625538228907	0.7376724324207488	0.7375254956474053
Generación 77	0.7374625538228907	0.7376724324207488	0.737541241661011
Generación 78	0.7374625538228907	0.7376724324207488	0.7375254954874438
Generación 79	0.7374625538228907	0.7376724324207488	0.7375412418209726
Generación 80	0.6849688841317817	0.7376724324207488	0.7322918948441437
Generación 81	0.7374625538228907	0.7376724324207488	0.7375149919577038
Generación 82	0.7374625538228907	0.7376724324207488	0.7375988814851316
Generación 83	0.7374625538228907	0.7647563188830193	0.7403177955759028
Generación 84	0.7374625538228907	0.7647563188830193	0.7403388037529472
Generación 85	0.7341118529270929	0.7647565290097983	0.745441540678097
Generación 86	0.7343210467194865	0.7647565290097983	0.7427327636236988
Generación 87	0.7343210483156344	0.7647565290097983	0.745441152269562
Generación 88	0.7343210483156344	0.7647565290097983	0.7481495008107953
Generación 89	0.7376722260483639	0.7647565290097983	0.7484846596985667
Generación 90	0.7376722260483639	0.7647565290097983	0.7512143970400013
Generación 91	0.7376722260483639	0.7647565290097983	0.7512143764027629
Generación 92	0.7376724324207488	0.7647565290097983	0.7539228693615005
Generación 93	0.7376724324207488	0.7647565290097983	0.7539228483488226
Generación 94	0.7376722260483639	0.7647565290097983	0.7539228487242621
Generación 95	0.7374627601659444	0.7647565290097983	0.7539019231486981
Generación 96	0.7376724324207488	0.7647565290097983	0.7593397096919883
Generación 97	0.7376724324207488	0.7647565290097983	0.7566313000330834
Generación 98	0.7376724324207488	0.7647565290097983	0.7593397096919883
Generación 99	0.7376724324207488	0.7647565290097983	0.7593397096919883
Generación 100	0.7376724324207488	0.7647565290097983	0.7593397096919883
Generación 101	0.7376724324207488	0.7647565290097983	0.7593397083888764
Generación 102	0.7376724324207488	0.7647565290097983	0.7566312993815274
Generación 103	0.7511534455589098	0.7656107748413311	0.7634816445963067
Generación 104	0.7647565290097983	0.7656107748413311	0.7649273781761048
Generación 105	0.7647565290097983	0.9990074267549361	0.7884378925337718
Generación 106	0.7647565290097983	0.9990074267549361	0.8117775577251324
Generación 107	0.7647565290097983	0.9990074267549361	0.8352880720827993
Generación 108	0.7647565290097983	0.9990074267549361	0.8588840102789316
Generación 109	0.7647565290097983	0.9990074267549361	0.8822237589160424
Generación 110	0.7647565290097983	0.9990074267549361	0.8822238423617927
Generación 111	0.7647565290097983	0.9990074267549361	0.9055633391722763
Generación 112	0.7647565224942388	0.9990074267549361	0.9522304400062481
Generación 113	0.9989463260218949	0.9990074267549361	0.9990012861494162
Generación 114	0.9990035171456783	0.9990074267549361	0.9990070253683756
Generación 115	0.9990073299454715	0.9990074267549361	0.999007406648355
Generación 116	0.999006950154541	0.9990074267549361	0.9990073664351973
Generación 117	0.9990069427076607	0.9990074267549361	0.9990073277114162

Generación 118	0.9375150148131947	0.9990074267549361	0.991302514082338
Generación 119	0.9990069427076607	0.9990074267549361	0.9990073768608323
Generación 120	0.999007419308054	0.9990074267549361	0.999007426010248
Generación 121	0.999007419308054	0.9990074267549361	0.999007426010248
Generación 122	0.999007419308054	0.9990074267549361	0.999007426010248
Generación 123	0.999007419308054	0.9990074267549361	0.999007426010248
Generación 124	0.999007419308054	0.9990074267549361	0.9990074245208715
Generación 125	0.999007419308054	0.9990074267549361	0.9990074237761833
Generación 126	0.997056224977893	0.9990074267549361	0.9988123043431673
Generación 127	0.9990073076048266	0.9990074267549361	0.9990074126058606
Generación 128	0.9951069231171562	0.9990074267549361	0.9986173622420825
Generación 129	0.9990073076048266	0.9990074267549361	0.9990074111164841
Generación 130	0.999007419308054	0.9990074267549361	0.9990074222868068
Generación 131	0.24950383240203894	0.9990074267549361	0.8489115839832113
Generación 132	0.24950383240203894	0.9990074267549361	0.9238619426745404
Generación 133	0.24950383612363067	0.9990074267549361	0.9238619318763769
Generación 134	0.24950383612363067	0.9990074267549361	0.9177127040858626
Generación 135	0.7630494678582099	0.9990074267549361	0.9690672665146322
Generación 136	0.7630494613499265	0.9990074267549361	0.9754116272356823
Generación 137	0.7630494613499265	0.9990074267549361	0.9518158306951813
Generación 138	0.7630494613499265	0.9990074267549361	0.9517914172841918
Generación 139	0.7630494678582099	0.9990074267549361	0.9517914168172602
Generación 140	0.7647565224942388	0.9990074490955824	0.951962125259616
Generación 141	0.7647565224942388	0.9990074490955824	0.9753872176405384
Generación 142	0.7647565224942388	0.9990074490955824	0.9753811179467577
Generación 143	0.7647565290097983	0.9990074490955824	0.9755792919735959
Generación 144	0.9989769468991445	0.9990074565424646	0.9990043862162391
Generación 145	0.9990074267549361	0.9990074565424646	0.9990074401593241
Generación 146	0.9834512183004097	0.9990074565424646	0.9974518193138714
Generación 147	0.9834512183004097	0.9990074565424646	0.9974518222926243
Generación 148	0.983451240466433	0.9990074565424646	0.9974518036579582
Generación 149	0.983451240466433	0.9990074565424646	0.9958473856131842
Generación 150	0.983451240466433	0.9990224397255768	0.9974045330927428
Generación 151	0.99851946983193	0.9990224397255768	0.9989144479858473
Generación 152	0.9985194772769932	0.9990224397255768	0.9989618429140155
Generación 153	0.9985194772769932	0.9990227003684243	0.9989634618720542
Generación 154	0.9987786939727917	0.9990227003684243	0.9989908818601325
Generación 155	0.9987786939727917	0.9990227003684243	0.9989937891341574
Generación 156	0.9990072107953679	0.9990227003684243	0.9990166668806999
Generación 157	0.9990072107953679	0.9990227003684243	0.9990154038840879
Generación 158	0.9990083799560675	0.9990227003684243	0.9990169267771245
Generación 159	0.9990083799560675	0.9990227003684243	0.9990212161986186
Generación 160	0.9990083799560675	0.9990227003684243	0.9990212139645369
Generación 161	0.9970571772477018	0.9990227003684243	0.9988230672809768
Generación 162	0.9989921979391557	0.9990227003684243	0.9990195957628458
Generación 163	0.24951145800175664	0.9990227003684243	0.9240714957048208
Generación 164	0.2495113500739589	0.9990227003684243	0.8491203498825944
Generación 165	0.9990224620663909	0.9990227003684243	0.9990226154733242

Generación 166	0.9990224397255768	0.9990227003684243	0.9990226221755695
Generación 167	0.9990224397255768	0.9990227003684243	0.9990226221755696
Generación 168	0.9990224397255768	0.9990227003684243	0.9990226221755696
Generación 169	0.9990226780276076	0.9990227003684243	0.9990226936661794
Generación 170	0.9990226780276076	0.9990227003684243	0.9990226959002613
Generación 171	0.9990226780276076	0.9990227003684243	0.9990226891980161
Generación 172	0.9990226780276076	0.9990227003684243	0.9990226914320978
Generación 173	0.9990188875392999	0.9990227003684243	0.9990223123832671
Generación 174	0.9990207939529526	0.9990227003684243	0.9990225052587138
Generación 175	0.9834663725299851	0.9990227003684243	0.9974670608823356
Generación 176	0.9680321150042733	0.9990227003684243	0.9943680008565327
Generación 177	0.9680321150042733	0.9990227003684243	0.9928245654229408
Generación 178	0.9680321150042733	0.9990227003684243	0.9897255009289747
Generación 179	0.9680321150042733	0.9990227003684243	0.9928245535078387
Generación 180	0.9680320563600849	0.9990227003684243	0.9897255010221068
Generación 181	0.9680321150042733	0.9990227003684243	0.9928245773380432
Generación 182	0.9680321150042733	0.9990227003684243	0.9897255247591792
Generación 183	0.9680321150042733	0.9990227003684243	0.9897255247591792
Generación 184	0.9680321150042733	0.9990227003684243	0.9928245832955943
Generación 185	0.9680321150042733	0.9990227003684243	0.9926294616255911
Generación 186	0.9680321150042733	0.9990227003684243	0.9924342922951839
Generación 187	0.8779895996155614	0.9990227003684243	0.9774262827883884
Generación 188	0.8779900464172774	0.9990227003684243	0.9772342583014331
Generación 189	0.9680316458508156	0.9990227003684243	0.9893337093736323
Generación 190	0.9680246085625902	0.9990227003684243	0.9893329579844059
Generación 191	0.7376855571238073	0.9990227003684243	0.9662983018298332
Generación 192	0.9680246085625902	0.9990227003684243	0.9959190307019506
Generación 193	0.9680246085625902	0.9990227003684243	0.9959182681368535
Generación 194	0.9680241394109514	0.9990227003684243	0.9959182688820934
Generación 195	0.9680246085625902	0.9990227003684243	0.9959167430068406
Generación 196	0.9375224236586386	0.9990227003684243	0.9928673347417648
Generación 197	0.9990150747174514	0.9990227003684243	0.9990165998476461
Generación 198	0.9990150747174514	0.9990227003684243	0.9990165998476461
Generación 199	0.9990150747174514	0.9990227003684243	0.9990165998476461

Explicación de Funcionamiento

Funcionamiento de Cromosoma:

- Mostrar sus genes en diferentes formatos
- Setear sus genes de manera random al inicializarse con el numero indicado de estos
- Calcular función objetivo a partir del cuadrado de la valuación a número real del conjunto de genes dividido el coeficiente
- Calcular su fitness a partir de dividir la valuación de la función objetivo por la valuación de la función objetivo acumulada por todos los cromosomas
- Mutar un gen en una posición random
- Copiar en sí mismo la parte indicada de otro cromosoma

Funcionamiento de Población:

- Mostrar los cromosomas que la conforman
- Inicializarse con el numero de cromosomas, la probabilidad de crossover y la probabilidad de mutación indicados
- Reproducirse, lo que incluiría la selección y crossover
- Seleccionar a los cromosomas dentro de una ruleta de manera probabilística según su fitness
- Crear una ruleta que permita que los cromosomas más aptos tengan mayores posibilidades
- Calcular la puntuación objetivo acumulada por todos los cromosomas que la conforman
- Calcular el mínimo, máximo, segundo máximo y promedio en cada generación (al inicializarse y después de cada reproducción)
- Pasar directo a la siguiente generación a los dos cromosomas con mayor fitness de la generación actual (Elitismo)

Funcionamiento de la Grafica:

- Mostrar la grafica de líneas de los valores máximo, mínimo y promedio de cada generación
- Setearse a través de una tabla enviada desde población que contenga todos estos valores

Funcionamiento Conjunto:

- Se inicializa la población con el numero de cromosomas, el número de genes, la probabilidad de crossover indicados por el usuario *[entrada]*
- Se muestra la generación inicial *[proceso]*
- Se reproducen los cromosomas de la población y se obtiene una nueva generación generalmente mas optima que remplaza a la anterior
- Se agregan los valores máximo, mínimo y promedio de esta generación a una tabla de valores históricos

- Se repite hasta llegar al número de iteraciones indicado por el usuario
- Se generan los gráficos de línea a partir de los valores de cada generación obtenidos de la tabla de valores históricos *[salida]*
- Se analiza si la población convergió a la solución optima o se estanco en un máximo local

Conclusión

A través de agregar la variación de Elitismo al método de la Ruleta implementado en el algoritmo genético se logran corridas considerablemente mejores que cuando no estaba esta modificación, gracias a que a partir de esta se mantienen los dos cromosomas con mayor fitness. Significando esto que estos cromosomas máximos no corren el riesgo de ser alterados por mutaciones inesperadas, lo que causa que la puntuación objetivo máxima de la próxima generación solo pueda ser igual o superior a la de la actual.