

28/05/2019

Trabajo Práctico 1.1: Algoritmo Genético Canónico con Elitismo

UTN FRRo | Profesores: Díaz – Lombardo

Algoritmos Genéticos 2019

Antonelli (44852) – Recalde (44704) – ROHN (41355)

Índice

[Índice 1](#_Toc9099362)

[Introducción 2](#_Toc9099363)

[Código 3](#_Toc9099364)

[Explicación de Funcionamiento 24](#_Toc9099365)

[Conclusión 25](#_Toc9099370)

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:

en el dominio

Donde:

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\_\_**

1. **from** GeneticAlgorithmsElitism.Population **import** Population
2. **from** GeneticAlgorithmsElitism.Graphs **import** Graphic
3. # from GeneticAlgorithmsElitism.Chromosome import Chromosome
4. # import random

7. **if** \_\_name\_\_ == '\_\_main\_\_':
8. # ImportantValues
9. iterationLimit = 50  # 20,100,200  # Population Iterations
10. initPopulationNum = 10  # Initial Population Size
11. chromsomeSize = 30  # Chromosome Size
12. crossoverProb = 0.75  # Probability of CrossOver
13. mutationProb = 0.05  # Probability of Mutation
15. # Initialize
16. **class** Main(object):
17. # First Population
18. pob = Population(initPopulationNum, chromsomeSize, crossoverProb, mutationProb)
19. graphicsData = {'averageOPs': [], 'minOPs': [], 'maxOPs': []}  # Dictionary for Graphics
21. # Iterations
22. **for** iterationCount **in** range(iterationLimit):
23. **print**()
24. averageOP, minOP, maxOP, elitChrom, secondElitChrom = pob.showPopulation(iterationCount)  # Show Actual Population and Return Data
26. # Update Dictionary with important values
27. graphicsData['averageOPs'].append(averageOP)
28. graphicsData['minOPs'].append(minOP)
29. graphicsData['maxOPs'].append(maxOP)
31. # In the last iteration, the chromosomes population mustn't reproduce
32. **if** iterationCount < iterationLimit - 1:
33. pob.reproduce(elitChrom, secondElitChrom)  # Reproduction of Actual Generation
35. # Graph Population's Evolution
36. graph = Graphic(graphicsData, iterationLimit)
37. graph.showPlots()
39. # Last Reproduction Message
40. **print**("Last Generation Reached Correctly")
41. **print**("------------")
42. **print**()
43. **print**()
45. # Final Tables
46. **print**("TABLAS FINALES")
47. **print**()
48. **print**("Población ------------ Mínimo ------------ Máximo ------------ Promedio")
49. **for** value **in** range(iterationLimit):
50. **print**("Generación ", value, ":", graphicsData['minOPs'][value], "---", graphicsData['maxOPs'][value], "---",
51. graphicsData['averageOPs'][value])

* **Chromosome**

1. **import** random

4. **class** Chromosome(object):
5. # Class Attribute (Constant)
6. # coef = random.randint(1, 230 - 1)
7. coef = (2\*\*30)-1  # (2^30)-1
9. # Constructor / Instance Attributes
10. **def** \_\_init\_\_(self, large, newBody):
12. # Chromosome's Genes
13. **if** newBody **is** None:
14. self.body = []
15. **for** \_ **in** range(large):
16. self.body.append(str((random.randint(0, 1))))
17. **else**:
18. self.body = newBody
19. self.large = large
20. # Initialize Objective Function Punctuation and Fitness
21. self.setObjectivePunctuation()
22. self.fitness = 0
24. # Show All Genes of the Chromosome
25. **def** getBody(self):
26. **return** self.body
28. **def** toBinInteger(self):
29. str\_bin\_num = ''.join(str(i) **for** i **in** self.body)
31. **return** int(str\_bin\_num)
33. # Real Number to pass on Objective Function
34. **def** getRealValue(self):
35. num = ''.join(str(i) **for** i **in** self.body)
36. **return** int(num, 2)  # Convert body to String and then to Binary Int
38. **def** calcObjPunc(self):
39. **return** (self.getRealValue() / self.getCoef()) \*\* 2
41. **def** calcFitness(self, totalObj):
42. # if totalObj == 0: totalObj = 1  # Prevent Division by Zero Error
43. self.fitness = (self.getObjectivePunctuation() / totalObj)  # Update Fitness
44. **return** self.fitness
46. # Class Methods
47. @classmethod
48. **def** getCoef(cls):
49. **return** cls.coef
51. @classmethod
52. **def** setCoef(cls, coeficient):
53. cls.coef = coeficient
55. # Getters and Setters
56. **def** getLarge(self):
57. **return** self.large
59. **def** setLarge(self, large):
60. self.large = large
62. **def** getObjectivePunctuation(self):
63. **return** self.objectivePunctuation
65. **def** setObjectivePunctuation(self):
66. self.objectivePunctuation = self.calcObjPunc()
68. **def** getFitness(self):
69. **return** self.fitness
71. **def** setFitness(self, fitness):
72. self.fitness = fitness
74. **def** copy(self, another\_crom, start, end):
75. **for** i **in** range(start, end):
76. self.body[i] = (another\_crom.body[i])
78. **def** mutate(self):
79. mutPos = random.randint(0, self.getLarge()-1)
81. **if** self.body[mutPos] == '0':  # If is a '0' then change to '1', and vice-versa
82. self.body[mutPos] = '1'
83. **elif** self.body[mutPos] == '1':
84. self.body[mutPos] = '0'
86. **print**("Mutated Chrom in position:", mutPos, ":", self.toBinInteger())

* **Population**

1. #!/usr/bin/env python
2. # -\*- coding: utf-8 -\*-
3. **from** GeneticAlgorithmsElitism.Chromosome **import** Chromosome
4. **import** random

7. **class** Population(object):
8. # Class Attributes
9. population = []  # Initial Population (Array of Chromosomes)
10. totalObjPunc = 0  # The Sum of All Objective Functions Punctuation
11. totalFitness = 0  # The Sum of All Objective Values
13. # Constructor / Instance Attributes
14. **def** \_\_init\_\_(self, numChroms, chromSize, crossProb, mutProb):
15. self.numChroms = numChroms
16. self.chromSize = chromSize
17. self.crossProb = crossProb
18. self.mutProb = mutProb
19. # print("Objective Function Coeficient:", Chromosome.getCoef())
20. **print**("Start Algorithm")
21. **for** \_ **in** range(numChroms):
22. oneChrom = Chromosome(chromSize, None)  # Initialization of Chromosomes
23. self.addChrom(oneChrom)  # Add to Population
25. # Show Actual Population and Stats
26. **def** showPopulation(self, numIter):
27. self.setTotalFitness(0)
28. self.setTotalObjPunc(self.calcTotalObjPunc())
29. large = self.getChromSize()
30. averageObjPunc = self.getTotalObjPunc() / len(self.population)
31. fitness = 0
32. maxVal = 0
33. secondMaxVal = 0
34. minVal = 0
35. maxChrom = 0
36. secondMaxChrom = 0
37. minChrom = 0
38. **print**("Population ", (numIter + 1), ":")
39. **for** i **in** range(len((self.population))):
40. fitness = self.population[i].calcFitness(self.getTotalObjPunc())
41. self.updateTotalFitness(fitness)
42. **if** i == 0:
43. maxVal = fitness
44. minVal = fitness
45. **elif** fitness > maxVal:
46. maxVal = fitness
47. maxChrom = i
48. **elif** fitness < minVal:
49. minVal = fitness
50. minChrom = i
51. **elif** (fitness > secondMaxVal) **and** (fitness < maxVal):
52. secondMaxVal = fitness
53. secondMaxChrom = i
54. **for** j **in** range(large):
55. **print**(self.population[i].getBody()[j], end='')
56. **print**()
57. fitness = self.getTotalFitnessAverage()
58. **print**()
59. **print**("Chromosome --- Value --- Objective Punctuation --- Fitness")
60. **print**("Max Values: Chrom Nº", maxChrom, "with:", self.population[maxChrom].getRealValue(), "Val,",
61. self.population[maxChrom].getObjectivePunctuation(), "OP,", round(maxVal, 4), "Fit")
62. **print**("Second Max Values: Chrom Nº", secondMaxChrom, "with:", self.population[secondMaxChrom].getRealValue(), "Val,",
63. self.population[secondMaxChrom].getObjectivePunctuation(), "OP,", round(secondMaxVal, 4), "Fit")
64. **print**("Min Values: Chrom Nº", minChrom, "with:", self.population[minChrom].getRealValue(), "Val,",
65. self.population[minChrom].getObjectivePunctuation(), "OP,", round(minVal, 4), "Fit")
66. **print**("Average OP:", averageObjPunc, "--- Average Fitness:", fitness)  # round(fitness,6)
67. **print**()
68. # Return Important Data to use on Graphics
69. elitChrom = self.population[maxChrom]
70. secondEliChrom = self.population[secondMaxChrom]
71. **return** (averageObjPunc, self.population[minChrom].getObjectivePunctuation(),
72. self.population[maxChrom].getObjectivePunctuation(), elitChrom, secondEliChrom)
74. # Calculate Total of Objective Functions Punctuation in the actual Generation
75. **def** calcTotalObjPunc(self):
76. acumObjPunc = 0
77. **for** chromosome **in** self.population:
78. acumObjPunc += chromosome.getObjectivePunctuation()  # Add Every Objective Function Punctuation
79. #  self.setTotalObjPunc(acumulator)
80. **return** acumObjPunc
82. # Update Total Fitness
83. @classmethod
84. **def** updateTotalFitness(cls, fitness):
85. cls.totalFitness += fitness
87. # Add to Population
88. **def** addChrom(self, Chrom):
89. self.population.append(Chrom)
91. # Reproduction
92. **def** reproduce(self, elitChrom, secondElitChrom):
93. parents = []  # List of Potential Parents
94. newGeneration = []  # List of Children
95. **print**("Roulette Results: ", end='')
97. # TO-DO:Send the second best Chromosome
99. # Elitism
100. parents.append(elitChrom)
101. parents.append(secondElitChrom)
102. self.addChildren(parents[0], parents[1], newGeneration)
104. **for** \_ **in** range(2, len(self.population), 2):
105. **for** i **in** range(2):
106. lastParent = self.roulette()  # Parents Selected by Roulette
107. parents.append(self.population[lastParent])
108. **print**()
109. **for** i **in** range(2, len(parents), 2):
110. father1 = parents[i]
111. father2 = parents[i + 1]
112. **if** self.crossPosibility():  # CrossOver Probability Evaluation
113. son1, son2 = self.cross(father1, father2)  # CrossOver
114. **print**("Successful CrossOver in reproduction:", (i + 2) / 2 - 1)  # Only Print
115. **else**:
116. son1, son2 = self.copy(father1, father2)  # Direct Assignation (Without CrossOver)
117. **print**("CrossOver didn't happen in reproduction:", (i + 2) / 2 - 1)  # Only Print
119. # Individual Mutation Probability Evaluation even when Crossover is not successful
120. **if** self.mutationPosibility():
121. son1.mutate()
122. **if** self.mutationPosibility():
123. son2.mutate()
124. son1.setObjectivePunctuation()
125. son2.setObjectivePunctuation()
126. self.addChildren(son1, son2, newGeneration)
127. **print**()
128. **print**("Successful Elitism Applied")
129. **print**()
130. self.replacePopulation(newGeneration)
131. self.setTotalFitness(0)
133. # Genetic Operator (Roulette Method)
134. **def** roulette(self):
135. # Generator of a Bidimensional List (Fitness Range of Chromosomes)
136. newRoulette = [[0] \* 2 **for** \_ **in** range(len(self.population))]
137. acum = 0  # Acumulator of Relative Fitness from 0 to 1 (Fills Roulette)
138. **for** i **in** range(len(self.population)):
139. newRoulette[i][0] = acum  # Range Min: Last Acum Value
140. acum += round(self.population[i].getFitness(), 6)  # Acum's Value From Zero
141. newRoulette[i][1] = acum  # Range Max: New Acum Value
142. ranNum = round(random.uniform(0, 1), 6)  # Random Number from 0.000000 to 0.999999
143. # print("Random: ", ranNum)  # Only Print
144. **for** i **in** range(len(newRoulette)):
145. **if** newRoulette[i][0] < ranNum < newRoulette[i][1]:
146. # Return Selected Chromosome if the Random Number Exists in its Range
147. **print**(i, end=', ')
148. **return** i
150. **def** crossPosibility(self):  # CrossOver posibility evaluation
151. **if** self.getCrossProb()\*100 >= random.randint(1, 100):
152. **return** True
153. **else**:
154. **return** False
156. **def** cross(self, parent1, parent2):
157. crom\_size = parent1.getLarge()
158. son1 = Chromosome(crom\_size, None)
159. son2 = Chromosome(crom\_size, None)
160. cut = random.randint(1, crom\_size - 2)  # Random Cut Point (Except by zero or all genes)
162. son1.copy(parent1, 0, cut)
163. son1.copy(parent2, cut, crom\_size)
165. son2.copy(parent2, 0, cut)
166. son2.copy(parent1, cut, crom\_size)
168. **print**()
169. **print**("Son 1:", son1.toBinInteger())  # Only Print
170. **print**("Son 2:", son2.toBinInteger())  # Only Print
171. **print**("Cut Point on:", cut)  # Only Print
172. **return** son1, son2
174. **def** copy(self, chrom1, chrom2):
175. # newGeneration.append(chrom1)
176. # newGeneration.append(chrom2)
177. son1 = chrom1
178. son2 = chrom2
179. **print**()
180. **print**("Son 1 (Identical):", self.listToInt(chrom1.getBody()))  # Only Print
181. **print**("Son 2 (Identical):", self.listToInt(chrom2.getBody()))  # Only Print
182. **return** son1, son2
184. **def** mutationPosibility(self):  # Mutation posibility evaluation
185. **if** self.getMutProb() \* 100 >= random.randint(1, 100):
186. **return** True
187. **else**:
188. **return** False
190. **def** addChildren(self, son1, son2, newGeneration):
191. newGeneration.append(son1)
192. newGeneration.append(son2)
194. **def** replacePopulation(self, newGeneration):  # Replace All Population in every Iteration
195. self.population = []
196. **for** i **in** range(len(newGeneration)):
197. self.population.append(newGeneration[i])
199. **def** listToInt(self, arr):
200. num = ''.join(str(i) **for** i **in** arr)
201. **return** int(num)
203. # Class Getters and Setters
204. @classmethod
205. **def** getTotalObjPunc(cls):
206. **return** cls.totalObjPunc
208. @classmethod
209. **def** setTotalObjPunc(cls, total):
210. cls.totalObjPunc = total
212. @classmethod
213. **def** getTotalFitness(cls):
214. **return** cls.totalFitness
216. @classmethod
217. **def** setTotalFitness(cls, total):
218. cls.totalFitness = total
220. @classmethod
221. **def** getTotalFitnessAverage(cls):
222. **return** cls.totalFitness / len(cls.population)
224. # Getters and Setters
225. **def** getNumChroms(self):
226. **return** self.numChroms
228. **def** setNumChroms(self, numChroms):
229. self.numChroms = numChroms
231. **def** getChromSize(self):
232. **return** self.chromSize
234. **def** setChromSize(self, chromSize):
235. self.chromSize = chromSize
237. **def** getCrossProb(self):
238. **return** self.crossProb
240. **def** setCrossProb(self, crossProb):
241. self.crossProb = crossProb
243. **def** getMutProb(self):
244. **return** self.mutProb
246. **def** setMutProb(self, mutProb):
247. self.mutProb = mutProb

* **Graphs**

1. **from** matplotlib **import** pyplot
2. # import math
3. # import numpy as np
5. # pyplot.plot(x1,y1,'b-',x2,y2,'r-',x3,y3,'g-')
6. # pyplot.legend('value 1', 'value 2', 'value 3')
7. # pyplot.savefig('generations.png')  # Save Image --> Better Save Manually

10. **class** Graphic(object):
11. **def** \_\_init\_\_(self, graphicsData, iterationLimit):
12. self.averageOPs = graphicsData['averageOPs']
13. self.minOPs = graphicsData['minOPs']
14. self.maxOPs = graphicsData['maxOPs']
15. self.generations = []
16. **for** i **in** range(iterationLimit):
17. self.generations.append(i)
18. # pyplot.ion()
20. # Show one Plot with all Graphs
21. **def** showPlots(self):
22. # with pyplot.style.context(('dark\_background')):
23. self.drawAll(self.minOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA MÍNIMA", 221)
24. self.drawAll(self.maxOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA MÁXIMA", 222)
25. self.drawAll(self.averageOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA PROMEDIO", 212)
26. pyplot.show()  # Draw all the "Subplots"
28. # Show one Graph in each Plot
29. **def** showPlotsApart(self):
30. self.draw(self.minOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA MÍNIMA")
31. self.draw(self.maxOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA MÁXIMA")
32. self.draw(self.averageOPs, self.generations, "Puntuación Objetiva", "Generación", "PUNTUACIÓN OBJETIVA PROMEDIO")
34. # Generate one Subplot for every Graph (and Show all Graphs)
35. **def** drawAll(self, axisY, axisX, labY, labX, title, region):
36. pyplot.subplot(region)
37. pyplot.axis([-1, len(self.generations), 0, 1])
38. pyplot.grid(True)
39. pyplot.plot(axisX, axisY)
40. pyplot.ylabel(labY)
41. pyplot.xlabel(labX)
42. pyplot.title(title)
44. # Generate one plot for every Graph
45. **def** draw(self, axisY, axisX, labY, labX, title):
46. pyplot.axis([-1, len(self.generations), 0, 1])
47. pyplot.grid(True)
48. pyplot.plot(axisX, axisY)
49. pyplot.ylabel(labY)
50. pyplot.xlabel(labX)
51. pyplot.title(title)
52. pyplot.show()

Repositorio GitHub:

<https://github.com/alereca/Genetic-Algorithms-Cpp>

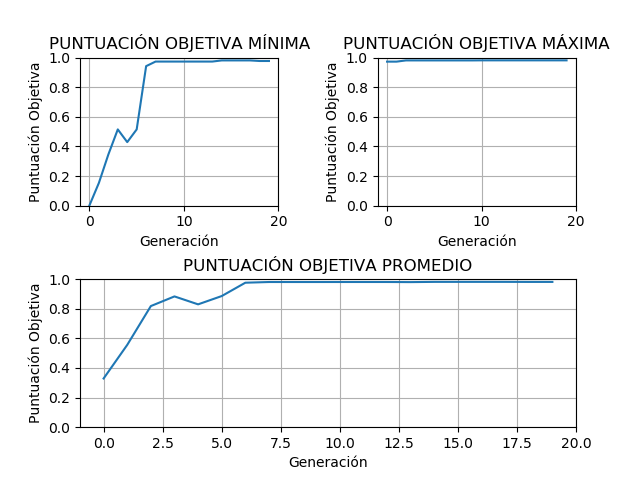
Derivadado de la anterior version sin Elitismo:

<https://github.com/NicoCaptain/Genetic-Algorithms>

Gráficas y Tablas

***Python: Matplotlib***

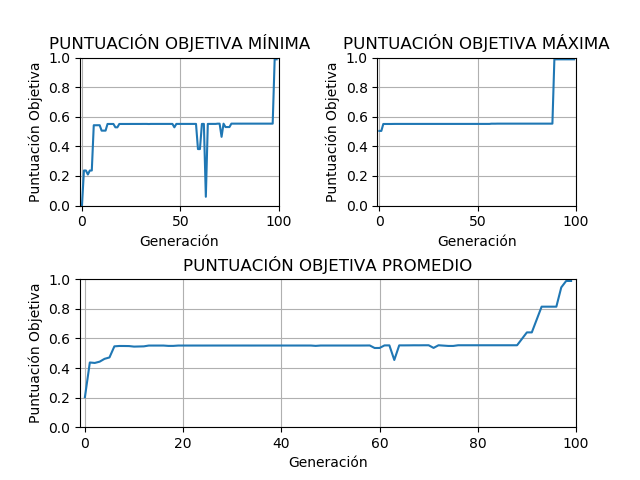
## Algoritmo en 20 Generaciones



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

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

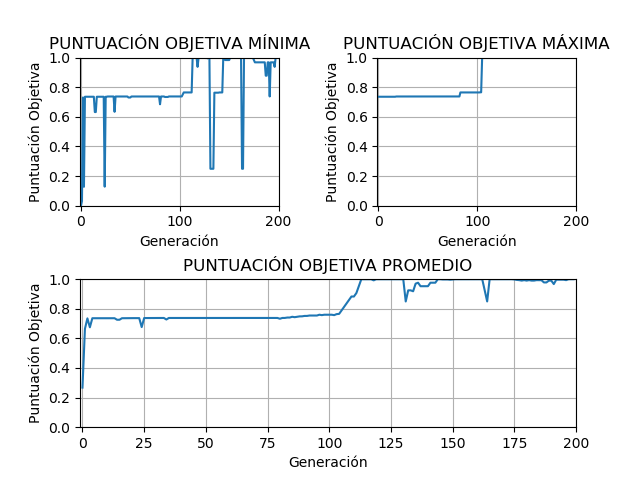
## Algoritmo en 100 Generaciones



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

|  |  |  |  |
| --- | --- | --- | --- |
| 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.