Universidade Federal de Santa Catarina

Centro Tecnológico

Departamento de Automação e Sistemas

UNIVERSIDADE FEDERAL
DE SANTA CATARINA

Exercício 1: Algoritmo Genético

Inteligência Artificial

Juliano Ricardo da Silva
Prof. Dr. Eric Antonelo

Florianópolis - SC

Fevereiro de 2022

Conteúdo

1 Algoritmo Genético

O algoritmo genético programado pode ser conferido abaixo:

```
#Juliano Ricardo da Silva
#Nao usei o template fornecido

#Algoritmo genetico

L = 4 * 8 #size of chromossome in bits

import struct
import random
import math
import matplotlib.pyplot as plot
import numpy as np
import os

def floatToBits(f):
        s = struct.pack('>f', f)
        return struct.unpack('>L', s)[0]

def bitsToFloat(b):
        s = struct.pack('>L', b)
        return struct.unpack('>f', s)[0]

#exemplo 1.23 -> '0010111100'

def get_bits(x):
        x = floatToBits(x)
        N = 4 * 8
        bits = ''
        for bit in range(N):
                b = x & (2**bit)
                bits += '1' if b > 0 else '0'
        return bits

#exemplo '00010111100' -> 1.23

def get_float(bits):
        x = 0
        assert(len(bits) == L)
        for i, bit in enumerate(bits):
                bit = int(bit)  #0 or 1
                x += bit * (2**i)
        return bitsToFloat(x)
```

```
43
44 #size of population
45 global n
46
47 #gets population size by user input
48 def pop_len():
49         p = int(input("digit the size of population:  "))
50         if (p<=2):
51                 print("minimum value is 4")
52                 p = 4
53                 return p
54         elif(p%2 != 0):
55                 print("odd number")
56                 p = p - 1
57                 print("changed to even")
58                 return p
59         else:
60                 return p
61
62 #gets the generation number by user input
63 def gen_len():
64         p = int(input("digit the number of generations:  "))
65         return p
66
67 #generates a random list of people (chromossomes)
68 def population():
69         for i in range(g):
70                 p = random.SystemRandom().uniform(0, math.pi)
71                 person = get_bits(p)
72                 people.append(person)
73
74 #fitness calculation for each chromossome
75 def calc_fitness(p):
76         for t in range(len(p)):
77                 p[t] = get_float(p[t])
78                 fitness    = p[t] + abs(math.sin(32*p[t]))
79                 #if fitness doesnt respect the limits [0,pi), their fitness
                       is equal zero to avoid their spread
80                 if fitness < 0 or fitness > math.pi:
81                         fitness = 0
82                         fitnessList.append(fitness)
83                 else:
84                      fitnessList.append(fitness)
85         return fitnessList
86
```

```
87
88 #selects two chromossomes to posterior crossover
89 def roullette_selection(listpop,fit):
90         store = []
91         st = []
92         while not(len(store) == 2):
93                         store.extend((random.choices(listpop,fit,k=1)))
94                         for i in store:
95                                 st.append(get_bits(i))
96         #while there are two chromossomes
97         while st[-1] == st[-2]:
98                 print("********************TWO EQUAL CHROMOSSOMES, MAKE NEW
                    SELECTION************")
99                 st = []
100                store.extend((random.choices(listpop,fit,k=1))) #repeat
                    selection in population
101                for i in store: st.append(get_bits(i))
102        return  st[-2:]
103
104 def roullette(lst, ft):
105        store2 = []
106        c = 0
107        while not(len(store2) == 2):
108                        lst=lst[-n:]
109                        ft = ft[-n:]
110                        store2.extend((random.choices(lst,ft,k=1)))
111        while (store2[-1] == store2[-2]):
112                print("********************TWO EQUALS CHROMOSSOMES, MAKE NEW
                    SELECTION************")
113                store2[-1] = lst[0+c]
114                store2[-2] = lst[-1+c]
115                c+=1
116                print("NOW, THE COUPLE IS: ", store2)
117        return store2
118
119 #single point crossover
120 def crossover():
121        pc = random.randint(1,10)/10
122        if pc > 1: pc = 1
123        if pc < 0.1: pc = 0.1
124        if   0.1 <= pc <= 0.7:
125                print('crossover resulted in: \n')
126                d1 = dad[0:16]+mom[16:32]
127                d2 = mom[0:16]+dad[16:32]
128                print(d1,d2)
```

```
129
130         else:
131                 print("identhical copy \n")
132                 d1 = dad[:]
133                 d2 = mom[:]
134         descendants.append(d1)
135         descendants.append(d2)
136         return descendants
137
138 #mutation
139 def mutation():
140     pm = random.randint(1,1000)
141     if pm > 1000: pm = 1000
142     if pm < 1: pm = 1
143     if  pm == 1:
144         md = []
145         sd = descendants[random.randint(0,1)]
146         if sd == descendants[-2]:
147             new_population.append(descendants[-1])
148         else:
149             new_population.append(descendants[-2])
150
151         ap = random.randint(0,32)
152         if (0 >= ap or ap >= 32): ap = 0
153         if sd[ap] == '0':
154                 m = '1'
155         else:
156                 m = '0'
157         md = sd[:ap] + m + sd[ap+1:]
158         print("selected descendant:",sd)
159         print("position:",ap)
160         print("the mutated chromossome is: \n",md)
161         new_population.append(md)
162         return new_population
163     else:
164         new_population.append(descendants[-2])
165         new_population.append(descendants[-1])
166         print("doesn't occurred a mutation")
167     return new_population
168
169 #fitness calculation for each chromossome after their evolution
170 def calc_avg_fitness(newfit):
171                 avg_fit = sum(newfit)/g
172                 return avg_fit
173
```

```
174 chrome = []
175 n = pop_len()
176 g = n
177 iterations = 0
178 new_pop = []
179 iters = gen_len()
180 people = []
181 new_chrome = []
182 chromossome = []
183 avg = []
184 firstgen = []
185 secgen = []
186 lastgen = []
187
188 while iterations != iters:
189     iterations+=1
190
191     #first generation to be created and selected until chromossomes_evolved
          = size of population
192     if(iterations==1 and people ==[]):
193             print("*****************GERACAO NRO **************", iterations)
194             print("The population number is ", g)
195             population()   #population
196             print("\n*****The population of chromossomes in bit chain: *****
                  \n",people)
197             fitnessList = []
198             calc_fitness(people) #calculates the fitness for each
                  chromossome
199             print("\n*****List of fitness: *****\n ",fitnessList)
200             avg.append(calc_avg_fitness(fitnessList)) #calculates the
                  average fitness of the first population
201             print("\n******AVERAGE FITNESS: ",avg)
202             while(len(chrome) != g):
203                 couple = []
204                 couple = roullette_selection(people,fitnessList) #selects a
                      couple of chromossomes
205                 print("\n*****The selected couple is: *****\n", couple)
206                 dad = couple[0]
207                 mom = couple[1]
208                 descendants = []
209                 crossover() #chromossomes crossover (or not) to generate
                      descendants
210                 new_population = []
211                 mutation()  #chromossomes mutation (or not) to generate
                      descendants
```

```
212                 for i in new_population:
213                     chrome.append(i)
214                 print("\n*****New chromossomes: ******\n",chrome) #the new
                        population of chromossomes
215             x = []
216             new_gen = chrome
217             print("\n*****population list to create next generation is*****\
                    n", new_gen)
218             for i in new_gen:
219                 firstgen.append(get_float(i))
220
221     #if the first generation was created, the new one will be generated
            based on the fitnesslist from previous generation
222     if(people != []):
223                 for i in new_gen:
224                         new_pop.append(i)
225                         chromossome = []
226                         x = []
227                         new_population = []
228                         fitnessList = []
229                         descendants = []
230                         new_chrome = []
231
232     #is the first generation already created? therefore, repeats the natural
            selection process for each next generation
233     if(iterations >= 2 and len(avg) != iters):
234             print("************************************************")
235             print("***************GENERATION NUMBER: ", iterations)
236             print("************************************************")
237             print("\n New population is: \n",    new_pop[-n:])
238             calc_fitness(new_pop[-n:])
239             print("\n New fitness list is: \n", fitnessList[-n:])
240             avg.append(calc_avg_fitness(fitnessList[-n:]))
241             #while number of chromossomes is different of population size,
                    repeats the process
242             while(len(chromossome) != g):
243                     x = []
244                     x = roullette(new_pop, fitnessList)#selects two
                            chromossomes to crossover
245                     print("\n*****Couple selected:  *****\n", x[-n:])
246                     dad = x[0]
247                     mom = x[1]
248                     crossover()
249                     mutation()
250
```

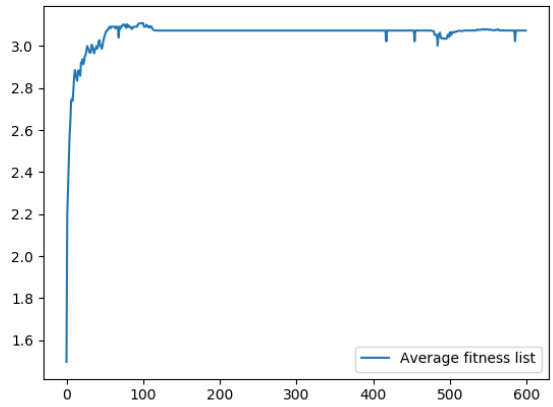
```
251                         for i in [new_population[-2],new_population[-1]]:
252                                 new_chrome.append(i)
253                         chromossome.extend([new_chrome[-2],new_chrome[-1]])
254                         print("\n*****New chromossomes******\n", chromossome)
255
256
257                 new_gen  = chromossome
258                 print("\n*****population list to create next generation is*****\
                        n", new_gen)
259
260     if(iterations == 10):
261         for j in chromossome:
262                 secgen.append(get_float(j))
263     if(iterations == 600):
264         for k in chromossome:
265                 lastgen.append(get_float(i))
266
267 print("done!")
268 print("population length: ", len(chrome))
269 print("AVG FITNESS LIST: ", avg)
270
271 print("FIRST GEN:", firstgen)
272 print("SECOND GEN:", secgen)
273 print("LAST GEN:", lastgen)
274
275 #plot avg fitness list
276 xx_value = list(range(0,iters))
277 yy_value = [f for f in avg]
278 plot.plot(xx_value, yy_value, label = "Average fitness list")
279 plot.legend()
280 plot.show()
281
282 yval  = [f for f in firstgen] #1st generation chromossomes
283 yval2 = [g for g in secgen]   #10th generation chromossomes
284
285 fg = []
286 sg = []
287 lg = []
288
289 #calculates the fitness from the generations of chromossomes selected
290 def fit_gen(g,h):
291         for t in g:
292                 fitness   = t + abs(math.sin(32*t))
293                 h.append(fitness)
294         return h
```

```
295 fit_gen(firstgen , fg)
296 fit_gen(secgen ,   sg)
297 fit_gen(lastgen , lg)
298
299 print("FIRST FIT GEN:", fg)
300 print("SECOND FIT GEN:", sg)
301 print("LAST FIT GEN:", lg)
302
303
304 #math fitness function
305 x = np.linspace(0,math.pi,200)
306 y =  x + abs(np.sin(32*x))
307
308 #plot fitness function vs x and chromossomes vs fitness
309 fig , ax = plot.subplots()
310 ax.plot(x, y, color ='blue', label = "fitness function")
311 ax.scatter(firstgen , fg, color='orange', label = "Chomossomes First Gen")
312 ax.scatter(secgen , sg, color='black', label =   "Chomossomes Sec Gen")
313 ax.scatter(lastgen , lg, color='green', label =   "Chomossomes Third Gen")
314
315 plot.legend()
316 plot.show()
317
318 #pause
319 os.system("pause")
```

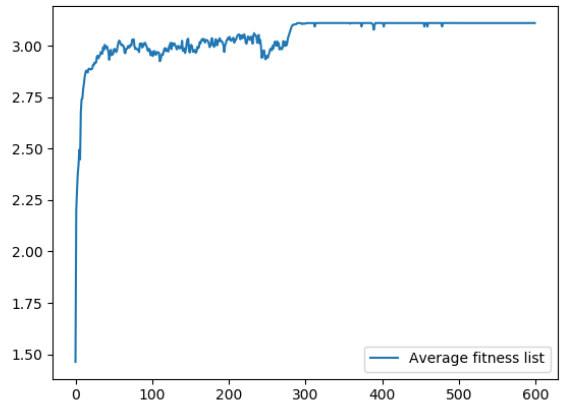
2 Análise

2.a Aptidão média

Os gráficos de aptidão média, considerando **pm=0.001 e pc = 0.7** são mostrados na figura 1:



(a) População igual a 60 cromossomos.



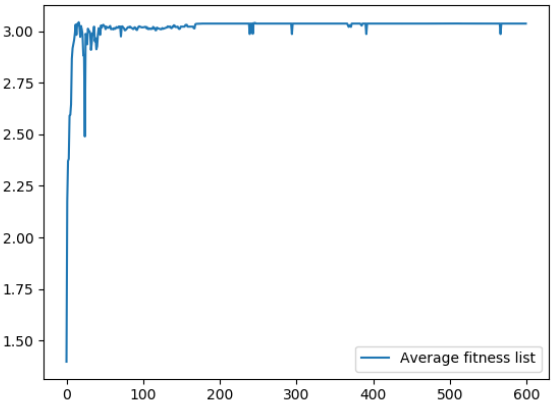
(b) População igual a 180 cromossomos.

Figura 1: Gráfico da aptidão média para várias gerações.

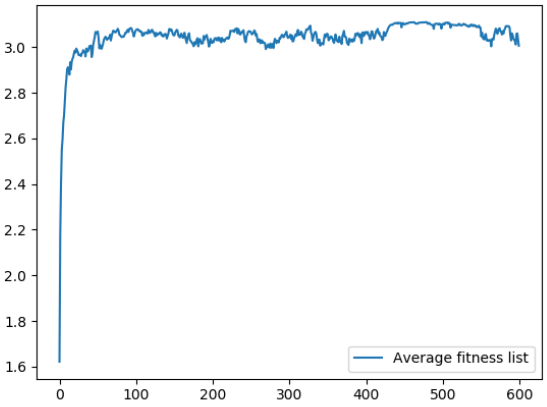
Como mostrado na figura 1, com uma taxa de mutação baixa e uma taxa de cruzamento relativamente alta, tem-se que para populações menores, a aptidão média oscila nas primeiras 100 gerações e depois tende a crescer e estabilizar-se num valor de aptidão bem superior àquela das primeiras gerações. Para populações bem maiores, há um comportamento similar, mas há maior variabilidade no valor da aptidão média por um período bem maior de tempo (em gerações).

Isto mostra que, para populações maiores, o crescimento da aptidão média ao decorrer das gerações é mais devagar, provavelmente, por conta da maior variabilidade genética da população, fazendo com que diversos cromossomos de menor aptidão se cruzem mais vezes do que ocorre em pequenas populações.

Os gráficos de aptidão média, considerando **pm=0.001 e pc = 0.2** são mostrados abaixo:



(a) População igual a 60 cromossomos.

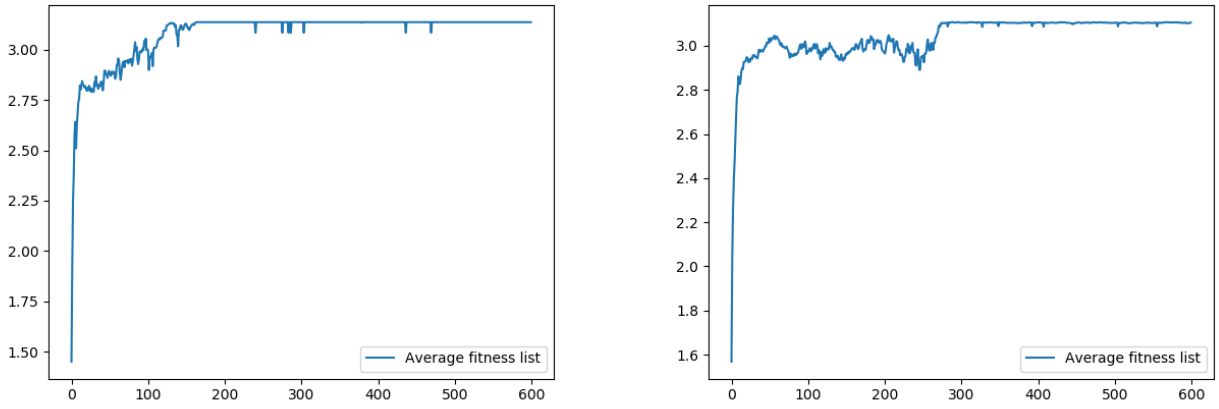


(b) População igual a 180 cromossomos.

Figura 2: Gráfico da aptidão média para várias gerações.

Para uma taxa de cruzamento menor, nota-se que houve uma oscilação maior no valor de aptidão média nas primeiras 100 gerações da população de tamanho igual a 60. Enquanto que, para uma população três vezes maior, o valor médio da aptidão oscila ao decorrer de todas as gerações, mostrando que menos cruzamentos em uma população maior torna a busca por uma solução ótima mais vagarosa.

Na figura 3 são mostrados os gráficos de aptidão média, considerando **pm=0.001 e pc = 0.9**. Logo, pode-se concluir que, tanto para populações menores, quanto para populações maiores, o aumento na taxa de cruzamento causa maior inserção de novos cromossomos na população, os quais tendem a serem mais aptos que os de gerações anteriores. Além disso, a busca por um valor ótimo é mais rápida, convergindo para um valor de aptidão média maior do que a obtida com taxa de cruzamento menor.

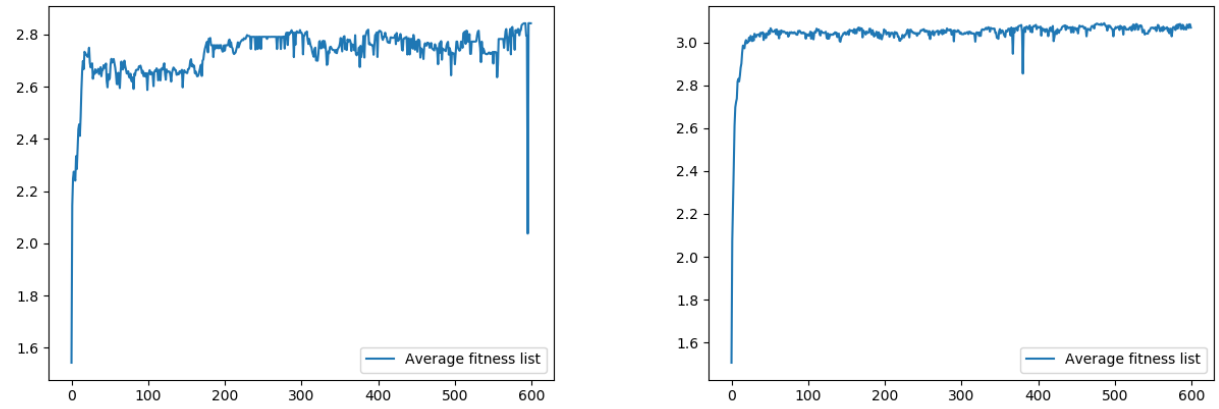


(a) População igual a 60 cromossomos.

(b) População igual a 180 cromossomos.

Figura 3: Gráfico da aptidão média para várias gerações.

Os gráficos de aptidão média, considerando **pm=0.01 e pc = 0.7** são mostrados na figura 4:



(a) População igual a 60 cromossomos.

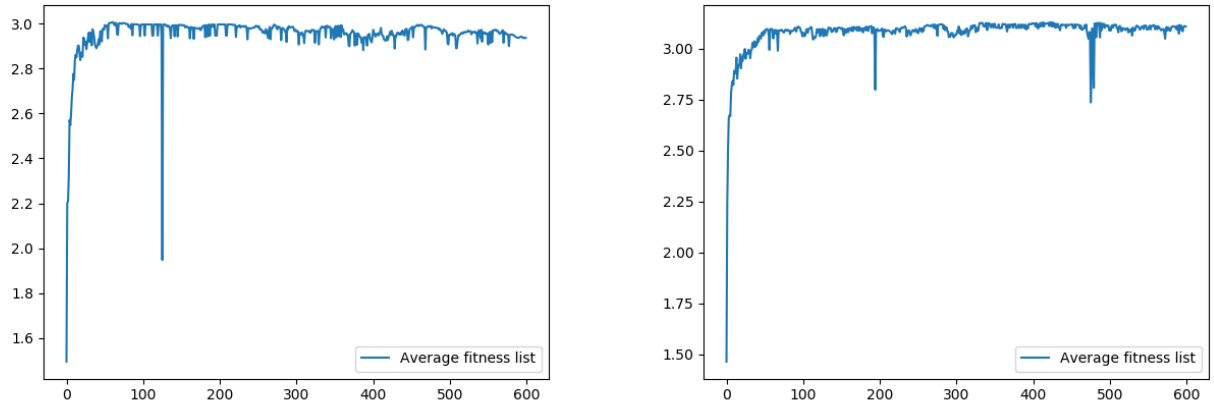
(b) População igual a 180 cromossomos.

Figura 4: Gráfico da aptidão média para várias gerações.

O aumento da taxa de mutação causou, em uma população menor, oscilações significativas no valor da aptidão média, estabilizando em valor cuja diferença para as aptidões médias das primeiras gerações é menor do que o observado em situações anteriores. Já para uma população muito maior, este aumento na

taxa de mutação também ocasionou variações, mas o valor de aptidão média alcançado foi maior. Com isso, pode-se concluir que as mutações em populações maiores tendem a diversificar mais os cromossomos, cuja recombinação contínua ao decorrer das gerações causam um aumento na aptidão média da população, enquanto que em populações menores, há um fator de aleatoriedade mais decisivo que, portanto, pode levar a um comportamento mais caótico na aptidão média.

Os gráficos de aptidão média, considerando **pm=0.01 e pc = 0.2** são mostrados na figura 5:



(a) População igual a 60 cromossomos.

(b) População igual a 180 cromossomos.

Figura 5: Gráfico da aptidão média para várias gerações.

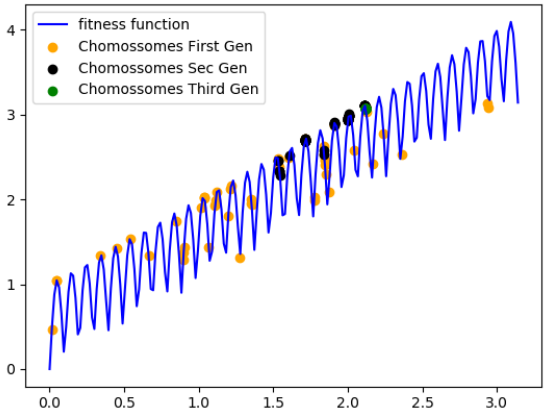
Finalmente, para uma menor taxa de cruzamento e maior taxa de mutação, tem-se que o comportamento é bastante parecido em populações maiores e menores.Isso indica que uma taxa de cruzamento baixa aliada a uma taxa de mutação alta, resultam em um comportamento bastante oscilatório no valor da aptidão média, haja vista que ser, por um lado, a mutação causa atrapalha a busca por uma solução ótima, apesar de evitar uma estagnação na aptidão média, por outro lado, a baixa taxa de cruzamento diminui a probabilidade de um cromossomo que foi modificado (mutante) ser recombinado com outro modificado.

2.b Aptidão em função do cromossomo

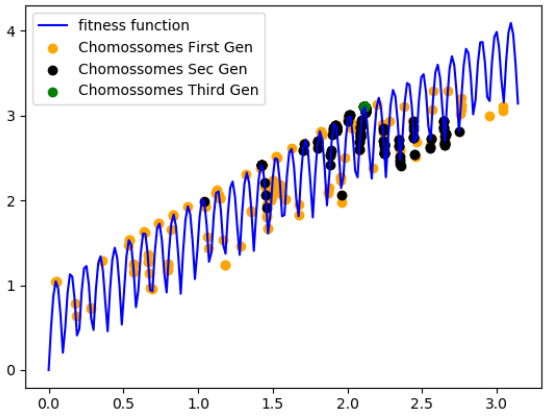
Finalmente, visando visualizar melhor o aumento da aptidão ao decorrer das gerações, foram plotados para cada situação descrita anteriormente os gráficos da aptidão em função dos cromossomos (para a primeira, décima e última geração) e também a função matemática do fitness.

Para a primeira situação, possível ver no caso da população menor mostrada na figura 6(a) que, logo na primeira geração de cromossomos, há um cromossomo com a maior aptidão de todas. Entretanto, os outros cromossomos da população inicial encontram-se bastante dispersos na parte baixa do gráfico de fitness, indicando portanto que a aptidão média será baixa. Para a décima geração, nota-se uma concentração maior dos cromossomos em um valor um pouco mais elevado de aptidão média, mas também disperso no gráfico. A última geração, entretanto, está toda concentrada num valor de mutação maior, sem espalhamento da aptidão dos cromossomos, o que indica que a aptidão média da última geração é superior. Para uma população maior, o mesmo fenômeno ocorre, mudando apenas a maior variabilidade da primeira e décima gerações.

Observação: na legenda dos gráficos, considerar, respectivamente: a segunda geração como sendo a décima e a terceira é a última geração (última iteração do algoritmo genético).



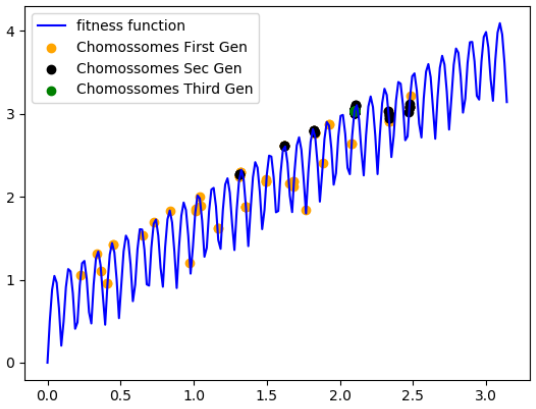
(a) População igual a 60 cromossomos.



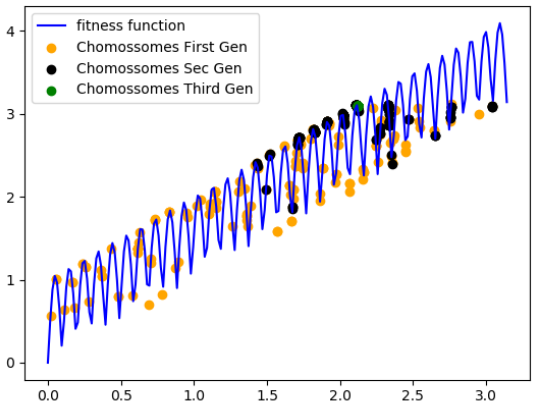
(b) População igual a 180 cromossomos.

Figura 6: Gráfico da aptidão em função do cromossomo para diversas gerações para pm =0.001 e pc = 0.7.

Para a segunda e demais situações, o mesmo fenômeno ocorre:

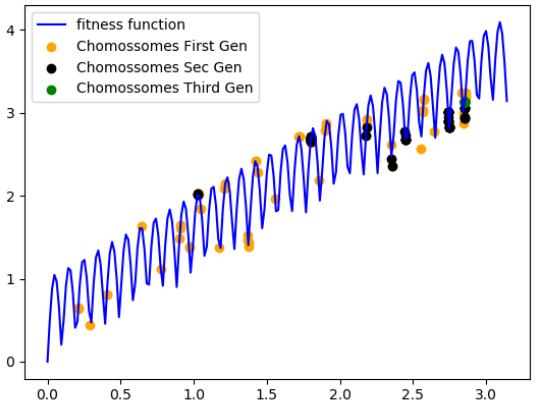


(a) População igual a 60 cromossomos.

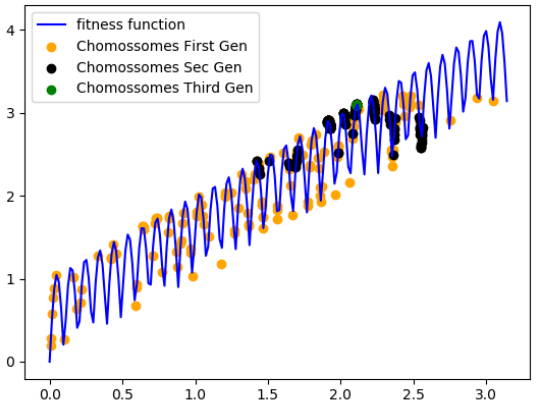


(b) População igual a 180 cromossomos.

Figura 7: Gráfico da aptidão em função do cromossomo para diversas gerações para pm =0.001 e pc = 0.2.

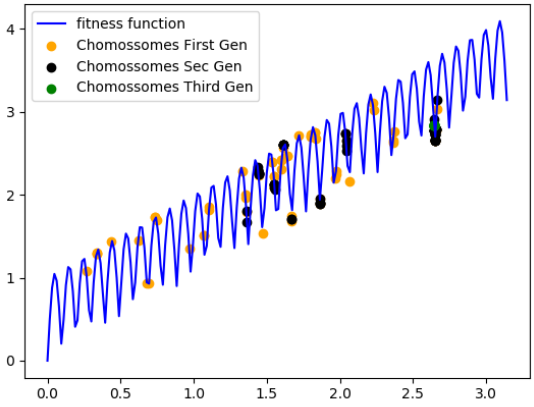


(a) População igual a 60 cromossomos.

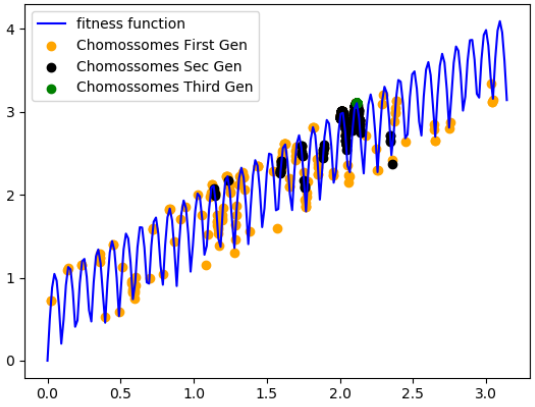


(b) População igual a 180 cromossomos.

Figura 8: Gráfico da aptidão em função do cromossomo para diversas gerações para pm =0.001 e pc = 0.9.

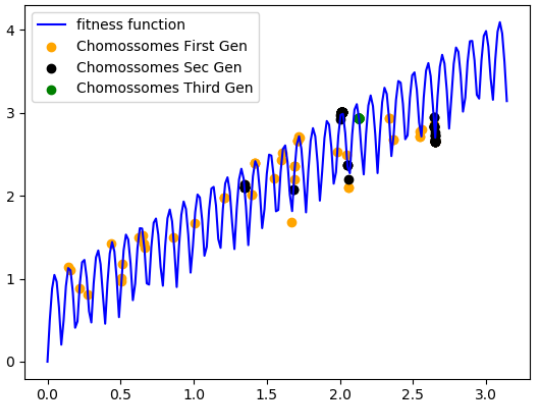


(a) População igual a 60 cromossomos.

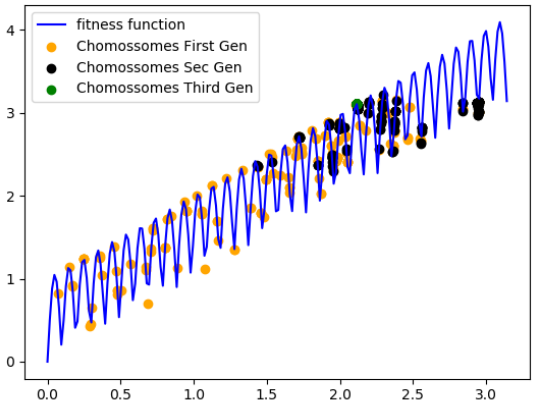


(b) População igual a 180 cromossomos.

Figura 9: Gráfico da aptidão em função do cromossomo para diversas gerações para pm =0.01 e pc = 0.7.



(a) População igual a 60 cromossomos.



(b) População igual a 180 cromossomos.

Figura 10: Gráfico da aptidão em função do cromossomo para diversas gerações para pm =0.01 e pc = 0.2.