تقریب توابع ریاضی

به کمک الگوریتم ژنتیک

پروژه درس هوش مصنوعی

گردآورنده: میرنیما قاسمیان

استاد درس: دکتر آرش عبدی

هدف پروژه: ارزیابی میزان تسلط به مفاهیم پایه الگوریتم های تکاملی (در اینجا برنامه نویسی ژنتیک)

پیاده سازی:

ابتدا تعدادی ورودی را در نظر گرفته و خروجی آنها را با جایگذاری در تابع اصلی به دست می آوریم. سپس

عملگر هایی که در این پیاده سازی مورد استفاده قرار می گیرند را معرفی میکنیم مانند :

عملگر های دو عملوندی: "^","-","+","*","/" عملگرهای تک عملوندی:

"sin"،"cos"

به صورت کلی سعی داریم با آموزش تعدادی از نقاط، تابع اولیه را مجدد به دست آوریم و پیش بینی کنیم ضابطه ی اولیه چه بوده است!

- به عنوان نخستین گام باید جمعیت اولیه(population) را تولید کنیم.

جمعیت تولید شده شامل n تا کروموزوم میباشد که تعداد n به عنوان ورودی Population Constructor داده می شود.

هر کروموزوم تخمینی از حالت اولیه مسئله می باشد(در واقع هرکدام یک درخت اولیه برای پیشبرد مسئله میباشد) که در ادامه به هر کروموزوم یک مقدار شایستگی تخصیص می دهیم (fitness)

```
101 class Population:
102
103     def __init__(self, size, depth, max_depth, num_selected, functions, terminals):
104
105         self.size = size
106         self.num_selected = num_selected
107         self.max_depth = max_depth
108         self.list = self.create_population(self.size, functions, terminals, depth)
109
110     def create_population(self, number, functions, terminals, depth):
111         population_list = []
112         for i in range(number):
113             if random.random() < 0.5:
114                 population_list.append(Chromosome(terminals, functions, depth, 'full'))
115             else:
116                 population_list.append(Chromosome(terminals, functions, depth, 'grow'))
117         return population_list
118
```

```python
class Chromosome:

    def __init__(self, terminals, functions, depth, method='full'):

        self.gen = []
        self.depth = depth
        self.func_set = functions
        self.terminal_set = terminals
        self.fitness = None
        if method == 'grow':
            self.grow()
        elif method == 'full':
            self.full()

    def grow(self, level = 0):
        if level == self.depth:
            self.gen.append(random.choice(self.terminal_set))
        else:
            if random.random() > 0.3:
                val = random.choice(self.func_set[2] + self.func_set[1])
                if val in self.func_set[2]:
                    self.gen.append(val)
                    self.grow(level + 1)
                    self.grow(level + 1)
                else:
                    self.gen.append(val)
                    self.grow(level + 1)
            else:
                val = random.choice(self.terminal_set)
                self.gen.append(val)

    def full(self, level = 0):

        if level == self.depth:
            self.gen.append(random.choice(self.terminal_set))
        else:
            val = random.choice(self.func_set[1] + self.func_set[2])
            if val in self.func_set[2]:
                self.gen.append(random.choice(self.func_set[2]))
                self.full(level + 1)
                self.full(level + 1)
            else:
                self.gen.append(random.choice(self.func_set[1]))
                self.full(level + 1)

```

هر کدام از کروموزوم ها نمایی از genotype بوده که پیمایش preorder traversal از درخت آزمایشی میباشد! بعد از ساخت جمعیت اولیه بخش اصلی الگوریتم در کلاس Algorithm پیاده سازی شده است که مسئول آموزش نقاط(train)، ترکیب کروموزوم ها(crossover)، عمل جهش(Mutate) و ساخت نسل جدید بر اساس شایستگی می باشد.

ابتدا توسط تابع calculate_fitness شایستگی هر کدام از کروموزوم ها را تعیین کرده و عددی به آن نسبت می دهیم.

```
207     def calculate_fitness(self, inputs, outputs):
208         diff = 0
209         for i in range(len(inputs)):
210             try:
211                 diff += (self.eval(inputs[i])[0] - outputs[i][0])**2
212             except RuntimeWarning:
213                 self.gen = []
214                 if random.random() > 0.5:
215                     self.grow()
216                 else:
217                     self.full()
218                 self.calculate_fitness(inputs, outputs)
219 
220         if len(inputs) == 0:
221             return 1e9
222         self.fitness = diff/(len(inputs))
223         return self.fitness
```

- برای تعیین مقدار fitness همانگونه که در تصویر بالا مشخص است از الگوریتم Mean Squared Error (MSE) استفاده می کنیم که همان میانگین مربعات تفاضل مقدار تخمین زده شده از مقدار اصلی میباشد.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(Y_i - \hat{Y}_i)^2$$

با توجه به این تعریف روند الگوریتم باید بگونه ای باشد که به مرور زمان به مقدار fitness کمتری برای کروموزوم های تخمین زده شده برسیم.
روند آموزش نقاط به تعداد متغیر iterations انجام می شود که در ابتدای برنامه مقدار آن را مشخص میکنیم. در هر iterate با استفاده از تابع selection دو مورد از کروموزوم ها با بهترین شایستگی را به صورت تصادفی انتخاب میکنیم.

```
31 def selection(population, num_sel):
32 
33     sample = random.sample(population.list, num_sel)
34     best = sample[0]
35     for i in range(1, len(sample)):
36         if population.list[i].fitness < best.fitness:
37             best = population.list[i]
38 
39     return best
```

- برای دقت بیشتر الگوریتم در این مرحله بهتر است از تابع roulette_selection استفاده کنیم تا کروموزوم های با شایستگی بیشتر و متنوع تری به دست آورده و در نتیجه عملیات crossover موفق تری داشته باشیم!!

```python
def roulette_selecion(population):

    fitness = [chrom.fitness for chrom in population.list]
    order = [x for x in range(len(fitness))]
    order = sorted(order, key=lambda x: fitness[x])
    fs = [fitness[order[i]] for i in range(len(fitness))]
    sum_fs = sum(fs)
    max_fs = max(fs)
    min_fs = min(fs)
    p = random.random()*sum_fs
    t = max_fs + min_fs
    choosen = order[0]
    for i in range(len(fitness)):
        p -= (t - fitness[order[i]])
        if p < 0:
            choosen = order[i]
            break
    return population.list[choosen]
```

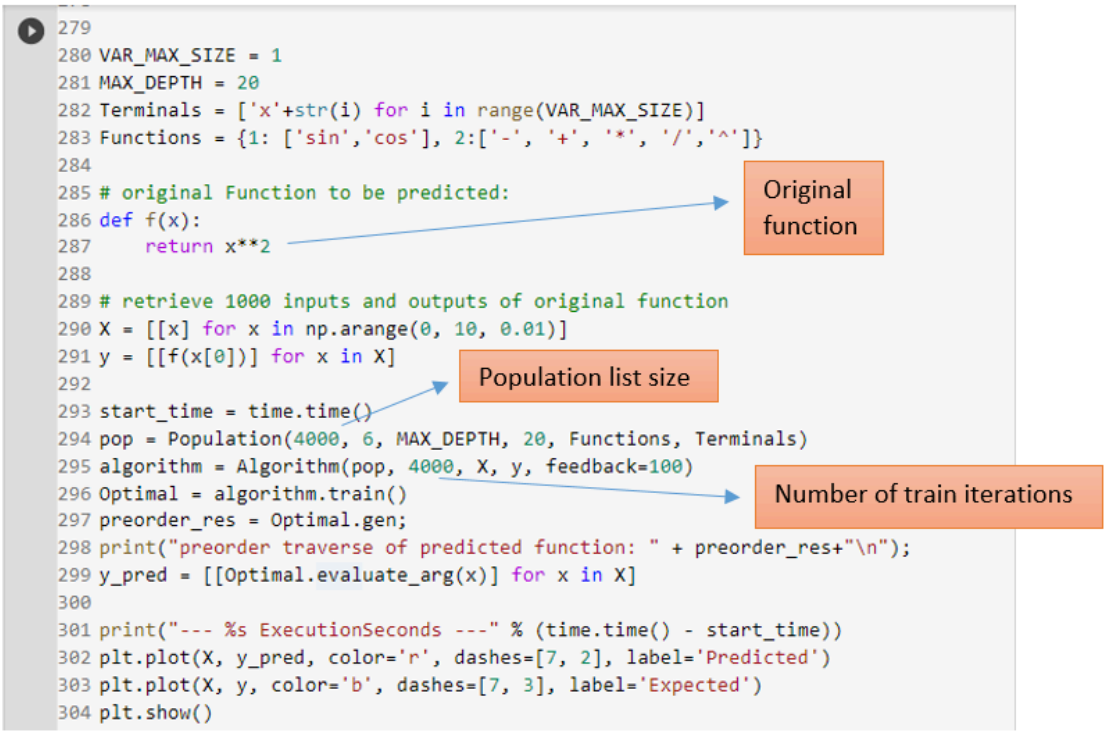
```python
def cross_over(mother, father, max_depth):
    #combine 2 chromosome into new childs
    child = Chromosome(mother.terminal_set, mother.func_set, mother.depth, None)
    start_m = np.random.randint(len(mother.gen))
    start_f = np.random.randint(len(father.gen))
    end_m = traversal(start_m, mother)
    end_f = traversal(start_f, father)
    child.gen = mother.gen[:start_m] + father.gen[start_f : end_f] + mother.gen[end_m :]
    if child.get_depth() > max_depth and random.random() > 0.2:
        child = Chromosome(mother.terminal_set, mother.func_set, mother.depth)
    return child
```

```python
def traversal(poz, chromosome):

    if chromosome.gen[poz] in chromosome.terminal_set:
        return poz + 1
    elif chromosome.gen[poz] in chromosome.func_set[1]:
        return traversal(poz + 1, chromosome)
    else:
        new_poz = traversal(poz + 1, chromosome)
        return traversal(new_poz, chromosome)
```

```python
def mutate(chromosome):

    poz = np.random.randint(len(chromosome.gen))
    if chromosome.gen[poz] in chromosome.func_set[1] + chromosome.func_set[2]:
        if chromosome.gen[poz] in chromosome.func_set[1]:
            chromosome.gen[poz] = random.choice(chromosome.func_set[1])
        else:
            chromosome.gen[poz] = random.choice(chromosome.func_set[2])
    else:
        chromosome.gen[poz] = random.choice(chromosome.terminal_set)
    return chromosome
```

کاملا به صورت تصادفی بخشی از کروموزوم های پدر و مادر را انتخاب کرده و آن را با یکدیگر جایگزین میکنیم تا نسل جدید ایجاد شود.
در تابع mutate یکی از ایندکس های کروموزوم را تغییر می دهیم.

- در گام آخر بعد از تعداد train هایی که اول برنامه مشخص کرده بودیم، باید بهترین کروموزوم به دست آمده (از نظر تابع شایستگی) را نمایش دهیم.
در نمودار نهایی خط قرمز تابع پیش بینی شده و خط آبی تابع اولیه می باشد.

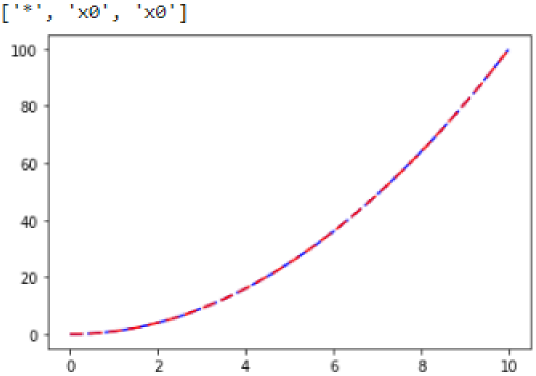


خروجی: به عنوان خروجی مقدار شایستگی هر کروموزوم به همراه پیمایش pre-order درخت آن آورده شده و روی نمودار نیز تابع پیش بینی شده با رنگ قرمز نمایش داده شده است. رنگ آبی تابع اولیه(اصلی) می باشد. در آخرین خط نیز زمان اجرای الگوریتم بر حسب ثانیه آورده شده.

در تصاویر زیر چند مثال مختلف بررسی شده و تابع اولیه هر تصویر کنار آن ذکر شده است:

F(x) = x^2

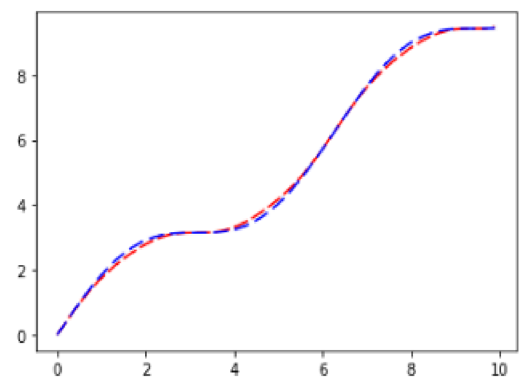
```
'x0']
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
Predicted function: ['*', 'x0', 'x0']
Fitness: 0.0
['*', 'x0', 'x0']
```




```
--- 313.2851884365082 seconds ---
```

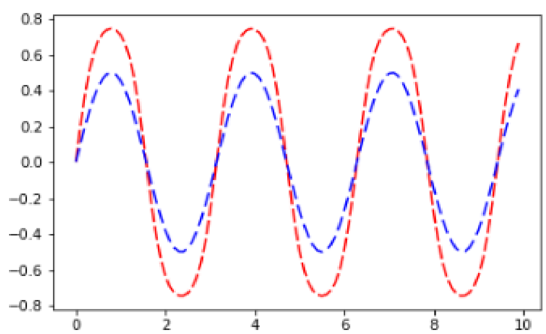
F(x) = Sin(x) + x

```
Predicted function: ['+', 'sin', 'sin', 'x0', 'x0']
Fitness: 0.007499217967913741
Predicted function: ['+', 'sin', 'sin', 'x0', 'x0']
Fitness: 0.007499217967913741
Predicted function: ['+', 'sin', 'sin', 'x0', 'x0']
Fitness: 0.007499217967913741
Predicted function: ['+', 'sin', 'sin', 'x0', 'x0']
Fitness: 0.007499217967913741
Predicted function: ['+', 'sin', 'sin', 'x0', 'x0']
Fitness: 0.007499217967913741
Predicted function: ['+', 'sin', 'sin', 'x0', 'x0']
Fitness: 0.007499217967913741
Predicted function: ['+', 'sin', 'sin', 'x0', 'x0']
Fitness: 0.007499217967913741
Predicted function: ['+', 'sin', 'sin', 'x0', 'x0']
Fitness: 0.007499217967913741
Predicted function: ['+', 'sin', 'sin', 'x0', 'x0']
Fitness: 0.007499217967913741
preorder traverse of predicted function:
['+', 'sin', 'sin', 'x0', 'x0']
--- 3.237107753753662 ExecutionSeconds ---
```

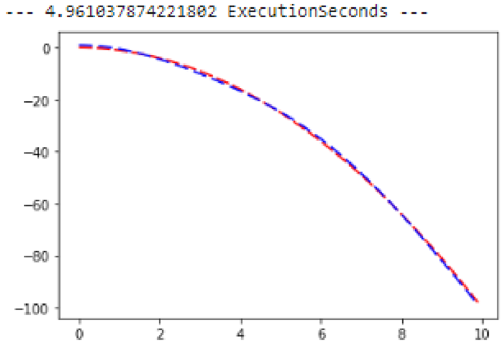


F(x) = Sin(x) * Cos(x)

```
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
Predicted function: ['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
Fitness: 0.046339145367830875
preorder traverse of predicted function:
['sin', 'sin', '*', 'sin', '+', 'x0', 'x0', '^', 'sin', 'x0', '-', 'x0', 'x0']
--- 13.922616004943848 ExecutionSeconds ---
```




```
Predicted function: ['-', 'sqrt', 'x0', '*', 'x0', 'x0']
Fitness: 5.945325446602774
Predicted function: ['-', 'sqrt', 'x0', '*', 'x0', 'x0']
Fitness: 5.945325446602774
Predicted function: ['-', 'sqrt', 'x0', '*', 'x0', 'x0']
Fitness: 5.945325446602774
Predicted function: ['-', 'x0', '+', 'x0', '*', 'x0', 'x0']
Fitness: 0.5242272965752303
Predicted function: ['-', 'x0', '+', 'x0', '*', 'x0', 'x0']
Fitness: 0.5242272965752303
preorder traverse of predicted function:
['-', 'x0', '+', 'x0', '*', 'x0', 'x0']
--- 4.961037874221802 ExecutionSeconds ---
```

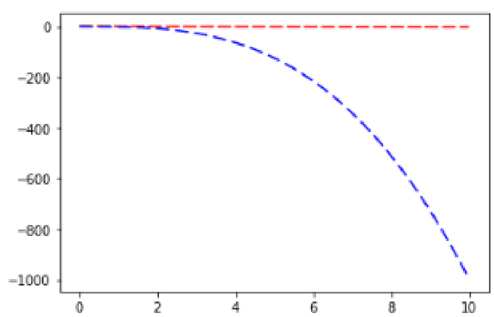


F(x) = Cos(x) – (x^2)

در مثال های قبل فقط عملگر های ابتدایی برای تخمین استفاده شده بود، در تصویر بعدی ln, tan, abs,sqrt نیز اضافه شده اند.

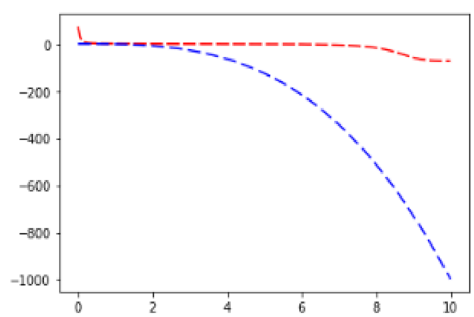
F(x) = - (x^3)

```
Predicted function: ['cos', 'sqrt', 'x0']
Fitness: 141907.48973153104
Predicted function: ['cos', 'sqrt', 'x0']
Fitness: 141907.48973153104
Predicted function: ['tg', 'cos', 'sqrt', 'x0']
Fitness: 141697.67145162972
Predicted function: ['tg', 'cos', 'sqrt', 'x0']
Fitness: 141697.67145162972
Predicted function: ['tg', 'cos', 'sqrt', 'x0']
Fitness: 141697.67145162972
Predicted function: ['tg', 'cos', 'sqrt', 'x0']
Fitness: 141697.67145162972
Predicted function: ['tg', 'cos', 'sqrt', 'x0']
Fitness: 141697.67145162972
Predicted function: ['tg', 'cos', 'sqrt', 'x0']
Fitness: 141697.67145162972
Predicted function: ['tg', 'cos', 'sqrt', 'x0']
Fitness: 141697.67145162972
Predicted function: ['tg', 'cos', 'sqrt', 'x0']
Fitness: 141697.67145162972
Predicted function: ['tg', 'cos', 'sqrt', 'x0']
Fitness: 141697.67145162972
Predicted function: ['tg', 'cos', 'sqrt', 'x0']
Fitness: 141697.67145162972
Predicted function: ['tg', 'cos', 'sqrt', 'x0']
Fitness: 141697.67145162972
preorder traverse of predicted function:
['tg', 'cos', 'sqrt', 'x0']
--- 81.19548177719116 ExecutionSeconds ---
```



F(x) = - (x^3)

```
Predicted function: [
Fitness: 141907.48973153104
Predicted function: ['cos', 'sqrt', 'x0']
Fitness: 141907.48973153104
Predicted function: ['cos', 'sqrt', 'x0']
Fitness: 141907.48973153104
<ipython-input-51-50aa4b6ce8db>:200: RuntimeWarning: invalid value encounter
  return np.tan(left), poz
Predicted function: ['tg', 'tg', 'cos', 'x0']
Fitness: 140245.50327858154
Predicted function: ['tg', 'tg', 'cos', 'x0']
Fitness: 140245.50327858154
Predicted function: ['tg', 'tg', 'cos', 'x0']
Fitness: 140245.50327858154
Predicted function: ['tg', 'tg', 'cos', 'sqrt', 'x0']
Fitness: 131750.69767556913
<ipython-input-51-50aa4b6ce8db>:190: RuntimeWarning: invalid value encounter
  return np.sin(left), poz
Predicted function: ['tg', 'tg', 'cos', 'sqrt', 'x0']
Fitness: 131750.69767556913
Predicted function: ['tg', 'tg', 'cos', 'sqrt', 'x0']
Fitness: 131750.69767556913
Predicted function: ['tg', 'tg', 'sin', 'tg', 'cos', 'sqrt', 'x0']
Fitness: 125580.87767924518
Predicted function: ['tg', 'tg', 'sin', 'tg', 'cos', 'sqrt', 'x0']
Fitness: 125580.87767924518
Predicted function: ['tg', 'tg', 'sin', 'tg', 'cos', 'sqrt', 'x0']
Fitness: 125580.87767924518
Predicted function: ['tg', 'tg', 'sin', 'tg', 'cos', 'sqrt', 'x0']
Fitness: 125580.87767924518
Predicted function: ['tg', 'tg', 'sin', 'tg', 'cos', 'sqrt', 'x0']
Fitness: 125580.87767924518
Predicted function: ['tg', 'tg', 'sin', 'tg', 'cos', 'sqrt', 'x0']
Fitness: 125580.87767924518
preorder traverse of predicted function:
['tg', 'tg', 'sin', 'tg', 'cos', 'sqrt', 'x0']
--- 297.0269994735718 ExecutionSeconds ---
```



نکته قابل توجه در اشکال خروجی اینست که توابع پیش بینی شده کاملا وابسته به تعداد مجموعه ی جمعیت اولیه و تعداد دفعات train می باشد و در صورت افزایش این مقادیر توابع رفته رفته به واقعیت نزدیک تر میشوند و شایستگی بهتری پیدا می کنند.

به عنوان مثال در شکل سمت راست تعداد train های نقاط اولیه 2 برابر شده و اعضای جمعیت اولیه نیز بیشتر شدند تا در نتیجه به جواب بهتری با fitness کمتر برسیم.

- جمع بندی:
  بنابراین به کمک الگوریتم ژنتیک در کنار عملیاتی مانند (MSE, roulette selection,...) توانستیم با داشتن اطلاعاتی از تابع اولیه، تابعی نظیر آن را تخمین بزنیم و ارزش گذاری کنیم.