گزارش تمرین دوم

(پردازش تکاملی)

تهیه شده توسط:

سارا لیمویی

استاد:

دکتر حمزه

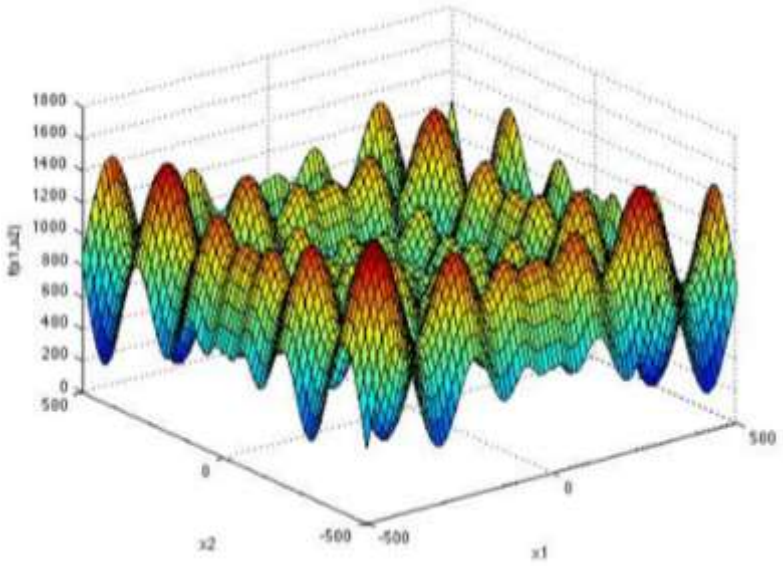
پاییز ۱۴۰۰

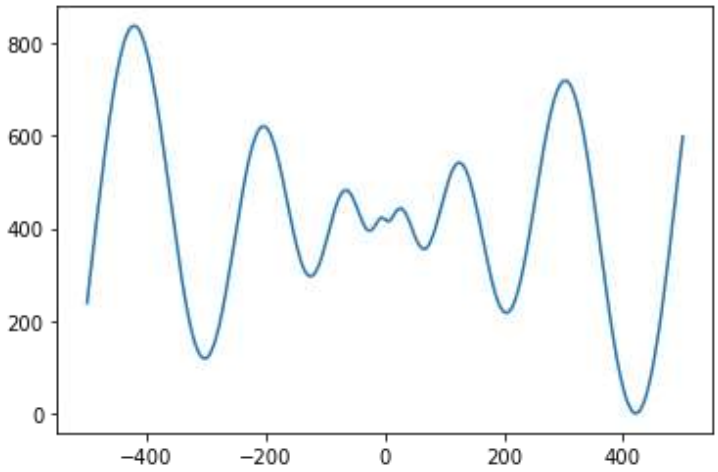

بسم الله الرحمن الرحیم

فهرست مطالب

مقدمه

در این تمرین می خواهیم global minima ی تابع Schwefel را پیدا کنیم. شکل تابع در ۳ بعد و در ۲ بعد به ترتیب به صورت زیر می باشد:





تابع فوق به صورت زیر تعریف می شود:

$$f(x1 \ \dots \ xn) \ = \ \Sigma(-x_i \ sin(\sqrt{|x_i|})) \ + \ \alpha.n$$

$$xi \in [-500, 500], i = 1, \dots, n$$

$$\alpha = 418.9829$$

همچنین می دانیم که:

$$f\ (x^*) = 0, at\ x^* = (420.9687, \dots , 420.9687)$$

در واقع می خواهیم به کمک الگوریتم تکاملی، این نقطه ی global minima را پیدا کنیم.

توضیحات

چون تابعی که می خواهیم minimize کنیم یک تابع پیوسته می باشد یعنی فضایی که در آن کار میکنیم، پیوسته می باشد، از الگوریتم evolutionary strategy یا ES استفاده می کنیم. این الگوریتم به صورت کلی بر اساس mutation کار میکند. در این تمرین، method های مختلف موجود برای هر کدام از عملگر های الگوریتم از جمله mutation، cross over یا recombination و survival selection پیاده سازی شده و تست می شود. در ادامه به توضیح هر کدام از این method ها می پردازیم.

اکنون، ابتدا ساختار کلی الگوریتم را توضیح می دهیم. الگوریتم ES مشابه هر الگوریتم تکاملی دیگر، دارای تعدادی عملگر می باشد که ساختار کلی آن در ادامه توضیح داده شده است.

کلاس SchwefelFunctionES تابع objective function ای که باید minimize شود به همراه آدرس یک فایل configuration را به عنوان ورودی دریافت می کند و یک object از کلاس مذکور می سازیم. با صدا زدن تابع es_main_loop الگوریتم ES برای پیدا کردن global minimum تابع objective اجرا می شود.

```
def es_main_loop(self):
    self.print_es_configuration()
    self.initialize_population()
    for epoch in range(self.max_iter):
        self.parent_selection()

        self.recombination()
        if self.is_mutation_enabled == True:
            self.parent_selection()
            self.mutation()
        self.survival_selection()

        # print(len(self.population))
        if epoch%500 == 0:
            print(f'epoch: {epoch} | best chrom: {np.round(self.best_chromosome, 2)} | \
                    best fitness: {np.round(self.best_fitness, 2)}')

    return self.best_chromosome, self.best_fitness
```

Configuration file

در صورتی که به جای آدرس فایل configuration، مقدار None را پاس بدهیم، به صورت پیش فرض با configuration زیر الگوریتم را اجرا می کند.

- ES-type: (μ + λ) # only children will transfer to next generation
- N: 3 # objective function dimension
- x-range: (-500, +500) # range of chromosome values
- μ: 20 # number of parents selected
- λ: 100 # population size
- mutation: enabled
- mutation-type: type-3 # type-3
- recombination-type: type-1 # type-1
- mutation rate: 80%
- recombination rate: 100%
- scale_pop: 7

برای خوندن از فایل به صورت زیر عمل می کنیم:

```
# read from configuration file
else:
    settings = []
    with open(config_file_path,'r') as f:
        for line in f:
            settings.append(line.split()[1])

    self.es_type = settings[0]
    self.max_iter = int(settings[1])
    self.dimension = int(settings[2])
    self.mu = int(settings[3])
    self.landa = int(settings[4])
    self.is_mutation_enabled = True if settings[5]=="True" else False
    self.mutation_type = settings[6]
    self.sigma_min = 0
    self.sigma_max = 1

    if self.mutation_type == 'type-1':
        self.mutation_step_size = np.random.uniform(low=self.sigma_min, high=self.sigma_max, size=(self.landa,1))

    if self.mutation_type == 'type-2':
        self.mutation_step_size = np.random.uniform(low=self.sigma_min, high=self.sigma_max, size=(self.landa,self.dimen

    if self.mutation_type == 'type-3':
        self.mutation_step_size = np.random.uniform(low=self.sigma_min, high=self.sigma_max, size=(self.landa,self.dimen
        self.k = int(self.dimension*(self.dimension-1)/2)
        self.min_alpha = math.radians(-180)
        self.max_alpha = math.radians(180)
        self.mutation_alpeha = np.random.uniform(low=self.min_alpha, high=self.max_alpha, size=(self.landa,self.k))

    self.recombination_type = settings[7]
    self.xi_min = int(settings[8])
    self.xi_max = int(settings[9])
    # create same bounds for all dimensions
    self.boundaries = np.array([[self.xi_min, self.xi_max] for i in range(self.dimension)])

    self.recombination_rate = int(settings[10])
    self.mutation_rate = int(settings[11])
    self.scale_pop = int(settings[12])
```

اکنون به توضیح هر عملگر configuration ها می پردازیم.

به صورت کلی دو نوع الگوریتم ES داریم که بسته به عملیاتی که در survival selection انجام می شود دسته بندی می شوند.

۱. (μ, λ): یا generational که در آن نسل بعد فقط فرزندان جایگزین می شوند.
۲. $(\mu + \lambda)$: یا non-generational که در آن نسل بعد از بین فرزندان و والدین انتخاب می شود.

N یا بعد تابع objective می باشد که در فرمولی که در ابتدا برای Schwefel objective function تعریف شده، جایگزین می شود.

μ تعداد والدینی است که برای تولید فرزند جدید انتخاب می شوند.

λ تعداد کل جمعیت می باشد.

تعداد فرزندان جدیدی که در هر نسل تشکیل می شود، برابر است با scale_pop*λ. به عبارت دیگر، در هر iteration، به تعداد حدودا ۷ یا ۸ برابر جمعیت، فرزند تولید می شود که از این تعداد، بسته به روش survival selection، به تعداد landa یعنی جمعیت که در این تمرین ۱۰۰ در نظر گرفته شده است، انتخاب می شود برای آنکه به نسل بعد منتقل شود.

Parent selection نیز به صورت رندوم صورت می گیرد.

Mutation به روش های متفاوتی می تواند پیاده سازی شود.

۱) Type-1: در این تایپ از mutation، علاوه بر پارامتر های $x_1, x_2, \ldots, x_n$، پارامتر σ را داریم . در هر نسل برای اعمال عملگر mutation به ترتیب به صورت زیر عمل می کنیم:

$$\sigma' = \sigma \cdot e^{\tau \cdot N(0,1)}$$

$$\tau = \frac{1}{n^{\frac{1}{2}}}$$

$$x' = x + N(0, \sigma')$$

همچنین درصورتی که $\sigma' < \varepsilon_0$ شود، $\sigma' = \varepsilon_0$ می شود که در این پروژه مقدار ε_0 برابر 0.25 در نظر گرفته شده است.

```
if self.mutation_type == 'type-1':
    # in this type, sigma is a number
    taw = 1/np.sqrt(self.dimension)                    # taw is the learning rate in this method
    new_sigma = sigma * np.exp(taw * random.gauss(0, 1))
    if (new_sigma < self.epsilon):
        new_sigma = self.epsilon
    rand_num = random.gauss(0, 1)
    for d_indx in range(self.dimension):
        chrom[d_indx] += new_sigma * rand_num
```

۲) Type-2: در این تایپ از mutation، علاوه بر پارامتر های $x_1, x_2, ..., x_n$ ،n پارامتر σ_i را داریم .
در هر نسل برای اعمال عملگر mutation به ترتیب به صورت زیر عمل می کنیم:

$\sigma' = \sigma . e^{\tau'.N(0,1) + \tau.N(0,1)}$

$\tau = \frac{1}{(2n)^{\frac{1}{2}}}$ $\tau' = \frac{1}{(2n^{\frac{1}{2}})^{\frac{1}{2}}}$

$x' = x + \sigma'. N(0, 1)$

همچنین درصورتی که $\sigma' < \varepsilon_0$ شود، $\sigma' = \varepsilon_0$ می شود که در این پروژه مقدار ε_0 برابر 0.25 در نظر گرفته شده است.

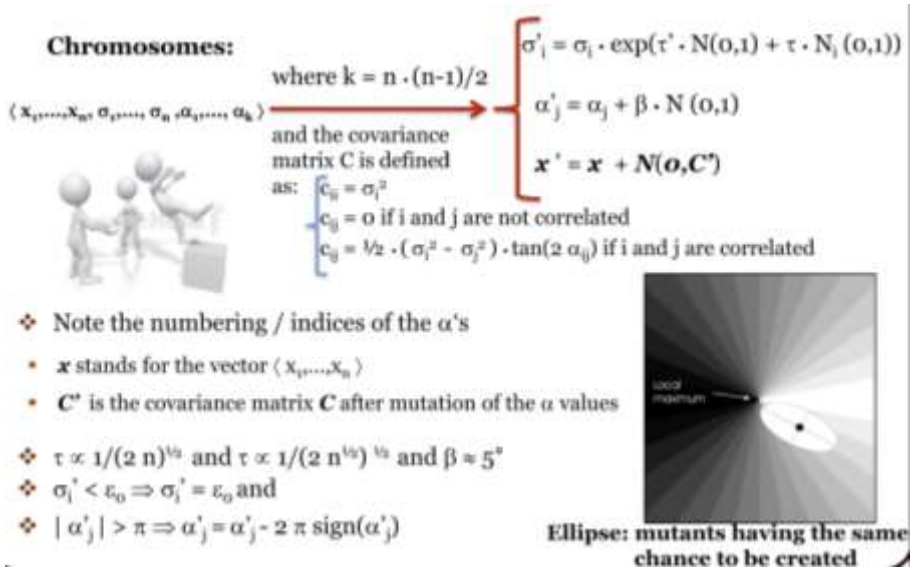
```
if self.mutation_type == 'type-2':
    # in this type, sigma is an array
    taw = 1/np.sqrt(2*self.dimension)                    # taw is the learning rate in this method
    taw_prim = 1/np.sqrt((2*np.sqrt(self.dimension)))    # taw_primn is the learning rate in this method
    # here mutation_step_size is a d-dimensional array
    new_sigma = sigma
    rand_num = random.gauss(0, 1)
    for i in range(len(sigma)):
        new_sigma[i] = sigma[i] * np.exp(taw_prim*rand_num + taw*random.gauss(0, 1))
    for i in range(len(sigma)):
        if new_sigma[i] < self.epsilon:
            new_sigma[i] = self.epsilon
    for d_indx in range(self.dimension):
        chrom[d_indx] += new_sigma[d_indx] * random.gauss(0, 1)
```

۳) Type-3:

در این تایپ از mutation، علاوه بر پارامتر های $x_1, x_2, ..., x_n$ و n پارامتر σ_i، k پارامتر alpha داریم که در آن k به صورت زیر تعریف می شود:

$K = n*(n-1)/2$

برای اعمال mutation بر هر کروموزوم، به صورت زیر عمل می کنیم:



همان طور که در شکل بالا نیز پیداست، ابتدا مقادیر sigma ها و سپس alpha ها و نهایتا x_i های کروموزوم مورد نظر آپدیت می شوند. برای آپدیت کردن مقادیر آلفا ها، ابتدا تعداد k تا alpha به ازای هر کروموزوم جمعیت به صورت رندوم می سازیم. لازم به ذکر است که alpha باید بین $(-\pi, \pi)$ باشد. جهت سهولت در محاسبات، همه ی اعداد به رادیان تبدیل شده اند و beta نیز که باید ۵ درجه باید، به رادیان تبدیل می شود.

سپس باید ماتریس C را با مقادیر جدید alpha بسازیم. برای اینکه ۳ ماتریس مربعی با ابعاد (k, k) می سازیم.

ماتریس اولی، یک ماتریس بالا مثلثی است که مقادیر آن به صورت $1/2*(\sigma_i-\sigma_j)*\tan(2*\alpha_{ij})$ تعیین می شوند. به همین ترتیب یک ماتریس پایین مثلثی نیز تشکیل می دهیم. سپس برای عناصر روی قطر اصلی، ماتریس قطری با مقادیر $½*\sigma_j\text{^}2$ می سازیم. سپس حاصل جمع این ۳ ماتریس برابر با ماتریس C خواهد بود. سپس برای آپدیت کردن مقدار x_i ها از این ماتریس C به عنوان ماتریس covariance تابع multivariate_normal با میانگین صفر استفاده می کنی. قطعه کد این بخش در ادامه آورده شده است:

```
if self.mutation_type == 'type-3':
    beta = math.radians(5)      # beta = 5 degree -> but since we are working with radians, lets convert it
    # update sigma
    taw = 1/np.sqrt(2*self.dimension)                       # taw is the learning rate in this method
    taw_prim = 1/np.sqrt((2*np.sqrt(self.dimension)))       # taw_primn is the learning rate in this method
    # here mutation_step_size is a d-dimensional array
    new_sigma = sigma
    rand_num = random.gauss(0, 1)
    for i in range(len(sigma)):
        new_sigma[i] = sigma[i] * np.exp(taw_prim*rand_num + taw*random.gauss(0, 1))
    for i in range(len(sigma)):
        if new_sigma[i] < self.epsilon:
            new_sigma[i] = self.epsilon
    # update alpha
    new_alpha = alpha
    rand_num = random.gauss(0, 1)
    for i in range(len(sigma)):
        new_alpha[i] = alpha[i] + beta*rand_num
    for i in range(len(alpha)):
        if new_alpha[i] > math.radians(180):
            new_alpha[i] = new_alpha[i] - 2*math.radians(180)*np.sign(new_alpha[i])
    # create c-matrix
    chrom_C_matrix = self.create_C_matrix(self.k, new_sigma, new_alpha)
    # update chromosome
    mean = np.zeros(chrom_C_matrix.shape[0])
    chrom = chrom + np.random.multivariate_normal(mean, chrom_C_matrix)
```

چون این ماتریس C که در تابع multivariate_normal صدا زده می شود باید positive-semidefinite باشد، به کمک تابع get_near_psd، در صورتی که این ویژگی را نداشته باشد، آن را positive-semidefinite، می کنیم.

Recombination یا cross-over نیز به روش های متفاوتی می تواند پیاده سازی شود:

	Two fixed parents	Two parents selected for each i
$z_i = (x_i + y_i)/2$	Local intermediary	Global intermediary
z_i is x_i or y_i chosen randomly	Local discrete	Global discrete

در هر مرحله دو parent به صورت رندوم انتخاب می شود و به دو صورت میانگین هر دو parent و یا انتخاب یکی از ژن ها به صورت رندوم از هر یک از والدین برای هر ژن فرزند جدید، آن را تولید می کنیم.

- **type-1:** Local intermidiary
- **type-2:** global intermidiary
- **type-3:** Local discrete
- **type-4:** global discrete

برای survival selection، یا به تعداد جعیت بعد از sort کردن فرزندان تولید شده به صورت صعودی، مثلا ۱۰۰ تای اول را به نسل بعد می بریم و یا اینکه علاوه بر همه ی ۷۰۰ فرزند تولید شده، ۱۰۰ تا جمعیت موجود را نیز سورت کرده و در مجموعه بهترین ۱۰۰ تا را انتخاب می کنیم. تعداد فرزندان به پارامتر scale_pop بستگی دارد.

```
def survival_selection(self):

    # only select childs or childs+parents
    if self.es_type == '(mu,landa)':
        # select top landa of offsprings
        t = [(chrom, self.objective_fn(chrom)) for chrom in self.children]
        t.sort(key=lambda x: x[1])
        t = t[:self.landa]
        self.population = []
        for c in t:
            self.population.append(c[0])
        self.population = np.array(self.population)

    if self.es_type == '(mu+landa)':
        t = []
        for c in self.children:
            t.append(c)
        for c in self.population:
            t.append(c)
        t1 = [(chrom, self.objective_fn(chrom)) for chrom in t]
        t1.sort(key=lambda x: x[1])
        t1 = t1[:self.landa]
        self.population = []
        for c in t1:
            self.population.append(c[0])
        self.population = np.array(self.population)

    return self.population
```

نتایج

در این بخش نتایج حاصل از اجرای الگوریتم را بررسی می کنیم. الگوریتم را با تنظیمات مختلفی که در بخش های قبل تر توضیح داده شده است، می توان تست کرد و نتیجه را مشاهده کرد که در ادامه بخشی از نتایج آورده شده است:

```
*******************************
******   ES-(mu,landa)   ******
*******************************
Schwefel dimension  : 2
x_min , x_max       : -500, 500
number of iterations: 25
population size     : 100
mutation             : enabled
mutation rate        : 60
mutation type       : type-1
recombination type  : type-3
recombination rate  : 100
*******************************
*******************************
```

```
epoch: 0 | best chrom: [ 423.17 -309.76] |          best fitness: 125.67927
epoch: 5 | best chrom: [420.35 421.21] |          best fitness: 0.05588
epoch: 10 | best chrom: [420.96 420.98] |          best fitness: 5e-05
epoch: 15 | best chrom: [420.97 420.97] |          best fitness: 3e-05
epoch: 20 | best chrom: [420.97 420.97] |          best fitness: 3e-05
epoch: 25 | best chrom: [420.97 420.97] |          best fitness: 3e-05
```

(شکل ۱)

توانست global minima را پیدا کند.

```
*******************************
******   ES-(mu+landa)   ******
*******************************
Schwefel dimension  : 2
x_min , x_max       : -500, 500
number of iterations: 25
population size     : 100
mutation             : enabled
mutation rate        : 60
mutation type       : type-1
recombination type  : type-3
recombination rate  : 100
*******************************
*******************************
```

```
epoch: 0 | best chrom: [430.05 430.74] |    best fitness: 22.43147
epoch: 5 | best chrom: [420.58 420.86] |    best fitness: 0.02073
epoch: 10 | best chrom: [420.97 420.96] |    best fitness: 3e-05
epoch: 15 | best chrom: [420.97 420.97] |    best fitness: 3e-05
epoch: 20 | best chrom: [420.97 420.97] |    best fitness: 3e-05
epoch: 25 | best chrom: [420.97 420.97] |    best fitness: 3e-05
```

توانست global minima را پیدا کند.

(شکل ۲)

```
*******************************
******   ES-(mu,landa)   ******
*******************************
Schwefel dimension  : 2
x_min , x_max       : -500, 500
number of iterations: 25
population size     : 100
mutation             : enabled
mutation rate        : 60
mutation type       : type-2
recombination type  : type-2
recombination rate  : 100
*******************************
*******************************
```

```
epoch: 0 | best chrom: [ 210.38 -288.53] |    best fitness: 365.34727
epoch: 5 | best chrom: [ 429.71 -305.83] |    best fitness: 129.45713
epoch: 10 | best chrom: [-302.53 -302.52] |    best fitness: 236.87669
epoch: 15 | best chrom: [-302.52 -302.52] |    best fitness: 236.87669
epoch: 20 | best chrom: [-302.52 -302.52] |    best fitness: 236.87669
epoch: 25 | best chrom: [-302.52 -302.52] |    best fitness: 236.87669
```

در local minima متوقف می شود و نمی تواند global minima را پیدا کند.

(شکل ۳)

```
*******************************
******   ES-(mu,landa)   ******
*******************************
Schwefel dimension  : 2
x_min , x_max       : -500, 500
number of iterations: 25
population size     : 100
mutation             : disabled
recombination type  : type-1
recombination rate  : 100
*******************************
*******************************
```

```
epoch: 0 | best chrom: [418.98 -20.45] |                    best fitness: 399.39582
epoch: 5 | best chrom: [ 404.72 -122.99] |                   best fitness: 329.20732
epoch: 10 | best chrom: [ 203.81 -302.53] |                  best fitness: 335.57803
epoch: 15 | best chrom: [ 203.81 -302.52] |                  best fitness: 335.57803
epoch: 20 | best chrom: [ 203.81 -302.52] |                  best fitness: 335.57803
epoch: 25 | best chrom: [ 203.81 -302.52] |                  best fitness: 335.57803
```

(شکل ۴)

باتنظیمات فوق و با disable کردن mutation، به جواب خوبی نرسیدیم.

```
*******************************
******   ES-(mu,landa)   ******
*******************************
Schwefel dimension  : 3
x_min , x_max       : -500, 500
number of iterations: 25
population size     : 100
mutation             : enabled
mutation rate        : 60
mutation type       : type-3
recombination type  : type-4
recombination rate  : 100
*******************************
*******************************
```

```
epoch: 0 | best chrom: [220.15 441.56 436.61] |          best fitness: 334.11258
epoch: 5 | best chrom: [416.23 430.65 411.65] |          best fitness: 25.52433
epoch: 10 | best chrom: [419.16 426.1  418.96] |         best fitness: 4.24545
epoch: 15 | best chrom: [420.84 421.07 421.05] |         best fitness: 0.00439
epoch: 20 | best chrom: [420.97 420.96 420.99] |         best fitness: 0.00012
epoch: 25 | best chrom: [420.97 420.96 420.97] |         best fitness: 4e-05
```

(شکل ۵)

تنظیمات mutation-type-3