Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования Московский государственный технический университет имени Н.Э. Баумана

Лабораторная работа №7. Вариант 1. «Методы решения задач многокритериальной и многоэкстремальной оптимизации» по курсу «Методы оптимизации»

Студент группы ИУ9-82 Белогуров А.А.

Преподаватель Каганов Ю.Т.

Москва, 2018

Содержание

1 Цель работы

1. Изучение методов решения задач многоэкстремальной и многокритериальной оптимизации.

2. Разработка программ реализации алгоритмов мультистарта (метода ближайшего соседа, метода конкурирующих точек), генетического алгоритма, алгоритмом многокритериальной оптимизации.

3. Решение задачи:

   (a) многоэкстремальной оптимизации для функции Шекеля;

   (b) многоэкстремальной оптимизации для заданных многоэкстремальных функций;

   (c) многокритериальной оптимизации.

2 Постановка задачи

2.1 Задача 7.1

Дано: 1 Вариант. Функция Шекеля:

$$f(x) = -\sum_{i=1}^{3} \frac{a}{f_i^0 + b\sum_{j=1}^{3}(x_j - x_{ij}^0)^2},$$
$$\sum_{i=1}^{3} \frac{1}{f_i^0} = f^0 = -c, x_{ij}^0 \in [\alpha, \beta]^3, \quad (1)$$

где $a = 2$, $b = 3$, $c = 1$, $\alpha = 0$, $\beta = 5$.

1. Реализовать алгоритмы многоэкстремальной оптимизации методом мультистарта на одном из языков высокого уровня.
2. Решить задачу многоэкстремальной оптимизации с помощью метода мультистарта.

1. Реализовать генетический алгоритм многоэкстремальной оптимизации на одном из языков высокого уровня.
2. Решить задачу многоэкстремальной оптимизации с помощью генетического алгоритма.

2.2 Задача 7.2

Дано: 1 Вариант.

$$\begin{cases} f_1(x) = x_1^2 + x_2^2 \to min, \\ f_2(x) = 4(x_1 - 5)^2 + 2(x_2 - 6)^4 \to min, \\ g_1(x) = x_2 - x_1 + 1 \leq 0, \\ g_2(x) = x_1 + x_2 - 2 \leq 0. \end{cases} \quad (2)$$

1. Реализовать метод свертки критериев (метод «идеальной точки») многокритериальной оптимизации на одном из языков высокого уровня.

2. Требуется найти множества сильно эффективных решений (оптимальных по Парето) и слабо эффективных решений (оптимальных по Слейтеру) для (2).

3. Поиск четырех эффективных точек определяется путем задания бинарных отношений между критериями и получением вектора весовых коэффициентов методом Т. Саати для (2).

4. Исследовать полученное представление решений в пространстве критериев с учетом заданных ограничений для (2).

5. Реализовать алгоритмы программированием на C++ и Python

3 Исследование

Найдем экстремум для функции Шекеля:

$$f(x) = -\sum_{i=1}^{3} \frac{a}{f_i^0 + b\sum_{j=1}^{3}(x_j - x_{ij}^0)^2},$$

$$\sum_{i=1}^{3} \frac{1}{f_i^0} = f^0 = -c, x_{ij}^0 \in [\alpha, \beta]^3, \quad (3)$$

где $a = 2, b = 3, c = 1, \alpha = 0, \beta = 5$, с помощью сервиса WolframAlpha.com:

$$min(f(x)) = 18, \quad (x_1, x_2, x_3) = (4, 4, 4) \quad (4)$$

3.1 Задача 7.1

3.1.1 Алгоритм метода ближайшего соседа

1. Генерируются K точек $(z_1, z_2, ..., z_k)$ случайным образом с помощью, например, алгоритма Монте-Карло. Считается, что все они принадлежат различным кластерам (кластер ассоциируется с окрестностью локального минимума).

2. Находится ближайшая пара точек (z_i, z_j), т.е. такая пара, для которой $\rho_{ij} = argmin_{k,l=1,...,K}\rho_{kl}$.

3. Если расстояние $\rho_{ij} = \rho(z_i, z_j)$ между ближайшими соседями не превосходит некоторое достаточно малое число $\sigma > 0$, то точки z_i и z_j, а также соответствующие им кластеры объединяются и число кластеров K уменьшается на единицу. После этого происходит переход на Ш.2.

4. Остановка алгоритма происходит, либо если расстояние между ближайшеми соседями превосходит σ, либо если остается лишь один кластер.

В качестве метрики $\rho(z_i, z_j)$ обычно выбирается евклидова метрика. Качество алгоритма как процедуры кластеризации существенно зависит от того, насколько удачно выбрано число σ. Это число должно быть достаточно малым, во всяком случае меньше расстояний между соседними точками локальных минимумов.

3.1.2 Алгоритм метода конкурирующих точек

1. Моделируется равномерное на X распределение. В результате получается N точек $x_1, x_2, ..., x_N$.

2. Используя точки $x_1, x_2, ..., x_N$ в качестве начальных, проводится одна или несколько итераций какого либо алгоритма локальной оптимизации. Получаем точки $(z_1, z_2, ..., z_N)$.

3. Применяется какой либо метод кластеризации к точкам $(z_1, z_2, ..., z_N)$. Пусть m - число получившихся кластеров. Если $m = 1$, то переход на Ш.5. Иначе переход на Ш.4.

4. Выбираются представители $x_1, x_2, ..., x_m$ от всех кластеров (естественно выбирать в кластерах с наименьшим значением целевой функции). Положим $N = m$ и переход на Ш.2.

5. Считаем, что находимся в окрестности точки глобального минимума. Выбрать представителя от единственного кластера. Используя его в качестве начальной точки для алгоритма локальной оптимизации, обладающего высокой скоростью сходимости в окрестности точки экстремума.

3.1.3 Генетический алгоритм

Генетический алгортим - это эвристический алгоритм поиска, используемый для решения задач оптимизации и моделирования путём

случайного подбора, комбинирования и вариации искомых параметров с использованием механизмов, аналогичных естественному отбору в природе.

В нем используется основных 5 принципов:

1. **Формирование исходной популяции** - в зависимости от поставленной задачи генерируются N особей $x^k = (x_1^k, ..., x_n^k)^T$, для которой вычисляется функция фитнеса;

2. **Отбор (селекция)** - операция, которая осуществляет отбор особей (хромосом) x^k в соответствии со значениями функции фитнеса $\mu(x^k)$ для последующего их скрещивания.

3. **Кроссинговер (скрещивание)** – это операция, при которой из нескольких, обычно двух хромосом (особей) $(x_i, x_j), \quad i \neq j$, называемых родителями, порождается одна или несколько новых, называемых потомками.

4. **Мутация** - это преобразование хромосомы, случайно изменяющее один или несколько из её генов. Оператор мутации предназначен для того, чтобы поддерживать разнообразие особей в популяции. Необходимо определить параметр $Pm \in (0, 1]$ - вероятность мутации.

5. **Формирование новой популяции**

   (a) С равной вероятностью из потомков мутантов предыдущего шага выбирается один $x^m = (x_1, x_2, ..., x_p^M, ..., x_n)$.

   (b) Выбранный потомок добавляется в популяцию вместо хромосомы, которой соответствует наименьшее значение функции фитнеса (наихудшее из допустимых значений).

   (c) Вычисляется значение функции фитнеса для мутантного потомка $\mu_M = \mu(x^M)$.

6. **Проверка условия останова генетического алгоритма** - условием окончания работы генетического алгоритма является формирование заданного количества популяций $t = Np$.

3.2 Задача 7.2

3.2.1 Методы свертки критериев (метод «идеальной точки»)

1. Решение оптимизационных задач для каждого из критериев отдельно и получение «идеальной точки» с учетом ограничений.

2. Выбор весовых коэффициентов. Величины весовых коэффициентов - подбираются исходя из предварительно обусловленных соображений, которые могут быть выражены в виде дополнительных неформализуемых критериев. Чаще всего весовые коэффициенты выбираются путём бинарных сравнений и построения обратно симметричной матрицы. Далее вычисляются собственные значения и, для максимального собственного значения, выбирается соответствующий собственный вектор, элементы которого используются как весовые коэффициенты (метод Т. Саати). В данном случае весовые коэффициенты могут варьироваться различным образом для исследования пространства эффективных точек.

3. Получение оптимальных по Парето и Слейтеру решений. Для решения задачи можно использовать методы штрафных функций или метод модифицированных функций Лагранжа.

4 Практическая реализация

Все методы были реализованы на языке программирования **Python**. Для алгоритмов с кластеризацией использовались следующие классы:

Листинг 1. Вспомогательные классы для кластеризации.

```
class FunValue:
        def __init__(self, x_values, fun_value):
                self.x_values = x_values
                self.fun_value = fun_value

        def __str__(self):
                return "{}, {}".format(self.x_values, self.fun_value)


class Cluster:
        def __init__(self, fun_values=None):
                if fun_values is None:
                        fun_values = []
                self.funValues = [fun_values]

        def euclidean_distance(self, another_cluster):
                min_distance = float("INF")
                for i in self.funValues:
                        distance = 0
                        for j in another_cluster.funValues:
                                distance += pow(i.fun_value - j.fun_value, 2)

                        if distance < min_distance:
                                min_distance = distance

                return math.sqrt(min_distance)

        def merge(self, another_cluster):
                self.funValues += another_cluster.funValues

        def min_by_value(self):
                min_fun_value = None

                for item in self.funValues:
                        if min_fun_value is None:
                                min_fun_value = item
                        elif item.fun_value < min_fun_value.fun_value:
                                min_fun_value = item
```

```
39
40                  assert min_fun_value is not None
41                  return min_fun_value
42
43          def __str__(self):
44                  return "{}".format(self.funValues)
```

Листинг 2. Метод ближайшего соседа.

```
1   def nearest_neighbor(clusters, epsilon):
2           while True:
3                   min_distance = float("INF")
4                   min_i = -1
5                   min_j = -1
6
7                   for i in range(len(clusters)):
8                           for j in range(len(clusters)):
9                                   if i != j and
                                      ↪   clusters[i].euclidean_distance(clusters[j])
                                      ↪   < min_distance:
10                                          min_distance =
                                              ↪   clusters[i].euclidean_distance(clusters[j])
11                                          min_i = i
12                                          min_j = j
13
14                  assert min_i > -1
15                  assert min_j > -1
16
17                  if min_distance < epsilon:
18                          clusters[min_i].merge(clusters[min_j])
19                          clusters.remove(clusters[min_j])
20                  else:
21                          break
22
23                  if len(clusters) <= 1:
24                          break
25
26          # for cluster in clusters:
27          #         print()
28          #         for funValue in cluster.funValues:
29          #                 print(funValue)
30
31          return clusters
```

Листинг 3. Метод конкурирующих точек

```
def competing_points(cluster_count, epsilon):
        not_changed_in_inter = False
        step = (beta - alpha) / (cluster_count - 1)
        clusters = [generate_cluster([alpha + i * step for _ in range(3)])
          ↪   for i in range(cluster_count)]

        while True:
                for cluster in clusters:
                        for item in cluster.funValues:
                                nelder_mead = nelderMead(item.x_values)

                                item.fun_value = nelder_mead.fun
                                item.x_values = nelder_mead.x

                clusters = nearest_neighbor(clusters, epsilon)

                print(len(clusters))
                if len(clusters) != 1:
                        if not_changed_in_inter:
                                not_changed_in_inter = False
                                epsilon *= 10
                        else:
                                not_changed_in_inter = True

                        min_from_clusters = []
                        for cluster in clusters:
                                min_from_clusters.append(Cluster(cluster.min_by_value()))

                        clusters = min_from_clusters
                else:
                        min_fun_value = clusters[0].min_by_value()
                        nelder_mead = nelderMead(min_fun_value.x_values)
                        print("f({}) = {}".format(nelder_mead.x,
                          ↪   nelder_mead.fun))
                        break
```

Листинг 4. Генетический алгоритм

```
class GeneticAlgorithm():
        def __init__(self):
                self.alpha = 0
                self.beta = 5
                self.population_size = 60
                self.number_of_genes = 20
```

```
7                 self.crossover_probability = 0.5
8                 self.mutation_probability = 1/self.number_of_genes
9                 self.tournament_selection_parameter = 0.75
10                self.tournament_size = 3
11                self.number_of_variables = 3
12                self.variable_range = 5
13                self.number_of_generations = 150
14                self.number_of_best_individual_copies = 1
15                self.fitness = [0 for x in range(self.population_size)]
16
17                self.population =
                    ↪   self.initialize_population(self.population_size,
                    ↪   self.number_of_genes)
18
19        def run_ga(self):
20                k = 0
21                for iGenerations in range(self.number_of_generations):
22                        print(k)
23                        k += 1
24
25                        maximum_fitness = 0.0
26                        x_best = [0 for _ in range(self.number_of_variables)]
27                        best_individual = None
28
29                        # Decode chromosome and evaluate individual
30                        for i in range(self.population_size):
31                                chromosome = self.population[i]
32                                x = self.decode_chromosome(chromosome,
                                    ↪   self.number_of_variables,
                                    ↪   self.variable_range)
33                                self.fitness[i] = self.evaluate_individual(x)
34                                if self.fitness[i] > maximum_fitness:
35                                        maximum_fitness = self.fitness[i]
36                                        x_best = x
37                                        best_individual = chromosome
38                        temp_population = self.population
39
40                        # Tournament selection
41                        for i in range(round(self.population_size/2)):
42                                j = i*2
43                                i1 = self.tournament_select(self.fitness,
                                    ↪   self.tournament_selection_parameter,
                                    ↪   self.tournament_size)
44                                i2 = self.tournament_select(self.fitness,
                                    ↪   self.tournament_selection_parameter,
                                    ↪   self.tournament_size)
45                                chromosome_1 = self.population[i1]
46                                chromosome_2 = self.population[i2]
47
```

```
48                                  # Crossover
49                                  r = random.random()
50                                  if r < self.crossover_probability:
51                                          new_chromosome_pair =
                                              ↪   self.cross(chromosome_1,
                                              ↪   chromosome_2)
52                                          temp_population[j] =
                                              ↪   new_chromosome_pair[0]
53                                          temp_population[j+1] =
                                              ↪   new_chromosome_pair[1]
54                                  else:
55                                          temp_population[j] = chromosome_1
56                                          temp_population[j+1] = chromosome_2
57
58                          # Mutate
59                          for i in range(self.population_size):
60                                  original_chromosome = temp_population[i]
61                                  mutated_chromosome =
                                      ↪   self.mutate(original_chromosome,
                                      ↪   self.mutation_probability)
62                                  temp_population[i] = mutated_chromosome
63
64                          # Insert best individual
65                          if best_individual is not None:
66                                  temp_population =
                                      ↪   self.insert_best_individual(temp_population,
                                      ↪   best_individual,
                                      ↪   self.number_of_best_individual_copies)
67                          population = temp_population
68
69              print("f({}) = {}".format(x_best, 1/maximum_fitness))
70
71          def initialize_population(self, population_size, number_of_genes):
72                  population = [[0 for y in range(number_of_genes)] for x in
                        ↪   range(population_size)]
73                  step = (self.beta - self.alpha) / self.number_of_genes
74                  for i in range(population_size):
75                          for j in range(number_of_genes):
76                                  while True:
77                                          population[i][j] = step *
                                              ↪   random.random() + step
78
79                                          if 0 <= population[i][j] <= 1:
80                                                  break
81                  return population
82
83          def decode_chromosome(self, chromosome, number_of_variables,
              ↪   variable_range):
84                  n_genes = len(chromosome)
```

```
85                  n_split = round(n_genes/self.number_of_variables)
86                  x = [0.0 for x in range(self.number_of_variables)]
87
88                  for i in range(self.number_of_variables):
89                          for j in range(n_split):
90                                  x[i] = x[i] +
                                      ↪   chromosome[n_split*(i-1)+j]*2**(-j)
91                          x[i] = -self.variable_range +
                              ↪   2*self.variable_range*x[i]/(1-2**(-n_split))
92                  return x
93
94          def evaluate_individual(self, x):
95                  # The fitness function
96                  x_values = x
97                  if len(x_values) == 3:
98                          a = 2
99                          b = 3
100                         c = 1
101
102                         g = - 3 * a / (
103                                         -c / 3.0 + b * (pow(x_values[0] -
                                              ↪   4.0, 2) + pow(x_values[1] - 4.0,
                                              ↪   2) + pow(x_values[2] - 4.0, 2)))
104
105                         fitness_value = 1/g
106                 else:
107                         print("ERROR: Wrong size individual")
108                         sys.exit()
109                 return fitness_value
110
111         def tournament_select(self, fitness, tournament_selection_parameter,
              ↪   tournament_size):
112                 i_tmp_vector = [0 for x in range(tournament_size)]
113                 fitness_vector = [0 for x in range(tournament_size)]
114                 i_selected = None
115                 for i in range(tournament_size):
116                         i_tmp_vector[i] =
                              ↪   int(random.random()*self.population_size)
117                         fitness_vector[i] = fitness[i_tmp_vector[i]]
118
119                 no_chosen_index = True
120                 while no_chosen_index:
121                         idx_maximum =
                              ↪   fitness_vector.index(max(fitness_vector))
122                         if len(fitness_vector) > 1:
123                                 if random.random() <
                                      ↪   tournament_selection_parameter:
124                                         i_selected =
                                              ↪   i_tmp_vector[idx_maximum]
```

```
                                        no_chosen_index = False
                                else:
                                        fitness_vector.pop(idx_maximum)
                                        i_tmp_vector.pop(idx_maximum)
                        else:
                                i_selected = i_tmp_vector[0]
                                no_chosen_index = False

                return i_selected

        def cross(self, chromosome_1, chromosome_2):
                n_genes = len(chromosome_1)
                crossover_point = round(random.random() * n_genes)

                new_chromosome_pair = [[0 for y in range(n_genes)] for x in
                  ↪   range(2)]
                for j in range(n_genes):
                        if j < crossover_point:
                                new_chromosome_pair[0][j] = chromosome_1[j]
                                new_chromosome_pair[1][j] = chromosome_2[j]
                        else:
                                new_chromosome_pair[0][j] = chromosome_2[j]
                                new_chromosome_pair[1][j] = chromosome_1[j]

                return new_chromosome_pair

        def mutate(self, chromosome, mutation_probability):
                mutated_chromosome = chromosome
                for j in range(self.number_of_genes):
                        if random.random() < mutation_probability:
                                mutated_chromosome[j] = 1-chromosome[j]

                return mutated_chromosome

        def insert_best_individual(self, population, best_individual,
          ↪   number_of_best_individual_copies):
                for i in range(number_of_best_individual_copies):
                        population[i] = best_individual

                return population
```

Листинг 5. Метод "идеальной точки".

```
r0 = 1.0
accelerator = 2.0
epsilon = 10 ** -3
```

```
k = 0

def function_cut(a):
        if a > 0:
                return a
        return 0

def function_1(x):
        return x[0] ** 2 + x[1] ** 2

def function_2(x):
        return 4 * (x[0] - 5) ** 2 + 2 * (x[1] - 6) ** 4

def cond_1(x):
        return x[1] - x[0] + 1

def cond_2(x):
        return x[0] + x[1] - 2


def effective_points(w1, w2, x1, x2):
        def F(x, r):
                return (w1 * (function_1(x) - function_1(x1)) + w2 *
                  ↪   (function_2(x) - function_2(x2)) + (r / 2.) * (
                                (function_cut(cond_1(x)) ** 2) +
                                  ↪   (function_cut(cond_2(x)) ** 2)))

        def P(x, r):
                return (r / 2.) * ((function_cut(cond_1(x)) ** 2) +
                  ↪   (function_cut(cond_2(x)) ** 2))

        def barrier_method(r, k, x0):
                res = (minimize(lambda x: F(x, r), x0, method='CG'))
                newx = res.x

                if np.fabs(P(newx, r)) <= epsilon:
                        return newx
                else:
                        return barrier_method(accelerator * r, k + 1, newx)

        result = barrier_method(r0, k, [0.0, 0.0])
        print('w1 = {}, w2 = {}, \nx = {}, f_1(x) = {:.2f}, f_2(x) =
          ↪   {:.2f}\n'.format(w1, w2, result, function_1(result),
          ↪   function_2(result)) )
```

5 Результаты.

Были получены следующие результаты:

Листинг 5. Результаты выполнения программ.

```
Start Competitng points method method:
    f([4.0605263157894731, 4.0605263157894731, 4.0605263157894731]) =
     ↪  19.975867872760627

Start Genetic Algorithm:
    f([4.038834661591123, 4.030962344985451, 3.9695648282003884]) =
     ↪  18.56699665784427

Start Effective points method:
    w1 = 1, w2 = 1.0,
    x = [ 1.50000002  0.50000124], f_1(x) = 2.50, f_2(x) = 1879.12

    w1 = 2, w2 = 0.5,
    x = [ 1.50000002  0.50000247], f_1(x) = 2.50, f_2(x) = 1879.12

    w1 = 3, w2 = 0.3333333333333333,
    x = [ 1.5         0.50000329], f_1(x) = 2.50, f_2(x) = 1879.12

    w1 = 4, w2 = 0.25,
    x = [ 1.49999992  0.5000049 ], f_1(x) = 2.50, f_2(x) = 1879.12
```