

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

Лабораторная работа №6. Вариант 1.
«Методы решения задач линейного и
линейного целочисленного программирования»
по курсу
«Методы оптимизации»

Студент группы ИУ9-82

Белогуров А.А.

Преподаватель

Каганов Ю.Т.

Москва, 2018

Содержание

1	Цель работы	3
2	Постановка задачи	4
2.1	Задача 6.1	4
2.2	Задача 6.2	4
3	Исследование	5
3.1	Задача 6.1	5
3.1.1	Симлекс-метод	5
3.2	Задача 6.2	5
3.2.1	Метод ветвей и границ	5
4	Практическая реализация	7
5	Результаты.	14

1 Цель работы

1. Изучение симплекс-метода линейного программирования и методов линейного целочисленного программирования.
2. Разработка программы реализации алгоритма симплекс-метода линейного программирования и алгоритмов линейного целочисленного программирования.
3. Решение задачи линейного и линейного целочисленного программирования.

2 Постановка задачи

Дано: 1 Вариант. Функция:

$$f(x) = x_1 - x_2 + x_3. \quad (1)$$

Функции ограничений:

$$\begin{cases} -x_1 + 2x_2 - x_3 = 4 \\ 3x_1 + x_2 + x_4 = 14 \\ x_1, \dots, x_4 \geq 0 \end{cases} \quad (2)$$

2.1 Задача 6.1

1. Найти условный максимум для задачи линейного программирования симплекс-методом Дж. Данцига.
2. Реализовать алгоритмы с помощью языка программирования высокого уровня.

2.2 Задача 6.2

1. Найти условный максимум для задачи линейного целочисленного программирования методом «ветвей и границ».
2. Реализовать алгоритмы с помощью языка программирования высокого уровня.

3 Исследование

Найдем условный максимум для функции:

$$f(x) = x_1 - x_2 + x_3 \quad (3)$$

с помощью сервиса WolframAlpha.com:

$$\max(f(x)) = 10, \quad (x_1, x_2, x_3, x_4) = (0, 14, 24, 0) \quad (4)$$

3.1 Задача 6.1

3.1.1 Симлекс-метод

Это алгоритм решения оптимизационной задачи линейного программирования путём перебора вершин выпуклого многогранника в многомерном пространстве.

Сущность метода: построение базисных решений, на которых монотонно убывает линейный функционал, до ситуации, когда выполняются необходимые условия локальной оптимальности.

3.2 Задача 6.2

3.2.1 Метод ветвей и границ

Общий алгоритмический метод для нахождения оптимальных решений различных задач оптимизации, особенно дискретной и комбинаторной оптимизации. По существу, метод является вариацией полного перебора с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений.

Общая идея метода может быть описана на примере поиска минимума функции $f(x)$ на множестве допустимых значений переменной

x . Функция f и переменная x могут быть произвольной природы. Для метода ветвей и границ необходимы две процедуры: ветвление и нахождение оценок (границ).

Так в данном случае необходимо решить задачу целочисленного линейного программирования, то необходимо изменить исходную функцию, так как её решением при первой итерации являются целые числа.

Новая функция для **Задачи 6.2** будет выглядеть следующим образом:

$$f(x) = -x_1 + 11x_2 - 1x_3 + 5x_4 \quad (5)$$

Функции ограничений не изменяются.

4 Практическая реализация

Все методы были реализованы на языке программирования **Python**.

Листинг 1. Симлекс-метод.

```
1 class MatrixSimplex:
2     def __init__(self, A, b, c, B_idx):
3         self.A = np.matrix(A)
4         self.b = np.matrix(b)
5         self.c = np.matrix(c)
6
7         self.B_idx = B_idx
8         self.N_idx = [i for i in range(self.A.shape[1]) if i not in
9             ↪ self.B_idx]
10
11     def get_B(self):
12         return self.A[:, self.B_idx]
13
14     def get_Cb(self):
15         return self.c[:, self.B_idx]
16
17     def get_Cn(self):
18         return self.c[:, self.N_idx]
19
20     def get_N(self):
21         return self.A[:, self.N_idx]
22
23     def computeZ(self, InvB, N):
24         #  $z = Cb * InvB * b - (Cb * InvB * N - Cn) * Xn$ 
25         Cb = self.get_Cb()
26         Cb_InvB = np.dot(Cb, InvB)
27
28         Cb_InvB_b = np.dot(Cb_InvB, self.b.T)
29
30         Cb_InvB_N = np.dot(Cb_InvB, N)
31         bracket_term = np.subtract(Cb_InvB_N, self.get_Cn())
32         return Cb_InvB_b, -1 * bracket_term
33
34     def computeXb(self, InvB, N):
35         return np.dot(InvB, self.b.T)
36
37     def is_optimal(self, z):
38         for x in np.nditer(z):
39             if x > 0:
40                 return False
```

```

40         return True
41
42     def compute_entering_variable(self, z):
43         max_ascent = 0
44         index = -1
45         it = np.nditer(z.T, flags=['f_index'])
46         while not it.finished:
47             if it[0] > max_ascent:
48                 max_ascent = it[0]
49                 index = it.index
50             it.iternext()
51         if index is not -1: # else returns None
52             return self.N_idx[index]
53
54     def replace(self, ev_idx, dv_idx):
55         for i in range(len(self.B_idx)):
56             if self.B_idx[i] == dv_idx:
57                 self.B_idx[i] = ev_idx
58                 break
59
60         self.N_idx = [i for i in range(self.A.shape[1]) if i not in
61                        ↪ self.B_idx]
62
63     def calculate_value(self, z, Xb):
64         print("z = ", z[0, 0])
65         for i in range(self.c.shape[1]):
66             index = 0
67             in_basis = False
68             for j in self.B_idx:
69                 if i == j:
70                     print("x[{}] = {}".format(j,
71                                                ↪ Xb[index, 0]))
72                     in_basis = True
73                     index += 1
74             if not in_basis:
75                 print("x[{}] = {}".format(i, 0))
76         print("")
77         return
78
79     def do_simplex(self):
80         print("\n##### Start Simplex #####\n")
81         while True:
82             InvB = np.linalg.inv(self.get_B())
83             N = self.get_N()
84
85             z, z_var = self.computeZ(InvB, N)
86
87             Xb = self.computeXb(InvB, N)

```



```

87         if self.is_optimal(z_var):
88             print("Optimal solution found..")
89             self.calculate_value(z, Xb)
90             return z
91
92         ev_idx = self.compute_entering_variable(z_var)
93         if ev_idx is None:
94             print("Invalid")
95             return 0
96
97         ev = self.A[:, ev_idx]
98         dv_idx = self.min_ratio_test(Xb, ev)
99
100        if dv_idx is None:
101            print("Invalid")
102            return 0
103
104        self.replace(ev_idx, dv_idx)

```

Листинг 2. Метод ветвей и границ.

```

1  class Node:
2      def __init__(self, x_bounds=[], freeze_var_list=[], index=0,
3          ↪ upper_or_lower=0):
4          self._x_bounds = x_bounds
5          self._freeze_var_list = freeze_var_list
6          self._index = index
7          self._upper_or_lower = upper_or_lower
8
9      def freeze_lower_var(self, index, val):
10         self._x_bounds[index] = (None, val)
11         self._freeze_var_list.append(index)
12
13     def freeze_upper_var(self, index, val):
14         self._x_bounds[index] = (val, None)
15         self._freeze_var_list.append(index)
16
17     def freeze_var(self, index, val):
18         self._x_bounds[index] = (val, val)
19         self._freeze_var_list.append(index)
20
21     def set_lp_res(self, res):
22         self._res = res
23
24     def check_integer_var_all_solved(self, m):
25         return True if m == len(self._freeze_var_list) else False

```

```

25
26
27 def check_all_integers(list):
28     is_integer = True
29     for i in range(len(list) - 1):
30         if not is_int(list[i]):
31             is_integer = False
32             break
33
34     return is_integer
35
36
37 def calculate(A, b, c):
38     global node_counter
39
40     x_bounds = [(0, None) for i in range(len(c))]
41
42     print("\n##### Start B & B #####\n")
43
44     node = Node(copy.deepcopy(x_bounds), [], node_counter)
45
46     node_counter += 1
47     res = solve_LP(x_bounds, A, b, c)
48
49     if check_all_integers(res.x):
50         print_result(res)
51         return res
52
53
54     lower = floor(res['x'][integer_var[0]])
55     upper = lower + 1
56
57     lower_node = Node(copy.deepcopy(x_bounds), [], node_counter, 1)
58     lower_node.freeze_lower_var(integer_var[0], lower)
59     add_dangling_node(lower_node, A, b, c)
60
61     node_counter += 1
62
63     upper_node = Node(copy.deepcopy(x_bounds), [], node_counter, 2)
64     upper_node.freeze_upper_var(integer_var[0], upper)
65     add_dangling_node(upper_node, A, b, c)
66
67     node_counter += 1
68
69     while len(dangling_nodes) > 0:
70
71         index = np.argmin(dangling_nodes_obj)
72
73         x_b = dangling_nodes[index]._x_bounds

```

```

74     frez = dangling_nodes[index]._freeze_var_list
75     res = dangling_nodes[index]._res
76     frez_var_index = len(frez)
77
78     u_or_l = dangling_nodes[index]._upper_or_lower
79     arbitrary_node = Node(copy.deepcopy(x_b),
80         ↪ copy.deepcopy(frez), node_counter,
81         ↪ copy.deepcopy(u_or_l))
82     u_or_l_b = lower - 1 if (u_or_l == 1) else upper + 1
83     arbitrary_node.freeze_var(integer_var[frez_var_index - 1],
84         ↪ u_or_l_b)
85     x_b_arbi = arbitrary_node._x_bounds
86     if check_bounds(x_b_arbi, integer_var[frez_var_index - 1],
87         ↪ u_or_l, x_bounds):
88         add_dangling_node(arbitrary_node, A, b, c)
89     else:
90         print("arbitrary Node infeasibile: ",
91             ↪ arbitrary_node._index)
92
93     node_counter += 1
94
95     lower = floor(res['x'][integer_var[frez_var_index]])
96     upper = lower + 1
97
98     lower_node = Node(copy.deepcopy(x_b), copy.deepcopy(frez),
99         ↪ node_counter, 1)
100     lower_node.freeze_lower_var(integer_var[frez_var_index],
101         ↪ lower)
102     add_dangling_node(lower_node, A, b, c)
103
104     node_counter += 1
105
106     upper_node = Node(copy.deepcopy(x_b), copy.deepcopy(frez),
107         ↪ node_counter, 2)
108     upper_node.freeze_upper_var(integer_var[frez_var_index],
109         ↪ upper)
110     add_dangling_node(upper_node, A, b, c)
111
112     node_counter += 1
113
114     # убрать break в санукке
115     # if check_all_integers():
116     break
117
118     result = get_max_integer_node()
119     print_result(result._res)
120
121 def is_int(n):

```

```

114         return int(n) == float(n)
115
116
117 def solve_LP(x_bounds, A_eq, b_eq, c):
118     return Simplex(c, A_eq=A_eq, b_eq=b_eq, bounds=x_bounds)
119
120
121 def print_result(result):
122     x_list = result.x
123     print("z = {}".format(result.fun))
124
125     for i, item in enumerate(x_list):
126         print("x[{}] = {}".format(i, item))
127
128 def get_max_integer_node():
129     global dangling_nodes
130     global dangling_nodes_obj
131     global result_Node
132
133     integer_nodes = []
134     for node in dangling_nodes:
135         if check_all_integers(node._res.x):
136             integer_nodes.append(node)
137
138     max_node_result = float("-INF")
139     max_node = None
140     for node in integer_nodes:
141         if node._res.fun >= max_node_result:
142             max_node = node
143             max_node_result = node._res.fun
144
145     return max_node
146
147 def add_dangling_node(node, A, b, c):
148     global z_star
149     global dangling_nodes
150     global dangling_nodes_obj
151     global result_Node
152
153     res = solve_LP(node._x_bounds, A, b, c)
154     if check_feasibility(res) and res['fun'] > z_star:
155         node.set_lp_res(res)
156
157         if check_all_integers(res.x):
158             dangling_nodes_obj.append(res['fun'])
159             dangling_nodes.append(node)
160         elif len(dangling_nodes) == 0:
161             dangling_nodes_obj.append(res['fun'])
162             dangling_nodes.append(node)

```

```
163         else:
164             return
165
166         if node.check_integer_var_all_solved(len(integer_var)):
167             z_star = res['fun']
168             result_Node = node
169         return True
170     else:
171         return False
```

5 Результаты.

Программы, представленные в **Листинге 1** и **Листинге 2** дали следующие результаты:

Листинг 5. Результаты выполнения программ.

```
1  Start Simplex method:
2      z = 10.0
3      x[0] = 0
4      x[1] = 14.0
5      x[2] = 24.0
6      x[3] = 0
7
8  Start B&B method
9      [(None, 3), (None, 3), (0, None), (0, None)]
10     z = 56.0
11     x[0] = 2.0
12     x[1] = 3.0
13     x[2] = 0.0
14     x[3] = 5.0
```

Проверка полученных значений для ограничений:

$$\begin{cases} -x_1 + 2x_2 - x_3 = 4 \\ 3x_1 + x_2 + x_4 = 14 \\ x_1, \dots, x_4 \geq 0 \end{cases} \quad (6)$$

Симплекс-метод:

$$\begin{cases} -1 \times 0 + 2 \times 14 - 1 \times 24 + 0 \times 0 = 4 \\ 3 \times 0 + 1 \times 14 + 0 \times 0 + 1 \times 0 = 14 \\ x_1, \dots, x_4 \geq 0 \end{cases} \quad (7)$$

Проверка прошла успешно.

Метод ветвей и границ:

$$\begin{cases} -1 \times 2 + 2 \times 3 - 1 \times 0 + 0 \times 5 = 4 \\ 3 \times 2 + 1 \times 3 + 0 \times 0 + 1 \times 5 = 14 \\ -\infty \leq x_1 \leq 3, \quad -\infty \leq x_2 \leq 3, \quad 0 \leq x_3 \leq \infty, \quad 0 \leq x_4 \leq \infty \end{cases} \quad (8)$$

Проверка прошла успешно.