

|  |
| --- |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |

**Институт** Информационных Технологий

**Кафедра** Вычислительной Техники

**ПРАКТИЧЕСКАЯ РАБОТА №5**

**по дисциплине**

**«Теория принятия решений»**

**Симплексный метод**

Студент группы: ИКБО-15-22 Оганнисян Г.А. *(Ф.И.О. студента)*

Преподаватель Железняк Л.М.

*(Ф.И.О. преподавателя)*

|  |  |  |
| --- | --- | --- |
|  |  |  |

Москва 2024

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc133167282)

[1 СИМПЛЕКСНЫЙ МЕТОД 4](#_Toc133167283)

[1.1 Постановка задачи 4](#_Toc133167284)

[1.2 Математическая модель задачи 4](#_Toc133167285)

[ЗАКЛЮЧЕНИЕ 13](#_Toc133167286)

[СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ 14](#_Toc133167287)

[ПРИЛОЖЕНИЯ 15](#_Toc133167288)

ВВЕДЕНИЕ

В данной работе мы рассматриваем применение симплекс-метода для решения задач линейного программирования с произвольным количеством переменных. Симплекс-метод представляет собой эффективный алгоритм, который начинает решение задачи с анализа вершин многогранника условий. Путем последовательного перехода от одной вершины к другой, улучшая значение функции цели, метод стремится к достижению оптимального решения. В случае максимизации функции цели, метод ищет вершину, где значение функции увеличивается, а при минимизации - уменьшается.

Преимущество симплекс-метода заключается в том, что он гарантирует нахождение оптимального решения или устанавливает, что задача неразрешима, за конечное число шагов. Этот метод представляет собой целенаправленный перебор опорных решений ЗЛП в многомерном пространстве переменных.

Таким образом, симплекс-метод является мощным инструментом для решения сложных задач линейного программирования, обеспечивая эффективный и надежный подход к оптимизации целевой функции при линейных ограничениях.

**1 СИМПЛЕКСНЫЙ МЕТОД**

* 1. **Постановка задачи**

Решить прямую ЗЛП с помощью симплексного метода и обратную с помощью теорем двойственности. Определить интервалы устойчивости.

***Задача.*** Фабрика может производить тарелки и кружки. На производство тарелки идет 5 единиц материала, на производство кружки – 20 единиц (керамики). Тарелка требует 10 человеко-часа, кружка – 15. На производство тарелки тратится 0,5 кВатт электроэнергии, кружки – 0,3. Расходы при производстве тарелки равны 1 рубль, а кружки – 2 рубля. Имеется 400 единиц материала и 450 человеко-часов, 25 кВатт энергии и объем накладных расходов равен 300 рублей. Эти данные представлены в таблице 1.1.

*Таблица 1.1.* Исходные данные задачи.

|  |  |  |  |
| --- | --- | --- | --- |
| Ресурс | Товар | | Объем  ресурса |
| Тарелки | Кружки |
| Материал | 5 | 20 | 400 |
| Человеко-часы | 10 | 15 | 450 |
| Электроэнергия | 0,5 | 0,3 | 25 |
| Расходы | 1 | 2 | 300 |

Прибыль при производстве тарелки – 20 рублей, при производстве кружки – 50 рублей. Сколько надо сделать тарелок и кружек, чтобы получить максимальную прибыль?

**1.2 Математическая модель задачи**

Пусть х1 – тарелки, х2 – кружки. Максимальный выпуск продукции составит .

Ограничения задачи:

Таким образом, переходим к задаче линейного программирования:

Приведем задачу к канонической форме. Для этого в левые части ограничений вводим дополнительные переменные: х3 ≥ 0, х4 ≥ 0, х5 ≥ 0, х6 ≥ 0. Эти переменные выбираются так, чтобы они обращали неравенства в равенства.

Построим начальную симплекс-таблицу. Запишем систему в векторной форме:

Векторы 𝐴3, 𝐴4, 𝐴5, 𝐴6 являются линейно независимыми единичными векторами 3х-мерного пространства и образуют базис этого пространства.

Поэтому за базисные переменные выбираем переменные *x*3,𝑥4, 𝑥5, 𝑥6. Небазисными переменными являются 𝑥1, 𝑥2. Разложение позволяет найти первое базисное допустимое решение.

Для этого свободные переменные 𝑥1, 𝑥2 приравниваем нулю. В результате получим разложение

Которому соответствует первоначальный опорный план

Для проверки плана 𝑥(0) на оптимальность построим первую симплекс-таблицу. Введем в рассмотрение вектор коэффициентов целевой функции при базисных переменных.

В левый столбец Таблицы 1.2 запишем переменные *x*3, 𝑥4, 𝑥5, 𝑥6 образующие базис, в верхней строке – небазисные переменные 𝑥1, 𝑥2. В строке 𝑐j запишем коэффициенты целевой функции, соответствующие небазисным переменным с1 = 20, с2 = 50. В столбце запишем коэффициенты целевой функции, соответствующие базисным переменным Столбец, определяемый переменной 𝑥1, состоит из коэффициентов вектора . Аналогично, столбец, определяемый переменной 𝑥2, состоит из коэффициентов вектора . Крайний правый столбец заполняется элементами столбца , в нем же в результате вычислений получаем оптимальный план.

Заполнение f-строки (Таблица 1.3). Найдем относительные оценки ∆1, ∆2 и значение целевой функции 𝑄.

*Таблица 1.2 – Начальная симплекс-таблица задачи*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | сj | 20 | 50 |  |  |
|  |  | X1 | X2 |  |
| 0 | X3 | 5 | 20 | 400 |
| 0 | X4 | 10 | 15 | 450 |
| 0 | X5 | 0,5 | 0,3 | 25 |
| 0 | X6 | 1 | 2 | 300 |
|  | f |  |  |  |
|  |  | Δ1 | Δ2 | Q |  |

*Таблица 1.3 – Заполнение f-строки*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | сj | 20 | 50 |  |  |  | |
|  |  | X1 | X2 |  |  | |
| 0 | X3 | 5 | 20 | 400 | 400 / 20 = 20 *min* | |
| 0 | X4 | 10 | 15 | 450 | 450 / 15 = 30 | |
| 0 | X5 | 0,5 | 0,3 | 25 | 25 / 0,3 = 32 | |
| 0 | X6 | 1 | 2 | 300 | 300 / 2 = 150 | |
|  | f | -20 | -50 | 0 |  | |
|  |  | Δ1 | Δ2 | Q |  |  | |

Для оптимальности опорного решения в задаче на максимум требуется выполнение не отрицательности всех относительных оценок ∆i ≥ 0. Так как оценки ∆1= −20 и ∆2= −50 в f-строке отрицательны, то это свидетельствуют о возможности улучшения полученного решения. Наибольшая по модулю отрицательная оценка ∆2= −50. В базис будет включена соответствующая ей небазисная переменная 𝑥2. Составим отношения свободных членов к положительным элементам разрешающего столбца. Данные отношения приведены справа от таблицы. Наименьшему частному соответствует строка с переменной 𝑥5. Эта переменная исключается из базиса. В Таблице 1.3 разрешающий столбец и разрешающая строка выделены. Разрешающим элементом является число 𝑎13 = 20.

Далее построим новую симплекс-таблицу. Ниже поэтапно демонстрируется процесс заполнения новой симплекс-таблицы (Таблицы 1.4, 1.5).

*Таблица 1.4 – Новая симплекс-таблица*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | сj | 20 | 0 |  |  |
|  |  | X1 | X3 |  |
| 50 | X2 |  | 0,05 |  |
| 0 | X4 |  |  |  |
| 0 | X5 |  |  |  |
| 0 | X6 |  |  |
|  | f |  |  |  |
|  |  | Δ1 | Δ2 | Q |

В Таблице 1.4 переменные 𝑥2 и 𝑥3 меняются местами вместе с коэффициентами 𝑐𝑗. Разрешающий элемент заменяется на обратный. В Таблице 1.5 элементы разрешающей строки делятся на разрешающий элемент. Элементы разрешающего столбца делятся на разрешающий элемент и меняют знак.

*Таблица 1.5 – Симплекс преобразования*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | сj | 20 | 0 |  |  |
|  |  | X1 | X3 |  |
| 50 | X2 | 0.25 | 0.05 | 20 |
| 0 | X4 |  | -0.75 |  |
| 0 | X5 |  | -0.02 |  |
| 0 | X6 |  | -0.1 |
|  | f |  | 2.5 |  |
|  |  | Δ1 | Δ2 | Q |

*Таблица 1.6 – Итерация 1*

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | сj | 20 | 0 |  |  |  | |
|  |  | X1 | X3 |  |  | |
| 50 | X2 | 0.25 | 0.05 | 20 | 20 / 0.25 =  *80* | |
| 0 | X4 | 6.25 | -0.75 | 150 | 150 / 6.25 = 24 *min* | |
| 0 | X5 | 0.42 | -0.02 | 19 | 19 / 0.42 = 45.23 | |
| 0 | X6 | 0.5 | -0.1 | 260 | 260 / 0.5 = 520 | |
|  | f | -7.5 | 2.5 | 1000 |  | |
|  |  | Δ1 | Δ2 | Q |  |  | |

Остальные элементы (Таблица 1.6) рассчитываются по «правилу прямоугольника».

Базисное решение, которое дает последняя таблица

Это решение не является оптимальным, так как в f-строке имеется отрицательная оценка ∆1.

В Таблице 1.6 разрешающий столбец и разрешающая строка выделены. Разрешающим элементом является число 𝑎21 = 6.25.

Поэтому, построим новую симплекс-таблицу. Ниже поэтапно демонстрируется процесс заполнения новой симплекс-таблицы (Таблицы 1.7, 1.8).

*Таблица 1.7 – Новая симплекс-таблица*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | сj | 0 | 0 |  |  |
|  |  | X4 | X3 |  |
| 50 | X1 |  |  |  |
| 20 | X2 | 0.16 |  |  |
| 0 | X5 |  |  |  |
| 0 | X6 |  |  |
|  | f |  |  |  |
|  |  | Δ1 | Δ2 | Q |

В Таблице 1.4 переменные 𝑥1 и 𝑥3 меняются местами вместе с коэффициентами 𝑐𝑗. Разрешающий элемент заменяется на обратный. В Таблице 1.5 элементы разрешающей строки делятся на разрешающий элемент. Элементы разрешающего столбца делятся на разрешающий элемент и меняют знак.

*Таблица 1.8 – Симплекс преобразования*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | сj | 0 | 0 |  |
|  |  | X4 | X3 |  |
| 50 | X1 | -0.04 |  |  |
| 20 | X2 | 0.16 | -0.12 | 24 |
| 0 | X5 | -0.07 |  |  |
| 0 | X6 | -0.08 |  |
|  | f | 1.2 |  |  |
|  |  | Δ1 | Δ2 | Q |

*Таблица 1.9 – Итерация 2*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | сj | 0 | 0 |  |
|  |  | X4 | X3 |  |
| 50 | X1 | -0.04 | 0.08 | 14 |
| 20 | X2 | 0.16 | -0.12 | 24 |
| 0 | X5 | -0.07 | 0.04 | 8.8 |
| 0 | X6 | -0.08 | -0.04 | 248 |
|  | f | 1.2 | 1.6 | 1180 |
|  |  | Δ1 | Δ2 | Q |

Остальные элементы (Таблица 1.6) рассчитываются по «правилу прямоугольника».

Если в последней таблице f-строке не содержит отрицательных оценок, то это свидетельствует об оптимальности полученного решения:

Подставляем базисное решение, которое дает последняя таблица

Где n – количество итераций, n – степень у x.

Проверим решение по «правилу прямоугольника».

Таким образом, предприятие должно выпускать в течении недели 𝑥1 = 24 шт. тарелок и 𝑥2 = 14 шт. кружек. Тогда предприятие получит программу по максимальному доходу - 1180 [шт.].

**ЗАКЛЮЧЕНИЕ**

В ходе работы мы изучили и применили симплекс-метод для решения задач линейного программирования с произвольным количеством переменных. Симплекс-метод представляет собой эффективный алгоритм, основанный на последовательном переходе от одной вершины многогранника условий к другой с целью улучшения значения целевой функции.

Плюсы симплекс-метода:

1. Гарантированное нахождение оптимального решения или определение неразрешимости задачи в конечное число шагов.
2. Эффективность в решении сложных задач оптимизации при линейных ограничениях.
3. Относительная простота реализации и понимания алгоритма.
4. Возможность применения к широкому спектру задач линейного программирования.

Минусы симплекс-метода:

1. Неэффективность в случае большого количества переменных или огромных размеров задачи.
2. Возможность зацикливания в некоторых случаях, что приводит к невозможности нахождения оптимального решения.
3. Не всегда обеспечивает оптимальное решение на практике из-за ограничений математической модели или особенностей конкретной задачи.

Тем не менее, несмотря на некоторые ограничения, симплекс-метод остается мощным инструментом для решения множества задач линейного программирования, обеспечивая надежное и точное определение оптимальных решений.

**СПИСОК ИНФОРМАЦИОННЫХ ИСТОЧНИКОВ**

1. Болотова Л. С. Многокритериальная оптимизация. Болотова Л. С., Сорокин А. Б. [Электронный ресурс] / Метод. указания по вып. курсовой работы — М.: МИРЭА, 2015.
2. Сорокин А. Б. Методы оптимизации: гибридные генетические алгоритмы. Сорокин А. Б. [Электронный ресурс] / Учебно-метод. пособие — М.: МИРЭА, 2016.
3. Сорокин А. Б. Линейное программирование: практикум. Сорокин А. Б., Бражникова Е. В., Платонова О. В. [Электронный ресурс] / Учебно-метод. пособие — М.: МИРЭА, 2017.

**ПРИЛОЖЕНИЯ**

Приложение А – Код реализации симплексного метода на языке Python.

**Приложение А**

Код реализации симплексного метода на языке Python.

*Листинг А.1. Реализация симплексного метода.*

import numpy as np

class LinearModel:

# Инициализация параметров модели

def \_\_init\_\_(self, A=np.empty([0, 0]), b=np.empty([0, 0]), c=np.empty([0, 0]), minmax="MAX"):

self.A = A # Матрица коэффициентов ограничений

self.b = b # Вектор правой части ограничений

self.c = c # Вектор коэффициентов целевой функции

self.x = [float(0)] \* len(c) # Начальное решение (все переменные равны нулю)

self.minmax = minmax # Тип оптимизации (минимизация или максимизация)

self.printIter = True # Флаг для печати итераций

self.optimalValue = None # Оптимальное значение целевой функции

self.transform = False # Флаг для преобразования модели

def addA(self, A): # Установка матрицы коэффициентов ограничений

self.A = A

def addB(self, b): # Установка вектора правой части ограничений

self.b = b

def addC(self, c): # Установка вектора коэффициентов целевой функции

self.c = c

self.transform = False

def setObj(self, minmax): # Установка типа оптимизации

self.minmax = minmax

self.transform = False

def setPrintIter(self, printIter): # Установка флага для печати итераций

self.printIter = printIter

def printSoln(self): # Печать решения и оптимального значения

print(" Коэффициенты: ")

print("", self.x)

print("\n Оптимальное значение: ")

print("", self.optimalValue)

def getTableau(self): # Создание симплекс-таблицы

num\_var = len(self.c) # Получение количества переменных

num\_slack = len(self.A) # Получение количества ограничений

# Создание верхней строки таблицы

t1 = np.hstack(([None], [0], self.c, [0] \* num\_slack))

*Продолжение листинга А.1. Реализация симплексного метода.*

|  |
| --- |
| # Создание базисных переменных и расширение матрицы А, если необходимо  basis = np.array([0] \* num\_slack) # Создание массива для базисных переменных  for i in range(0, len(basis)):  basis[i] = num\_var + i # Установка индексов базисных переменных  A = self.A  if not ((num\_slack + num\_var) == len(self.A[0])):  # Если матрица A не квадратная, добавляем единичную матрицу для расширения  B = np.identity(num\_slack)  A = np.hstack((self.A, B))  # Создание нижних строк таблицы  t2 = np.hstack((np.transpose([basis]), np.transpose([self.b]), A))  # Объединение верхней и нижней частей таблицы  tableau = np.vstack((t1, t2)) # Слияние верхней и нижней частей таблицы  tableau = np.array(tableau, dtype='float') # Преобразование в массив NumPy  return tableau # Возвращение симплекс-таблицы  def optimize(self): # Оптимизация симплекс-методом  tableau = self.getTableau() # Получение симплекс-таблицы  if self.printIter:  print(" Стартовая таблица:")  self.print\_table(tableau, True) # Печать начальной симплекс-таблицы  optimal = False # Флаг для проверки на оптимальность  iter = 0 # Счетчик итераций  while 1:  if self.printIter:  if iter > 0:  print("\n=====================\n")  print(" Итерация :", iter)  self.print\_table(tableau, False) # Печать текущей симплекс-таблицы  for profit in tableau[0, 2:]:  if profit > 0:  optimal = False  break  optimal = True  if optimal:  break  n = tableau[0, 2:].tolist().index(np.amax(tableau[0, 2:])) + 2 # Выбор разрешающего столбца  minimum = 99999 # Инициализация минимального значения  r = -1 # Инициализация разрешающей строки  for i in range(1, len(tableau)):  if tableau[i, n] > 0:  val = tableau[i, 1] / tableau[i, n]  if val != 0 and val < minimum:  minimum = val # Обновление минимального значения  r = i # Обновление разрешающей строки  pivot = tableau[r, n] # Получение разрешающего элемента  print("\n Разрешающий столбец:", n - 1)  print(" Разрешающая строка:", r)  print(" Разрешающий элемент: ", pivot)  tableau[r, 1:] = tableau[r, 1:] / pivot # Деление строки на разрешающий элемент  for i in range(0, len(tableau)): |

*Продолжение листинга А.1. Реализация симплексного метода.*

|  |
| --- |
| if i != r:  mult = tableau[i, n] / tableau[r, n] # Вычисление множителя  tableau[i, 1:] = tableau[i, 1:] - mult \* tableau[r, 1:] # Обновление строк  tableau[r, 0] = n - 2 # Обновление индекса базисной переменной в таблице  iter += 1 # Увеличение счетчика итераций  if self.printIter:  print("\n----------------------------------\n")  print(" Финальная таблица была получена за", iter, "итерации")  self.print\_table(tableau, False) # Печать финальной симплекс-таблицы  else:  print("Решено")  self.x = np.array([0] \* len(self.c), dtype=float) # Создание массива для решения  for key in range(1, (len(tableau))):  if tableau[key, 0] < len(self.c):  self.x[int(tableau[key, 0])] = tableau[key, 1] # Обновление значений переменных  self.optimalValue = -1 \* tableau[0, 1] # Установка оптимального значения  def print\_table(self, tableau, start): # Функция для печати симплекс-таблицы  print("ind A0\t\t ", end="") # Печать заголовка столбца с индексом и A0  for i in range(1, len(self.c) + 1): # Печать заголовков столбцов переменных x  print("x\_" + str(i), end="\t ")  for i in range(1, 5): # Печать заголовков столбцов правой части ограничений  print("b\_" + str(i), end="\t ")  print() # Переход на новую строку после печати заголовка  for j in range(0, len(tableau)): # Перебор строк таблицы  for i in range(0, len(tableau[0])): # Перебор элементов в строке  if not np.isnan(tableau[j, i]): # Проверка, что элемент не NaN  if i == 0: # Если это первый столбец (индекс базисной переменной)  print('x\_' + str(int(tableau[j, i]) + 1), end="\t ")  else:  if j == 0 and start is False: # Если это первая строка и start равно False  if round(tableau[j, i], 2) == 0:  print(round(tableau[j, i], 2), end="\t ") # Если значение округленное до 2 знаков после запятой равно 0  else:  print((-1) \* round(tableau[j, i], 2), end="\t ") # В противном случае, печать отрицательного значения  else:  print(round(tableau[j, i], 2), end="\t ") # Если не первая строка или start равно True  else:  print('F', end="\t ") # Если элемент NaN, печать символа 'F' вместо значения  print() # Переход на новую строку после печати строки таблицы |

*Продолжение листинга А.1. Реализация симплексного метда*

|  |
| --- |
| if \_\_name\_\_ == '\_\_main\_\_':  model1 = LinearModel()  A = np.array(  [  [16, 12],  [0.2, 0.4],  [6, 5],  [3, 4]  ]  )  b = np.array(  [1200, 30, 600, 300]  )  c = np.array(  [260, 300]  )  model1.addA(A)  model1.addB(b)  model1.addC(c)  print("\n Дано:")  print("> A =\n", A, "\n")  print("> А0 =\n", b, "\n")  print("> C =\n", c, "\n\n")  model1.optimize()  print("\n")  model1.printSoln() |