Петрозаводский государственный университет Институт математики и информационных технологий

Кафедра Информатики и математического обеспечения

Направление подготовки бакалавриата

09.03.04 Программная инженерия Профиль направления подготовки бакалавриата

“Системное и прикладное программное обеспечение”

Отчет по дисциплине «Верификация программного обеспечения»

Система решения симплекс-методом

Выполнил:

студент группы 22407

С.А. Калюшин

Преподаватель:

К.А. Кулаков, к.ф.–м.н., доцент

Петрозаводск — 2022

1. Объект тестирования

1.1. Описания приложения

Приложение представляет собой интерфейс для получения данных и алгоритм решения симплекс-методом. Симплекс-метод — алгоритм решения оптимизационной задачи линейного программирования путём перебора вершин выпуклого многогранника в многомерном пространстве. Написано на языке Python.

1.2. Функциональность объекта тестирования

В приложении реализованы следующие функции:

1. Парсинг входного текста для получения структуры уравнений.
2. Уведомления о некорректном вводе.
3. Приведение уравнения к каноническому виду.
4. Решение системы уравнений.
5. Преобразование этапов решения для демонстрации:
    a. Метода искусственного базиса
    b. Базисного допустимого решения
    c. Итерации решения симплекс-методом

1.3. Требования к объекту тестирования

К приложению сформулированы следующие требования:

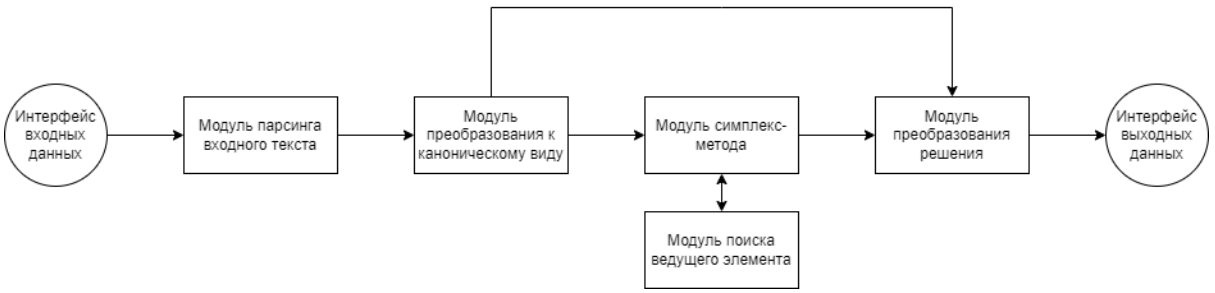
- Приложение должно работать с целочисленными коэффициентами уравнений.
- Приложение должно в равной мере обрабатывать входные данные на русском и английском языке.

2. Стратегия тестирования

2.1. Используемые инструменты

- Для блочного тестирования: библиотека Unittest для тестов и библиотека mock для создания заглушек.
- Для интеграционного тестирования: инструменты и окружение аналогичны тем, которые используются в блочном тестировании.
- Для аттестационного тестирования: специальные инструменты не используются. Тестирование проводится ручным методом.

2.2. Архитектура



Архитектура представляет собой набор модулей, которые последовательно преобразуют входные данные для представления их через интерфейс выходных данных.

- Модуль парсинга входного текста – представляет собой модуль, который преобразует входные данные в массивы коэффициентов, знаков, ограничений. Определяет предел целевой функции и количество её переменных.
  Включает в себя методы parse(), parse_func(), parse_restrict().
  - parse(self):
    Описание: метод получает текст из поля ввода целевой функции, вызывает метод parse_func(), записывает количество переменных и коэффициенты целевой функции, а также её предел, вызывает метод parse_restrict(), записывает количество ограничений и

коэффициенты ограничений.
Возвращаемое значение: code

- parse_func(self, string):
  Описание: метод преобразует аргумент string в массив коэффициентов и предел функции.
  Возвращаемое значение: mass_coef, limit, code
- parse_restrict(self, string):
  Описание: метод преобразует аргумент string в матрицу коэффициентов ограничений, массив их знаков и массив коэффициентов правой части.
  Возвращаемое значение: matrix_coef, signs, right_coef, code

- Модуль преобразования к каноническому виду – преобразует систему ограничений адаптируя её для симплекс метода, после чего получает базисное решение.
  Включает в себя методы transform_canon_view(), find_basis().
  - transform_canon_view(self):
    Описание: метод создает таблицу коэффициентов на основе параметров (глобальных переменных) и приводит к каноническому виду.
    Возвращаемое значение: table, code
  - find_basis(self, table):
    Описание: метод предназначен для нахождения базиса и базисного допустимого решения.
    Возвращаемое значение: table, basis, code
- Модуль симплекс метода – основной модуль решения.
  Включает в себя методы solve_simplex(), simplex(), find_column(), find_row(), find_new_table().
  - solve_simplex(self, table, basis):
    Описание: метод решает систему симплекс-методом: вызывая требуемое кол-во раз метод simplex()
    Возвращаемое значение: new_table, last_basis, code
  - simplex(self, table):
    Описание: метод симплекса, вызывает методы find_column(), find_row(), find_new_table() для поиска ведущего элемента.
    Возвращаемое значение: table, basis, code

    - find_column(self, table):
      Описание: метод поиска ведущего столбца.
      Возвращаемое значение: index
    - find_row(self, table, index_column):
      Описание: метод поиска ведущей строки.
      Возвращаемое значение: index
    - simplex(self, table, index_column, index_row):
      Описание: метод поиска новой таблицы после преобразования.
      Возвращаемое значение: new_table
- Модуль преобразования решения – модуль формирования фрейма вывода, трансформирует данные с этапов решения в соответствующий вид (текст, таблица). Включает в себя методы show_hide_chapter(), generate_output_view().
    - show_hide_chapter(self, index):
      Описание: метод скрытия/расркытия фрейма по его индексу.
      Возвращаемое значение: нет
    - generate_output_view(self):
      Описание: генерация фреймов этапов решений.
      Возвращаемое значение: нет

2.3. Стратегии блочного тестирования

Блочное тестирование будет проводиться методом автоматизированного тестирования с использованием библиотек Unittest и mock. Для проведения блочных тестов будут использоваться различные варианты входных данных, составленные для проведения тестов и направленные на покрытие различных вариантов выполнения.

Блочное тестирование должно быть применено к следующим методам: parse, parse_func, parse_restrict, simplex, solve_simplex, find_column, find_row, find_new_table, transform_canon_view, find_basis. Для проведения блочного тестирования метода parse

должна быть использована заглушка для вызова parse_func и parse_restrict. Для каждого пути в блок-схеме метода должен быть разработан как минимум один тест.

Блочное тестирование не будет приведено для метода show_hide_chapter, т.к. метод лишь переключает состояния фреймов. Метод generate_output_view будет в аттестационном тестировании, т.к. только формирует фрейм выходных данных на основе значений глобальных переменных.

2.4. Стратегии интеграционного тестирования

Интеграционное тестирование будет проводиться методом автоматизированного тестирования с использованием библиотеки Unittest.

Интеграционное тестирование будет проведено для взаимодействий: parse, parse_func, parse_restrict, solve_simplex, simplex, find_column, find_row.

Будут рассмотрены следующие комбинации:

1. parse + (parse_func и parse_restrict)
   Описание: метод parse внутри себя вызывает методы parse_func и parse_restrict
2. simplex + (find_column и find_row)
   Описание: метод simplex внутри себя вызывает методы find_column и find_row

3. solve_simplex + simplex

   Описание: solve_simplex внутри себя вызывает некоторое количество раз метод simplex

Последовательность проведения важна для комбинации №2 и №3, т.к. №3 комбинация включает в себя и №2.

2.5. Стратегии аттестационного тестирования

Аттестационное тестирование будет проводиться методом ручного тестирования. В рамках аттестационного тестирования должно быть проверено функционирование сервисов (см. п. 1.2) отображения выходных данных.

За выходные данные отвечает метод generate_output_view(). Результатом работы метода должно стать визуальное отображение этапов решения: парсинг входного текста, канонический вид, базисное допустимое решение, решение симплекс-методом и окончательный ответ.

2.6. Стратегии нагрузочного тестирования

Нагрузочное тестирование предполагает измерение изменений во времени выполнения решения для разного количества переменных системы.

3. Детальный план тестов

3.1. Входные/выходные данные

matrix_restrict1: [[1, 1], [-1, 1], [1, -1], [0, 1]]

mass_func1: [1, 2]

signs1 = [1, 1, 2, 2]

right_values1 = [4, 2, 2, 6]

table_for_simplex1:

[[4, 0, 0, 0, -1, 1, 0, 0, 1],

[4, 1, 0, 0, 1, 0, 1, 0, -1],

[2, 2, 0, -1, 1, 0, 0, 1, -1],

[2, -1, 1, 0, -1, 0, 0, 0, 1],

[1999999, 2000000, 0, -1000000, 1000000, 0, 0, 0, -2000000]]

new_table_for_simplex1:

[[4, 0, 0, 0, -1, 1, 0, 0, 1],

[3, 0, 0, 0.5, 0.5, 0, 1, -0.5, -0.5],

[1, 1, 0, -0.5, 0.5, 0, 0, 0.5, -0.5],

[3, 0, 1, -0.5, -0.5, 0, 0, 0.5, 0.5],

[-7, 0, 0, 1.5, 0.5, 0, 0, -1000001.5, -1000001.5]]

final_table_for_simplex1:

[[4, 0, 0, 0, -1, 1, 0, 0, 1],

[6, 0, 0, 1, 1, 0, 2, -1, -1],

[4, 1, 0, 0, 1, 0, 1, 0, -1],

[6, 0, 1, 0, 0, 0, 1, 0, 0],

[-16, 0, 0, 0, -1, 0. -3, -1000000, -999999]]

table_for_canon_view1:

[1, 1, -1, 0, 0, 0, 4],

[-1, 1, 0, -1, 0, 0, 2]

[1, -1, 0, 0, 1, 0, 2]

[0, 1, 0, 0, 0, 1, 6]]

table_for_find_basis0:

[[-2, -1, 0, 0, 0],

[1, -1, 1, 0, 10],

[2, 0, 0, 1, 40]]

3.2. Блочное тестирование

3.2.1. Тест Б-1

Цель теста (описание): Проверка результата парсинга целевой функции при корректных данных.

Тип теста: Позитивный

Объект тестирования: метод parse_func(self, string)

Входные данные: string = “2x1+3x2-4x3=>Min”

Ожидаемый результат: mass_coef = [2, 3, -4], limit = “Min”, code = 0

3.2.2. Тест Б-2

Цель теста (описание): Проверка результата парсинга целевой функции при отсутствии коэффициента в явном виде.

Тип теста: Позитивный

Объект тестирования: метод parse_func(self, string)

Входные данные: string = “x1+3x2-4x3=>Min”

Ожидаемый результат: mass_coef = [1, 3, -4], limit = “Min”, code = 0

3.2.3. Тест Б-3

Цель теста (описание): Проверка результата парсинга целевой функции при отсутствии последовательного порядка коэффициентов.

Тип теста: Позитивный

Объект тестирования: метод parse_func(self, string)

Входные данные: string = “3x2+2x1-4x3=>Min”

Ожидаемый результат: mass_coef = [2, 3, -4], limit = “Min”, code = 0

3.2.4. Тест Б-4

Цель теста (описание): Проверка результата парсинга целевой функции при пропуске коэффициента

Тип теста: Негативный

Объект тестирования: метод parse_func(self, string)

Входные данные: string = “3x2+4x3=>Min”

Ожидаемый результат: mass_coef = [], limit = “”, code = 1

3.2.5. Тест Б-5

Цель теста (описание): Проверка результата парсинга целевой функции при несоответствии формата ввода

Тип теста: Негативный

Объект тестирования: метод parse_func(self, string)

Входные данные: string = “3x2+2x1-4x3=>abdcf”

Ожидаемый результат: mass_coef = [], limit = “”, code = 1

3.2.6. Тест Б-6

Цель теста (описание): Проверка результата парсинга ограничений.

Тип теста: Позитивный

Объект тестирования: метод parse_restrict(self, string)

Входные данные: string[] = (“3x1+2x2-4x3=>4”, “1x1+2x2-4x3=>2”), count_var = 3

Ожидаемый результат: matrix_coef = [[3, 2, -4],[1, 2, -4]], signs = [“=>”, “=>”], right_coef = [4, 2], code = 0

3.2.7. Тест Б-7

Цель теста (описание): Проверка результата парсинга ограничений при меньшем кол-ве переменных.

Тип теста: Позитивный

Объект тестирования: метод parse_restrict(self, string)

Входные данные: string[] = (“3x1+2x2=>4”, “1x1+2x2-4x3=>2”), count_var = 3

Ожидаемый результат: matrix_coef = [[3, 2, 0],[1, 2, -4]], signs = [“=>”, “=>”], right_coef = [4, 2], code = 0

3.2.8. Тест Б-8

Цель теста (описание): Проверка результата парсинга ограничений при отсутствии явного коэффициента.

Тип теста: Позитивный

Объект тестирования: метод parse_restrict(self, string)

Входные данные: string[] = (“x1+2x2-4x3=>4”, “1x1+2x2-4x3=>2”), count_var = 3

Ожидаемый результат: matrix_coef = [[1, 2, -4],[1, 2, -4]], signs = [“=>”, “=>”], right_coef = [4, 2], code = 0

3.2.9. Тест Б-9

Цель теста (описание): Проверка результата парсинга ограничений при большем количестве переменных в ограничении, чем было выявлено в целевой функции.

Тип теста: Негативный

Объект тестирования: метод parse_restrict(self, string)

Входные данные: string[] = (“x1+2x2-4x3+5x4=>4”, “1x1+2x2-4x3=>2”), count_var = 3

Ожидаемый результат: matrix_coef = [], signs = [], right_coef =], code = 1

3.2.10. Тест Б-10

Цель теста (описание): Проверка результата парсинга ограничений при некорректном вводе.

Тип теста: Негативный

Объект тестирования: метод parse_restrict(self, string)

Входные данные: string[] = (“x1+2x2-4x3+5x4=>4fdsfs”, “1x1+2x2-4x3=>2”), count_var = 3

Ожидаемый результат: matrix_coef = [], signs = [], right_coef =], code = 1

3.2.11. Тест Б-11

Цель теста (описание): Проверка результата парсинга входных данных.

Тип теста: Позитивный

Объект тестирования: метод parse(self)

Входные данные: func_entry = “2x1+3x2-4x3=>Min”, text_restrict=”3x1+2x2-4x3=>4\n1x1+2x2-4x3=>2” (заглушки для parse_func и parse_restrict)

Ожидаемый результат: count_var = 3 , count_restrict = 2, code = 0

3.2.12. Тест Б-12

Цель теста (описание): Проверка результата парсинга входных данных при некорректных данных.

Тип теста: Негативный

Объект тестирования: метод parse(self)

Входные данные: func_entry=”3x2+4x3=>Min”, text_restrict=”x1+2x2-4x3+5x4=>4fdsfs\n1x1+2x2-4x3=>2” (заглушки для parse_func и parse_restrict)

Ожидаемый результат: count_var = 0 , count_restrict = 0, code = 1

3.2.13. Тест Б-13

Цель теста (описание): Проверка результата парсинга входных данных при отсутствии целевой функции.

Тип теста: Позитивный

Объект тестирования: метод parse(self)

Входные данные: func_entry=””, text_restrict=”3x1+2x2-4x3=>4\n1x1+2x2-4x3=>2” (заглушки для parse_func и parse_restrict)

Ожидаемый результат: count_var = 0 , count_restrict = 2, code = 0

3.2.14. Тест Б-14

Цель теста (описание): Проверка результата парсинга входных данных при отсутствии ограничений.

Тип теста: Позитивный

Объект тестирования: метод parse(self)

Входные данные: func_entry=”2x1+3x2-4x3=>Min”, text_restrict=”” (заглушки для parse_func и parse_restrict)

Ожидаемый результат: count_var = 3 , count_restrict = 0, code = 0

3.2.15. Тест Б-15

Цель теста (описание): Проверка результата парсинга входных данных при отсутствии целевой функции и ограничений.

Тип теста: Позитивный

Объект тестирования: метод parse(self)

Входные данные: func_entry="", text_restrict="" (заглушки для parse_func и parse_restrict)

Ожидаемый результат: count_var = 0 , count_restrict = 0, code = 0

3.2.16. Тест Б-16

Цель теста (описание): Проверка корректного поиска ведущего столбца.

Тип теста: Позитивный

Объект тестирования: метод find_column(self, table)

Входные данные: table = table_for_simplex1 (см. в тестовых данных)

Ожидаемый результат: index = 0

3.2.17. Тест Б-17

Цель теста (описание): Проверка поиска ведущего столбца при некорректных данных.

Тип теста: Негативный

Объект тестирования: метод find_column(self, table)

Входные данные: table = []

Ожидаемый результат: index = -1

3.2.18. Тест Б-18

Цель теста (описание): Проверка корректного поиска ведущей строки.

Тип теста: Позитивный

Объект тестирования: метод find_row(self, table, index_column)

Входные данные: table = table_for_simplex1 (см. в тестовых данных), index_column = 0

Ожидаемый результат: index = 2

3.2.19. Тест Б-19

Цель теста (описание): Проверка поиска ведущей строки при некорректном индексе.

Тип теста: Негативный

Объект тестирования: метод find_row(self, table, index_column)

Входные данные: table = table_for_simplex1 (см. в тестовых данных), index_column = -1

Ожидаемый результат: index = -1

3.2.20. Тест Б-20

Цель теста (описание): Проверка поиска ведущей строки при некорректных данных.

Тип теста: Негативный

Объект тестирования: метод find_row(self, table, index_column)

Входные данные: table =[], index_column = 0

Ожидаемый результат: index = -1

3.2.21. Тест Б-21

Цель теста (описание): Проверка поиска новой таблицы.

Тип теста: Позитивный

Объект тестирования: метод find_new_table(self, table, index_column, index_row)

Входные данные: table = table_for_simplex1 (см. в тестовых данных), index_column = 0, index_row = 2

Ожидаемый результат: new_table = new_table_for_simplex1 (см. в тестовых данных)

3.2.22. Тест Б-22

Цель теста (описание): Проверка поиска новой таблицы при некорректном индексе столбца.

Тип теста: Негативный

Объект тестирования: метод find_new_table(self, table, index_column, index_row)

Входные данные: table = table_for_simplex1 (см. в тестовых данных), index_column = 24, index_row = 2

Ожидаемый результат: new_table = null

3.2.23. Тест Б-23

Цель теста (описание): Проверка поиска новой таблицы при некорректном индексе строки.

Тип теста: Негативный

Объект тестирования: метод find_new_table(self, table, index_column, index_row)

Входные данные: table = table_for_simplex1 (см. в тестовых данных), index_column = 0, index_row = -4

Ожидаемый результат: new_table = null

3.2.24. Тест Б-24

Цель теста (описание): Проверка поиска новой таблицы при некорректных данных таблицы.

Тип теста: Негативный

Объект тестирования: метод find_new_table(self, table, index_column, index_row)

Входные данные: table = [], index_column = 0, index_row = 2

Ожидаемый результат: new_table = null

3.2.25. Тест Б-25

Цель теста (описание): Проверка симплекс метода.

Тип теста: Позитивный

Объект тестирования: метод simplex(self, table, basis) с заглушками для find_column = 0, find_row = 2, find_new_table = new_table_for_simplex1

Входные данные: table = table_for_simplex1 (см. в тестовых данных), basis = [5, 6, 7, 2]

Ожидаемый результат: new_table = new_table_for_simplex1, new_basis = [5, 6, 1, 2], code = 0

3.2.26. Тест Б-26

Цель теста (описание): Проверка поиска решения.

Тип теста: Позитивный

Объект тестирования: метод solve_simplex(self, table, basis) с заглушками для simplex()

Входные данные: table = table_for_simplex1 (см. в тестовых данных), basis = [5, 6, 7, 2]

Ожидаемый результат: new_table=final_table_for_simplex1, last_basis=[5, 3, 1, 2], code = 0

3.2.27. Тест Б-27

Цель теста (описание): Проверка поиска решения при некорректном базисном решении.

Тип теста: Позитивный

Объект тестирования: метод solve_simplex(self, table, basis) с заглушками для simplex()

Входные данные: table = [], basis = [5, 6, 7, 2]

Ожидаемый результат: new_table=[], basis = [5, 6, 7, 2], code = 1

3.2.28. Тест Б-28

Цель теста (описание): Проверка поиска решения при некорректном базисе.

Тип теста: Позитивный

Объект тестирования: метод solve_simplex(self, table, basis) с заглушками для simplex()

Входные данные: table = table_for_simplex1 (см. в тестовых данных), basis = [0, 6, 7, 2]

Ожидаемый результат: new_table=table_for_simplex1, basis = [0, 6, 7, 2], code = 1

3.2.29. Тест Б-29

Цель теста (описание): Проверка приведения к каноническому виду.

Тип теста: Позитивный

Объект тестирования: метод transform_canon_view(self)

Входные данные: matrix_restrict = matrix_restrict1, mass_func = mass_func1, signs = signs1, right_values = right_values1

Ожидаемый результат: new_table=table_for_canon_view1, code = 0

3.2.30. Тест Б-30

Цель теста (описание): Проверка приведения к каноническому виду при отсутствии коэффициентов ограничений.

Тип теста: Негативный

Объект тестирования: метод transform_canon_view(self)

Входные данные: matrix_restrict =[], mass_func = mass_func1, signs = signs1, right_values = right_values1

Ожидаемый результат: new_table=null, code = 1

3.2.31. Тест Б-31

Цель теста (описание): Проверка приведения к каноническому виду при отсутствии целевой функции.

Тип теста: Негативный

Объект тестирования: метод transform_canon_view(self)

Входные данные: matrix_restrict = matrix_restrict1, mass_func = [], signs = signs1, right_values = right_values1

Ожидаемый результат: new_table=null, code = 1

3.2.32. Тест Б-32

Цель теста (описание): Проверка приведения к каноническому виду при отсутствии знаков.

Тип теста: Негативный

Объект тестирования: метод transform_canon_view(self)

Входные данные: matrix_restrict = matrix_restrict1, mass_func = mass_func1, signs = [], right_values = right_values1

Ожидаемый результат: new_table=null, code = 1

3.2.33. Тест Б-33

Цель теста (описание): Проверка приведения к каноническому виду при отсутствии правых значений.

Тип теста: Негативный

Объект тестирования: метод transform_canon_view(self)

Входные данные: matrix_restrict = matrix_restrict1, mass_func = mass_func1, signs = signs1, right_values = []

Ожидаемый результат: new_table=null, code = 1

3.2.34. Тест Б-34

Цель теста (описание): Проверка приведения к каноническому виду при несоответствии количества переменных.

Тип теста: Негативный

Объект тестирования: метод transform_canon_view(self)

Входные данные: matrix_restrict = [2, 5, 6, 7, 8], mass_func = mass_func1, signs = signs1, right_values = []

Ожидаемый результат: new_table=null, code = 1

3.2.35. Тест Б-35

Цель теста (описание): Проверка поиска базисного решения.

Тип теста: Позитивный

Объект тестирования: метод find_basis(self, table)

Входные данные: table = table_for_canon_view1 (см. в тестовых данных)

Ожидаемый результат: new_table=table_for_simplex1 , code = 0

3.2.36. Тест Б-36

Цель теста (описание): Проверка поиска базисного решения при отсутствии решения.

Тип теста: Позитивный

Объект тестирования: метод find_basis(self, table)

Входные данные: table = table_for_find_basis0 (см. в тестовых данных)

Ожидаемый результат: new_table=table_for_find_basis0 , code = 1

Пример блочных тестов

```
class TestCalculate(unittest.TestCase):
    # Блочные тесты на функцию parse_func
    # Позитивный тест
    def test_b1(self):
        window = Tk()
        calculator = simplex.SimplexCalculator(window)
        mass_coef, limit, code = calculator.parse_func('2x1+3x2-4x3=>Min')
        self.assertTrue(mass_coef == [2, 3, -4] and limit == 'Min' and code == 0)

    # Позитивный тест
    def test_b2(self):
        window = Tk()
        calculator = simplex.SimplexCalculator(window)
        mass_coef, limit, code = calculator.parse_func('x1+3x2-4x3=>Min')
        self.assertTrue(mass_coef == [1, 3, -4] and limit == 'Min' and code == 0)

    # Позитивный тест
    def test_b3(self):
        window = Tk()
```

3.3. Интеграционное тестирование

3.3.1. Тест И-1

Цель теста (описание): Проверка результата парсинга входных данных.

Тип теста: Позитивный

Объект тестирования: метод parse() + parse_func()

Входные данные: func_entry=”2x1+3x2-4x3=>Min”, (заглушка для parse_restrict)

Ожидаемый результат: count_var = 3 , count_restrict = 2, code = 0

3.3.2. Тест И-2

Цель теста (описание): Проверка результата парсинга входных данных при отсутствии целевой функции.

Тип теста: Негативный

Объект тестирования: метод parse() + parse_func()

Входные данные: func_entry=””, (заглушка для parse_restrict)

Ожидаемый результат: count_var = 0 , count_restrict = 2, code = 1

3.3.3. Тест И-3

Цель теста (описание): Проверка результата парсинга при несоответствии формата ввода

Тип теста: Негативный

Объект тестирования: метод parse() + parse_func()

Входные данные: func_entry=”fdsfsdd”, (заглушка для parse_restrict)

Ожидаемый результат: count_var = 0 , count_restrict = 2, code = 1

3.3.4. Тест И-4

Цель теста (описание): Проверка результата парсинга

Тип теста: Позитивный

Объект тестирования: метод parse() + parse_func() + parse_restrict()

Входные данные: func_entry = “2x1+3x2-4x3=>Min”, text_restrict=”3x1+2x2-4x3=>4\n1x1+2x2-4x3=>2”

Ожидаемый результат: count_var = 3 , count_restrict = 2, code = 0

3.3.5. Тест И-5

Цель теста (описание): Проверка результата парсинга при несоответствии формата ввода

Тип теста: Позитивный

Объект тестирования: метод parse() + parse_func() + parse_restrict()

Входные данные: func_entry = “2x1+3x2-4x3=>sfdsdf”, text_restrict=”3x1+2x2-4x3=>4\n1x1+2x2-4x3=>2”

Ожидаемый результат: count_var = 0 , count_restrict = 0, code = 1

3.3.6. Тест И-6

Цель теста (описание): Проверка симплекс метода.

Тип теста: Позитивный

Объект тестирования: метод simplex(self, table, basis) + find_column(self, table) (с заглушками для find_row = 2, find_new_table = new_table_for_simplex1)

Входные данные: table = table_for_simplex1 (см. в тестовых данных), basis = [5, 6, 7, 2]

Ожидаемый результат: new_table = new_table_for_simplex1, new_basis = [5, 6, 1, 2], code = 0

3.3.7. Тест И-7

Цель теста (описание): Проверка симплекс метода.

Тип теста: Позитивный

Объект тестирования: метод simplex(self, table, basis) + find_column(self, table) + find_row (self, table, index_column) (с заглушкой для find_new_table = new_table_for_simplex1)

Входные данные: table = table_for_simplex1 (см. в тестовых данных), basis = [5, 6, 7, 2]

Ожидаемый результат: new_table = new_table_for_simplex1, new_basis = [5, 6, 1, 2], code = 0

3.3.8. Тест И-8

Цель теста (описание): Проверка симплекс метода.

Тип теста: Позитивный

Объект тестирования: метод simplex(self, table, basis) + find_column(self, table) + find_row (self, table, index_column) + find_new_table(self, table, index_column, index_row)

Входные данные: table = table_for_simplex1 (см. в тестовых данных), basis = [5, 6, 7, 2]

Ожидаемый результат: new_table = new_table_for_simplex1, new_basis = [5, 6, 1, 2], code = 0

3.3.9. Тест И-9

Цель теста (описание): Проверка поиска решения.

Тип теста: Позитивный

Объект тестирования: метод solve_simplex(self, table, basis) + simplex(self, table, basis)

Входные данные: table = table_for_simplex1 (см. в тестовых данных), basis = [5, 6, 7, 2]

Ожидаемый результат: new_table=final_table_for_simplex1, last_basis=[5, 3, 1, 2], code = 0

3.3.10. Тест И-10

Цель теста (описание): Проверка поиска решения при некорректном базисном решении.

Тип теста: Позитивный

Объект тестирования: метод solve_simplex(self, table, basis) с заглушками для simplex(self, table, basis)

Входные данные: table = [], basis = [5, 6, 7, 2]

Ожидаемый результат: new_table=[], basis = [5, 6, 7, 2], code = 1

3.3.11. Тест И-11

Цель теста (описание): Проверка поиска решения при некорректном базисе.

Тип теста: Позитивный

Объект тестирования: метод solve_simplex(self, table, basis) + simplex(self, table, basis)

Входные данные: table = table_for_simplex1 (см. в тестовых данных), basis = [0, 6, 7, 2]

Ожидаемый результат: new_table=table_for_simplex1, basis = [0, 6, 7, 2], code = 1

Пример интеграционного теста

```
# Интеграционные тесты на метод parse -> вызываемые методы parse_func и parse_restrict
# Позитивный тест
def test_i4(self):
    window = Tk()
    calculator = simplex.SimplexCalculator(window)
    calculator.func_entry.insert(0, '2x1+3x2-4x3=>Min')
    calculator.text_restrict.insert(END, "3x1+2x2-4x3=>4\n1x1+2x2-4x3=>2")
    code = calculator.parse()
    self.assertEqual(calculator.count_var, 3)
    self.assertEqual(calculator.count_restrict, 2)
    self.assertEqual(code, 0)
```

3.4. Аттестационное тестирование

3.4.1. Тест А-1

Цель теста (описание)	Проверка парсинга текста
Функциональное требование	1
Тип	Позитивный
Сценарий	1. Запустить приложение 2. Ввести целевую функцию 3. Ввести ограничения

	4. Нажать на кнопку <<Решить симплекс-методом>>
Ожидаемый результат	В разделе входных данных представлены введёные пользователем уравнения

3.4.2. Тест А-2

Цель теста (описание)	Проверка приведения к каноническому виду
Функциональное требование	3
Тип	Позитивный
Сценарий	1. Запустить приложение 2. Ввести целевую функцию 3. Ввести ограничения 4. Нажать на кнопку <<Решить симплекс-методом>>
Ожидаемый результат	В разделе <<Задача в каноническом виде>> выводится система уравнений с дополнительными переменными

3.4.3. Тест А-3

Цель теста (описание)	Проверка решения системы
Функциональное требование	4
Тип	Позитивный
Сценарий	1. Запустить приложение 2. Ввести целевую функцию 3. Ввести ограничения 4. Нажать на кнопку <<Решить симплекс-методом>>
Ожидаемый результат	В разделе <<Результат>> выводится решение системы

3.4.4. Тест А-4

Цель теста (описание)	Проверка вывода метода искусственного базиса
Функциональное требование	5a
Тип	Позитивный
Сценарий	1. Запустить приложение 2. Ввести целевую функцию 3. Ввести ограничения 4. Нажать на кнопку <<Решить симплекс-методом>>
Ожидаемый результат	В разделе <<Метод искусственного базиса>> отображается поэтапное изменение базиса

3.4.5. Тест A-5

Цель теста (описание)	Проверка отображения БДР
Функциональное требование	5b
Тип	Позитивный
Сценарий	1. Запустить приложение 2. Ввести целевую функцию 3. Ввести ограничения 4. Нажать на кнопку <<Решить симплекс-методом>>
Ожидаемый результат	В разделе <<БДР>> выводится представление базисного допустимого решения

3.4.6. Тест A-6

Цель теста (описание)	Проверка представлений итераций симплекс-метода
Функциональное требование	5c
Тип	Позитивный

Сценарий	1. Запустить приложение 2. Ввести целевую функцию 3. Ввести ограничения 4. Нажать на кнопку <<Решить симплекс-методом>>
Ожидаемый результат	В разделе <<Симплекс-метод>> выводятся итерации решения симплекс-методом

3.4.7. Тест А-7

Цель теста (описание)	Проверка системы уведомлений при отсутствии целевой функции
Функциональное требование	2
Тип	Негативный
Сценарий	1. Запустить приложение 2. Ввести ограничения 3. Нажать на кнопку <<Решить симплекс-методом>>
Ожидаемый результат	Выводится уведомление об отсутствии ввода целевой функции

3.4.8. Тест А-8

Цель теста (описание)	Проверка системы уведомлений при некорректном вводе целевой функции
Функциональное требование	2
Тип	Негативный
Сценарий	1. Запустить приложение 2. Ввести некорректно целевую функцию: “abcdefg” 3. Ввести ограничения 4. Нажать на кнопку <<Решить симплекс-методом>>

Ожидаемый результат	Выводится уведомление о некорректном вводе целевой функции

3.4.9. Тест А-9

Цель теста (описание)	Проверка системы уведомлений при некорректном вводе ограничений
Функциональное требование	2
Тип	Негативный
Сценарий	1. Запустить приложение 2. Ввести целевую функцию 3. Ввести некорректно ограничения: “abcdefg” 4. Нажать на кнопку <<Решить симплекс-методом>>
Ожидаемый результат	Выводится уведомление о некорректном вводе ограничений

4. Журнал тестирования

4.1. Блочное тестирование

Тест	Дата	№ попытки	Результат	Ошибка
Б1	12.12.2022	1	Пройден	
Б2	12.12.2022	1	Пройден	
Б3	12.12.2022	1	Не пройден	№1
Б3	12.12.2022	2	Пройден	
Б4	12.12.2022	1	Не пройден	№2
Б4	12.12.2022	2	Пройден	
Б5	12.12.2022	1	Пройден	
Б6	12.12.2022	1	Пройден	
Б7	12.12.2022	1	Пройден	
Б8	12.12.2022	1	Пройден	
Б9	12.12.2022	1	Пройден	
Б10	12.12.2022	1	Пройден	
Б11	12.12.2022	1	Пройден	
Б12	12.12.2022	1	Пройден	
Б13	12.12.2022	1	Пройден	
Б14	12.12.2022	1	Не пройден	№3
Б14	12.12.2022	2	Пройден	
Б15	12.12.2022	1	Пройден	
Б16	09.12.2022	1	Пройден	
Б17	09.12.2022	1	Пройден	
Б18	09.12.2022	1	Пройден	
Б19	09.12.2022	1	Пройден	
Б20	09.12.2022	1	Пройден	

Б21	09.12.2022	1	Пройден	
Б22	09.12.2022	1	Пройден	
Б23	09.12.2022	1	Пройден	
Б24	09.12.2022	1	Пройден	
Б25	09.12.2022	1	Пройден	
Б26	12.12.2022	0	Не пройден	№4
Б27	12.12.2022	1	Пройден	
Б28	12.12.2022	0	Не пройден	№5
Б29	12.12.2022	1	Пройден	
Б30	12.12.2022	1	Пройден	
Б31	12.12.2022	1	Пройден	
Б32	12.12.2022	1	Не пройден	№6
Б32	12.12.2022	2	Пройден	
Б33	12.12.2022	1	Пройден	
Б34	12.12.2022	1	Пройден	
Б35	12.12.2022	1	Пройден	
Б36	12.12.2022	1	Пройден	

4.2. Интеграционное тестирование

Тест	Дата	№ попытки	Результат	Ошибка
И1	12.12.2022	1	Пройден	
И2	12.12.2022	1	Пройден	
И3	12.12.2022	1	Пройден	
И4	12.12.2022	1	Пройден	
И5	12.12.2022	1	Пройден	

И6	12.12.2022	1	Пройден	
И7	12.12.2022	1	Пройден	
И8	12.12.2022	1	Пройден	
И9	12.12.2022	1	Пройден	
И10	12.12.2022	1	Пройден	
И11	12.12.2022	1	Пройден	

4.3. Аттестационное тестирование

Тест	Дата	№ попытки	Результат	Ошибка
А1	12.12.2022	1	Пройден	
А2	12.12.2022	1	Пройден	
А3	12.12.2022	1	Пройден	
А4	12.12.2022	1	Пройден	
А5	12.12.2022	1	Пройден	
А6	12.12.2022	1	Пройден	
А7	12.12.2022	1	Не пройден	№7
А8	12.12.2022	1	Не пройден	№8
А9	12.12.2022	1	Не пройден	№9

4.4. Нагрузочное тестирование

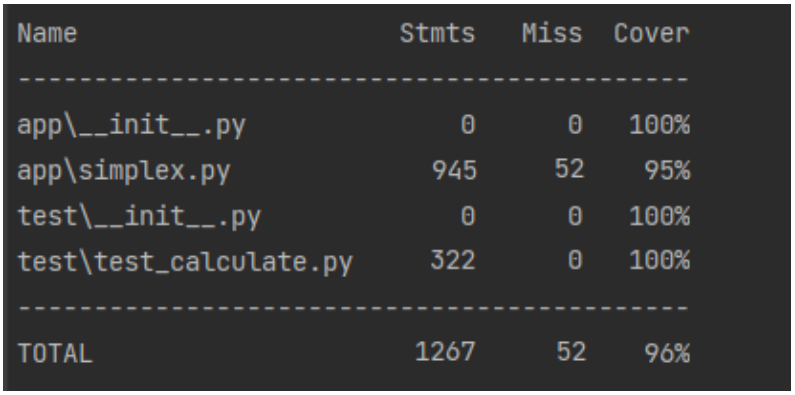
Нагрузочное тестирование провести не удалось, за неимением решаемой системы с большим количеством переменных (10, 50 и 100)

5. Журнал ошибок

№ отчёта об ошибке	Дата	№ теста	Описание	Статус
1	12.12.2022	Б3	Неверное распределение коэффициентов из-за опечатки в индексе	Решено
2	12.12.2022	Б4	Не было реализации проверки	Решено
3	12.12.2022	Б14	Неверное составление теста, а конкретно заглушки для parse_restrict	Решено
4	12.12.2022	Б26	Не вышло составить тест с вариантами заглушки	Не решено
5	12.12.2022	Б28	Не вышло составить тест с вариантами заглушки	Не решено
6	12.12.2022	Б32	Не было реализации проверки массива знаков	Решено
7	12.12.2022	А7	Нет реализации системы уведомлений	Не решено
8	12.12.2022	А8	Нет реализации системы уведомлений	Не решено
9	12.12.2022	А9	Нет реализации системы уведомлений	Не решено

6. Покрытие тестами

Для оценки покрытия использовался пакет coverage для Python.



```
Name                      Stmts   Miss  Cover
-------------------------------------------
app\__init__.py               0      0   100%
app\simplex.py              945     52    95%
test\__init__.py              0      0   100%
test\test_calculate.py      322      0   100%
-------------------------------------------
TOTAL                      1267     52    96%
```

В результате покрытие составляет 95% (96% учитывает еще и файл с тестами, поэтому не является искомым значением).

Такое высокое значение обусловлено линейностью программы (после одного метода выполняется другой), поэтому один запуск всей программы проходит почти по всему коду.

7. Результаты

В рамках дисциплины «Верификация программного обеспечения» были протестированы функциональные возможности информационной системы решения симплекс-методом.

Данное тестирование помогло работе проекта выявить ошибки в приложении. В ходе блочного, интеграционного, аттестационного и нагрузочного тестирования модулей приложения в рамках выполнения проекта было выявлено 9 ошибок, 4 из которых были решены.

Оценки покрытия 95% (причина высокого показателя указана в п.6).

Заявленная в п. 1.2. функциональность была протестирована.