Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Лабораторная работа №1
по дисциплине
“Линейная алгебра и анализ данных”

Семестр I

Выполнили:
студенты

Тиганов Вадим Игоревич
гр. J3112
ИСУ 467701

Вагин Арсений Антонович
гр. J3112
ИСУ 465339

Волков Владимир Алексеевич
гр. J3112
ИСУ 465430

Отчет сдан:
11.12.2024

Санкт-Петербург
2024

Цель лабораторной работы

Освоить основные концепции линейной алгебры и анализа данных по работе с матрицами. Научиться реализовывать и тестировать алгоритмы работы с матрицами в разреженно-строчном формате. Изучить и понять принципы работы алгоритмов, а также .tex верстания для создания отчета.

Задачи лабораторной работы

Реализация хранения матриц в разреженно-строчном виде.

Реализация операций над матрицами.

Вычисление определителя и проверка существования обратной матрицы.

Тестирование и проверка правильности работы алгоритмов.

Верстка отчета в формате LaTeX

Ход выполнения лабораторной работы

Задача 1

Задача заключалась в реализации следующих функий в классе: (был выбран ЯП Python, полный листинг кода см. в приложении А)

- Ввод матрицы заданного размера пользователем.
- Подсчет следа матрицы.
- Поиск и вывод элемента матрицы по заданным индексам.
- Тестирование работы программы и создание консольного пользовательского интерфейса.

Использованные библиотеки и инструменты языка

В ходе написания программы были использованы только стандартные средства языка. Также для удобства и лучшей читаемости кода была импортирована библиотека typing.

Реализация функций и основные идеи

Все функции были реализованы в классе MatrixKeeper

Написаны функции:
1. inputMatrix - ввод матрицы пользователем,
2. trace - поиск следа матрицы,
3. findByIndex - поиск элемента по введенному индексу.

Суть работы алгоритмов:

- inputMatrix: Приглашение пользователя ко вводу. Вначале через пробел вводятся два числа типа int - размер матрицы. Вторым приглашением вводится матрица по строке, элементы в строке разделяются пробелом.
- trace: След матрицы - сумма элементов главной диагонали этой матрицы. Циклом, оставаясь в пределах матрицы, проходимся по элементам с индексами вида [i][i], считаем сумму таких элементов. Можем так делать по той причине, что матрица имеет следующую структуру в классе:

```
self.matrix: Optional[List[List[float]]]
```

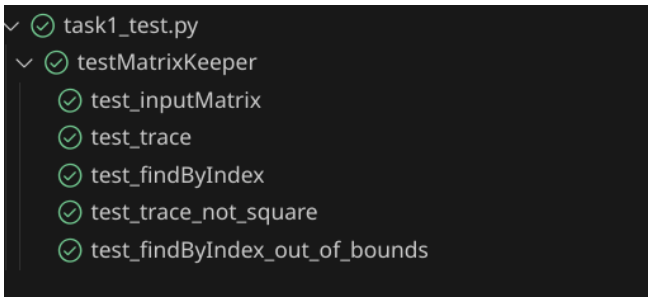
- то есть храним матрицу как список, каждый элемент которого является тоже списком.

- findByIndex: возвращаем элемент из матрицы, отнимая от индексов по единице, т.к. в ЯП отсчет начинается с нуля.

```
return self.matrix[n-1][m-1]
```

Тестирование программы

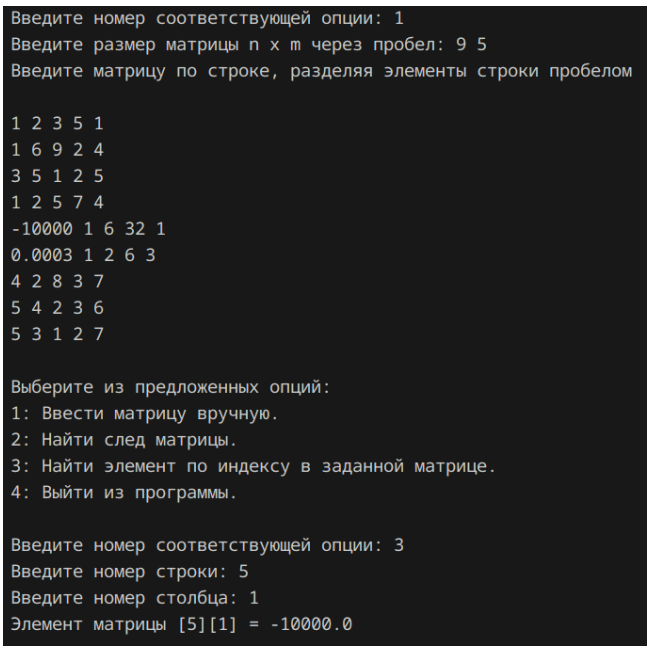
Написаны юниттесты для каждой функции класса с помощью стандартной библиотеки unittest. Также протестировано вручную.
Листинг кода теста см. в приложении A-test.



Успешное прохождение unittests

```
Выберите из предложенных опций:
1: Ввести матрицу вручную.
2: Найти след матрицы.
3: Найти элемент по индексу в заданной матрице.
4: Выйти из программы.

Введите номер соответствующей опции: 1
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

2 2
2 2

Выберите из предложенных опций:
1: Ввести матрицу вручную.
2: Найти след матрицы.
3: Найти элемент по индексу в заданной матрице.
4: Выйти из программы.

Введите номер соответствующей опции: 2
След матрицы: 4.0
```

Ручной тест поиска следа матрицы 2x2 со всеми элементами, равными 2.



Ручной тест поиска страшного элемента в страшной матрице 9x5.

Итак, справились с первым заданием.

Задача 2

Во второй задаче требуется реализовать три функции для операций с матрицами: (полный листинг кода см. в приложении Б)

- Сложение двух матриц.
- Умножение двух матриц.
- Умножение матрицы на скаляр.

Использованные библиотеки и инструменты языка

В ходе написания программы были использованы только стандартные средства языка. Также для удобства и лучшей читаемости кода была импортирована библиотека typing.

Были реализованы 3 функции:
1. matrixAddition - сложение двух матриц.
2. matrixByMatrixMultiplication - перемножение двух матриц.
3. matrixScalarMultiplication - умножение одной из двух матриц на заданное число.

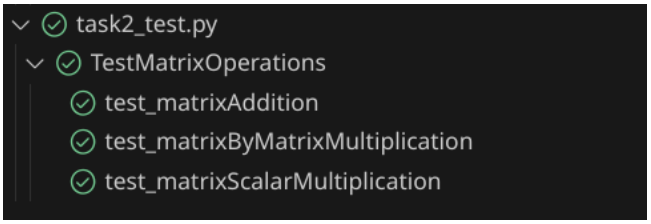
Согласно техническому заданию, функция ввода матрицы пользователем была импортирована из файла предыдущего задания. (inputMatrix)

Суть работы алгоритмов:

- matrixAddition: Классическое сложение матрицы. Возвращаем матрицу, где каждый элемент с определенными индексами равен сумме элементов с соответствующими индексами из складываемых матриц. $(\mathrm{A}+\mathrm{B})_{i,k} = A_{i,k} + B_{i,k}$
- matrixByMatrixMultiplication: Умножение матрицы на матрицу. Возвращаем матрицу, где каждый элемент с определенными индексами равен сумме произведений элементов соответствующей строки первой матрицы и столбца второй матрицы. $(\mathrm{AB})_{i,j} = \sum_{k=1}^{n} A_{i,k} \cdot B_{k,j}$
- matrixScalarMultiplication: Умножение матрицы на скаляр. Возвращаем матрицу, где каждый элемент равен произведению соответствующего элемента исходной матрицы и скаляра. $(\mathrm{cA})_{i,j} = c \cdot A_{i,j}$

Тестирование программы

Написаны юниттесты для каждой функции класса с помощью стандартной библиотеки unittest. Также протестировано вручную.
Листинг кода теста см. в приложении Б-test.



Успешное прохождение unittests

```
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

1 2
3 4

Выберите из предложенных опций:
1: Ввести первую матрицу вручную.
2: Ввести вторую матрицу вручную.
3: Сложить две матрицы.
4: Умножить матрицу на матрицу.
5: Умножить матрицу на скаляр.
6: Выйти из программы.

Введите номер соответствующей опции: 2
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

6 5
7 9

Выберите из предложенных опций:
1: Ввести первую матрицу вручную.
2: Ввести вторую матрицу вручную.
3: Сложить две матрицы.
4: Умножить матрицу на матрицу.
5: Умножить матрицу на скаляр.
6: Выйти из программы.

Введите номер соответствующей опции: 3
Результат сложения матриц:
[7.0, 7.0]
[10.0, 13.0]
```

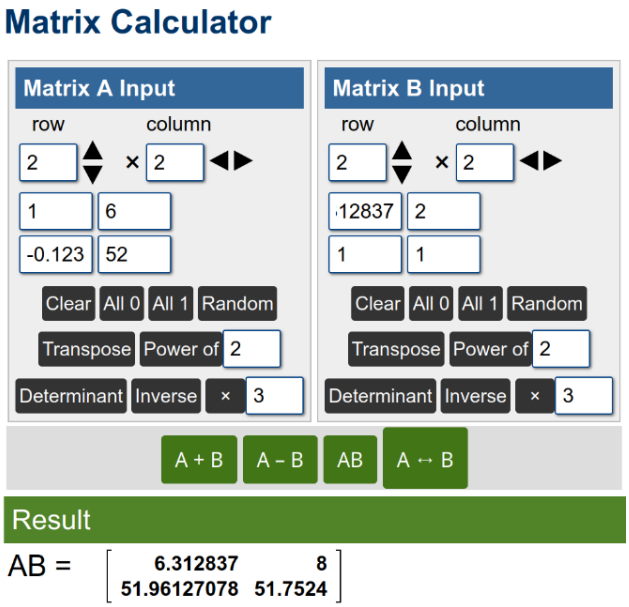
Ручной тест сложения двух матриц 2 на 2.

```
Введите номер соответствующей опции: 1
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

1 2
6 7

Выберите из предложенных опций:
1: Ввести первую матрицу вручную.
2: Ввести вторую матрицу вручную.
3: Сложить две матрицы.
4: Умножить матрицу на матрицу.
5: Умножить матрицу на скаляр.
6: Выйти из программы.

Введите номер соответствующей опции: 5
Выберите матрицу для умножения на скаляр (1 - первая матрица, 2 - вторая матрица): 1
Введите скаляр: 2.718281828459045
Результат умножения матрицы на скаляр:
[2.718281828459045, 5.43656365691809]
[16.30969097075427, 19.027972799213316]
```

Ручной тест умножения матрицы 2 на 2 на число Эйлера с небольшим количеством знаков после запятой.

```
Введите номер соответствующей опции: 1
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

1 6
-0.1238 52

Выберите из предложенных опций:
1: Ввести первую матрицу вручную.
2: Ввести вторую матрицу вручную.
3: Сложить две матрицы.
4: Умножить матрицу на матрицу.
5: Умножить матрицу на скаляр.
6: Выйти из программы.

Введите номер соответствующей опции: 2
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

0.312837 2
1 1

Выберите из предложенных опций:
1: Ввести первую матрицу вручную.
2: Ввести вторую матрицу вручную.
3: Сложить две матрицы.
4: Умножить матрицу на матрицу.
5: Умножить матрицу на скаляр.
6: Выйти из программы.

Введите номер соответствующей опции: 4
Результат умножения матриц:
[6.312837, 8.0]
[51.9612707794, 51.7524]
```

Ручной тест умножения двух страшных матриц 2 на 2.



Удостоверимся в правильности полученного результата, умножив матрицы на сайте-калькуляторе. Результат правильный.

Справились со вторым заданием.

Задача 3

В третьей задача требуется реализовать следующие функции: (полный листинг кода см. в приложении В)

- Вычисление определителя матрицы. По тз - матрицы размером до 100x100
- Ответ на вопрос: существует ли матрица, обратная данной.

Использованные библиотеки и инструменты языка

В ходе написания программы были использованы только стандартные средства языка. Также для удобства и лучшей читаемости кода была импортирована библиотека typing.

Были реализованы 3 функции:
1. determinantOfMatrix - поиск определителя матрицы.
2. isMatrixInvertable - функция, которая отвечает на вопрос, существует ли обратная матрица к данной.
3. gauss - вспомогательная функция - метод Гаусса, с помощью которого считал определитель любой квадратной матрицы.

Суть работы алгоритмов:

- determinantOfMatrix: оболочка функции, проверяем размер матрицы и обращаемся к функции Гаусса для поиска определителя.

- isMatrixInvertable: сравниваем полученный из первой функции определитель с нулем, возвращаем логическое значение "да" или "нет".

- gauss: самая интересная функция. Для матрицы 1x1 и 2x2 посчитано вручную.

```
    n = len(matrix)
    if n == 1:
      return matrix[0][0]
    elif n == 2:
      return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]
```

1x1: Определитель равен единственному элементу.
2x2: Определитель считаем как разность произведения элементов главной и побочной диагоналей.

Проверка на вырожденную матрицу:

```
    for i in range(n):
      max_row = max(range(i, n), key=lambda r: abs(matrix[r][i]))
      if matrix[max_row][i] == 0:
        raise ValueError("Matrix is singular,       determinant is zero.")
```

Как работает:

Цикл проходит по каждому столбцу i от 0 до $n-1$. На каждом шаге ищется ведущий элемент в текущем столбце. Для этого используется команда:

```
    max_row = max(range(i, n), key=lambda r: abs(matrix[r][i]))
```

Здесь range(i, n) генерирует индексы строк от текущей строки i до последней n-1, lambda r: abs(matrix[r][i]) вычисляет модуль элемента в текущем столбце для каждой строки r, а max(..., key=...) выбирает индекс строки max_row, где модуль элемента в столбце максимален.

Далее выполняется проверка:

```
    if matrix[max_row][i] == 0:
      raise ValueError("Matrix is singular, determinant is zero.")
```

Если максимальный элемент в столбце равен 0, то все элементы ниже i-й строки в этом столбце тоже равны 0. Это означает, что матрица вырожденная, и её определитель равен нулю, поэтому дальнейшие вычисления прекращаются.

Приведение к нижнетреугольному виду и вычисление определителя:

Для приведения матрицы к нижнетреугольному виду используется следующий алгоритм. Если строка с максимальным элементом в текущем столбце max_row не совпадает с текущей строкой i, строки меняются местами:

```
if max_row != i:
  matrix[i], matrix[max_row] = matrix[max_row], matrix[i]
  swap_count += 1
```

После этого элементы ниже ведущего в текущем столбце обнуляются. Для каждой строки j, начиная с i + 1, вычисляется коэффициент factor, с помощью которого корректируются элементы строки:

```
for j in range(i + 1, n):
  factor = matrix[j][i] / matrix[i][i]
  for k in range(i, n):
    matrix[j][k] -= factor * matrix[i][k]
```

После завершения приведения к нижнетреугольному виду вычисляется определитель матрицы. Определитель является произведением всех диагональных элементов приведённой матрицы:

```
determinant = 1.0
for i in range(n):
  determinant *= matrix[i][i]
```

Учитывается знак определителя, зависящий от числа перестановок строк:

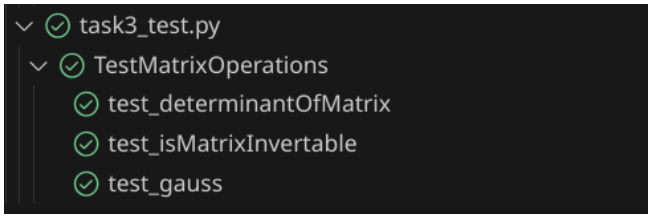
```
return (-1) ** swap_count * determinant
```

Здесь каждая перестановка строк меняет знак определителя, что учитывается выражением (-1) ** swap_count.

Таким образом реализован метод Гаусса для поиска определителя любой квадратной матрицы.

Тестирование программы

Написаны юниттесты для каждой функции класса с помощью стандартной библиотеки unittest. Также протестировано вручную.
Листинг кода теста см. в приложении B-test.



Успешное прохождение unittests

```
Введите номер соответствующей опции: 1
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

1 2
3 4

Выберите из предложенных опций:
1: Ввести матрицу вручную.
2: Вычислить определитель матрицы.
3: Проверить, существует ли обратная матрица.
4: Выйти из программы.

Введите номер соответствующей опции: 2
Определитель матрицы: -2.0
```

Вручную проверям подсчет определителя матрицы 2x2. Ответ верный.

```
Введите номер соответствующей опции: 1
Введите размер матрицы n x m через пробел: 2 2
Введите матрицу по строке, разделяя элементы строки пробелом

0 0
0 0

Выберите из предложенных опций:
1: Ввести матрицу вручную.
2: Вычислить определитель матрицы.
3: Проверить, существует ли обратная матрица.
4: Выйти из программы.

Введите номер соответствующей опции: 2
Определитель матрицы: 0.0


Выберите из предложенных опций:
1: Ввести матрицу вручную.
2: Вычислить определитель матрицы.
3: Проверить, существует ли обратная матрица.
4: Выйти из программы.

Введите номер соответствующей опции: 3
Матрица необратима.
```

Вручную проверяем подсчет определителя и обратимости нулевой матрицы. Ответ верный.

```
1 2 3
5 6 8

Выберите из предложенных опций:
1: Ввести матрицу вручную.
2: Вычислить определитель матрицы.
3: Проверить, существует ли обратная матрица.
4: Выйти из программы.

Введите номер соответствующей опции: 2
Определитель можно вычислить только для квадратной матрицы.
```

Конечно же, вручную пытаемся обработать не квадратную матрицу.

Итак, все задания сделаны, тесты пройдены. Пора перейти к выводам.

Вывод

В ходе выполнения данной работы были реализованы и протестированы различные функции для работы с матрицами. Это включало ввод матрицы, вычисление её следа, поиск элементов по индексам, а также операции сложения, умножения матриц и умножения матрицы на скаляр. Кроме того, были реализованы функции для вычисления определителя матрицы и проверки её обратимости.

Работа над этими задачами позволила нам углубить понимание основ линейной алгебры и её применения в программировании. Особенно интересным оказался метод Гаусса для вычисления определителя матрицы, который демонстрирует применение алгоритмов линейной алгебры в решении сложных задач.

Процесс тестирования, как автоматического, так и ручного, подтвердил корректность реализованных алгоритмов и их способность работать с матрицами различных размеров. Это важно для обеспечения надежности и точности вычислений, что является важным аспектом в машинном обучении и анализе данных.

В целом, выполнение данной работы не только укрепило наши навыки программирования и работы с матрицами, но и заставило задуматься, как полученные знания помогут в будущем. Линейная алгебра является основой для многих алгоритмов машинного обучения, и понимание её принципов и методов является ключом для успешного применения этих алгоритмов на практике.

В будущем мы планируем продолжить изучение машинного обучения, включая более сложные алгоритмы и методы. Уверены, что полученные знания и навыки станут основой для дальнейшего развития в этой области.

Приложения

Пришлось все перевести на английский. Не понял, как в "lstlisting" в LaTeX позволить писать русскими буквами без ошибок компиляции.

Приложение А.
Полный листинг кода к первому заданию

```
from typing import List, Optional

class MatrixKeeper:
  def __init__(self):
    self.matrix: Optional[List[List[float]]] = None

  def inputMatrix(self) -> None:
  """Input matrix
  First prompt - input matrix size n X m
  Second prompt - input the matrix row by row, elements separated by space"""

    try:
      n, m = map(int, input("Enter the matrix size n x m separated by space: ").split())
      self.matrix = []
      print("Enter the matrix row by row, separating elements with space\n")
      for _ in range(n):
        row = list(map(float, input().split()))
        if len(row) != m:
          raise ValueError("The number of elements in the row does not match m.")
        self.matrix.append(row)
    except ValueError as e:
      print(f"Input error: {e}")
      self.matrix = None

  def trace(self) -> float:
  """Find the trace of the matrix"""

    if self.matrix is None:
      raise ValueError("Matrix has not been entered.")

    if len(self.matrix) != len(self.matrix[0]):
      raise ValueError("The matrix must be square to calculate the trace.")

  return sum(self.matrix[i][i] for i in range(len(self.matrix)))

  def findByIndex(self, n: int, m: int) -> float:
    """Finds an element in the matrix by index n, m and outputs it"""

    if self.matrix is None:
      raise ValueError("Matrix has not been entered.")

    if n <= 0 or m <= 0 or n > len(self.matrix) or m > len(self.matrix[0]):
      raise IndexError("Indexes are out of the matrix bounds.")

    return self.matrix[n-1][m-1]

def main():
```

```
    matrix_keeper = MatrixKeeper()

    while True:
      print("\nChoose from the following options:")
      print("1: Enter the matrix manually.")
      print("2: Find the trace of the matrix.")
      print("3: Find an element by index in the given matrix.")
      print("4: Exit the program.\n")

      try:
        option = int(input("Enter the number of the corresponding option: "))
        except ValueError:
        print("Incorrect input. Please enter a number.\n")
        continue

        if option == 1:
          matrix_keeper.inputMatrix()
        elif option == 2:
        try:
          trace_value = matrix_keeper.trace()
          print(f"Trace of the matrix: {trace_value}\n")
        except ValueError as e:
          print(e)
        elif option == 3:
          try:
            n = int(input("Enter the row number: "))
            m = int(input("Enter the column number: "))
            element = matrix_keeper.findByIndex(n, m)
            print(f"Matrix element [{n}][{m}] = {element}\n")
          except (ValueError, IndexError) as e:
          print(e)
        elif option == 4:
          print("Exiting the program.\n")
          break
        else:
          print("Incorrect input. Please choose an option from 1 to 4.\n")

if __name__ == "__main__":
main()
```

Приложение A-test.

Полный листинг кода к тестам первого задания

```
import unittest

from codefiles.task1 import MatrixKeeper

class testMatrixKeeper(unittest.TestCase):

  def setUp(self):
    self.keeper = MatrixKeeper()

  def test_inputMatrix(self):

    self.keeper.matrix = [
    [1.32, 2.32, 3.45],
    [2.1, 4.312, 4.24],
    [3.1, 1.12, 9.125]
    ]

```

```
18        self.assertEqual(self.keeper.matrix, [
19        [1.32, 2.32, 3.45],
20        [2.1, 4.312, 4.24],
21        [3.1, 1.12, 9.125]
22        ])
23
24
25    def test_trace(self):
26
27        self.keeper.matrix = [
28        [1.32, 2.32, 3.45],
29        [2.1, 4.312, 4.24],
30        [3.1, 1.12, 9.125]
31        ]
32
33        self.assertEqual(self.keeper.trace(), 14.757)
34
35    def test_findByIndex(self):
36
37        self.keeper.matrix = [
38        [1.0, 2.0, 3.0],
39        [4.0, 5.0, 6.0],
40        [7.0, 8.0, 9.0]
41        ]
42        self.assertEqual(self.keeper.findByIndex(1, 1), 1.0)
43        self.assertEqual(self.keeper.findByIndex(2, 2), 5.0)
44        self.assertEqual(self.keeper.findByIndex(3, 3), 9.0)
45
46    def test_trace_not_square(self):
47
48        self.keeper.matrix = [
49        [1.0, 2.0, 3.0],
50        [4.0, 5.0, 6.0]
51        ]
52        with self.assertRaises(ValueError):
53        self.keeper.trace()
54
55    def test_findByIndex_out_of_bounds(self):
56
57        self.keeper.matrix = [
58        [1.0, 2.0, 3.0],
59        [4.0, 5.0, 6.0],
60        [7.0, 8.0, 9.0]
61        ]
62        with self.assertRaises(IndexError):
63        self.keeper.findByIndex(4, 4)
```

Приложение Б.

Полный листинг кода ко второму заданию

```
1  from typing import List, Optional
2  from codefiles.task1 import MatrixKeeper
3
4  def matrixAddition(matrix_keeper1: MatrixKeeper, matrix_keeper2: MatrixKeeper)
       -> Optional[List[List[float]]]:
5  """Addition of two matrices"""
6
7    if matrix_keeper1.matrix is None or matrix_keeper2.matrix is None:
8      raise ValueError("One or both matrices have not been entered.")
9
```

```
10      if len(matrix_keeper1.matrix) != len(matrix_keeper2.matrix) or len(
            matrix_keeper1.matrix[0]) != len(matrix_keeper2.matrix[0]):
11          raise ValueError("Matrices must be of the same size for addition.")
12
13      result = []
14      for i in range(len(matrix_keeper1.matrix)):
15          row = []
16          for j in range(len(matrix_keeper1.matrix[0])):
17              row.append(matrix_keeper1.matrix[i][j] + matrix_keeper2.matrix[i][j])
18              result.append(row)
19
20      return result
21
22  def matrixByMatrixMultiplication(matrix_keeper1: MatrixKeeper, matrix_keeper2:
        MatrixKeeper) -> Optional[List[List[float]]]:
23  """Matrix by matrix multiplication"""
24
25      if matrix_keeper1.matrix is None or matrix_keeper2.matrix is None:
26          raise ValueError("One or both matrices have not been entered.")
27
28      if len(matrix_keeper1.matrix[0]) != len(matrix_keeper2.matrix):
29          raise ValueError("The number of columns in the first matrix must be equal
                to the number of rows in the second matrix.")
30
31      result = [[0 for _ in range(len(matrix_keeper2.matrix[0]))] for _ in range(
            len(matrix_keeper1.matrix))]
32      for i in range(len(matrix_keeper1.matrix)):
33          for j in range(len(matrix_keeper2.matrix[0])):
34              for k in range(len(matrix_keeper2.matrix)):
35                  result[i][j] += matrix_keeper1.matrix[i][k] * matrix_keeper2.matrix[k
                        ][j]
36
37      return result
38
39  def matrixScalarMultiplication(matrix_keeper: MatrixKeeper, scalar: float) ->
        Optional[List[List[float]]]:
40  """Matrix by scalar multiplication"""
41
42      if matrix_keeper.matrix is None:
43          raise ValueError("Matrix has not been entered.")
44          result = []
45      for row in matrix_keeper.matrix:
46          result.append([element * scalar for element in row])
47
48      return result
49
50  def main():
51      matrix_keeper1 = MatrixKeeper()
52      matrix_keeper2 = MatrixKeeper()
53
54      while True:
55          print("\nChoose from the following options:")
56          print("1: Enter the first matrix manually.")
57          print("2: Enter the second matrix manually.")
58          print("3: Add two matrices.")
59          print("4: Multiply matrix by matrix.")
60          print("5: Multiply matrix by scalar.")
61          print("6: Exit the program.\n")
62
63          try:
```

```
64          option = int(input("Enter the number of the corresponding option: "))
65          except ValueError:
66          print("Incorrect input. Please enter a number.\n")
67          continue
68
69          if option == 1:
70            matrix_keeper1.inputMatrix()
71          elif option == 2:
72            matrix_keeper2.inputMatrix()
73          elif option == 3:
74            try:
75            result = matrixAddition(matrix_keeper1, matrix_keeper2)
76            print("Result of matrix addition:")
77            for row in result:
78            print(row)
79            except ValueError as e:
80            print(e)
81          elif option == 4:
82            try:
83              result = matrixByMatrixMultiplication(matrix_keeper1, matrix_keeper2
                  )
84              print("Result of matrix multiplication:")
85              for row in result:
86              print(row)
87              except ValueError as e:
88              print(e)
89          elif option == 5:
90            try:
91              matrix_choice = int(input("Choose the matrix to multiply by scalar
                    (1 - first matrix, 2 - second matrix): "))
92              if matrix_choice == 1:
93                matrix_keeper = matrix_keeper1
94              elif matrix_choice == 2:
95                matrix_keeper = matrix_keeper2
96              else:
97                raise ValueError("Incorrect matrix choice.")
98
99              scalar = float(input("Enter the scalar: "))
100             result = matrixScalarMultiplication(matrix_keeper, scalar)
101             print("Result of matrix by scalar multiplication:")
102             for row in result:
103               print(row)
104           except ValueError as e:
105             print(e)
106         elif option == 6:
107           print("Exiting the program.\n")
108           break
109         else:
110           print("Incorrect input. Please choose an option from 1 to 6.\n")
111
112   if __name__ == "__main__":
113   main()
```

Приложение Б-test.

Полный листинг кода к тестам второго задания

```
1 import unittest
2 from codefiles.task2 import matrixAddition, matrixByMatrixMultiplication,
      matrixScalarMultiplication
3 from codefiles.task1 import MatrixKeeper
```

```
4
5 class TestMatrixOperations(unittest.TestCase):
6
7   def setUp(self):
8
9     self.matrix_keeper1 = MatrixKeeper()
10    self.matrix_keeper2 = MatrixKeeper()
11
12    self.matrix_keeper1.matrix = [
13    [1, 2, 3],
14    [4, 5, 6],
15    [7, 8, 9]
16    ]
17
18    self.matrix_keeper2.matrix = [
19    [9, 8, 7],
20    [6, 5, 4],
21    [3, 2, 1]
22    ]
23
24  def test_matrixAddition(self):
25    result = matrixAddition(self.matrix_keeper1, self.matrix_keeper2)
26    expected = [
27    [10, 10, 10],
28    [10, 10, 10],
29    [10, 10, 10]
30    ]
31  self.assertEqual(result, expected)
32
33  def test_matrixByMatrixMultiplication(self):
34
35    self.matrix_keeper1.matrix = [
36    [1, 2, 3],
37    [4, 5, 6],
38    [7, 8, 9]
39    ]
40
41    self.matrix_keeper2.matrix = [
42    [9, 8],
43    [7, 6],
44    [5, 4]
45    ]
46
47    result = matrixByMatrixMultiplication(self.matrix_keeper1, self.
        matrix_keeper2)
48    expected = [
49    [38, 32],
50    [101, 86],
51    [164, 140]
52    ]
53    self.assertEqual(result, expected)
54
55  def test_matrixScalarMultiplication(self):
56    scalar = 2
57    result = matrixScalarMultiplication(self.matrix_keeper1, scalar)
58    expected = [
59    [2, 4, 6],
60    [8, 10, 12],
61    [14, 16, 18]
62    ]
```

```
63      self.assertEqual(result, expected)
64
65  if __name__ == '__main__':
66  unittest.main()
```

Приложение В.
Полный листинг кода к третьему заданию

```
1   from task1 import MatrixKeeper
2   from typing import List
3
4   def determinantOfMatrix(matrix_keeper: MatrixKeeper) -> float:
5   """Calculates the determinant of a matrix. Matrix size: up to 100x100."""
6     if matrix_keeper.matrix is None:
7       raise ValueError("Matrix is not defined.")
8
9     matrix = matrix_keeper.matrix
10    if len(matrix) != len(matrix[0]):
11      raise ValueError("Determinant can only be calculated for a square matrix.")
12
13    return gauss(matrix)
14
15  def isMatrixInvertable(matrix_keeper: MatrixKeeper) -> bool:
16  """Checks if the inverse matrix exists (detA != 0)."""
17    try:
18      det = determinantOfMatrix(matrix_keeper)
19      return det != 0
20    except ValueError:
21     return False
22
23  def gauss(matrix: List[List[float]]) -> float:
24  """Calculates the determinant of a matrix using the Gauss method."""
25    n = len(matrix)
26    if n == 1:
27     return matrix[0][0]  # For a 1x1 matrix
28    elif n == 2:
29     return matrix[0][0] * matrix[1][1] - matrix[0][1] * matrix[1][0]  # For a 2
          x2 matrix
30
31    swap_count = 0
32
33    for i in range(n):
34      max_row = max(range(i, n), key=lambda r: abs(matrix[r][i]))
35      if matrix[max_row][i] == 0:
36        raise ValueError("The matrix is degenerate, the determinant is zero.")
37
38      if max_row != i:
39        matrix[i], matrix[max_row] = matrix[max_row], matrix[i]
40       swap_count += 1
41
42      for j in range(i + 1, n):
43        factor = matrix[j][i] / matrix[i][i]
44         for k in range(i, n):
45           matrix[j][k] -= factor * matrix[i][k]
46
47    determinant = 1.0
48    for i in range(n):
49    determinant *= matrix[i][i]
50
51    return (-1) ** swap_count * determinant
```

```
52
53 def main():
54   matrix_keeper = MatrixKeeper()
55
56   while True:
57     print("\nChoose from the following options:")
58     print("1: Enter the matrix manually.")
59     print("2: Calculate the determinant of the matrix.")
60     print("3: Check if the inverse matrix exists.")
61     print("4: Exit the program.\n")
62
63     try:
64       option = int(input("Enter the number of the corresponding option: "))
65    except ValueError:
66      print("Incorrect input. Please enter a number.\n")
67       continue
68
69       if option == 1:
70         matrix_keeper.inputMatrix()
71       elif option == 2:
72         try:
73           det = determinantOfMatrix(matrix_keeper)
74           print(f"Determinant of the matrix: {det}\n")
75         except ValueError as e:
76         print(e)
77       elif option == 3:
78       try:
79         is_invertable = isMatrixInvertable(matrix_keeper)
80         if is_invertable:
81           print("Yes.\n")
82         else:
83         print("No.\n")
84       except ValueError as e:
85         print(e)
86       elif option == 4:
87         print("Exiting the program.\n")
88         break
89       else:
90       print("Incorrect input. Please choose an option from 1 to 4.\n")
91
92 if __name__ == "__main__":
93 main()
```

Приложение B-test.
Полный листинг кода к тестам третьего задания

```
1 import unittest
2 from codefiles.task3 import MatrixKeeper, determinantOfMatrix,
    isMatrixInvertable, gauss
3
4 class TestMatrixOperations(unittest.TestCase):
5
6   def setUp(self):
7     self.matrix_keeper = MatrixKeeper()
8
9   def test_determinantOfMatrix(self):
10
11     self.matrix_keeper.matrix = [
12     [1, 2, 3],
13     [0, 5, 6],
```

```
14      [7, 8, 9]
15      ]
16      result = determinantOfMatrix(self.matrix_keeper)
17      expected = -24
18      self.assertAlmostEqual(result, expected)
19
20      self.matrix_keeper.matrix = [
21      [4, 7],
22      [2, 6]
23      ]
24      result = determinantOfMatrix(self.matrix_keeper)
25      expected = 10
26      self.assertAlmostEqual(result, expected)
27
28      self.matrix_keeper.matrix = [[1 if i == j else 0 for j in range(100)] for i
            in range(100)]
29      result = determinantOfMatrix(self.matrix_keeper)
30      expected = 1
31      self.assertAlmostEqual(result, expected)
32
33  def test_isMatrixInvertable(self):
34      self.matrix_keeper.matrix = [
35      [1, 2, 3],
36      [0, 5, 6],
37      [7, 8, 9]
38      ]
39      result = isMatrixInvertable(self.matrix_keeper)
40      self.assertTrue(result)
41
42      self.matrix_keeper.matrix = [
43      [1, 2, 3],
44      [4, 6, 8],
45      [7, 10, 12]
46      ]
47      result = isMatrixInvertable(self.matrix_keeper)
48      self.assertTrue(result)
49
50  def test_gauss(self):
51      matrix = [
52      [1, 2, 3],
53      [0, 5, 6],
54      [7, 8, 9]
55      ]
56      result = gauss(matrix)
57      expected = -24
58      self.assertAlmostEqual(result, expected)
59
60      matrix = [
61      [4, 7],
62      [2, 6]
63      ]
64      result = gauss(matrix)
65      expected = 10
66      self.assertAlmostEqual(result, expected)
67
68      matrix = [
69      [5]
70      ]
71      result = gauss(matrix)
72      expected = 5
```

```
73      self.assertAlmostEqual(result, expected)
74
75      matrix = [[1 if i == j else 0 for j in range(100)] for i in range(100)]
76      result = gauss(matrix)
77      expected = 1
78      self.assertAlmostEqual(result, expected)
79
80      matrix = [
81      [1, 2],
82      [2, 4]
83      ]
84      result = gauss(matrix)
85      expected = 0
86      self.assertAlmostEqual(result, expected)
87
88  if __name__ == '__main__':
89  unittest.main()
```

Список использованной литературы

- Курош, А. Г. Курс высшей алгебры. Москва: Наука, 1977.
- Гельфанд, И. М. Лекции по линейной алгебре. Москва: Наука, 1971.
- Стрэнг, Г. Линейная алгебра и ее приложения. Москва: Мир, 1986.
- Википедия. Определитель матрицы. Доступно на: https://ru.wikipedia.org/wiki/
- MathWorld. Determinant. Доступно на: https://mathworld.wolfram.com/Determina
- Khan Academy. Introduction to the determinant. Доступно на: https://www.khanac algebra/matrix-transformations/determinant-depth/v/linear-algebra-introduction-to-the-determinant
- GeeksforGeeks. Determinant of a Matrix. Доступно на: https://www.geeksforgeeks.c of-a-matrix/