МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра информационных систем

ОТЧЕТ

по практической работе №3

по дисциплине «Объектно-ориентированное программирование»

Студент гр. 0362 _____ Мирошников Н.Ю.

_____ Константинов И.П.

Преподаватель _____ Егоров С.С.

Санкт-Петербург
2023

1. Задание для лабораторной работы

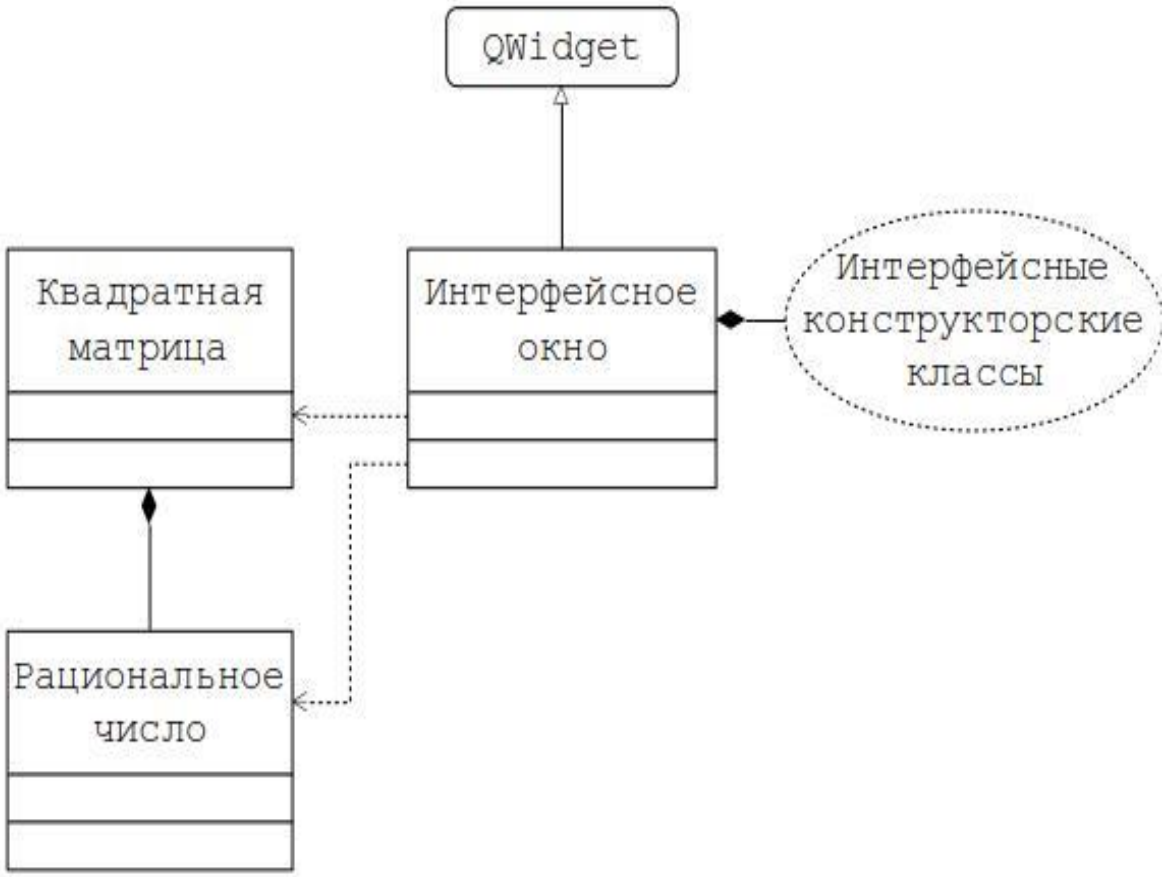

Рисунок 1 — Диаграмма классов работы №2

Создать GUI приложение, реализующее функции перечисленные в описании работы №1, но на множестве рациональных чисел. Для этого требуется разработать и реализовать класс рациональных чисел.

Рациональное число — это несократимая дробь a/b, где a и b — целые, причем b>0.

Приложение должно включать основной модуль, модуль «interface», модуль «matrix», модуль «rational» и файл number.h.

При необходимости расширения функциональности класса «Квадратная матрица» следует только дополнить его протокол без каких-либо изменений уже существовавшей реализации.

Реализовать и отладить программу, удовлетворяющую сформулированным требованиям и заявленным целям. Разработать контрольные примеры и оттестировать на них программу. Оформить отчет, сделать выводы по работе.

2. Спецификации классов (понятий или предметов), которые требуется разработать.

1. Класс «TRational»:

Атрибуты:

Наименование	Тип	Область видимости
divisible	int	private
divisor	int	private
text_representation	string	public

Атрибуты divisible являются объектами класса int и используются для хранения делимого и делителя рационального числа.

Атрибут text_representation является объектом класса string и используется для хранения текстовой записи рационального числа.

Методы:

Метод «__reduction(self)» ничего не возвращает (тип «void»), находится в области видимости «private». Метод предназначен для сокращения рационального числа.

Метод «__bringing(self, other, lcm)» возвращает массив из двух элементов типа int, находится в области видимости «private». Метод предназначен для приведения двух рациональных чисел к одному знаменателю.

Метод «gcd_func(self, x, y)» возвращает значение типа int либо саму себя, находится в области видимости «public». Метод предназначен для нахождения НОД двух значений типа int.

Метод «lcm_func(self, x, y)» возвращает значение типа int, находится в области видимости «public». Метод предназначен для нахождения НОК двух значений типа int.

Метод «__init__(self)» ничего не возвращает (тип «void»), находится в области видимости «private». Метод инициализирует атрибуты divisible, divisor и text_representation.

Метод «__add__(self, other)» возвращает объект типа TComplex, находится в области видимости «private». Метод вычисляет сумму двух рациональных чисел.

Метод «__sub__ (self, other)» возвращает объект типа TComplex, находится в области видимости «private». Метод вычисляет разность двух рациональных чисел.

Метод «__mul__(self, other)» возвращает объект типа TComplex, находится в области видимости «private». Метод вычисляет произведение двух рациональных чисел или рационального числа и целого числа.

Метод «__truediv__(self, other)» возвращает объект типа TComplex, находится в области видимости «private». Метод вычисляет частное двух рациональных чисел или рационального числа и целого числа.

2. Класс «TInterface»:

Атрибуты:

Наименование	Тип	Область видимости
window	QMainWindow	public
ui	GreetingsWindow	public

Атрибут window является объектом класса QMainWindow и используется для создания основного окна пользовательского интерфейса.

Атрибут ui является объектом класса GreetingsWindow и используется для установки пользовательского интерфейса.

Методы:

Метод «__init__(self)» ничего не возвращает (тип «void»), находится в области видимости «private». Метод инициализирует атрибуты window и ui, устанавливает пользовательский интерфейс.

Метод «show(self)» ничего не возвращает (тип «void»), находится в области видимости «public». Метод выводит на экран пользовательский интерфейс.

3. Диаграмма классов, дополненную атрибутами и методами.

На рисунке 2 представлена диаграмма классов, дополненная атрибутами и методами.

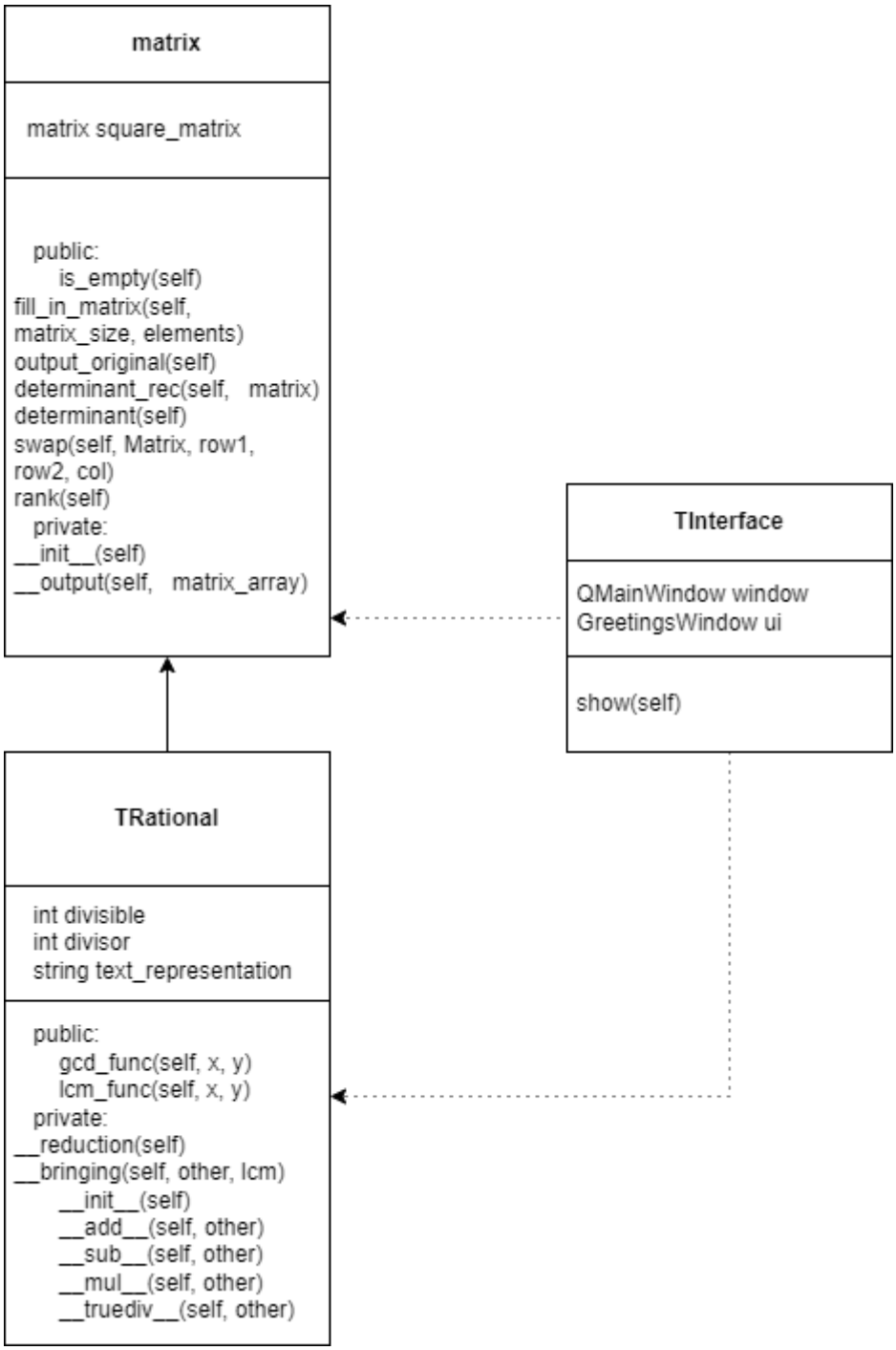

Рисунок 2 — Диаграмма классов, дополненная атрибутами и методами

4. Описание контрольного примера с исходными и ожидаемыми (расчетными) данными.

Пример:

$$A = \begin{pmatrix} \frac{1}{1} & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{pmatrix}$$

$$\det(A) = \left(\frac{1}{1}\right)\left(\frac{1}{5}\right)\left(\frac{1}{9}\right) + \left(\frac{1}{2}\right)\left(\frac{1}{6}\right)\left(\frac{1}{7}\right) + \left(\frac{1}{3}\right)\left(\frac{1}{8}\right)\left(\frac{1}{4}\right) - \\ - \left(\frac{1}{8}\right)\left(\frac{1}{6}\right)\left(\frac{1}{1}\right) - \left(\frac{1}{9}\right)\left(\frac{1}{4}\right)\left(\frac{1}{2}\right) = \frac{1}{3360};$$

$$A = \begin{pmatrix} \frac{1}{1} & \frac{1}{2} & \frac{1}{3} \\ \frac{1}{4} & \frac{1}{5} & \frac{1}{6} \\ \frac{1}{7} & \frac{1}{8} & \frac{1}{9} \end{pmatrix} \sim \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$rank(A) = 3$;

$$A^T = \begin{pmatrix} \frac{1}{1} & \frac{1}{4} & \frac{1}{7} \\ \frac{1}{2} & \frac{1}{5} & \frac{1}{8} \\ \frac{1}{3} & \frac{1}{6} & \frac{1}{9} \end{pmatrix}.$$

5. Скриншоты программы на контрольных примерах.

На рисунке 3 запустилась программа и на экране появилось меню.

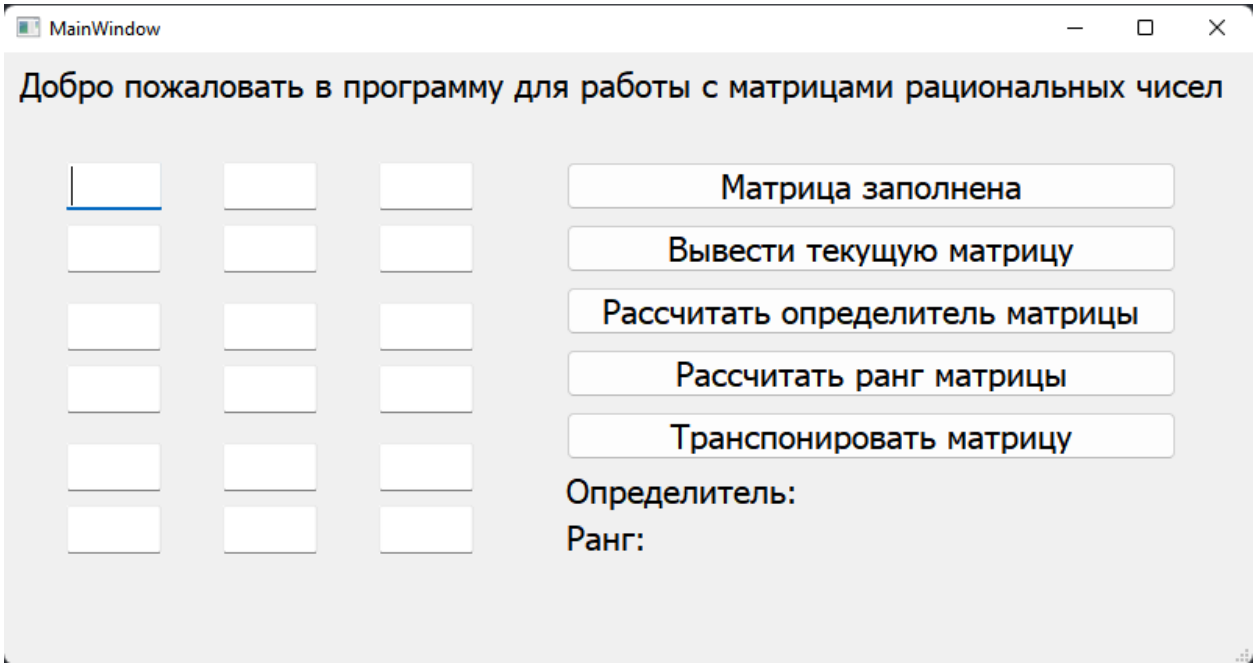

Рисунок 3 — Меню

На рисунке 4 был произведён ввод значений в матрицу и нажата кнопка “Матрица заполнена”, после чего программа зафиксировала введённые значения.

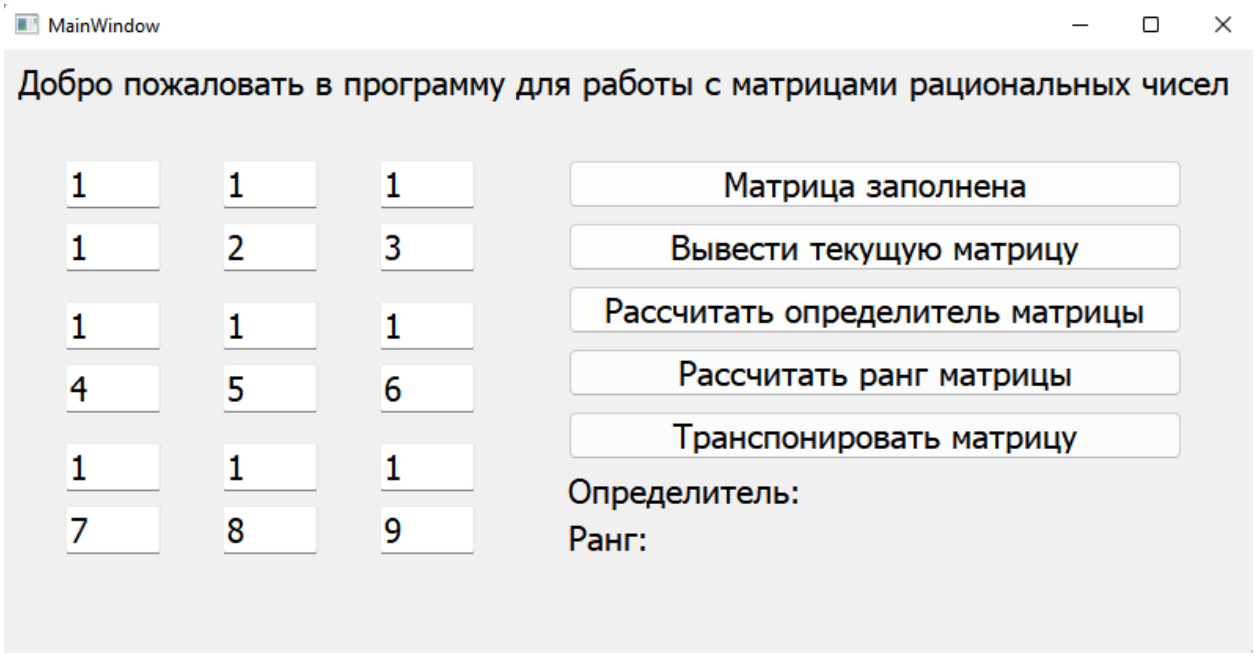

Рисунок 4 — Ввод значений матрицы

На рисунке 5 была нажата кнопка “Вывести текущую матрицу”.

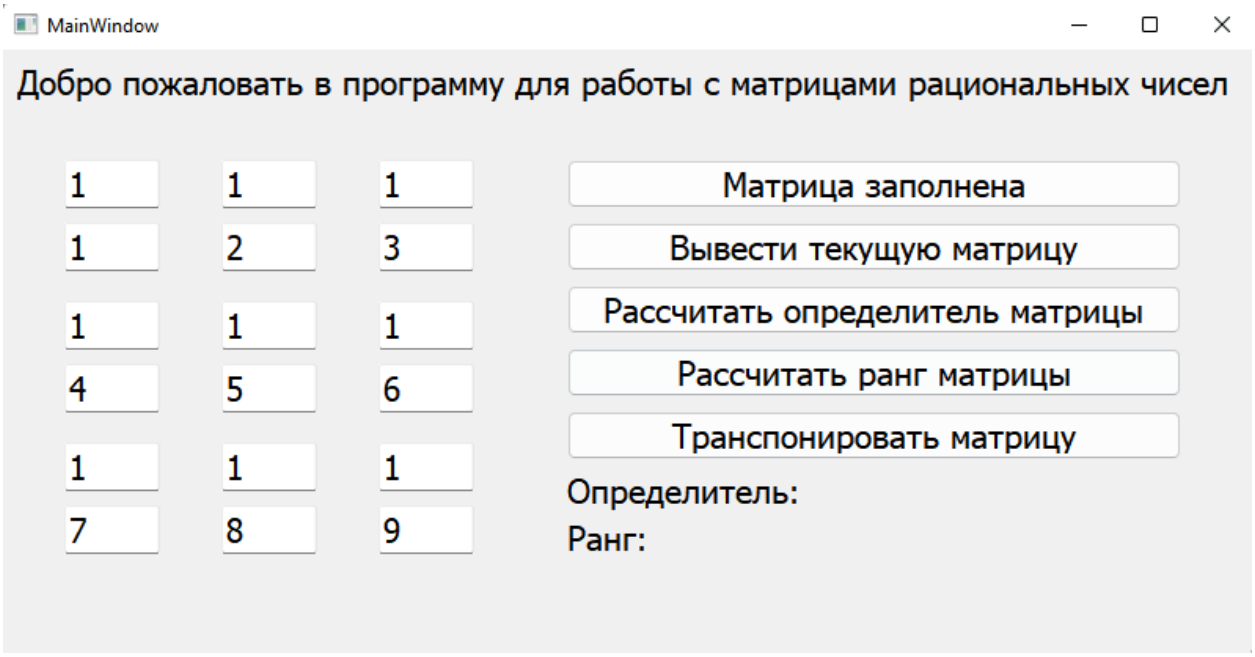

Рисунок 5 — Вывод текущей матрицы

На рисунке 6 была нажата кнопка “Рассчитать определитель матрицы” и выведен результат расчётов.

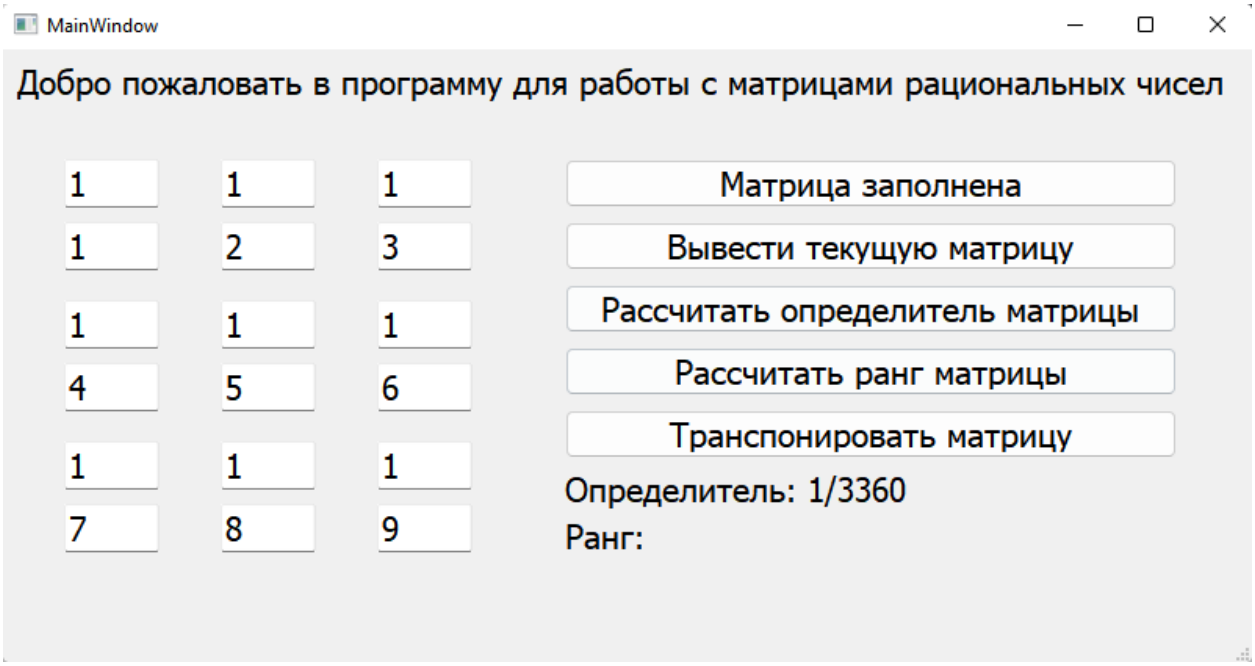

Рисунок 6 — Вывод результата расчётов определителя матрицы

На рисунке 7 была нажата кнопка “Рассчитать ранг матрицы” и выведен результат расчётов.

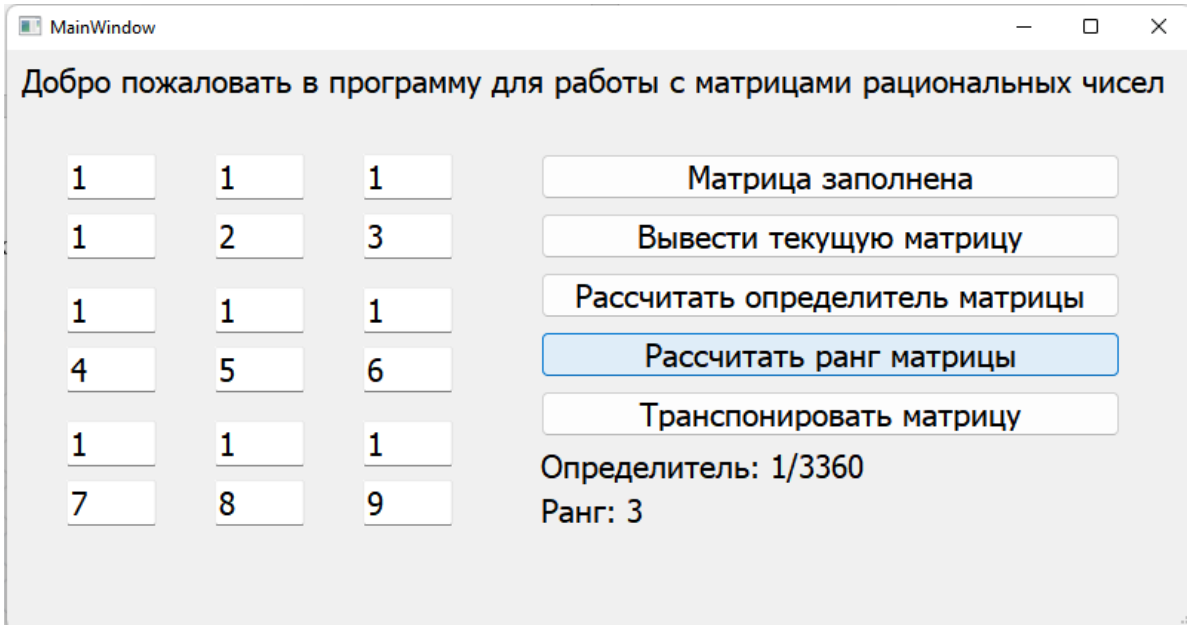

Рисунок 7 — Вывод результата расчёта ранга матрицы

На рисунке 8 была нажата кнопка “Транспонировать матрицу” и выведен результат транспонирования.

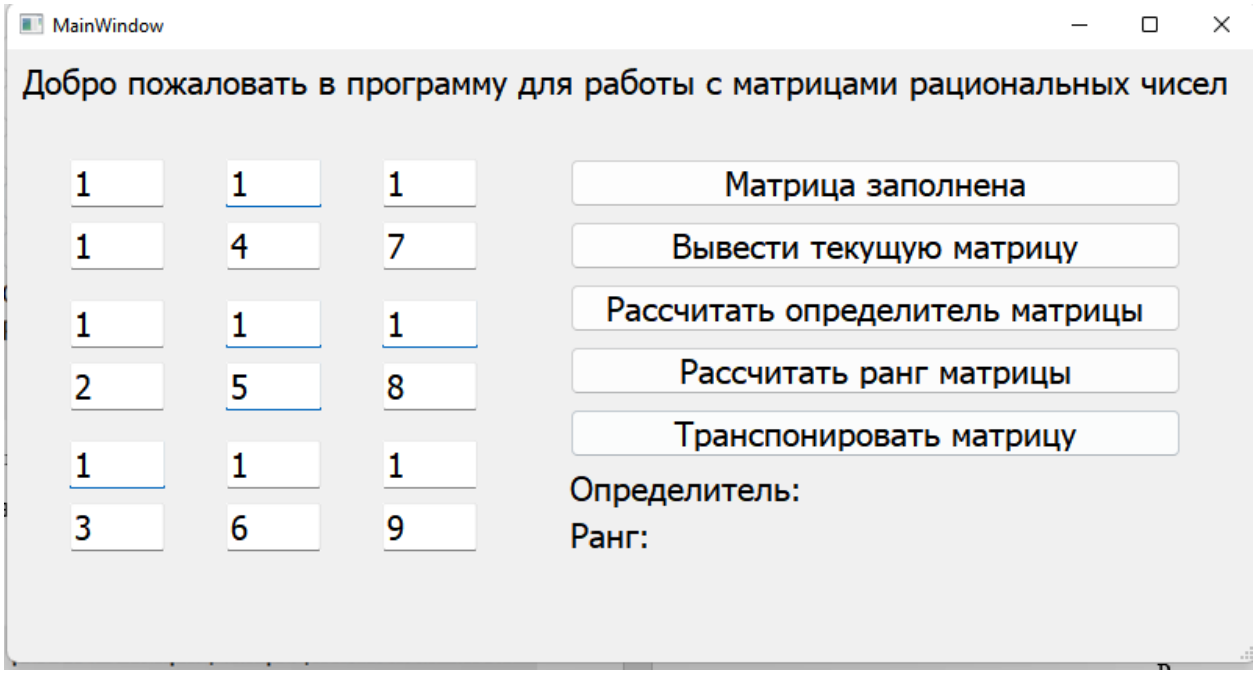

Рисунок 8 — Вывод матрицы

6. Выводы по выполненной работе.

В рамках данной практической работы был реализован пользовательские классы TRational и TInterface, имплементирующие работу с рациональными числами и пользовательский интерфейс соответственно. Была реализована и отлажена программа, удовлетворяющая сформулированным требованиям и заявленным целям. Были разработаны контрольные примеры, и программа была оттестирована на них.

Приложение 1. Исходный код.

Файл main.py:

```
from PyQt5.QtWidgets import QApplication
from interface import TInterface

import sys

def main():
    application = QApplication(sys.argv)
    interface = TInterface()
    interface.show()
    sys.exit(application.exec_())



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

Файл matrix.py:

```
from copy import copy
from rational import TRational

class matrix():

    def __init__(self):
        self.__matrix_array = []

    def is_empty(self):
        return not bool(len(self.__matrix_array))

    def get_matrix_array(self):
        return self.__matrix_array

    def fill_in_matrix(self, matrix_size, elements):
        self.__matrix_array = [[] for i in range(matrix_size)]
        element_id = 0
```

```
        for i in range(matrix_size):
            while element_id < matrix_size * (i + 1):

self.__matrix_array[i].append(elements[element_id])
                element_id += 1
            element_id = (i + 1) * matrix_size

    def __output(self, matrix_array):
        matrix_text =
'\n'.join(['.join(['{:12}'.format(item.value.text_representatio
n) for item in row]) for row in matrix_array])
        print(matrix_text)

    def output_original(self):
        matrix_text =
'\n'.join(['.join(['{:12}'.format(item.value.text_representatio
n) for item in row]) for row in self.__matrix_array])
        print(matrix_text)

    def determinant_rec(self, matrix):
        if isinstance(matrix[0][0].value, float):
            det = 0
        elif isinstance(matrix[0][0].value, TRational):
            det = TRational(0, 1)
        match ln := len(matrix):
            case 1:
                return matrix[0][0].value
            case 2:
                return matrix[0][0].value * matrix[1][1].value -
matrix[0][1].value * matrix[1][0].value
            case _:
                if isinstance(matrix[0][0].value, float):
                    for k in range(ln):
                        det += matrix[0][k].value * (-1.0) **
float(k) * self.determinant_rec([matrix[i][:k] + matrix[i][k+1:]
for i in range(1,ln)])
```

```
                elif isinstance(matrix[0][0].value, TRational):
                    for k in range(ln):
                        det += matrix[0][k].value * (-1) ** k *
self.determinant_rec([matrix[i][:k] + matrix[i][k+1:] for i in
range(1,ln)])

        return det

    def determinant(self):
        matrix = copy(self.__matrix_array)
        det = self.determinant_rec(matrix)
        if isinstance(matrix[0][0].value, float):
            return str(det)
        elif isinstance(matrix[0][0].value, TRational):
            return det.text_representation

    def transpose(self):
        transposed_matrix = [[] for i in
range(len(self.__matrix_array))]
        for i in range(len(self.__matrix_array)):
            for j in range(len(self.__matrix_array)):

transposed_matrix[i].append(self.__matrix_array[j][i])

        return transposed_matrix

    def swap(self, Matrix, row1, row2, col):
        for i in range(col):
            temp = Matrix[row1][i].value
            Matrix[row1][i].value = Matrix[row2][i].value
            Matrix[row2][i].value = temp

    def rank(self):
        matrix = copy(self.__matrix_array)
        size = len(matrix)
        rank = copy(size)
```

```
        for row in range(0, rank, 1):
            if matrix[row][row].value != 0:
                for col in range(0, size, 1):
                    if col != row:
                        multiplier = (matrix[col][row].value /
                                      matrix[row][row].value)
                        for i in range(rank):
                            matrix[col][i].value -= (multiplier
*

matrix[row][i].value)
            else:
                reduce = True
                for i in range(row + 1, size, 1):
                    if matrix[i][row].value != 0:
                        self.swap(matrix, row, i, rank)
                        reduce = False
                        break
                if reduce:
                    rank -= 1
                    for i in range(0, size, 1):
                        matrix[i][row].value =
matrix[i][rank].value
                row -= 1

        return str(rank)
```

Файл number.py:

```
from complex import TComplex

class number():

    def __init__(self, divisible, divisor):
        self.value = TRational(divisible, divisor)
```

Файл rational.py:

```
class TRational():

    def __reduction(self):
        gcd = self.gcd_func(self.__divisible, self.__divisor)
        self.__divisible /= gcd
        self.__divisor /= gcd
        self.__divisible = int(self.__divisible)
        self.__divisor = int(self.__divisor)
        self.text_representation                                    =
f'{self.__divisible}/{self.__divisor}'

    def __init__(self, divisible, divisor):
        self.__divisible = divisible
        self.__divisor = divisor
        self.text_representation                                    =
f'{self.__divisible}/{self.__divisor}'
        if self.__divisible != 0 and self.__divisor != 0:
            self.__reduction()

    def gcd_func(self, x, y):
        if(y == 0):
            return x
        else:
            return self.gcd_func(y, x % y)

    def lcm_func(self, x, y):
        if x > y:
            greater = x
        else:
            greater = y
        while(True):
            if((greater % x == 0) and(greater % y == 0)):
                lcm = greater
                break
            greater += 1
        return lcm

    def __bringing(self, other, lcm):
        int_part = int(lcm / self.__divisor)
        self.__divisor = int_part * self.__divisor
        self.__divisible = int_part * self.__divisible
        self.text_representation                                    =
f'{self.__divisible}/{self.__divisor}'
        int_part = int(lcm / other.__divisor)
        divisor = other.__divisor * int_part
        divisible = other.__divisible * int_part
        return [divisible, divisor]
```

```
    def __add__(self, other):
        if isinstance(other, int):
            result = TRational(self.__divisible + self.__divisor
* other, self.__divisor)
            result.__reduction()
            return result
        elif isinstance(other, TRational):
            if self.__divisor == other.__divisor:
                result     =     TRational(self.__divisible     +
other.__divisible, self.__divisor)
                result.__reduction()
                return result
            else:
                lcm          =     self.lcm_func(self.__divisor,
other.__divisor)
                new = self.__bringing(other, lcm)
                result  =  TRational(self.__divisible  +  new[0],
lcm)
                result.__reduction()
                return result

    def __sub__(self, other):
        if isinstance(other, int):
            result = TRational(self.__divisible - self.__divisor
* other, self.__divisor)
            result.__reduction()
            return result
        elif isinstance(other, TRational):
            if self.__divisor == other.__divisor:
                result     =     TRational(self.__divisible     -
other.__divisible, self.__divisor)
                result.__reduction()
                return result
            else:
                lcm        =        self.lcm_func(self.__divisor,
other.__divisor)
                new = self.__bringing(other, lcm)
                result  =  TRational(self.__divisible  -  new[0],
lcm)
                result.__reduction()
                return result

    def __mul__(self, other):
        if isinstance(other, int):
            result   =   TRational(self.__divisible   *   other,
self.__divisor)
            result.__reduction()
            return result
        elif isinstance(other, TRational):
            result      =      TRational(self.__divisible      *
other.__divisible, self.__divisor * other.__divisor)
            result.__reduction()
            return result
```

```
    def __truediv__(self, other):
        if isinstance(other, int):
            result = TRational(self.__divisible, self.__divisor
* other)
            result.__reduction()
            return result
        elif isinstance(other, TRational):
            result      =      TRational(self.__divisible      *
other.__divisor, self.__divisor * other.__divisible)
            result.__reduction()
            return result
```

Файл interface.py:

```
from PyQt5 import QtWidgets, QtCore, QtGui
from PyQt5.QtCore import Qt
from PyQt5.QtWidgets import QMainWindow

from greetings import GreetingsWindow

class TInterface:

    #def create_input_table()

    def __init__(self):
        self.window = QMainWindow()
        self.ui = GreetingsWindow()
        self.ui.setupUi(self.window)

    def show(self):
        self.window.show()
```

Файл greetings.py:

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtWidgets import QMainWindow
from matrix import matrix
from number import number
from PyQt5.QtCore import pyqtSlot

class GreetingsWindow(QMainWindow):
```

```
    def setupUi(self, MainWindow):
        self.square_matrix = matrix()
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(800, 392)
        font = QtGui.QFont()
        font.setPointSize(16)
        MainWindow.setFont(font)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(10, 0, 781, 41))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.label_2 = QtWidgets.QLabel(self.centralwidget)
        self.label_2.setGeometry(QtCore.QRect(360,   260,   391,
41))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.label_2.setFont(font)
        self.label_2.setObjectName("label_2")
        self.label_3 = QtWidgets.QLabel(self.centralwidget)
        self.label_3.setGeometry(QtCore.QRect(360,   290,   391,
41))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.label_3.setFont(font)
        self.label_3.setObjectName("label_3")
        self.lineEdit = QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit.setGeometry(QtCore.QRect(40, 70, 61, 31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit.setFont(font)
        self.lineEdit.setText("")
        self.lineEdit.setObjectName("lineEdit")
```

```
        self.lineEdit_2                                          =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_2.setGeometry(QtCore.QRect(40,   110,   61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_2.setFont(font)
        self.lineEdit_2.setText("")
        self.lineEdit_2.setObjectName("lineEdit_2")
        self.lineEdit_3                                          =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_3.setGeometry(QtCore.QRect(140,   110,   61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_3.setFont(font)
        self.lineEdit_3.setText("")
        self.lineEdit_3.setObjectName("lineEdit_3")
        self.lineEdit_4                                          =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_4.setGeometry(QtCore.QRect(140,   70,   61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_4.setFont(font)
        self.lineEdit_4.setText("")
        self.lineEdit_4.setObjectName("lineEdit_4")
        self.lineEdit_5                                          =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_5.setGeometry(QtCore.QRect(240,   110,   61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_5.setFont(font)
        self.lineEdit_5.setText("")
        self.lineEdit_5.setObjectName("lineEdit_5")
```

```
        self.lineEdit_6                                          =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_6.setGeometry(QtCore.QRect(240,   70,   61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_6.setFont(font)
        self.lineEdit_6.setText("")
        self.lineEdit_6.setObjectName("lineEdit_6")
        self.lineEdit_7                                          =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_7.setGeometry(QtCore.QRect(40,   200,   61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_7.setFont(font)
        self.lineEdit_7.setText("")
        self.lineEdit_7.setObjectName("lineEdit_7")
        self.lineEdit_8                                          =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_8.setGeometry(QtCore.QRect(40,   160,   61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_8.setFont(font)
        self.lineEdit_8.setText("")
        self.lineEdit_8.setObjectName("lineEdit_8")
        self.lineEdit_9                                          =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_9.setGeometry(QtCore.QRect(140,   200,   61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_9.setFont(font)
        self.lineEdit_9.setText("")
        self.lineEdit_9.setObjectName("lineEdit_9")
```

```
        self.lineEdit_10                                        =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_10.setGeometry(QtCore.QRect(140,  160,  61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_10.setFont(font)
        self.lineEdit_10.setText("")
        self.lineEdit_10.setObjectName("lineEdit_10")
        self.lineEdit_11                                        =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_11.setGeometry(QtCore.QRect(240,  200,  61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_11.setFont(font)
        self.lineEdit_11.setText("")
        self.lineEdit_11.setObjectName("lineEdit_11")
        self.lineEdit_12                                        =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_12.setGeometry(QtCore.QRect(240,  160,  61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_12.setFont(font)
        self.lineEdit_12.setText("")
        self.lineEdit_12.setObjectName("lineEdit_12")
        self.lineEdit_13                                        =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_13.setGeometry(QtCore.QRect(40,  290,  61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_13.setFont(font)
        self.lineEdit_13.setText("")
        self.lineEdit_13.setObjectName("lineEdit_13")
```

```
        self.lineEdit_14                                        =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_14.setGeometry(QtCore.QRect(40,   250,   61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_14.setFont(font)
        self.lineEdit_14.setText("")
        self.lineEdit_14.setObjectName("lineEdit_14")
        self.lineEdit_15                                        =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_15.setGeometry(QtCore.QRect(140,  290,  61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_15.setFont(font)
        self.lineEdit_15.setText("")
        self.lineEdit_15.setObjectName("lineEdit_15")
        self.lineEdit_16                                        =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_16.setGeometry(QtCore.QRect(140,  250,  61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_16.setFont(font)
        self.lineEdit_16.setText("")
        self.lineEdit_16.setObjectName("lineEdit_16")
        self.lineEdit_17                                        =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_17.setGeometry(QtCore.QRect(240,  290,  61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_17.setFont(font)
        self.lineEdit_17.setText("")
        self.lineEdit_17.setObjectName("lineEdit_17")
```

```
        self.lineEdit_18                                        =
QtWidgets.QLineEdit(self.centralwidget)
        self.lineEdit_18.setGeometry(QtCore.QRect(240,  250,  61,
31))
        font = QtGui.QFont()
        font.setPointSize(16)
        self.lineEdit_18.setFont(font)
        self.lineEdit_18.setText("")
        self.lineEdit_18.setObjectName("lineEdit_18")
        self.pushButton                                         =
QtWidgets.QPushButton(self.centralwidget)
        self.pushButton.setGeometry(QtCore.QRect(360,  150,  391,
31))
        self.pushButton.setObjectName("pushButton")
        self.pushButton.clicked.connect(self.find_determinant)
        self.pushButton_2                                       =
QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_2.setGeometry(QtCore.QRect(360,      190,
391, 31))
        self.pushButton_2.setObjectName("pushButton_2")
        self.pushButton_2.clicked.connect(self.find_rank)
        self.pushButton_3                                       =
QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_3.setGeometry(QtCore.QRect(360,      230,
391, 31))
        self.pushButton_3.setObjectName("pushButton_3")
        self.pushButton_3.clicked.connect(self.transpose_matrix)
        self.pushButton_4                                       =
QtWidgets.QPushButton(self.centralwidget)
        self.pushButton_4.setGeometry(QtCore.QRect(360,      110,
391, 31))
        self.pushButton_4.setObjectName("pushButton_4")

self.pushButton_4.clicked.connect(self.current_matrix_output)
        self.pushButton_5                                       =
QtWidgets.QPushButton(self.centralwidget)
```

```
        self.pushButton_5.setGeometry(QtCore.QRect(360, 70, 391,
31))
        self.pushButton_5.setObjectName("pushButton_5")

self.pushButton_5.clicked.connect(self.matrix_filled_Clicked)
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 21))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)

        self.retranslateUi(MainWindow)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow",
"MainWindow"))
        self.label.setText(_translate("MainWindow",        "Добро
пожаловать в программу для работы с матрицами рациональных
чисел"))
        self.label_2.setText(_translate("MainWindow",
"Определитель:"))
        self.label_3.setText(_translate("MainWindow", "Ранг:"))
        self.pushButton.setText(_translate("MainWindow",
"Рассчитать определитель матрицы"))
        self.pushButton_2.setText(_translate("MainWindow",
"Рассчитать ранг матрицы"))
        self.pushButton_3.setText(_translate("MainWindow",
"Транспонировать матрицу"))
        self.pushButton_4.setText(_translate("MainWindow",
"Вывести текущую матрицу"))
```

```
        self.pushButton_5.setText(_translate("MainWindow",
"Матрица заполнена"))

    @pyqtSlot()
    def matrix_filled_Clicked(self):
        elements       =       [number(int(self.lineEdit.text()),
int(self.lineEdit_2.text())),
number(int(self.lineEdit_4.text()),
int(self.lineEdit_3.text())),
number(int(self.lineEdit_6.text()),
int(self.lineEdit_5.text())),
                    number(int(self.lineEdit_8.text()),
int(self.lineEdit_7.text())),
number(int(self.lineEdit_10.text()),
int(self.lineEdit_9.text())),
number(int(self.lineEdit_12.text()),
int(self.lineEdit_11.text())),
                    number(int(self.lineEdit_14.text()),
int(self.lineEdit_13.text())),
number(int(self.lineEdit_16.text()),
int(self.lineEdit_15.text())),
number(int(self.lineEdit_18.text()),
int(self.lineEdit_17.text()))]
        self.square_matrix.fill_in_matrix(3, elements)

    @pyqtSlot()
    def transpose_matrix(self):
        transposed_matrix = self.square_matrix.transpose()
        divisible,                  divisor                  =
transposed_matrix[0][0].value.text_representation.split(sep='/')
        self.lineEdit.setText(divisible)
        self.lineEdit_2.setText(divisor)
        divisible,                  divisor                  =
transposed_matrix[0][1].value.text_representation.split(sep='/')
        self.lineEdit_4.setText(divisible)
        self.lineEdit_3.setText(divisor)
```

```
        divisible,                  divisor                  =
transposed_matrix[0][2].value.text_representation.split(sep='/')
        self.lineEdit_6.setText(divisible)
        self.lineEdit_5.setText(divisor)
        divisible,                  divisor                  =
transposed_matrix[1][0].value.text_representation.split(sep='/')
        self.lineEdit_8.setText(divisible)
        self.lineEdit_7.setText(divisor)
        divisible,                  divisor                  =
transposed_matrix[1][1].value.text_representation.split(sep='/')
        self.lineEdit_10.setText(divisible)
        self.lineEdit_9.setText(divisor)
        divisible,                  divisor                  =
transposed_matrix[1][2].value.text_representation.split(sep='/')
        self.lineEdit_12.setText(divisible)
        self.lineEdit_11.setText(divisor)
        divisible,                  divisor                  =
transposed_matrix[2][0].value.text_representation.split(sep='/')
        self.lineEdit_14.setText(divisible)
        self.lineEdit_13.setText(divisor)
        divisible,                  divisor                  =
transposed_matrix[2][1].value.text_representation.split(sep='/')
        self.lineEdit_16.setText(divisible)
        self.lineEdit_15.setText(divisor)
        divisible,                  divisor                  =
transposed_matrix[2][2].value.text_representation.split(sep='/')
        self.lineEdit_18.setText(divisible)
        self.lineEdit_17.setText(divisor)

    @pyqtSlot()
    def current_matrix_output(self):
        matrix = self.square_matrix.get_matrix_array()
        divisible,                  divisor                  =
matrix[0][0].value.text_representation.split(sep='/')
        self.lineEdit.setText(divisible)
        self.lineEdit_2.setText(divisor)
```

```
        divisible,                  divisor                  =
matrix[0][1].value.text_representation.split(sep='/')
        self.lineEdit_4.setText(divisible)
        self.lineEdit_3.setText(divisor)
        divisible,                  divisor                  =
matrix[0][2].value.text_representation.split(sep='/')
        self.lineEdit_6.setText(divisible)
        self.lineEdit_5.setText(divisor)
        divisible,                  divisor                  =
matrix[1][0].value.text_representation.split(sep='/')
        self.lineEdit_8.setText(divisible)
        self.lineEdit_7.setText(divisor)
        divisible,                  divisor                  =
matrix[1][1].value.text_representation.split(sep='/')
        self.lineEdit_10.setText(divisible)
        self.lineEdit_9.setText(divisor)
        divisible,                  divisor                  =
matrix[1][2].value.text_representation.split(sep='/')
        self.lineEdit_12.setText(divisible)
        self.lineEdit_11.setText(divisor)
        divisible,                  divisor                  =
matrix[2][0].value.text_representation.split(sep='/')
        self.lineEdit_14.setText(divisible)
        self.lineEdit_13.setText(divisor)
        divisible,                  divisor                  =
matrix[2][1].value.text_representation.split(sep='/')
        self.lineEdit_16.setText(divisible)
        self.lineEdit_15.setText(divisor)
        divisible,                  divisor                  =
matrix[2][2].value.text_representation.split(sep='/')
        self.lineEdit_18.setText(divisible)
        self.lineEdit_17.setText(divisor)

    @pyqtSlot()
    def find_determinant(self):
        determinant = self.square_matrix.determinant()
```

```
        self.label_2.setText('Определитель: ' + determinant)

    @pyqtSlot()
    def find_rank(self):
        rank = self.square_matrix.rank()
        self.label_3.setText('Ранг: ' + rank)
```