Министерство науки и высшего образования РФ

ФГАОУ ВО Пермский национальный исследовательский  
политехнический университет

Кафедра «Вычислительная математика, механика и биомеханика»

Отчёт по лабораторной работе № 6  
тема «Задачи линейного программирования»  
по дисциплине «Информатика»

Выполнил: студент группы ПМ-23-2б Воронин М.А.

Проверил: ст. пр. каф. ВММБ Ильиных Г.В.

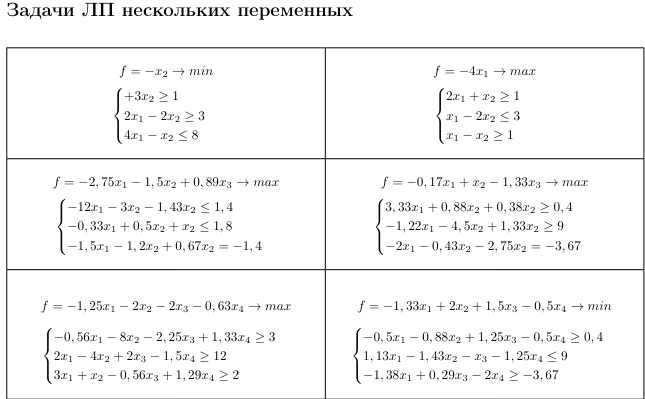
Пермь, 2024

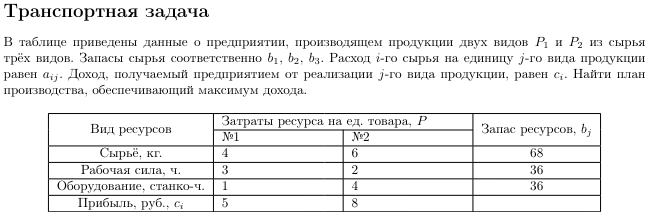
**Задание 1**

***1.1 Постановка задачи***

Каждая задача ЛП (всего 7 штук включая транспортную задачу) решается на отдельном листе электронной таблицы с соответствующим названием.

В рамках решения задачи на отдельном листе оформляются (разными цветами с подписями) ячейки под вектор неизвестных (параметры оптимизации), коэффициенты целевой функции, целевая функция, ограничения.





***1.2 Алгоритм решения***

Для решения задачи 1 используется надстройка Поиск решения. Для решения транспортной задачи была составлена целевая функция:

Так же была составлена система ограничений:

Данные были введены в таблицу и была применена надстройка Поиск Решения.

***1.3 Проверка решений***

Таблица 1 – Проверка решений задачи 1

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Транспортная задача |  |

**Задание 2**

***2.1 Постановка задачи***

Разработать программу, определяющую вектор неизвестных задачи с помощью метода scipy.optimize.linprog. С помощью программы определить вектор неизвестных, сравнить с результатами Задания 1

***2.2 Алгоритм решения***

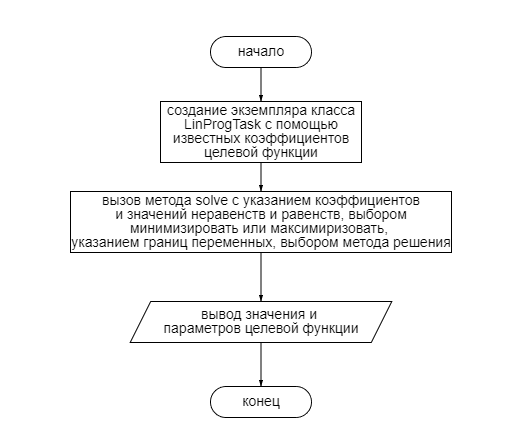


Рис.2 – Блок-схема задания 2

***2.3 Код программы***

from scipy.optimize import linprog  
  
  
class LinProgTask:  
 count = 0  
  
 def \_\_init\_\_(self, target\_vector=None):  
 self.vector\_x = []  
 if target\_vector is None:  
 target\_vector = []  
 self.target\_vector = target\_vector  
 self.target\_value = 0  
 LinProgTask.count += 1  
  
 def solve(self, uneq\_vector=None, uneq\_value=None, eq\_vector=None, eq\_value=None, x\_bounds=None, max=False,  
 method='highs'):  
 if (uneq\_value and uneq\_vector) or (eq\_value and eq\_vector):  
 if not max:  
 res = linprog(self.target\_vector, A\_ub=uneq\_vector, b\_ub=uneq\_value, A\_eq=eq\_vector, b\_eq=eq\_value,  
 bounds=x\_bounds, method=method)  
 self.vector\_x = res.x  
 self.target\_value = res.fun  
  
 if max:  
 self.target\_vector = [-x for x in self.target\_vector]  
 res = linprog(self.target\_vector, A\_ub=uneq\_vector, b\_ub=uneq\_value, A\_eq=eq\_vector, b\_eq=eq\_value,  
 bounds=x\_bounds, method=method)  
 self.vector\_x = -res.x  
 self.target\_value = -res.fun  
  
 def \_\_str\_\_(self):  
 return f"Параметры целевой функции равны: {self.vector\_x}\nЗначение целевой функции равно: {self.target\_value}"  
  
 def \_\_del\_\_(self):  
 del self.vector\_x  
 del self.target\_value  
 del self.target\_vector  
 LinProgTask.count -= 1  
  
 def \_\_len\_\_(self):  
 return len(self.vector\_x)

***2.4 Проверка решений***

|  |  |
| --- | --- |
| Решение Python | Решение Excel |
|  |  |

**Задание 3**

***3.1 Постановка задачи***

Разработать графический пользовательский интерфейс программы, реализующей решение задачи линейного программирования.

***3.2 Алгоритм решения***

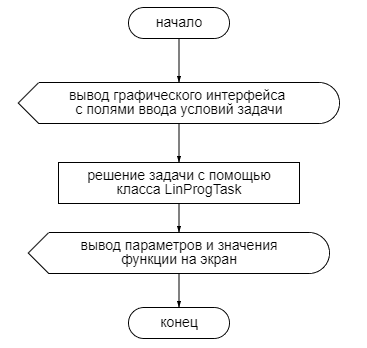
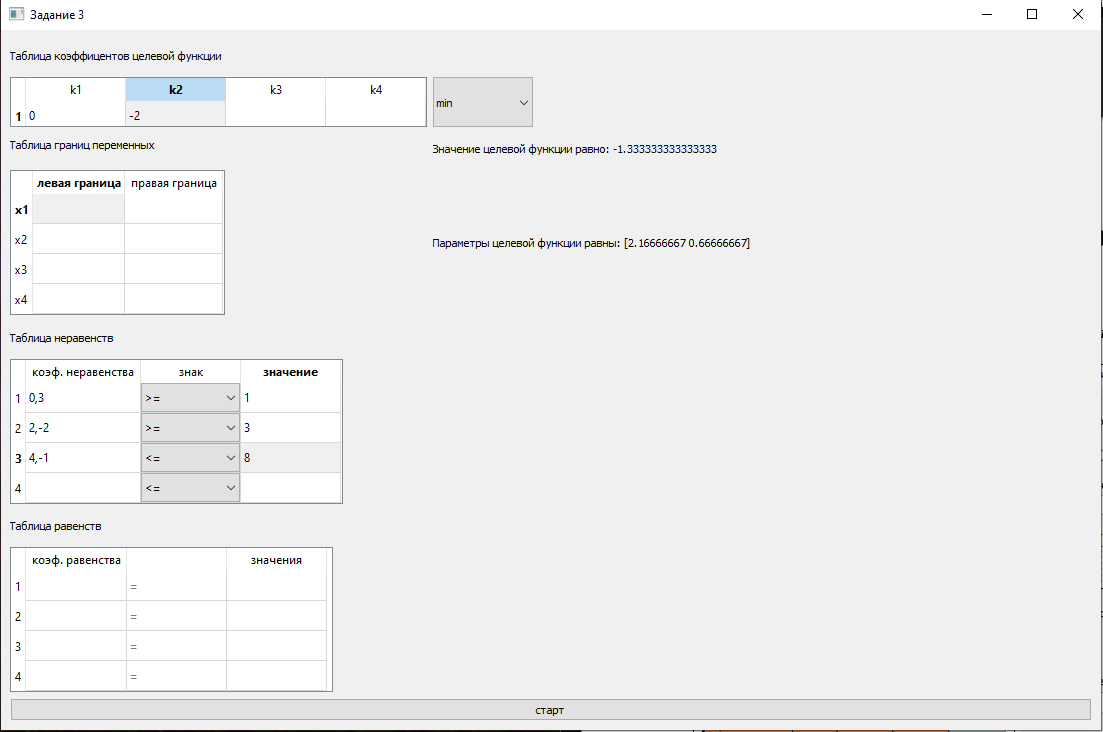


Рис.3 – Блок-схема задания 3

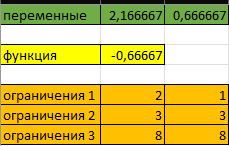
***3.3 Код программы***

from PyQt5.QtWidgets import (QApplication, QLabel, QMainWindow, QPushButton, QFormLayout, QWidget, QComboBox, QTableWidget, QTableWidgetItem)  
import matplotlib.pyplot as plt  
from PyQt5.QtCore import Qt as qt  
  
  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self, parent=None):  
 super(MainWindow, self).\_\_init\_\_(parent)  
  
 self.setWindowTitle('Задание 3')  
 self.fig = plt.figure()  
 self.setGeometry(100, 100, 1100, 700)  
 cental\_widget = QWidget()  
 layout = QFormLayout()  
 cental\_widget.setLayout(layout)  
 plt.grid(True)  
 self.setCentralWidget(cental\_widget)  
  
 self.k\_table\_label = QLabel('Таблица коэффицентов целевой функции')  
 self.k\_table = QTableWidget()  
 self.k\_table.setRowCount(1)  
 self.k\_table.setColumnCount(4)  
 self.k\_table.setHorizontalHeaderLabels(['k1', 'k2', 'k3', 'k4'])  
 self.k\_table.setFixedSize(417, 50)  
  
 self.uneq\_table\_label = QLabel('Таблица неравенств')  
 self.uneq\_table = QTableWidget()  
 self.uneq\_table.setRowCount(4)  
 self.uneq\_table.setColumnCount(3)  
 self.uneq\_table.setHorizontalHeaderLabels(['коэф. неравенства', 'знак', 'значение'])  
 self.uneq\_table.resizeColumnToContents(0)  
 self.uneq\_table.setFixedSize(333, 145)  
 for i in range(self.uneq\_table.rowCount()):  
 widget = QComboBox()  
 widget.addItems(['<=', '>='])  
 self.uneq\_table.setCellWidget(i, 1, widget)  
  
 self.eq\_table\_label = QLabel('Таблица равенств')  
 self.eq\_table = QTableWidget()  
 self.eq\_table.setColumnCount(3)  
 self.eq\_table.setRowCount(4)  
 self.eq\_table.setHorizontalHeaderLabels(['коэф. равенства', '', 'значения'])  
 self.eq\_table.resizeColumnToContents(0)  
 self.eq\_table.setFixedSize(323, 145)  
 for i in range(self.eq\_table.rowCount()):  
 self.eq\_table.setItem(i, 1, QTableWidgetItem('='))  
 self.eq\_table.item(i, 1).setFlags(qt.ItemFlag.ItemIsEditable)  
  
 self.start\_button = QPushButton('старт')  
 self.start\_button.clicked.connect(self.solve\_task)  
  
 self.minmax = QComboBox()  
 self.minmax.addItems(['min', 'max'])  
 self.minmax.setFixedSize(100, 50)  
  
 self.bounds\_label = QLabel('Таблица границ переменных')  
 self.bounds = QTableWidget()  
 self.bounds.setColumnCount(2)  
 self.bounds.setRowCount(4)  
 self.bounds.setHorizontalHeaderLabels(['левая граница', 'правая граница'])  
 self.bounds.setVerticalHeaderLabels(['x1', 'x2', 'x3', 'x4'])  
 self.bounds.resizeColumnToContents(0)  
 self.bounds.resizeColumnToContents(1)  
 self.bounds.setFixedSize(215, 145)  
  
 self.answer\_label = QLabel('Значение целевой функции равно: ')  
 self.answer = QLabel('Параметры целевой функции равны: ')  
 layout.addRow(self.k\_table\_label)  
 layout.addRow(self.k\_table, self.minmax)  
 layout.addRow(self.bounds\_label, self.answer\_label)  
 layout.addRow(self.bounds, self.answer)  
 layout.addRow(self.uneq\_table\_label)  
 layout.addRow(self.uneq\_table)  
  
 layout.addRow(self.eq\_table\_label)  
 layout.addRow(self.eq\_table)  
  
 layout.addRow(self.start\_button)  
  
 def solve\_task(self):  
 target\_vector = []  
 for i in range(self.k\_table.columnCount()):  
 if self.k\_table.item(0, i) is not None:  
 if self.k\_table.item(0, i).text() != '':  
 target\_vector.append(float(self.k\_table.item(0, i).text()))  
 task = LinProgTask(target\_vector)  
 uneq\_vector = []  
 uneq\_value = []  
 for i in range(self.uneq\_table.rowCount()):  
 if self.uneq\_table.item(i, 0) is not None and self.uneq\_table.item(i, 2) is not None:  
 k\_uneq\_vector = list(map(float, self.uneq\_table.item(i, 0).text().split(',')))  
 if self.uneq\_table.cellWidget(i, 1).currentText() == '<=':  
 uneq\_vector.append(k\_uneq\_vector)  
 uneq\_value.append(float(self.uneq\_table.item(i, 2).text()))  
 else:  
 uneq\_vector.append([-x for x in k\_uneq\_vector])  
 uneq\_value.append(-float(self.uneq\_table.item(i, 2).text()))  
  
 eq\_vector = []  
 eq\_value = []  
 for i in range(self.eq\_table.rowCount()):  
 if self.eq\_table.item(i, 0) is not None and self.eq\_table.item(i, 2) is not None:  
 k\_eq\_vector = list(map(float, self.eq\_table.item(i, 0).text().split(',')))  
 eq\_vector.append(k\_eq\_vector)  
 eq\_value.append(float(self.eq\_table.item(i, 2).text()))  
 x\_bounds = []  
 for i in range(self.bounds.rowCount()):  
 if self.bounds.item(i, 0) is not None and self.bounds.item(i, 1) is not None:  
 try:  
 x\_bounds.append((float(self.bounds.item(i, 0).text()),  
 float(self.bounds.item(i, 1).text())))  
 except ValueError:  
 if self.bounds.item(i, 0).text() == '' or self.bounds.item(i, 0).text() == 'None':  
 x\_bounds.append((float(self.bounds.item(i, 0).text()),  
 None))  
 elif self.bounds.item(i, 1).text() == '' or self.bounds.item(i, 1).text() == 'None':  
 x\_bounds.append((None, float(self.bounds.item(i, 1).text())))  
  
 if len(x\_bounds) != len(target\_vector):  
 if len(x\_bounds) < len(target\_vector):  
 while len(x\_bounds) != len(target\_vector):  
 x\_bounds.append((None, None))  
 elif len(x\_bounds) > len(target\_vector):  
 x\_bounds = [(None, None) for \_ in range(len(target\_vector))]  
 if len(eq\_vector) == 0 or len(eq\_value) == 0:  
 eq\_vector = None  
 eq\_value = None  
 if len(uneq\_vector) == 0 or len(uneq\_value) == 0:  
 uneq\_vector = None  
 uneq\_value = None  
  
 if self.minmax.currentText() == 'min':  
 task.solve(uneq\_vector, uneq\_value, eq\_vector, eq\_value, x\_bounds=x\_bounds)  
 else:  
 task.solve(uneq\_vector, uneq\_value, eq\_vector, eq\_value, x\_bounds=x\_bounds, max=True)  
 self.answer\_label.setText(str(task).split('\n')[1])  
 self.answer.setText(str(task).split('\n')[0])  
  
  
app = QApplication([])  
main = MainWindow()  
main.show()  
app.exec()

***3.4 Проверка решений***

******

а)



б)

Рис.4 – Проверка решений задания 3:

а) Графический интерфейс программы, б) Решение Excel

from scipy.optimize import linprog  
  
class LinProgTask:  
 count = 0  
  
 def \_\_init\_\_(self, target\_vector=None):  
 self.vector\_x = []  
 if target\_vector is None:  
 target\_vector = []  
 self.target\_vector = target\_vector  
 self.target\_value = 0  
 LinProgTask.count += 1  
  
 def solve(self, uneq\_vector=None, uneq\_value=None, eq\_vector=None, eq\_value=None, x\_bounds=None, max=False,  
 method='highs'):  
 if (uneq\_value and uneq\_vector) or (eq\_value and eq\_vector):  
 if not max:  
 res = linprog(self.target\_vector, A\_ub=uneq\_vector, b\_ub=uneq\_value, A\_eq=eq\_vector, b\_eq=eq\_value,  
 bounds=x\_bounds, method=method)  
 self.vector\_x = res.x  
 self.target\_value = res.fun  
  
 if max:  
 self.target\_vector = [-x for x in self.target\_vector]  
 res = linprog(self.target\_vector, A\_ub=uneq\_vector, b\_ub=uneq\_value, A\_eq=eq\_vector, b\_eq=eq\_value,  
 bounds=x\_bounds, method=method)  
 self.vector\_x = -res.x  
 self.target\_value = -res.fun  
  
 def \_\_str\_\_(self):  
 return f"Параметры целевой функции равны: {self.vector\_x}\nЗначение целевой функции равно: {self.target\_value}"  
  
 def \_\_del\_\_(self):  
 del self.vector\_x  
 del self.target\_value  
 del self.target\_vector  
 LinProgTask.count -= 1  
  
 def \_\_len\_\_(self):  
 return len(self.vector\_x)  
  
from PyQt5.QtWidgets import (QApplication, QLabel, QMainWindow, QPushButton, QFormLayout, QWidget, QComboBox,QTableWidget, QTableWidgetItem)  
import matplotlib.pyplot as plt  
from PyQt5.QtCore import Qt as qt  
  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self, parent=None):  
 super(MainWindow, self).\_\_init\_\_(parent)  
  
 self.setWindowTitle('Задание 3')  
 self.fig = plt.figure()  
 self.setGeometry(100, 100, 1100, 700)  
 cental\_widget = QWidget()  
 layout = QFormLayout()  
 cental\_widget.setLayout(layout)  
 plt.grid(True)  
 self.setCentralWidget(cental\_widget)  
  
 self.k\_table\_label = QLabel('Таблица коэффицентов целевой функции')  
 self.k\_table = QTableWidget()  
 self.k\_table.setRowCount(1)  
 self.k\_table.setColumnCount(4)  
 self.k\_table.setHorizontalHeaderLabels(['k1', 'k2', 'k3', 'k4'])  
 self.k\_table.setFixedSize(417, 50)  
  
 self.uneq\_table\_label = QLabel('Таблица неравенств')  
 self.uneq\_table = QTableWidget()  
 self.uneq\_table.setRowCount(4)  
 self.uneq\_table.setColumnCount(3)  
 self.uneq\_table.setHorizontalHeaderLabels(['коэф. неравенства', 'знак', 'значение'])  
 self.uneq\_table.resizeColumnToContents(0)  
 self.uneq\_table.setFixedSize(333, 145)  
 for i in range(self.uneq\_table.rowCount()):  
 widget = QComboBox()  
 widget.addItems(['<=', '>='])  
 self.uneq\_table.setCellWidget(i, 1, widget)  
  
 self.eq\_table\_label = QLabel('Таблица равенств')  
 self.eq\_table = QTableWidget()  
 self.eq\_table.setColumnCount(6)  
 self.eq\_table.setRowCount(4)  
 self.eq\_table.setHorizontalHeaderLabels(['k1','k2','k3','k4','знак','правая часть '])  
 for i in range(self.eq\_table.rowCount()):  
 self.eq\_table.setItem(i, 4, QTableWidgetItem('='))  
  
  
 self.start\_button = QPushButton('старт')  
 self.start\_button.clicked.connect(self.solve\_task)  
  
 self.minmax = QComboBox()  
 self.minmax.addItems(['min', 'max'])  
 self.minmax.setFixedSize(100, 50)  
  
 self.bounds\_label = QLabel('Таблица границ переменных')  
 self.bounds = QTableWidget()  
 self.bounds.setColumnCount(2)  
 self.bounds.setRowCount(4)  
 self.bounds.setHorizontalHeaderLabels(['левая граница', 'правая граница'])  
 self.bounds.setVerticalHeaderLabels(['x1', 'x2', 'x3', 'x4'])  
 self.bounds.resizeColumnToContents(0)  
 self.bounds.resizeColumnToContents(1)  
 self.bounds.setFixedSize(215, 145)  
  
 self.answer\_label = QLabel('Значение целевой функции равно: ')  
 self.answer = QLabel('Параметры целевой функции равны: ')  
 layout.addRow(self.k\_table\_label)  
 layout.addRow(self.k\_table, self.minmax)  
 layout.addRow(self.bounds\_label, self.answer\_label)  
 layout.addRow(self.bounds, self.answer)  
 layout.addRow(self.uneq\_table\_label)  
 layout.addRow(self.uneq\_table)  
  
 layout.addRow(self.eq\_table\_label)  
 layout.addRow(self.eq\_table)  
  
 layout.addRow(self.start\_button)  
  
 def solve\_task(self):  
 target\_vector = []  
 for i in range(self.k\_table.columnCount()):  
 if self.k\_table.item(0, i) is not None:  
 if self.k\_table.item(0, i).text() != '':  
 target\_vector.append(float(self.k\_table.item(0, i).text()))  
 task = LinProgTask(target\_vector)  
 uneq\_vector = []  
 uneq\_value = []  
 for i in range(self.uneq\_table.rowCount()):  
 if self.uneq\_table.item(i, 0) is not None and self.uneq\_table.item(i, 2) is not None:  
 k\_uneq\_vector = list(map(float, self.uneq\_table.item(i, 0).text().split(',')))  
 if self.uneq\_table.cellWidget(i, 1).currentText() == '<=':  
 uneq\_vector.append(k\_uneq\_vector)  
 uneq\_value.append(float(self.uneq\_table.item(i, 2).text()))  
 else:  
 uneq\_vector.append([-x for x in k\_uneq\_vector])  
 uneq\_value.append(-float(self.uneq\_table.item(i, 2).text()))  
  
 for i in range(self.eq\_table.rowCount()):  
 for j in range(4):  
 if self.eq\_table.item(i, j) is not None:  
 A\_eq = []  
 for i in range(self.eq\_table.rowCount()):  
 subvector\_2 = []  
 for j in range(4):  
 if self.eq\_table.item(i, j) is not None:  
 subvector\_2.append(float(self.eq\_table.item(i, j).text()))  
 if len(subvector\_2) > 0:  
 A\_eq.append(subvector\_2)  
 else:  
 A\_eq = None  
  
 for i in range(self.eq\_table.rowCount()):  
 if self.eq\_table.item(i, 5) is not None:  
 b\_eq = []  
 for i in range(self.eq\_table.rowCount()):  
 if self.eq\_table.item(i, 5) is not None:  
 b\_eq.append(float(self.eq\_table.item(i, 5).text()))  
 else:  
 b\_eq = None  
 x\_bounds = []  
 for i in range(self.bounds.rowCount()):  
 if self.bounds.item(i, 0) is not None and self.bounds.item(i, 1) is not None:  
 try:  
 x\_bounds.append((float(self.bounds.item(i, 0).text()),  
 float(self.bounds.item(i, 1).text())))  
 except ValueError:  
 if self.bounds.item(i, 0).text() == '' or self.bounds.item(i, 0).text() == 'None':  
 x\_bounds.append((float(self.bounds.item(i, 0).text()),  
 None))  
 elif self.bounds.item(i, 1).text() == '' or self.bounds.item(i, 1).text() == 'None':  
 x\_bounds.append((None, float(self.bounds.item(i, 1).text())))  
  
 if len(x\_bounds) != len(target\_vector):  
 if len(x\_bounds) < len(target\_vector):  
 while len(x\_bounds) != len(target\_vector):  
 x\_bounds.append((None, None))  
 elif len(x\_bounds) > len(target\_vector):  
 x\_bounds = [(None, None) for \_ in range(len(target\_vector))]  
 if len(uneq\_vector) == 0 or len(uneq\_value) == 0:  
 uneq\_vector = None  
 uneq\_value = None  
  
 if self.minmax.currentText() == 'min':  
 task.solve(uneq\_vector, uneq\_value, A\_eq=A\_eq, b\_eq=b\_eq, x\_bounds=x\_bounds)  
 else:  
 task.solve(uneq\_vector, uneq\_value, A\_eq=A\_eq, b\_eq=b\_eq, x\_bounds=x\_bounds, max=True)  
 self.answer\_label.setText(str(task).split('\n')[1])  
 self.answer.setText(str(task).split('\n')[0])  
app = QApplication([])  
main = MainWindow()  
main.show()  
app.exec()