Поиск оптимального портфеля оптовых закупок

# Краткое описание программы

Программа «**Поиск оптимального портфеля закупок**» (Далее **ПОП**) вычисляет оптимальное количество партий различных товаров, которые необходимо закупить предприятию, чтобы при их продаже получить наибольшую прибыль

**ПОП получает на входе:**

1. Книгу Excel, с приведенными в ней к нужному формату данными (об этом в главе «**Подготовка данных**»)
2. Коэффициенты, указываемые на форме приложения (об этом в главе «**Работа с формой**»)

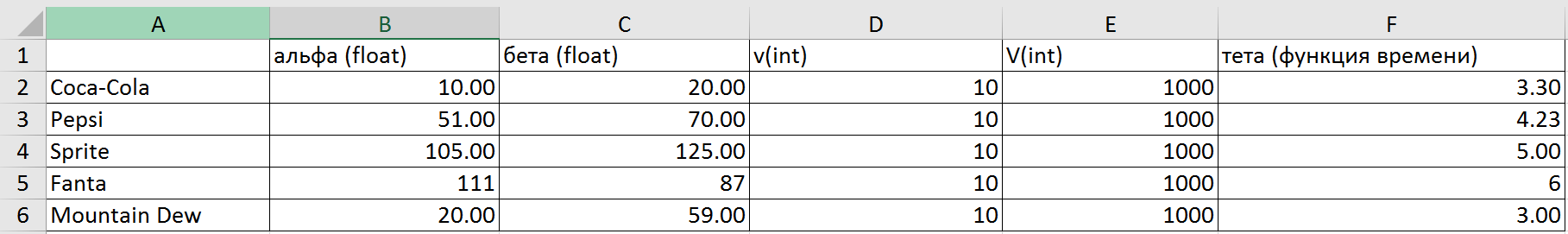
**Компоненты ПОП:**

1. Собственноручно-написанный алгоритм
2. CBC (Coin-or branch and cut) – решатель задач линейного программирования с открытым исходным кодом, написанный на C++ (<https://github.com/coin-or/Cbc>, версия: 2.10.2)
3. Graphviz – программа-визуализатор с открытым исходным кодом (<https://graphviz.gitlab.io/about/>, версия: 2.38)
4. Интерпретатор Python 3.7.2
5. Загружаемые библиотеки Python (нужны ссылки?):
   * 1. anytree==2.6.0
     2. graphviz==0.10.1
     3. matplotlib==3.0.3
     4. numpy==1.15.4
     5. openpyxl==2.6.1
     6. Pillow==6.0.0
     7. PuLP==1.6.9
     8. pydot==1.4.1
     9. pyinstaller==3.4
     10. python-docx==0.8.10
     11. sympy==1.3
6. Стандартные библиотеки Python:
   * 1. tkinter
     2. math
     3. platform
     4. os
     5. string
     6. sys
     7. ctypes

**ПОП возвращает:**

1. Окно сообщения с результатами
2. Документ DOCX с подробными результатами (на усмотрение пользователя)

# Подготовка данных



**Ограничения:**

1. Первая строка (1) – должна быть либо пуста, либо содержать названия столбцов. Ее ПОП в расчете не использует.
2. Данные должны обладать целостностью, иначе ПОП не сможет работать.
3. Данные должны начинаться со второй строки (2)
4. Колонки должны иметь данные такого формата:

**А)** Названия товаров. Они будут использованы в файле с результатами

**В)** Альфа – это цена закупки одной единицы товара этого наименования. Любое дробное или целое число.

**С)** Бета – это цена продажи одной единицы товара этого наименования. Любое дробное или целое число.

**D)** v – это минимальное количество закупаемого товара, количество товара в одной партии. Только целое число

**E)** V – это максимальное количество закупаемого товара. Только целое число

**F)** Тета – это число, либо **функция от x**: степенная, показательная, линейная. Степенная и показательная функция пишутся в формате **a\*\*x, x\*\*a**. Обязательно **через точку**. **Например:**

2.3 \* (2.4\*\*x)

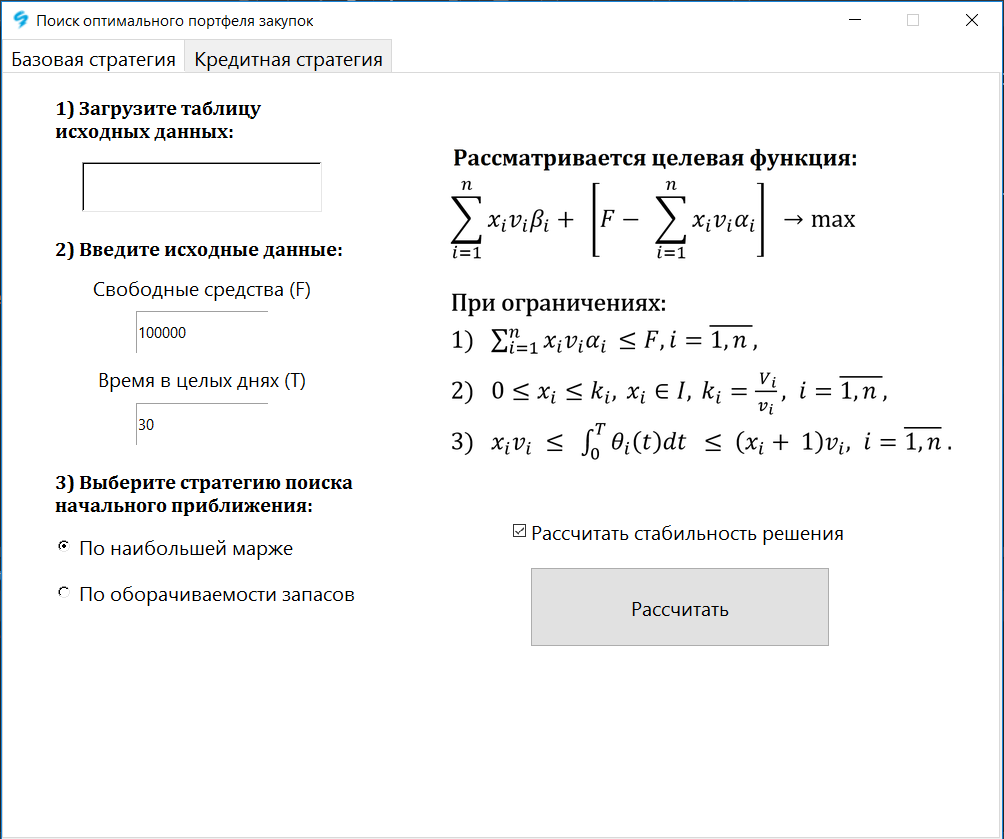
или

x \* (x+1) \* (x+2)

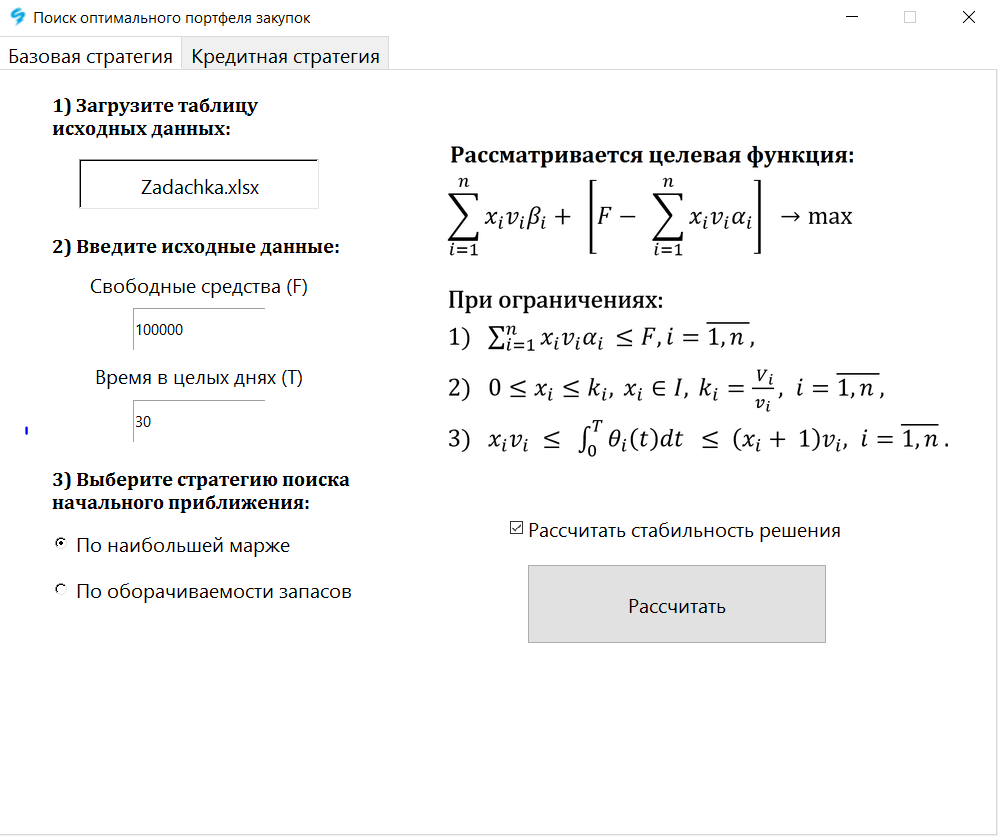
или

4.3245

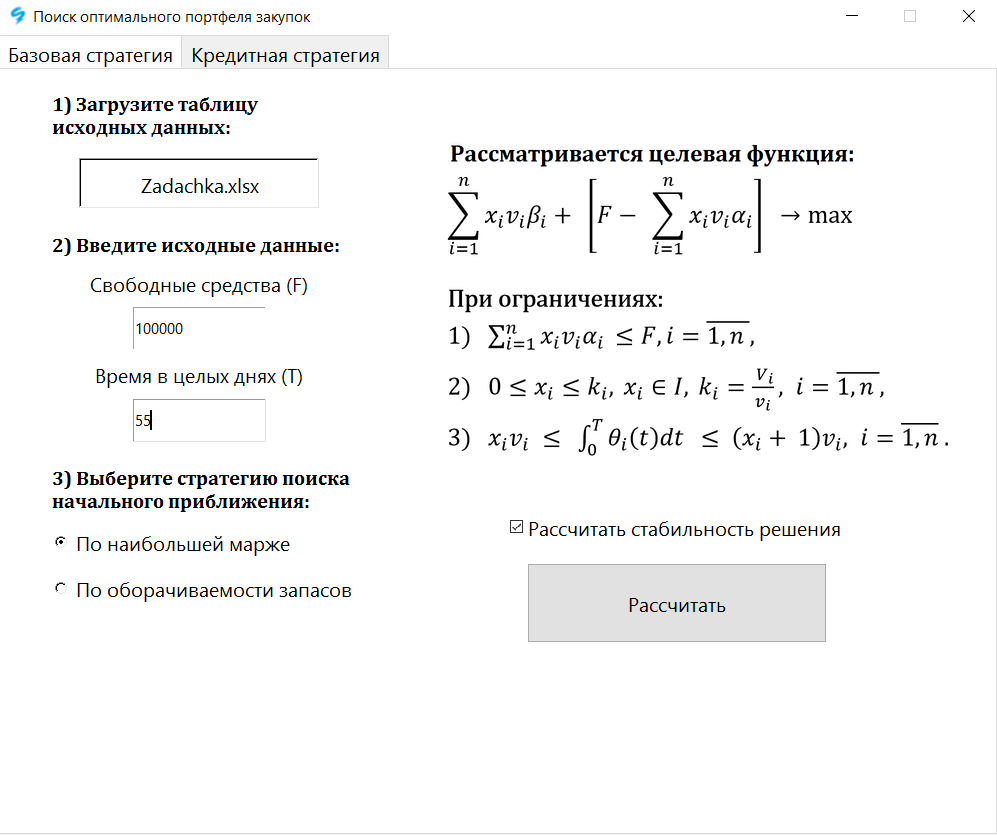
# Работа с формой



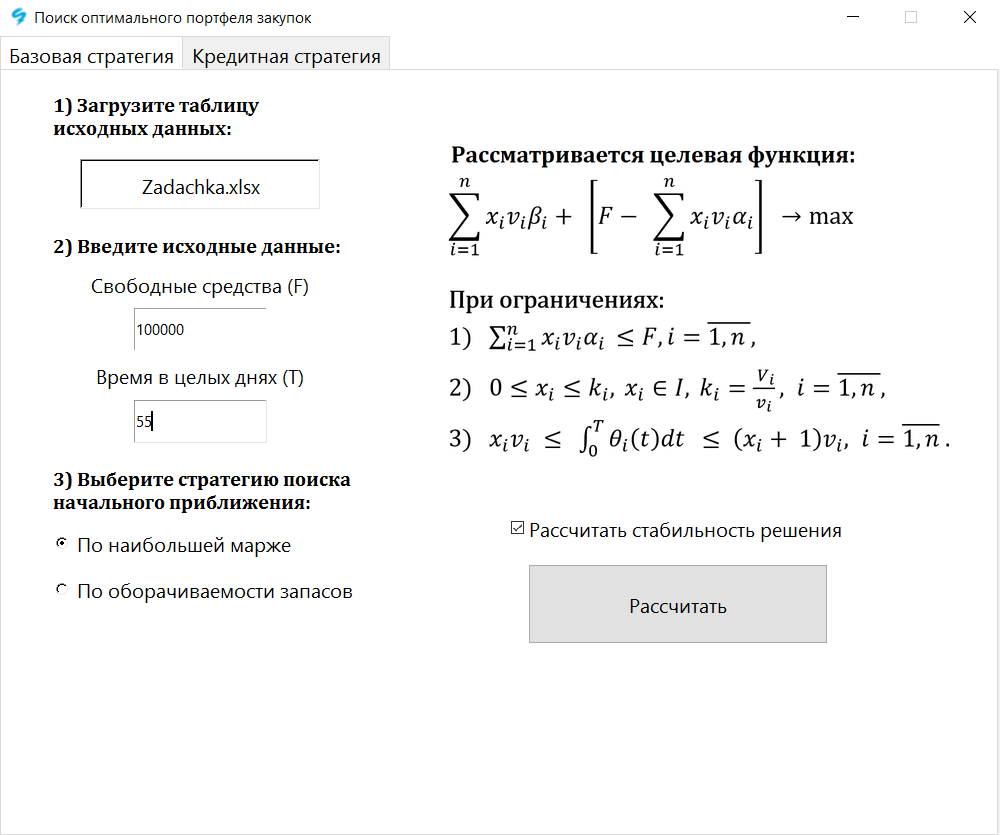
* + - 1. Выберите файл Excel c подготовленными данными



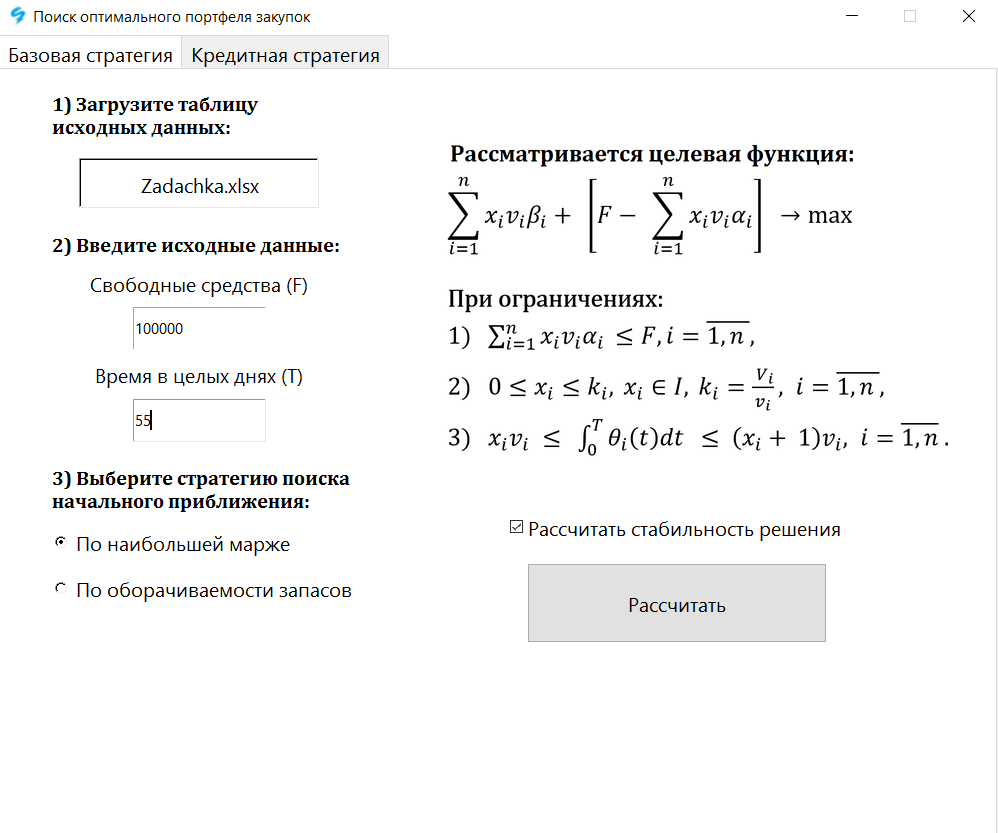
* + - 1. Введите коэффициенты, либо оставьте значения по умолчанию



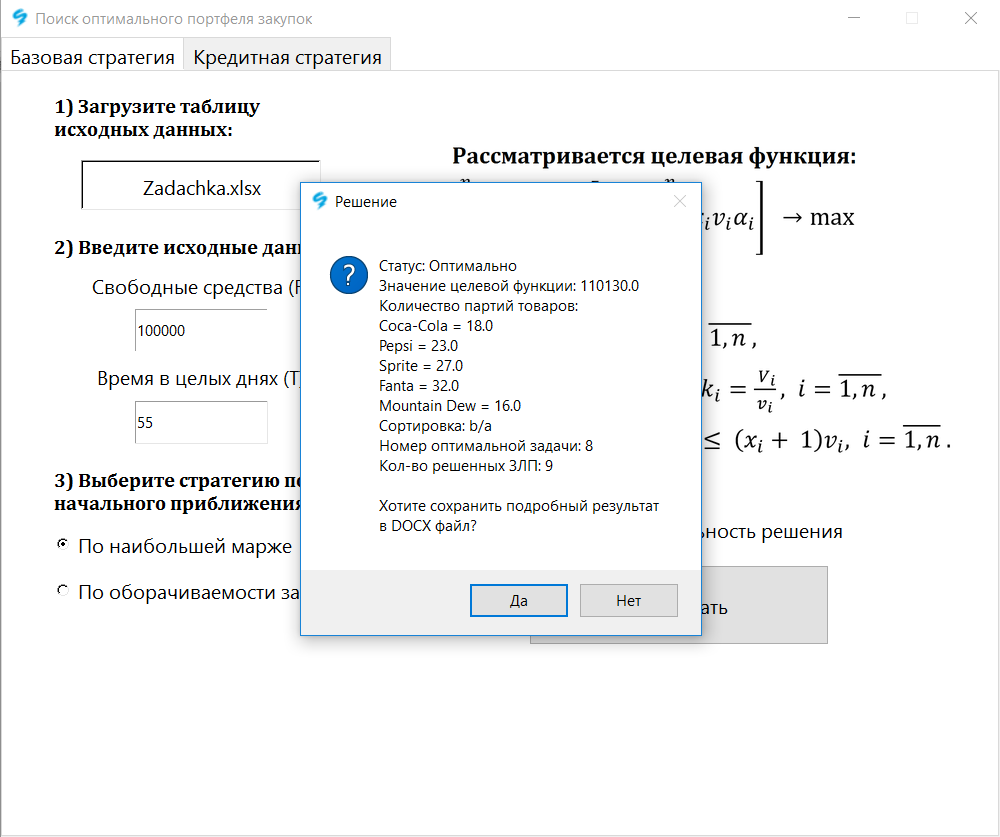
3) Выберите вариант сортировки данных из Excel, либо оставьте значение по умолчанию



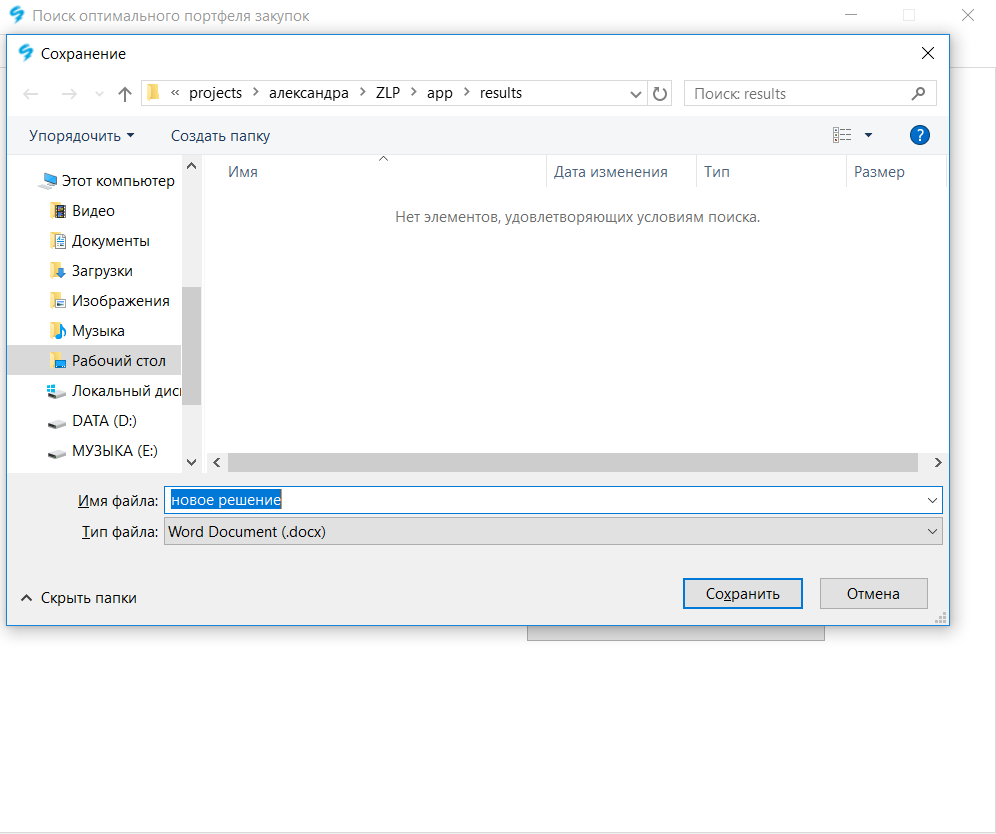
4) Чтобы посчитать стабильность оптимального решения при разном уровне инфляции, оставьте галочку в пункте **«Рассчитать стабильность решения»**, либо уберите ее, если это не нужно

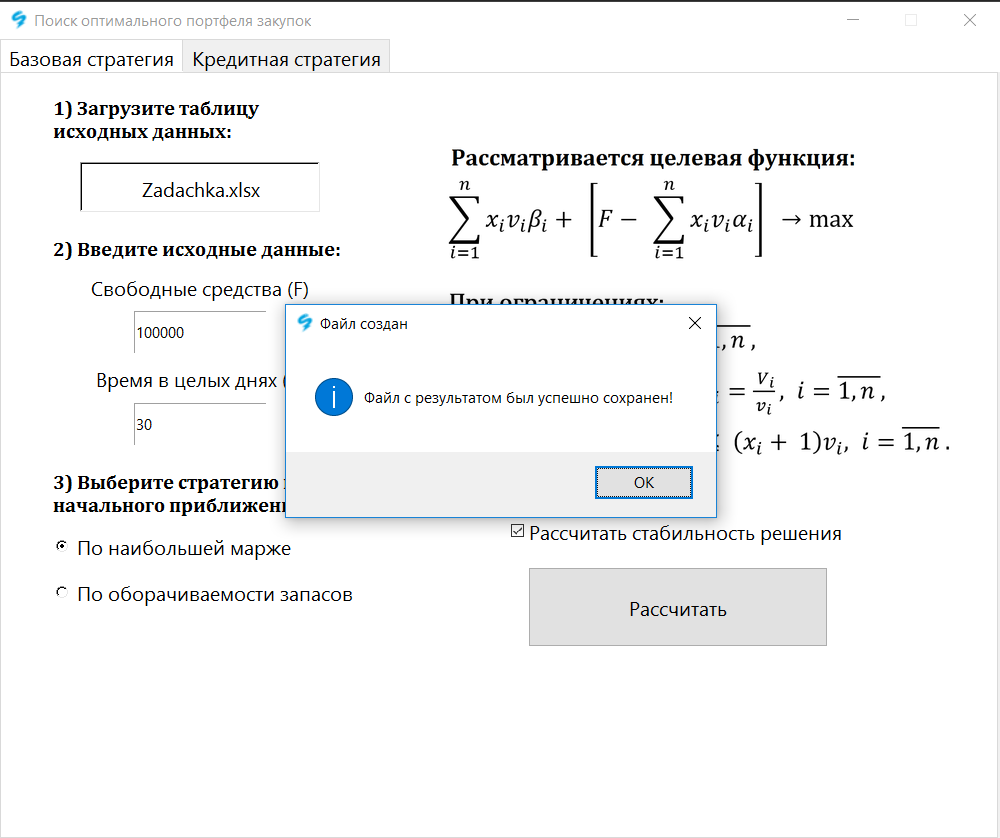


1. Нажмите кнопку **«Рассчитать»**, когда все данные в форме будут заполнены



1. Дождитесь сообщения с результатом вычисления. Если Вы хотите сохранить более подробный результат, содержащий **дерево решений подзадач** и график **стабильности решения**, нажмите «Да» и выберите место, в которое Вы хотите сохранить документ DOCX с результатом. По умолчанию открывается папка **results**, находящаяся в папке с программой.





1. После нажатия кнопки «ОК», файл будет автоматически открыт

# Как работает сама программа

**В алгоритме ПОП три файла с кодом**:

1. int\_linear\_main.py
2. user\_form.py
3. user\_form\_support.py

4.1 user\_form.py, user\_form\_support.py

Здесь содержится код самой формы. Основная библиотека – **tkinter**. Она считается стандартной для создания форм. Основу этого кода я сгенерировал с помощью построителя форм на tkinter – **PAGE** (он выглядит так: <https://www.youtube.com/watch?v=znFj1drEL9E>), не уверен, что тебе нужна будет эта информация, можешь просто сказать, что писала на ткинтере. Основа кода претерпела очень много изменений с тех пор и писались врукопашную.

Файл **user\_form\_support** – это часть сгенерированного кода. Он очень короткий и я сам до конца не уверен, зачем было создавать этот вспомогательный файл))) Подозреваю, что это просто из-за того, что PAGE очень древний. Я не стал его трогать и оставил в паре с **user\_form.py.**

Файл **user\_form.py** – это основной файл формы. Здесь использованы библиотеки

import ctypes  
import sys  
import tkinter.font as tkFont  
from PIL import Image, ImageTk as itk  
from numpy import linspace  
import app.user\_form\_support  
from app.int\_linear\_main import integer\_lp  
from tkinter import ttk, filedialog  
import tkinter as tk  
from tkinter import messagebox  
import os.path

**По порядку о них:**

**ctypes, sys, numpy, os** – это все стандартные библиотеки и они были использованы по одному-несколько раз для работы с файлами.

if sys.platform == 'win32':  
 ctypes.windll.shcore.SetProcessDpiAwareness(1)

**Перевод кода:** если ОС – Шиндовс, то нужно применить хак, чтобы окно формы не замыливалось, как это бывает со старыми приложениями. Остальные случаи даже и не думаю, что стоит упоминать, настолько они незначительные

**PIL –** библиотека для помещения изображений на форму. Использовалась, чтобы можно было менять размер изображений прямо внутри программы.

import app.user\_form\_support  
from app.int\_linear\_main import integer\_lp

- так мы обращаемся к нашему вспомогательному дурацкому файлу user\_form\_support и нашему важному файлу **int\_linear\_main.py,** а точнее даже к главной функции внутри него - **integer\_lp.**

Обращение происходит по клику на кнопку **«Рассчитать»**  на форме **(self.exe\_button\_p1),**

self.exe\_button\_p1.configure(command=lambda: call\_linear\_prog(filepath.get(), zadacha=1))

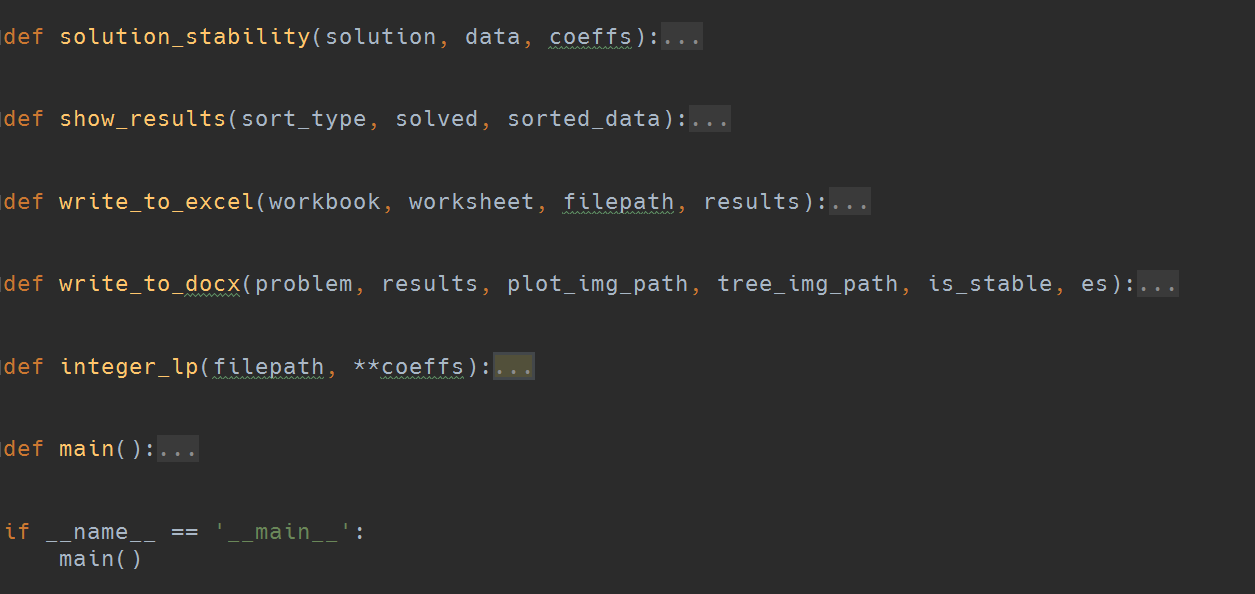
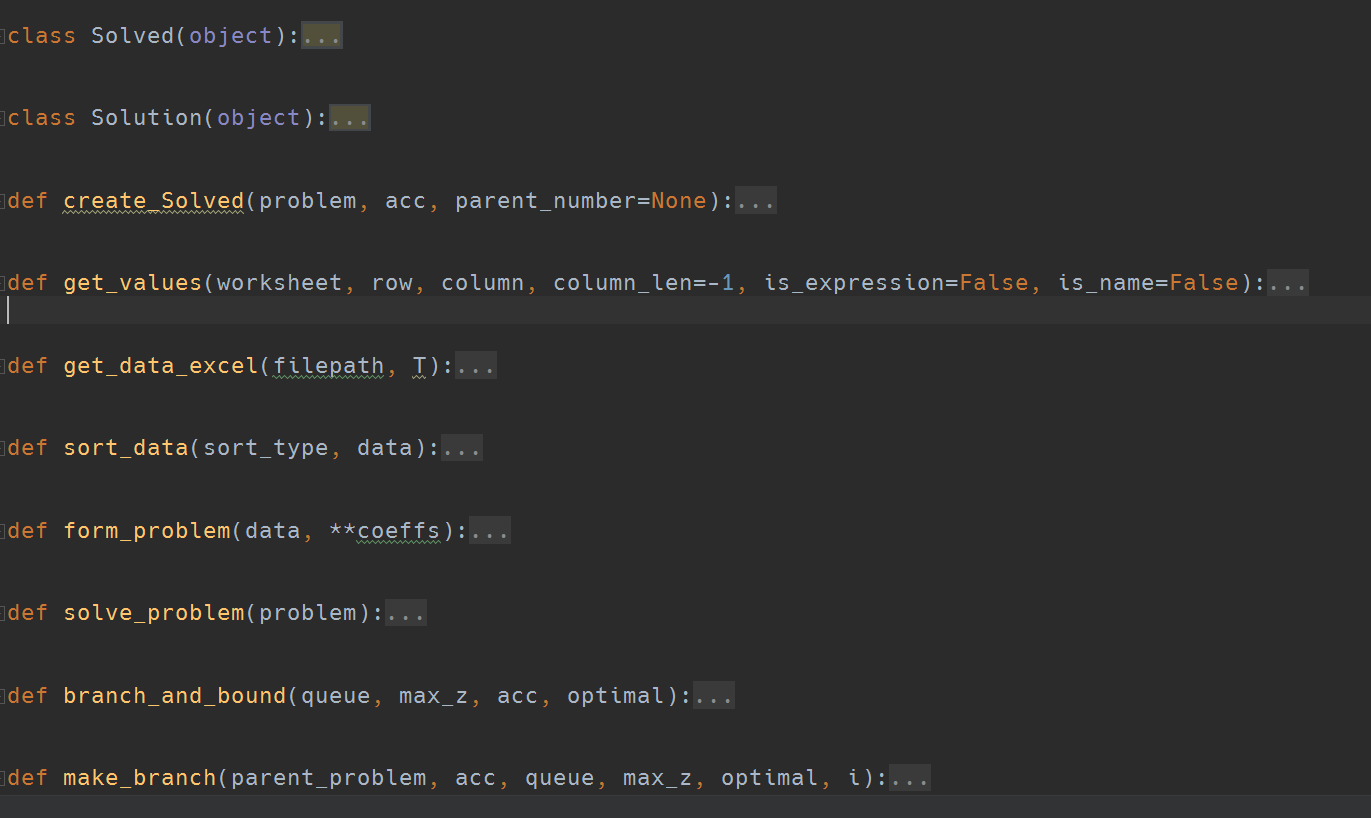
через функцию **call\_linear\_prog**:

def call\_linear\_prog(filepath, zadacha):  
 if filepath == '':  
 raise Exception(  
 messagebox.showinfo('Ошибка', 'Вы не загрузили таблицу исходных данных')  
 )  
 else:  
 coeffs = {'sort': self.sort\_var.get(),  
 'zadacha': zadacha}  
 if zadacha == 1:  
 try:  
 coeffs['T'] = int(self.T\_entry\_p1.get())  
 coeffs['F'] = float(self.F\_entry\_p1.get())  
 coeffs['stable'] = self.sol\_stability.get()  
 except ValueError:  
 messagebox.showinfo('Ошибка', 'Неверное введены коэффициенты')  
 raise  
 elif zadacha == 2:  
 try:  
 coeffs['T'] = int(self.T\_entry\_p2.get())  
 coeffs['F'] = float(self.F\_entry\_p2.get())  
 coeffs['y'] = float(self.y\_entry\_p2.get())  
 if self.auto\_D.get() == 1:  
 Ds = [k \* coeffs['F'] for k in linspace(0.1, 1.5, 15)]  
 coeffs['auto\_D'] = Ds  
 else:  
 coeffs['D'] = float(self.D\_entry\_p2.get())  
 except ValueError:  
 messagebox.showinfo('Ошибка', 'Неверное введены коэффициенты')  
 raise  
 integer\_lp(filepath, \*\*coeffs)

Она, **во-первых**, проверяет форму на ошибки (не загружена таблица Excel, неверно введены коэффициенты), **а во-вторых**, забирает коэффициенты с формы.

# 4.2 int\_linear\_main.py

В этом файле происходят все вычисления. Вот список функций в этом файле:



Вот так выглядит его главная функция **integer\_lp**, которая по порядку вызывает все остальные функции в этом файле:

def integer\_lp(filepath, \*\*coeffs):  
 plot\_img\_path = None  
 is\_stable = False  
 data, workbook, worksheet = get\_data\_excel(filepath, coeffs['T'])  
 sorted\_data = sort\_data(coeffs['sort'], data)  
 # если автоподбор параметра D  
 if 'auto\_D' in coeffs:  
 # решаем проблему с первым значением D и убираем его из списка  
 coeffs['D'] = coeffs['auto\_D'].pop(0)  
 problem = form\_problem(sorted\_data, \*\*coeffs)  
 solution\_problem = solve\_problem(problem)  
 solution\_problem.coeff\_D = coeffs['D']  
 if not solution\_problem.has\_sol:  
 solution\_problem.func\_value = 0  
 for d in coeffs['auto\_D']:  
 coeffs['D'] = d  
 problem = form\_problem(sorted\_data, \*\*coeffs)  
 solved\_problem = solve\_problem(problem)  
 if solved\_problem.has\_sol and solved\_problem.func\_value > solution\_problem.func\_value:  
 solution\_problem = solved\_problem  
 solution\_problem.coeff\_D = d  
 else:  
 problem = form\_problem(sorted\_data, \*\*coeffs)  
 solution\_problem = solve\_problem(problem)  
 # если стабильность решения и True  
 if 'stable' in coeffs and coeffs['stable']:  
 is\_stable = True  
 if solution\_problem.has\_sol:  
 plot\_img\_path, es = solution\_stability(solution\_problem, sorted\_data, coeffs)  
 results, tree\_img\_path = show\_results(coeffs['sort'], solution\_problem, sorted\_data)  
 write\_to\_excel(workbook, worksheet, filepath, results)  
 try:  
 answer = messagebox.askyesno('Решение', "\n".join(results) +  
 '\n\nХотите сохранить подробный результат\nв DOCX файл?')  
 if answer:  
 write\_to\_docx(problem, results, plot\_img\_path, tree\_img\_path, is\_stable, es)  
 except PermissionError:  
 messagebox.showinfo('Ошибка', 'Доступ к выбранному файлу невозможен. Может быть,'  
 'Вы пытаетесь переписать открытый файл')  
 # удаляю временные файлы, которые закидываются в doc с результатами, графика может не быть  
 os.remove(tree\_img\_path)  
 try:  
 os.remove(plot\_img\_path)  
 except TypeError:  
 pass

1. получаем книгу и лист Excel, с которыми работаем и данные с этого листа в функции get\_data\_excel
2. Затем сортируем эти данные с помощью сортировки, которую указали на форме
3. Если нам нужно подобрать коэффициент D, то есть мы указали это на форме, то тогда начинается ветка, которая решает все проблемы с разными D подряд и по ходу дела обновляет переменную с функцией, обладающей максимальным значением **(solution\_problem).** Ее мы используем дальше.
4. Если нам не нужен D, то мы формируем одну проблему с помощью функции **form\_problem**
5. Затем решаем ее в функции **solve\_problem** и получаем **solution\_problem**. Как ты могла заметить, в ветке с разными D мы точно так же формируем и решаем проблему, теми же функциями, только делаем это столько раз, сколько D
6. Если на форме мы отметили галочкой, что хотим посчитать стабильность решения, и у **solution\_problem** есть решение, запускается функция **solution\_stability**, которая возвращает путь к картинке с графиком (**plot\_img\_path**) и набор значений инфляции, при которых меняется оптимальное решение (**es**).
7. Формируем результаты наших вычислений в **show\_results**, создаем картинку с деревом решений (**tree\_img\_path**) там же. **results** используются в функции **write\_to\_excel**, **write\_to\_docx** и показываются в сообщении с результатами:

messagebox.askyesno('Решение', "\n".join(results)..

1. Пишем результаты в док, если нажали ОК в сообщении с результатами

if answer:  
 write\_to\_docx(problem, results, plot\_img\_path, tree\_img\_path, is\_stable, es)

1. В конце удаляем временные картинки с графиком и деревом решений, так как **write\_to\_docx** уже закинула их в документ

# 4.2.1 Классы

class Solved(object):  
  
 def \_\_init\_\_(self, problem, number, func\_value, vars\_value, status,  
 cont\_var=None, cont\_var\_value=None, parent\_number=None):  
 self.problem = problem  
 self.status = status  
 self.number = number  
 self.func\_value = func\_value  
 self.vars\_value = vars\_value  
 self.cont\_var = cont\_var  
 self.cont\_var\_value = cont\_var\_value  
 self.parent\_number = parent\_number  
 self.make\_node()  
  
 def \_\_repr\_\_(self):  
 return str(self.number)  
  
 # чтобы решить проблему с pyinstaller  
 if platform.system() == 'Windows':  
 solver = COIN\_CMD(path=os.path.join(os.getcwd(), 'solver\\win\\cbc.exe'))  
 else:  
 solver = COIN\_CMD(path=os.path.join(os.getcwd(), 'solver/linux/cbc'))  
 tree = []  
 statuses = ['Не решено', 'Оптимально', 'Неопределенно', 'Не ограниченно', 'Нерешаемо']  
  
 def nodenamefunc(node):  
 node\_params = [str(node.name), node.status, str(node.func\_value), \*node.xs]  
 return "\n".join(node\_params)  
  
 def make\_node(self):  
 status = Solved.statuses[self.status]  
 xs = [str(x[0]) + ' = ' + str(x[1]) for x in zip(self.problem.variables(), self.vars\_value)]  
 new\_node = Node(name=self.number, status=status, xs=xs,  
 func\_value=self.func\_value, parent\_name=self.parent\_number)

# ищем родителя для new\_node  
 for node in self.tree:  
 if node.name == new\_node.parent\_name:  
 new\_node.parent = node  
 self.tree.append(new\_node)

Класс, который построен на основе класса LpProblem из билиотеки PuLP (библиотека, которая, собственно, является интерфейсом к решателю линейных задач). Точнее даже, он **служит оберткой для него**. В классе LpProblem есть уже все нужные свойства решенной проблемы (статус, значение функции, значения переменных), но беда в том, что там есть большой сюрприз, который мне будет лучше **объяснить тебе по звонку**.

В общем, пришлось создать свой класс, в котором содержится сама **решенная** **проблема**, **класса LpProblem** и отдельно ее параметры: значение функции и значение переменных. Плюс здесь есть свойства, которых **нет в LpProblem:**

**Cont\_var –** нецелочисленная переменная в иксах, ее название

**Cont\_var\_value –** значение этой переменной

**Number –** номер решенной проблемы

**Parent\_number -** номер родителя проблемы (то есть та проблема, от которой она ветвится)

**Функция make\_node() –** она создает из проблемы узел в дереве, которое потом строитсяв картинку и **добавляется в DOCX с результатами**

Функция **create\_Solved** используется для (огого) создания объекта класса Solved:

def create\_Solved(problem, acc, parent\_number=None):

# здесь происходит решение проблемы с помощью библиотеки PuLP  
 problem\_copy = problem.deepcopy()  
 problem\_copy.solve(Solved.solver)  
 acc += 1  
 # создаем объект решенной задачи  
 solved = Solved(problem=problem,  
 status=problem\_copy.status,  
 number=acc,  
 func\_value=value(problem\_copy.objective),  
 vars\_value=[var.varValue for var in problem\_copy.variables()],  
 parent\_number=parent\_number)  
 for v in problem\_copy.variables():  
 if v.varValue != int(v.varValue):  
 solved.cont\_var, solved.cont\_var\_value = v, v.varValue  
 break  
 return solved, acc

Можно было бы обойтись и без нее, то есть создавать объект класса **Solved,** вызывая сам класс, но тогда код класса был бы еще больше.

Solution

class Solution(object):  
 def \_\_init\_\_(self, acc, solution=None, optimal\_problems=[], auto\_coeff\_D=False):  
 self.acc = acc  
 self.coeff\_D = auto\_coeff\_D  
 self.has\_sol = False  
 if solution is not None:  
 for x in solution.vars\_value:  
 if x != 0:  
 self.has\_sol = True  
 self.optimal\_problems = optimal\_problems  
 self.variables = solution.problem.variables()  
 self.func\_value = solution.func\_value  
 self.vars\_value = solution.vars\_value  
 self.number = solution.number  
 break  
  
 def \_\_repr\_\_(self):  
 return str(self.number)

этот класс создается, когда найдена **оптимальная проблема, либо решение не найдено.**

# возвращаем задачу, если она не оптимальна  
if first.status != 1:  
 return Solution(acc=acc, solution=None)

...

# если проблема целочислена на первом шаге  
else:  
 return Solution(acc=acc, solution=first, optimal\_problems=[first])

if optimal:  
 solution = optimal[0]  
 for prob in optimal[1:]:  
 if prob.func\_value > solution.func\_value:  
 solution = prob  
 return Solution(acc=acc, solution=solution, optimal\_problems=optimal)  
else:  
 return Solution(acc=acc, solution=None)

Объект этого класса хранит в себе **оптимальную задачу класса Solved,** если она имеется в данном решении, значение коэффициента D, номер задачи, значение функции, значение переменных, флажок **has\_sol** (имеется ли оптимальная проблема) и набор других целочисленных решенных проблем **класса Solved**, полученных в этом решении (**optimal\_problems**). Все эти значения используются при формировании результатов. **Optimal\_problems** используется при расчете стабильности решения.

# 4.2.2 Про дерево решений:

Дерево решений формируется библиотекой **anytree.** Как было сказано выше, узлы этого дерева формируются при создании объекта класса Sovled, с помощью функций

def make\_node(self):  
 status = Solved.statuses[self.status]  
 xs = [str(x[0]) + ' = ' + str(x[1]) for x in zip(self.problem.variables(), self.vars\_value)]  
 new\_node = Node(name=self.number, status=status, xs=xs,  
 func\_value=self.func\_value, parent\_name=self.parent\_number)  
 for node in self.tree:  
 if node.name == new\_node.parent\_name:  
 new\_node.parent = node  
 self.tree.append(new\_node)

это функция формирующая объект класса **anytree.Node,** со статусом проблемы, значениями иксов и значением функции)

def nodenamefunc(node):  
 node\_params = [str(node.name), node.status, str(node.func\_value), \*node.xs]  
 return "\n".join(node\_params)

это функция отвечающая за данные в узле дерева, которые будут отображаться на картинке (эта функция из библиотеки **anytree,** просто ее **можно изменять)**

Этой строкой кода дерево решений преобразуется в картинку:

tree\_img\_path = 'results/temp\_tree.png'  
DotExporter(Solved.tree[0], nodenamefunc=Solved.nodenamefunc).to\_picture(tree\_img\_path)

Кстати, само дерево (**tree**), хранится в классе **Solved.** Оттуда мы его и берем.

! **мы устанавливаем graphviz (наш визуализатор), потому что библиотека pyinstaller, создающая из нашей программы exe, не пакует встроенный в библиотеку anytree graphviz. Единственный способ это обойти, это установить graphviz себе на компьютер в путь**

**C:\Program Files (x86)\Graphviz2.38**

**Этот путь так же является стандартным в установщике graphviz. Иначе картинка с деревом решений создаваться не будет(**

**Вот строка кода, которая говорит нашей программе, что graphviz находится в месте установки:**

# чтобы прописать graphviz в PATH  
os.environ["PATH"] += os.pathsep + r'C:\Program Files (x86)\Graphviz2.38\bin'

# 4.2.3 Про стабильность решений:

Стабильность решений рассчитывается по алгоритму, который ты мне предоставила, и визуализируется с помощью библиотеки **matplotlib. Это очень популярная библиотека для создания графиков.**

def solution\_stability(solution, data, coeffs):  
  
 def minimum\_e(optimal\_list, e\_min\_problem, e\_min\_prob\_index, es):  
 *"""  
 Эта функция находит интервалы значений инфляции (е), в рамках которых  
 оптимальный портфель закупок сохраняется. Первый шаг расчета начинается с  
 оптимальной проблемы, у которой е = 0.  
 """* e\_min = float('inf')  
 x\_l = e\_min\_problem.vars\_value  
 l = e\_min\_prob\_index  
 n = len(optimal\_list)  
 for i in range(l + 1, n):  
 x = optimal\_list[i].vars\_value  
 e = (  
 (  
 sum(x\_l[i] \* (data['beta'][i] - data['alpha'][i]) for i in range(0, len(x))) -  
 sum(x[i] \* (data['beta'][i] - data['alpha'][i]) for i in range(0, len(x)))  
 ) /  
 (  
 sum(x[i] \* data['beta'][i] for i in range(0, len(x))) -  
 sum(x\_l[i] \* data['beta'][i] for i in range(0, len(x)))  
 )  
 )  
 if 0 < e < e\_min:  
 e\_min = e  
 e\_min\_problem = optimal\_list[i]  
 e\_min\_prob\_index = i  
 if l != n - 1 and e\_min != float('inf'):  
 es[e\_min] = e\_min\_problem  
 return minimum\_e(optimal\_list, e\_min\_problem, e\_min\_prob\_index, es)  
 else:  
 return es  
  
 def make\_stability\_plot(es, data, coeffs):  
 # нужно построить функцию от неизвестного e, имея две точки: значение функции при e = 0  
 # и при e, пересекающем оптимальную функцию. По сути, нужно только посчитать  
 # значение целевой функции в этой точке и построить графики. Нужно использовать  
 # проблемы l+1  
 alpha = data['alpha']  
 v = data['v']  
 F = coeffs['F']  
 n = len(alpha)  
  
 x\_opt = es[0].vars\_value  
 opt\_number = es[0].number  
 beta = [b \* 2 for b in data['beta']]  
 sum\_var1 = sum([x\_opt[i] \* v[i] \* beta[i] for i in range(0, n)])  
 sum\_var2 = sum([x\_opt[i] \* v[i] \* alpha[i] for i in range(0, n)])  
 opt\_zero\_f\_val = es[0].func\_value  
 opt\_one\_f\_val = sum\_var1 + F - sum\_var2  
 fig, ax = plt.subplots(nrows=1, ncols=1)  
 ax.plot([0, 1], [opt\_zero\_f\_val, opt\_one\_f\_val], label='Оптимальная задача: ' + str(opt\_number))  
 del es[0]

# если точки смены оптимального решения  
 if es:  
 intersection\_f\_list = []  
 zero\_f\_list = [problem.func\_value for problem in es.values()]  
 for e, problem in es.items():  
 x = problem.vars\_value  
 beta = [b \* (1 + e) for b in data['beta']]  
 sum\_var1 = sum([x[i] \* v[i] \* beta[i] for i in range(0, n)])  
 sum\_var2 = sum([x[i] \* v[i] \* alpha[i] for i in range(0, n)])  
 intersection\_f\_list.append(sum\_var1 + F - sum\_var2)  
 for zero\_f, cross\_f, e, problem in zip(zero\_f\_list, intersection\_f\_list, es, es.values()):  
 ax.plot([0, e, 2\*e], [zero\_f, cross\_f, cross\_f\*2],  
 label='Целочисленная задача: ' + str(problem.number))  
 ax.plot([e, e], [0, cross\_f], ':')  
  
 ax.set\_ylabel('Значение целевой функции от инфляции, F(e)')  
 ax.set\_xlabel('Инфляция в долях, e')  
 ax.set\_title('Устойчивость решения')  
 ax.legend()  
 ax.grid()  
 plot\_path = 'results/temp\_plot.png'  
 fig.savefig(plot\_path, bbox\_inches='tight')  
 return plot\_path, es  
  
 optimal\_list = solution.optimal\_problems  
 optimal\_list.sort(key=lambda problem: sum(problem.vars\_value[i] \* data['beta'][i] for i in range(0, len(data['beta']))))  
 sol\_num = solution.number  
 index\_list = [x.number for x in optimal\_list]  
 sol\_index = index\_list.index(sol\_num) if sol\_num in index\_list else None  
 es = {0: solution}  
 if len(optimal\_list) > 1:  
 es = minimum\_e(optimal\_list, solution, sol\_index, es)  
 return make\_stability\_plot(es, data, coeffs)

Этими строками мы сохраняем полученный график

plot\_path = 'results/temp\_plot.png'  
fig.savefig(plot\_path, bbox\_inches='tight')