Министерство образования и науки России

Федеральное государственное бюджетное   
образовательное учреждение высшего образования  
 «Сыктывкарский государственный университет имени Питирима Сорокина»  
(ФГБОУ ВО «СГУ им. Питирима Сорокина»)  
Институт точных наук и информационных технологий  
Кафедра математического моделирования и кибернетики

Допустить к защите  
Зав. кафедрой математического  
моделирования и кибернетики  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ к.ф.-м.н. Беляев Ю.Н.   
«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г.

Выпускная квалификационная работа

**Метод обобщённой реакции**

|  |  |
| --- | --- |
| Исполнитель: обучающийся 149 группы Грабовский А.А. |  |
|  |  |
| Научный руководитель: к.ф.-м.н., доцент, Ермоленко А.В. |  |

Сыктывкар 2017

**Аннотация**

Разработано программное обеспечение для построения поверхностей функций прогиба и реакции прямоугольной пластины по классической теории. Проанализирован способ нахождения численного решения по теории Кармана. В работе использовано два языка программирования для разных задач: C++ для вычисления матриц прогиба и реакции пластины, Python 3 для создания графического интерфейса и построения поверхностей.

Данная работа апробирована на молодёжной научной конференции, посвящённой памяти Фролова.

**Annotation**

A software was developed for constructing the functions of the deflection functions and the reaction of a rectangular plate according to the classical theory. The method of finding a numerical solution in the Karman's theory is analyzed. The work uses two programming languages for different tasks: C ++ for computing the deflection matrices and plate response, Python 3 for creating a graphical interface and constructing surfaces.

This work was approved at a youth scientific conference dedicated to the memory of Frolov.

**Содержание**

Введение…………………………………………………………………………..4

1. Обзор теории пластин…………………………………………………….......5
   1. Основные уравнения теории упругости……………………………....... 5
   2. Основные понятия и гипотезы………………………………………….. 7
   3. Перемещения и деформации в пластине……………………………......8
   4. Напряжения в пластине…………………………………………………10
   5. Усилия в пластине…………………………………………………….. ..10
   6. Дифференциальное уравнение изогнутой срединной поверхности пластины……………………………………………………………….... 11
2. Обзор необходимых сведений из численных методов……………………14
   1. Численное дифференцирование……………………………………….. 14
   2. Способы нахождения обратной матрицы………………………….......15
3. Метод обобщённой реакции……………………………………………….. 17
4. Задача о расчёте прогиба прямоугольной пластины по классической теории с постоянной нагрузкой…………………………………………….18
5. Задача о нахождении прогиба прямоугольной пластины по нелинейной теории……………………………………………………………………....... 21
   1. Постановка задачи…………………………………………………….....21
   2. Решение по классической теории………………………………………21
   3. Решение по теории Кармана………………………………………….... 21

Численный эксперимент…………………..........................................................23

Заключение…………………………………………………………................... 26

Источники………………………………………………………………………. 27

Приложение…………………………………………………………………….. 28

**Введение**

Строительство является важнейшей отраслью любого государства. Поэтому исследование строительных конструкций, в которых применяются различные виды пластин, является актуальной задачей. В связи с тем, что не всегда удаётся найти аналитическое решение чаще стали применять численные методы [4]. Их использование стало возможным благодаря росту производительности процессоров. В данной работе описывается нахождение численного решения прогиба прямоугольной пластины методом обобщённой реакции [1].

Объект исследования – это прямоугольная пластина, жёстко закреплённая по контуру. Предметом исследования является прогиб пластины при малых деформациях.

Цель – создать программное обеспечение для построения поверхностей функций прогиба и реакции прямоугольной пластины по классической и обобщённой теориям.

Задачи:

1. Изучение литературы.
2. Нахождение способа вычисления численного решения.
3. Составление итерационных схем для классической и обобщённой теорий.
4. Выбор языка программирования.
5. Написание программного кода.
6. Тестирование программы.
7. Построение поверхностей.

Введение раскрывает актуальность, определяет объект, предмет, цель и задачи исследования. В первом разделе выводится уравнение Софи Жермен–Лагранжа, используя основные уравнения теории упругости. Во втором разделе затрагиваются некоторые сведения из численных методов. В третьем разделе описывается метод обобщённой реакции. В четвёртом разделе рассматривается численное решение уравнения Софи Жермен–Лагранжа для постоянной нагрузки, которое используется в начальном приближении итерационной схемы. В пятом разделе поясняется нахождение численного решения по классической и обобщённой теориям. В разделе численный эксперимент рассказывается о результатах расчетов, и делаются выводы. В заключении подводятся итоги проделанной работы и даются рекомендации.

**1. Обзор теории пластин**

**1.1 Основные уравнения теории упругости**

Рассмотрим основные группы формул из теории упругости.

1. Дифференциальные уравнения равновесия

(1.1)

где – объёмные силы, действующие в каждой точке тела,

– тензор напряжений. (1.2)

Тензор напряжений элементарного объёма тела выглядит следующим образом

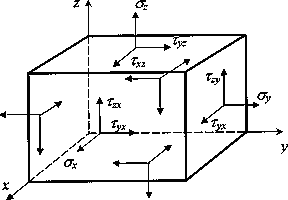


Рис. 1.1. Тензор напряжений элементарного объёма тела

1. Формулы Коши для линейных деформаций

(1.3)

Эти уравнения связывают деформации элемента тела и функции , выражающие перемещения его точек. Тензор деформаций записывается в следующем виде

. (1.4)

1. Обобщённый закон Гука

(1.5)

где

– модуль сдвига, (1.6)

*E* – модуль Юнга, *–* коэффициент Пуассона.

В перечисленных уравнениях содержатся 15 неизвестных функций: , , , , , , , , , , , , , , . Для нахождения этих функций используются уравнения (1.1), (1.3), (1.5). Таким образом, для решения задачи требуется проинтегрировать эти уравнения.

Решения указанных уравнений находятся в зависимости от того, какие величины были приняты за основные неизвестные:

1. Решение в перемещениях, если основными неизвестными являются функции, выражающие перемещения.
2. Решение в напряжениях, если основными неизвестными являются 6 компонент тензора напряжений.
3. Решение в смешанной форме, если основными неизвестными являются некоторые функции, выражающие перемещения и некоторые компоненты тензора напряжений.

**1.2 Основные понятия и гипотезы**

Пластиной называется тело, ограниченное двумя параллельными плоскостями, расстояние между которыми, называемое толщиной пластины , мало по сравнению с его другими размерами.

Плоскость, делящая пластину пополам по толщине, называется срединной плоскостью. При изгибе пластины срединная плоскость превращается в изогнутую срединную поверхность пластины.

Линия пересечения боковой поверхности пластины со срединной плоскостью называется контуром пластины.

Для исследования деформаций пластины прямоугольную систему координат будем располагать так, чтобы координатная плоскость *xOy* совпала со срединной плоскостью пластины. Ось *z* будем направлять вниз. При таком выборе системы координат функция *w*, выражающая перемещения в направлении оси *z*, будет представлять собой прогиб пластины. Положение начала координат в срединной плоскости будем выбирать в каждом рассматриваемом случае в зависимости от очертания контура пластины и характера закрепления её краёв.

Тонкими называются пластины, имеющие отношения толщины к наименьшему характерному размеру в следующих пределах: и величину ожидаемых прогибов не более ¼ *h*.

Пластины, у которых , рассчитываются по теории толстых плит, а пластины, имеющие прогибы более ¼ *h* рассчитываются по геометрически нелинейной теории гибких пластин или мембран.

Теория тонких пластин основана на следующих гипотезах, предложенных Кирхгофом:

1. Гипотеза прямых нормалей: любой линейный элемент, нормальный к срединной плоскости пластины, остаётся прямолинейным и нормальным к ней и после деформации, причём длина его не изменяется.

Любой линейный элемент, нормальный к срединной плоскости, направлен вдоль оси *z*, и, следовательно, первая часть гипотезы предполагает, что прямые углы между этим элементом и осями *x* и *y* остаются прямыми, т.е. сдвиги в указанных плоскостях отсутствуют:

(1.7)

Допущение о сохранении длины прямолинейного элемента предполагает, что линейная деформация в направлениях оси *z* отсутствует:

(1.8)

1. Гипотеза о нерастяжимости срединной плоскости: в срединной плоскости отсутствуют линейные относительные деформации и деформации сдвига, т.е. срединная плоскость является нейтральной. Следовательно, в срединной плоскости перемещения отсутствуют

(1.9)

1. Гипотеза об отсутствии давления между слоями пластины. Ввиду малости давления между слоями пластины, параллельными срединной плоскости, напряжением по сравнению с напряжениями и можно пренебрегать.

**1.3 Перемещения и деформации в пластине**

Рассмотрим пластину с нормальной нагрузкой к срединной плоскости. Под действием этой нагрузки пластина получит перемещения. Для их определения воспользуемся гипотезами Кирхгофа.

Согласно первой гипотезе линейная деформация в направлении оси *z* равна нулю (1.7). Подставляя это условие в третью формулу Коши (1.3) получаем:

(1.10)

Откуда следует, что прогибы пластины *w* не зависят от координаты *z*, т.е. Это означает, что все точки пластины, лежащие на одной вертикали, получают одинаковые прогибы. Следовательно, достаточно определить прогибы срединной плоскости пластины, чтобы узнать прогибы всех её точек.

Рассматривая условия для сдвигов (1.7), из формул Коши (1.3) получаем, что , . Откуда находим производные от функций *u* и *v*, выражающие перемещения по координате *z*:

, (1.11)

. (1.12)

Интегрируя эти уравнения по *z*, получаем:

. (1.13)

Для вычисления функций и воспользуемся гипотезой о нерастяжимости срединной плоскости. Согласно этой гипотезе составляющие перемещения и на срединной плоскости при *z=0* равны нулю. Подставляя эти условия в формулы (1.13), получаем:

, (1.14)

. (1.15)

С учётом формул (1.14), (1.15) формулы (1.13) примут следующий вид:

. (1.16)

Таким образом, функции, выражающие перемещения точек пластины в направлениях осей х и у, выражены через функцию прогибов срединной плоскости пластины.

Компоненты тензора деформаций, отличные от нуля, найдем с помощью формул Коши (1.3), подставив в них значения функций (1.16):

(1.17)

**1.4 Напряжения в пластине**

Для вычисления нормальных напряжений и возьмём две первые формулы закона Гука (1.5) и на основании третьей гипотезы Кирхгофа отбросим напряжение . Тогда получим

, (1.18)

. (1.19)

Откуда с учётом формул (1.17) находим

, (1.20)

. (1.21)

Четвёртая формула закона Гука (1.5) после подстановки угловой деформации из формул (1.17) примет такой вид:

. (1.22)

**1.5 Усилия в пластине**

Исследуем, какие усилия создаются напряжениями в сечениях пластины, нормальных к её срединной плоскости. Подсчитаем изгибающий момент. Обозначим через погонный (приходящийся на единицу ширины сечения) изгибающий момент в сечении с нормалью *x*. Изгибающий момент в рассматриваемом сечении создаётся нормальными напряжениями . Равнодействующая этих напряжений на площадке толщиной *dz* и шириной, равной единице, равна , а изгибающий момент . Суммируя моменты от напряжения на всех таких площадках по толщине пластины, получаем выражение для погонного изгибающего момента в сечении с нормалью х:

. (1.23)

Подставляя сюда значение нормального напряжения из формулы (1.20) и вынося за знак интеграла величины, не зависящие от координаты z, находим:

. (1.24)

После интегрирования получаем:

*.*  (1.25)

Входящая сюда величина

(1.26)

называется цилиндрической жёсткостью пластины и является физической и геометрической характеристикой пластины при ее изгибе.

Аналогично найдём погонный изгибающий момент в сечении с нормалью *y*, подставляя формулу (1.21):

. (1.27)

Погонный крутящий момент в сечении с нормалями *х* и *y* равен:

. (1.28)

После подстановки касательного напряжения из формулы (1.22) и интегрирования находим:

. (1.29)

**1.6 Дифференциальное уравнение изогнутой срединной поверхности пластины**

Вырежем из срединной плоскости пластины бесконечно малый элемент *dxdy* и покажем действующие на него нагрузки (Рис. 1.2)*.*

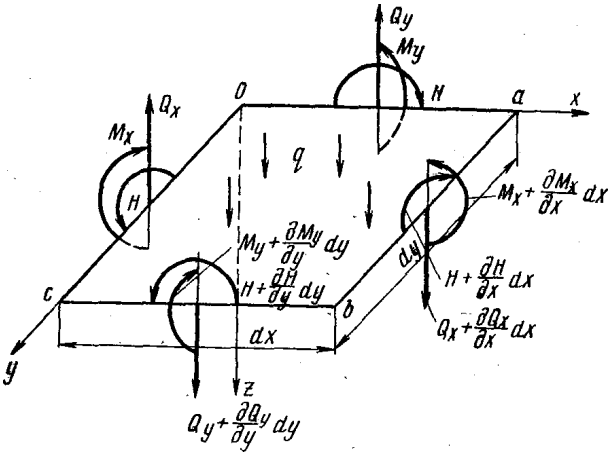


Рис. 1.2. Нагрузки, действующие на бесконечно малый элемент пластины

На грани элемента срединной плоскости действует погонная поперечная сила . При проецировании эту погонную силу необходимо умножить на длину грани . На грани поперечная сила получает бесконечно малое приращение, величина которой равна *.* Аналогично получается на гранях . На срединную плоскость пластины действует нормальная поверхностная нагрузка *q*.

Рассматриваемый элемент срединной плоскости находится в равновесии, следовательно, должно выполняться условие равновесия, т.е. сумма всех внешних сил равна нулю и сумма всех моментов внешних сил равна нулю. А именно шесть условий равновесия: три уравнения проекций на координатные оси и три уравнения моментов относительно этих осей.

Спроектируем все силы на ось *z*:

. (1.30)

После упрощения получаем:

. (1.31)

Уравнение моментов всех сил относительно оси *y* дает:

. (1.32)

После упрощения получаем:

. (1.33)

Аналогично из уравнения моментов относительно оси *х* получаем:

. (1.34)

Исключим из уравнений (1.31), (1.33), (1.34) поперечные силы . В результате получим:

. (1.35)

Подставив в полученное уравнение (1.35) моменты из формул (1.25), (1.27) и (1.29), найдем:

. (1.36)

Откуда после упрощения получаем:

(1.37)

или

. (1.38)

Полученное уравнение представляет собой дифференциальное уравнение изогнутой срединной поверхности пластины, его обычно называют уравнением Софи Жермен–Лагранжа.

Уравнение (1.38) должно быть дополнено граничными условиями. Условия на контуре пластины зависят от характера закрепления ее краёв.

**2. Обзор необходимых сведений из численных методов**

* 1. **Численное дифференцирование**

Задача численного дифференцирования состоит в приближенном вычислении производных функции по заданным в конечном числе точек значениям этой функции [4]. Введём на плоскости равномерную сетку с шагами , т.е. множество точек

. (2.1)

Пусть определены значения функции в точках сетки. Существуют три основные разностные отношения для вычисления производных: левая конечная разность

, (2.2)

правая конечная разность

, (2.3)

и центральная конечная разность

. (2.4)

Все остальные разностные отношения выводятся из комбинаций уравнений (2.2)–(2.4). Например, получим разностное отношение для производной второго порядка

.

Выпишем разностные отношения, которые будем использовать в дальнейшем

(2.5)

(2.6)

(2.7)

(2.8)

(2.9)

. (2.10)

* 1. **Способы нахождения обратной матрицы**

Квадратная матрица порядка называется обратной, если существует невырожденная матрица того же порядка такая, что выполняется условие

,

где – единичная матрица порядка. Невырожденность матрицы означает, что её определитель не равен нулю, т.е. .

Далее нам понадобиться понятие элементарного преобразования строк (столбцов), к которым относятся: перестановка местами любых двух строк матрицы, умножение на ненулевую константу любой строки матрицы и прибавление к любой строке матрицы другой строки, умноженной на ненулевое число. Есть различные способы нахождения обратной матрицы. Рассмотрим три основных метода.

В методе Жордана–Гаусса нужно взять исходную матрицу и единичную матрицу одной размерности. Затем с помощью элементарных преобразований строк нужно привести матрицу к единичному виду. Те же самые операции необходимо применить и к единичной матрице . В результате матрица станет единичной, а матрица – обратной. Сложность алгоритма оценивается как .

Во втором методе используется следующая формула

*,*  (2.11)

где – присоединённая матрица, т.е. матрица, которая получается с помощью алгебраических дополнений для соответствующих элементов транспонированной матрицы. Сложность алгоритма - .

Третьим методом является LU – разложение. Методом исключения Гаусса расширенная матрица приводится к диагональному виду. В результате получаем расширенную матрицу , где – это верхняя треугольная матрица с единицами на главной диагонали, а – нижняя треугольная матрица. При умножении этих матриц получается исходная матрица . Обратная матрица находится по следующей формуле

. (2.12)

После умножения уравнения (2.12) на , а затем на получим два уравнения

, (2.13)

. (2.14)

Так как элементы, стоящие на главной диагонали матрицы равны единице, то у матрицы элементы с теми же индексами тоже равны единице. Это позволит вычислить часть неизвестных элементов обратной матрицы . Решая систему линейных алгебраических уравнений из формулы (2.13) находится первая часть матрицы , а из (2.14) – вторая часть. Сложность алгоритма оценивается как .

1. **Метод обобщённой реакции**

Метод обобщённой реакции решения контактных задач со свободной границей был предложен и обоснован Тарасовым В.Н. и Михайловским Е.И. [1]. Этот метод представляет собой итерационную схему, общий вид которой для уравнения Софи Жермен–Лагранжа записывается следующим образом

(3.1)

(3.2)

где – начальная нагрузка, – матрица прогиба, – матрица реакции,   
 – некоторая константа, – зазор между пластиной и жёстким основанием, – обратная матрица к матрице бигармонического оператора, нахождение которой описано в следующем разделе. Обратная матрица ищется с помощью метода Жордана–Гаусса.

– начальные приближения. (3.3)

Формула (3.2) называется положительной срезкой функции, которая записывается в следующем виде

. (3.4)

Уравнение (3.2) эквивалентно следующим условиям

– условие непроникновения, (3.5)

– условие односторонности связи, (3.6)

– условие дополняющей нежёсткости. (3.7)

Аналогично для системы Кармана, описанной в разделе 5.3, можно записать итерационную схему в следующем виде

(3.8)

. (3.9)

Реакция и начальные приближения вычисляются по формулам (3.2), (3.3).

1. **Задача о расчёте прогиба прямоугольной пластины по классической теории c постоянной нагрузкой**

Рассмотрим пластину, у которой прогиб, отсчитываемый от срединной плоскости, мал по сравнению с толщиной пластины. В основе исследования этих пластин в пределах упругих деформаций лежит следующее дифференциальное уравнение:

, (4.1)

вывод которого был описан в первом разделе.

Для заданной прямоугольной пластины, жёстко закрепленной по контуру, требуется найти прогиб и построить её поверхность. Искать решение будем методом сеток. Пластина нагружена равномерно распределённой нагрузкой . Математически постановка данной задачи имеет следующий вид:

(4.2)

Решение исходной краевой задачи (4.2) будем искать в виде таблицы значений в узлах сетки

(4.3)

На пластину нанесена сетка. Для аппроксимации производных будем использовать формулы (2.8)–(2.10).

Аппроксимируем формулами (2.2)–(2.3) условия закрепления пластины

, ,  
 ,  
 ,  
 ,  
 .

(4.4)

Подставим в краевую задачу (4.2) формулы (2.2)–(2.10). Получим разностную схему:

(4.5)

.

Уравнение (4.5) можно представить в виде следующего сеточного шаблона, изображённого на Рис. 4.1, если приравнять шаги сетки по .

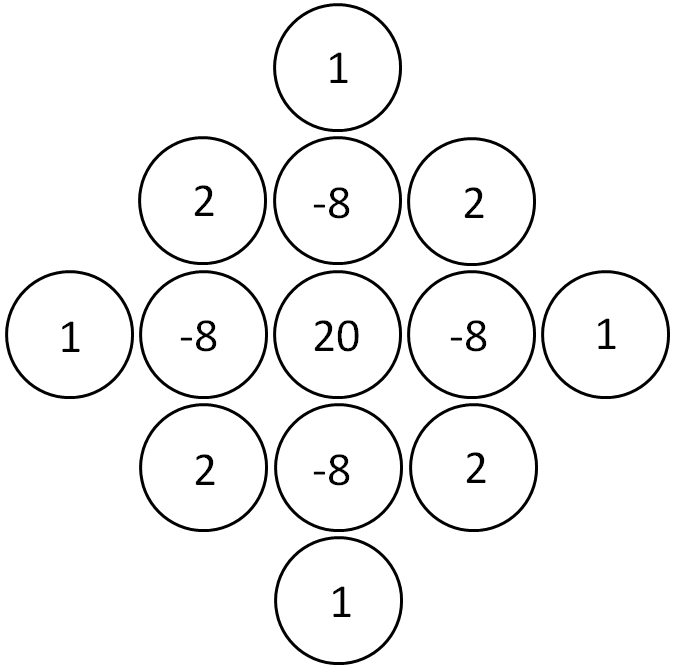


Рис. 4.1. Сеточный шаблон бигармонического оператора при

Этот шаблон нужно применить ко всем внутренним узлам (узлы нумеруются по порядку слева направо и сверху вниз), при этом центр шаблона должен совпадать с номером узла, к которому он применяется. Коэффициенты для узлов берутся из разностного уравнения (4.5). Например, для первого узла при и 9 внутренних узлах получим первую строку матрицы следующего вида

,

для второго узла 2-ю строку

,

и. т. д.

Таким образом, применив данный шаблон к каждому внутреннему узлу сетки, получим матрицу коэффициентов A размера . Например для 9 внутренних узлов данная матрица имеет следующий вид:



, (4.6)

A =

где

(4.7)

(4.8)

(4.9)

Формулу (4.1) запишем в матричном виде

. (4.10)

Умножив левую и правую части формулы (4.10) слева на обратную матрицу к матрице , получим формулу (3.3), которая определяет начальные приближения для задачи из следующего раздела.

1. **Задача о нахождении прогиба прямоугольной пластины по нелинейной теории**
   1. **Постановка задачи**

Аналогично предыдущему разделу требуется найти прогиб прямоугольной пластины жёстко закреплённой по контуру. Искать решение будем с помощью метода обобщённой реакции, описанного в разделе 3.

* 1. **Решение по классической теории**

Математически данная задача выглядит аналогично системе (4.2). Так как начальные приближения были найдены в предыдущем разделе, нам осталось воспользоваться формулами (3.1)–(3.3) для нахождения матриц прогиба и реакции пластины и построить поверхности. Данная задача была решена. Код программы представлен в приложении.

* 1. **Решение по теории Кармана**

Система Кармана записывается в следующем виде

, (5.1)

где – это функция напряжений Эри, получающаяся из условий

. (5.2)

Граничные условия функции напряжений записываются аналогично функции прогиба

(5.3)

Из формулы (5.3) после аппроксимации получается

,

,

,

(5.4)

,

.

Воспользуемся итерационной схемой, описанной формулами (3.2), (3.3), (3.8), (3.9). В данных формулах вторые частные производные от некоторой функции заменяются матричными записями

(5.5)

(5.6)

(5.7)

где или и

,

(5.8)

,

(5.9)

.

(5.10)

В итоге система (5.1) заменяется матричной записью

(5.11)

Для данной системы начальные приближения и реакция находятся также как и ранее.

Система (5.11) была запрограммирована, но что-то не заладился численный эксперимент.

**Численный эксперимент**

По классической теории были построены графики для материалов сталь и резина. Условием окончания расчётов было выбрано следующее условие

. (6)

При вычислении нормы использовалась евклидова норма. В формуле (6) производится деление на норму функции прогиба на к-ой итерации для получения безразмерной величины. Были получены следующие графики.

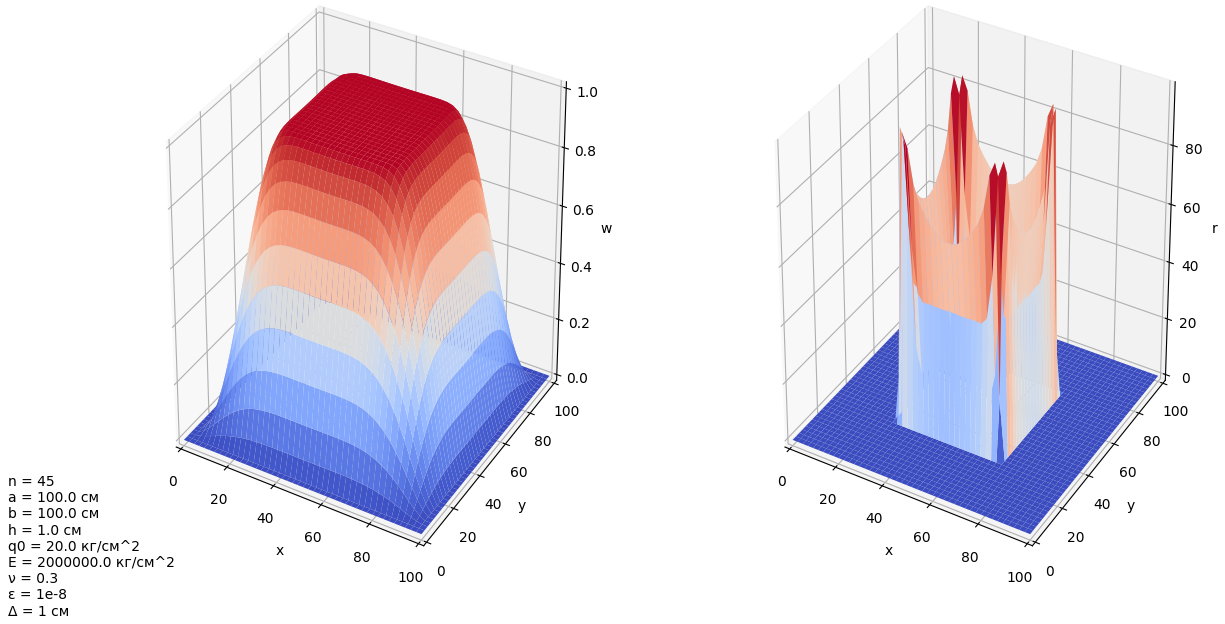
****

Рис. 6.1. Прогиб и реакция пластины. Материал сталь

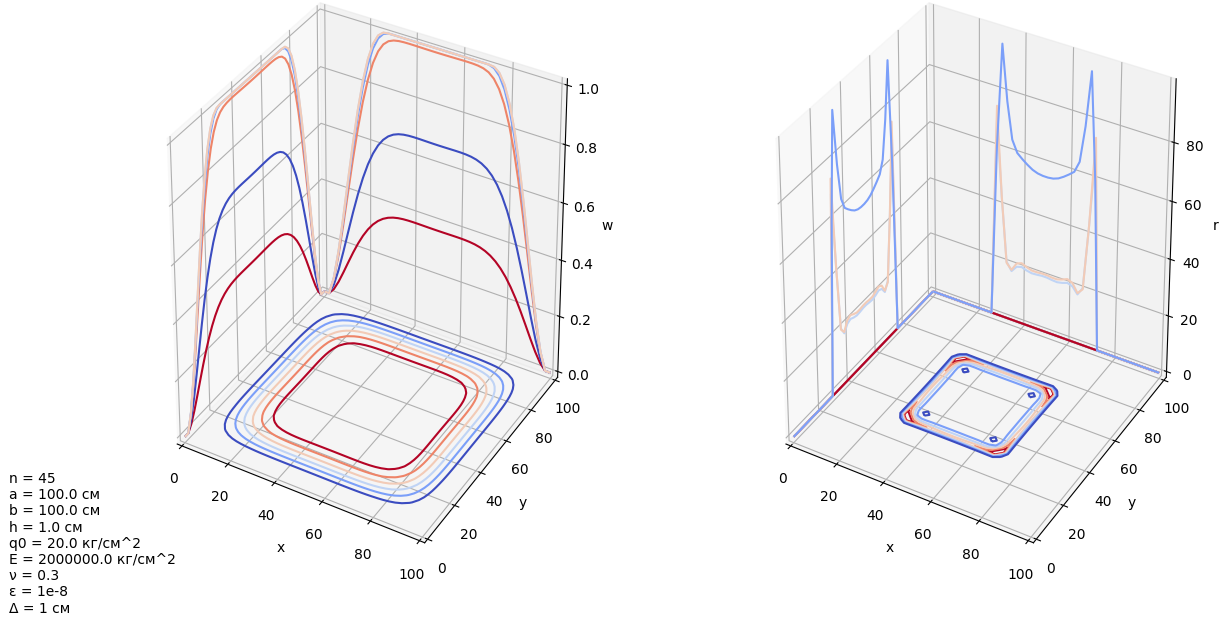
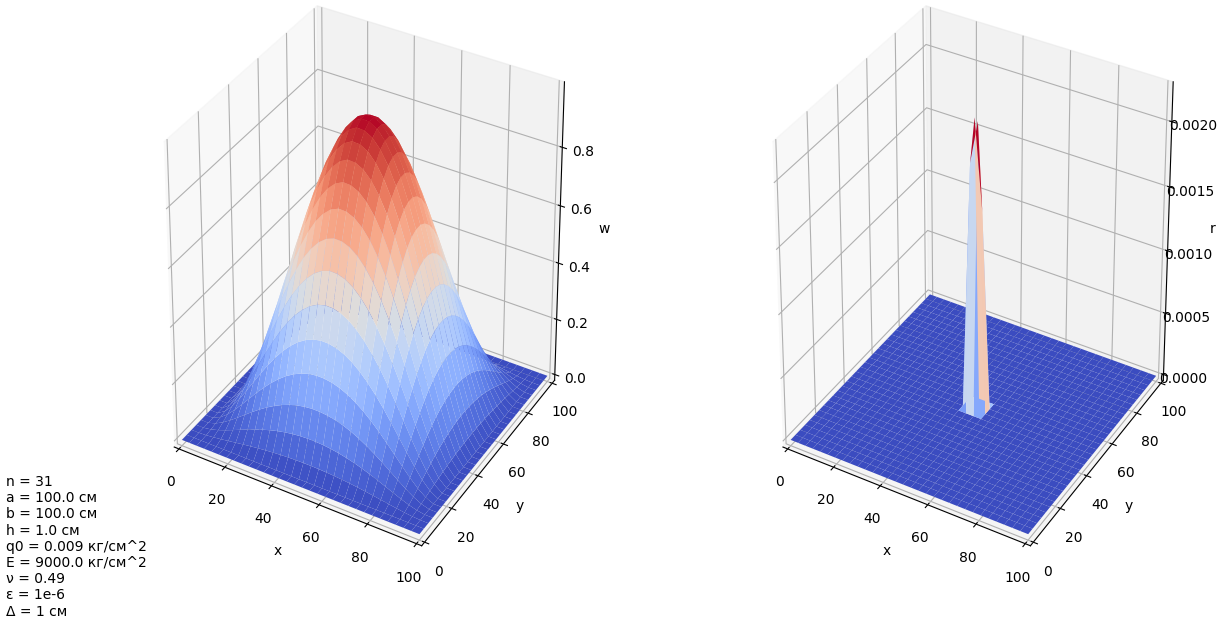
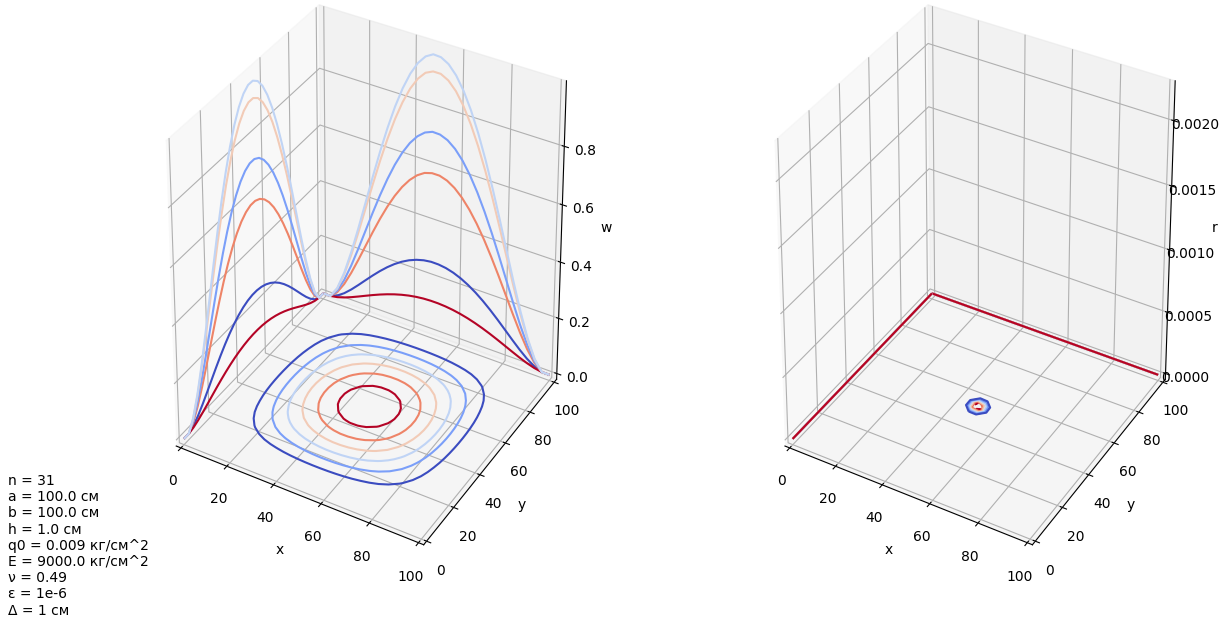
****

Рис. 6.2. Контуры пластины. Материал сталь

  
Рис. 6.3. Прогиб и реакция пластины. Материал резина

Рис. 6.4. Контуры пластины. Материал резина

По результатам расчётов можно утверждать, что в случае средней и большой зоны контакта расчёты не соответствуют теории Герца, а в случае малой зоны контакта соответствуют. Пиковые нагрузки на краях аналогичны результатам, полученным в одномерном случае.

В работе Ермоленко А.В. в одномерном случае было получено, что классической теории соответствует распределение контактных реакций, содержащее сосредоточенные силы на краях зоны контакта [2]. Также был сделан вывод, что итерационная схема не при любой начальной нагрузке приводит к допустимому решению . Условие односторонности связи выполняется только при нагрузке, которая превышает определённую нагрузку для заданных параметров пластины.

По классической теории в одномерном случае в средней части реакция равна половине действующей нагрузки, а на границе зоны контакта у нас   
δ–функции (сосредоточенные силы). В двумерном случае тоже получаются сосредоточенные силы.

Последовательность функции прогиба сходится в евклидовом пространстве , а последовательность , соответствующая реакции пластины, вообще говоря, не сходятся в . К аналогичным выводам пришли Тарасов В.Н. и Михайловский Е.И. [3]. Это связано с тем, что реакция описывается через δ–функции, которые нельзя представить численно.

**Заключение**

Было разработано программное обеспечение для построения поверхностей функций прогиба и реакции прямоугольной пластины по классической теории. Проанализирован способ нахождения численного решения по теории Кармана. В работе использовано два языка программирования для разных задач: C++ для вычисления матриц прогиба и реакции пластины, Python 3 для построения поверхностей. Скрипт на языке Python вызывает скомпилированный код, написанный на языке программирования C++, чтобы создать JSON-файлы для соответствующих функций, по которым в дальнейшем и строятся данные поверхности. Для создания JSON-файла использовалась библиотека rapidjson. Графический интерфейс программы был создан с помощью библиотеки PyQt 5 и изображён на Рис. 7.1.

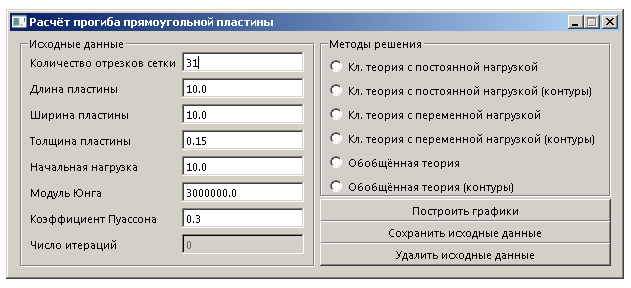


Рис. 7.1. Графический интерфейс.

Рекомендуется решить проблему быстрого увеличения значений функции прогиба по теории Кармана при малых итерациях.

**Источники**

1. Михайловский Е.И. Метод обобщённой реакции в задаче с неизвестной границей контакта/ Е.И. Михайловский, В.Н. Тарасов.– Тезисы докл. III Всесоюз. конф. по нелинейной теории упругости.– Сыктывкар: Сыктывкарский ун-т. 1989. С. 80-81.
2. Михайловский Е. И., Бадокин К.В., Ермоленко А.В. Теория изгиба пластин типа Кармана без гипотез Кирхгофа // Вестник Сыктывкарского университета. Сер.1: Мат. Мех. Инф. 1999. Вып.3. С.181-202.
3. Михайловский Е.И., Тарасов В.Н. О сходимости метода обобщенной реакции в контактных задачах со свободной границей // РАН. ПММ. 1993. Т. 57. Вып. 1. С. 128-136.
4. А.А. Самарский, А.В. Гулин. Численные методы: Учеб. пособие для вузов. – М.: Наука. Гл. ред. физ-мат. лит., 1989. 432 с.
5. Прохорёнок Н.А., Дронов В.А. Python 3 и PyQt 5. Разработка приложений.–Спб.: БХВ-Петербург, 2016.–832 с.: ил.
6. Matplotlib: Python plotting – Matplotlib 2.0.2 documentation [Электронный ресурс]: – Режим доступа: http://matplotlib.org/gallery.html – Загл. с экрана. 17.06.2017.
7. The world's leading software development platform · GitHub [Электронный ресурс]: – Режим доступа: https://github.com/miloyip/rapidjson – Загл. с экрана. 17.06.2017.
8. The official home of the Python Programming Language [Электронный ресурс]: – Режим доступа: https://docs.python.org/3/ – Загл. с экрана. 17.06.2017.

**Приложение**

Заголовочный файл C++ Helper.h.

#pragma once  
#include "iostream"  
using namespace std;

class Helper {

public:  
 Helper(void); ~Helper(void);  
 void set\_args(int argc, char\* argv[], int &n, double &a, double &b,   
 double &h, double &q0, double &E, double &nu);

void init\_matr\_idx(int n, int\*\* &matr\_idx);

void init\_A1(int n, double hx, double hy, int\*\* &matr\_idx, double\*\* &A1);

void init\_A2(int max\_idx, double\*\* &A2);  
 void init\_Axx\_Ayy\_Axy(int n, double hx, double hy, int\*\* &matr\_idx,  
 double\*\* &Axx, double\*\* &Ayy, double\*\* &Axy);

void front\_process(int max\_idx, double\*\* &A1, double\*\* &A2);

void back\_process(int max\_idx, double\*\* &A1, double\*\* &A2);  
 void init\_w\_r(int n, double\* &w, double\* &r, double\*\* &W, double\*\* &R);  
 void save\_matrix\_to\_json(int n, double\*\* matrix, string matrix\_name);

};

Вспомогательный класс C++ Helper.cpp для решения различных задач, присутствующие, как в классической, так и обобщённой теориях [7].

#include "Helper.h"

#include "math.h"  
#include "rapidjson/writer.h"  
#include "rapidjson/stringbuffer.h"  
#include "fstream"  
using namespace rapidjson;  
  
Helper::Helper(void) {}

Helper::~Helper(void) {}  
  
void Helper::set\_args(int argc, char\* argv[], int &n, double &a, double &b,  
 double &h, double &q0, double &E, double &nu) {  
 if (argc > 1) { n = atoi(argv[1]); } if (argc > 2) { a = atof(argv[2]); }  
 if (argc > 3) { b = atof(argv[3]); } if (argc > 4) { h = atof(argv[4]); }  
 if (argc > 5) { q0 = atof(argv[5]); } if (argc > 6) { E = atof(argv[6]); }  
 if (argc > 7) { nu = atof(argv[7]); }  
}

void Helper::init\_matr\_idx(int n, int\*\* &matr\_idx) {  
 int k = 1;  
 for (int i = 2; i <= n - 2; i++)  
 for (int j = 2; j <= n - 2; j++) { matr\_idx[i][j] = k++; }  
}

void Helper::init\_A1(int n,double hx,double hy,int\*\* &matr\_idx,double\*\* &A1) {  
 double hx4 = pow(hx, 4), hy4 = pow(hy, 4), hx2hy2 = pow(hx, 2) \* pow(hy, 2);  
 int k = 0;  
 for (int i = 2; i <= n - 2; i++) {  
 for (int j = 2; j <= n - 2; j++) {  
 A1[k][matr\_idx[i][j] - 1] = 6 / hx4 + 6 / hy4 + 8 / hx2hy2;  
 if (matr\_idx[i - 2][j] != 0) { A1[k][matr\_idx[i - 2][j] - 1] = 1 / hx4; }  
 if (matr\_idx[i + 2][j] != 0) { A1[k][matr\_idx[i + 2][j] - 1] = 1 / hx4; }  
 if (matr\_idx[i][j - 2] != 0) { A1[k][matr\_idx[i][j - 2] - 1] = 1 / hy4; }  
 if (matr\_idx[i][j + 2] != 0) { A1[k][matr\_idx[i][j + 2] - 1] = 1 / hy4; }  
 if (matr\_idx[i-1][j] != 0) A1[k][matr\_idx[i-1][j]-1] = -4 / hx4 -4 / hx2hy2;  
 if (matr\_idx[i+1][j] != 0) A1[k][matr\_idx[i+1][j]-1] = -4 / hx4 -4 / hx2hy2;  
 if (matr\_idx[i][j-1] != 0) A1[k][matr\_idx[i][j-1]-1] = -4 / hy4 -4 / hx2hy2;  
 if (matr\_idx[i][j+1] != 0) A1[k][matr\_idx[i][j+1]-1] = -4 / hy4 -4 / hx2hy2;  
 if (matr\_idx[i-1][j-1] != 0) A1[k][matr\_idx[i - 1][j - 1] - 1] = 2 / hx2hy2;  
 if (matr\_idx[i-1][j+1] != 0) A1[k][matr\_idx[i - 1][j + 1] - 1] = 2 / hx2hy2;  
 if (matr\_idx[i+1][j-1] != 0) A1[k][matr\_idx[i + 1][j - 1] - 1] = 2 / hx2hy2;  
 if (matr\_idx[i+1][j+1] != 0) A1[k][matr\_idx[i + 1][j + 1] - 1] = 2 / hx2hy2;  
 k++;  
}}}  
  
void Helper::init\_A2(int max\_idx, double\*\* &A2) {  
 for (int i = 0; i < max\_idx; i++) { A2[i][i] = 1.0; }  
}

void Helper::init\_Axx\_Ayy\_Axy(int n, double hx, double hy, int\*\* &matr\_idx,  
 double\*\* &Axx, double\*\* &Ayy, double\*\* &Axy) {  
 double hx2 = pow(hx, 2), hy2 = pow(hy, 2); int k = 0;  
 for (int i = 2; i <= n - 2; i++) {  
 for (int j = 2; j <= n - 2; j++) {  
 Axx[k][matr\_idx[i][j] - 1] = -2 / hx2; Ayy[k][matr\_idx[i][j] - 1] = -2 / hy2;  
 Axy[k][matr\_idx[i][j] - 1] = 1 / (hx \* hy);  
 if (matr\_idx[i - 1][j] != 0) {  
 Axx[k][matr\_idx[i - 1][j] - 1] = 1 / hx2;  
 Axy[k][matr\_idx[i - 1][j] - 1] = -1 / (2 \* hx \* hy); }  
 if (matr\_idx[i + 1][j] != 0) {  
 Axx[k][matr\_idx[i + 1][j] - 1] = 1 / hx2;  
 Axy[k][matr\_idx[i + 1][j] - 1] = -1 / (2 \* hx \* hy); }  
 if (matr\_idx[i][j - 1] != 0) {  
 Ayy[k][matr\_idx[i][j - 1] - 1] = 1 / hy2;  
 Axy[k][matr\_idx[i][j - 1] - 1] = -1 / (2 \* hx \* hy); }  
 if (matr\_idx[i][j + 1] != 0) {  
 Ayy[k][matr\_idx[i][j + 1] - 1] = 1 / hy2;  
 Axy[k][matr\_idx[i][j + 1] - 1] = -1 / (2 \* hx \* hy); }  
 if (matr\_idx[i - 1][j - 1] != 0) {   
 Axy[k][matr\_idx[i - 1][j - 1] - 1] = 1 / (2 \* hx \* hy); }  
 if (matr\_idx[i + 1][j + 1] != 0) {   
 Axy[k][matr\_idx[i + 1][j + 1] - 1] = 1 / (2 \* hx \* hy); }  
 k++;  
}}}  
  
void Helper::front\_process(int max\_idx, double\*\* &A1, double\*\* &A2) {  
 double tmp;  
 for (int k = 0; k < max\_idx; k++) {  
 if (A1[k][k] == 0.0) {  
 for (int j = 0; j < max\_idx; j++) {  
 tmp = A1[k][j]; A1[k][j] = A1[k + 1][j]; A1[k + 1][j] = tmp;  
 tmp = A2[k][j]; A2[k][j] = A2[k + 1][j]; A2[k + 1][j] = tmp;  
 }}  
 if (A1[k][k] != 0.0 && A1[k][k] != 1.0) {  
 tmp = A1[k][k];  
 for (int j = k; j < max\_idx; j++) { A1[k][j] /= tmp; }  
 for (int j = 0; j < max\_idx; j++) { A2[k][j] /= tmp; }  
 if (k == max\_idx - 1) { break; }  
 }  
 for (int i = k + 1; i < max\_idx; i++) {  
 tmp = A1[i][k];  
 for (int j = k; j < max\_idx; j++) { A1[i][j] -= A1[k][j] \* tmp; }  
 for (int j = 0; j < max\_idx; j++) { A2[i][j] -= A2[k][j] \* tmp; }  
}}}  
  
void Helper::back\_process(int max\_idx, double\*\* &A1, double\*\* &A2) {  
 double tmp;  
 for (int k = max\_idx - 1; k > 0; k--) {  
 for (int i = k - 1; i > -1; i--) {  
 tmp = A1[i][k];  
 for (int j = k; j > i; j--) { A1[i][j] -= A1[k][j] \* tmp; }  
 for (int j = 0; j < max\_idx; j++) { A2[i][j] -= A2[k][j] \* tmp; }  
}}}  
  
void Helper::init\_w\_r(int n, double\* &w, double\* &r,   
 double\*\* &W, double\*\* &R) {  
 int k = 0;  
 for (int i = 2; i <= n - 2; i++) {  
 for (int j = 2; j <= n - 2; j++) {  
 W[i][j] = w[k]; R[i][j] = r[k]; k++;  
}}}  
  
void Helper::save\_matrix\_to\_json(int n, double\*\* matrix, string matrix\_name) {  
 ofstream fout((matrix\_name + ".json").c\_str());  
 StringBuffer sb; Writer<StringBuffer> writer(sb);  
 writer.StartObject(); writer.Key(matrix\_name.c\_str()); writer.StartArray();  
 for (int i = 0; i < n + 1; i++) {  
 writer.StartArray();  
 for (int j = 0; j < n + 1; j++) { writer.Double(matrix[i][j]); }  
 writer.EndArray(); }  
 writer.EndArray(); writer.EndObject();  
 fout << sb.GetString() << endl; fout.close();  
}

Файл C++ ct\_const.cpp для построения JSON-файла матрицы прогиба по классической теории с постоянной нагрузкой.  
#include "math.h"  
#include "Helper.h"  
  
int \*\*matr\_idx = NULL; double \*\*A1 = NULL, \*\*A2 = NULL;  
double \*\_w = NULL; double \*\*w = NULL;  
  
void alloc\_arrays(int n, int max\_idx, int\*\* &matr\_idx,  
 double\*\* &A1, double\*\* &A2, double\* &\_w, double\*\* &w) {  
 matr\_idx = new int\*[n + 1]; w = new double\*[n + 1];  
 for (int i = 0; i < n + 1; i++) {  
 matr\_idx[i] = new int[n + 1](); w[i] = new double[n + 1](); }  
 A1 = new double\*[max\_idx]; A2 = new double\*[max\_idx];  
 for (int i = 0; i < max\_idx; i++) {   
 A1[i] = new double[max\_idx](); A2[i] = new double[max\_idx](); }  
 \_w = new double[max\_idx];  
}  
  
void release\_arrays(int n, int max\_idx, int\*\* &matr\_idx,  
 double\*\* &A1, double\*\* &A2, double\* &\_w, double\*\* &w) {  
 for (int i = 0; i < n + 1; i++) { delete[] matr\_idx[i]; delete[] w[i]; }  
 delete[] matr\_idx; delete[] w;  
 for (int i = 0; i < max\_idx; i++) { delete[] A1[i]; delete[] A2[i]; }  
 delete[] A1; delete[] A2; delete[] \_w;  
}  
  
void init\_w(int n, int max\_idx, double q0, double D,  
 double\*\* &A2, double\* &\_w, double\*\* &w) {  
 double s; int k = 0;  
 for (int i = 0; i < max\_idx; i++) {  
 s = 0;  
 for (int j = 0; j < max\_idx; j++) { s += A2[i][j] \* (q0 / D); }  
 \_w[i] = s; }  
 for (int i = 2; i <= n - 2; i++)  
 for (int j = 2; j <= n - 2; j++) { w[i][j] = \_w[k++]; }  
}  
  
int main(int argc, char\* argv[]) {  
 int n; double a, b, h, q0, E, nu;  
 Helper helper; helper.set\_args(argc, argv, n, a, b, h, q0, E, nu);  
 double D = E \* pow(h, 3) / (12 \* (1 - pow(nu,2)));  
 int max\_idx = (n - 3) \* (n - 3); double hx = a / n, hy = b / n;  
 alloc\_arrays(n, max\_idx, matr\_idx, A1, A2, \_w, w);  
 helper.init\_matr\_idx(n, matr\_idx);  
 helper.init\_A1(n, hx, hy, matr\_idx, A1); helper.init\_A2(max\_idx, A2);  
 helper.front\_process(max\_idx, A1, A2); helper.back\_process(max\_idx, A1, A2);  
 init\_w(n, max\_idx, q0, D, A2, \_w, w); helper.save\_matrix\_to\_json(n, w);  
 release\_arrays(n, max\_idx, matr\_idx, A1, A2, \_w, w);  
 return 0;  
}  
  
Файл C++ ct.cpp для построения JSON-файла матрицы прогиба по классической теории с переменной нагрузкой.  
#include "math.h"  
#include "stdlib.h"  
#include "Helper.h"  
  
const double beta = 1, delta = 1;  
  
int \*\*matr\_idx = NULL; double \*\*A1 = NULL, \*\*A2 = NULL;  
double \*vec\_tmp = NULL, \*\_w = NULL, \*\_r = NULL, \*\_r1 = NULL;  
double \*\*w = NULL, \*\*r = NULL;  
  
void alloc\_arrays(int n, int max\_idx, int\*\* &matr\_idx, double\*\* &A1,   
 double\*\* &A2,double\* &vec\_tmp, double\* &\_w,   
 double\* &\_r, double\* &\_r1, double\*\* &w, double\*\* &r) {  
 matr\_idx = new int\*[n + 1]; w = new double\*[n + 1]; r = new double\*[n + 1];  
 for (int i = 0; i < n + 1; i++) {  
 matr\_idx[i] = new int[n + 1]();  
 w[i] = new double[n + 1](); r[i] = new double[n + 1](); }  
 A1 = new double\*[max\_idx]; A2 = new double\*[max\_idx];  
 for (int i = 0; i < max\_idx; i++) {  
 A1[i] = new double[max\_idx](); A2[i] = new double[max\_idx](); }  
 vec\_tmp = new double[max\_idx]; \_w = new double[max\_idx];  
 \_r = new double[max\_idx](); \_r1 = new double[max\_idx]();  
}  
  
void release\_arrays(int n, int max\_idx, int\*\* &matr\_idx, double\*\* &A1,   
 double\*\* &A2, double\* &vec\_tmp, double\* &\_w,   
 double\* &\_r, double\* &\_r1, double\*\* &w, double\*\* &r) {  
 for (int i = 0; i < n + 1; i++) { delete[] matr\_idx[i]; delete[] w[i]; delete[] r[i]; }  
 delete[] matr\_idx; delete[] w; delete[] r;  
 for (int i = 0; i < max\_idx; i++) { delete[] A1[i]; delete[] A2[i]; }  
 delete[] A1; delete[] A2; delete[] vec\_tmp; delete[] \_w; delete[] \_r; delete[] \_r1;  
}  
  
void calc\_vec\_w(int i, int max\_idx, double D, double q0,   
 double\*\* &A2, double\* &\_w, double\* &\_r) {  
 double tmp = 0;  
 for (int j = 0; j < max\_idx; j++) { tmp += A2[i][j] \* (q0 - \_r[j]); }  
 tmp \*= 1 / D; \_w[i] = tmp;  
}  
  
void calc\_w\_r(double\* &\_r, double\* &\_r1, double\* &\_w, int i, int max\_idx,  
 double D, double q0, double\*\* &A2) {  
 \_r[i] = \_r1[i] + beta \* (\_w[i] - delta); if (\_r[i] < 0) { \_r[i] = 0; }  
 calc\_vec\_w(i, max\_idx, D, q0, A2, \_w, \_r); \_r1[i] = \_r[i];  
}  
  
void generalized\_reaction\_method(int num\_iters, int max\_idx,  
 double D, double q0, double\*\* &A2,   
 double\* &vec\_tmp, double\* &\_r, double\* &\_r1, double\* &\_w) {  
 for (int i = 0; i < max\_idx; i++) {  
 calc\_vec\_w(i, max\_idx, D, q0, A2, \_w, \_r); vec\_tmp[i] = \_w[i]; }  
 if (num\_iters == 0) {  
 double norma = 1, sum1 = 0, sum2 = 0, eps = 1e-6;  
 while (norma >= eps) {  
 sum1 = 0; sum2 = 0;  
 for (int i = 0; i < max\_idx; i++) {  
 calc\_w\_r(\_r, \_r1, \_w, i, max\_idx, D, q0, A2);  
 sum1 += pow(\_w[i] - vec\_tmp[i], 2); sum2 += pow(\_w[i], 2);  
 norma = sqrt(sum1) / sqrt(sum2); vec\_tmp[i] = \_w[i]; }  
 }  
 } else {  
 for (int k = 1; k <= num\_iters; k++)  
 for (int i = 0; i < max\_idx; i++)  
 calc\_w\_r(\_r, \_r1, \_w, i, max\_idx, D, q0, A2);  
}}  
  
int main(int argc, char\* argv[]) {  
 int n, num\_iters; double a, b, h, q0, E, nu;   
 Helper helper; helper.set\_args(argc, argv, n, a, b, h, q0, E, nu);  
 if (argc > 8) { num\_iters = atoi(argv[8]); }  
 double D = E \* pow(h, 3) / (12 \* (1 - pow(nu,2)));  
 int max\_idx = (n - 3) \* (n - 3); double hx = a / n, hy = b / n;  
 alloc\_arrays(n, max\_idx, matr\_idx, A1, A2, vec\_tmp, \_w, \_r, \_r1, w, r);  
 helper.init\_matr\_idx(n, matr\_idx);  
 helper.init\_A1(n, hx, hy, matr\_idx, A1); helper.init\_A2(max\_idx, A2);  
 helper.front\_process(max\_idx, A1, A2); helper.back\_process(max\_idx, A1, A2);  
 generalized\_reaction\_method(num\_iters,max\_idx,D,q0,A2,vec\_tmp,\_r,\_r1, \_w);  
 helper.init\_w\_r(n, \_w, \_r, w, r);  
 helper.save\_matrix\_to\_json(n, w, "w");  
 helper.save\_matrix\_to\_json(n, r, "r");  
 release\_arrays(n, max\_idx, matr\_idx, A1, A2, vec\_tmp, \_w, \_r, \_r1, w, r);  
 return 0;  
}  
  
Файл C++ karman.cpp для построения JSON-файла матрицы прогиба по теории Кармана.  
#include "math.h"  
#include "stdlib.h"  
#include "Helper.h"  
  
const double beta = 1, delta = 1;  
  
int \*\*matr\_idx = NULL; double \*\*A1 = NULL, \*\*A2 = NULL;  
double \*\*Axx = NULL, \*\*Ayy = NULL, \*\*Axy = NULL;  
double \*r = NULL, \*r\_prev = NULL, \*fi = NULL, \*w = NULL;  
double \*\*matr\_tmp1 = NULL, \*\*matr\_tmp2 = NULL, \*\*matr\_tmp3 = NULL,  
 \*\*matr\_tmp4 = NULL, \*\*matr\_tmp5 = NULL, \*\*matr\_tmp6 = NULL,  
 \*\*matr\_tmp7 = NULL, \*\*matr\_tmp8 = NULL, \*\*matr\_tmp9 = NULL;  
double \*vec\_tmp = NULL; double \*\*W = NULL, \*\*R = NULL;  
  
void alloc\_arrays(int n, int max\_idx, double\*\* &Axx, double\*\* &Ayy,   
 double\*\* &Axy, int\*\* &matr\_idx, double\* &vec\_tmp, double\*\* &A1,  
 double\*\* &A2, double\* &r, double\* &r\_prev, double\* &fi, double\* &w,  
 double\*\* &matr\_tmp1, double\*\* &matr\_tmp2, double\*\* &matr\_tmp3,  
 double\*\* &matr\_tmp4, double\*\* &matr\_tmp5, double\*\* &matr\_tmp6,  
 double\*\* &matr\_tmp7, double\*\* &matr\_tmp8, double\*\* &matr\_tmp9,  
 double\*\* &W, double\*\* &R) {  
 matr\_tmp1 = new double\*[n - 3]; matr\_tmp2 = new double\*[n - 3];  
 matr\_tmp3 = new double\*[n - 3]; matr\_tmp4 = new double\*[n - 3];  
 matr\_tmp5 = new double\*[n - 3]; matr\_tmp6 = new double\*[n - 3];  
 matr\_tmp7 = new double\*[n - 3]; matr\_tmp8 = new double\*[n - 3];  
 matr\_tmp9 = new double\*[n - 3];  
 for (int i = 0; i < n - 3; i++) {  
 matr\_tmp1[i] = new double[n - 3](); matr\_tmp2[i] = new double[n - 3]();  
 matr\_tmp3[i] = new double[n - 3](); matr\_tmp4[i] = new double[n - 3]();  
 matr\_tmp5[i] = new double[n - 3](); matr\_tmp6[i] = new double[n - 3]();  
 matr\_tmp7[i] = new double[n - 3](); matr\_tmp8[i] = new double[n - 3]();  
 matr\_tmp9[i] = new double[n - 3](); }  
 matr\_idx = new int\*[n + 1]; W = new double\*[n + 1]; R = new double\*[n + 1];  
 for (int i = 0; i < n + 1; i++) {  
 matr\_idx[i] = new int[n + 1]();  
 W[i] = new double[n + 1](); R[i] = new double[n + 1](); }  
 r = new double[max\_idx](); r\_prev = new double[max\_idx];  
 fi = new double[max\_idx](); w = new double[max\_idx]();  
 vec\_tmp = new double[max\_idx]();  
 A1 = new double\*[max\_idx]; A2 = new double\*[max\_idx];  
 Axx = new double\*[max\_idx]; Ayy = new double\*[max\_idx];  
 Axy = new double\*[max\_idx];  
 for (int i = 0; i < max\_idx; i++) {  
 A1[i] = new double[max\_idx](); A2[i] = new double[max\_idx]();  
 Axx[i] = new double[max\_idx](); Ayy[i] = new double[max\_idx]();  
 Axy[i] = new double[max\_idx](); }  
}  
  
void release\_arrays(int n, int max\_idx, double\*\* &Axx, double\*\* &Ayy,   
 double\*\* &Axy, int\*\* &matr\_idx, double\* &vec\_tmp, double\*\* &A1,  
 double\*\* &A2, double\* &r, double\* &r\_prev, double\* &fi, double\* &w,  
 double\*\* &matr\_tmp1, double\*\* &matr\_tmp2, double\*\* &matr\_tmp3,  
 double\*\* &matr\_tmp4, double\*\* &matr\_tmp5, double\*\* &matr\_tmp6,  
 double\*\* &matr\_tmp7, double\*\* &matr\_tmp8, double\*\* &matr\_tmp9,  
 double\*\* &W, double\*\* &R) {

delete[] r; delete[] r\_prev; delete[] fi; delete[] w;  
 for (int i = 0; i < n - 3; i++) {  
 delete[] matr\_tmp1[i]; delete[] matr\_tmp2[i]; delete[] matr\_tmp3[i];  
 delete[] matr\_tmp4[i]; delete[] matr\_tmp5[i]; delete[] matr\_tmp6[i];  
 delete[] matr\_tmp7[i]; delete[] matr\_tmp8[i]; delete[] matr\_tmp9[i]; }  
 delete[] matr\_tmp1; delete[] matr\_tmp2; delete[] matr\_tmp3;  
 delete[] matr\_tmp4; delete[] matr\_tmp5; delete[] matr\_tmp6;  
 delete[] matr\_tmp7; delete[] matr\_tmp8; delete[] matr\_tmp9;  
 for (int i = 0; i < n + 1; i++) {  
 delete[] matr\_idx[i]; delete[] W[i]; delete[] R[i]; }  
 delete[] matr\_idx; delete[] W; delete[] R;  
 for (int i = 0; i < max\_idx; i++) {  
 delete[] A1[i]; delete[] A2[i]; delete[] Axx[i]; delete[] Ayy[i]; delete[] Axy[i];}  
 delete[] A1; delete[] A2; delete[] Axx; delete[] Ayy; delete[] Axy;  
}  
  
void vector\_to\_matrix(int max\_idx, int n, double\* &v, double\*\* &M) {  
 int k = 0;  
 for (int i = 0; i < n - 3; i++)  
 for (int j = 0; j < n - 3; j++) { M[i][j] = v[k++]; }  
}  
  
void matrix\_mul\_matrix(int n, double\*\* &M1, double\*\* &M2,  
 double\*\* &matr\_tmp, double c = 1) {  
 double tmp;  
 for (int i = 0; i < n - 3; i++) {  
 for (int j = 0; j < n - 3; j++) {  
 tmp = 0;  
 for (int k = 0; k < n - 3; k++) { tmp += M1[i][k] \* M2[k][j]; }  
 matr\_tmp[i][j] = c \* tmp;  
}}}  
  
void matrix\_mul\_vector(int max\_idx, double\*\* &M, double\* &v,   
 double\* &vec\_tmp, double c = 1, double q0 = 0) {  
 double tmp;  
 for (int i = 0; i < max\_idx; i++) {  
 tmp = 0;  
 for (int j = 0; j < max\_idx; j++) {  
 if (q0 == 0) { tmp += M[i][j] \* v[j]; }  
 else { tmp += M[i][j] \* q0; }  
 }  
 vec\_tmp[i] = c \* tmp;  
}}  
  
void calc\_vec\_fi(int max\_idx, int n, double E, double h, double\*\* &A2,   
 double\*\* &matr\_tmp1, double\*\* &matr\_tmp2, double\*\* &matr\_tmp3,   
 double\*\* &matr\_tmp4, double\*\* &matr\_tmp5,   
 double\*\* &Axx, double\*\* &Ayy, double\*\* &Axy,  
 double\* &w, double\* &fi, double\* &vec\_tmp) {   
 matrix\_mul\_vector(max\_idx, Axy, w, vec\_tmp);  
 vector\_to\_matrix(max\_idx, n, vec\_tmp, matr\_tmp1);  
 matrix\_mul\_matrix(n, matr\_tmp1, matr\_tmp1, matr\_tmp4);  
 matrix\_mul\_vector(max\_idx, Axx, w, vec\_tmp);  
 vector\_to\_matrix(max\_idx, n, vec\_tmp, matr\_tmp2);  
 matrix\_mul\_vector(max\_idx, Ayy, w, vec\_tmp);  
 vector\_to\_matrix(max\_idx, n, vec\_tmp, matr\_tmp3);  
 matrix\_mul\_matrix(n, matr\_tmp3, matr\_tmp4, matr\_tmp5);  
 int k = 0;  
 for (int i = 0; i < n - 3; i++)  
 for (int j = 0; j < n - 3; j++)  
 vec\_tmp[k++] = matr\_tmp4[i][j] - matr\_tmp5[i][j];  
 matrix\_mul\_vector(max\_idx, A2, vec\_tmp, fi, E \* h);  
}  
  
void calc\_vec\_w(int max\_idx, int n, double D, double q0, double\*\* &A2,  
 double\*\* &matr\_tmp1, double\*\* &matr\_tmp2, double\*\* &matr\_tmp3,   
 double\*\* &matr\_tmp4, double\*\* &matr\_tmp5, double\*\* &matr\_tmp6,   
 double\*\* &matr\_tmp7, double\*\* &matr\_tmp8, double\*\* &matr\_tmp9,   
 double\*\* &Axx, double\*\* &Ayy, double\*\* &Axy,   
 double\* &w, double\* &fi) {  
 matrix\_mul\_vector(max\_idx, Ayy, fi, vec\_tmp);  
 vector\_to\_matrix(max\_idx, n, vec\_tmp, matr\_tmp4);  
 matrix\_mul\_vector(max\_idx, Axx, fi, vec\_tmp);  
 vector\_to\_matrix(max\_idx, n, vec\_tmp, matr\_tmp5);  
 matrix\_mul\_vector(max\_idx, Axy, fi, vec\_tmp);  
 vector\_to\_matrix(max\_idx, n, vec\_tmp, matr\_tmp6);  
 matrix\_mul\_matrix(n, matr\_tmp4, matr\_tmp2, matr\_tmp7);  
 matrix\_mul\_matrix(n, matr\_tmp5, matr\_tmp3, matr\_tmp8);  
 matrix\_mul\_matrix(n, matr\_tmp6, matr\_tmp1, matr\_tmp9);  
 int k = 0;  
 for (int i = 0; i < n - 3; i++)  
 for (int j = 0; j < n - 3; j++)  
 vec\_tmp[k++] = (q0 - r[k]) + matr\_tmp7[i][j] + matr\_tmp8[i][j]   
 - 2 \* matr\_tmp9[i][j];  
 matrix\_mul\_vector(max\_idx, A2, vec\_tmp, w, 1 / D);  
}  
  
void generalized\_reaction\_method(int num\_iters, int max\_idx, int n,   
 double q0, double D, double E, double h, double\*\* &A2, double\* &w,  
 double\* &r, double\* &r\_prev, double\* &fi, double\* &vec\_tmp,  
 double\*\* &matr\_tmp1, double\*\* &matr\_tmp2, double\*\* &matr\_tmp3,  
 double\*\* &matr\_tmp4, double\*\* &matr\_tmp5, double\*\* &matr\_tmp6,  
 double\*\* &matr\_tmp7, double\*\* &matr\_tmp8, double\*\* &matr\_tmp9,  
 double\*\* &Axx, double\*\* &Ayy, double\*\* &Axy) {  
 matrix\_mul\_vector(max\_idx, A2, vec\_tmp, w, 1 / D, q0);  
 for (int k = 1; k <= num\_iters; k++) {  
 for (int i = 0; i < max\_idx; i++) {  
 r[i] = r\_prev[i] + beta \* (w[i] - delta); if (r[i] < 0) { r[i] = 0; }  
 r\_prev[i] = r[i]; }  
 calc\_vec\_w(max\_idx, n, D, q0, A2, matr\_tmp1,   
 matr\_tmp2, matr\_tmp3, matr\_tmp4, matr\_tmp5, matr\_tmp6,   
 matr\_tmp7, matr\_tmp8, matr\_tmp9, Axx, Ayy, Axy, w, fi);  
 calc\_vec\_fi(max\_idx, n, E, h, A2, matr\_tmp1, matr\_tmp2, matr\_tmp3,   
 matr\_tmp4, matr\_tmp5, Axx, Ayy, Axy, w, fi, vec\_tmp);  
}}  
  
int main(int argc, char\* argv[]) {  
 int n, num\_iters; double a, b, h, q0, E, nu;  
 Helper helper; helper.set\_args(argc, argv, n, a, b, h, q0, E, nu);  
 if (argc > 8) { num\_iters = atoi(argv[8]); }  
 double D = E \* pow(h, 3) / (12 \* (1 - pow(nu,2)));  
 int max\_idx = (n - 3) \* (n - 3); double hx = a / n, hy = b / n;  
 alloc\_arrays(n, max\_idx, Axx, Ayy, Axy, matr\_idx, vec\_tmp, A1, A2,   
 r, r\_prev, fi, w, matr\_tmp1, matr\_tmp2, matr\_tmp3, matr\_tmp4,   
 matr\_tmp5, matr\_tmp6, matr\_tmp7, matr\_tmp8, matr\_tmp9, W, R);  
 helper.init\_matr\_idx(n, matr\_idx); helper.init\_A1(n, hx, hy, matr\_idx, A1);  
 helper.init\_A2(max\_idx, A2);  
 helper.init\_Axx\_Ayy\_Axy(n, hx, hy, matr\_idx, Axx, Ayy, Axy);  
 helper.front\_process(max\_idx, A1, A2); helper.back\_process(max\_idx, A1, A2);  
 generalized\_reaction\_method(num\_iters, max\_idx, n, q0, D, E, h, A2, w, r,   
 r\_prev, fi, vec\_tmp, matr\_tmp1, matr\_tmp2, matr\_tmp3, matr\_tmp4,   
 matr\_tmp5, matr\_tmp6, matr\_tmp7, matr\_tmp8, matr\_tmp9, Axx, Ayy, Axy);  
 helper.init\_w\_r(n, w, r, W, R);   
 helper.save\_matrix\_to\_json(n, W, "w"); helper.save\_matrix\_to\_json(n, R, "r");  
 release\_arrays(n, max\_idx, Axx, Ayy, Axy, matr\_idx, vec\_tmp, A1, A2,   
 r, r\_prev, fi, w, matr\_tmp1, matr\_tmp2, matr\_tmp3, matr\_tmp4, matr\_tmp5,  
 matr\_tmp6, matr\_tmp7, matr\_tmp8, matr\_tmp9, W, R);  
 return 0;  
}  
  
Файл window.ui для хранения дерева виджетов формы в формате XML.  
<?xml version="1.0" encoding="UTF-8"?>  
<ui version="4.0"><class>MainWindow</class><widget class="QMainWindow" name="MainWindow"><property name="geometry"><rect>  
<x>0</x><y>0</y><width>610</width><height>243</height></rect></property>  
<property name="sizePolicy"><sizepolicy hsizetype="Preferred" vsizetype="Preferred"><horstretch>0</horstretch><verstretch>0</verstretch>  
</sizepolicy></property><property name="minimumSize">  
<size><width>610</width><height>243</height></size></property><property name="maximumSize"><size><width>610</width><height>243</height></size></property><property name="windowTitle"><string>Расчёт прогиба прямоугольной пластины</string></property><property name="autoFillBackground"><bool>false</bool></property>  
<widget class="QWidget" name="centralwidget"><widget class="QGroupBox" name="groupBoxInputData"><property name="geometry"><rect><x>10</x>  
<y>5</y><width>295</width><height>230</height></rect></property><property name="title"><string>Исходные данные</string></property>  
<widget class="QWidget" name="layoutWidget"><property name="geometry">  
<rect><x>10</x><y>15</y><width>274</width><height>204</height></rect>  
</property><layout class="QGridLayout" name="gridLayout"><item row="0" column="0"><widget class="QLabel" name="label\_n"><property name="text">  
<string>Количество отрезков сетки</string></property></widget></item>  
<item row="0" column="1"><widget class="QLineEdit" name="lineEdit\_n"/>  
</item><item row="1" column="0"><widget class="QLabel" name="label\_a">  
<property name="text"><string>Длина пластины</string></property></widget>  
</item><item row="1" column="1"><widget class="QLineEdit" name="lineEdit\_a"><property name="cursorPosition"><number>0</number>  
</property></widget></item><item row="2" column="0">  
<widget class="QLabel" name="label\_b"><property name="text"><string>  
Ширина пластины</string></property></widget></item><item row="2" column="1"><widget class="QLineEdit" name="lineEdit\_b"><property name="cursorPosition"><number>0</number></property></widget></item>  
<item row="3" column="0"><widget class="QLabel" name="label\_h">  
<property name="text"><string>Толщина пластины</string></property>  
</widget></item><item row="3" column="1"><widget class="QLineEdit" name="lineEdit\_h"><property name="cursorPosition"><number>0</number>  
</property></widget></item><item row="4" column="0"><widget class="QLabel" name="label\_q0"><property name="text"><string>Начальная нагрузка</string></property></widget></item><item row="4" column="1">  
<widget class="QLineEdit" name="lineEdit\_q0"><property name="cursorPosition"><number>0</number></property></widget></item>  
<item row="5" column="0"><widget class="QLabel" name="label\_E">  
<property name="text"><string>Модуль Юнга</string></property></widget>  
</item><item row="5" column="1"><widget class="QLineEdit" name="lineEdit\_E"><property name="cursorPosition"><number>0</number>  
</property></widget></item><item row="6" column="0"><widget class="QLabel" name="label\_nu"><property name="text"><string>Коэффициент Пуассона</string></property></widget></item><item row="6" column="1">  
<widget class="QLineEdit" name="lineEdit\_nu"><property name="cursorPosition"><number>0</number></property></widget></item>  
<item row="7" column="0"><widget class="QLabel" name="label\_num\_iters">  
<property name="text"><string>Число итераций</string></property></widget>  
</item><item row="7" column="1"><widget class="QLineEdit" name="lineEditNumIters"><property name="toolTip"><string>Если число итераций 0, то вычисляется норма!</string></property>  
<property name="text"><string>0</string></property><property name="cursorPosition"><number>0</number></property></widget></item>  
</layout></widget></widget><widget class="QGroupBox" name="groupBoxMethods"><property name="geometry"><rect><x>310</x>  
<y>5</y><width>291</width><height>160</height></rect></property>  
<property name="title"><string>Методы решения</string></property>  
<widget class="QWidget" name="layoutWidget"><property name="geometry">  
<rect><x>10</x><y>20</y><width>272</width><height>140</height></rect>  
</property><layout class="QVBoxLayout" name="verticalLayout"><item>  
<widget class="QRadioButton" name="radioButtonCtConst"><property name="focusPolicy"><enum>Qt::ClickFocus</enum></property><property name="text"><string>Кл. теория с постоянной нагрузкой</string>  
</property></widget></item><item><widget class="QRadioButton"  
name="radioButtonCtConstContour"><property name="focusPolicy">  
<enum>Qt::ClickFocus</enum></property><property name="text"><string>  
Кл. теория с постоянной нагрузкой (контуры)</string></property></widget>  
</item><item><widget class="QRadioButton" name="radioButtonCt"><property name="focusPolicy"><enum>Qt::ClickFocus</enum></property><property name="text"><string>Кл. теория с переменной нагрузкой</string></property>  
</widget></item><item><widget class="QRadioButton" name="radioButtonCtContour"><property name="focusPolicy"><enum>  
Qt::ClickFocus</enum></property><property name="text"><string>Кл. теория с переменной нагрузкой (контуры)</string></property></widget></item><item>  
<widget class="QRadioButton" name="radioButtonKarman"><property name="focusPolicy"><enum>Qt::ClickFocus</enum></property><property name="text"><string>Обобщённая теория</string></property></widget></item>  
<item><widget class="QRadioButton"name="radioButtonKarmanContour">  
<property name="focusPolicy"><enum>Qt::ClickFocus</enum></property>  
<property name="text"><string>Обобщённая теория (контуры)</string>  
</property></widget></item></layout></widget></widget><widget class="QPushButton" name="pushButtonPlot"><property name="geometry">  
<rect><x>310</x><y>167</y><width>291</width><height>23</height></rect>  
</property><property name="focusPolicy"><enum>Qt::ClickFocus</enum>  
</property><property name="text"><string>Построить графики</string>  
</property></widget><widget class="QPushButton" name="pushButtonSaveInputData"><property name="geometry"><rect><x>310  
</x><y>189</y><width>291</width><height>23</height></rect></property>  
<property name="focusPolicy"><enum>Qt::ClickFocus</enum></property>  
<property name="text"><string>Сохранить исходные данные</string>  
</property></widget><widget class="QPushButton" name="pushButtonDelInputData"><property name="geometry"><rect><x>310  
</x><y>211</y><width>291</width><height>23</height></rect></property>  
<property name="focusPolicy"><enum>Qt::ClickFocus</enum></property>  
<property name="text"><string>Удалить исходные данные</string></property>  
</widget></widget></widget><resources/><connections/></ui>  
  
  
Файл Python 3 DeflectionRectPlate.pyw является основным файлом для создания окна приложения и построения графиков [5], [6], [8].  
import sys, os.path, json, math, configparser  
from subprocess import Popen, PIPE, STDOUT  
from mpl\_toolkits.mplot3d import Axes3D  
import matplotlib.pyplot as plt, numpy as np  
from matplotlib import cm  
from PyQt5 import uic  
from PyQt5.QtWidgets import QMainWindow, QApplication, QMessageBox  
from enum import Enum  
  
w\_JSON\_FILE = 'w.json'; r\_JSON\_FILE = 'r.json'  
INPUT\_DATA\_INI\_FILE = 'input\_data.ini'; WINDOW\_UI\_FILE = 'window.ui'  
  
class PlotType(Enum): surface = 1; contour = 2  
  
def getListFromJSON(fname, key):  
 tmp = None  
 with open(fname, 'r') as f: s = f.read(); tmp = json.loads(s)[key]  
 return tmp  
  
class MainWindow(QMainWindow):  
 def \_\_init\_\_(self):  
 super().\_\_init\_\_()  
 window = uic.loadUi(WINDOW\_UI\_FILE, self)  
 self.loadInputData(window, INPUT\_DATA\_INI\_FILE)  
 self.deleteJSONFiles(w\_JSON\_FILE, r\_JSON\_FILE)  
 self.inputData = ()  
  
 window.lineEditNumIters.setDisabled(True)  
 window.pushButtonPlot.clicked.connect(self.plot)  
 window.pushButtonSaveInputData.clicked.connect(self.saveInputData)  
 window.pushButtonDelInputData.clicked \  
 .connect(lambda: self.delInputData(window))  
  
 window.radioButtonCtConst.clicked \  
 .connect(lambda: self.toggleLineEditNumIters('radioButtonCtConst'))  
 window.radioButtonCtConstContour \  
 .clicked.connect \  
 (lambda: self.toggleLineEditNumIters('radioButtonCtConstContour'))  
 window.radioButtonCt.clicked \  
 .connect(lambda: self.toggleLineEditNumIters('radioButtonCt'))  
 window.radioButtonCtContour.clicked \  
 .connect(lambda: self.toggleLineEditNumIters('radioButtonCtContour'))  
 window.radioButtonKarman.clicked \  
 .connect(lambda: self.toggleLineEditNumIters('radioButtonKarman'))  
 window.radioButtonKarmanContour.clicked \  
 .connect(lambda: self.toggleLineEditNumIters('radioButtonKarmanContour'))  
  
 self.show()  
   
 def deleteJSONFiles(self, fname1, fname2):  
 if os.path.isfile(fname1): os.remove(fname1)  
 if os.path.isfile(fname2): os.remove(fname2)  
  
 def loadInputData(self, window, fname):  
 if os.path.isfile(fname):  
 config = configparser.ConfigParser(); config.read(fname)  
 window.lineEdit\_n.setText(config.get('Input Data', 'n'))  
 window.lineEdit\_a.setText(config.get('Input Data', 'a'))  
 window.lineEdit\_b.setText(config.get('Input Data', 'b'))  
 window.lineEdit\_h.setText(config.get('Input Data', 'h'))  
 window.lineEdit\_q0.setText(config.get('Input Data', 'q0'))  
 window.lineEdit\_E.setText(config.get('Input Data', 'E'))  
 window.lineEdit\_nu.setText(config.get('Input Data', 'nu'))  
 window.lineEditNumIters.setText(config.get('Input Data', 'numIters'))  
  
 def saveInputData(self):  
 if self.checkLineEdits():  
 n, a, b, h, q0, E, nu, numIters, fname\_exe = self.getValues('')  
 config = configparser.ConfigParser(); config.add\_section('Input Data')  
 config.set('Input Data', 'n', str(n)); config.set('Input Data', 'a', str(a))  
 config.set('Input Data', 'b', str(b)); config.set('Input Data', 'h', str(h))  
 config.set('Input Data', 'q0', str(q0)); config.set('Input Data', 'E', str(E))  
 config.set('Input Data', 'nu', str(nu))  
 config.set('Input Data', 'numIters', str(numIters))  
 with open(INPUT\_DATA\_INI\_FILE, 'w') as f: config.write(f)  
  
 def delInputData(self, window):  
 window.lineEdit\_n.setText(''); window.lineEdit\_a.setText('')  
 window.lineEdit\_b.setText(''); window.lineEdit\_h.setText('')  
 window.lineEdit\_q0.setText(''); window.lineEdit\_E.setText('')  
 window.lineEdit\_nu.setText('')  
 if os.path.isfile(INPUT\_DATA\_INI\_FILE):  
 os.remove(INPUT\_DATA\_INI\_FILE)  
  
 def toggleLineEditNumIters(self, rbName):  
 if rbName == 'radioButtonCtConst' \  
 or rbName == 'radioButtonCtConstContour':  
 window.lineEditNumIters.setDisabled(True)  
 else: window.lineEditNumIters.setDisabled(False)  
  
 def checkLineEdits(self):  
 try:  
 int(window.lineEdit\_n.text()); float(window.lineEdit\_a.text())  
 float(window.lineEdit\_b.text()); float(window.lineEdit\_h.text())  
 float(window.lineEdit\_q0.text()); float(window.lineEdit\_E.text())  
 float(window.lineEdit\_nu.text()); int(window.lineEditNumIters.text())  
 return True  
 except ValueError:  
 QMessageBox.warning(self, 'Ошибка','Введены неверные данные')  
 return False  
  
 def getValues(self, fname\_exe):  
 n = int(window.lineEdit\_n.text()); a = float(window.lineEdit\_a.text())  
 b = float(window.lineEdit\_b.text()); h = float(window.lineEdit\_h.text())  
 q0 = float(window.lineEdit\_q0.text()); E = float(window.lineEdit\_E.text())  
 nu = float(window.lineEdit\_nu.text())  
 numIters = int(window.lineEditNumIters.text())  
 return (n, a, b, h, q0, E, nu, numIters, fname\_exe)  
  
 def getXY(self, n, a, b):  
 hx = a / n; hy = b / n;   
 x=np.arange(0, a+hx, hx); y=np.arange(0, b+hy, hy); x, y = np.meshgrid(x, y);  
 return (x, y)  
  
 def plot(self):  
 if self.checkLineEdits():  
 if window.radioButtonCtConst.isChecked():  
 n, a, b, h, q0, E, nu, numIters, fname\_exe = self.getValues('ct\_const.exe')  
 self.plotCtConst(n, a, b, h, q0, E, nu, numIters, self.getXY(n, a, b), \  
 PlotType(1), w\_JSON\_FILE, r\_JSON\_FILE, fname\_exe)  
 if window.radioButtonCtConstContour.isChecked():  
 n, a, b, h, q0, E, nu, numIters, fname\_exe = self.getValues('ct\_const.exe')  
 self.plotCtConst(n, a, b, h, q0, E, nu, numIters, self.getXY(n, a, b), \  
 PlotType(2), w\_JSON\_FILE, r\_JSON\_FILE, fname\_exe)  
 if window.radioButtonCt.isChecked():  
 n, a, b, h, q0, E, nu, numIters, fname\_exe = self.getValues('ct.exe')  
 self.plotCtKarman(n, a, b, h, q0, E, nu, numIters, self.getXY(n, a, b), \  
 PlotType(1), w\_JSON\_FILE, r\_JSON\_FILE, fname\_exe)  
 if window.radioButtonCtContour.isChecked():  
 n, a, b, h, q0, E, nu, numIters, fname\_exe = self.getValues('ct.exe')  
 self.plotCtKarman(n, a, b, h, q0, E, nu, numIters, self.getXY(n, a, b), \  
 PlotType(2), w\_JSON\_FILE, r\_JSON\_FILE, fname\_exe)  
 if window.radioButtonKarman.isChecked():  
 n, a, b, h, q0, E, nu, numIters, fname\_exe = self.getValues('karman.exe')  
 self.plotCtKarman(n, a, b, h, q0, E, nu, numIters, self.getXY(n, a, b), \  
 PlotType(1), w\_JSON\_FILE, r\_JSON\_FILE, fname\_exe)  
 if window.radioButtonKarmanContour.isChecked():  
 n, a, b, h, q0, E, nu, numIters, fname\_exe = self.getValues('karman.exe')  
 self.plotCtKarman(n, a, b, h, q0, E, nu, numIters, self.getXY(n, a, b), \  
 PlotType(2), w\_JSON\_FILE, r\_JSON\_FILE, fname\_exe)  
  
 def isChangedInputData(self, fname\_exe):  
 for i in range(len(self.inputData)):  
 if self.getValues(fname\_exe)[i] != self.inputData[i]:  
 return True  
 return False  
  
 def startProgram(self, fname1, fname2, n, a, b, h, q0, E, nu,   
 numIters, cmd, fname\_exe):  
 self.inputData = (n, a, b, h, q0, E, nu, numIters, fname\_exe)  
 self.deleteJSONFiles(fname1, fname2)  
 p = Popen(cmd, shell=True, stdin=PIPE, stdout=PIPE, stderr=STDOUT)  
 p.wait()  
 return p  
  
 def plotContours(self, ax, x, y, z, b):  
 ax.contour(x, y, z, zdir='z', offset=0, cmap=cm.coolwarm, antialiased=True)  
 ax.contour(x, y, z, zdir='x', offset=0, cmap=cm.coolwarm, antialiased=True)  
 ax.contour(x, y, z, zdir='y', offset=b, cmap=cm.coolwarm, antialiased=True)  
  
 def setAxes(self, ax, a, b, z):  
 ax.set\_xlabel('x'); ax.set\_xlim(0, a); ax.set\_ylabel('y')  
 ax.set\_ylim(0, b); ax.set\_zlabel(z)  
  
 def plotSurfaceOrContour(self, fig, bool\_const, plotType, a, b, x, y, z,   
 z\_str, num=0, text=''):  
 if bool\_const: ax = fig.gca(projection='3d')  
 else:  
 ax = fig.add\_subplot(1, 2,num,projection='3d'); self.setAxes(ax, a, b, z\_str)  
 if num == 1: ax.text2D(-0.2, -0.05, text, transform=ax.transAxes)  
 if plotType.name == 'surface':  
 ax.plot\_surface(x,y,z, cmap=cm.coolwarm, linewidth=0, antialiased=True)  
 elif plotType.name == 'contour': self.plotContours(ax, x, y, z, b)  
  
 def plotCtConst(self, n, a, b, h, q0, E, nu, numIters, tuple\_xy,   
 plotType, fname1, fname2, fname\_exe):  
 if os.path.isfile(fname2) or self.isChangedInputData(fname\_exe)   
 or not os.path.isfile(fname1):  
 cmd = '{0} {1} {2} {3} {4} {5} {6} {7}'. \  
 format(fname\_exe, n, a, b, h, q0, E, nu)  
 self.startProgram(fname1, fname2, n, a, b, h, q0, E, nu,   
 numIters, cmd, fname\_exe)  
 w = getListFromJSON(fname1, 'w')  
 if w:  
 z = np.array([[w[i][j] for i in range(n + 1)] for j in range(n + 1)])  
 try:  
 x = tuple\_xy[0]; y = tuple\_xy[1]; fig = plt.figure('Прогиб пластины')  
 self.plotSurfaceOrContour(fig, True, plotType, a, b, x, y, w, 'w')  
 plt.show()  
 except (ValueError, TypeError):  
 QMessageBox.warning(self, 'Ошибка',  
 'Поменяйте количество отрезков сетки')  
  
 def plotCtKarman(self, n, a, b, h, q0, E, nu, numIters, tuple\_xy,   
 plotType, fname1, fname2, fname\_exe):  
 if not os.path.isfile(fname2) or self.isChangedInputData(fname\_exe) or \  
 (not os.path.isfile(fname1) and not os.path.isfile(fname2)):  
 cmd = '{0} {1} {2} {3} {4} {5} {6} {7} {8}' \  
 .format(fname\_exe, n, a, b, h, q0, E, nu, numIters)  
 self.startProgram(fname1, fname2, n, a, b, h, q0, E, nu,   
 numIters, cmd, fname\_exe)  
 \_w = getListFromJSON(fname1, 'w'); \_r = getListFromJSON(fname2, 'r')  
 if \_w != None and \_r != None:   
 w = np.array([[\_w[i][j] for i in range(n + 1)] for j in range(n + 1)])  
 r = np.array([[\_r[i][j] for i in range(n + 1)] for j in range(n + 1)])  
 try:  
 x = tuple\_xy[0]; y = tuple\_xy[1]  
 fig = plt.figure(figsize=plt.figaspect(0.5),   
 num='Прогиб и реакция пластины')  
 if numIters == 0:  
 text = 'n = {0}\na = {1}\nb = {2}\nh = {3}\nq0 = {4}\n  
 E = {5}\n\u03bd = {6}\n\nпогрешность = 1e-6' \  
 .format(n, a, b, h, q0, E, nu)  
 else:  
 text = 'n = {0}\na = {1}\nb = {2}\nh = {3}\nq0 = {4}\n  
 E = {5}\n\u03bd = {6}\n\nчисло итераций = {7}' \  
 .format(n, a, b, h, q0, E, nu, numIters)  
 self.plotSurfaceOrContour(fig, False, plotType, a, b, x, y,   
 w, 'w', 1, text)  
 self.plotSurfaceOrContour(fig, False, plotType, a, b, x, y,   
 r, 'r', 2, text)  
 plt.show()  
 except (ValueError, TypeError):  
 QMessageBox.warning(self, 'Ошибка',  
 'Поменяйте количество отрезков сетки')  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 app = QApplication(sys.argv)  
 window = MainWindow()  
 sys.exit(app.exec\_())