СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Многие методы численного расчета сделаны на переходе от протяженной задачи к дискретной, т.е. к такой задачи, при котором не находят функции распределения той или иной физической величины, а могут найти лишь значения этой величины в некотором конечном наборе точек. Так же пространство, которые рассчитываются в задачи через параметры, разбивается на конечное число элементов, целиком заполняющих это пространство. Как правило, элементы могут являться простыми фигурами и для них возможно решить задачу. В пространственной задачи, элементы могут включать в себе объем, например, тетраэдр является таким элементом. Для плоских задач, либо пространственных задач, но теме или иным способом сводящихся к плоским, используются различные плоские полигональные фигуры.

На сегодняшний день, в современных задачах такого же класса разбиение включает в себя большое количество элементов, как правило не менее тысячи. Разрабатывать вручную такие разбиения невозможно, следовательно, в дополнение к основным расчетным алгоритмам должны быть созданы и алгоритмы генерации разбиения.

Анализ напряженно-деформированного состояния методом конечных элементов представляет из себя такую задачу, так как в процессе эксплуатации плоская конструкция-кронштейн подвергается различным нагрузкам, которые отрицательно влияют на прочность, долговечность, надёжность не только самой конструкции, но и на безопасность эксплуатации в целом. Нагрузки, действующие на кронштейн, зависят от различных факторов и, прежде всего, от материала и размеров конструкции, условий ее эксплуатации, способа закрепления. В результате неправильной эксплуатации, конструкция подвергается ударным воздействиям. Таким образом, необходим алгоритм, разбивающий полигональную фигуру. Треугольники получаемой сетки должны удовлетворять двум перечисленным выше требованием. Желательно, чтобы алгоритм был адаптивным и требовал, как можно меньшего участия человека.

Курсовой проект ставит целью написание специализированного программного обеспечения, способного произвести моделирование напряженно-деформированного состояния плоской конструкции-кронштейна во время его эксплуатации.

1 ОБЗОР ЧИСЛЕННЫХ МЕТОДОВ МОДЕЛИРОВАНИЯ В МЕХАНИКЕ

1.1 Понятие и виды моделирования

Моделирование биологических, химических и многочисленных физических явлений часто подводит к решению линейных и нелинейных уравнений или систем уравнений в частных производных. Есть традиционные математические возможности, которые позволяют достичь решение в определенных случаях, однако для решения конкретных проблем, которые возникают в науки и технике, необходимо воспользоваться численными методами. С ростом производительности ЭВМ численное моделирование приобретает специфическое значение, так как может позволить дополнить или даже изменить прямой эксперимент. Последний часто дорог, его постановка бывает трудной или не реальной.

Разработано множество методов численного решения уравнений в частных производных. Наибольше всего используются методы конечных разностей или конечных элементов.

Суть метода заключается в том, что область расчета (одно- , двух- или трехмерная), занимаемая конструкцией, разбивается на некоторое число малых, но конечных по размерам подобластей. Последние носят название конечных элементов (КЭ), а сам процесс разбивки называют дискретизацией. Пример такой области показан на рисунке 1.1, и пример разбивки на подобласти на рисунке 1.2.

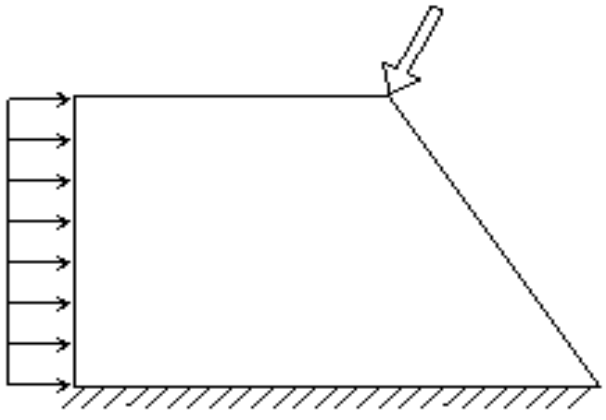

Рисунок 1.1 – Пример области на разбитие

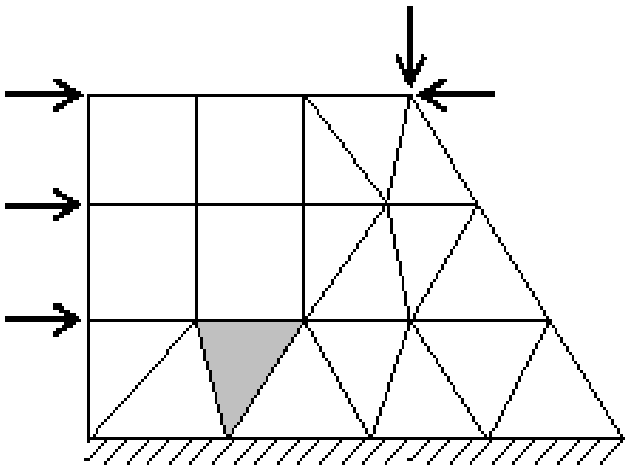

Рисунок 1.2. – Пример разбития области на подобласти

В зависимости от типа конструкции и характера ее деформации КЭ могут иметь различную форму. Так, при расчете стержневых систем (фермы, балки, рамы), КЭ представляют собой участки стержней; для двумерных континуальных конструкций (пластины, плиты, оболочки). Чаще всего применяют треугольные и прямоугольные (плоские или изогнутые) КЭ, а для трехмерных областей (толстые плиты, массивы) – КЭ в форме тетраэдра или параллелепипеда. В отличие от реального сооружения в дискретной модели конечные элементы связываются между собой только в отдельных точках (узлах) определенным конечным числом узловых параметров [6, с. 467].

Для этих элементов, названных конечными, аналитическими методами получаются точными или приближенные решения уравнений, которые описываются их напряженно-деформированное состояние (НДС). На основе таких решений составляются уравнения, позволяющие описать НДС всей конструкции. Как правило, это бывают системы алгебраических или дифференциальных уравнений второго порядка. Для того что бы составить и решить уравнения МКЭ, необходимо использовать вычислительные машины. Создан ряд программ для ЭВМ, которые позволяют реализовать МКЭ. К таким программам относятся *NASTRAN, ANSYS, DINA, MARC*. Благодаря этим программам хорошо демонстрированы и допускают самостоятельное использование их в широком круге специалистов. Перечисленные качества, такие как доступность программ МКЭ и физическая наглядность получаемых результатов обеспечили популярность МКЭ. Современный инженер обязан быть ознакомлен с основами МКЭ и уметь им пользоваться.

В зависимости от того, какие неизвестными являются основными, МКЭ осуществляется в форме метода перемещений, метода сил или смешанного метода. Самым популярным среди разработчиков программ МКЭ является метод

перемещений, который позволяет получить очень простые и универсальные алгоритмы. В качестве основных неизвестных в методе перемещений рассматриваются перемещения узловых точек расчетной схемы. Все остальные характеристики НДС находятся в независимости от узловых перемещений. В любой конструкции методом перемещений основными этапами являются:

– идеализация конструкции или выбора расчетной схемы;

– вычисление матриц жесткости (МЖ) отдельных элементов в любой удобной для этих целей местной системе координат;

– преобразование МЖ элементов в общую для своей конструкции систему координат;

– формирование с использованием МЖ элементов системы уравнений или равновесия конструкции;

– решение системы уравнений относительно узловых перемещений;

– вычисление деформаций элементов по известным узловым перемещениям и вычисление напряжений в элементах по известным деформациям;

Этот алгоритм в точности повторяет схему счета стержневых конструкций методом перемещений, с той лишь разницей, что в качестве конечных элементов могут использоваться как стержни так и элементы пластин, оболочек и массивных сред. Примеры построения расчетных схем МКЭ, часто называемых в литературе сетками конечных элементов показаны на рисунке 1.3.

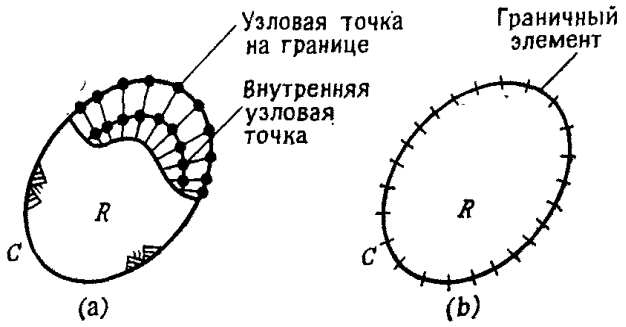

Рисунок 1.3 – Сравнение метода граничных элементов с МКЭ

Метод граничных элементов основан на использовании фундаментальных решений линейных дифференциальных уравнений с постоянными коэффициентами и, следовательно, в своем исходном виде можно применить к решению линейных задач в областях, содержащих однородные среды.

Все МГЭ строятся на основе общих принципов. При этом различают прямые и непрямые МГЭ.

В прямы*х* МГЭ искомыми переменными краевой задачи являются величины, имеющие реальный физический смысл, например, в задачах теории упругости – усилия и смещения, возникающие в элементах конструкции.

В непрямых МГЭ решение исходной задачи выражается через функции плотности, которые сами по себе не имеют реального физического смысла. После того как функции плотности найдены, значения реальных физических параметров задачи могут быть получены из них путем простого интегрирования.

Если область состоит из нескольких смежных подобластей, содержащих однородные, но различные среды, то в рамках МГЭ легко соединить между собой решения для отдельных подобластей. Но учет неоднородности среды и её анизотропии требует применения процедуры последовательных приближений. Похожая ситуация возникают и при использовании МГЭ для решения нелинейных задач.

В методе конечных элементов (МКЭ) учет неоднородности и анизотропии среды не вызывает принципиальных трудностей. Но в случае нелинейных задач при матричном представлении уравнений относительно узловых значений искомых функций элементы матриц зависят от этих значений, что также приводит к итерациям.

Известно, что названия исходных задач в виде граничных интегральных уравнений (ГИУ) приводит к уменьшению размерности на единицу, а именно для двумерных задач можно иметь одномерные ГИУ вдоль контура области, а для трехмерных двумерные ГИУ по поверхности, ограничивающей область. Нужно отметить, что некоторые задачи удается свести к одномерным ГИУ вдоль контура осевого сечения рассматриваемой области. Следовательно, следующие применение МГЭ для решения ГИУ позволяет решать системы линейных алгебраических уравнений (СЛАУ) с матрицами меньшего порядка, чем при использовании МКЭ для решения той же задачи. Однако матрицы. Которые формируются на основе МГЭ не содержат нулевых элементов.

Матрица, которая в противоположность сформирована на основе МКЭ, имеет наибольшее число нулевых элементов. Матрицу, оптимизируя нумерации узлов КЭ, можно свести к ленточной с минимально возможной шириной ленты. Благодаря этому, можно уменьшить затраты памяти ЭВМ и число арифметических операция при решении СЛАУ с ленточной матрицей. Так же вычисления каждого элемента матрицы требует меньшего числа операций, чем для элемента матрицы при её формировании на основе МГЭ, хотя при отсутствии в ГИУ интеграла по области объем вычислений также уменьшается [2, с. 105].

В МГЭ погрешности могут быть связаны лишь с аппроксимацией границы при помощи ГЭ и приближенном представлении на границе искомых и заданных функций, тогда как ГИУ является точной формулировкой искомого решения во всей области. В МКЭ же возникают дополнительные погрешности при представлении искомого решения во всей области. Для иллюстрации различий между этими типами вычислительных приемов можно сопоставить ме-

тоды граничных элементов с методами конечных элементов. Для легкости можно представить *R* двумерной плоской областью, которая ограничена контуром *C*. Метод конечных элементов требует, чтобы вся область была разбита на сетку элементов. Так же цель состоит в отыскании решения задачи в узлах сетки, решение же между узлами выражается в простой приближенной форме через значения в узлах. Связывая эти приближенные выражения с исходными дифференциальными уравнениями в частных производных, в конечном итоге можно прийти к системе линейных алгебраических уравнений, в которых неизвестные параметры – узловые значения в *R* – выражаются через известные величины в узлах сетки, находящихся на границе области. Эта система уравнений большая, но разряженная, т.е. хотя она и содержит большое число неизвестных параметров и, следовательно, большое число линейных уравнений, но каждое уравнение включает в явном виде только часть неизвестных параметров.

Проведенное сравнение показывает, что ни МКЭ, ни МГЭ не имеют абсолютного преимущества при решении широкого класса задач, но существуют типы задач, для которых применение одного из этих методов предпочтительнее.

1.2 Понятие о конечных элементах

Метод конечных элементов (МКЭ) – основной метод современной строительной механики, лежащий в основе подавляющего большинства современных программных комплексов, предназначенных для выполнения расчетов строительных конструкций на ЭВМ. В его основе лежат две главные идеи: дискретизация исследуемого объекта на конечное множество элементов и кусочно-элементная аппроксимация исследуемых функций.

Метод может полностью автоматизировать расчет механических систем, но требует большое количество выполнения вычислительных операция по сравнению с классическими методами механики деформируемого твердого тела. На сегодняшний день уровень развития вычислительной техники открывает широкие возможности для внедрения МКЭ в инженерную практику. Следовательно, знание основ МКЭ и современных программных средств, которые позволяют решать на его основе разнообразные задачи, также для инженера является абсолютно необходимыми [2, с. 32].

В МКЭ исследуемая конструкция мысленно разбивается на отдельные части – конечные элементы, соединяющиеся между собой в узлах. Совокупность соединенных между собой и прикрепленных к основанию конечных элементов образует расчетную схему, называемую конечно-элементной схемой или конечно-элементной моделью.

Каждый отдельно конечный элемент должен быть достаточно простым, чтобы имелась возможность легко определить перемещения и напряжения в любой его части по заданным перемещениям узлов. Связь между перемещениями узлов элемента и силами, действующими на них, задается при помощи матрицы жесткости элемента. Количество перемещений узлов элемента, которые однозначно определяют положение данного элемента, называют числом степеней свободы элемента.

Аналогично, для всей конечно-элементной схемы вводятся матрица жесткости системы K, или глобальная матрица жесткости, устанавливающая связь между перемещениями узлов системы и силами, действующими на них, а также число степеней свободы системы, или глобальное число степеней свободы - количество координат узлов системы, которые достаточно знать, чтобы однозначно определить положение всей системы. Обычно, все степени свободы представляются в виде вектора, называемого вектором узловых перемещений. Матрица жесткости системы формируется из матриц жесткости элементов.

Классификация конечных элементов, используемых в механике:

– простейшие конструкционные элементы, относятся элементы типа стержень, балка, труба, брус, панель, работающая на сдвиг. Все вариации элементов представлены на рисунке 1.4. Уравнения, описывающие данные элементы, выводятся из теоретических положений сопротивления материалов, то есть из упрощенных механических формулировок. Исторически первыми стали использоваться именно эти типы конечных элементов;

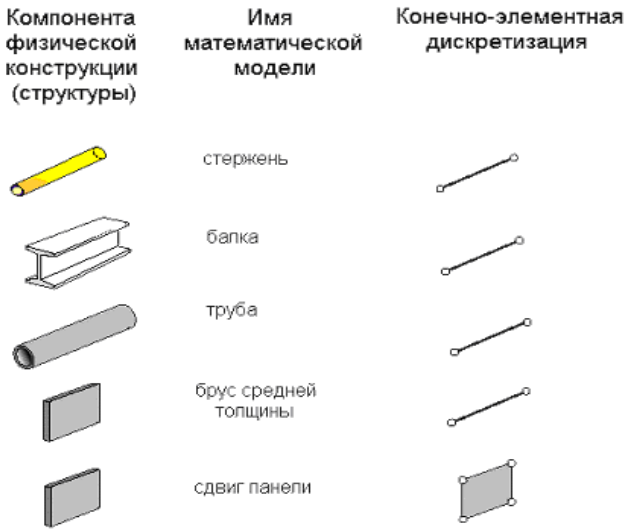

Рисунок 1.4 – Простейшие конструкционные элементы

– континуальные элементы, представляют собой конечные объемы или площади сплошной среды. Например, к континуальным элементам относятся

пластины, оболочки, осесимметричные элементы, трехмерные твердотельные элементы. Все вариации элементов представлены на рисунке 1.5. Уравнения, описывающие данный тип конечных элементов, получаются из общих соотношений механики сплошной среды и, в частности, теории упругости;

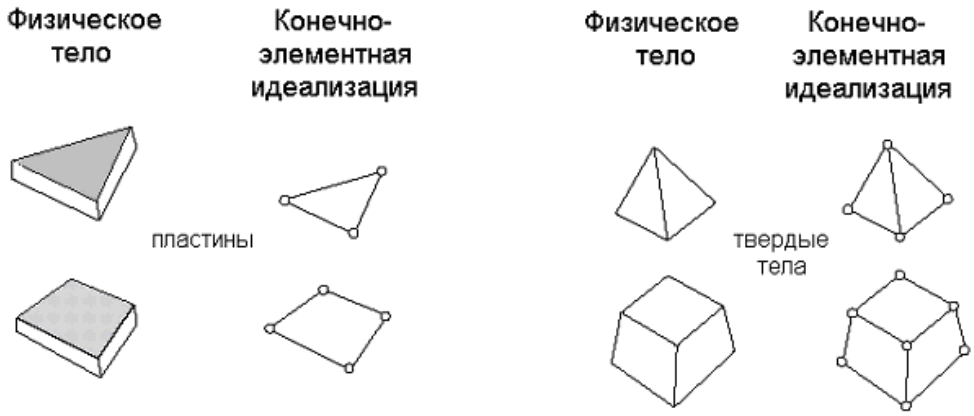

Рисунок 1.5 – Континуальные конечные элементы

– специальные элементы, обладают свойствами как конструкционных, так и континуальных элементов. Они выводятся из уравнений механики сплошной среды, но включают в себя некоторые особенности, непосредственно связанные с физическими особенностями решаемых задач. В качестве примера можно привести следующие специальные элементы: элемент с трещиной для задач механики разрушения; многослойная панель; бесконечные и полубесконечные элементы; контактные и штрафные элементы; абсолютно твердотельные элементы. Все вариации элементов представлены на рисунке 1.6;

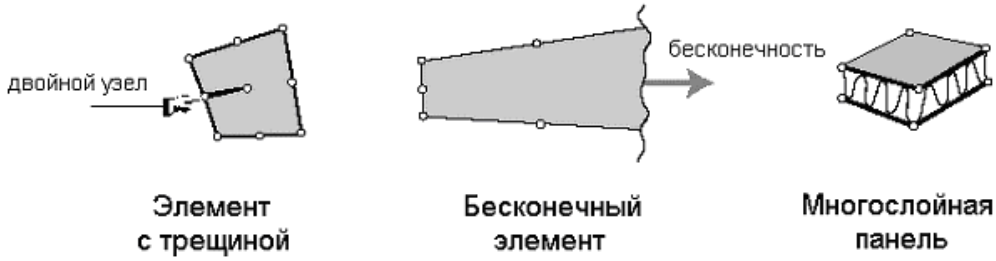

Рисунок 1.6 – Специальные конечные элементы

– макроэлементы, представляют собой более сложный тип конечных элементов. Как правило, они получаются путем сборки из более простых кон-

струкционных элементов. Число таких элементов, входящих в макроэлемент, как правило, невелико, представлено на рисунке 1.7;

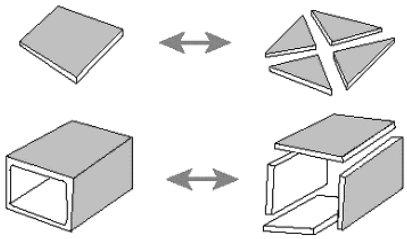

Рисунок 1.7 – Макроэлементы

– подструктуры, можно определить, как макроэлементы с явно выраженными структурными особенностями или функциями. Как правило, они получаются путем разделения полной конструкции на функциональные компоненты. Например, крылья и фюзеляж самолета, пролет и тросы подвесного моста. Заметим, что различия между понятиями полной конструкции, подструктур и макроэлементов не всегда очевидны и четко определены. Поэтому часто используется понятие суперэлемента как обобщенного названия для всех типов макроэлементов или подструктур, представляющих собой комбинацию простейших конструкционных элементов.

Виды конечных элементов:

– *прямой метод* аналогичен матричному методу перемещений для стержневых систем, в основе его лежат положения, которые использовались на ранней стадии развития МКЭ. Этот метод удобен своей простотой и очевидным геометрическо-физическим значением отдельных шагов аппроксимации. Соотношения для КЭ здесь строятся непосредственно на основе трех групп уравнений (трех сторон задачи): статической, геометрической и физической. Однако область применения прямого метода весьма ограничена: его можно использовать лишь для конечных элементов простой геометрии с малым числом степеней свободы в узле;

– *вариационный метод* основан на принципах стационарности некоторой переменной, зависящей от одной или нескольких функций (такая переменная носит название функционала). Применительно к механике деформируемого твердого тела эта переменная представляет собой потенциальную (функционал Лагранжа) или дополнительную (функционал Кастилиано) энергию системы или формируется на основе этих двух энергий (функционалы Хеллингера-Рейсснера, Ху-Вашицу). Если в функционал подставить аппроксимирующие выражения искомых функций и применить к нему экстремальные принципы (соответственно принцип Лагранжа, принцип Кастилиано и т. д.), получим си-

стему алгебраических уравнений, решением которой будут значения узловых неизвестных. В отличие от прямого, вариационный метод может одинаково успешно применяться как к простым, так и сложным задачам;

– *метод невязок* представляет собой наиболее общий подход к построению основных соотношений МКЭ. Этот метод целесообразно применять при решении задач, у которых трудно или невозможно сформулировать вариационное уравнение, т.е. функционал. Суть метода взвешенных невязок заключается во введении некоторой невязки – отклонении приближенного аппроксимативного решения от точного решения дифференциальных уравнений для данной задачи. Чтобы получить ”наилучшее” решение, необходимо минимизировать некоторый интеграл от невязок по расчетной области. Для повышения эффективности в подынтегральное выражение наряду с самой невязкой обычно вводится так называемая весовая функция, в этом случае метод называется методом взвешенных невязок. Выбор схемы минимизации и весовых функций определяет различные варианты метода невязок. Наиболее часто применяемые из них – это метод Галеркина, который приводит к тем же уравнениям, что и вариационный подход, а также метод наименьших квадратов [3, с. 24];

– *метод энергетического баланса (метод Одена)* основан на балансе различных видов энергии, записанном в интегральной форме. Этот метод успешно применяется при решении нелинейных и динамических задач.

1.3 Атрибуты конечных элементов и построение сетки

Одним из важнейших этапов в процессе конечно-элементного моделирования является создание сетки конечных элементов. Существуют основные типы конечных элементов и их свойства, так называемые атрибутами элементов.

Собственная размерность. Положения конечных элементов могут описываться одной, двумя или тремя пространственными координатами в зависимости от размерности задачи. Соответствующее число внутренних или локальных координат называется собственной размерностью элемента.

Узловые точки. Узловые точки, или узлы необходимы для описания геометрии элемента и задания физических степеней свободы (числа неизвестных функций). Как правило, узлы расположены в угловых или крайних точках элемента, но иногда вводят дополнительные узлы, расположенные внутри элемента. Число узлов связано с порядком аппроксимации, который обеспечивает данный конечный элемент. Элементы, имеющие только угловые узлы, называются линейными и обеспечивают линейную интерполяцию геометрии и функций. Элементы, имеющие дополнительные узлы на своих границах между угловыми точками, могут обеспечивать квадратичную или даже кубическую интерполяцию. При наличии современных автоматических генераторов конечно-

элементных сеток часто бывает проще и удобнее разбить конструкцию на большое число линейных элементов простой формы, чем использовать элементы высокого порядка, требующие для построения сетки значительной работы вручную. В то же время квадратичная и кубическая интерполяции обеспечивают более высокую точность расчета.

Геометрия элемента. Геометрия элемента определяется расположением узловых точек. Большинство элементов, используемых в расчетах, имеют простую геометрическую форму. Например, в одномерном случае элементы обычно представляют собой прямолинейные отрезки или сегменты кривых линий; в двумерном случае элементы имеют трех- или четырехстороннюю; в трехмерных задачах наиболее распространены такие геометрические фигуры, как тетраэдры, гексаэдры и призмы.

Степени свободы. В качестве степеней свободы могут фигурировать как узловые значения неизвестной функции, так и ее производные по пространственным координатам в узлах. В первом случае элементы относятся к типу лагранжевых элементов, во втором - эрмитовых. Например, в простейшей задаче о растяжении стержня неизвестной функцией являются продольные смешения узлов стержня. Соответственно в качестве степеней свободы выступают узловые значения данной функции и, следовательно, конечный элемент относится к лагранжевому типу. Наоборот, в задаче об изгибе стержня неизвестной функцией являются поперечные смешения узлов центральной оси стержня, а в качестве степеней свободы используются как узловые значения самой функции. Таким образом, конечный элемент, применяемый в расчетах стержня на изгиб, относится к типу эрмитовых элементов. Заметим также, что данные обозначения происходят от названий полиномов Лагранжа и Эрмита, широко используемых в прикладной математике для интерполяции функций по узловым значениям [1, с. 127].

Определяющие соотношения. Для конечных элементов, используемых в механических расчетах, определяющее соотношение задает поведение материала, из которого изготовлена конструкция. Например, в качестве такого соотношения во многих случаях используется обобщенный закон Гука, связывающий тензоры деформаций и напряжений в точке.

2 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ НАПРЯЖЕННО-ДЕФОРМИРОВАННОГО СОСТОЯНИЯ ПЛОСКОЙ КОНСТРУКЦИИ-КРОНШТЕЙНА

2.1 Постановка задачи

Необходимо разработать приложение, которое будет моделировать напряженно-деформированное состояние плоской конструкции-кронштейна, провести аналогичный расчет в системе *Ansys*.

Для решения задачи нужно: разбить область на конечные элементы, пронумеровать узлы и элементы, задать характеристики материала и граничные условия.

Исходными данными для проекта является схема плоской конструкции с приложенной распределённой нагрузкой (Приложения А), значения характеристик материала (модуль упругости – 110E + 9 Па, коэффициент Пуассона – 0,35), нагрузка 6000Па.

2.2 Описание математической модели

Для решения задачи используется метод конечных элементов, который был описан выше. Деталь разбивается на треугольные конечные элементы, как показано на рисунке 2.1.

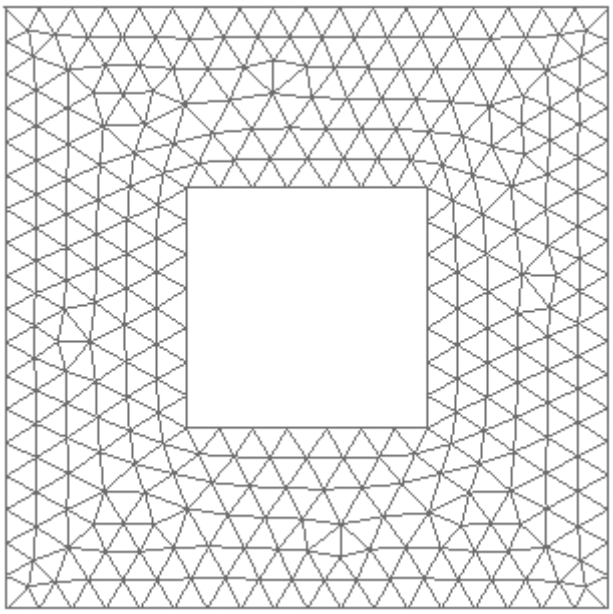

Рисунок 2.1 – Конечно-элементное представление тела

Каждый узел перемещения имеют два компонента как показано по формуле (2.1):

$$\{\delta_i\} = \begin{Bmatrix} u_i \\ v_i \end{Bmatrix}. \qquad (2.1)$$

Шесть компонентов перемещений узлов элемента позволяют образовать вектор перемещений:

$$\{\delta\} = \begin{Bmatrix} u_i & v_i & u_j & v_j & u_k & v_k \end{Bmatrix}^T. \qquad (2.2)$$

Перемещение любой точки внутри конечного элемента определяется соотношениями (2.3) и (2.4):

$$u(x,y) = u = N_i u_i + N_j u_j + N_k u_k. \qquad (2.3)$$

$$v(x,y) = V = N_i v_i + N_j v_j + N_k v_k. \qquad (2.4)$$

При объединении (2.3) и (2.4) в одно уравнение получается следующее соотношение:

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = [N]\{\delta\} = \begin{bmatrix} N_i & 0 & N_j & 0 & N_k & 0 \\ 0 & N_i & 0 & N_j & 0 & N_k \end{bmatrix} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_k \\ v_k \end{Bmatrix}. \qquad (2.5)$$

Перемещения и деформация связаны между собой следующим образом:

$$\{\varepsilon\} = \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \gamma_{xy} \end{Bmatrix} = \begin{pmatrix} \partial/\partial x & 0 \\ 0 & \partial/\partial y \\ \partial/\partial y & \partial/\partial x \end{pmatrix} \begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{Bmatrix} \dfrac{\partial u}{\partial x} \\ \dfrac{\partial v}{\partial y} \\ \dfrac{\partial u}{\partial y} + \dfrac{\partial u}{\partial x} \end{Bmatrix}. \qquad (2.6)$$

Далее необходимо подставить формулу (2.5) и (2.6) и получится соотношение по формуле (2.7):

$$\{\varepsilon\}=\begin{Bmatrix} \frac{\partial N_i}{\partial x} & 0 & \frac{\partial N_i}{\partial x} & 0 & \frac{\partial N_g}{\partial x} & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 & \frac{\partial N_i}{\partial y} & 0 & \frac{\partial N_g}{\partial y} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & \frac{\partial N_g}{\partial y} & \frac{\partial N_g}{\partial x} \end{Bmatrix} \begin{Bmatrix} u_i \\ v_i \\ u_j \\ v_j \\ u_g \\ v_g \end{Bmatrix}. \qquad (2.7)$$

Соотношение (2.7) можно представить в виде:

$$\{\varepsilon\} = [B]\{\delta\}. \qquad (2.8)$$

Матрица [B] – градиентная матрица вида (2.9):

$$[B]=\frac{1}{2A}\begin{Bmatrix} b_i & 0 & b_j & 0 & b_k & 0 \\ 0 & c_i & 0 & c_j & 0 & c_k \\ c_i & b_i & c_j & b_j & c_k & b_k \end{Bmatrix}. \qquad (2.9)$$

Функции формы $N_{i,j,k}$ линейно зависят от координат x, y, и следовательно, градиентная матрица не зависит от координат точки внутри конечного элемента, и деформации и напряжения внутри конечного элемента в этом случае постоянны.

При плоском деформированном состоянии в изотропном материале матрица упругих постоянных [D] определяется по формуле (2.10):

$$[D]=\frac{E(1-\nu)}{(1+\nu)(1-2\nu)}\begin{bmatrix} 1 & \frac{\nu}{1-\nu} & 0 \\ \frac{\nu}{1-\nu} & 1 & 0 \\ 0 & 0 & \frac{1-2\nu}{2(1-\nu)} \end{bmatrix}, \qquad (2.10)$$

где E – модуль упругости,

ν – коэффициент Пуассона.

Матрица жесткости конечного элемента имеет вид:

$$\left[K^e\right]=\left[B^e\right]^T\left[D^e\right]\left[B^e\right]h^e A^e, \qquad (2.11)$$

где h^e – толщина,

A^e – площадь элемента.

Уравнение равновесия *i*-го узла имеет вид:

$$\sum_{e=1}^{m}\sum_{j=1}^{t}\left[K_{ij}^e\right]\left\{U_j\right\}=\left\{F_i\right\}. \qquad (2.12)$$

Для учета условий закрепления существует следующий метод. Пусть имеется некоторая система *N* уравнений (2.13):

$$\begin{bmatrix} K_{11} & K_{12} & K_{13} & \cdot & K_{1n} & \cdot & K_{1N} \\ K_{21} & K_{22} & K_{23} & \cdot & K_{2n} & \cdot & K_{2N} \\ K_{31} & K_{32} & K_{33} & \cdot & K_{3n} & \cdot & K_{3N} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ K_{n1} & K_{n2} & K_{3n} & \cdot & K_{nn} & \cdot & K_{nN} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ K_{N1} & K_{N2} & K_{N3} & \cdot & K_{Nn} & \cdot & K_{NN} \end{bmatrix}\begin{Bmatrix} U_1 \\ U_2 \\ U_3 \\ \cdot \\ U_n \\ \cdot \\ U_N \end{Bmatrix}=\begin{Bmatrix} F_1 \\ F_2 \\ F_3 \\ \cdot \\ F_n \\ \cdot \\ F_N \end{Bmatrix}. \qquad (2.13)$$

В случае, когда одна из опор неподвижна, т.е. $U_i=0$, используют следующую процедуру. Пусть $U_2=0$, тогда:

$$\begin{bmatrix} K_{11} & 0 & K_{13} & \cdot & K_{1n} & \cdot & K_{1N} \\ 0 & 1 & 0 & \cdot & 0 & \cdot & 0 \\ K_{31} & 0 & K_{33} & \cdot & K_{3n} & \cdot & K_{3N} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ K_{n1} & 0 & K_{3n} & \cdot & K_{nn} & \cdot & K_{nN} \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ K_{N1} & 0 & K_{N3} & \cdot & K_{Nn} & \cdot & K_{NN} \end{bmatrix}\begin{Bmatrix} U_1 \\ 0 \\ U_3 \\ \cdot \\ U_n \\ \cdot \\ U_N \end{Bmatrix}=\begin{Bmatrix} F_1 \\ 0 \\ F_3 \\ \cdot \\ F_n \\ \cdot \\ F_N \end{Bmatrix}. \qquad (2.14)$$

То есть соответствующие строка и столбец задаются нулевыми, а диагональный элемент – единичным. Следовательно, приравнивается нулю и F_2.

Для решения полученной системы необходимо выбрать метод Гаусса. Алгоритм решения методом гаусса подразделяется на два этапа:

– *прямой ход*: при помощи операций над строками система превращается в треугольную форму, или указывается, что система несовместна: среди элементов первого столбца матрицы, выбирается ненулевой элемент, затем он перемещается на верхнее положение перестановкой строк. После этого вычитается, получившаяся после изменений, первая строка, домноженная на размер, равную отношению первого элемента каждой из этих строк к первому элементу первой строки, зануляя столбец под ним.

После того, как указанные операции были осуществлены, первая строка и первый столбец внутренне вычёркиваются и продолжают пока не останется матрица нулевого размера;

– *обратный ход*: его суть состоит в том, чтобы показать, приобретённые основные переменные через небазисные и создать фундаментальную систему решений, либо показать в численном виде единственное решение системы линейных уравнений.

Рисунок 2.2 – Графическая схема алгоритма

Благодаря графической схеме алгоритма, можно разработать программу, которая позволит вычислять деформацию с помощью МКЭ.

3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ АНАЛИЗА НАПРЯЖЕННО-ДЕФОРМИРОВАННОГО СОСТОЯНИЯ ПЛОСКОЙ КОНСТРУКЦИИ-КРОНШТЕЙНА

3.1 Исходные данные

В программе исходные данные берутся из файла, так как эти файлы были получены благодаря моделирование в программе *ANSYS*. Исходными данными для решения задачи являются:

– файл с закрепленными узлами;
– файл с узлами, к которым приложена сила;
– файл узлов из *ANSYS*;
– файл элементов из *ANSYS*;
– файл с данными для построения модели;
– файл с материалами.

Эти файлы нужны для того, чтобы можно было запустить свою деталь в программе, так как в этих в файлах находятся данные, которые были рассчитаны и смоделированы в *ANSYS*. Файл с закрепленными узлами показывает, где была закреплена делать, что бы при расчетах деформации деталь не могла переместиться, также нужно знать, где именно была приложена сила, для этого нужен файл с узлами, к которым приложена сила, благодаря этому файлу программа может вычислить куда была приложена сила, чтобы посчитать деформацию тела. Но перед вычислением деформации и напряженности, необходимо указать материал детали, для этого существует файл с материалами, который позволит узнать из какого материала сделана деталь, так же необходимо знать вид конечного элемента, ведь виды конечного элемента бывает разные и от вида зависит какая будет построена сетка. Благодаря исходным данными программа может без проблем приложить нагрузку на закрепленное тело, построить сетку и произвести расчеты по которым будет определенны результаты вычисления.

3.2 Проектирование и реализация задачи

Для реализации математической модели был выбран язык программирования *C#*. Листинг программы приведен в приложении Б.

Для проектирования пользовательского интерфейса была использована технология *Windows Form*. Интерфейс обеспечивает взаимодействие человека с программной системой. Интерфейс служит переводчиком в диалоге пользователя и программы. Разумно спроектированный интерфейс пользователя очень

важен для успешной работы программы, так как благодаря удобному интерфейсу пользователь с легкостью сможет посчитать математическую модель за короткий промежуток времени. Сложный в применении интерфейс создает предпосылки для многочисленных ошибок пользователя. Нечеткие и двусмысленные фразы могут запутать пользователя, что может привести к искажению данных и формированию неправильных результатов.

В реализованной программе используются такие элементы пользовательского интерфейса, которые показаны в таблице 3.1:

Таблица 3.1 – Элементы графического интерфейса пользователя

Элементы	Описание
Окна	Отображают на экране различную информацию
Кнопки	Реализуют различные функции
Поля ввода	Реализуют текстовый ввод данных в поле

На рисунке 3.1 изображен шаблон пользовательского интерфейса для программы «*Pipe_segment*». В данном окне производится анализ напряженно-деформируемого состояния литого диска автомобильного колеса. Для работы с данными созданы соответствующие кнопки. При нажатии на кнопку «Построить сетку» произойдет построение сетки конечными элементами на деталь считывая с файла с узлами. При нажатии на кнопку «Решение» произойдет расчет детали на деформацию и напряженность и результат будет выведен на экран. Также если нужно увидеть деформацию, то нужно нажать на кнопку «Деформация» и тогда на экране осуществиться графический вывод деформации детали как показано в приложении В рисунок В.2. Если нужно отобразить напряженность, то необходимо нажать на кнопку «Напряжение», следовательно, на экране осуществиться графический вывод напряженности как показано в приложении В рисунок В.3, так же если необходимо отобразить смещение, то нужно нажать на кнопки «Перемещение по OX», «Перемещение по OY», следовательно, на экране осуществиться графический вывод смещения как показано в приложении В, рисунок В.4, рисунок В.5.

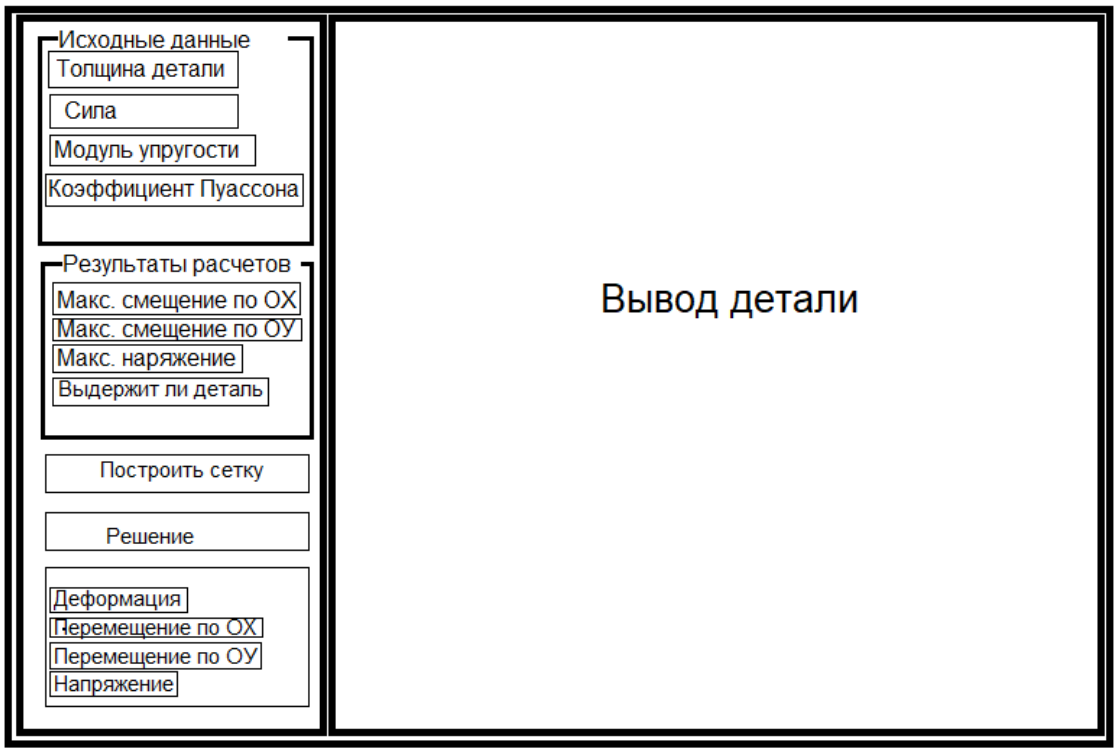

Рисунок 3.1 – Шаблон пользовательского интерфейса

3.3 Структура программного комплекса

Для реализации конечно-элементной сетки в программе были разработаны классы *Node* и *Element*, которые хранят характеристики узлов и элементов соответственно. Данные, для создания сеток используются из файла узлов из *ANSYS*, а также файл с узлами, к которым приложена сила.

В классе *Node* реализованы:

– *_displacementX* – признак перемещения по оси OX;
– *_displacementY* – признак перемещения по оси *OY*;
– *_forceX* – горизонтальная составляющая силы в узле;
– *_forceY* – вертикальная составляющая силы в узле;
– *_offsetX* – смещение по оси *OX*;
– *_offsetY* – смещение по оси *OY*;
– *_posX* – признак перемещения по оси *OX*;
– *_posY* – признак перемещения по оси *OY*;
– *SetFixed* – метод закрепления узлов;
– *SetForce* – метод закрепления сил.

В классе *Elements* реализованы:

– *_n1* – номер узла в 1-й вершине;
– *_n2* – номер узла во 2-й вершине;
– *_n3* – номер узла в 3-й вершине;

– *_density* – плотность материала элемента;
– *_elasticity* – модуль Юнга;
– *_factor_Puason* – коэффициент Пуассона.

В классе *FileHelper* реализованы:
– *MaterialsSource* – файл с материалами;
– *PSModelElementsSource* – файл элементов из *ANSYS*;
– *PSModelNodesSource* – файл узлов из*ANSYS*;
– *PSModelForcedNodesSource* – файл с узлами с приложенной силой;
– *PSModelFixedSource* – файл с закрепленными узлами.

Данные элементы позволяют передать в программу данные из *ANSYS* которые затем можно изменять в созданном программном обеспечении.

4 ВЕРИФИКАЦИЯ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

4.1 Моделирование задачи в пакете *ANSYS*

Верификация – это процесс оценки системы или её компонентов с целью определения того, удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа. То есть, выполняются ли задачи, цели и сроки по разработке продукта. Верификация проверяет соответствие одних создаваемых в ходе разработки и сопровождения ПО артефактов другим, ранее созданным или используемым в качестве исходных данных, а также соответствие этих артефактов и процессов их разработки правилам и стандартам. В частности, верификация проверяет соответствие между нормами стандартов, описанием требований (техническим заданием) к ПО, проектными решениями, исходным кодом, пользовательской документацией и функционированием самого ПО.

Для того чтобы верифицировать результаты, нужно смоделировать задачу в *ANSYS*.

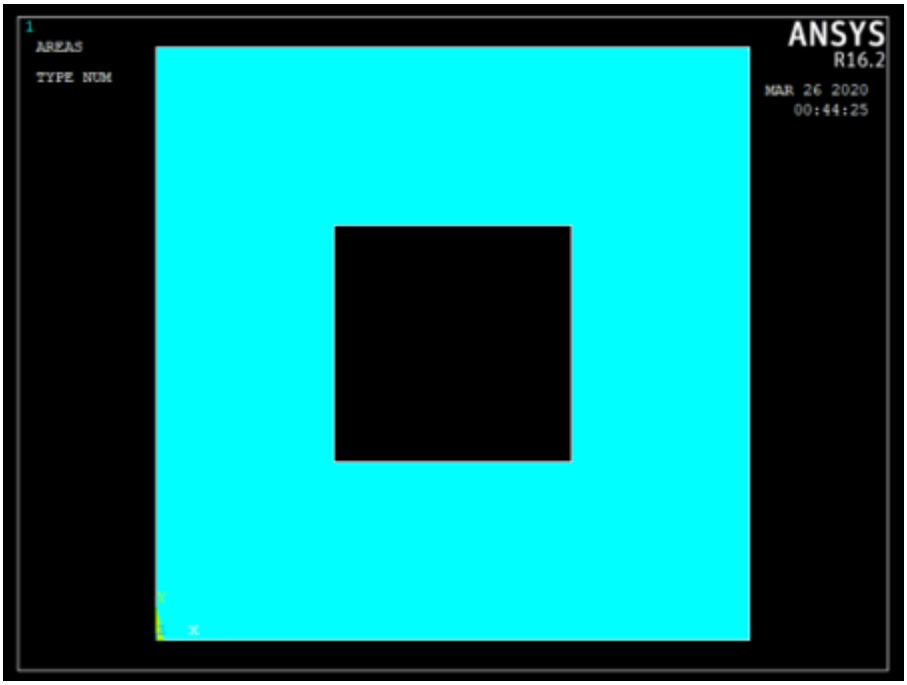

Рисунок 4.1 – Построенная делать

После того как была построена делать и сделано отверстие нужно выбрать тип конечного элемента, для этого нужно нажать на *Preprocessor,* далее выбрать *Element Type,* далее нажать на *Add/Edit/Delete* выбрать тип конечного элемента и нажать *Add* как показано на рисунке 4.2 и рисунке 4.3.

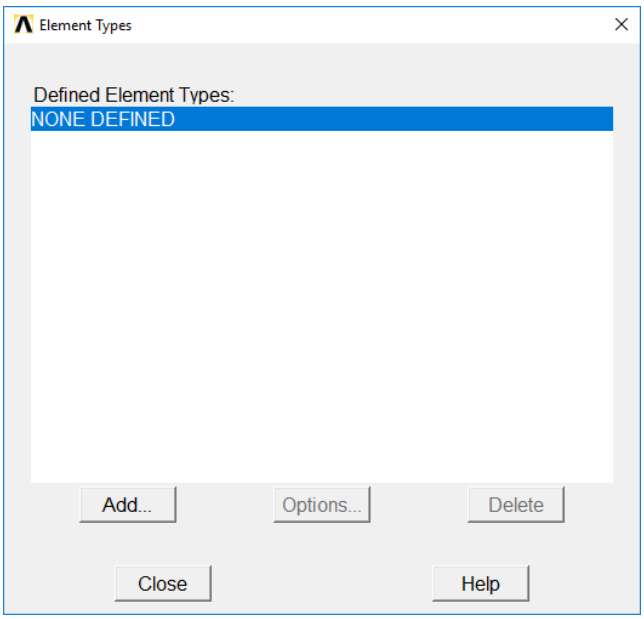

Рисунок 4.2 – Окно *ElementType*

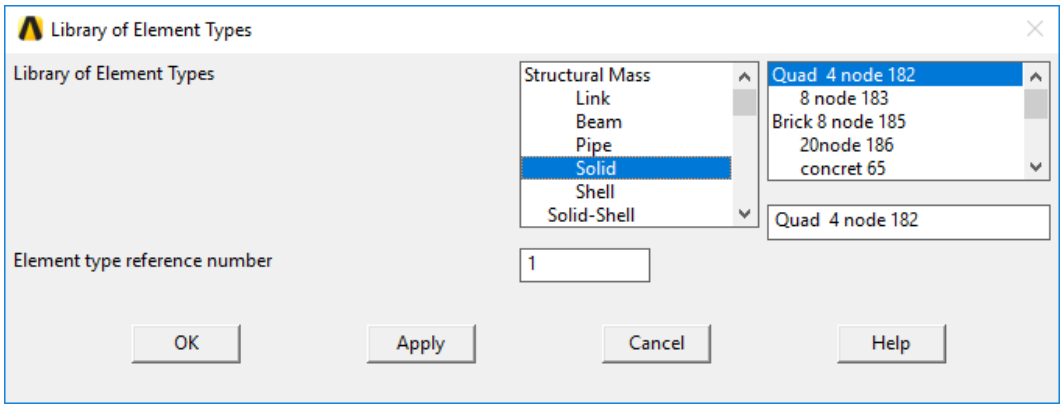

Рисунок 4.3 – Выбор тип конечного элемента

После того как был выбран тип конечного элемента, нужно задать ему материал, так как что бы построить сетку необходимо задать тип конечного элемента и материал, для этого нужно нажать на *Preprocessor*, выбрать пункт *Material props*, далее откроется окно, где нужно выбрать пункт *Structural*, далее нажать на *Liner*, потом выбрать *Elastic* и нажать на *Isotropic*, далее откроется окно где нужно задать характеристики материала как показано на рисунке 4.4.

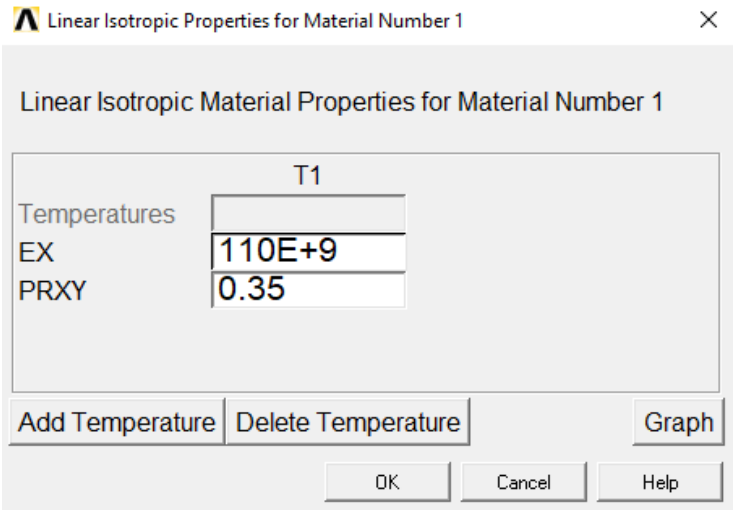

Рисунок 4.4 – Процесс задачи свойств и характеристик материал

После того как была нарисована деталь, выбран тип конечных элементов, задан материал, нужно построить конечно-элементную сетку с треугольными элементами. Для этого нужно выбрать пункт *Preprocessor*, далее выбрать пункт *Meshing* и нажать на *MeshTool*.

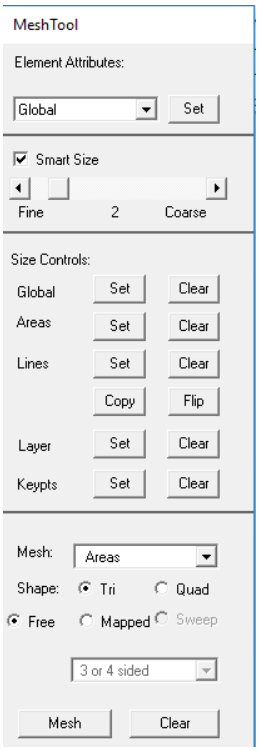

Рисунок 4.5 – Вид окна выбора работы по деталям

После нажатия появиться окно как показано на рисунке 4.5, где нужно будет указать параметры для построения конечно-элементной сетки и нажать на кнопку *Mesh*. После нажатия будет построена сетка как показано на рисунке 4.6.

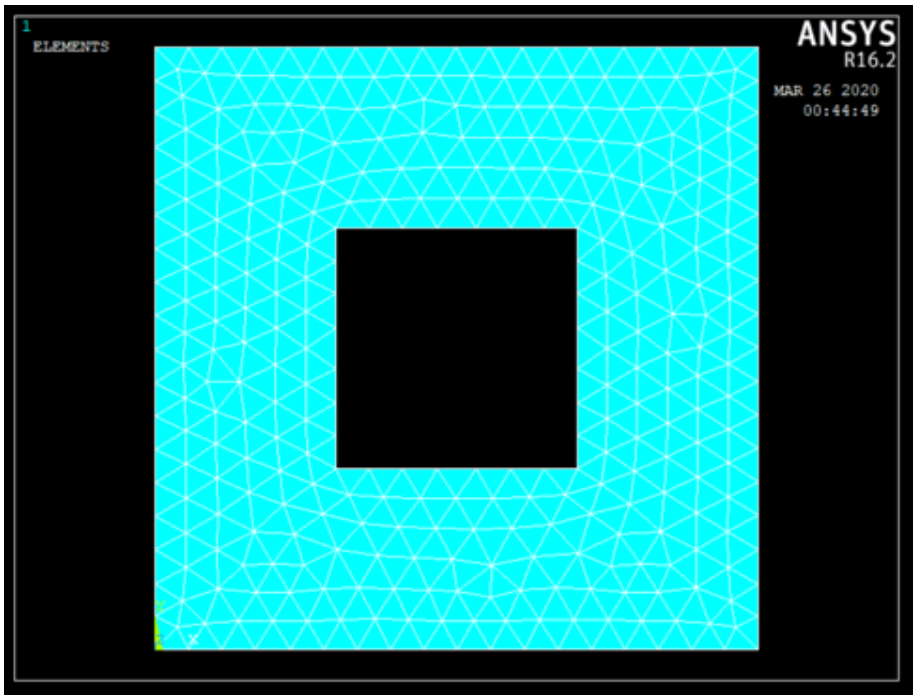

Рисунок 4.6 – Конечно-элементная сетка

После того как была построена сетка, необходимо закрепить деталь и приложить нагрузку, чтобы можно было произвести расчеты на деформацию и напряженность.

После того как была закреплена деталь и приложена к ней нагрузка, необходимо произвести расчеты детали. Для этого нужно выбрать пункт *Solution*, далее выбрать пункт *Solve* и нажать на *Current LS*. После этого будет произведен расчёт и можно будет посмотреть результаты. Для просмотра результата деформации необходимо выбрать пункт *General Posproc*, далее выбрать *Plot Results* и нажать на *Deformed Shape*. После этого будет отображена деформация как показано в приложении Б на рисунке Б.1. Если нужно будет посмотреть результаты напряжения и смещения по осям то нужно выбрать пункт *General Posproc*, далее выбрать *Plot Results*, далее выбрать *Nodal Solution* и нажать на *DOF Solution*, после этого нужно выбрать пункт который нужно отобразить, результаты будут отображены в приложении Б.

4.2 Исследование полученных результатов

В результате проделанной работы было разработано приложение, которой исследует заданную деталь. Внешний вид программы приведен в приложении В. Данное приложение, разбивает деталь на конечные элементы и находит перемещения в узлах при заданных нагрузках и закреплениях. Так же программа

подстроена на изменение размера детали чтобы найти оптимальную по размеру деталь.

Результат выполнения программы имеет расхождения с пакетом *ANSYS* в пределах допуска, что говорит о правильной работе программы. Программа проста в использовании, имеет понятный пользователю интерфейс и не требует затрат на поддержание своей работы.

Таблица 4.1 – Данные для сравнения

	ANSYS	Разработанное приложение	% расхождения результатов
Максимальное смещение по *X*	0.24160323	0,2047485	18%
Максимальное смещение по *Y*	0.120826832	0,1078811	12%
Максимальное напряжение	15.7461675	14,99635	5%

Расхождение по напряжению не превышает 5%, что говорит о правильной работе приложения и достаточной точности продукта. Есть расхождения в пунктах максимальное смещение по осям, но так как основной задачей было найти максимальное напряжение, то это расхождение не является важным показателем.

Программа выдает вывод о том, выдержит ли нагрузку деталь или нет.

Разработанное программное обеспечение возможно использовать без изменений не только для расчета плоской конструкции-кронштейна, но и для расчета любой детали. Программу возможно использовать в организациях занимающихся расчетами деталей или конструкций на прочность.

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта были закреплены знания о методе конечных элементов, получены навыки работы в компьютерном моделировании.

В соответствии с заданием курсового проекта была построена модель рассматриваемой системы, реализовано приложение, которое моделирует напряженно-деформированное состояние исследуемой конструкции. В программе реализован графический интерфейс пользователя.

Разработанное приложение позволяет достаточно быстро и эффективно исследовать модели, решать задачу деформации плоской конструкции под действием нагрузки, используя метод конечных элементов. Программа была протестирована, значения ее работы были сравнены с результатами моделирования в пакетах *ANSYS*. Полученные значения перемещений были представлены численно и графически.

Погрешность результатов находится в пределах нормы.

Разработанный программный комплекс позволяет достаточно быстро рассчитывать перемещения в узлах плоской конструкции. Он удобен для использования и прост в обращении.

СПИСОК ИССПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Каплун, А. Б. *ANSYS* в руках инженера: Практическое руководство / А. Б. Каплун, Е. М. Морозов, М. А. Олферьева ; под ред. А. Б. Каплун. – 3-е изд. – СПб. : Питер, 2015. – 266 с.
2. Чикуров, Н. Г. Моделирование технических систем : учебник / Н. Г. Чикуров. – 4-е изд., перераб. и доп. – Уфа : Уфимск. гос. авиац. техн. ун-т, 2014. – 129 с.
3. Шимановский, А. О. Применение метода конечных элементов в решении задач прикладной механики / А. О. Шимановский, А. В. Путято. – 2-е изд. – Гомель : БелГУТ, 2016. – 63 с
4. Варвак, П.М. Справочник по теории упругости : справочник / П. М. Варвак. – 4-е изд. – Новосибирск : НГУ, 2013. – 403 с.
5. Метод Гаусса для чайников. – Электрон. данные. – Режим доступа: https://zaochnik.ru/blog/metod-gaussa-dlya-chajnikov-reshaem-slau-legko/. – Дата доступа: 22.03.2020.
6. Бахвалов, Н. С. Численные методы : учебник / Н. С. Бахвалов. – 5-е изд. – М. : Наука, 2016. – 568 с.
7. Понятие о конечных элементах. – Электрон. данные. – Режим доступа: http://www.stroitmeh.ru/lect32.htm. – Дата доступа: 15.03.2020.

ПРИЛОЖЕНИЕ А
(обязательное)
Листинг программного кода

Program.cs

```
using System;
using System.Windows.Forms;

namespace CourseProject
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Node.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace CourseProject
{
    class Node
    {
        float x;
        float y;
        int index;
        bool fixation;

        public Node() { }

        public Node(int _index, float _x, float _y, bool _fixation)
        {
            index = _index - 1;
            x = _x;
            y = _y;
            fixation = _fixation;
        }

        public float X
        {
            get { return (x); }
            set { x = value; }
        }
```

```
        public float Y
        {
            get { return (y); }
            set { y = value; }
        }

        public int Index
        {
            get { return (index); }
            set { index = value; }
        }

        public bool Fixation
        {
            get { return (fixation); }
            set { fixation = value; }
        }
    }
}
```

Element.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;

namespace CourseProject
{
    class Element
    {
        Node n1;
        Node n2;
        Node n3;
        SolidBrush colorElement;
        int index;
        float[,] a;
        float[,] a_1;
        float[,] q;
        float[,] k;
        float[,] b;
        float[,] d;
        float[,] sig;
        float[,] e;
        float[,] stress;
        float s;

        public Element(int _index, Node _n1, Node _n2, Node _n3)
        {
            index = _index;
            n1 = _n1;
            n2 = _n2;
            n3 = _n3;
            sig = new float[6, 1];
        }

        public void CreateMatrix(float t, float elasticModulus, float poissonsRatio)
        {
            CreateA();
            CreateQ();
            CreateD(elasticModulus, poissonsRatio);
```

```
    CreateB();
    CreateK(t);
}

void CreateA()
{
    a = new float[6, 6];
    a_1 = new float[6, 6];

    a[0, 0] = 1;
    a[0, 1] = n1.X;
    a[0, 2] = n1.Y;

    a[1, 3] = 1;
    a[1, 4] = n1.X;
    a[1, 5] = n1.Y;

    a[2, 0] = 1;
    a[2, 1] = n2.X;
    a[2, 2] = n2.Y;

    a[3, 3] = 1;
    a[3, 4] = n2.X;
    a[3, 5] = n2.Y;

    a[4, 0] = 1;
    a[4, 1] = n3.X;
    a[4, 2] = n3.Y;

    a[5, 3] = 1;
    a[5, 4] = n3.X;
    a[5, 5] = n3.Y;

    a_1 = MatrixHelper.FindingTheInverseMatrix(a);
}

void CreateQ()
{
    q = new float[3, 6];

    q[0, 1] = 1;
    q[1, 5] = 1;
    q[2, 2] = 1;
    q[2, 4] = 1;
}

void CreateD(float elasticModulus, float poissonsRatio)
{
    d = new float[3, 3];
    float koeff = elasticModulus / (1 - poissonsRatio * poissonsRatio);
    d[0, 0] = 1;
    d[0, 1] = poissonsRatio;
    d[0, 2] = 0;
    d[1, 0] = poissonsRatio;
    d[1, 1] = 1;
    d[1, 2] = 0;
    d[2, 0] = 0;
    d[2, 1] = 0;
    d[2, 2] = (1 - poissonsRatio) / 2;
    d = MatrixHelper.MultiplicationMatrixAndNumber(d, koeff);
}

void CreateB()
{
```

```
            b = new float[3, 6];
            b = MatrixHelper.MatrixMultiplication(q, a_1);
        }

        void CreateK(float t)
        {
            k = new float[6, 6];
            float[,] transpB = MatrixHelper.Transpose(b);
            k = MatrixHelper.MatrixMultiplication(transpB, d);
            k = MatrixHelper.MatrixMultiplication(k, b);
            k = MatrixHelper.MultiplicationMatrixAndNumber(k, t);
            float ds = triangleArea(n1.X, n1.Y, n2.X, n2.Y, n3.X, n3.Y);
            k = MatrixHelper.MultiplicationMatrixAndNumber(k, ds);
        }

        public void SolveStress()
        {
            float[,] bb = MatrixHelper.MatrixMultiplication(q, a_1);
            e = MatrixHelper.MatrixMultiplication(bb, sig);
            stress = MatrixHelper.MatrixMultiplication(d, e);
            s = (float)Math.Sqrt(Math.Pow(stress[0, 0], 2) + Math.Pow(stress[1, 0], 2) -
stress[0, 0] * stress[1, 0] + 3 * (Math.Pow(stress[2, 0], 2)));
        }

        float triangleArea(float x1, float y1, float x2, float y2, float x3, float y3)
        {
            float a, b, c, p;
            c = (float)Math.Sqrt(Math.Pow(y1 - y2, 2) + Math.Pow(x1 - x2, 2));
            a = (float)Math.Sqrt(Math.Pow(y2 - y3, 2) + Math.Pow(x2 - x3, 2));
            b = (float)Math.Sqrt(Math.Pow(y1 - y3, 2) + Math.Pow(x1 - x3, 2));

            p = (a + b + c) / 2;
            return ((float)Math.Sqrt(p * (p - a) * (p - b) * (p - c)));
        }

        public SolidBrush ColorElement
        {
            get { return (colorElement); }
            set { colorElement = value; }
        }

        public Node N1
        {
            get { return (n1); }
            set { n1 = value; }
        }

        public Node N2
        {
            get { return (n2); }
            set { n2 = value; }
        }

        public Node N3
        {
            get { return (n3); }
            set { n3 = value; }
        }

        public int Index
        {
            get { return (index); }
            set { index = value; }
        }
```

```
        public float S
        {
            get { return (s); }
            set { s = value; }
        }

        public float[,] A
        {
            get { return (a); }
            set { a = value; }
        }

        public float[,] Q
        {
            get { return (q); }
            set { q = value; }
        }

        public float[,] K
        {
            get { return (k); }
            set { k = value; }
        }

        public float[,] B
        {
            get { return (b); }
            set { b = value; }
        }

        public float[,] D
        {
            get { return (d); }
            set { d = value; }
        }

        public float[,] Sig
        {
            get { return (sig); }
            set { sig = value; }
        }

        public float[,] Stress
        {
            get { return (stress); }
            set { stress = value; }
        }
    }
}
```

MatrixHelper.cs

```
using System.Collections.Generic;

namespace CourseProject
{
    static class MatrixHelper
    {
        //Произведение матриц
        static public float[,] MatrixMultiplication(float[,] a, float[,] b)
        {
```

```
    float[,] c = new float[a.GetLength(0), b.GetLength(1)];
    for (int i = 0; i < a.GetLength(0); i++)
    {
        for (int j = 0; j < b.GetLength(1); j++)
        {
            for (int k = 0; k < b.GetLength(0); k++)
            {
                c[i, j] = c[i, j] + a[i, k] * b[k, j];
            }
        }
    }
    return (c);
}

//Транспонирование матрицы
static public float[,] Transpose(float[,] a)
{
    float[,] b = new float[a.GetLength(1), a.GetLength(0)];
    for (int i = 0; i < a.GetLength(0); i++)
        for (int j = 0; j < a.GetLength(1); j++)
            b[j, i] = a[i, j];
    return (b);
}

//Произвадение матрицы на число
static public float[,] MultiplicationMatrixAndNumber(float[,] a, float b)
{
    for (int i = 0; i < a.GetLength(0); i++)
        for (int j = 0; j < a.GetLength(1); j++)
            a[i, j] = a[i, j] * b;
    return (a);
}

//Нахождение обратной матрицы с помощью метода Гаусса
static public float[,] FindingTheInverseMatrix(float[,] a)
{
    float[,] a_1 = new float[a.GetLength(0), a.GetLength(1)];
    for (int i = 0; i < 6; i++)
        a_1[i, i] = 1;

    float variable;

    for (int i = 0; i < 6; i++)
    {
        if (a[i, i] == 0)
        {
            int index = i + 1;
            while (index < 6)
            {
                if (a[index, i] != 0)
                    break;
                index++;
            }

            if (index != 6)
            {
                for (int j = 0; j < 6; j++)
                {
                    a[i, j] = a[i, j] + a[index, j];
                    a[index, j] = a[i, j] - a[index, j];
                    a[i, j] = a[i, j] - a[index, j];


                    a_1[i, j] = a_1[i, j] + a_1[index, j];
```

```
                    a_1[index, j] = a_1[i, j] - a_1[index, j];
                    a_1[i, j] = a_1[i, j] - a_1[index, j];
                }
            }
        }

        variable = a[i, i];
        for (int j = 0; j < 6; j++)
        {
            a[i, j] = a[i, j] / variable;
            a_1[i, j] = a_1[i, j] / variable;
        }

        for (int j = 0; j < 6; j++)
        {
            if (j != i)
            {
                variable = a[j, i];
                if (variable != 0)
                    for (int k = 0; k < 6; k++)
                    {
                        a[j, k] = a[j, k] - variable * a[i, k];
                        a_1[j, k] = a_1[j, k] - variable * a_1[i, k];
                    }
            }
        }
    }
    return (a_1);
}

//Решение СЛАУ методом Гаусса
static public float[,] Gaus(List<Node> nodes, float[,] globalK)
{
    float variable;
    for (int i = 0; i < nodes.Count * 2; i++)
    {
        if (globalK[i, i] == 0)
        {
            int index = i + 1;
            while (index < nodes.Count * 2)
            {
                if (globalK[index, i] != 0)
                    break;
                index++;
            }

            if (index != nodes.Count * 2)
            {
                for (int j = i; j < nodes.Count * 2 + 1; j++)
                {
                    globalK[i, j] = globalK[i, j] + globalK[index, j];
                    globalK[index, j] = globalK[i, j] - globalK[index, j];
                    globalK[i, j] = globalK[i, j] - globalK[index, j];
                }
            }
        }

        variable = globalK[i, i];
        for (int j = nodes.Count * 2; j >= i; j--)
        {
            globalK[i, j] = globalK[i, j] / variable;
        }
        for (int j = i + 1; j < nodes.Count * 2; j++)
        {
```

```
                    variable = globalK[j, i];
                    if (variable != 0)
                        for (int k = nodes.Count * 2; k >= i; k--)
                            globalK[j, k] = globalK[j, k] - variable * globalK[i, k];
                }

            }

            for (int i = nodes.Count * 2 - 1; i > 0; i--)
            {
                for (int j = i - 1; j >= 0; j--)
                {
                    globalK[j, nodes.Count * 2] -= globalK[j, i] * globalK[i, nodes.Count *
2];
                    globalK[j, i] = 0;
                }
            }

            return (globalK);
        }
    }
}
```

FileHelper.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace CourseProject
{
    static class FileHelper
    {
        //Считывание из файла узлов
        static public List<Node> ReadFileNodes()
        {
            Node node;
            List<Node> nodes = new List<Node>();
            StreamReader file = new StreamReader("4n.txt");
            while (!file.EndOfStream)
            {
                string s = file.ReadLine();
                s = s.TrimStart();
                s = s.TrimEnd();
                List<string> ss = s.Split().ToList<string>();
                for (int i = 0; i < ss.Count; i++)
                    if (ss[i] == "")
                    {
                        ss.RemoveAt(i);
                        i--;
                    }
                if (ss.Count == 5)
                    node = new Node(Convert.ToInt32(ss[0]), Convert.ToSingle(ss[1]), Con-
vert.ToSingle(ss[2]), Convert.ToBoolean(ss[4]));
                else
                    node = new Node(Convert.ToInt32(ss[0]), Convert.ToSingle(ss[1]), Con-
vert.ToSingle(ss[2]), false);
                nodes.Add(node);
            }
            return (nodes);
```

```
        }

        //Считывание из файла элементов
        static public List<Element> ReadFileElements(List<Node> nodes)
        {
            Element element;
            List<Element> elements = new List<Element>();
            StreamReader file = new StreamReader("4e.txt");
            while (!file.EndOfStream)
            {
                string s = file.ReadLine();
                s = s.TrimStart();
                s = s.TrimEnd();
                List<string> ss = s.Split().ToList<string>();
                for (int i = 0; i < ss.Count; i++)
                    if (ss[i] == "")
                    {
                        ss.RemoveAt(i);
                        i--;
                    }
                element = new Element(Convert.ToInt32(ss[0]), nodes[Convert.ToInt32(ss[6]) -
1], nodes[Convert.ToInt32(ss[7]) - 1], nodes[Convert.ToInt32(ss[8]) - 1]);
                elements.Add(element);
            }
            return (elements);
        }
    }
}
```

Form1.cs

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using System.Drawing.Drawing2D;

namespace CourseProject
{
    public partial class Form1 : Form
    {
        float dR = 15, delta = 0.3f, thickness;
        float force1 = 33.3f, force2;
        float dopStress = 1, maxStress;
        float widthDetails= 1500,  lengthDetails = 1500, prevwidthDetails , prevlengthDe-
tails;
        float elasticModulus, poissonsRatio;
        bool edit = false;
        List<Node> nodes;
        List<Element> elements;
        int[] indexNodes1 = new int[] { 77, 78, 79, 80, 81, 82 };
        int[] indexNodes2 = new int[] { 5, 6 };
        float[,] globalK;
        Graphics gr;
        SolidBrush c1, c2, c3, c4, c5, c6, c7, c8, c9;
        public Form1()
        {
            InitializeComponent();
```

```
            gr = panel.CreateGraphics();
            c1 = new SolidBrush(Color.Blue);
            c2 = new SolidBrush(Color.DarkTurquoise);
            c3 = new SolidBrush(Color.LightSkyBlue);
            c4 = new SolidBrush(Color.Aqua);
            c5 = new SolidBrush(Color.LimeGreen);
            c6 = new SolidBrush(Color.GreenYellow);
            c7 = new SolidBrush(Color.Yellow);
            c8 = new SolidBrush(Color.Orange);
            c9 = new SolidBrush(Color.Red);
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            nodes = FileHelper.ReadFileNodes();
            elements = FileHelper.ReadFileElements(nodes);
        }

        private void buttonGrid_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < nodes.Count; i++)
            {
                nodes[i].X = nodes[i].X / dR;
                nodes[i].Y = nodes[i].Y / dR;
            }
           // detailsRadius = Convert.ToSingle(textBoxLength.Text);
            thickness = Convert.ToSingle(textBoxThickness.Text);

            //widthDetails = detailsRadius * 2;
            //lengthDetails = widthDetails * 2;

            //dR = lengthDetails / defaultR;
            //dR = detailsRadius / defaultR;
           // dR = 10000;
            for (int i = 0; i < nodes.Count; i++)
            {
                nodes[i].X = nodes[i].X * dR;
                nodes[i].Y = nodes[i].Y * dR;
            }

            if (edit)
                gr.TranslateTransform(-(panel.Width - prevlengthDetails * delta) / 2, -(440
- (panel.Height - prevwidthDetails * delta) / 2));
            edit = true;
            gr.TranslateTransform((panel.Width - lengthDetails * delta) / 2, (440 - (pan-
el.Height - widthDetails * delta) / 2));
            prevlengthDetails = lengthDetails;
            prevwidthDetails = widthDetails;

            gr.Clear(Color.Linen);
            Meshing(Color.Green);
            DrawingFixation();
            DrawingForce();
            groupBoxResult.Visible = false;
            comboBox.Visible = false;
            buttonSolve.Enabled = true;
```

```
        }

        //Метод отрисовки сетки
        void Meshing(Color color)
        {
            Pen p = new Pen(color, 1);
            for (int i = 0; i < elements.Count; i++)
            {
                gr.DrawLine(p, elements[i].N1.X * delta, -elements[i].N1.Y * delta, ele-
ments[i].N2.X * delta, -elements[i].N2.Y * delta);
                gr.DrawLine(p, elements[i].N2.X * delta, -elements[i].N2.Y * delta, ele-
ments[i].N3.X * delta, -elements[i].N3.Y * delta);
                gr.DrawLine(p, elements[i].N3.X * delta, -elements[i].N3.Y * delta, ele-
ments[i].N1.X * delta, -elements[i].N1.Y * delta);
            }
        }

        //Метод отрисовки закрепления
        void DrawingFixation()
        {
            Pen p = new Pen(Color.Blue, 3);
            gr.DrawLine(p, nodes[indexNodes1[1]].X * delta - 140, -nodes[indexNodes1[1]].Y *
delta -210, nodes[indexNodes1[4]].X * delta -180, -nodes[indexNodes1[4]].Y * delta-210);
            gr.DrawLine(p, nodes[indexNodes1[1]].X * delta - 140, -nodes[indexNodes1[1]].Y *
delta -210, nodes[indexNodes1[4]].X * delta -193, -nodes[indexNodes1[4]].Y * delta-195);
            gr.DrawLine(p, nodes[indexNodes1[1]].X * delta - 140, -nodes[indexNodes1[1]].Y *
delta +75, nodes[indexNodes1[4]].X * delta -193, -nodes[indexNodes1[4]].Y * delta+90);
            gr.DrawLine(p, nodes[indexNodes1[1]].X * delta - 125, -nodes[indexNodes1[1]].Y *
delta +90, nodes[indexNodes1[4]].X * delta -193, -nodes[indexNodes1[4]].Y * delta+90);
        }

        //Метод отрисовки сил
        void DrawingForce()
        {
            Pen p = new Pen(Color.Red, 4);

            p.EndCap = LineCap.ArrowAnchor;

            for (int i = 0; i < indexNodes1.Length-1; i = i + 1)
                gr.DrawLine(p, nodes[indexNodes1[i]].X * delta -10, -nodes[indexNodes1[i]].Y
* delta -100, nodes[indexNodes1[i]].X * delta -10, -nodes[indexNodes1[i]].Y * delta);
            }

        private void buttonSolve_Click(object sender, EventArgs e)
        {
            gr.Clear(Color.Linen);
            elasticModulus = Convert.ToSingle(textBoxElasticModulus.Text);
            poissonsRatio = Convert.ToSingle(textBoxPoissonsRatio.Text);
            force2 = Convert.ToSingle(textBoxForce.Text);
            force1 = force1 + force2;

            for (int i = 0; i < elements.Count; i++)
                elements[i].CreateMatrix(thickness, elasticModulus, poissonsRatio);
            globalK = new float[nodes.Count * 2, nodes.Count * 2 + 1];

            int[] listIndex = new int[3];
```

```
            for (int i = 0; i < elements.Count; i++)
            {
                listIndex[0] = elements[i].N1.Index;
                listIndex[1] = elements[i].N2.Index;
                listIndex[2] = elements[i].N3.Index;

                for (int j = 0; j < 3; j++)
                    for (int k = 0; k < 3; k++)
                    {
                        globalK[2 * listIndex[j], 2 * listIndex[k]] = globalK[2 * listIn-
dex[j], 2 * listIndex[k]] + elements[i].K[j * 2, k * 2];
                        globalK[2 * listIndex[j] + 1, 2 * listIndex[k]] = globalK[2 *
listIndex[j] + 1, 2 * listIndex[k]] + elements[i].K[j * 2 + 1, k * 2];
                        globalK[2 * listIndex[j], 2 * listIndex[k] + 1] = globalK[2 *
listIndex[j], 2 * listIndex[k] + 1] + elements[i].K[j * 2, k * 2 + 1];
                        globalK[2 * listIndex[j] + 1, 2 * listIndex[k] + 1] = globalK[2 *
listIndex[j] + 1, 2 * listIndex[k] + 1] + elements[i].K[j * 2 + 1, k * 2 + 1];
                    }
            }

            for (int i = 0; i < indexNodes1.Length; i++)
                globalK[indexNodes1[i] * 2, nodes.Count * 2] = force1;
            for (int i = 0; i < indexNodes2.Length; i++)
                globalK[indexNodes2[i] * 2 + 1, nodes.Count * 2] = force2;

            for (int i = 0; i < nodes.Count; i++)
            {
                if (nodes[i].Fixation)
                {
                    for (int j = 0; j < nodes.Count * 2; j++)
                    {
                        globalK[2 * i, j] = 0;
                        globalK[2 * i + 1, j] = 0;
                        globalK[j, 2 * i] = 0;
                        globalK[j, 2 * i + 1] = 0;
                    }
                    globalK[2 * i, 2 * i] = 1;
                    globalK[2 * i + 1, 2 * i + 1] = 1;
                }
            }

            globalK = MatrixHelper.Gaus(nodes, globalK);

            for (int i = 0; i < elements.Count; i++)
            {
                elements[i].Sig[0, 0] = globalK[2 * elements[i].N1.Index, 2 * nodes.Count];
                elements[i].Sig[1, 0] = globalK[2 * elements[i].N1.Index + 1, 2 *
nodes.Count];
                elements[i].Sig[2, 0] = globalK[2 * elements[i].N2.Index, 2 * nodes.Count];
                elements[i].Sig[3, 0] = globalK[2 * elements[i].N2.Index + 1, 2 *
nodes.Count];
                elements[i].Sig[4, 0] = globalK[2 * elements[i].N3.Index, 2 * nodes.Count];
                elements[i].Sig[5, 0] = globalK[2 * elements[i].N3.Index + 1, 2 *
nodes.Count];
            }
```

```
            for (int i = 0; i < elements.Count; i++)
                elements[i].SolveStress();

            MaxDeformationXAndY();
            if (maxStress > dopStress)
                labelStresOk.Text = "нет";
            else
                labelStresOk.Text = "да";
        }

        //Определение максимальных смещений и напряжения
        void MaxDeformationXAndY()
        {
            float maxX = globalK[0, 2 * nodes.Count];
            float maxY = globalK[1, 2 * nodes.Count];

            for (int i = 0; i < nodes.Count; i++)
            {
                if (globalK[i * 2, 2 * nodes.Count] > maxX)
                {
                    maxX = globalK[i * 2, 2 * nodes.Count];
                }
                if (globalK[i * 2 + 1, 2 * nodes.Count] > maxY)
                {
                    maxY = globalK[i * 2 + 1, 2 * nodes.Count];
                }
            }

            maxStress = elements[0].S;
            for (int i = 0; i < elements.Count; i++)
                if (elements[i].S > maxStress)
                    maxStress = elements[i].S;

            labelMaxDeformationX.Text = maxX.ToString();
            labelMaxDeformationY.Text = maxY.ToString();
            labelStress.Text = maxStress.ToString();
            groupBoxResult.Visible = true;
            comboBox.Visible = true;
            gr.Clear(Color.Linen);
            Meshing(Color.Green);
        }

        //Отрисовка цветов
        private void DrawingDeformation()
        {
            for (int i = 0; i < elements.Count; i++)
            {
                Point[] points = new Point[3];
                points[0] = new Point(Convert.ToInt32((elements[i].N1.X + elements[i].Sig[0,
0] * 400f) * delta), -Convert.ToInt32((elements[i].N1.Y + elements[i].Sig[1, 0] * 400f) *
delta));
                points[1] = new Point(Convert.ToInt32((elements[i].N2.X + elements[i].Sig[2,
0] * 400f) * delta), -Convert.ToInt32((elements[i].N2.Y + elements[i].Sig[3, 0] * 400f) *
delta));
```

```
                points[2] = new Point(Convert.ToInt32((elements[i].N3.X + elements[i].Sig[4,
0] * 400f) * delta), -Convert.ToInt32((elements[i].N3.Y + elements[i].Sig[5, 0] * 400f) *
delta));
                gr.FillPolygon(elements[i].ColorElement, points);

                Pen p = new Pen(Color.Black, 1);
                gr.DrawLine(p, (elements[i].N1.X + elements[i].Sig[0, 0] * 400f) * delta, -
(elements[i].N1.Y + elements[i].Sig[1, 0] * 400f) * delta, (elements[i].N2.X + ele-
ments[i].Sig[2, 0] * 400f) * delta, -(elements[i].N2.Y + elements[i].Sig[3, 0] * 400f) *
delta);
                gr.DrawLine(p, (elements[i].N2.X + elements[i].Sig[2, 0] * 400f) * delta, -
(elements[i].N2.Y + elements[i].Sig[3, 0] * 400f) * delta, (elements[i].N3.X + ele-
ments[i].Sig[4, 0] * 400f) * delta, -(elements[i].N3.Y + elements[i].Sig[5, 0] * 400f) *
delta);
                gr.DrawLine(p, (elements[i].N3.X + elements[i].Sig[4, 0] * 400f) * delta, -
(elements[i].N3.Y + elements[i].Sig[5, 0] * 400f) * delta, (elements[i].N1.X + ele-
ments[i].Sig[0, 0] * 400f) * delta, -(elements[i].N1.Y + elements[i].Sig[1, 0] * 400f) *
delta);
            }
        }

        private void comboBox_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (comboBox.SelectedIndex == 0)
                Deformation();
            else if (comboBox.SelectedIndex == 1)
                DeformationX();
            else if (comboBox.SelectedIndex == 2)
                DeformationY();
            else if (comboBox.SelectedIndex == 3)
                Stress();
        }

        //Отрисовка деформированной детали
        void Deformation()
        {
            gr.Clear(Color.Linen);
            Pen p = new Pen(Color.Blue, 1);
            for (int i = 0; i < elements.Count; i++)
            {
                gr.DrawLine(p, (elements[i].N1.X + elements[i].Sig[0, 0] * 400f) * delta, -
(elements[i].N1.Y + elements[i].Sig[1, 0] * 400f) * delta, (elements[i].N2.X + ele-
ments[i].Sig[2, 0] * 400f) * delta, -(elements[i].N2.Y + elements[i].Sig[3, 0] * 400f) *
delta);
                gr.DrawLine(p, (elements[i].N2.X + elements[i].Sig[2, 0] * 400f) * delta, -
(elements[i].N2.Y + elements[i].Sig[3, 0] * 400f) * delta, (elements[i].N3.X + ele-
ments[i].Sig[4, 0] * 400f) * delta, -(elements[i].N3.Y + elements[i].Sig[5, 0] * 400f) *
delta);
                gr.DrawLine(p, (elements[i].N3.X + elements[i].Sig[4, 0] * 400f) * delta, -
(elements[i].N3.Y + elements[i].Sig[5, 0] * 400f) * delta, (elements[i].N1.X + ele-
ments[i].Sig[0, 0] * 400f) * delta, -(elements[i].N1.Y + elements[i].Sig[1, 0] * 400f) *
delta);
            }
        }

        //Отрисовка смещения по оси X
```

```
        void DeformationX()
        {
            gr.Clear(Color.Linen);
            float[] maxmin = new float[elements.Count];
            for (int i = 0; i < elements.Count; i++)
                maxmin[i] = Math.Abs(globalK[2 * elements[i].N1.Index, 2 * nodes.Count]) +
Math.Abs(globalK[2 * elements[i].N2.Index, 2 * nodes.Count]) + Math.Abs(globalK[2 * ele-
ments[i].N3.Index, 2 * nodes.Count]);

            float maxX = maxmin[0];
            float minX = maxmin[0];

            for (int i = 0; i < maxmin.Length; i++)
            {
                if (maxmin[i] > maxX)
                    maxX = maxmin[i];

                if (maxmin[i] < minX)
                    minX = maxmin[i];
            }

            float dX = (maxX - minX) / 9;

            for (int i = 0; i < elements.Count; i++)
            {
                float sumX = Math.Abs(globalK[2 * elements[i].N1.Index, 2 * nodes.Count]) +
Math.Abs(globalK[2 * elements[i].N2.Index, 2 * nodes.Count]) + Math.Abs(globalK[2 * ele-
ments[i].N3.Index, 2 * nodes.Count]);
                if (sumX >= minX && sumX < minX + dX)
                    elements[i].ColorElement = c1;
                else if (sumX >= minX + dX && sumX < minX + 2 * dX)
                    elements[i].ColorElement = c2;
                else if (sumX >= minX + 2 * dX && sumX < minX + 3 * dX)
                    elements[i].ColorElement = c3;
                else if (sumX >= minX + 3 * dX && sumX < minX + 4 * dX)
                    elements[i].ColorElement = c4;
                else if (sumX >= minX + 4 * dX && sumX < minX + 5 * dX)
                    elements[i].ColorElement = c5;
                else if (sumX >= minX + 5 * dX && sumX < minX + 6 * dX)
                    elements[i].ColorElement = c6;
                else if (sumX >= minX + 6 * dX && sumX < minX + 7 * dX)
                    elements[i].ColorElement = c7;
                else if (sumX >= minX + 7 * dX && sumX < minX + 8 * dX)
                    elements[i].ColorElement = c8;
                else if (sumX >= minX + 8 * dX && sumX <= maxX)
                    elements[i].ColorElement = c9;
            }

            DrawingDeformation();
        }

        //Отрисовка смещения по оси Y
        void DeformationY()
        {
            gr.Clear(Color.Linen);
            float[] maxmin = new float[elements.Count];
```

```
            for (int i = 0; i < elements.Count; i++)
                maxmin[i] = Math.Abs(globalK[2 * elements[i].N1.Index + 1, 2 * nodes.Count])
+ Math.Abs(globalK[2 * elements[i].N2.Index + 1, 2 * nodes.Count]) + Math.Abs(globalK[2 *
elements[i].N3.Index + 1, 2 * nodes.Count]);

            float maxY = maxmin[0];
            float minY = maxmin[0];

            for (int i = 0; i < maxmin.Length; i++)
            {
                if (maxmin[i] > maxY)
                    maxY = maxmin[i];

                if (maxmin[i] < minY)
                    minY = maxmin[i];
            }

            float dX = (maxY - minY) / 9;

            for (int i = 0; i < elements.Count; i++)
            {
                float sumX = Math.Abs(globalK[2 * elements[i].N1.Index + 1, 2 *
nodes.Count]) + Math.Abs(globalK[2 * elements[i].N2.Index + 1, 2 * nodes.Count]) +
Math.Abs(globalK[2 * elements[i].N3.Index + 1, 2 * nodes.Count]);
                if (sumX >= minY && sumX < minY + dX)
                    elements[i].ColorElement = c1;
                else if (sumX >= minY + dX && sumX < minY + 2 * dX)
                    elements[i].ColorElement = c2;
                else if (sumX >= minY + 2 * dX && sumX < minY + 3 * dX)
                    elements[i].ColorElement = c3;
                else if (sumX >= minY + 3 * dX && sumX < minY + 4 * dX)
                    elements[i].ColorElement = c4;
                else if (sumX >= minY + 4 * dX && sumX < minY + 5 * dX)
                    elements[i].ColorElement = c5;
                else if (sumX >= minY + 5 * dX && sumX < minY + 6 * dX)
                    elements[i].ColorElement = c6;
                else if (sumX >= minY + 6 * dX && sumX < minY + 7 * dX)
                    elements[i].ColorElement = c7;
                else if (sumX >= minY + 7 * dX && sumX < minY + 8 * dX)
                    elements[i].ColorElement = c8;
                else if (sumX >= minY + 8 * dX && sumX <= maxY)
                    elements[i].ColorElement = c9;
            }

            DrawingDeformation();
        }

        //Отрисовка напряжения
        void Stress()
        {
            gr.Clear(Color.Linen);
            float[] maxmin = new float[elements.Count];
            for (int i = 0; i < elements.Count; i++)
                maxmin[i] = elements[i].S;

            float maxS = maxmin[0];
```

```
            float minS = maxmin[0];

            for (int i = 0; i < maxmin.Length; i++)
            {
                if (maxmin[i] > maxS)
                    maxS = maxmin[i];

                if (maxmin[i] < minS)
                    minS = maxmin[i];
            }

            float dX = (maxS - minS) / 9;

            for (int i = 0; i < elements.Count; i++)
            {
                float sumX = elements[i].S;
                if (sumX >= minS && sumX < minS + dX)
                    elements[i].ColorElement = c1;
                else if (sumX >= minS + dX && sumX < minS + 2 * dX)
                    elements[i].ColorElement = c2;
                else if (sumX >= minS + 2 * dX && sumX < minS + 3 * dX)
                    elements[i].ColorElement = c3;
                else if (sumX >= minS + 3 * dX && sumX < minS + 4 * dX)
                    elements[i].ColorElement = c4;
                else if (sumX >= minS + 4 * dX && sumX < minS + 5 * dX)
                    elements[i].ColorElement = c5;
                else if (sumX >= minS + 5 * dX && sumX < minS + 6 * dX)
                    elements[i].ColorElement = c6;
                else if (sumX >= minS + 6 * dX && sumX < minS + 7 * dX)
                    elements[i].ColorElement = c7;
                else if (sumX >= minS + 7 * dX && sumX < minS + 8 * dX)
                    elements[i].ColorElement = c8;
                else if (sumX >= minS + 8 * dX && sumX <= maxS)
                    elements[i].ColorElement = c9;
            }
            DrawingDeformation();
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

(обязательное)

Результаты моделирования в *ANSYS*

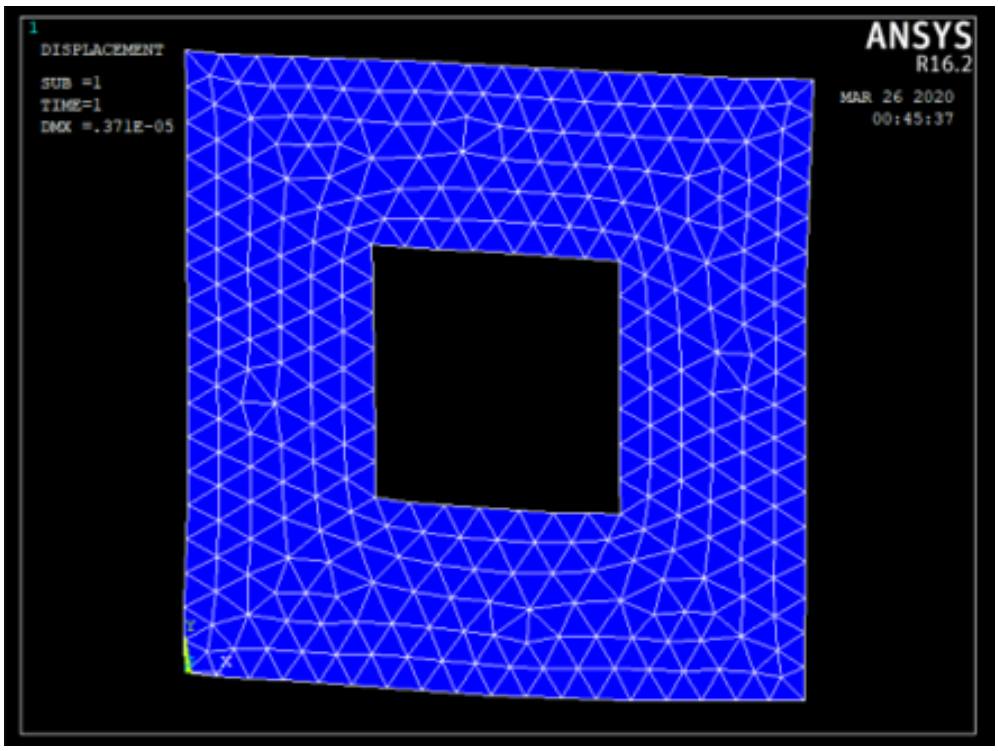

Рисунок Б.1 – Деформация детали

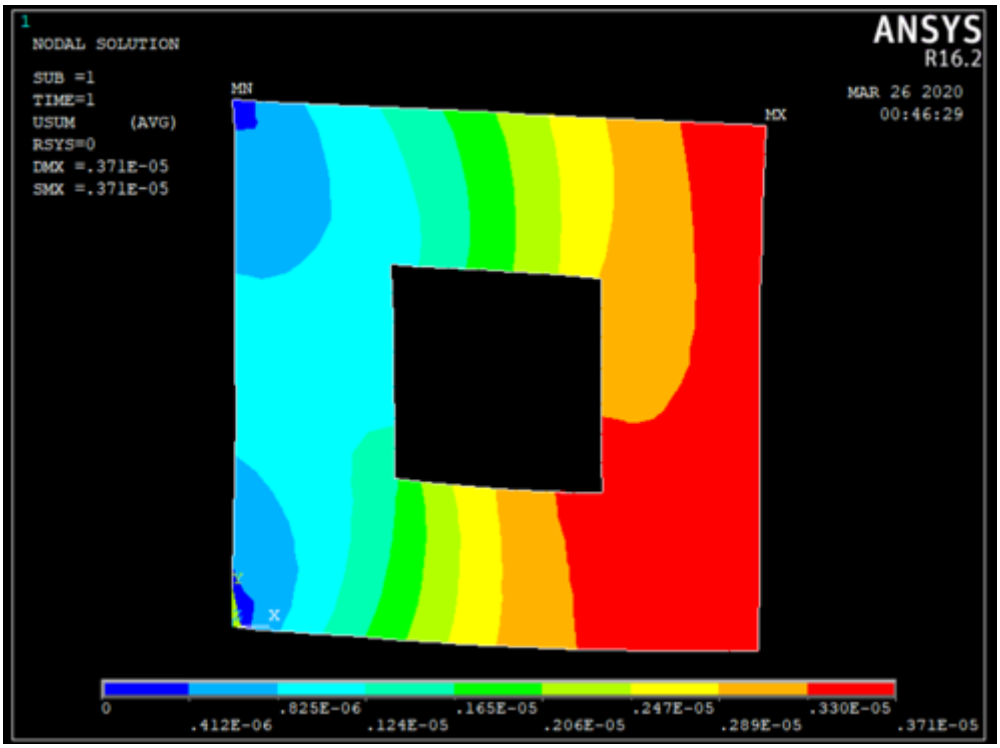

Рисунок Б.2 – Смещение в узлах по OX

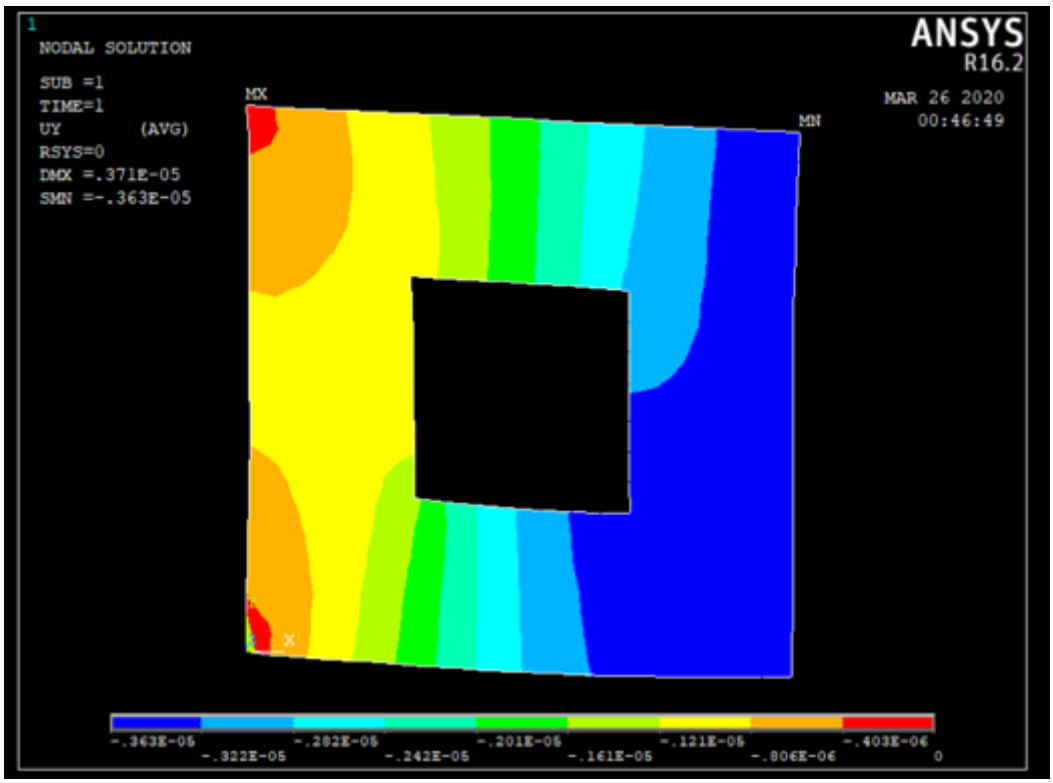

Рисунок Б.3 – Смещение в узлах по OY

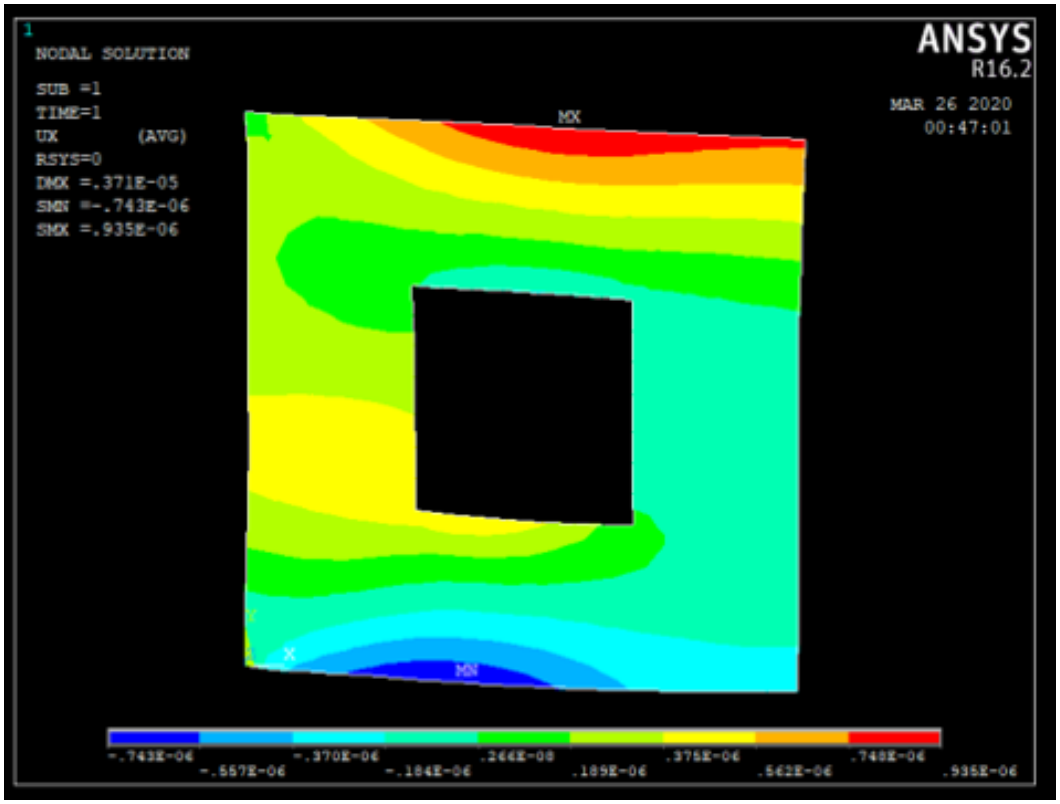

Рисунок Б.4 – Сумма смещений в узлах по OX и OY

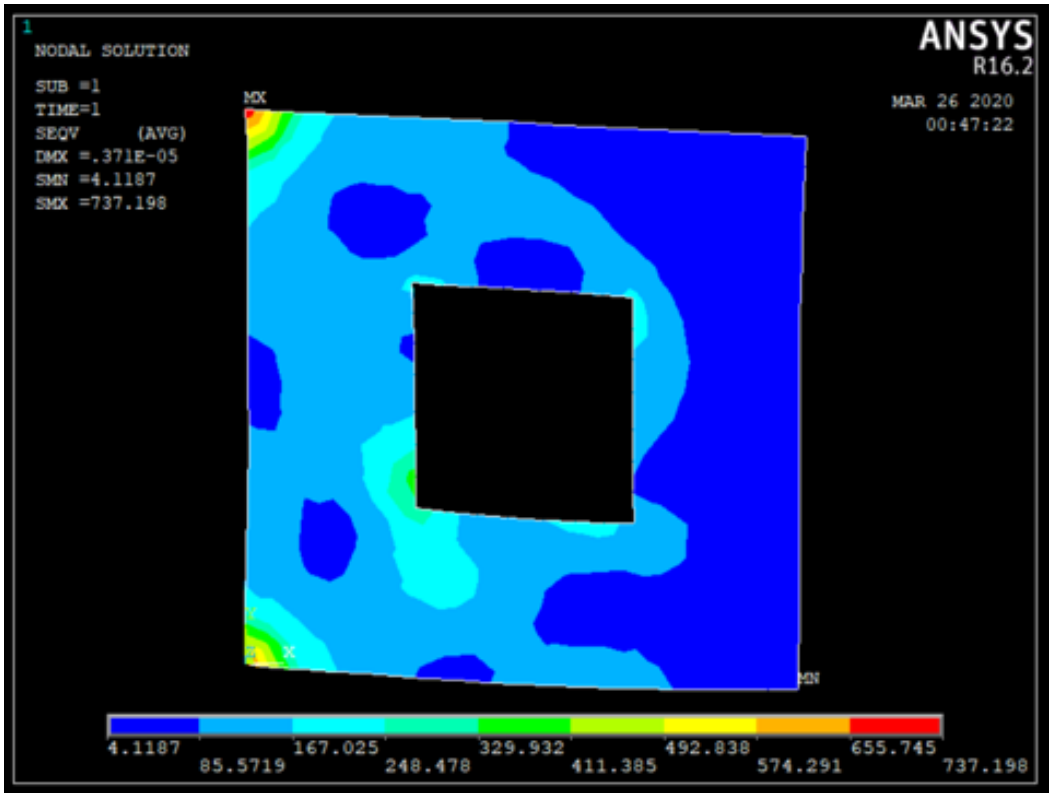

Рисунок Б.5 – Напряженность детали

ПРИЛОЖЕНИЕ В
(обязательное)
Результаты разработанного приложения

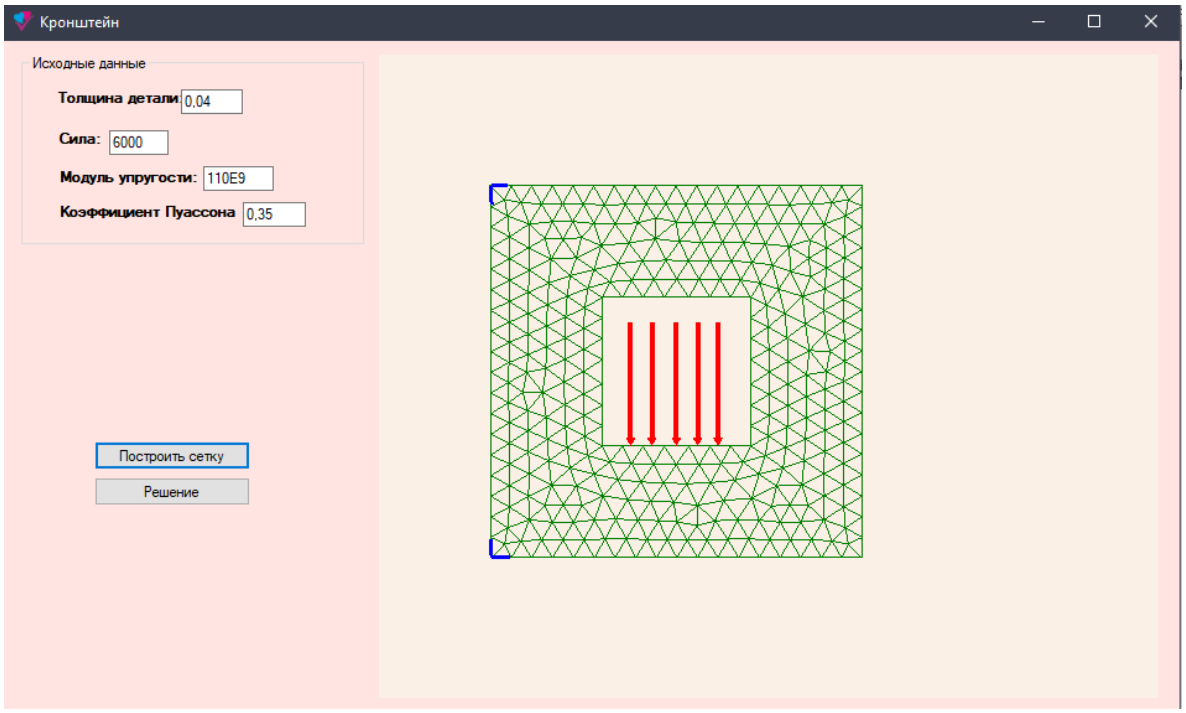

Рисунок В.1 – Внешний вид программы

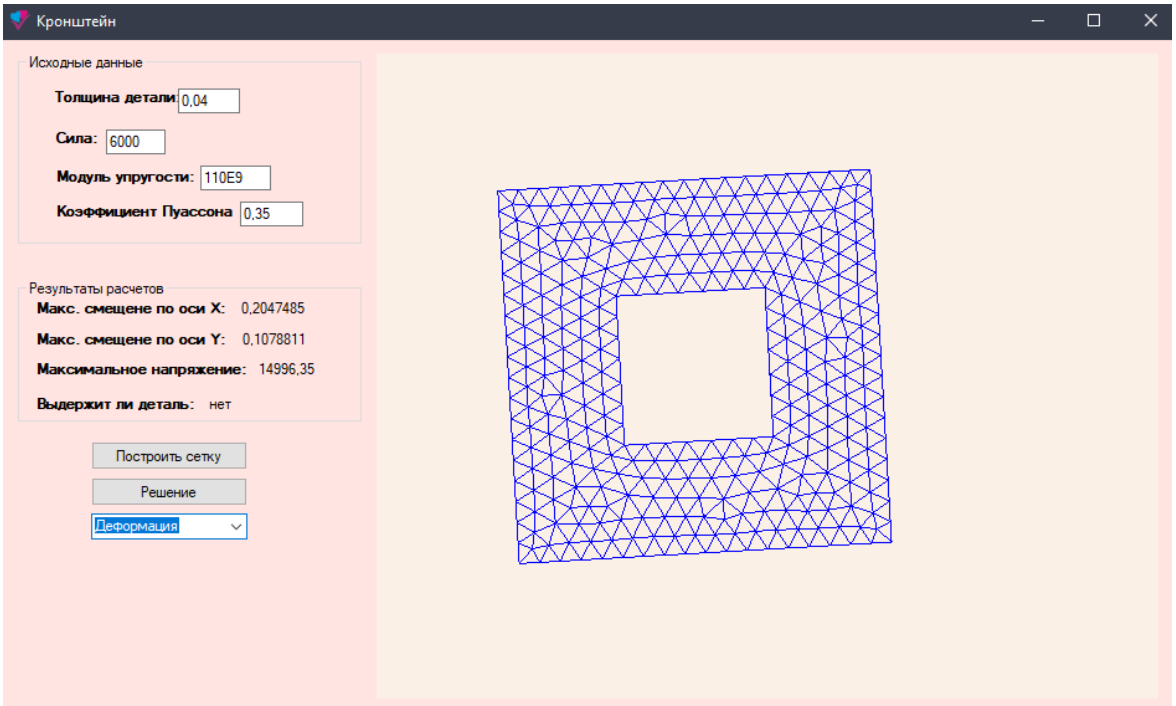

Рисунок В.2 – Деформация детали

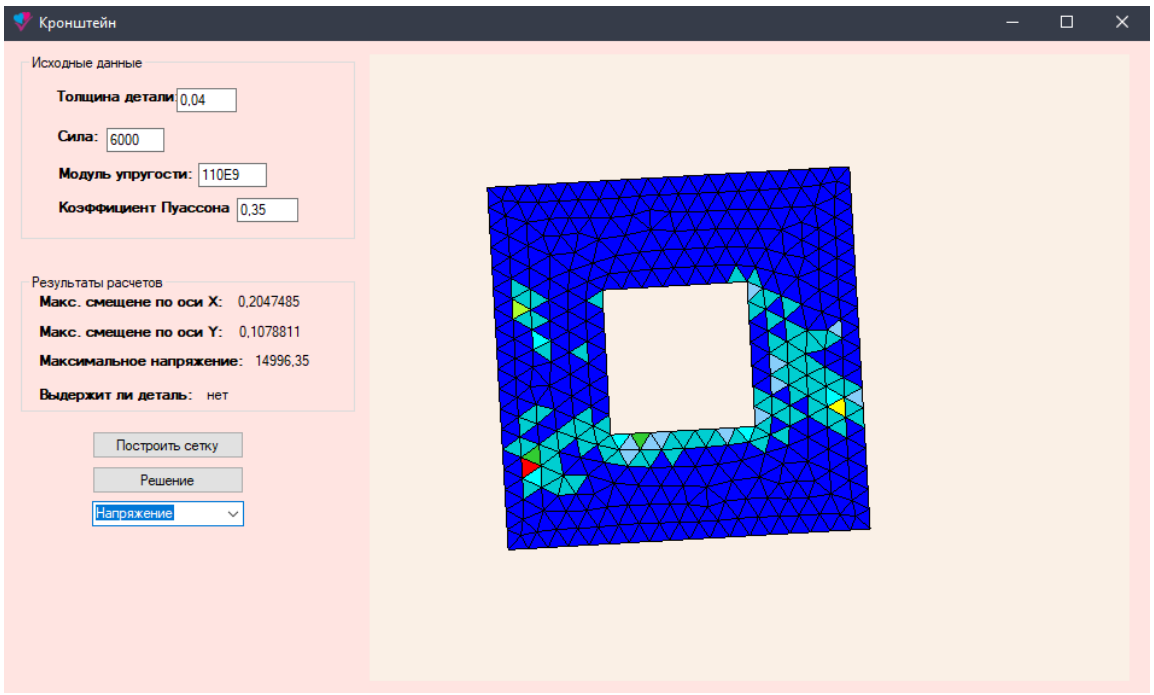

Рисунок В.3 – Напряженность детали

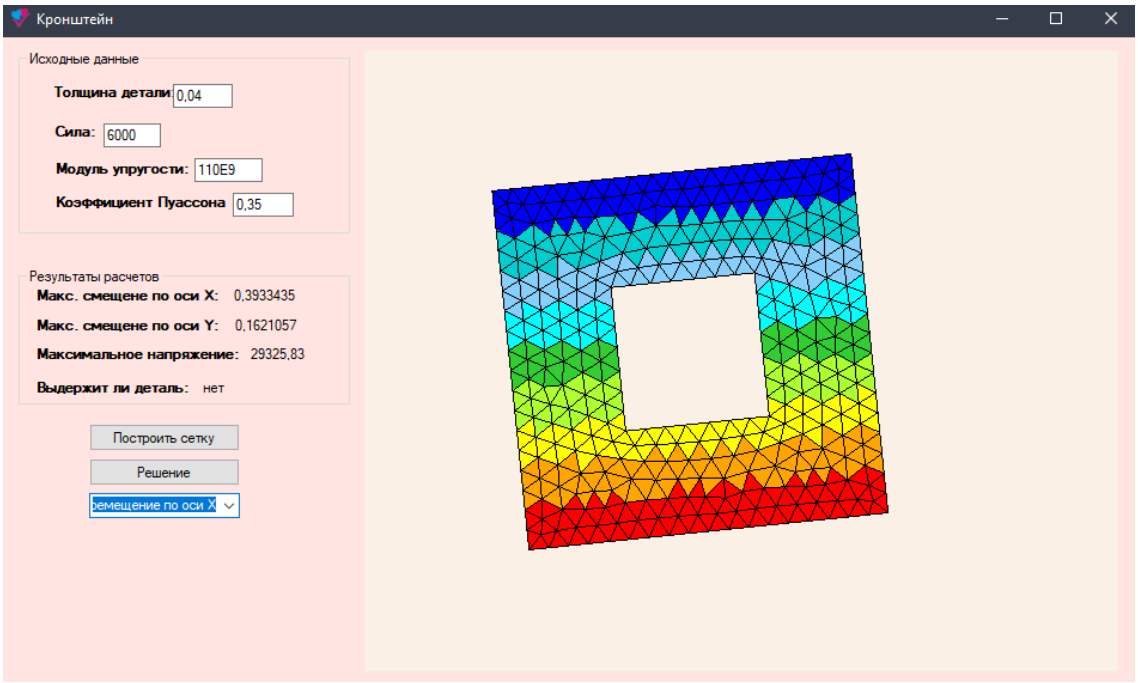

Рисунок В.4 – Смещения в узлах по OX

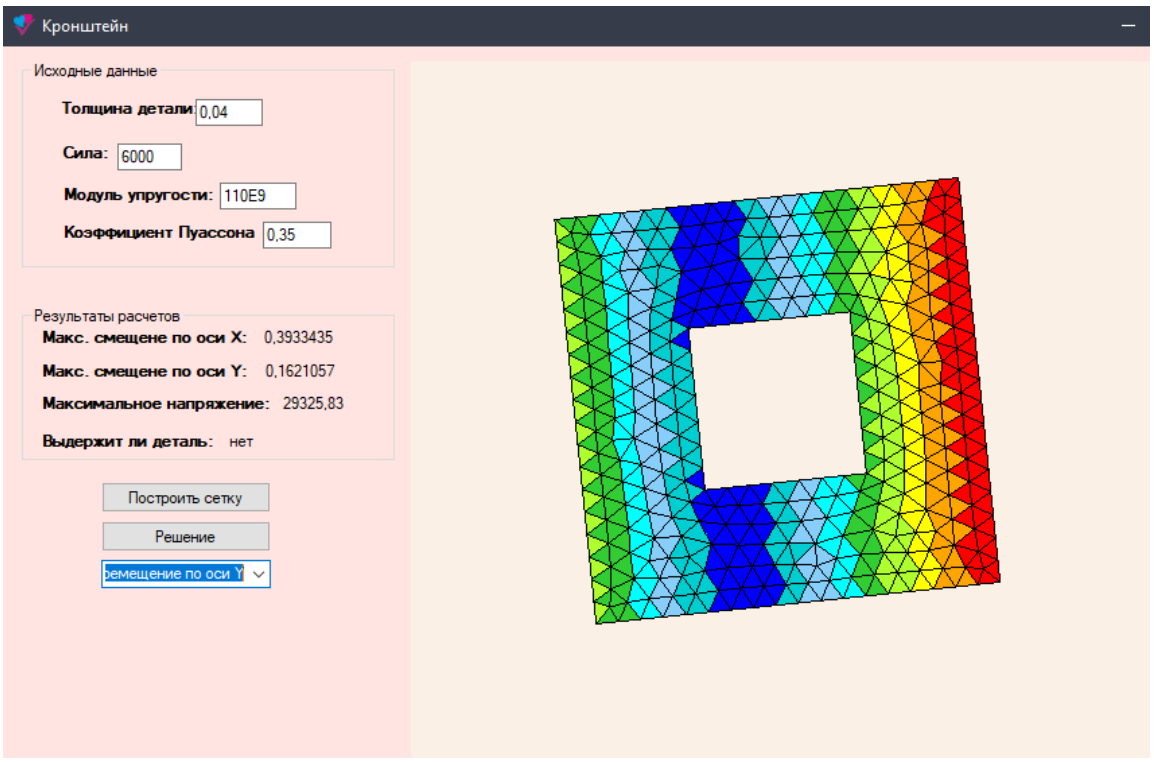

Рисунок В.5 – Смещения в узлах по ОУ

ПРИЛОЖЕНИЕ Г

(обязательное)

Чертеж детали