МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования

«Гомельский государственный технический университет

имени П.О.Сухого»

Факультет автоматизированных и информационных систем

Кафедра «Информационные технологии»

направление специальности 1–40 05 01–01

«Информационные системы и

технологии в проектировании и производстве»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проекту

по дисциплине

«Компьютерные системы конечно-элементных расчетов»

на тему: «Определение размеров равноплечего рычага подвески железнодорожной платформы»

Исполнитель: студент гр. ИТ-31

Миронов Н.А.

Руководитель: ассистент

Левцова Т.С.

Дата проверки \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка работы\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

по защите курсового проекта\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2018

# 

# СОДЕРЖАНИЕ

[СОДЕРЖАНИЕ 3](#_Toc512463228)

[ВВЕДЕНИЕ 4](#_Toc512463229)

[1 ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ МОДЕЛИРОВАНИЯ В МЕХАНИКЕ 5](#_Toc512463230)

[1.1 Численные методы решения задачи 5](#_Toc512463231)

[1.2 Понятие о конечных элементах 7](#_Toc512463232)

[1.3 Общий алгоритм метода конечных элементов 9](#_Toc512463233)

[1.4 Программные средства конечно-элементного анализа 11](#_Toc512463234)

[2 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ ЗАДАЧИ 12](#_Toc512463235)

[2.1 Постановка задачи 12](#_Toc512463236)

[2.2 Описание математической модели 12](#_Toc512463237)

[2.3 Графическая схема алгоритма 17](#_Toc512463238)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ЗАДАЧИ 20](#_Toc512463239)

[3.1 Реализация задачи на языке программирования C# 20](#_Toc512463240)

[3.2 Результаты выполнения разработанного приложения 23](#_Toc512463241)

[4 ВЕРИФИКАЦИЯ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ 26](#_Toc512463242)

[4.1 Моделирование задачи в пакете *Ansys* 26](#_Toc512463243)

[4.2 Получение результатов в пакете *Ansys* 33](#_Toc512463244)

[4.3 Исследование полученных результатов 35](#_Toc512463245)

[ЗАКЛЮЧЕНИЕ 37](#_Toc512463246)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 38](#_Toc512463247)

[ПРИЛОЖЕНИЕ A. Листинг приложения 39](#_Toc512463248)

[ПРИЛОЖЕНИЕ Б. *Ansys* скрипт создания модели детали 52](#_Toc512463249)

[ПРИЛОЖЕНИЕ В. Чертёж детали 54](#_Toc512463251)

# ВВЕДЕНИЕ

На современном этапе развития технологий требуется внедрение сложнейших технических разработок в производство за минимально возможные сроки. Для создания сложных конструкций используют новые материалы и технологии в автоматизации процесса проектирования. Механизмы, как правило, имеют сложную форму деталей и часто обладают большими габаритами. Проверка прочности таких конструкций с помощью полунатурного или натурного эксперимента требует большого количества средств и времени, но она является необходимой частью успешного функционирования конструкции. Использование расчётов на прочность с помощью автоматизированных систем позволяет за меньший промежуток времени исследовать необходимые объекты на прочность.

Чтобы произвести расчёт на прочность детали, необходимо знать геометрию конструкции и нагрузки, действующие на неё. Большое количество технических задач решаются с помощью теорий упругости и пластичности, теории оболочек и пластин. Но эти теории сложны для реализации при сложной геометрии. Задачи со сложной геометрией решаются численными методами, к примеру методом конечных элементов.

Метод конечных элементов является одним из наиболее распространённых методов решения задач математической физики. Это связано с большой универсальностью метода, сочетающего в себе лучшие качества вариационных и разностных методов. К его несомненным достоинствам относятся возможность использования разнообразных сеток, сравнительная простота и единообразие способов построения схем высоких порядков точности в областях сложной формы.

Для автоматизации составления конечно-элементной математической модели необходимо использовать генераторы сеток конечных элементов, которые позволяют автоматически разбить заданную область конструкции на конечные элементы.

Объектом исследования данного проекта является применение метода конечных элементов к описанию математической модели физической системы.

Предметом исследования является применения метода конечных элементов к определению размеров равноплечего рычага подвески железнодорожной платформы.

Таким образом, реализованное в данной работе программное обеспечение найдет свое практическое применение в таких отраслях промышленности как машиностроение и др.

# 1 ОБЗОР СУЩЕСТВУЮЩИХ МЕТОДОВ МОДЕЛИРОВАНИЯ В МЕХАНИКЕ

## **1.1 Численные методы решения задачи**

При выполнении инженерных расчетов, на практике используют как аналитические, так и численные методы. Применение аналитических методов требует высокого уровня математической подготовки инженера. Кроме того, как правило, аналитические расчеты позволяют получить решение задач для тел, имеющих достаточно простую геометрическую форму и схему нагружения.

Однако применение численных методов, к которым относятся методы конечных разностей, конечных элементов, граничных элементов и другие, не ограничено ни сложностью геометрии тела, ни способами приложения нагрузок. Далее в этой главе методы решения будут рассмотрены более подробно.

К наиболее хорошо известным методам относятся метод конечных разностей (МКР) и метод конечных элементов [1, c. 49]. Метод конечных разностей ‒ метод численного решения краевых задач для дифференциальных уравнений называют также методом сеток. Идея метода конечных разностей (метода сеток) известна давно, с соответствующих трудов Эйлера. Однако практическое применение этого метода было тогда весьма ограничено из-за огромного объема ручных вычислений, связанных с размерностью получаемых систем алгебраических уравнений, на решение которых требовались годы. В настоящее время, с появлением быстродействующих компьютеров, ситуация в корне изменилась. Этот метод стал удобен для практического использования и является одним из наиболее эффективных при решении различных задач математической физики.

Основная идея метода конечных разностей (метода сеток) для приближенного численного решения краевой задачи для двумерного дифференциального уравнения в частных производных состоит в том, что:

* на плоскости в области *А*, в которой ищется решение, строится сеточная область *Аs* (рисунок 1.1), состоящая из одинаковых ячеек размером *s* (*s* – шаг сетки) и являющаяся приближением данной области *A*;
* заданное дифференциальное уравнение в частных производных заменяется в узлах сетки *Аs* соответствующим конечно-разностным уравнением;
* с учетом граничных условий устанавливаются значения искомого решения в граничных узлах области *Аs*.

Решая полученную систему конечно-разностных алгебраических уравнений, получим значения искомой функции в узлах сетки *Аs*, т.е. приближенное численное решение краевой задачи. Выбор сеточной области *Аs* зависит от конкретной задачи, но всегда надо стремиться к тому, чтобы контур сеточной области *Аs* наилучшим образом аппроксимировал контур области *А*.

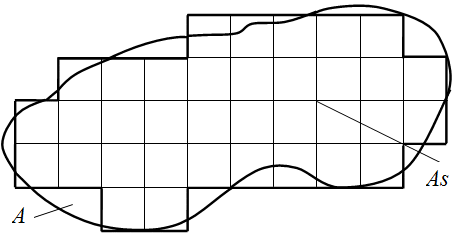


Рисунок 1.1 – Построение сеточной области

Большим преимуществом метода конечных элементов является слабая зависимость от граничных условий задачи, геометрии конструкций и характера исходного напряженного состояния. Недостатком является высокий порядок систем алгебраических уравнений. Для МКР также характерны затруднения при учете смешанных граничных условий, рассмотрении многосвязных областей и стыковок областей, описываемых различными дифференциальными уравнениями.

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. Но диапазон его применения чрезвычайно широк: строительство и машиностроение, гидро- и аэродинамика, горное дело и новейшая техника, а также различные задачи математической физики – теплопроводности, фильтрации, распространения волн и т. д.

Метод конечных элементов, как и многие другие численные методы, основан на представлении реальной континуальной конструкции ее дискретной моделью и замене дифференциальных уравнений, описывающих НДС сплошных тел, системой алгебраических уравнений. Вместе с тем МКЭ допускает ясную геометрическую, конструктивную и физическую интерпретацию. Суть метода заключается в том, что область (одно- , двух- или трехмерная), занимаемая конструкцией, разбивается на некоторое число малых, но конечных по размерам подобластей (рисунок 1.2). Последние носят название конечных элементов (КЭ), а сам процесс разбивки – дискретизацией.

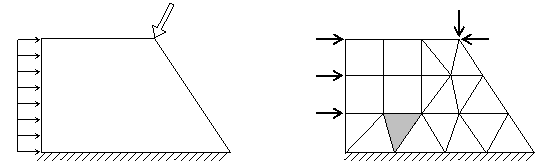


Рисунок 1.2 – Конструкция, разбитая на конечные элементы

МКЭ – это вариационный метод. Функционал энергии для всей рассматриваемой области здесь представляется в виде суммы функционалов отдельных ее частей – конечных элементов. По области каждого элемента, независимо от других, задается свой закон распределения искомых функций. Такая кусочно-непрерывная аппроксимация выполняется с помощью специально подобранных аппроксимирующих функций, называемых также координатными или интерполирующими. С их помощью искомые непрерывные величины (перемещения, напряжения и т.д.) в пределах каждого КЭ выражаются через значения этих величин в узловых точках, а произвольная заданная нагрузка заменяется системой эквивалентных узловых сил.

Метод конечных элементов позволяет практически полностью автоматизировать расчет стержневых систем, хотя, как правило, требует выполнения значительно большего числа вычислительных операций по сравнению с классическими методами строительной механики. Однако в современных условиях большой объем вычислений не является серьезной проблемой, и, в связи с этим, при внедрении ЭВМ в инженерную практику МКЭ получил широчайшее распространение. Поэтому, знание основ метода конечных элементов и современных программных средств, позволяющих на его основе решать разнообразные задачи, в наше время для инженера является абсолютно необходимым.

### 1.2 Понятие о конечных элементах

Суть МКЭ – это разбиение некоторой конструкции на непересекающиеся компоненты (подобласти), простые геометрические объекты, называемые конечными элементами. Множество элементов, на которые разбита конструкция, называется конечно-элементной сеткой.

Физическая реакция любого элемента выражается с помощью конечного числа степеней свободы или значений искомых функций во множестве узловых точек. Поведение математической модели, таким образом, аппроксимируется поведением дискретной модели, полученной путем сборки или объединения всех элементов. Способ разбиения-сборки естественно возникает при исследовании многих искусственных или живых систем. Отличие метода от метода конечных разностей состоит в том, что конечные элементы не перекрывают друг друга.

Основные типы конечных элементов задач механики представлены на рисунке 1.3.

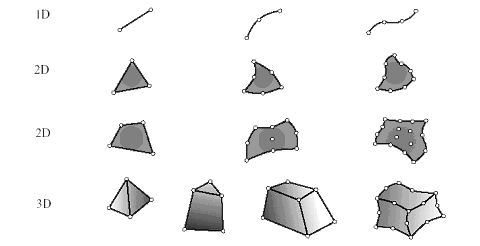


Рисунок 1.3 ­­− Основные типы конечных элементов для одно-, дву- и трехмерных задач механики

Основные свойства конечных элементов:

* собственная размерность (конечные элементы могут описываться одной, двумя или тремя пространственными координатами в зависимости от размерности задачи, для решения которой они предназначены).
* Узловые точки. Каждый элемент описывается множеством характерных точек, называемых узловыми точками или узлами для краткости.
* Геометрия элемента. Геометрия элемента определяется расположением узловых точек. Большинство элементов, используемых в расчетах, имеют достаточно простую геометрическую форму.
* Степени свободы. Степени свободы определяют физическое состояние элемента, т.е. физическое поле, которое описывает данный элемент. Благодаря общим степеням свободы в соседних элементах осуществляется сборка модели и формирование глобальной системы конечно-элементных уравнений; в качестве степеней свободы могут фигурировать как узловые значения неизвестной функции, так и ее производные по пространственным координатам в узлах.
* Узловые силы. Система узловых сил полностью соответствует степеням свободы элемента и выражается с помощью глобального вектора узловых сил.
* 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 тензор деформаций и тензор напряжений в точке. Для линейного упругого стержневого элемента достаточно задать один модуль Юнга Е и один коэффициент температурного расширения α.
* Свойства сечения. Свойствам сечения относятся площади и моменты инерции одномерных и двумерных конечных элементов, таких как балки, стержни, пластины.

### 1.3 Общий алгоритм метода конечных элементов

Обобщенный расчет МКЭ состоит из последовательности шагов (операций с матрицами), в результате выполнения которых определяются необходимые данные, для решения задачи (перемещения, деформации, напряжения).

Для решения задачи методом конечных элементов необходимо выполнить следующие шаги:

Дискретизация конечных элементов (конструкция) разбивается на конечные элементы. Конечные элементы могут иметь различную форму и различные размеры. В результате разбивки создаётся сетка из границ элементов. Пересечения этих границ образуют узлы. На границах и внутри элементов могут быть созданы дополнительные узловые точки. Ансамбль из всех конечных элементов и узлов является основой конечно-элементной модели деформируемого тела. Дискретная модель должна достаточно хорошо покрывать область исследуемого объекта.

Пример конечно-элементной модели представлен на рисунке 1.4.

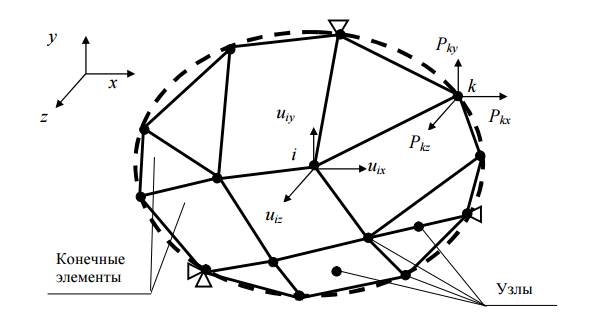


Рисунок 1.4 − Конечно-элементная модель

Аппроксимация искомой функции с помощью функций формы (искомая функция – поле перемещений точек деформированного тела аппроксимируется с помощью множества кусочно-непрерывных функций, называемых функциями формы);

Реализация вариационного принципа (на этом этапе осуществляется вычисление матриц жесткостей элементов и построение глобальной матрицы системы алгебраических уравнений и вектора узловых сил. Глобальная матрица жесткости может быть получена несколькими методами):

* методом непосредственного сложения жесткостей.
* Методом конгруэнтного преобразования.
* При помощи конечно-разностных операторов.
* Задание граничных условий (силовые граничные условия учитываются общими векторами узловых сил). Граничные условия в перемещениях (связи) могут учитываться как при формировании матриц элементов, так и после сборки общих матриц модели).
* Решение общей системы уравнений равновесия конечно-элементной модели (общая система уравнений равновесия), полученная МКЭ для линейно упругой модели тела, является с математической точки зрения системой линейных алгебраических уравнений (СЛАУ). После учета правильно наложенных связей, не допускающих движения модели как абсолютно твёрдого тела, определитель матрицы [*K*] не равен нулю и, следовательно, существует единственное решение – общий вектор узловых перемещений {*U*}).
* Анализ результатов решения (после определения общего вектора узловых перемещений {*U*} находят элементные векторы узловых перемещений {}. Через них путём интерполяции с помощью функций формы вычисляются перемещения любых точек элементов) [3].

### 1.4 Программные средства конечно-элементного анализа

Совершенствование МКЭ обусловлено взаимосвязью трех факторов: наличием высокопроизводительной вычислительной техники; разработкой математических моделей исследуемых явлений, имеющих достаточный уровень соответствия реальным моделям.

Для МКЭ характерны особенности, которые следует учитывать при выборе и разработке программы расчета. Такими особенностями являются большие объемы исходных данных, промежуточных и окончательных результатов расчета. Поэтому расчет по МКЭ состоит из трех основных этапов: разработка расчетной конечно-элементной схемы и подготовка исходных данных; проверка самого расчета; обработка результатов расчета.

В настоящее время существует достаточно систем для решения задач методом конечных элементов, среди них: *ANSYS*, *MSC.Marc*, *FEMLAB*, *COSMOS/M*, *PzFLEX*, *ATILA* и т.д. В курсовом проекте будет использоваться система *ANSYS*.

*ANSYS* – универсальная программный продукт (МКЭ) анализа, созданный более 30 лет, и продолжающий развиваться.

*ANSYS* является одним из самых распространенных комплексов сегодня, использующим метод конечных элементов. Многоцелевая направленность программы, независимость от аппаратных средств (от персональных компьютеров до рабочих станций и суперкомпьютеров), средства геометрического моделирования на базе *B*-сплайнов, полная совместимость с *CAD/CAM/CAE* системами ведущих производителей и «дружеский» интерфейс привели к тому, что именно *ANSYS* в настоящее время используется во многих университетах для обучения студентов и выполнения научно-исследовательских работ.

Возникновение необходимости разработки приложения, а не использование математического пакета *ANSYS* заключалась в том, что поставленная задача требовала определённые условия применения и поддержку.

*ANSYS Mechanical* предоставляет пользователю возможность проведения различных расчетов в рамках механики сплошной среды.

### 2 АЛГОРИТМИЧЕСКИЙ АНАЛИЗ ЗАДАЧИ

### 2.1 Постановка задачи

Разработать приложение для определения размеров равноплечего рычага подвески железнодорожной платформы. Результаты решения необходимо верифицировать в системе *Ansys*.

Разработанное *Windows*-приложение,должновыполнять следующие действия:

* разбивать плоскую конструкцию на конечные элементы;
* графически изображать конструкцию до и после приложения нагрузки;
* определять, выдержит ли конструкция заданную нагрузку.

### 2.2 Описание математической модели

Поверхность разбивается на конечные элементы треугольного типа, и нумерацией узлов по часовой стрелке (рисунок 2.1).

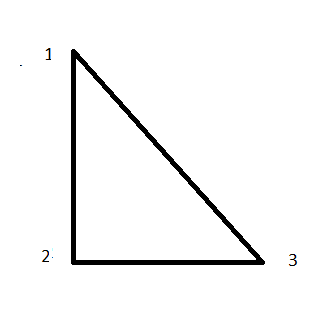


Рисунок 2.1 – Конечный элемент

Данный элемент, имеет три узла, пронумерованных по часовой стрелке. Каждый узел имеет две компоненты перемещения: по *х*, по *у*, формула 2.1.

. (2.1)

Компоненты перемещений узлов элемента образуют вектор перемещений {*δ*}, формула 2.2:

. (2.2)

Перемещения внутри элемента (рисунок 2.2) должны однозначно определяться этими шестью величинами.

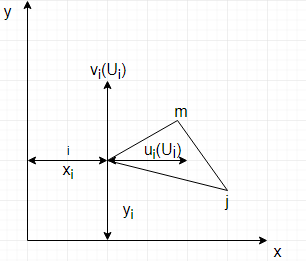


Рисунок 2.2 – Перемещения узлов элемента

Простейшим представлением являются линейные полиномы, формула 2.3 и формула 2.4:

, (2.3)

. (2.4)

Значения шести постоянных i легко найти из двух систем, состоящих из трех уравнений, которые получаются в результате подстановки в формулу 2.3узловых координат и приравнивания значения перемещений соответствующим перемещениям узловых точек.

Записав, формулы 2.5 – 2.7:

, (2.5)

, (2.6)

. (2.7)

Выражают 1, 2, 3 через величины узловых перемещений *ui*, *uj*, *um* и окончательный вид, формулы 2.8 – 2.11:

, (2.8)

, (2.9)

, (2.10)

. (2.11)

Остальные коэффициенты получаются циклической перестановкой индексов *i*, *j*, *m*, а величина 2 определяется соотношением, формула 2.12:

. (2.12)

Аналогично можно представить перемещение *v* в вертикальном направлении, формула 2.13:

.(2.13)

Соотношения 2.5 – 2.11 в стандартной форме определяют перемещения любой точки внутри элемента, формула 2.14:

, (2.14)

где *I* – единичная матрица размерности 2*x*2, *N*i – координатные функции, которые называются функциями формы, формула 2.15:

. (2.15)

Для каждого из элементов плоскости существуют матрицы, определяющие его поведение.

Матрица [*E*] – матрица механических характеристик.

Ее вид, формула 2.16:

, (2.16)

где *G* – модуль упругости материала, а *µ*– коэффициент Пуассона.

Матрица [*Q*] – матрица дифференциальных операторов.

Ее вид, формула 2.17:

. (2.17)

Матрица [*A*] – матрица неизвестных при коэффициентах.

Ее вид, формула 2.18:

, (2.18)

где *x*1, *y*1 – координаты первого узла элемента;*x*2, *y*2 – координаты второго узла элемента;*x*3, *y*3 – координаты третьего узла элемента.

Зная эти матрицы для элемента *i*, для него составляется матрица жесткости по формуле 2.19:

, (2.19)

где *dx*, *dy* – стороны треугольника.

Далее, зная для каждого элемента локальную матрицу жесткости, находится глобальная матрица жесткости по формуле 2.20:

. (2.20)

Определяется вектор нагрузки *R*. Для плоской задачи для *i*-ого узла нагрузку можно задать в двух направлениях: по x и по y, формула 2.21:

, (2.21)

где *Rxi* – компонента вектора нагрузок для *i*-ого узла, определяющая давление, прикладываемое к узлу по оси абсцисс.

*Ryi* – компонента вектора нагрузок для *i*-ого узла, определяющая давление прикладываемое к узлу по оси ординат.

Далее необходимо задать граничные условия.

Для закрепленных узлов в матрице [*K*] необходимо сбросить значения в строках и столбцах, а на главной диагонали для них приравнять к 1.

Далее необходимо составить глобальный вектор нагрузок, который будет состоять из компонент нагрузок для каждого узла, формула 2.22:

. (2.22)

Далее необходимо составить систему уравнений, формула 2.23:

. (2.23)

Или, если переписать в матричной форме, формула 2.24:

, (2.24)

где [*К*] – глобальная матрица жесткости. В данной системе элементы глобальной матрицы жесткости являются коэффициентами при неизвестных.

{*u*} – вектор столбец искомых значений.

{*R*} – глобальный вектор столбец нагрузок. В данной системе элементы вектора выступают в роли свободных членов уравнения.

Решая данную систему, получают смещения узлов. Вектор {*u*} будет иметь вид, формула 2.25:

, (2.25)

где *u*i, *vi*– смещения узла относительно первоначального положения.

Полученная системы в данном проекте решается методом Гаусса. Алгоритм решения методом Гаусса подразделяется на два этапа[6]:

* прямой ход путём элементарных преобразований над строками систему приводят к ступенчатой или треугольной форме, либо устанавливают, что система несовместна. Выбирается разрешающая строка *k*-ая , где *k* = 0…*n* - 1, и для каждой следующей строки выполняется преобразование элементов.
* Обратный ход, осуществляется определение значений неизвестных. Из последнего уравнения преобразованной системы вычисляется значение переменной *х*n, после этого из предпоследнего уравнения становится возможным определение переменной *xn*-1 и так далее.

### 2.3 Графическая схема алгоритма

Краткая графическая схема алгоритма представлена на рисунке 2.3.

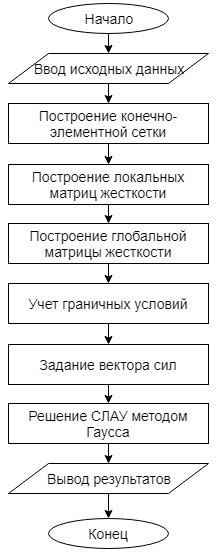


Рисунок 2.3 – Графическая схема алгоритма

Первый этап – геометрическое моделирование включает создание геометрии модели конструкции, пригодной для МКЭ, с учетом всех параметров, которые могут оказать существенное влияние на результаты расчетов. На этой стадии помимо ввода геометрических параметров конструкции задаются физические свойства материалов, из которых она изготовлена.

На этапе создания сетки конечных элементов выполняются мероприятия по созданию максимально возможного количества областей с регулярной сеткой конечных элементов. В местах, где предполагаются большие градиенты напряжений, необходима более мелкая сетка. Форма конечных элементов также влияет на точность вычислений. Следует избегать слишком вытянутых элементов, так как элементы с примерно одинаковыми размерами сторон дают меньшую ошибку.

На этапе моделирования граничных условий учитывают как действие активных сил, так и наложенных на систему связей. Приложение силовых факторов должно учитывать особенности реальной работы конструкции при рассматриваемых режимах эксплуатации. Количество связей должно быть достаточным, чтобы обеспечить построение кинематический неизменяемой модели.

Граничные условия (перемещения или силы) прикладываются только к узлам. Необходимо обратить особое внимание на то, что число граничных условий должно быть минимально необходимым, чтобы отразить реальную работу конструкции.

Численное решение системы уравнений равновесия выполняется, как правило, автоматически с использованием ЭВМ.

На пятом этапе проводят анализ полученных результатов путем получения полей законов распределения напряжений и деформаций, а также построения необходимых графических зависимостей либо табличных форм вывода результатов.

Несмотря на высокий уровень развития программного обеспечения, реализующего МКЭ, результаты расчетов могут оказаться неверными. Использование вычислительной техники в роли «черного ящика», без понимания основных процессов и этапов вычислений, может привести к существенным ошибкам.

### 3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ ЗАДАЧИ

### 3.1 Реализация задачи на языке программирования C#

Структура программного комплекса включает пять пользовательских классов: *Form1*, *InputDate*, *Node*, *Element*, *Matrix*. Их элементы описаны в таблицах 3.1 – 3.5.

Класс *Form1* предназначен для вызова основных методов а также отображения графического интерфейса программы.

В таблице 3.1 представлена структура класса *Form1.*

Таблица 3.1 – Структура класса *Form1*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя переменной | Вид элемента | Тип | Спецификатор | Описание |
| *Thickness* | поле | *float* | *Public* | Толщина детали |
| *Force* | поле | *float* | *Public* | Сила |
| *elasticModulus* | поле | *float* | *Public* | Модуль  упругости |
| *poissonsRatio* | поле | *float* | *Public* | Коэффициент Пуассона |
| *MainForm* | конструктор | – | *Public* | Конструктор без параметров |
| *buttonDetails\_Click()* | метод | *Void* | *Private* | Отображение детали |
| *buttonGrid\_Click()* | метод | *Void* | *Private* | Построение сетки |
| *buttonSolve\_Click()* | метод | *Void* | *Private* | Решение |
| *MaxXY()* | метод | *Void* | *Private* | Определение максимальных смещений и напряжения |

Класс *InputDate* предназначен для считывания данных о узлах с текстого файла.

В таблице 3.2 представлена структура класса *InputDate.*

Таблица 3.2 – Структура класса *InputDate*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя переменной | Вид элемента | Тип | Спецификатор | Описание |
| *Nodes*() | метод | *List<Node>* | *static public* | Получение из файла узлов |
| *Elements(List<Node>nodes)* | метод | *List<Element>* | *static public* | Получение из файла элементов |

Класс *Node* предназначен для хранения полученных данных из файла и представления их в виде объекта.

В таблице 3.3 представлена структура класса *Node.*

Таблица 3.3 – Структура класса *Node*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя переменной | Вид элемента | Тип | Спецификатор | Описание |
| *Node*() | Конструктор | – | *Public* | Конструктор без параметров |
| *Node (int \_index, double \_x, double \_y, bool \_fixation)* | Конструктор | – | *Public* | Конструктор с параметров |
| *X* | Поле | *double* | *Public* | Координата по оси *х* |
| *Y* | Поле | *double* | *Public* | Координата по оси *у* |
| *Index* | Поле | *Int* | *Public* | Номер узла |
| *Fixing* | Поле | *bool* | *Public* | Закрепление |

Класс *Elements* предназначен для хранения полученных данных из файла и представления их в виде объекта.

В таблице 3.4 представлена структура класса *Element.*

Таблица 3.4 – Структура класса *Element*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя переменной | Вид элемента | Тип | Спецификатор | Описание |
| 1 | 2 | 3 | 4 | 5 |
| *Element()* | Конструктор | – | *Public* | Конструктор без параметров |
| *public Element(int Index, Node Node1, Node Node2)* | Конструктор | – | *Public* | Конструктор c параметрами |

Продолжение таблицы 3.4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| *CreateMatrix(float t,float elasticModulus, float poissonsRatio)* | Метод | *void* | *Public* | Создание матриц |
| *CreateA()* | метод | *void* | *Public* | Создание площадей треугольников |
| *CreateD(float elasticModulus, float poissonsRatio)* | метод | *void* | *Public* | Создание матрицы деформаций |
| *CreateB()* | метод | *void* | *Public* | Нахождение матрицы *B* – градиентная матрица |
| *CreateK(float t)* | метод | *void* | *Public* | Нахождение локальной матрицы каждого треугольника |
| *SolveStress()* | метод | *void* | *Public* | Нахождение напряжений |

Класс *Matrix* предназначен для обработки данных считанных из текстого файла.

В таблице 3.5 представлена структура класса *Matrix.*

Таблица 3.5 – Структура класса *Matrix*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя переменной | Вид элемента | Тип | Спецификатор | Описание |
| 1 | 2 | 3 | 4 | 5 |
| *Multiplication(double[,] matrix1, double[,] matrix2)* | метод | *double[,]* | *static public* | Нахождение произведений матриц |
| *Transpose(double[,] matrix)* | метод | *double[,]* | *static public* | Транспонирование матрицы |
| *MultiplicationMatrix\_Number(double[,] matrix, double num)* | метод | *double[,]* | *static public* | Произведение матрицы на число |

Продолжение таблицы 3.5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| *Inverse Matrix(double[,] matrix)* | метод | *double[,]* | *static public* | Нахождение обратной матрицы помощью метода Гаусса |
| *SolutionOfGaussMethods (List<Node> nodes, double[,] globalMatrix)* | метод | *double[,]* | *static public* | Решение СЛАУ методом Гаусса |

### 3.2 Результаты выполнения разработанного приложения

Разработанное приложение на языке *С#* разбивает модель на конечно-элементную сетку как показано на рисунке 3.1.

Для работы приложения используются исходные данные и координаты точек.

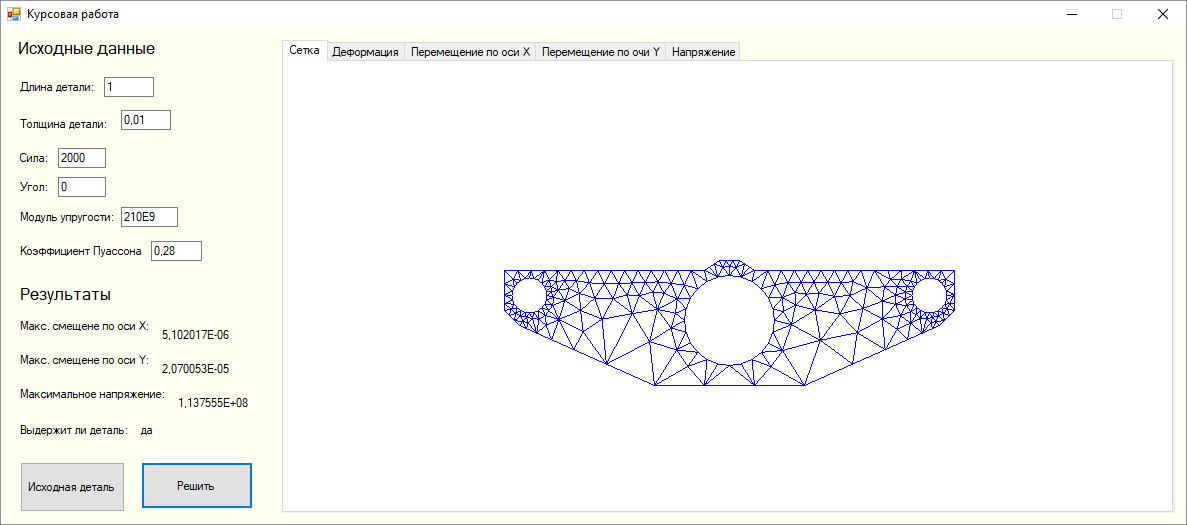


Рисунок 3.1 – Исходный рычаг, разбитый на конечные элементы

После составления конечно-элементной сетки и расчета, приложение строит деформированный рычаг, результат изображен на рисунке 3.2.

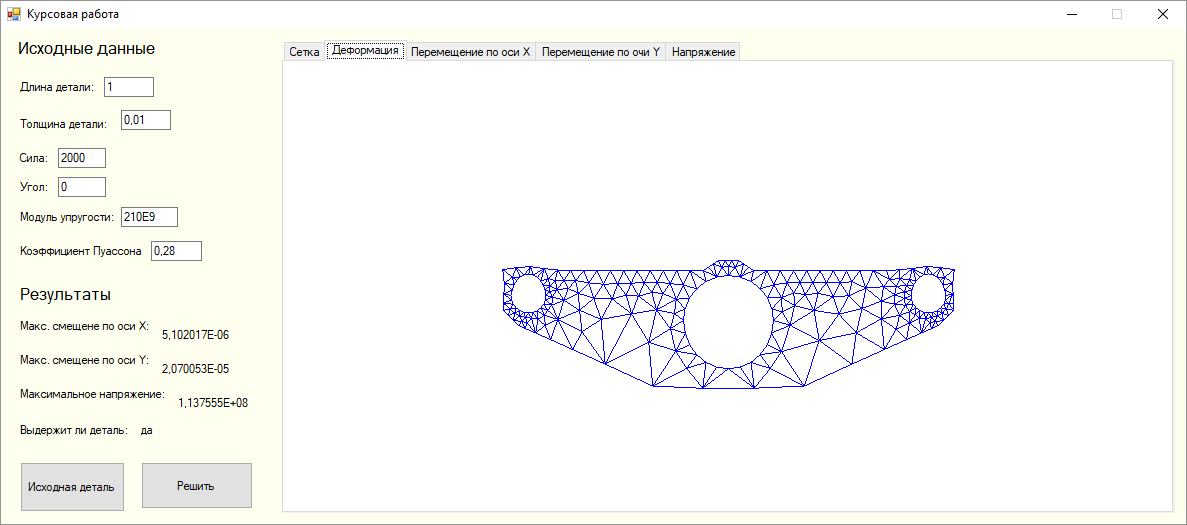


Рисунок 3.2 – Деформация рычага после приложения сил

Данное приложение выполняет вычисление оптимального размера рычага на каждом сечении, а также демонстрирует деформацию под приложенной силой.

Запуск программы осуществляется с помощью запуска .*exe*, который находится в папке *bin*/*debug*.

На главной форме окна как на рисунке 3.3, приложение предусматривает по умолчанию исходные данные и исходную фигуру. Для просмотра решения и деформации рычага пользователь по своему желанию изменяет, исходные данные или оставляет по умолчанию, и приложение ожидает нажатие кнопки «Решить».

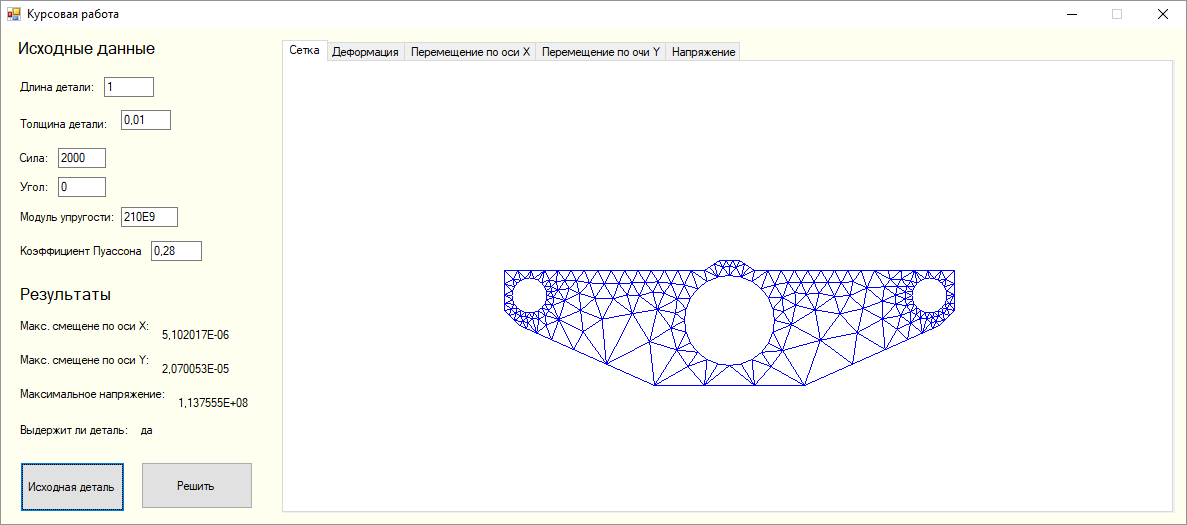


Рисунок 3.3 – Главное окно

Далее для того чтобы отобразить напряжение, пользователю необходимо нажать на вкладку «Напряжение» как показано на рисунке 3.4.

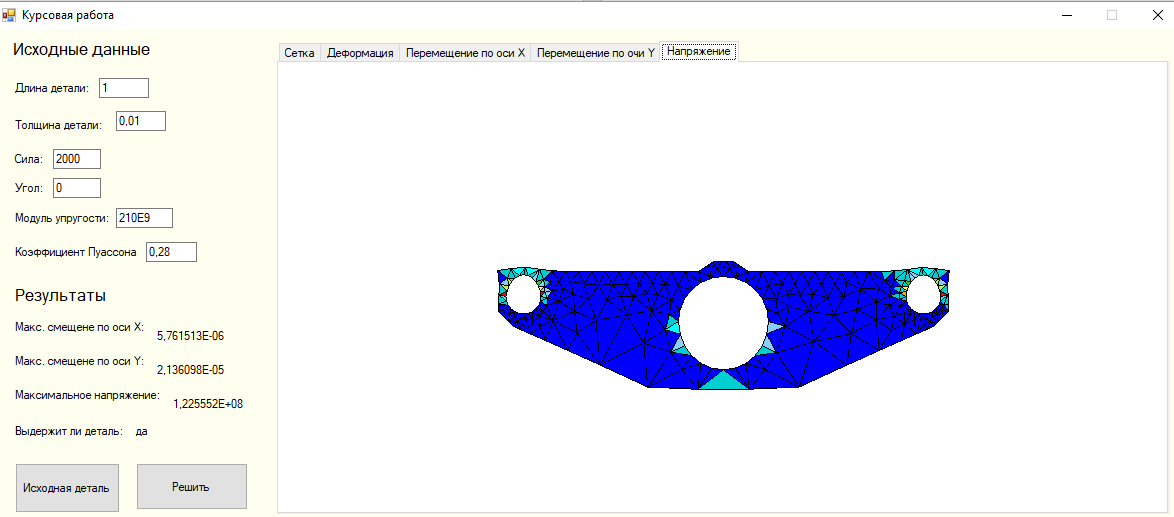


Рисунок 3.4 – Напряжение по всей площади детали

Координаты узлов и номера элементов вместе с узлами записаны в текстовых файлах, находящихся в папке с приложением, и имеющие названия *Nodes.txt* и *Elements.txt* соответственно. Данные необходимы для построения сетки.

### 4 ВЕРИФИКАЦИЯ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ

### 4.1 Моделирование задачи в пакете *Ansys*

Для проверки соответствия полученных в программе данных использовался конечно-элементный пакет *Ansys*.

Шаг 1. Создаем основную форму рычага при помощи точек:

*MainMenu* → *Preprocessor* → *Modeling* → *Create* → *Key points* → *IN Active CS.*

Затем необходимо создать объединить точки линиями.

Для создания линий по точкам необходимо выполнить:

*Main Menu* → *Preprocessor* → *Modeling* → *Create* → *Line* → *Standert Line*.

Также необходимо добавить две окружности по краям детали и одну в центре для этого необходимо выполнить:

*MainMenu* → *Preprocessor* → *Modeling* → *Create* → *Areas* → *Circle* →

*SolidCircle.*

Геометрические параметры, необходимые для построения детали сведены в таблицу 4.1.

Таблица 4.1 – Геометрические параметры фигуры, м

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Наименование фигуры | Координата *X* | Координата *Y* | Радиус |
| 1 | Круг-1 | 100 | 230 | 35 |
| 2 | Круг-2 | 900 | 230 | 35 |
| 3 | Круг-3 | 500 | 190 | 35 |

В таблице 4.2 представлены координаты точек для построения фигуры*.*

Таблица 4.2 – Координаты точек для построения фигуры

|  |  |  |  |
| --- | --- | --- | --- |
| № | Наименование точки | Координата *X* | Координата *Y* |
| 1 | 2 | 3 | 4 |
| 1 | Точка-1 | 50 | 280 |
| 2 | Точка-2 | 50 | 200 |
| 3 | Точка-3 | 80 | 170 |
| 4 | Точка-4 | 350 | 50 |
| 5 | Точка-5 | 650 | 50 |
| 6 | Точка-6 | 920 | 170 |
| 7 | Точка-7 | 950 | 200 |
| 8 | Точка-8 | 950 | 280 |

Продолжение таблицы 4.2

|  |  |  |  |
| --- | --- | --- | --- |
| 1 | 2 | 3 | 4 |
| 9 | Точка-9 | 550 | 280 |
| 10 | Точка-10 | 520 | 300 |
| 11 | Точка-11 | 480 | 300 |
| 12 | Точка-12 | 450 | 280 |

Шаг 2. Вырезаем из пластины отверстия:

*Main Menu* → *Preprocessor* → *Modeling* → *Operate* → *Booleans* → *Substract* → *Areas.*

Результат выполнения шага 2 представлен на рисунке 4.2.

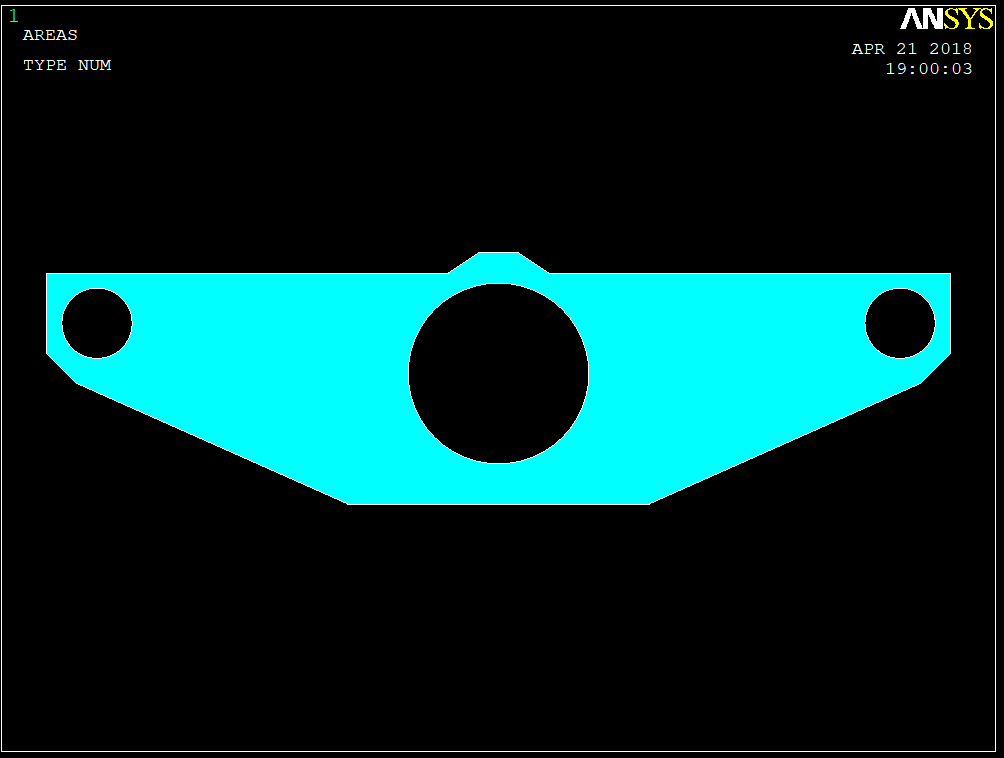


Рисунок 4.2 – Результат выполнения шага 2

Шаг 3. Проводим операцию выдавливания для создания пластины заданной толщины:

*Main Menu* → *Preprocessor* → *Modeling* → *Operate* → *Extrude* → *Areas* → *Along Normal* и выбираем всю пластину.

Появляется окно *Extrude Area a long Normal*: в строке *DIST Length of extrusion* вводим значение толщины пластины равное 0.001 м.

Результат выполнения шага 3 представлен на рисунке 4.3.

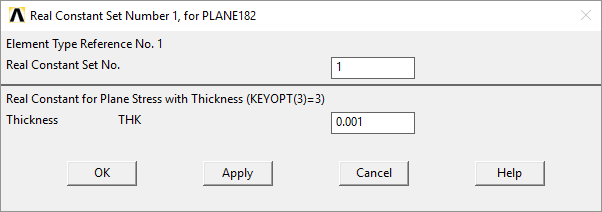


Рисунок 4.3 – Окно параметров выдавливания

Шаг 4. Выбираем тип конечного элемента:

*Main Menu* → *Preprocessor* → *Element Type* → *Add/Edit/Delete*

В появившемся окне присутствует надпись *NONE DEFINED*, означающая, что не выбран ни один из типов конечных элементов. Чтобы выбрать (добавить) элемент нажимаем кнопку «*Add*…».

В появившемся окне выбираем конечный элемент, соответствующий типу решаемой задачи – элемент из раздела *StructuralSolid* (прочностная задача для твердых тел) 4*node* 182.

Результат выполнения шага 4 представлен на рисунке 4.4.

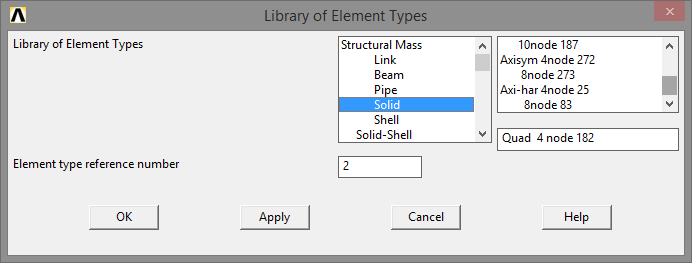


Рисунок 4.4 – Библиотека конечных элементов

Шаг 5. Задаем материал пластины:

*Main Menu* → *Preprocessor* → *Material Props* → *Material Models.*

Появляется окно, показанное на рисунке4.5.

В поле *Material Models Available* выбираем изотропный упругий материал:

*Structural* → *Linear* → *Elastic* → *Isotropic.*

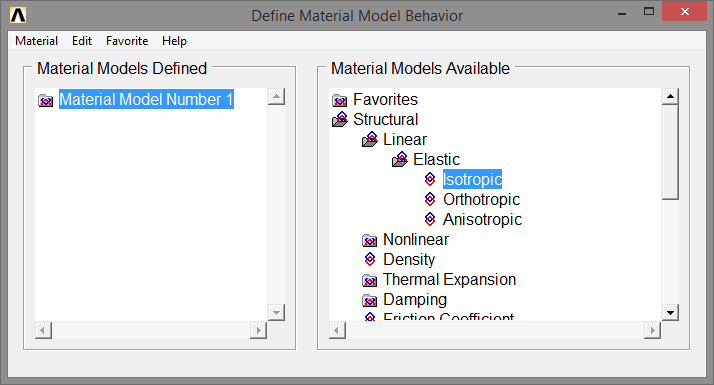


Рисунок 4.5 – Окно определения модели материала

В появившемся окне (рисунок 4.6) задаем значения механических характеристик материала полупространства: модуль упругости *EX* и коэффициент Пуассона *PRXY*.

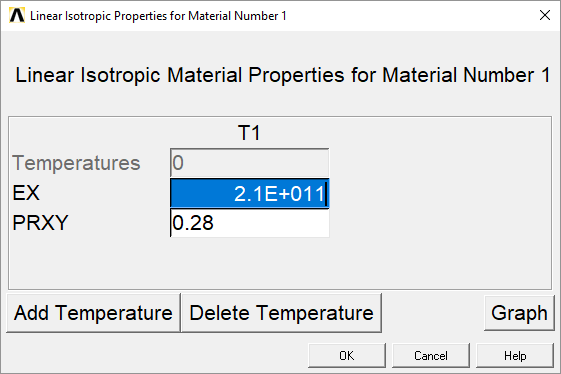


Рисунок 4.6 – Механические характеристики материала

После нажатия кнопки «ОК» в окне, показанном на рисунке 4.6 в поле *MaterialModelsDefined* появляется подпись *LinearIsotropic*, что свидетельствует о создании соответствующей модели материала. Закрываем окно, показанное на рисунке 4.6.

Шаг 6. Создаем конечно-элементную модель задачи с треугольными конечными элементами, для чего вызываем панель *Meshtool*:

*MainMenu* → *Preprocessor* → *Meshing* → *MeshTool*

Нажимаем кнопку «*Mesh*» панели *Meshtool*. В появившемся окне *MeshVolumes* нажимаем кнопку «*PickAll*» для выделения всей геометрической модели.

Полученная конечно-элементная модель показана на рисунке 4.7.

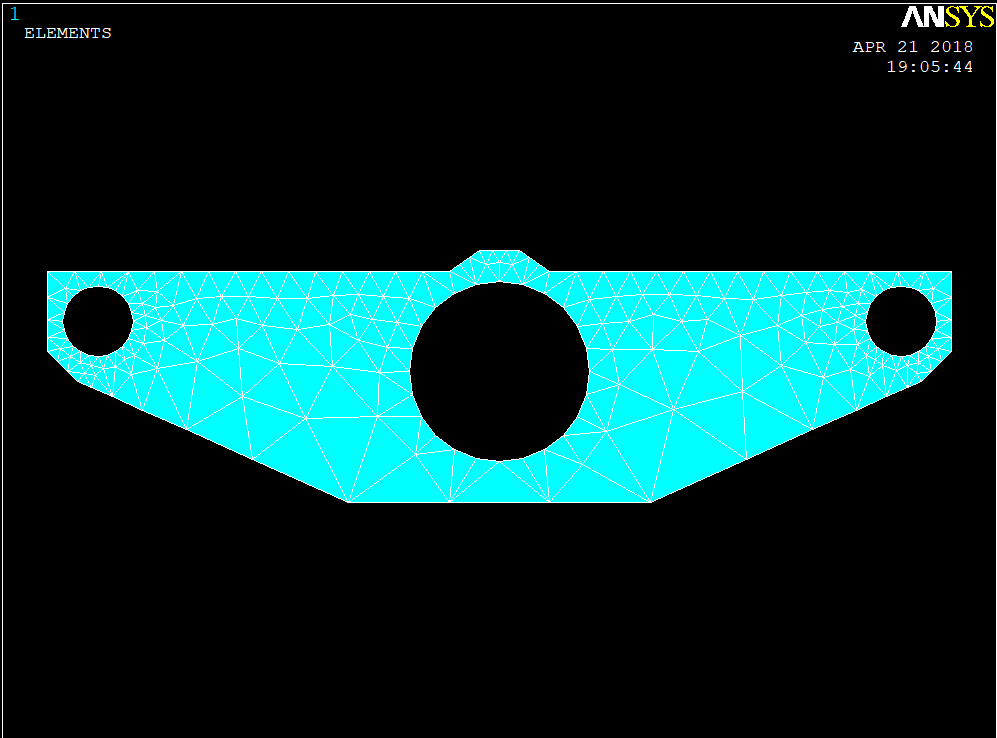


Рисунок 4.7 – Конечно-элементная модель задачи

Шаг 7. Определяем нагрузки и накладываем ограничения. На данном этапе для удобной работы необходимо включить нумерацию всех поверхностей пластины.

*UtilityMenu* → *PlotCtrls* → *Numbering*

В появившемся окне выбираем пункт *AREA Area numbers* и нажимаем кнопку «ОК».

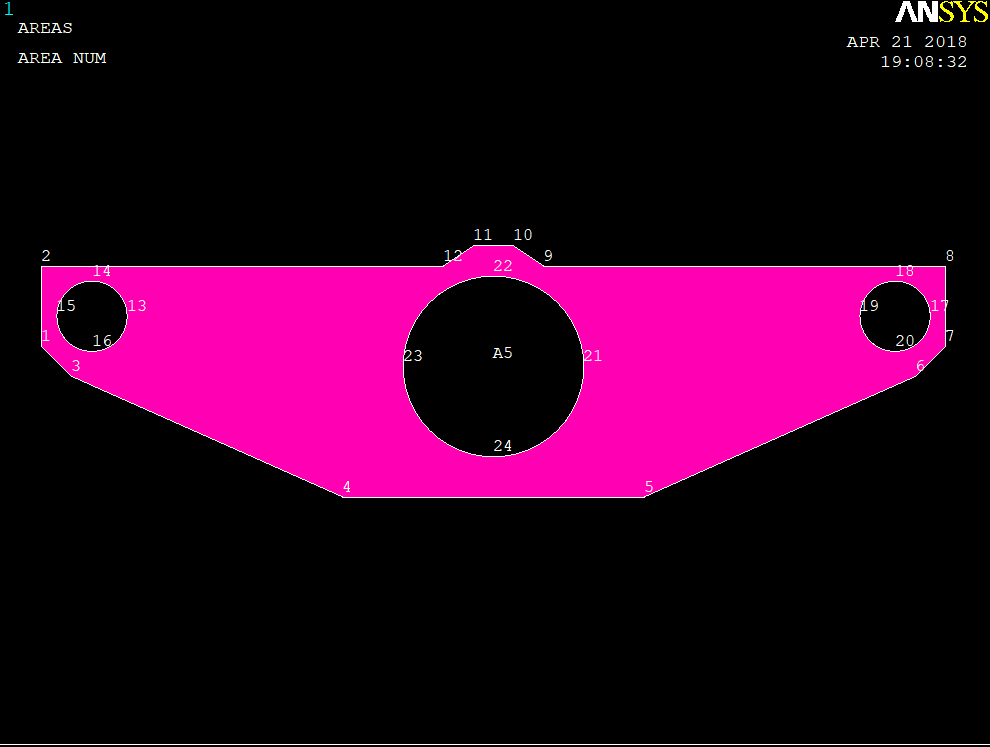


Рисунок 4.8 – Номера поверхностей модели

Определяем распределенную нагрузку, действующую на пластину:

*Main Menu* → *Preprocessor* → *Loads* → *Define Loads* → *Apply* → *Structural* → *Force* → *On Nodes*

Выбираем правую сторону. Далее появляется окно (рисунок 4.9), в котором задаем значение распределенной нагрузки (*VALUE Load PRES value*) 2000 Н/м2 , нажимаем кнопку «ОК».

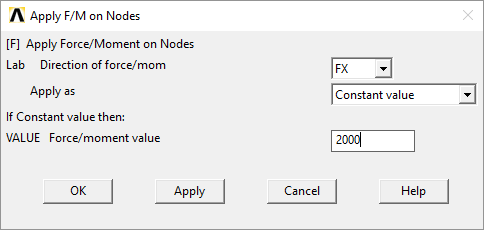


Рисунок 4.9 – Окно параметров распределенной нагрузки

Определяем перемещения на границах модели:

*Main Menu* → *Preprocessor* → *Loads* → *Define Loads* → *Apply* → *Structural* → *Displacement* → *On Areas.*

В списке с меткой *Lab2 DOFs to be constrained* второго окна *Apply U*, *Rot on Areas* (рисунок 4.10) выбираем левую сторону. В поле с меткой *VALUE Displacement value* вводим 0, нажимаем кнопку «ОК».

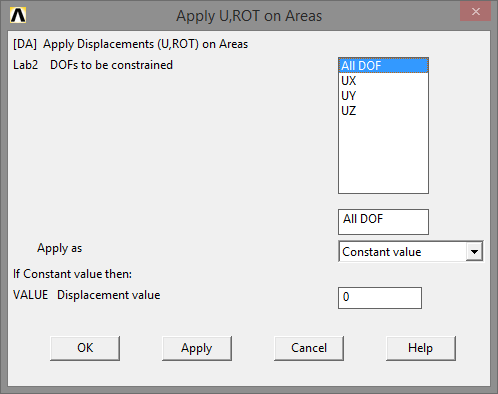


Рисунок 4.10 – Окно ограничений перемещений поверхности.

Результат выполнения шага 7 представлен на рисунке 4.11.

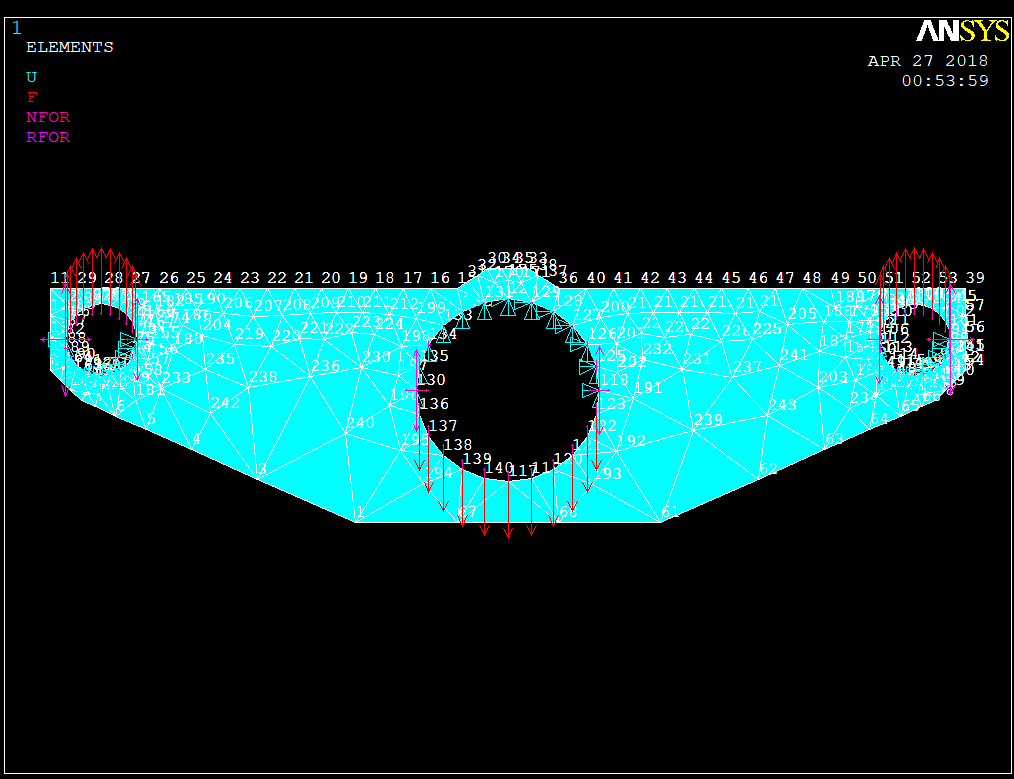


Рисунок 4.11 – Ограничения и нагрузки модели

### 4.2 Получение результатов в пакете *Ansys*

Шаг 1. Решение задачи:

*MainMenu* → *Solution* → *Solve* → *CurrentLS.*

Закрываем окно */STATUSCommand*.

В окне *SolveCurrentLoadStep* нажимаем кнопку «ОК». Решение продолжается до появления информационного сообщения *SolutionisDone*, которое следует закрыть кнопкой «*Close*».

Шаг 2. Просмотр результатов решения задачи:

*Main Menu* → *General Postproc* → *Plot Results* → *Contour Plot* → *Nodal Solution*

Просмотрим напряжения, полученные при решении задачи, для чего в поле*Item to be contoured* выбираем раздел *Stress* и подразделы: *XY Sheat stress, X-component of stress, Y-component of stress*, нажимаем кнопку «ОК».

Полученное суммарное смещение по оси *ХУ* показана на рисунке 4.12.

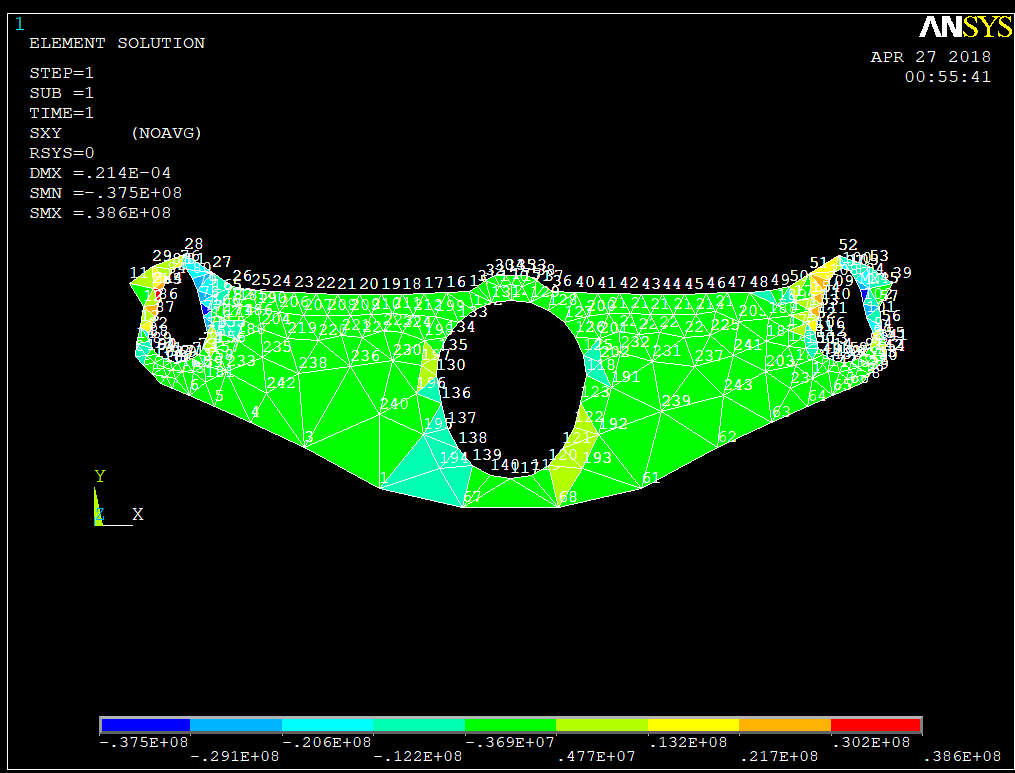


Рисунок 4.12­– Суммарное смещение по оси *ХУ*

Полученное суммарное смещение по оси *OX* показана на рисунке 4.13.

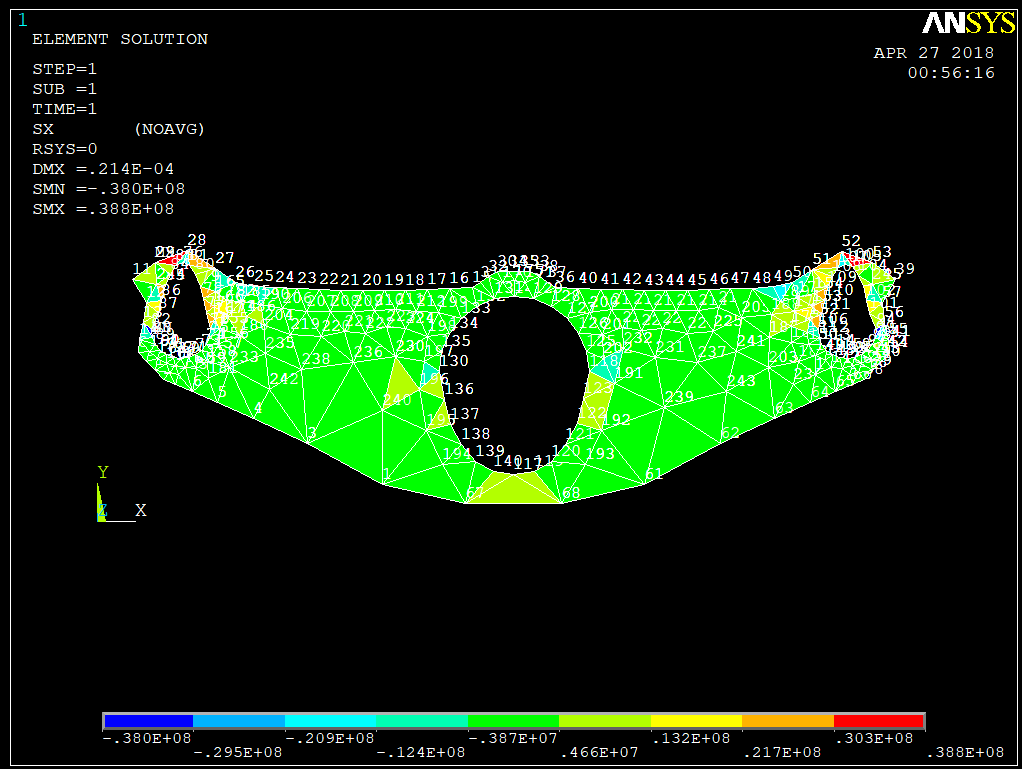


Рисунок 4.13­– Суммарное смещение по оси *OX*

Полученное суммарное смещение по оси *OY* показана на рисунке 4.14.

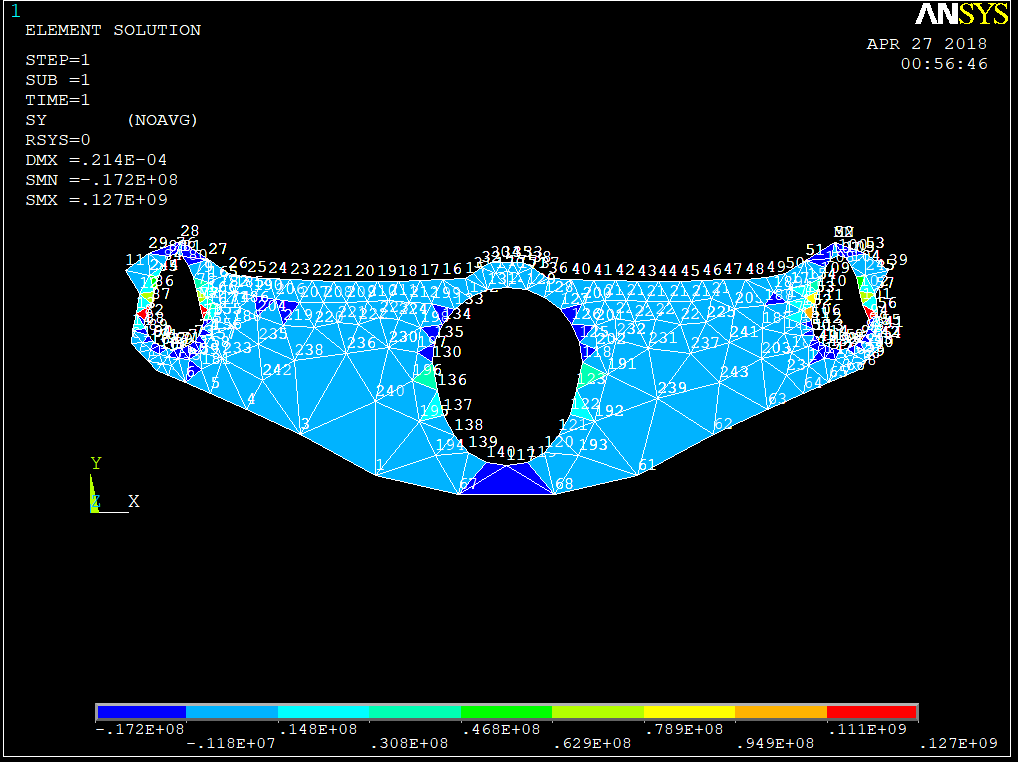


Рисунок 4.14­– Суммарное смещение по оси *OY*

Для просмотра смещения конечных элементов необходимо выбрать:

*Main Menu* → *General Postproc* → *Plot Results* → *Contour Plot* → *Nodal Solution* → *Element Solution* → *Stress* → *von Mises stress.*

Смещение конечных элементов *OY* показана на рисунке 4.15.

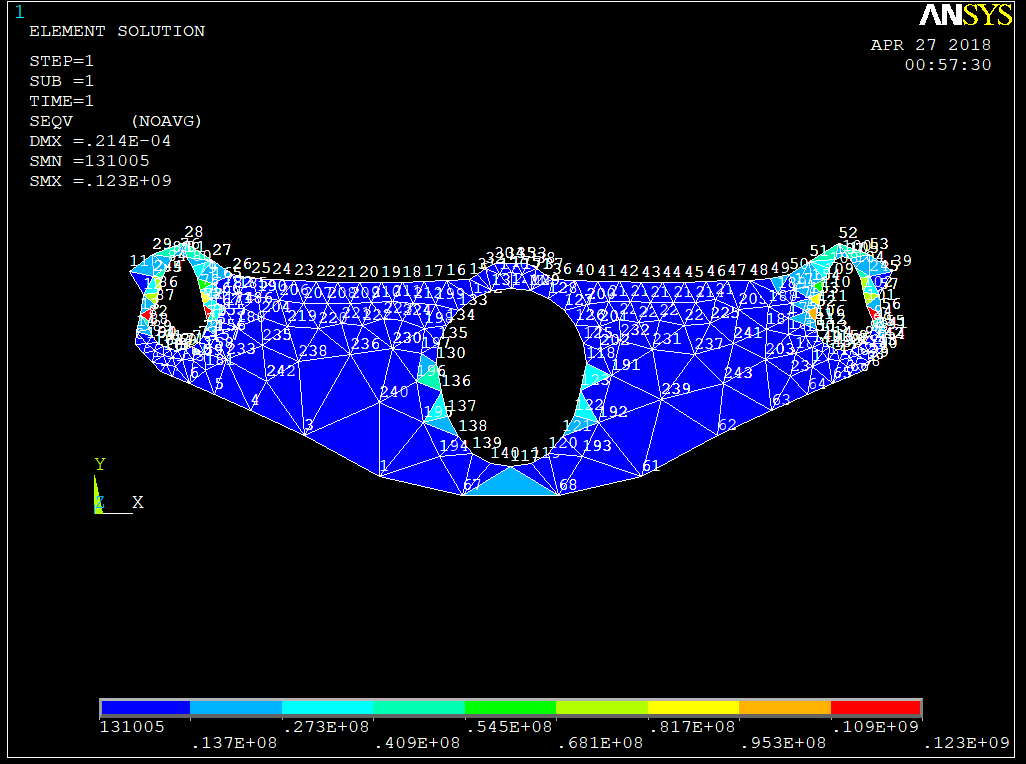


Рисунок 4.15 – Смещение конечных элементов

### 4.3 Исследование полученных результатов

В результате проделанной работы было разработано приложение, которое позволяет определить оптимальный размер равноплечего рычага подвески железнодорожной платформы. Результаты, полученные в программе, занесены в таблицу 4.3.

Таблица 4.3 – Результаты вычислений

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Толщина детали, м | *ANSYS* | | *Program* | | Погрешность |
| *UX*, м | *UY*, м | *UX*, м | *UY*, м |
| 0,01 | 0.3029 | 0.1331 | 0.2910 | 1.3314 | 1% |
| 0,02 | 0.1448 | 0.6661 | 0.1455 | 0.6657 | 1% |
| 0,03 | 0.0966 | 0.4447 | 0.0970 | 0.4443 | 2% |

Анализируя вычислительный эксперимент, следует сделать вывод, что при толщине детали 3 см и силе до 2000 Н\*м при предельно допустимом смещении по оси *X* 0,5 происходит смещение на 0,09 мм, что находиться в пределе нормы. При толщине детали 1 см и силе в 2000 Н\*м происходит смещение на 0,30 мм при предельно допустимом 0,08 мм. Отсюда следует вывод, что деталь данную нагрузку не выдержит. Деталь должна проектироваться с минимальной толщиной в 3 см, в зависимости от нагрузок на деталь, иначе будет происходить быстрый износ и поломка.

### ЗАКЛЮЧЕНИЕ

В результате проделанной работы было разработано приложение, которое моделирует напряженно-деформированное состояние равноплечего рычага подвески железнодорожной платформы при наличии равномерно распределенной нагрузки. Данное приложение разбивает элемент на конечные элементы и находит перемещения в узлах при заданных нагрузках и закреплениях.

В программе реализован графический интерфейс пользователя. Программа позволяет эффективно решить поставленную задачу, используя метод конечных элементов.

Данное приложение вычисляет искомые результаты, что делает возможным использования в практических условий для ускорения получения экспериментальных данных. Разработанный программный комплекс позволяет быстро рассчитывать перемещения в узлах плоской конструкции. При усовершенствовании программы, возможно, ее использование для расчета более сложных задач. Это позволит сократить количество натурных экспериментов, которые в свою очередь несут большие затраты как денежные так и временные.

### СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Википедия, свободная энциклопедия // Метод конечных элементов [Электронный ресурс]. – Режим доступа: http://*ru*.*wikipedia*.*org*. – Дата доступа: 18.04.2018.
2. Барабаш, М. Руководство пользователя по Лире 9.0 Книга 3.  
   Примеры расчета и проектирования / М. Барабаш – К.: Факт, 2002. – 106 с.
3. Барабаш, М. ЛИРА-САПР 2013/ М. Барабаш, В. Титок – К.: Наука, 2013. – 376 с.

4. Каплун, А.Б. Ansys в руках инженера: Практическое руководство / Е.М. Морозов, М.А. Олферьева. – М.: Едиториал УРСС, 2003. – 272 с.

5. Варвак, П.М. Справочник по теории упругости (для инженеров - строителей) / А.Ф. Рябов. – К.: Будивельник, 1971. – 418 с.

6. Ильин, В. Численные методы решения задач строительной механики / В. Ильин, А. Масленникова. – К.: Будивельник, 1971. – 418 с.

7. Зенкевич, О. Метод конечных элементов в технике / О. Зенкевич. – М.: Мир, 1975. – 541 с.

8. Галлагер, Р. Метод конечных элементов. Основы / Р. Галлагер. – М. : Мир, 1984. – 428 с.

# ПРИЛОЖЕНИЕ A

(*обязательное*)

**Листинг приложения**

namespaceCourseWork

{

public partial class MainForm : Form

{

float dLength = 1, thickness, defaultLength = 0.1f, delta = 9.5f;

//int[] indexNodes = new int[] { 419, 525, 420, 526, 17, 16, };

int[] indexNodes = new int[] { 8, 105, 112, 103, 104, 113, 216, 211, 215, 10, 99, 106 };

float elasticModulus, poissonsRatio, force;

float[,] globalK;

float dopStress = 245000, maxStress;

List<Node> nodes = new List<Node>();

List<Element> elements = new List<Element>();

Graphics gr;

SolidBrush color1, color2, color3, color4, color5, color6, color7, color8, color9;

public MainForm()

{

InitializeComponent();

gr = panel.CreateGraphics();

color1 = new SolidBrush(Color.Blue);

color2 = new SolidBrush(Color.DarkTurquoise);

color3 = new SolidBrush(Color.LightSkyBlue);

color4 = new SolidBrush(Color.Aqua);

color5 = new SolidBrush(Color.LimeGreen);

color6 = new SolidBrush(Color.GreenYellow);

color7 = new SolidBrush(Color.Yellow);

color8 = new SolidBrush(Color.Orange);

color9 = new SolidBrush(Color.Red);

}

private void MainForm\_Load(object sender, EventArgs e)

{

nodes = File.ReadFileNodes();

elements = File.ReadFileElements(nodes);

}

private void buttonDetails\_Click(object sender, EventArgs e) //отрисовкадетали

{

gr.Clear(Color.White);

for (int i = 0; i < nodes.Count; i++)

{

nodes[i].X = nodes[i].X / dLength;

nodes[i].Y = nodes[i].Y / dLength;

}

thickness = Convert.ToSingle(textBoxThickness.Text);

for (int i = 0; i < nodes.Count; i++)

{

nodes[i].X = nodes[i].X \* dLength;

nodes[i].Y = nodes[i].Y \* dLength;

}

DrawingDetails();

buttonGrid.Enabled = true;

}

void DrawingDetails() //Методотрисовкидетали

{

Pen p1 = new Pen(Color.Blue, 1);

gr.DrawEllipse(p1, 0, 0, 380, 380);//внешний круг

//дуга

gr.DrawArc(p1, 95, 95, 190, 190, 112, 316);

// шпонка

gr.DrawLine(p1, 153, 322, 227, 322);

gr.DrawLine(p1, 153, 280, 153, 322);

gr.DrawLine(p1, 227, 280, 227, 322);

}

private void buttonGrid\_Click(object sender, EventArgs e) //сетка, закрепления, силы

{

gr.Clear(Color.White);

DrawingDetails();

Meshing(Color.Green);

DrawingFixation();

DrawingForce();

buttonSolve.Enabled = true;

}

void Meshing(Color color) //Методотрисовкисетки

{

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, elements[i].N2.X \* delta, elements[i].N2.Y \* delta);

gr.DrawLine(p, elements[i].N2.X \* delta, elements[i].N2.Y \* delta, elements[i].N3.X \* delta, elements[i].N3.Y \* delta);

gr.DrawLine(p, elements[i].N3.X \* delta, elements[i].N3.Y \* delta, elements[i].N1.X \* delta, elements[i].N1.Y \* delta);

}

}

void DrawingFixation() //Методотрисовкизакрепления

{

Pen p1 = new Pen(Color.Blue, 5);

gr.DrawArc(p1, 95, 95, 190, 190, 112, 316);

}

void DrawingForce() //Методотрисовкисил

{

Pen p = new Pen(Color.Red, 1);

p.EndCap = LineCap.ArrowAnchor;

//верх

gr.DrawLine(p, 190, 300, 220, 300);

gr.DrawLine(p, 190, 315, 220, 315);

}

private void buttonSolve\_Click(object sender, EventArgs e) //Решение

{

gr.Clear(Color.White);

elasticModulus = Convert.ToSingle(textBoxElasticModulus.Text);

poissonsRatio = Convert.ToSingle(textBoxPoissonsRatio.Text);

force = Convert.ToSingle(textBoxForce.Text);

for (int i = 0; i < elements.Count; i++)

elements[i].CreateMatrix(thickness, elasticModulus, poissonsRatio);

globalK = new float[nodes.Count \* 2, nodes.Count \* 2 + 1];

// MessageBox.Show(globalK.Length.ToString() + " " + globalK.GetLength(0) + " " + globalK.GetLength(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 \* listIndex[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 < indexNodes.Length; i++)

globalK[indexNodes[i] \* 2, nodes.Count \* 2] = force;

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 = Matrix.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";

}

void MaxDeformationXAndY() //Определение максимальных смещений и напряжения

{

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();

gr.Clear(Color.White);

Meshing(Color.Green);

}

}

}

namespace CourseWork

{

class File

{

static public List<Node> ReadFileNodes() //Считываниеизфайлаузлов

{

Node node;

List<Node> nodes = new List<Node>();

StreamReader file = new StreamReader("Nodes.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]), Convert.ToSingle(ss[2]), true);

else

node = new Node(Convert.ToInt32(ss[0]), Convert.ToSingle(ss[1]), Convert.ToSingle(ss[2]), false);

nodes.Add(node);

}

return (nodes);

}

static public List<Element> ReadFileElements(List<Node> nodes) //Считываниеизфайлаэлементов

{

Element element;

List<Element> elements = new List<Element>();

StreamReader file = new StreamReader("Elements.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);

}

}

}

namespace CourseWork

{

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; }

}

}

}

namespace CourseWork

{

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 = Matrix.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 = Matrix.MultiplicationMatrixAndNumber(d, koeff);

}

void CreateB()

{

b = new float[3, 6];

b = Matrix.MatrixMultiplication(q, a\_1);

}

void CreateK(float t)

{

k = new float[6, 6];

float[,] transpB = Matrix.Transpose(b);

k = Matrix.MatrixMultiplication(transpB, d);

k = Matrix.MatrixMultiplication(k, b);

k = Matrix.MultiplicationMatrixAndNumber(k, t);

float ds = triangleArea(n1.X, n1.Y, n2.X, n2.Y, n3.X, n3.Y);

k = Matrix.MultiplicationMatrixAndNumber(k, ds);

}

public void SolveStress()

{

float[,] bb = Matrix.MatrixMultiplication(q, a\_1);

e = Matrix.MatrixMultiplication(bb, sig);

stress = Matrix.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; }

}

}

}

namespace CourseWork

{

class Matrix

{

static public float[,] MatrixMultiplication(float[,] a, float[,] b) //Произведениематриц

{

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);

}

static public float[,] Transpose(float[,] a) //Транспонированиематрицы

{

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);

}

static public float[,] MultiplicationMatrixAndNumber(float[,] a, float b) //Произвадениематрицыначисло

{

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);

}

static public float[,] FindingTheInverseMatrix(float[,] a) //НахождениеобратнойматрицыспомощьюметодаГаусса

{

float[,] A1 = new float[a.GetLength(0), a.GetLength(1)];

for (int i = 0; i < 6; i++)

A1[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];

A1[i, j] = A1[i, j] + A1[index, j];

A1[index, j] = A1[i, j] - A1[index, j];

A1[i, j] = A1[i, j] - A1[index, j];

}

}

}

variable = a[i, i];

for (int j = 0; j < 6; j++)

{

a[i, j] = a[i, j] / variable;

A1[i, j] = A1[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];

A1[j, k] = A1[j, k] - variable \* A1[i, k];

}

}

}

}

return (A1);

}

static public float[,] Gaus(List<Node> nodes, float[,] global) //РешениеСЛАУметодомГаусса

{

float variable;

for (int i = 0; i < nodes.Count \* 2; i++)

{

if (global[i, i] == 0)

{

int index = i + 1;

while (index < nodes.Count \* 2)

{

if (global[index, i] != 0)

break;

index++;

}

if (index != nodes.Count \* 2)

{

for (int j = i; j < nodes.Count \* 2 + 1; j++)

{

global[i, j] = global[i, j] + global[index, j];

global[index, j] = global[i, j] - global[index, j];

global[i, j] = global[i, j] - global[index, j];

}

}

}

variable = global[i, i];

for (int j = nodes.Count \* 2; j >= i; j--)

{

global[i, j] = global[i, j] / variable;

}

for (int j = i + 1; j < nodes.Count \* 2; j++)

{

variable = global[j, i];

if (variable != 0)

for (int k = nodes.Count \* 2; k >= i; k--)

global[j, k] = global[j, k] - variable \* global[i, k];

}

}

for (int i = nodes.Count \* 2 - 1; i > 0; i--)

{

for (int j = i - 1; j >= 0; j--)

{

global[j, nodes.Count \* 2] -= global[j, i] \* global[i, nodes.Count \* 2];

global[j, i] = 0;

}

}

return (global);

}

}

}

# ПРИЛОЖЕНИЕ Б

(*обязательное*)

# *Ansys* скрипт создания модели детали

/PREP7

CYL4,20,20,20, ,10

BLC4,16,20,8,14

ASBA, 1, 1

ASBA, 1, 2

!\*

ET,1,PLANE182

!\*

KEYOPT,1,1,0

KEYOPT,1,3,3

KEYOPT,1,6,0

!\*

!\*

R,1,0,02,

!\*

!\*

MPTEMP,,,,,,,,

MPTEMP,1,0

MPDATA,EX,1,,210E9

MPDATA,PRXY,1,,0.28

SMRT,6

SMRT,5

SMRT,4

SMRT,3

MSHAPE,1,2D

MSHKEY,0

!\*

CM,\_Y,AREA

ASEL, , , , 3

CM,\_Y1,AREA

CHKMSH,'AREA'

CMSEL,S,\_Y

!\*

AMESH,\_Y1

!\*

CMDELE,\_Y

CMDELE,\_Y1

CMDELE,\_Y2

!\*

/UI,MESH,OFF

FLST,2,1,4,ORDE,1

FITEM,2,15

FLST,2,4,4,ORDE,4

FITEM,2,5

FITEM,2,-6

FITEM,2,13

FITEM,2,-14

!\*

/GO

DL,P51X, ,ALL,

FLST,2,4,1,ORDE,4

FITEM,2,8

FITEM,2,10

FITEM,2,105

FITEM,2,-106

!\*

/GO

F,P51X,FX,50

# ПРИЛОЖЕНИЕ В

# (*обязательное*)

# Чертеж детали

