Всероссийский конкурс учебно-исследовательских работ старшеклассников по политехническим дисциплинам для учащихся 9-11 классов

Математическое моделирование

**Разработка приложения для решения задач стационарной теплопроводности с помощью методы конечных элементов**

Выполнила:

Шайхутдинова А. А.,

11 класс

Руководитель:

Трушков В. А.

Пермь. 2021.

Оглавление

[Введение 3](#_Toc63843232)

[Теоретическая часть 4](#_Toc63843233)

[Метод конечных элементов 4](#_Toc63843234)

[Построение дискретной модели на примере одномерной задачи о распределении температуры в стержне 6](#_Toc63843235)

[Преимущества МКЭ 8](#_Toc63843236)

[Метод Галеркина 9](#_Toc63843237)

[Уравнение стационарной теплопроводности 9](#_Toc63843238)

[Аппроксимация решения кусочно-линейными функциями 10](#_Toc63843239)

[Процедура ансамблирования конечных элементов 14](#_Toc63843240)

[Практическая часть 20](#_Toc63843241)

[Этапы разработки программы 20](#_Toc63843242)

[Разработка интерфейса программы 21](#_Toc63843243)

[Создание структуры классов геометрических фигур 23](#_Toc63843244)

[Создание базы данных 24](#_Toc63843245)

[Разработка и реализация алгоритма для решения задач стационарной теплопроводности в балочных конструкциях 25](#_Toc63843246)

[Полная структура программы 27](#_Toc63843247)

[Демонстрация работы программы 28](#_Toc63843248)

[Заключение 32](#_Toc63843249)

[Список литературы 33](#_Toc63843250)

[Приложение 34](#_Toc63843251)

# Введение

Метод конечных элементов (МКЭ) является наиболее часто используемым методом для решения различных задач математической физики. Основными преимуществами МКЭ считаются его большая универсальность, общая для всех сеточных методов, простота физической интерпретации и высокая алгоритмичность, что позволяет использовать данный метод для ускорения вычислений ЭВМ.

Целью работы является разработка программы для решения задач стационарной теплопроводности. Для ее осуществления необходимо выполнить ряд следующих задач:

* Изучить теорию МКЭ.
* Изучить методы решения систем линейных алгебраических уравнений (СЛАУ).
* Изучить язык программирования Python, с целью ускорения процесса моделирования данного явления.
* Разработать программу для решения задач стационарной теплопроводности.

##

# Теоретическая часть

## Метод конечных элементов

Метод конечных элементов (МКЭ) – численный метод решения дифференциальных уравнений, в настоящее время получивший глубокие теоретические обоснования и применяемый для решения весьма широкого спектра задач, например:

* стационарные задачи распространения тепла, диффузии, распределения электрического поля, другие задачи теории поля;
* задачи гидромеханики, в частности, течение жидкости в пористой среде;
* задачи механики и прочности, в т.ч. проектирование самолётов, ракет и различных пространственных оболочек.

Основная идея МКЭ состоит в том, что:

* любую непрерывную величину (например, температуру, давление, перемещение) можно аппроксимировать дискретной моделью, которая строится на множестве кусочно-непрерывных функций, определенных на конечном числе подобластей (элементов);
* кусочно-непрерывные функции определяются с помощью значений непрерывной величины в конечном числе точек рассматриваемой области.

При построении дискретной модели непрерывной величины поступают следующим образом.

1. В рассматриваемой области фиксируется конечное число точек. Эти точки называются узловыми (или просто узлами).

2. Значение непрерывной величины в каждой узловой точке считается переменной, которая должна быть определена.

3. Область определения непрерывной величины разбивается на конечное число подобластей, называемых элементами (или конечными элементами). Эти элементы имеют общие узловые точки и в совокупности аппроксимируют форму области.

4. Непрерывная величина аппроксимируется на каждом элементе полиномом (или какой-либо другой функцией), который определяется с помощью узловых значений этой величины. Для каждого элемента определяется свой полином, но полиномы подбираются таким образом, чтобы сохранялась непрерывность величины вдоль границ элемента. Этот полином еще называют функцией элемента.

Основная концепция МКЭ может быть наглядно проиллюстрирована на примере заданного распределения температуры в стержне.

## Построение дискретной модели на примере одномерной задачи о распределении температуры в стержне

Рассматривается непрерывная величина , область определения которой отрезок OL вдоль оси X. Фиксированы и пронумерованы пять точек на оси X. (рис.1.)[3]

Эти узловые точки можно расставить на любом расстоянии друг от друга. Значение в данном случае известно в каждой узловой точке. Эти фиксированные значения представлены графически на рисунке и обозначены в соответствии с номерами узловых точек через . (рис.2.)

****

****

Рис.1.

****

Рис.2.

Разбиение области на элементы можно произвести двумя различными способами. Можно, ограничить каждый элемент двумя соседними узловыми точками, образовав четыре элемента или разбить область на два элемента, каждый из которых содержит три узла. Соответствующий элементу полином определяется по значениям в узловых точках.

В случае разбиения области на четыре элемента на каждый элемент приходится по два узла, и функция элемента будет линейна по оси Х (две точки однозначно определяют прямую линию). Окончательная аппроксимация будет состоять из четырех кусочно-линейных функций, каждая из которых определена на отдельном элементе.

## Преимущества МКЭ

Важными преимуществами МКЭ, благодаря которым он широко используется, является следующее:

1) Свойства материалов не должны быть обязательно одинаковыми. Это позволяет применить метод к телам, составленным из нескольких материалов

2) Криволинейная область может быть аппроксимирована с помощью прямолинейных конечных элементов (например, с помощью треугольных, призматических, шестигранных конечных элементов).

3) Размеры элементов могут быть переменными. Это позволяет укрупнить или уменьшить сетку разбиения области на элементы, если в этом есть необходимость

Указанные выше преимущества МКЭ могут быть использованы при составлении достаточно общей программы для решения частных задач определенного класса.

## Метод Галеркина

Разрывный метод Галёркина — метод решения операторных уравнений, в основном дифференциальных уравнений. Является развитием классического МКЭ, основанного на [вариационной постановке Галёркина](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%93%D0%B0%D0%BB%D1%91%D1%80%D0%BA%D0%B8%D0%BD%D0%B0); а также частным случаем метода моментов (взвешенных невязок).

В отличие от классического МКЭ метод позволяет получить решение, которое может быть разрывно, что является плюсом в задачах, где решение имеет резкие скачки (то есть разрывно или близко к этому). Так же метод удобен при работе с несогласованными сетками и с базисами разного порядка на элементах, поскольку не требует дополнительного согласования (что нужно было делать в классическом методе).

Рассмотрим последовательность применения метода Галеркина для решения такой прикладной задачи, как уравнение стационарной теплопроводности.

### Уравнение стационарной теплопроводности

Распределение температуры в тонком однородном стержне с теплоизолированной боковой поверхностью (рис. 8, а) описывается параболическим уравнением стационарной теплопроводности

|  |  |
| --- | --- |
|  | (5.1) |

с граничными условиями

|  |  |
| --- | --- |
| ; , | (5.2) |

где – температура, – мощность внутреннего теплового источника, , – проекции векторов тепловых потоков на внешние нормали к торцевым поверхностям стержня на левом и правом концах,– коэффициент теплопроводности[2].



Рис.8.

Для упрощения выкладок величины и считаются постоянными.

### Аппроксимация решения кусочно-линейными функциями

Стержень, имеющий длину L, разбивается на 4 равных отрезка длиной каждый. Для произвольного отрезка (рис. 8, б) температурное поле описывается уравнением (5.1), граничные условия (5.2) записываются в форме

|  |  |
| --- | --- |
| , , | (5.3) |

где , – проекции векторов тепловых потоков на внешние нормали к границам конечного элемента. Для построения разрешающих соотношений метода Галеркина выбираются кусочно-линейные пробные функции в виде

 , .

 Эти функции позволяют искать решение задачи на отрезке в виде

|  |  |
| --- | --- |
|  | (5.4) |

где – узловые значения искомого распределения температуры[4]. Невязка [1] уравнения(5.1), получаемая на приближении (5.4), взвешивается с использованием тех же функций и

|  |  |
| --- | --- |
|  | (5.5) |

 Первое из этих уравнений преобразуется к виду

|  |
| --- |
|  |
|  |
| . |

Поскольку по построению пробные функции, , из последнего выражения следует

С учетом условия (5.3) и представления решения в виде (5.4) получается выражение

|  |  |
| --- | --- |
|  | (5.6) |

 Преобразования второго уравнения системы (5.5) приводят к уравнению

|  |  |
| --- | --- |
|  | (5.7) |

В итоге получена система линейных алгебраических уравнений относительно узловых значений температур и , то есть коэффициентов разложения (5.4) решения по пробным функциям. Подсчитываются интегралы в выражениях (5.6) и (5.7)

,

,

,

Подстановка полученных значений в формулы (5.6) и (5.7) приводит к системе линейных алгебраических уравнений относительно искомых коэффициентов и

|  |  |
| --- | --- |
|  | (5.8) |

### Процедура ансамблирования конечных элементов

Рассматривается композиция из четырех конечных элементов, для каждого из которых записывается собственная система уравнений вида (5.8):

 В итоге получена система восьми алгебраических уравнений с одиннадцатью неизвестными. Для замыкания системы уравнений следует добавить три уравнения теплового баланса

|  |  |
| --- | --- |
|  | (5.9) |

 Отметим, что внутренние переменные , можно исключить из системы уравнений, складывая уравнения попарно и используя равенства (14.10). После всех преобразований, получится система из пяти алгебраических уравнений относительно неизвестных

В матричной форме эта система уравнений имеет вид

|  |  |
| --- | --- |
| 2021-02-01_17-15-25.png | (5.10) |

Следует вернуться к вопросу о корректности постановки задачи (5.1)– (5.2). Неоднородная система линейных алгебраических уравнений (5.8)

имеет определитель, равный нулю. Также равен нулю определитель системы алгебраических уравнений (5.10). Покомпонентное сложение обоих уравнений последней системы приводит к выражению

|  |  |
| --- | --- |
| , | (5.11) |

являющемуся условием баланса тепла в отдельном конечном элементе: количество тепла , выделившееся за счет внутренних источников, должно быть выведено из него за счет тепловых потоков и с торцов. Это становится очевидным, если вспомнить, что решается стационарное уравнение теплопроводности, решением которого является температурное поле, установившееся за бесконечно большой промежуток времени. Невыполнение балансового соотношения (5.11) приведет либо к накоплению тепла в стержне (при ) и, следовательно, к бесконечно высоким температурам, либо – к принудительному отводу тепла из стержня (при ) и, соответственно, к бесконечно низким температурам. При точном выполнении соотношения (5.11) стержень будет находиться в состоянии термического равновесия при любом распределении температуры, удовлетворяющем уравнению (5.10). Это означает, что решение оказывается не единственным, то есть исходная задача сформулирована некорректно. Это следует из уравнений (5.1)–(5.2), которые определяют решение задачи с точностью до постоянной величины. Вырожденность системы уравнений (5.8) на уровне отдельного конечного элемента приводит к вырожденности системы алгебраических уравнений (5.10) для всего ансамбля конечных элементов. В этом случае суммирование всех уравнений системы (5.10) также приводит к балансовому соотношению. Несмотря на некорректность постановки задачи (5.1)–(5.2), рассмотренный порядок формирования разрешающих соотношений является верным и используется для построения решения прикладных задач. Для корректной постановки задачи следует изменить граничные условия. Пусть на левом конце стержня поддерживается постоянная температура . Для учета этого граничного условия к полученной системе (5.10) следует добавить уравнение (коэффициент , как это уже отмечалось ранее, аппроксимирует значение искомой температуры в первом узле) и считать поток тепла на левом конце стержня неизвестным. В этом случае получается система шести уравнений с неизвестными , имеющая ненулевой определитель

На практике уравнение, содержащее неизвестный поток тепла , как правило, исключается из системы уравнений,

В дальнейшем, после определения всех узловых температур , исключенное из системы уравнение

может быть использовано для определения теплового потока

.

В матричной форме преобразованная система уравнений имеет вид



При решении прикладных задач на границе рассматриваемой области могут быть заданы условия конвективного теплообмена, когда, например, на правом конце стержня тепловой поток ) , где – коэффициент теплоотдачи с поверхности в окружающую среду с температурой , то есть имеет место граничное условие третьего рода,

.

 Для включения этого граничного условия в полученную систему уравнений выполняется замена в последнем уравнении с учетом того, что ,

В результате всех преобразований система линейных алгебраических уравнений принимает вид



# Практическая часть

## Этапы разработки программы

Процесс разработки программы был разбит на следующие этапы:

1. Разработка интерфейса программы с помощью графической библиотеки Pygame[5].
2. Создание структуры классов геометрических фигур, необходимых для построения конструкций.
3. Создание базы данных с помощью SQLite3 для хранения информации о построенных конструкциях, и связь ее с программой[7].
4. Создание таблицы в базе данных, предназначенной для хранения характеристик материалов.
5. Создание и реализация алгоритма для разбиения созданной конструкции на конечные элементы.
6. Разработка и реализация алгоритма для решения задач стационарной теплопроводности в балочных конструкциях.
7. Реализация возможности графического отображения результатов, произведенных вычислений.

### Разработка интерфейса программы

 Для удобного использования программы необходимо было создать такие объекты, как меню, рабочее поле и окно подсказок. В процессе разработки интерфейса все эти объекты были созданы и связаны следующим образом (рис.9):



Рис.9.

* Class Window – класс, предназначенный для создания всего интерфейса (меню, рабочего поля и окна подсказок).
* Class Field – рабочая область (поле), именно здесь будет происходить создание конструкций, разбиение их на конечные элементы и отображение результатов. Внутри этого класса создается объект класса Cursor. Class Cursor – класс, предназначенный для отслеживания координат курсора на экране программы.
* Class Menu – класс, который объединяет в себе все кнопки, необходимые для обеспечения доступа к различным функциям программы. Одним из атрибутов класса Menu является список объектов класса Tab (вкладок), каждый из которых в свою очередь объединяет несколько кнопок, отвечающих за функции, объединенные темой вкладки.
* Class ShowHint – класс, позволяющий выводить на экран информацию о объектах, на которые пользователь наводит курсор.

### Создание структуры классов геометрических фигур

Для того чтобы пользователь мог построить с помощью программы различные конструкции, необходимо было создать структуру классов геометрических фигур, представленную на рисунке (рис.10).



Рис.10.

* Class Geometry – класс, атрибутом которого является словарь, содержащий списки геометрических тел. У класса Geometry есть метод, подсвечивающий геометрические объекты при наведении на них курсора.
* Class Point, Class Line, Class Area – классы точек, линий и многоугольников.

### Создание базы данных

Для хранения информации о построенных объектах и конечных элементах была создана база данных с помощью SQLite3. Структура базы данных представлена на рисунке (рис.11)



Рис.11.

Для связи базы данных с программой был создан класс Database, методы которого служат для организации доступа к данным, хранящимся в базе данных.

### Разработка и реализация алгоритма для решения задач стационарной теплопроводности в балочных конструкциях

Был разработан следующий алгоритм для решения задач стационарной теплопроводности в балочных конструкциях (рис. 12):



Рис.12.

Решение системы ленейных алгебраических уравнений (СЛАУ) в первой версии программы было написано вручную с помощью метода миноров, но данный метод оказался нерациональным, так как решение системы с 10-ю неизвестными занимало примерно 5 минут. Во второй версии решение происходило так же с помощью метода миноров, но определители вычислялись с помощью библиотеки NumPy[6]. В третьей и последней на данный момент версии решение производится с помощью метода Крамера и с использованием библиотеки NumPy. Последняя версия программы производит рассчеты системы с 75-ю неизвестными за 1 минуту.

## Полная структура программы

 Полная помодульная структура программы представлена на рисунке (рис. 13)



Рис.13.

## Демонстрация работы программы

При запуске программы открывается окно: в верхней части находится меню, средняя часть – рабочее пространство, нижняя – область подсказок (здесь также отображаются координаты курсора) (рис. 14).



Рис. 14.

Теперь можно начать создавать конструкцию: чтобы создать линию, необходимо создать две точки (нажав кнопку «Точка») и после этого соединить их, предварительно нажав кнопку «Линия» (рис.15).



Рис.15.

Таким образом можно создать конструкцию, при этом для создания смежной линии нужно использовать одну уже созданную точку, а вторую необходимо создать.

Следующий шаг – разбиение балок на конечные элементы. Для этого необходимо перейти во вкладку «КЭ» и нажать кнопку «Разбить линию», затем нажать на нужную линию (рис. 16). В консоли появится сообщение «Введите количество элементов», после чего надо ввести количество конечных элементов, на которое следует разбить данную линию (рис. 17).



Рис.16.



Рис.17.

Теперь нужно выбрать материал для каждой балки. Для этого во вкладке «КЭ» надо нажать кнопку «Выбрать материал», а затем нажать на необходимую линию. После этого в консоли появится сообщение «Введите id материала», надо ввести id необходимого материала (все id хранятся в базе данных).

Далее надо задать температуру одного узла. В текущей версии программы можно задавать только один узел с известной температурой, также можно задавать температуры только на границах линий. В ходе дальнейшей работы над программой планируется исправить данные недостатки. Для задания температуры необходимо нажать на кнопку «Задать температуру» во вкладке «МКЭ», затем нажать на узел, и ввести в консоли температуру (рис. 18).



Рис.18.

Последний шаг – необходимо нажать на кнопку «Температурный МКЭ» во вкладке «МКЭ» и ввести три параметра (коэффициент теплоотдачи с поверхности в окружающую среду, температуру окружающей среды, мощность внутренних тепловых источников). Затем программа выдаст результаты: на экране появится графическое отображение результатов, а в консоли списки узловых температур для каждой из балок (рис.19).



Рис.19.

# Заключение

В процессе учебно-исследовательской работы в связи с поставленными целями и задачами была изучена теория метода конечных элементов, в том числе метод взвешенных невязок, метод моментов, процедура аппроксимации кусочно-линейными функциями, метод Галеркина. Для разработки программы был изучен язык программирования Python, а также библиотеки Pygame и Numpy и модуль SQLite3.

С помощью всех вышеперечисленных инструментов была разработана программа для решения задач стационарной теплопроводности в балочных конструкциях.

В дальнейшей работе над данным проектом планируется расширить возможности уже функционирующей части программы, а также применить все изученные материалы для разработки алгоритмов для решения задач стационарной теплопроводности в двумерных объектах.

# Список литературы

1. Коннор Дж., Бреббиа К. Метод конечных элементов в механике жидкости. Пер. с англ. - Л., 1979, 264 с.
2. Методы вычислительной математики: учеб. пособие / М.Г. Бояршинов. – Пермь: Изд-во Перм. гос. техн. ун-та, 2008. – 421 с.
3. Основы метода конечных элементов : Введение. Расчет стержневых систем : конспект лекций / Владим. гос. ун-т ; сост. Л.Е. Кондратьева. – Владимир : Изд-во Владим. гос. ун-та, 2007. – 36 с.
4. Лекции по методу конечных элементов: Учебное пособие. - М.: Издательский отдел факультета ВМиК МГУ им. М.В. Ломоносова (лицензия ИД N 05899 от 24.09.2001 г.); МАКС Пресс, 2010. - 2-е изд., испр. и доп. - 264 с.
5. <https://www.pygame.org/docs/>
6. <https://numpy.org/doc/stable/reference/>
7. <https://sqlite.org/docs.html>

# Приложение

1. Главный файл программы – main.py

from modules.interface import \*

from modules.functions import \*

from modules.database import \*

from modules.elements import \*

window = Window()

geometry = Geometry()

finit\_elements = FinitElements()

cursor = window.field.get\_cursor()

current\_line = None

current\_id\_lines = []

current\_area = {}

hint = None

running = True

while running:

 for event in pygame.event.get():

 if event.type == pygame.QUIT:

 running = False

 if event.type == pygame.MOUSEMOTION:

 x, y = event.pos

 cursor.move(x, y)

 # print(cursor.get\_coords())

 geometry\_object = geometry.highlight\_object(cursor.get\_coords())

 window.field.draw(window.screen)

 window.show\_hint.draw(window.screen, cursor, geometry\_object, hint)

 # Добавить информацию об объекте

 if event.type == pygame.MOUSEBUTTONDOWN:

 pressed\_tab = find\_pressed\_button(window.menu.tabs)

 if check\_buttons(event.pos, window.menu.tabs):

 for tab in window.menu.tabs:

 tab.press(event.pos)

 elif check\_buttons(event.pos, pressed\_tab.buttons):

 for button in pressed\_tab.buttons:

 button.press(event.pos)

 pressed\_button = find\_pressed\_button(pressed\_tab.buttons)

 button\_point = window.menu.tabs[0].buttons[0]

 button\_line = window.menu.tabs[0].buttons[1]

 button\_area = window.menu.tabs[0].buttons[2]

 button\_mesh\_line = window.menu.tabs[1].buttons[0]

 button\_choose\_material = window.menu.tabs[1].buttons[2]

 button\_temperature = window.menu.tabs[2].buttons[0]

 button\_set\_temperature = window.menu.tabs[2].buttons[1]

 # create point

 if pressed\_button == button\_point:

 coords = cursor.get\_coords()

 geometry.add\_point(coords)

 # create line

 if pressed\_button == button\_line:

 coords = cursor.get\_coords()

 x, y = coords

 if x is not None and y is not None:

 if check\_point(coords, geometry):

 if not current\_line:

 current\_line = [find\_point(coords, geometry)]

 hint = ["Для завершения линии отметьте еще одну точку",

 f"Начало линии в точке c координатами {str(current\_line[0])}"]

 else:

 current\_line.append(find\_point(coords, geometry))

 coords1, coords2 = current\_line

 coords1, coords2 = current\_line

 x1, y1 = coords1

 x2, y2 = coords2

 id1 = DATABASE.get\_id\_point(x1, y1)

 id2 = DATABASE.get\_id\_point(x2, y2)

 geometry.add\_line(id1, id2)

 current\_line = None

 hint = None

 # create area

 if pressed\_button == button\_area:

 coords = cursor.get\_coords()

 x, y = coords

 if x is not None and y is not None:

 if check\_line(coords, geometry):

 line = find\_line(coords, geometry)

 current\_id\_lines.append(line.id)

 id\_points = DATABASE.get\_id\_points\_from\_line(line.id)

 hint = [f"Для завершения площади отметьте еще несколько линий до замкнутого контура.",

 f" Отмеченные линии: {', '.join(map(str, current\_id\_lines))}"]

 for id\_point in id\_points:

 if id\_point not in current\_area:

 current\_area[id\_point] = False

 else:

 current\_area[id\_point] = True

 if all(current\_area.values()):

 geometry.add\_area(current\_id\_lines)

 current\_area = {}

 current\_id\_lines = []

 hint = None

 if pressed\_button == button\_mesh\_line:

 coords = cursor.get\_coords()

 x, y = coords

 if x is not None and y is not None:

 if check\_line(coords, geometry):

 line = find\_line(coords, geometry)

 line.set\_color(GREEN)

 print('Введите количество элементов')

 count\_of\_elements = int(input())

 # dialog\_window = DialogWindow()

 # count\_of\_nodes = dialog\_window.get\_value()

 coords\_of\_elements = mesh\_line(line, count\_of\_elements)

 list\_of\_nodes = []

 list\_of\_finit\_lines = []

 for coords in coords\_of\_elements:

 x, y = coords

 finit\_elements.add\_node(coords)

 list\_of\_nodes.append(finit\_elements.get\_node(coords))

 for i in range(len(list\_of\_nodes) - 1):

 DATABASE.add\_finit\_line\_into\_db(list\_of\_nodes[i].id, list\_of\_nodes[i + 1].id)

 finitline = FinitLine([list\_of\_nodes[i], list\_of\_nodes[i + 1]])

 list\_of\_finit\_lines.append(finitline)

 line.set\_finit\_lines(list\_of\_finit\_lines)

 if line.material\_id:

 for finit\_line in list\_of\_finit\_lines:

 id\_finit\_line = DATABASE.get\_id\_finit\_line(finit\_line.node1.id, finit\_line.node2.id)

 DATABASE.set\_material\_of\_finit\_line(id\_finit\_line, line.material\_id)

 if pressed\_button == button\_choose\_material:

 coords = cursor.get\_coords()

 x, y = coords

 if x is not None and y is not None:

 if check\_line(coords, geometry):

 line = find\_line(coords, geometry)

 print('Введите id материала')

 material\_id = int(input())

 line.set\_material(material\_id)

 if pressed\_button == button\_set\_temperature:

 coords = cursor.get\_coords()

 x, y = coords

 if x is not None and y is not None:

 if check\_node(coords, finit\_elements):

 node = find\_node(coords, finit\_elements)

 print('Введите температуру')

 temperature = float(input())

 DATABASE.set\_temperature(node.id, temperature)

 if pressed\_button == button\_temperature:

 print('Введите коэффициент теплоотдачи с поверхности в окружающую среду')

 heat\_transfer = float(input())

 print('Введите температуру окружающейй среды')

 environment\_t = float(input())

 print('Введите мощность внутренних тепловых источников')

 heat\_source\_power = float(input())

 mke(geometry.get\_geometry\_objects()['lines'], heat\_transfer, environment\_t, heat\_source\_power)

 window.menu.draw(window.screen)

 window.field.set\_geometry\_objects(geometry.get\_geometry\_objects())

 window.field.set\_finit\_elements(finit\_elements.get\_finit\_elements())

 window.field.draw(window.screen)

 if event.type == pygame.KEYDOWN:

 if event.key == pygame.K\_ESCAPE:

 current\_line = None

 current\_area = None

 geometry.get\_geometry\_objects()

 pygame.display.flip()

# geometry.clear\_database()

DATABASE.close()

pygame.quit()

1. Модуль interface

import pygame

from modules.constants import \*

from modules.geometry import \*

from modules.functions import \*

class Window:

 def \_\_init\_\_(self):

 # initialization of screen from pygame

 pygame.init()

 self.screen = pygame.display.set\_mode(SIZE)

 self.screen.fill(BLACK)

 self.init\_ui()

 def init\_ui(self):

 # initialization of objects for window

 y\_coords = split\_line(HEIGHT, 1, 7, 2)

 # Create Menu

 self.menu = Menu((0, 0), (WIDTH, int(y\_coords[0])))

 tab\_geometry = Tab(self.screen, 'Геометрия')

 button\_point = Button(self.screen, 'Точка')

 button\_line = Button(self.screen, 'Линия')

 button\_area = Button(self.screen, 'Площадь')

 tab\_ke = Tab(self.screen, 'КЭ')

 button\_mesh\_line = Button(self.screen, 'Разбить линию')

 button\_mesh\_area = Button(self.screen, 'Разбить площадь')

 button\_choose\_material = Button(self.screen, 'Выбрать материал')

 tab\_mke = Tab(self.screen, 'МКЭ')

 button\_temperature = Button(self.screen, 'Температурный МКЭ')

 button\_set\_temperature = Button(self.screen, 'Задать температуру')

 button\_test23 = Button(self.screen, 'Тестовая кнопка 3')

 button\_test24 = Button(self.screen, 'Тестовая кнопка 4')

 self.menu.add\_tab(tab\_geometry)

 self.menu.add\_button(tab\_geometry, button\_point)

 self.menu.add\_button(tab\_geometry, button\_line)

 self.menu.add\_button(tab\_geometry, button\_area)

 self.menu.add\_tab(tab\_ke)

 self.menu.add\_button(tab\_ke, button\_mesh\_line)

 self.menu.add\_button(tab\_ke, button\_mesh\_area)

 self.menu.add\_button(tab\_ke, button\_choose\_material)

 self.menu.add\_tab(tab\_mke)

 self.menu.add\_button(tab\_mke, button\_temperature)

 self.menu.add\_button(tab\_mke, button\_set\_temperature)

 self.menu.add\_button(tab\_mke, button\_test23)

 self.menu.add\_button(tab\_mke, button\_test24)

 self.menu.tabs[0].state = 'pressed'

 self.menu.tabs[0].buttons[0].state = 'pressed'

 self.menu.draw(self.screen)

 # Create Field

 self.field = Field((0, int(y\_coords[0])), (WIDTH, int(y\_coords[1])))

 self.field.draw(self.screen)

 # Create ShowHint

 self.show\_hint = ShowHint((0, int(y\_coords[1])), (WIDTH, int(y\_coords[2])))

class DialogWindow:

 def \_\_init\_\_(self):

 self.screen = pygame.display.set\_mode((200, 200))

 self.screen.fill(YELLOWGREEN)

 self.init\_ui()

 def init\_ui(self):

 pass

class Menu:

 def \_\_init\_\_(self, coords1, coords2):

 self.x, self.y = coords1

 x2, y2 = coords2

 self.width = x2 - self.x

 self.height = y2 - self.y

 self.tabs\_height = self.height \* 0.4

 self.lenghts\_of\_tabs = []

 self.tabs = []

 def add\_tab(self, tab):

 self.tabs.append(tab)

 if len(self.tabs) == 0:

 tab.x, tab.y = self.x, self.y

 else:

 tab.x = sum(self.lenghts\_of\_tabs) + self.x

 tab.y = self.y

 self.lenghts\_of\_tabs.append(tab.width)

 def add\_button(self, tab, button):

 tab.add\_button(button)

 def draw(self, display):

 pygame.draw.rect(display, WHITE, (self.x, self.y, self.width, self.height))

 quantity\_of\_tabs = len(self.tabs)

 height\_of\_tabs = self.tabs\_height

 for i in range(quantity\_of\_tabs):

 tab = self.tabs[i]

 tab.draw(sum(self.lenghts\_of\_tabs[:i]) + 10 \* i, 0, height\_of\_tabs)

 for tab in self.tabs:

 if tab.state == 'pressed':

 quantity\_of\_buttons = len(tab.buttons)

 height\_of\_button = self.height - self.tabs\_height

 for i in range(quantity\_of\_buttons):

 button = tab.buttons[i]

 button.draw(sum(tab.lenghts\_of\_buttons[:i]) + 10 \* i, self.y + self.tabs\_height, height\_of\_button)

class Button:

 def \_\_init\_\_(self, display, text):

 self.display = display

 self.text = text

 self.state = 'unpressed'

 font\_size = 20

 self.font = pygame.font.SysFont('serif', font\_size)

 self.width, self.height = self.font.size(text)

 self.x, self.y = None, None

 def press(self, press\_pos):

 x\_press, y\_press = press\_pos

 if self.x + self.width >= x\_press >= self.x and self.y + self.height >= y\_press >= self.y:

 self.state = 'pressed'

 else:

 self.state = 'unpressed'

 def draw(self, x, y, height):

 self.x, self.y = x, y

 if self.state == 'unpressed':

 self.display.fill(WHITE, (x, y, self.width + 10, height))

 if self.state == 'pressed':

 self.display.fill(YELLOWGREEN, (x, y, self.width + 10, height))

 pygame.draw.rect(self.display, GREY, (x, y, self.width + 10, height), 1)

 text = self.font.render(self.text, 1, BLACK)

 self.display.blit(text, (x + 5, y + self.height \* 0.01))

class Tab(Button):

 def \_\_init\_\_(self, display, text):

 super().\_\_init\_\_(display, text)

 self.buttons = []

 self.lenghts\_of\_buttons = []

 def add\_button(self, button):

 self.buttons.append(button)

 if len(self.buttons) == 0:

 button.x, button.y = self.x, self.y + self.height

 else:

 button.x = sum(self.lenghts\_of\_buttons) + self.x

 button.y = self.y + self.height

 self.lenghts\_of\_buttons.append(button.width)

 def press(self, press\_pos):

 x\_press, y\_press = press\_pos

 if self.x + self.width >= x\_press >= self.x and self.y + self.height >= y\_press >= self.y:

 self.state = 'pressed'

 self.buttons[0].state = 'pressed'

 else:

 self.state = 'unpressed'

 for button in self.buttons:

 button.state = 'unpressed'

class Field:

 def \_\_init\_\_(self, coords1, coords2):

 self.x, self.y = coords1

 x2, y2 = coords2

 self.width = x2 - self.x

 self.height = y2 - self.y

 self.scale = 1

 self.grid\_size = 20

 self.cursor = Cursor(self)

 self.geometry\_objects = {}

 self.finit\_elements = {}

 def draw(self, display):

 display.fill(WHITE, (self.x, self.y, self.width, self.height))

 for i in range(self.height // (self.scale \* self.grid\_size)):

 point1 = self.x, self.y + self.scale \* self.grid\_size \* i

 point2 = self.x + self.width, self.y + self.scale \* self.grid\_size \* i

 pygame.draw.line(display, VERYLIGHTBLUE, point1, point2, 1)

 for i in range(self.width // (self.scale \* self.grid\_size)):

 point1 = self.x + self.scale \* self.grid\_size \* i, self.y

 point2 = self.x + self.scale \* self.grid\_size \* i, self.y + self.height

 pygame.draw.line(display, VERYLIGHTBLUE, point1, point2, 1)

 for list\_of\_geometry\_objects in self.geometry\_objects.values():

 for geometry\_object in list\_of\_geometry\_objects:

 geometry\_object.draw(display, self)

 if isinstance(geometry\_object, Line):

 geometry\_object.draw\_temperature\_mke(display, self)

 for list\_of\_finit\_elements in self.finit\_elements.values():

 for finit\_element in list\_of\_finit\_elements:

 finit\_element.draw(display, self)

 def set\_geometry\_objects(self, geometry\_objects):

 self.geometry\_objects = geometry\_objects

 def set\_finit\_elements(self, finit\_elements):

 self.finit\_elements = finit\_elements

 def get\_cursor(self):

 return self.cursor

class Cursor:

 def \_\_init\_\_(self, field):

 self.x, self.y = None, None

 self.field = field

 def move(self, x, y):

 field = self.field

 if field.x <= x <= field.x + field.width and field.y <= y <= field.y + field.height:

 self.x = abs(x - field.x)

 self.y = abs(y - field.height - field.y)

 else:

 self.x, self.y = None, None

 def get\_coords(self):

 return self.x, self.y

class ShowHint:

 def \_\_init\_\_(self, coords1, coords2):

 self.x, self.y = coords1

 x, y = coords2

 self.width = x - self.x

 self.height = y - self.y

 font\_size = 20

 self.font = pygame.font.SysFont('serif', font\_size)

 def draw(self, display, cursor, geometry\_object, hint):

 display.fill(WHITE, (self.x, self.y, self.width, self.height))

 pygame.draw.line(display, VERYLIGHTBLUE, (self.x, self.y), (self.x + self.width, self.y))

 font\_size = 15

 self.font = pygame.font.SysFont('serif', font\_size)

 x, y = cursor.x, cursor.y

 if x is None:

 x = ' '

 if y is None:

 y = ' '

 width1, height1 = self.font.size(str(x))

 text1 = self.font.render(str(x), 1, BLACK)

 width2, height2 = self.font.size(str(y))

 text2 = self.font.render(str(y), 1, BLACK)

 pygame.draw.rect(display, GREY, (self.x, self.y, width1 + 10, height1 + 10), 1)

 pygame.draw.rect(display, GREY, (self.x + width1 + 10, self.y, width2 + 10, height2 + 10), 1)

 display.blit(text1, (self.x + 5, self.y + 5))

 display.blit(text2, (self.x + width1 + 15, self.y + 5))

 if str(geometry\_object) == "None":

 geometry\_object\_str = ' '

 else:

 geometry\_object\_str = str(geometry\_object)

 text3 = self.font.render(geometry\_object\_str, 1, BLACK)

 width3, height3 = self.font.size(geometry\_object\_str)

 display.blit(text3, (self.x + 5, self.y + height1 + 15))

 pygame.draw.rect(display, GREY, (self.x, self.y + height1 + 10, width3 + 10, height3 + 10), 1)

 if hint:

 text4\_str = hint[0]

 text5\_str = hint[1]

 else:

 text4\_str = ' '

 text5\_str = ' '

 text4 = self.font.render(text4\_str, 1, BLACK)

 width4, height4 = self.font.size(text4\_str)

 display.blit(text4, (self.x + 5, self.y + height1 + height3 + 25))

 pygame.draw.rect(display, GREY, (self.x, self.y + height1 + height3 + 20, width4 + 10, height4 + 10), 1)

 text5 = self.font.render(text5\_str, 1, BLACK)

 width5, height5 = self.font.size(text5\_str)

 display.blit(text5, (self.x + 5, self.y + height1 + height3 + height4 + 35))

 pygame.draw.rect(display, GREY, (self.x, self.y + height1 + height3 + height4 + 30, width5 + 10, height5 + 10), 1)

1. Модуль geometry

import pygame

from modules.constants import \*

from modules.database import \*

from modules.matrix import \*

from modules.functions import \*

def check\_point(coords, coords\_of\_points):

 x, y = coords

 result = False

 for coord\_of\_point in coords\_of\_points:

 if coord\_of\_point[0] - 5 <= x <= coord\_of\_point[0] + 5:

 if coord\_of\_point[1] - 5 <= y <= coord\_of\_point[1] + 5:

 result = True

 return result

def check\_line\_to\_add\_into\_db(point\_id1, point\_id2, geometry):

 result = True

 point1 = min([point\_id1, point\_id2])

 point2 = max([point\_id1, point\_id2])

 for line in geometry.geometry\_objects['lines']:

 points\_from\_line = DATABASE.get\_id\_points\_from\_line(line.id)

 if points\_from\_line[0] == point1 and points\_from\_line[1] == point2:

 result = False

 return result

class Geometry:

 def \_\_init\_\_(self):

 self.geometry\_objects = {'points': [],

 'lines': [],

 'areas': []}

 def add\_point(self, coords):

 # добавляем точку в словарь, если это возможно\*

 x, y = coords

 coords\_of\_points = []

 for point in self.geometry\_objects['points']:

 coords\_of\_points.append((point.x, point.y))

 if x is not None or y is not None:

 if not check\_point(coords, coords\_of\_points):

 DATABASE.add\_point\_into\_db(x, y)

 id = DATABASE.get\_id\_point(x, y)

 self.geometry\_objects['points'].append(Point(id))

 def add\_line(self, point\_id1, point\_id2):

 x1, y1 = DATABASE.get\_coords\_point(point\_id1)

 x2, y2 = DATABASE.get\_coords\_point(point\_id2)

 if x1 != x2 or y1 != y2:

 if check\_line\_to\_add\_into\_db(point\_id1, point\_id2, self):

 DATABASE.add\_line\_into\_db(point\_id1, point\_id2)

 id = DATABASE.get\_id\_line(point\_id1, point\_id2)

 self.geometry\_objects['lines'].append(Line(id))

 def add\_area(self, list\_of\_lines\_id):

 DATABASE.add\_area\_into\_db(list\_of\_lines\_id)

 id = DATABASE.get\_id\_area(list\_of\_lines\_id)

 self.geometry\_objects['areas'].append(Area(id))

 def highlight\_object(self, event\_pos):

 x\_cursor, y\_cursor = event\_pos

 if x\_cursor is not None or y\_cursor is not None:

 for list\_of\_geometry\_objects in self.geometry\_objects.values():

 for geometry\_object in list\_of\_geometry\_objects:

 if isinstance(geometry\_object, Point):

 x\_go, y\_go = geometry\_object.get\_coords()

 if x\_go - 5 <= x\_cursor <= x\_go + 5 and y\_go - 5 <= y\_cursor <= y\_go + 5:

 geometry\_object.set\_color(ORANGE)

 return geometry\_object

 else:

 geometry\_object.set\_color(BLACK)

 if isinstance(geometry\_object, Line):

 x\_go, y\_go = geometry\_object.get\_coords()

 x1, y1 = x\_go

 x2, y2 = y\_go

 displacement = 5

 if x2 - x1 == 0:

 if ( x2 + displacement >= x\_cursor >= x2 - displacement and

 max(y1, y2) >= y\_cursor >= min(y1, y2) ):

 geometry\_object.set\_color(ORANGE)

 return geometry\_object

 else:

 geometry\_object.set\_color(BLACK)

 elif y2 - y1 == 0:

 if ( y2 + displacement >= y\_cursor >= y2 - displacement and

 max(x1, x2) >= x\_cursor >= min(x1, x2) ):

 geometry\_object.set\_color(ORANGE)

 return geometry\_object

 else:

 geometry\_object.set\_color(BLACK)

 else:

 k = (y2 - y1) / (x2 - x1)

 b = y1 - x1 \* k

 displacement \*= (k \*\* 2 + 1) \*\* 0.5

 k\_orto = -1 / k

 if y1 == max(y1, y2):

 y\_up, x\_up = y1, x1

 y\_down, x\_down = y2, x2

 else:

 y\_up, x\_up = y2, x2

 y\_down, x\_down = y1, x1

 b\_max = y\_up - x\_up \* k\_orto

 b\_min = y\_down - x\_down \* k\_orto

 if ( y\_cursor <= x\_cursor \* k + b + displacement and

 y\_cursor >= x\_cursor \* k + b - displacement and

 y\_cursor <= x\_cursor \* k\_orto + b\_max and

 y\_cursor >= x\_cursor \* k\_orto + b\_min):

 geometry\_object.set\_color(ORANGE)

 return geometry\_object

 else:

 geometry\_object.set\_color(BLACK)

 else:

 geometry\_object.set\_initial\_color()

 return None

 def get\_geometry\_objects(self):

 return self.geometry\_objects

class Point:

 def \_\_init\_\_(self, id):

 self.id = id

 self.x, self.y = DATABASE.get\_coords\_point(id)

 self.color = DATABASE.get\_color\_point(id)

 def draw(self, display, field):

 x = self.x + field.x

 y = -self.y + field.height + field.y

 pygame.draw.circle(display, self.color, (x, y), 2)

 def set\_color(self, color):

 self.color = color

 def set\_initial\_color(self):

 self.color = DATABASE.get\_color\_point(self.id)

 def get\_color(self):

 return self.color

 def get\_coords(self):

 return self.x, self.y

 def \_\_str\_\_(self):

 return f"Точка {self.id}"

class Line:

 def \_\_init\_\_(self, id):

 self.id = id

 self.coords\_start\_point = DATABASE.get\_coords\_line(id)[0]

 self.coords\_finish\_point = DATABASE.get\_coords\_line(id)[1]

 self.color = DATABASE.get\_color\_line(id)

 self.material\_id = None

 self.list\_of\_nodes = []

 def draw(self, display, field):

 x1, y1 = self.coords\_start\_point

 x1 = x1 + field.x

 y1 = -y1 + field.height + field.y

 x2, y2 = self.coords\_finish\_point

 x2 = x2 + field.x

 y2 = -y2 + field.height + field.y

 pygame.draw.line(display, self.color, (x1, y1), (x2, y2), 1)

 def set\_finit\_lines(self, list\_of\_finit\_lines):

 list\_of\_id\_finit\_lines = []

 for finit\_line in list\_of\_finit\_lines:

 id\_finit\_line = DATABASE.get\_id\_finit\_line(finit\_line.node1.id, finit\_line.node2.id)

 list\_of\_id\_finit\_lines.append(id\_finit\_line)

 DATABASE.set\_finit\_lines(self.id, list\_of\_id\_finit\_lines)

 def set\_material(self, material\_id):

 list\_of\_id\_finit\_lines = DATABASE.get\_list\_of\_id\_finit\_lines(self.id)

 if list\_of\_id\_finit\_lines:

 for id\_finit\_line in list\_of\_id\_finit\_lines:

 DATABASE.set\_material\_of\_finit\_line(id\_finit\_line, material\_id)

 def set\_color(self, color):

 self.color = color

 def set\_initial\_color(self):

 self.color = DATABASE.get\_color\_line(self.id)

 def get\_coords(self):

 return self.coords\_start\_point, self.coords\_finish\_point

 def \_\_str\_\_(self):

 return f"Линия {self.id}"

 def add\_nodes(self, list\_of\_nodes):

 for node in list\_of\_nodes:

 self.list\_of\_nodes.append(node)

 def temperature\_mke(self, heat\_transfer, environment\_t, heat\_source\_power):

 print('начали')

 results = []

 list\_of\_transcalency = []

 for id\_finit\_line in DATABASE.get\_list\_of\_id\_finit\_lines(self.id):

 transcalency = DATABASE.get\_transcalency\_of\_finit\_line(id\_finit\_line)

 list\_of\_transcalency.append(transcalency)

 matrix = TemperatureMatrix(len(list\_of\_transcalency) + 1)

 matrix.fill(self.id, heat\_transfer, environment\_t, heat\_source\_power, 1)

 print(matrix.determinant)

 if matrix.determinant != 0:

 results = matrix.find\_unknown()

 for i in range(len(DATABASE.get\_id\_nodes\_of\_line(self.id))):

 DATABASE.set\_temperature(DATABASE.get\_id\_nodes\_of\_line(self.id)[i], results[i])

 print('----------')

 print(results)

 print('----------')

 def draw\_temperature\_mke(self, display, field):

 list\_of\_coords\_chart = []

 list\_of\_coords\_local = []

 run = True

 if DATABASE.get\_list\_of\_id\_finit\_lines(self.id) is None:

 run = False

 else:

 for node\_id in DATABASE.get\_id\_nodes\_of\_line(self.id):

 if DATABASE.get\_temperature(node\_id) == 0:

 run = False

 break

 if run:

 x1, y1 = self.coords\_start\_point

 x2, y2 = self.coords\_finish\_point

 lenght = ((x1 - x2) \*\* 2 + (y1 - y2) \*\* 2) \*\* 0.5

 x\_pos1 = min(x1, x2)

 x\_pos2 = max(x1, x2)

 if x\_pos1 == x1:

 y\_pos1 = y1

 y\_pos2 = y2

 else:

 y\_pos1 = y2

 y\_pos2 = y1

 sinus\_of\_line\_slope = abs(y1 - y2) / lenght

 cosinus\_of\_line\_slope = (1 - sinus\_of\_line\_slope \*\* 2) \*\* 0.5

 for node\_id in DATABASE.get\_id\_nodes\_of\_line(self.id):

 temperature\_draw = DATABASE.get\_temperature(node\_id) / 10

 x\_1, y\_1 = DATABASE.get\_coords\_node(node\_id)

 x\_1 = x\_1 + field.x

 y\_1 = -y\_1 + field.height + field.y

 if y\_pos1 < y\_pos2:

 x\_2 = x\_1 - temperature\_draw \* sinus\_of\_line\_slope

 else:

 x\_2 = x\_1 + temperature\_draw \* sinus\_of\_line\_slope

 y\_2 = y\_1 - temperature\_draw \* cosinus\_of\_line\_slope

 pygame.draw.line(display, BLACK, (x\_1, y\_1), (x\_2, y\_2))

 list\_of\_coords\_chart.append((x\_2, y\_2))

 list\_of\_coords\_local.append((x\_1, y\_1))

 '''

 for i in range(len(list\_of\_coords\_chart) - 1):

 x\_start, y\_start = list\_of\_coords\_chart[i]

 x\_finish, y\_finish = list\_of\_coords\_chart[i + 1]

 pygame.draw.line(display, BLACK, (x\_start, y\_start), (x\_finish, y\_finish))

 '''

 # Color

 temperature\_to\_color = {}

 temperatures = DATABASE.get\_all\_temperatures(field.geometry\_objects['lines'])

 max\_temperature = max(temperatures)

 min\_temperature = min(temperatures)

 step = (max\_temperature - min\_temperature) / 14

 for i in range(14):

 temperature\_to\_color[step \* i + min\_temperature] = LIST\_OF\_COLORS[i]

 number = 0

 for finit\_line in DATABASE.get\_list\_of\_id\_finit\_lines(self.id):

 color = WHITE

 node1, node2 = DATABASE.get\_id\_nodes\_of\_finit\_line(finit\_line)

 temperature = (DATABASE.get\_temperature(node1) + DATABASE.get\_temperature(node2)) / 2

 for i in range(13):

 temperatures\_in\_dict = list(temperature\_to\_color.keys())

 if temperatures\_in\_dict[i] <= temperature <= temperatures\_in\_dict[i + 1]:

 color = temperature\_to\_color[temperatures\_in\_dict[i]]

 break

 coords1 = list\_of\_coords\_local[number]

 coords2 = list\_of\_coords\_local[number + 1]

 coords3 = list\_of\_coords\_chart[number + 1]

 coords4 = list\_of\_coords\_chart[number]

 fill\_polygon(display, color, coords1, coords2, coords3, coords4)

 number += 1

 for i in range(len(list\_of\_coords\_chart) - 1):

 x\_start, y\_start = list\_of\_coords\_chart[i]

 x\_finish, y\_finish = list\_of\_coords\_chart[i + 1]

 pygame.draw.line(display, BLACK, (x\_start, y\_start), (x\_finish, y\_finish))

 for i in range(len(list\_of\_coords\_chart)):

 x\_start, y\_start = list\_of\_coords\_chart[i]

 x\_finish, y\_finish = list\_of\_coords\_local[i]

 pygame.draw.line(display, BLACK, (x\_start, y\_start), (x\_finish, y\_finish))

1. Модуль functions

from modules.database import \*

from modules.matrix import \*

from modules.constants import\*

def split\_line(lenght, \*args):

 """

 Если аргумент один, то разбивается на равные по длине части

 Если аргументов несколько, то разбиение

 return tuple

 Example

 ---------------

 >>> split\_line(10, 5)

 return (2.0, 4.0, 6.0, 8.0, 10.0)

 >>> split\_line(10, 4)

 return (2.5, 5.0, 7.5, 10.0)

 >>> split\_line(10, 4, 8, 2, 6)

 return (2.0, 6.0, 7.0, 10.0)

 >>> split\_line(4, 8, 15, 16, 23, 42)

 return (0.3076923076923077, 0.8846153846153846, 1.5, 2.3846153846153846, 4.0)

 ---------------

 """

 if len(args) == 1:

 count\_of\_segment = args[0]

 lenght\_of\_segment = lenght / count\_of\_segment

 segments = [lenght\_of\_segment \* i for i in range(1, count\_of\_segment + 1)]

 else:

 count\_of\_segment = len(args)

 lenght\_of\_segments = [lenght \* args[i] / sum(args) for i in range(count\_of\_segment)]

 segments = [sum(lenght\_of\_segments[:i]) for i in range(1, count\_of\_segment + 1)]

 return tuple(segments)

def check\_buttons(coords, buttons):

 x\_pos, y\_pos = coords

 for button in buttons:

 if button.x <= x\_pos <= button.x + button.width:

 if button.y <= y\_pos <= button.y + button.height:

 return True

 else:

 return False

def find\_pressed\_button(buttons):

 for button in buttons:

 if button.state == 'pressed':

 return button

def check\_point(coords, geometry):

 x\_pos, y\_pos = coords

 for point in geometry.get\_geometry\_objects()['points']:

 x, y = point.x, point.y

 if x - 5 <= x\_pos <= x + 5:

 if y - 5 <= y\_pos <= y + 5:

 return True

def find\_point(coords, geometry):

 x\_pos, y\_pos = coords

 for point in geometry.get\_geometry\_objects()['points']:

 x, y = point.x, point.y

 if x - 5 <= x\_pos <= x + 5:

 if y - 5 <= y\_pos <= y + 5:

 coords = point.x, point.y

 return coords

def check\_node(coords, finit\_elements):

 x\_pos, y\_pos = coords

 for node in finit\_elements.get\_finit\_elements()['nodes']:

 x, y = node.x, node.y

 if x - 5 < x\_pos < x + 5:

 if y - 5 < y\_pos < y + 5:

 return True

def exactly\_check\_node(coords, finit\_elements):

 x\_pos, y\_pos = coords

 for node in finit\_elements.get\_finit\_elements()['nodes']:

 x, y = node.x, node.y

 if x == x\_pos:

 if y == y\_pos:

 return True

def find\_node(coords, finit\_elements):

 x\_pos, y\_pos = coords

 for node in finit\_elements.get\_finit\_elements()['nodes']:

 x, y = node.x, node.y

 if x - 5 <= x\_pos <= x + 5:

 if y - 5 <= y\_pos <= y + 5:

 coords = node.x, node.y

 return node

def check\_line(coords, geometry):

 x, y = coords

 result = False

 for line in geometry.get\_geometry\_objects()['lines']:

 x1, y1 = line.coords\_start\_point

 x2, y2 = line.coords\_finish\_point

 displacement = 5

 if x2 - x1 == 0:

 if ( x2 + displacement >= x >= x2 - displacement and

 max(y1, y2) >= y >= min(y1, y2)):

 result = True

 elif y2 - y1 == 0:

 if ( y2 + displacement >= y >= y2 - displacement and

 max(x1, x2) >= x >= min(x1, x2)):

 result = True

 else:

 k = (y2 - y1) / (x2 - x1)

 b = y1 - x1 \* k

 displacement \*= (k \*\* 2 + 1) \*\* 0.5

 k\_orto = -1 / k

 if y1 == max(y1, y2):

 y\_up, x\_up = y1, x1

 y\_down, x\_down = y2, x2

 else:

 y\_up, x\_up = y2, x2

 y\_down, x\_down = y1, x1

 b\_max = y\_up - x\_up \* k\_orto

 b\_min = y\_down - x\_down \* k\_orto

 if ( y <= x \* k + b + displacement and

 y >= x \* k + b - displacement and

 y <= x \* k\_orto + b\_max and

 y >= x \* k\_orto + b\_min):

 result = True

 return result

def find\_line(coords, geometry):

 x, y = coords

 result = None

 for line in geometry.get\_geometry\_objects()['lines']:

 x1, y1 = line.coords\_start\_point

 x2, y2 = line.coords\_finish\_point

 displacement = 5

 if x2 - x1 == 0:

 if ( x2 + displacement >= x >= x2 - displacement and

 max(y1, y2) >= y >= min(y1, y2)):

 result = line

 elif y2 - y1 == 0:

 if ( y2 + displacement >= y >= y2 - displacement and

 max(x1, x2) >= x >= min(x1, x2)):

 result = line

 else:

 k = (y2 - y1) / (x2 - x1)

 b = y1 - x1 \* k

 displacement \*= (k \*\* 2 + 1) \*\* 0.5

 k\_orto = -1 / k

 if y1 == max(y1, y2):

 y\_up, x\_up = y1, x1

 y\_down, x\_down = y2, x2

 else:

 y\_up, x\_up = y2, x2

 y\_down, x\_down = y1, x1

 b\_max = y\_up - x\_up \* k\_orto

 b\_min = y\_down - x\_down \* k\_orto

 if ( y <= x \* k + b + displacement and

 y >= x \* k + b - displacement and

 y <= x \* k\_orto + b\_max and

 y >= x \* k\_orto + b\_min):

 result = line

 return result

def mesh\_line(line, count\_of\_elements):

 x1, y1 = line.coords\_start\_point

 x2, y2 = line.coords\_finish\_point

 line\_lenght = ((x1 - x2) \*\* 2 + (y1 - y2) \*\* 2) \*\* 0.5

 lenghts\_of\_elements = split\_line(line\_lenght, count\_of\_elements)

 coords\_of\_elements = None

 x\_min = min(x1, x2)

 x\_max = max(x1, x2)

 if x\_min == x1:

 y\_x\_min = y1

 y\_x\_max = y2

 else:

 y\_x\_min = y2

 y\_x\_max = y1

 for i in range(len(lenghts\_of\_elements) + 1):

 if not coords\_of\_elements:

 coords\_of\_elements = []

 coords\_of\_elements.append((x1, y1))

 elif i == len(lenghts\_of\_elements):

 coords\_of\_elements.append((x2, y2))

 else:

 lenght = lenghts\_of\_elements[i - 1]

 x = x\_min + lenght \* ((x\_max - x\_min) / line\_lenght)

 cos = (x\_max - x\_min) / line\_lenght

 y = None

 if y\_x\_min <= y\_x\_max:

 y = y\_x\_min + lenght \* (1 - cos \*\* 2) \*\* 0.5

 else:

 y = y\_x\_min - lenght \* (1 - cos \*\* 2) \*\* 0.5

 '''

 if x1 >= x2 and y1 >= y2:

 y = max(y1, y2) - lenght \* (1 - cos \*\* 2) \*\* 0.5

 elif x1 >= x2 and y1 <= y2:

 y = min(y1, y2) + lenght \* (1 - cos \*\* 2) \*\* 0.5

 elif x1 <= x2 and y1 >= y2:

 y = max(y1, y2) - lenght \* (1 - cos \*\* 2) \*\* 0.5

 else:

 y = min(y1, y2) + lenght \* (1 - cos \*\* 2) \*\* 0.5

 '''

 coords\_of\_elements.append((x, y))

 return coords\_of\_elements

'''

x1, y1 x2, y2

y1 = kx1 + d => d = y1 - kx1 => d = y1 - x1(y2 - y1)/(x2 - x1)

y2 = kx2 + d => y2 = k(x2 - x1) + y1 => k = (y2 - y1)/(x2 - x1)

'''

def fill\_polygon(screen, color, \*points):

 points = [point for point in points]

 pygame.draw.polygon(screen, color, points)

def check\_temperature(line):

 result = False

 for node in DATABASE.get\_id\_nodes\_of\_line(line.id):

 if DATABASE.get\_temperature(node) != 0:

 result = True

 break

 return result

def find\_index\_temperature\_node(line):

 for node\_id in DATABASE.get\_id\_nodes\_of\_line(line.id):

 if DATABASE.get\_temperature(node\_id) != 0:

 return DATABASE.get\_id\_nodes\_of\_line(line.id).index(node\_id)

def check\_temperature\_line(list\_of\_lines, done\_lines):

 result = False

 for line in list\_of\_lines:

 if line not in done\_lines:

 if check\_temperature(line):

 result = True

 break

 return result

def find\_temperature\_line(list\_of\_lines, done\_lines):

 for line in list\_of\_lines:

 if line not in done\_lines:

 if check\_temperature(line):

 return line

def temperature\_mke\_for\_changed\_line(line, heat\_transfer, environment\_t, heat\_source\_power):

 print('начали')

 results = []

 list\_of\_transcalency = []

 for id\_finit\_line in DATABASE.get\_list\_of\_id\_finit\_lines(line.id):

 transcalency = DATABASE.get\_transcalency\_of\_finit\_line(id\_finit\_line)

 list\_of\_transcalency.append(transcalency)

 list\_of\_transcalency = list\_of\_transcalency[::-1]

 matrix = TemperatureMatrix(len(list\_of\_transcalency) + 1)

 matrix.fill(line.id, heat\_transfer, environment\_t, heat\_source\_power, 2)

 if matrix.determinant != 0:

 results = matrix.find\_unknown()[::-1]

 for i in range(len(DATABASE.get\_id\_nodes\_of\_line(line.id))):

 DATABASE.set\_temperature(DATABASE.get\_id\_nodes\_of\_line(line.id)[i], results[i])

 print('----------')

 print(results)

 print('----------')

def mke(list\_of\_lines, heat\_transfer, environment\_t, heat\_source\_power):

 done\_lines = []

 while check\_temperature\_line(list\_of\_lines, done\_lines):

 line = find\_temperature\_line(list\_of\_lines, done\_lines)

 if find\_index\_temperature\_node(line) == 0:

 line.temperature\_mke(heat\_transfer, environment\_t, heat\_source\_power)

 done\_lines.append(line)

 if find\_index\_temperature\_node(line) == len(DATABASE.get\_id\_nodes\_of\_line(line.id)) - 1:

 temperature\_mke\_for\_changed\_line(line, heat\_transfer, environment\_t, heat\_source\_power)

 done\_lines.append(line)

1. Модуль database

import sqlite3

from modules.constants import \*

class DateBase:

 def \_\_init\_\_(self):

 # self.db = sqlite3.connect("C:/Users/Specter/YandexDisk/[Work]/[Projects]/УИРы/МКЭ/Project/main/data/data.db")

 self.db = sqlite3.connect("data/data.db")

 # Point

 def add\_point\_into\_db(self, x, y):

 query = f"INSERT INTO points(x, y) VALUES ({x}, {y})"

 self.db.execute(query)

 self.db.commit()

 def get\_id\_point(self, x, y):

 query = f"SELECT id FROM points WHERE x={x} AND y={y}"

 id = self.db.execute(query).fetchone()[0]

 return id

 def get\_coords\_point(self, id):

 query = f"SELECT x, y FROM points WHERE id={id}"

 x, y = self.db.execute(query).fetchone()

 return x, y

 def get\_color\_point(self, id):

 query = f"SELECT color FROM points WHERE id={id}"

 color = self.db.execute(query).fetchone()[0]

 return str\_to\_color(color)

 def set\_color\_point(self, color):

 color = color\_to\_str(color)

 query = f"INSERT INTO points(color) VALUES('{color}')"

 self.db.execute(query)

 self.db.commit()

 def get\_point(self, id):

 query = f"SELECT x, y, color FROM points WHERE id={id}"

 x, y, color = self.db.execute(query).fetchone()

 return x, y, color

 # Line

 def add\_line\_into\_db(self, point\_id1, point\_id2):

 point1 = min([point\_id1, point\_id2])

 point2 = max([point\_id1, point\_id2])

 id\_points = ';'.join(map(str, [point1, point2]))

 query = f"INSERT INTO lines(id\_points) VALUES('{id\_points}')"

 self.db.execute(query)

 self.db.commit()

 def get\_id\_line(self, point\_id1, point\_id2):

 point1 = min([point\_id1, point\_id2])

 point2 = max([point\_id1, point\_id2])

 id\_points = ';'.join(map(str, [point1, point2]))

 query = f"SELECT id FROM lines WHERE id\_points='{id\_points}'"

 id = self.db.execute(query).fetchone()[0]

 return id

 def get\_id\_points\_from\_line(self, id):

 query = f"SELECT id\_points FROM lines WHERE id={id}"

 id\_points = map(int, self.db.execute(query).fetchone()[0].split(';'))

 return list(id\_points)

 def get\_coords\_line(self, id):

 result = []

 query = f"SELECT id\_points FROM lines WHERE id={id}"

 id\_points = map(int, self.db.execute(query).fetchone()[0].split(';'))

 for id\_point in id\_points:

 coords = self.get\_coords\_point(id\_point)

 result.append(coords)

 return result

 def get\_color\_line(self, id):

 query = f"SELECT color FROM lines WHERE id={id}"

 color = self.db.execute(query).fetchone()[0]

 return str\_to\_color(color)

 def set\_color\_line(self, color):

 color = color\_to\_str(color)

 query = f"INSERT INTO lines(color) VALUES('{color}')"

 self.db.execute(query)

 self.db.commit()

 def set\_finit\_lines(self, id, list\_of\_id\_finit\_lines):

 id\_finit\_lines = ';'.join(map(str, list\_of\_id\_finit\_lines))

 query = f"UPDATE lines SET id\_finit\_lines='{id\_finit\_lines}' WHERE id={id}"

 self.db.execute(query)

 self.db.commit()

 def get\_list\_of\_id\_finit\_lines(self, id):

 query = f"SELECT id\_finit\_lines FROM lines WHERE id={id}"

 if self.db.execute(query).fetchone()[0]:

 id\_finit\_lines = map(int, self.db.execute(query).fetchone()[0].split(';'))

 return list(id\_finit\_lines)

 def get\_id\_nodes\_of\_line(self, id):

 id\_of\_nodes = []

 finit\_lines = self.get\_list\_of\_id\_finit\_lines(id)

 for i in range(len(finit\_lines)):

 if i == 0:

 id\_of\_nodes.append(DATABASE.get\_id\_nodes\_of\_finit\_line(finit\_lines[i])[0])

 id\_of\_nodes.append(DATABASE.get\_id\_nodes\_of\_finit\_line(finit\_lines[i])[1])

 else:

 id\_of\_nodes.append(DATABASE.get\_id\_nodes\_of\_finit\_line(finit\_lines[i])[1])

 return list(id\_of\_nodes)

 def get\_known\_temperature(self, id):

 for node\_id in self.get\_id\_nodes\_of\_line(id):

 if self.get\_temperature(node\_id) != 0:

 return self.get\_temperature(node\_id)

 def get\_temperature\_of\_line(self, id):

 result = []

 for node in self.get\_id\_nodes\_of\_line(id):

 result.append(self.get\_temperature(node))

 return result

 # Area

 def add\_area\_into\_db(self, lines\_id):

 lines\_id = sorted(lines\_id)

 id\_lines = ';'.join(map(str, lines\_id))

 query = f"INSERT INTO areas(id\_lines) VALUES('{id\_lines}')"

 self.db.execute(query)

 self.db.commit()

 def get\_id\_area(self, lines\_id):

 lines\_id = sorted(lines\_id)

 id\_lines = ';'.join(map(str, lines\_id))

 query = f"SELECT id FROM areas WHERE id\_lines='{id\_lines}'"

 id = self.db.execute(query).fetchone()[0]

 return id

 def get\_list\_of\_coords(self, id):

 query = f"SELECT id\_lines FROM areas WHERE id={id}"

 id\_lines = map(int, self.db.execute(query).fetchone()[0].split(';'))

 list\_of\_coords\_of\_area = []

 for id\_line in id\_lines:

 list\_of\_coords\_of\_line = self.get\_coords\_line(id\_line)

 for coords\_of\_line in list\_of\_coords\_of\_line:

 if coords\_of\_line not in list\_of\_coords\_of\_area:

 list\_of\_coords\_of\_area.append(coords\_of\_line)

 return list\_of\_coords\_of\_area

 def get\_color\_area(self, id):

 query = f"SELECT color FROM areas WHERE id={id}"

 color = self.db.execute(query).fetchone()[0]

 return str\_to\_color(color)

 # Node

 def add\_node\_into\_db(self, coords):

 x, y = coords

 query = f"INSERT INTO nodes(x, y) VALUES({x}, {y})"

 self.db.execute(query)

 self.db.commit()

 def get\_id\_node(self, coords):

 x, y = coords

 query = f"SELECT id FROM nodes WHERE x={x} AND y={y}"

 id = self.db.execute(query).fetchone()[0]

 return id

 def get\_coords\_node(self, id):

 query = f"SELECT x, y FROM nodes WHERE id={id}"

 x, y = self.db.execute(query).fetchone()

 return x, y

 def get\_color\_node(self, id):

 query = f"SELECT color FROM nodes WHERE id={id}"

 color = self.db.execute(query).fetchone()[0]

 return str\_to\_color(color)

 def set\_temperature(self, id, temperature):

 query = f"UPDATE nodes SET temperature={temperature} WHERE id={id}"

 self.db.execute(query)

 self.db.commit()

 def get\_temperature(self, id):

 query = f"SELECT temperature FROM nodes WHERE id={id}"

 temperature = self.db.execute(query).fetchone()[0]

 return temperature

 def get\_all\_temperatures(self, lines):

 temperatures = []

 for line in lines:

 if self.get\_list\_of\_id\_finit\_lines(line.id) is not None:

 for node\_id in self.get\_id\_nodes\_of\_line(line.id):

 temperatures.append(self.get\_temperature(node\_id))

 return temperatures

 # Materials

 def add\_material\_into\_db(self, material):

 query = f"INSERT INTO materials(material) VALUES('{material}')"

 self.db.execute(query)

 self.db.commit()

 def set\_transcalensy(self, id, transcalensy):

 query = f"UPDATE materials SET lambda={transcalensy} WHERE id={id}"

 self.db.execute(query)

 self.db.commit()

 def get\_id\_material(self, material):

 query = f"SELECT id FROM materials WHERE material='{material}'"

 id = self.db.execute(query).fetchone()[0]

 return id

 def get\_name\_material(self, id):

 query = f"SELECT material FROM materials WHERE id={id}"

 name = self.db.execute(query).fetchone()[0]

 return name

 def get\_lambda(self, id):

 query = f"SELECT lambda FROM materials WHERE id={id}"

 transcalensy = self.db.execute(query).fetchone()[0]

 return transcalensy

 def get\_list\_of\_materials(self):

 query = f"SELECT material FROM materials"

 materials = self.db.execute(query).fetchone()[0]

 # Finit lines

 def add\_finit\_line\_into\_db(self, node\_id1, node\_id2):

 id\_nodes = ';'.join(map(str, [node\_id1, node\_id2]))

 query1 = f"INSERT INTO finit\_lines(id\_nodes) VALUES('{id\_nodes}')"

 self.db.execute(query1)

 self.db.commit()

 x1, y1 = self.get\_coords\_node(node\_id1)

 x2, y2 = self.get\_coords\_node(node\_id2)

 h = (((x1 - x2) \*\* 2 + (y1 - y2) \*\* 2) \*\* 0.5) / 100

 id = self.get\_id\_finit\_line(node\_id1, node\_id2)

 query2 = f"UPDATE finit\_lines SET h={h} WHERE id={id}"

 self.db.execute(query2)

 self.db.commit()

 def set\_material\_of\_finit\_line(self, id, material\_id):

 query = f"UPDATE finit\_lines SET id\_material={material\_id} WHERE id={id}"

 self.db.execute(query)

 self.db.commit()

 def get\_id\_finit\_line(self, node\_id1, node\_id2):

 id\_nodes = ';'.join(map(str, [node\_id1, node\_id2]))

 query = f"SELECT id FROM finit\_lines WHERE id\_nodes='{id\_nodes}'"

 id = self.db.execute(query).fetchone()[0]

 return id

 def get\_transcalency\_of\_finit\_line(self, id\_finit\_line):

 query = f"SELECT id\_material FROM finit\_lines WHERE id='{id\_finit\_line}'"

 material = self.db.execute(query).fetchone()[0]

 transcalensy = self.get\_lambda(material)

 return transcalensy

 def get\_h\_of\_finit\_line(self, id):

 query = f"SELECT h FROM finit\_lines WHERE id='{id}'"

 h = self.db.execute(query).fetchone()[0]

 return h

 def get\_id\_nodes\_of\_finit\_line(self, id):

 query = f"SELECT id\_nodes FROM finit\_lines WHERE id={id}"

 if self.db.execute(query).fetchone()[0]:

 id\_nodes = map(int, self.db.execute(query).fetchone()[0].split(';'))

 return list(id\_nodes)

 # Work database

 def clear(self):

 query = f"DELETE from points"

 self.db.execute(query)

 query = f"DELETE from sqlite\_sequence WHERE name='points'"

 self.db.execute(query)

 self.db.commit()

 query = f"DELETE from lines"

 self.db.execute(query)

 query = f"DELETE from sqlite\_sequence WHERE name='lines'"

 self.db.execute(query)

 self.db.commit()

 query = f"DELETE from areas"

 self.db.execute(query)

 query = f"DELETE from sqlite\_sequence WHERE name='areas'"

 self.db.execute(query)

 self.db.commit()

 query = f"DELETE from nodes"

 self.db.execute(query)

 query = f"DELETE from sqlite\_sequence WHERE name='nodes'"

 self.db.execute(query)

 self.db.commit()

 query = f"DELETE from finit\_lines"

 self.db.execute(query)

 query = f"DELETE from sqlite\_sequence WHERE name='finit\_lines'"

 self.db.execute(query)

 self.db.commit()

 '''

 query = f"DELETE from materials"

 self.db.execute(query)

 query = f"DELETE from sqlite\_sequence WHERE name='materials'"

 self.db.execute(query)

 self.db.commit()

 '''

 def close(self):

 self.db.close()

DATABASE = DateBase()

# DATABASE.clear()

1. Модуль matrix

import numpy as np

from modules.database import \*

def rewrite\_matrix(change\_matrix, matrix, degree):

 for i in range(degree):

 for j in range(degree):

 change\_matrix[i][j] = matrix[i][j]

class TemperatureMatrix:

 def \_\_init\_\_(self, degree):

 self.degree = degree

 self.matrix = np.zeros(pow(self.degree, 2)).reshape(self.degree, self.degree)

 self.free\_member\_matrix = np.zeros(self.degree)

 self.determinant = None

 def fill(self, line\_id, heat\_transfer, environment\_t, heat\_source\_power, way):

 if way == 1:

 list\_of\_id\_elements = DATABASE.get\_list\_of\_id\_finit\_lines(line\_id)

 elif way == 2:

 list\_of\_id\_elements = DATABASE.get\_list\_of\_id\_finit\_lines(line\_id)[::-1]

 known\_temperature = DATABASE.get\_known\_temperature(line\_id)

 for i in range(self.degree):

 if i == 0:

 self.matrix[i][i] = 1

 self.free\_member\_matrix[i] = known\_temperature

 elif i == self.degree - 1:

 element\_id = list\_of\_id\_elements[i - 1]

 transcalency = DATABASE.get\_transcalency\_of\_finit\_line(element\_id)

 h = DATABASE.get\_h\_of\_finit\_line(element\_id)

 self.matrix[i][i - 1] = -transcalency / h

 self.matrix[i][i] = (transcalency / h) + heat\_transfer

 self.free\_member\_matrix[i] = heat\_transfer \* environment\_t

 else:

 element\_id1 = list\_of\_id\_elements[i - 1]

 element\_id2 = list\_of\_id\_elements[i]

 transcalency1 = DATABASE.get\_transcalency\_of\_finit\_line(element\_id1)

 transcalency2 = DATABASE.get\_transcalency\_of\_finit\_line(element\_id2)

 h1 = DATABASE.get\_h\_of\_finit\_line(element\_id1)

 h2 = DATABASE.get\_h\_of\_finit\_line(element\_id2)

 self.matrix[i][i - 1] = -transcalency1 / h1

 self.matrix[i][i] = transcalency1 / h1 + transcalency2 / h2

 self.matrix[i][i + 1] = -transcalency2 / h2

 self.free\_member\_matrix[i] = heat\_source\_power \* (h1 + h2) / 2

 self.determinant = np.linalg.det(self.matrix)

 def find\_unknown(self):

 temperatures = []

 changeable\_matrix = np.zeros(pow(self.degree, 2)).reshape(self.degree, self.degree)

 rewrite\_matrix(changeable\_matrix, self.matrix, self.degree)

 for i in range(self.degree):

 for j in range(self.degree):

 changeable\_matrix[j][i] = self.free\_member\_matrix[j]

 temperatures.append(np.linalg.det(changeable\_matrix) / self.determinant)

 rewrite\_matrix(changeable\_matrix, self.matrix, self.degree)

 return(temperatures)

 def print(self):

 string = []

 for i in range(self.degree):

 for j in range(self.degree):

 string.append(self.matrix[i][j])

 print(' '.join(map(str, string)))

 string = []

 print(self.free\_member\_matrix)

1. Модуль elements

from modules.constants import \*

from modules.database import \*

from modules.functions import \*

import pygame

class FinitElements:

 def \_\_init\_\_(self):

 self.finit\_elements = {'nodes': [],

 'temperature\_finit\_lines': [],

 'finit\_areas': []}

 def add\_node(self, coords):

 x, y = coords

 if x is not None and y is not None:

 if not exactly\_check\_node(coords, self):

 DATABASE.add\_node\_into\_db(coords)

 node = Node(coords)

 self.finit\_elements['nodes'].append(node)

 # def add\_temperature\_finit\_line(self, nodes, transcalency):

 def get\_node(self, coords):

 x, y = coords

 for node in self.finit\_elements['nodes']:

 if x == node.x and y == node.y:

 return node

 def get\_finit\_elements(self):

 return self.finit\_elements

 def get\_nodes(self):

 return self.finit\_elements['nodes']

class Node:

 def \_\_init\_\_(self, coords):

 self.id = DATABASE.get\_id\_node(coords)

 self.x, self.y = coords

 self.color = DATABASE.get\_color\_node(self.id)

 def draw(self, display, field):

 x = self.x + field.x

 y = -self.y + field.height + field.y

 pygame.draw.circle(display, self.color, (int(x), int(y)), 2)

 def \_\_str\_\_(self):

 return f"({self.x}, {self.y})"

class FinitLine:

 def \_\_init\_\_(self, nodes):

 node1, node2 = nodes

 self.node1, self.node2 = node1, node2

 x1, y1 = node1.x, node1.y

 x2, y2 = node2.x, node2.y

 self.transcalency = None

 def set\_material(self, material\_id):

 self.transcalency = DATABASE.get\_lambda(material\_id)

1. Модуль constants

import pygame

# geometry

SIZE = WIDTH, HEIGHT = 800, 600

# color

WHITE = pygame.Color('white')

BLACK = pygame.Color('black')

GREY = pygame.Color(200, 200, 200)

RED = pygame.Color(255, 0, 0)

LIGHTRED = pygame.Color(255, 59, 0)

ORANGERED = pygame.Color(255, 118, 2)

ORANGE = pygame.Color(255, 177, 8)

YELLOW = pygame.Color(255, 235, 24)

YELLOWGREEN = pygame.Color(234, 255, 53)

GREEN = pygame.Color(160, 255, 99)

LIGHTGREEN = pygame.Color(99, 255, 160)

AQUAMARINE = pygame.Color(53, 255, 234)

CYAN = pygame.Color(24, 235, 255)

VERYLIGHTBLUE = pygame.Color(8, 177, 255)

LIGHTBLUE = pygame.Color(2, 118, 255)

BLUE = pygame.Color(0, 59, 255)

DARKBLUE = pygame.Color(0, 0, 255)

LIST\_OF\_COLORS = [DARKBLUE, BLUE, LIGHTBLUE, VERYLIGHTBLUE, CYAN,

 AQUAMARINE, LIGHTGREEN, GREEN, YELLOWGREEN,

 YELLOW, ORANGE, ORANGERED, LIGHTRED, RED]

DICT\_OF\_COLOR = {'WHITE' : WHITE, 'BLACK' : BLACK, 'GREY' : GREY,

 'RED' : RED, 'LIGHTRED' : LIGHTRED, 'ORANGERED' : ORANGERED,

 'ORANGE' : ORANGE, 'YELLOW' : YELLOW, 'YELLOWGREEN' : YELLOWGREEN,

 'GREEN' : GREEN, 'LIGHTGREEN' : LIGHTGREEN, 'AQUAMARINE' : AQUAMARINE,

 'CYAN' : CYAN, 'VERYLIGHTBLUE' : VERYLIGHTBLUE, 'LIGHTBLUE' : LIGHTBLUE,

 'BLUE' : BLUE, 'DARKBLUE' : DARKBLUE}

def str\_to\_color(string):

 return DICT\_OF\_COLOR[string]