|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра прикладной математики | | |
| Лабораторная работа № 1 | | |
| по дисциплине «Создание современных кроссплатформенных приложений на основе web-технологий» | | |
| **Создание страниц на HTML и CSS** | | |
|  | | |
|  |  |  |
| Группа ПМ-21 | ЯКОВЛЕВ СЕРГЕЙ |
| Место для ввода текста. | ГОЛЫШЕВ ДАНИЛА |
|  | ПОРСИН ДАНИЛ |
|  |  |
|  |  |
| Преподаватели | ПЕТРОВ Р.В. |
|  |  |
| Новосибирск, 2025 | | |

1. **Постановка задачи**

Условие задачи

МКЭ для двумерной краевой задачи для эллиптического уравнения в декартовой системе координат. Базисные функции линейные на треугольниках. Краевые условия всех типов. Коэффициент диффузии разложить по квадратичным базисным функциям. Матрицу СЛАУ генерировать в разреженном строчном формате. Для решения СЛАУ использовать МСГ или ЛОС с неполной факторизацией.

Исходный вид решаемого уравнения

В общем виде уравнение, определяющее эллиптическую краевую задачу для функции , имеет вид

В заданной системе координат и с заданными краевыми условиями имеем систему

заданную в некоторой расчётной области c границей

1. **Теоретическая часть**

Вариационная постановка

Для исходной краевой задачи имеем эквивалентную вариационную постановку в форме уравнения Галеркина

где – пространство пробных функций , которые вместе со своими производными до 1-го порядка включительно интегрируемы с квадратом на и удовлетворяют нулевым первым краевым условиям на границе , – множество функций, имеющих суммируемые с квадратом первые производные и удовлетворяющих только первым краевым условиям на границе .

Конечноэлементная дискретизация и переход к локальным матрицами

Рассмотрим конечномерное пространство размерности с базисом из финитных кусочно-полиномиальнаых функций , аппроксимирующее пространство . Заменим в записанном выше уравнении Галёркина функцию аппроксимирующей её функцией , а функцию – функцией , где - конечномерные пространства размерности , аппроксимирующие исходные пространства соответственно. Тогда получаем аппроксимацию данного уравнения

Так как , то имеем следующие представление для данной функции

Подставив данное представление в полученную аппроксимацию уравнения Галёркина, получаем следующую систему уравнений

где – множество индексов таких, что базисные функции пространства являются и базисными функциями пространства и .

Так как , то имеем следующее представление для данной функции

где компоненты вектора весов определяются следующим образом:



Для данных компонент имеем СЛАУ

Данные компоненты должны быть фиксированы и могут быть определены из условия

Конечноэлементная СЛАУ для компонент вектора весов может быть записана в матричном виде

где компоненты матрицы и вектора , введя следующие обозначения

где , – компонента матрицы жёсткости исходной расчётной области , – вклад от элемента в компоненту матрицы жёсткости , – компонента матрицы массы исходной расчётной области , – вклад от элемента в компоненту матрицу массы , – вклад от краевого условия третьего рода в компоненту глобальной матрицы ,– вклад от исходной расчётной области в компоненту глобальной матрицы , – вклад от исходной расчётной области в компоненту глобального вектора правой части , – вклад от элемента в компоненту вектора , – вклад от краевого условия второго рода в компоненту глобального вектора правой части , – вклад от ребра в компоненту вектора , – вклад от краевого условия третьего рода в компоненту глобального вектора правой части , – вклад от ребра в компоненту вектора ,определяются следующим образом

Аналитические выражения для вычисления элементов локальных матриц и векторов правой части

Рассмотрим -координаты треугольника с вершинами , представляющие собой множество из 3 линейных функций вида

заданные на данном треугольном элементе так, что каждая из них равно единице на одной вершине треугольника и нулю на двух остальных. Для данных функций верно

Рассмотрим локальную задачу на треугольном элементе

или, используя введённые выше определения,

заключающуюся в построении размерности количества узлов на элементе , следовательно размерности 3 локальной матрицы и локального вектора правой части . Посредством - координат на треугольном элементе простроим линейные на треугольнике локальные базисные функции для вычисления компонент и локальных матриц жесткости и массы соответственно, компонент локального вектора правой части и для построения локального линейного интерполянта функции правой части и построим квадратичные на треугольниках локальные базисные функции для построения локального квадратичного интерполянта коэффициента диффузии :

Тогда, исходя из свойств - координат и заменяя коэффициент диффузии своим локальным на квадратичным интерполянтом

где – значение коэффициента диффузии в соответствующем узле c локальным номером элемента , при этом значение соответствует серединному узлу соответствующего ребра, и функцию правой части своим локальным на линейным интерполянтом

где – значение функции правой части в соответствующем узле с локальным номером элемента , получаем следующие аналитические выражения для вычисления компонент и локальных матриц жёсткости и массы соответственно и компонент локального вектора правой части :

Рассмотрим ребра с координатами , и с координатами , , на которых заданы краевые условия второго и третьего рода соответственно. Построим на них линейные одномерные базисные функции и соответственно так, что каждая из функция равна 1 на одной вершине и нулю на оставшейся вершине соответствующего ребра, и локальные линейные интерполянты и параметров и соответственно на основе соответствующих построенных базисов

где – значения параметров в соответствующей вершине с номером ребра , – значение параметра в соответствующей вершине с номером ребра . Тогда получаем следующие аналитические выражения для вычисления компонент локальной матрицы ребра , компонент локального вектора ребра и компонент локального вектора ребра

где .

1. **Описание разработанной программы**

Входные и выходные данные

|  |  |  |  |
| --- | --- | --- | --- |
| Входные данные | Название файла | Содержание | Структура |
| nodes.txt | Узлы расчётной области | Вещественные числа, разделенных пробелом и/или знаком перехода на новую строку, каждая пара которых представляют собой координаты узла расчётной области |
| elems.txt | Треугольные элементы расчётной области | Целые числа, разделенный пробелом и/или знаком перехода на новую строку, в каждой четвёрке которых первые три числа – номера вершин треугольного элемента, четвёртое – номер подобласти, которой данный элемент принадлежит |
| first\_bound\_cond.txt, | Первое краевое условие | Целые числа, разделенные пробелом и/или знаком перехода на новую строку, в каждой тройке которых первые два числа – вершины ребра, на котором задано краевое условие, третье – номер соответствующих формул. |
| second\_bound\_cond.txt, | Второе краевое условие |
| third\_bound\_cond.txt | Третье краевое условие |
| slae\_params.txt | Параметры численного решения СЛАУ | Пара чисел, где первое – вещественное значения точности численного решения СЛАУ, второе – максимальное количество итерация численного решения СЛАУ. |
| Выходные данные | res.txt | Результат работы программы | До пустой строки: строки чисел, где первое – целый номер узла, второе – вещественное численное значение в данном узле, третье – аналитическое значение в данном узле, четвёртое – локальная погрешность в данном узле.  После пустой строки: строки чисел, где первое – целый номер треугольного элемента, второе – вещественное численное значение в центре массы данного треугольного элемента, третье – аналитическое значение в центре массы данного треугольного элемента, четвёртое – локальная погрешность в центре массы данного треугольного элемента. |

Основные модули программы

struct NODE {

double x, y;

int global\_num;

}

- структура узла, содержащая координаты double x, y, глобальный номер int global\_num;

struct ELEM {

std::vector<NODE> nodes;

NODE mass\_center;

int formula\_num;

std::vector<std::vector<double>> local\_mat;

std::vector<double> local\_vec;

}

- структура элемента, содержащая узлы-вершины std::vector<NODE> nodes, узелцентра массы NODE mass\_center, номер формулы краевого условия int formula\_num, локальную матрицу std::vector<std::vector<double>> local\_mat,

локальный вектор std::vector<double> local\_vec;

struct AREA {

std::vector<ELEM> grid;

std::vector<NODE> nodes;

std::vector<ELEM> first\_bound\_cond, second\_bound\_cond, third\_bound\_cond;

}

- структура расчётной области, содержащая сетку std::vector<ELEM> grid, узлы std::vector<NODE> nodes, ребра первого, второго и третьего краевых условий

std::vector<ELEM> first\_bound\_cond, second\_bound\_cond, third\_bound\_cond соответственно;

struct MAT {

int n;

std::vector<int> ig, jg;

std::vector<double> ggl, di;

}

- структура матрицы коэффициентов СЛАУ, содержащая размерность int n, представление в разреженном строчном формате std::vector<int> ig, jg, std::vector<double> ggl, di;

struct SLAE {

MAT mat;

std::vector<double> vec;

std::vector<double> sol;

double epsilon;

}

- структура СЛАУ, содержащая матрицу коэффициентов MAT mat, вектор правой части std::vector<double> vec, вектор-решение std::vector<double> sol, точность численного решения double epsilon;

double mass\_center\_num\_val(std::vector<double>& slae\_sol, ELEM& elem) – расчёт численного значения центре массы треугольного элемента ELEM& elem по численным значениям std::vector<double>& slae\_sol в вершинах данного элемента;

void portrait(MAT& mat, AREA area) – построение портрета глобальной матрицы MAT& mat по расчётной области AREA area;

void solve\_local(ELEM & elem) – решение локальной задачи на треугольном элементе ELEM & elem;

void add\_to\_global(SLAE& slae, ELEM & elem) – занесение локальных матрицы и вектора элемента ELEM & elem в глобальные матрицу и вектор конечноэлементой СЛАУ SLAE& slae соответсвенно;

void first\_bound\_cond(SLAE& slae, ELEM& edge) – учёт заданного на ребре ELEM& edge первого краевого условия в конечноэлеметной СЛАУ SLAE& slae;

void second\_bound\_cond(SLAE& slae, ELEM& edge) – учёт заданного на ребре ELEM& edge второго краевого условия в конечноэлеметной СЛАУ SLAE& slae;

void third\_bound\_cond(SLAE& slae, ELEM& edge) – учёт заданного на ребре ELEM& edge третьего краевого условия в конечноэлеметной СЛАУ SLAE& slae;

void solve\_SLAE\_CGM(SLAE& slae, int max\_iter) – численное решение СЛАУ SLAE& slae методом сопряжённых градиентов с максимальным количеством итераций int max\_iter;

1. **Тестирование**

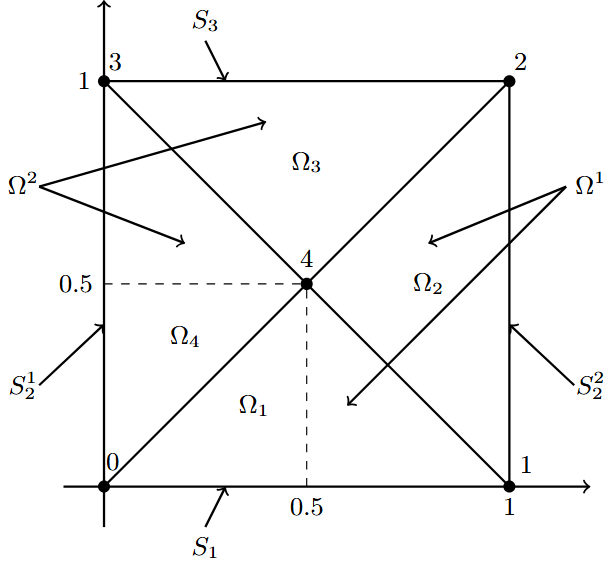
Тест 1

*Назначение*

Верификация программы на равномерной сетке при точности численного решения равной .

*Задача*

*Расчётная область*

**

*Результаты*

* Узлы сетки

|  |  |  |  |
| --- | --- | --- | --- |
| Узел |  |  |  |
| 0 | 0.000000000000000e+00 | 0.000000000000000e+00 | 0.000000000000000e+00 |
| 1 | 1.000000000000000e+00 | 1.000000000000000e+00 | 0.000000000000000e+00 |
| 2 | 2.000000000000000e+00 | 2.000000000000000e+00 | 4.440892098500626e-16 |
| 3 | 1.000000000000000e+00 | 1.000000000000000e+00 | 0.000000000000000e+00 |
| 4 | 1.000000000000000e+00 | 1.000000000000000e+00 | 2.220446049250313e-16 |

* Центры масс элементов

|  |  |  |  |
| --- | --- | --- | --- |
| Элемент |  |  |  |
| 0 | 6.666666666666667e-01 | 6.666666666666666e-01 | 1.110223024625157e-16 |
| 1 | 1.333333333333333e+00 | 1.333333333333333e+00 | 0.000000000000000e+00 |
| 2 | 1.333333333333333e+00 | 1.333333333333333e+00 | 0.000000000000000e+00 |
| 3 | 6.666666666666667e-01 | 6.666666666666666e-01 | 1.110223024625157e-16 |

*Вывод*

Полученные результаты соответствуют заданной точности, из чего следует корректность работы программы на равномерных сетках.

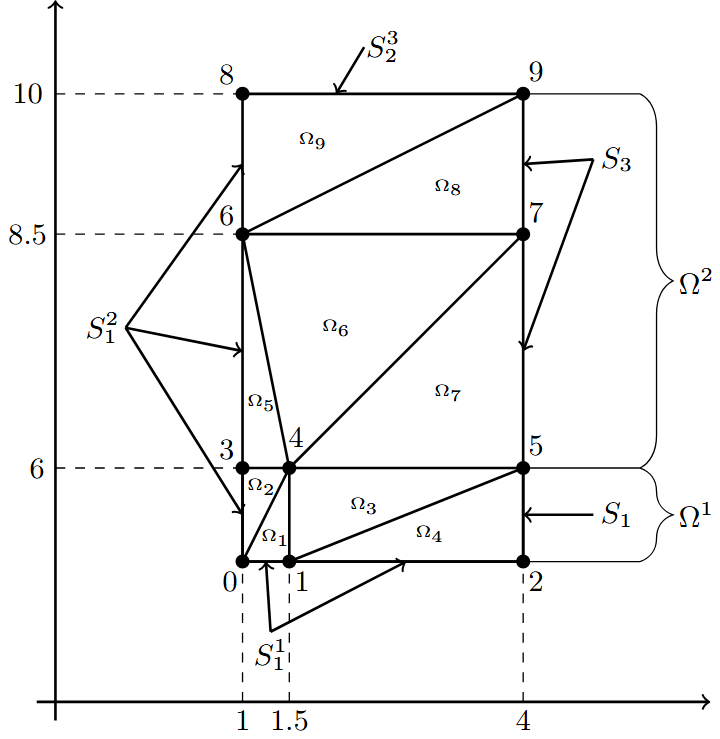
Тест 2

*Назначение*

Верификация программы на неравномерной сетке при точности численного решения равной .

*Задача*

*Расчётная область*



*Результаты*

* Узлы сетки

|  |  |  |  |
| --- | --- | --- | --- |
| Узел |  |  |  |
| 0 | 2.900000000000002e+01 | 2.900000000000000e+01 | 1.776356839400250e-14 |
| 1 | 2.950000000000001e+01 | 2.950000000000000e+01 | 1.065814103640150e-14 |
| 2 | 3.200000000000000e+01 | 3.200000000000000e+01 | 0.000000000000000e+00 |
| 3 | 3.500000000000004e+01 | 3.500000000000000e+01 | 3.552713678800501e-14 |
| 4 | 3.550000000000003e+01 | 3.550000000000000e+01 | 2.842170943040401e-14 |
| 5 | 3.800000000000000e+01 | 3.800000000000000e+01 | 0.000000000000000e+00 |
| 6 | 5.000000000000002e+01 | 5.000000000000000e+01 | 2.131628207280301e-14 |
| 7 | 5.300000000000003e+01 | 5.300000000000000e+01 | 2.842170943040401e-14 |
| 8 | 5.900000000000000e+01 | 5.900000000000000e+01 | 0.000000000000000e+00 |
| 9 | 6.200000000000001e+01 | 6.200000000000000e+01 | 1.421085471520200e-14 |

* Центры масс элементов

|  |  |  |  |
| --- | --- | --- | --- |
| Элемент |  |  |  |
| 1 | 3.133333333333335e+01 | 3.133333333333334e+01 | 1.421085471520200e-14 |
| 2 | 3.316666666666670e+01 | 3.316666666666666e+01 | 3.552713678800501e-14 |
| 3 | 3.433333333333334e+01 | 3.433333333333334e+01 | 7.105427357601002e-15 |
| 4 | 3.316666666666667e+01 | 3.316666666666666e+01 | 7.105427357601002e-15 |
| 5 | 4.016666666666670e+01 | 4.016666666666666e+01 | 3.552713678800501e-14 |
| 6 | 4.616666666666670e+01 | 4.616666666666666e+01 | 3.552713678800501e-14 |
| 7 | 4.216666666666669e+01 | 4.216666666666666e+01 | 2.131628207280301e-14 |
| 8 | 5.500000000000001e+01 | 5.500000000000000e+01 | 1.421085471520200e-14 |
| 9 | 5.700000000000001e+01 | 5.700000000000000e+01 | 7.105427357601002e-15 |

*Вывод*

Полученные результаты соответствуют заданной точности, из чего следует корректность работы программы на неравномерных сетках.

1. **Исследования**

Порядок аппроксимации

*Задача*

*Расчётная область*

Такая же, как и в тесте 1.

*Результаты*

* Узлы сетки

|  |  |  |  |
| --- | --- | --- | --- |
| Узел |  |  |  |
| 0 | 0.000000000000000e+00 | 0.000000000000000e+00 | 0.000000000000000e+00 |
| 1 | 1.000000000000000e+00 | 1.000000000000000e+00 | 0.000000000000000e+00 |
| 2 | 2.406423792536005e+00 | 2.000000000000000e+00 | 4.064237925360055e-01 |
| 3 | 1.404237685128464e+00 | 1.000000000000000e+00 | 4.042376851284637e-01 |
| 4 | 1.022605791983472e+00 | 5.000000000000000e-01 | 5.226057919834723e-01 |

* Центры масс элементов

|  |  |  |  |
| --- | --- | --- | --- |
| Элемент |  |  |  |
| 1 | 6.742019306611575e-01 | 2.777777777777778e-01 | 3.964241528833797e-01 |
| 2 | 1.476343194839826e+00 | 9.444444444444445e-01 | 5.318987503953815e-01 |
| 3 | 1.611089089882647e+00 | 9.444444444444445e-01 | 6.666446454382028e-01 |
| 4 | 8.089478257039787e-01 | 2.777777777777778e-01 | 5.311700479262009e-01 |

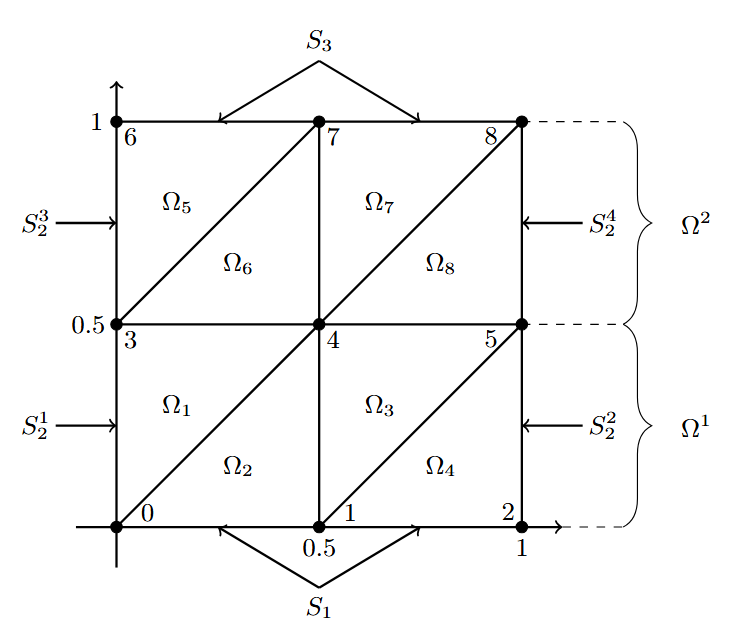
*Вывод*

Численное решение рассматриваемой задачи, аналитическим решением которой является квадратичная функция, имеет погрешности, не соответствует установленной точности, в то время как численное решение задачи теста 1, аналитическим решением которой является линейная функция, установленной точности соответствуют, из чего следует 1 порядок аппроксимации реализованного метода.

Порядок сходимости

*Задача*

*Расчётная область 1*



*Результаты 1*

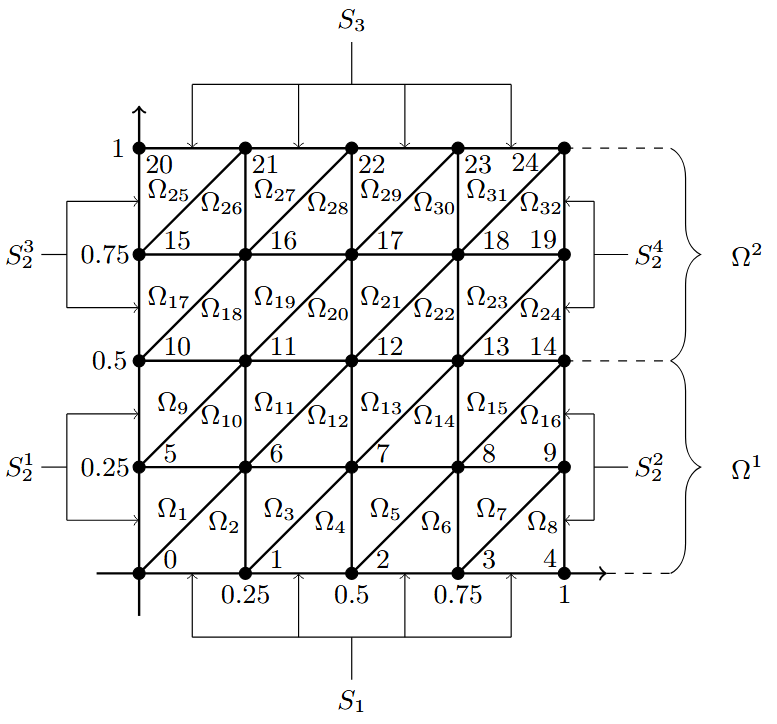
* Узлы сетки

|  |  |  |  |
| --- | --- | --- | --- |
| Узел |  |  |  |
| 0 | 1.000000000000000e+00 | 1.000000000000000e+00 | 0.000000000000000e+00 |
| 1 | 1.479425538604203e+00 | 1.479425538604203e+00 | 0.000000000000000e+00 |
| 2 | 1.841470984807897e+00 | 1.841470984807897e+00 | 0.000000000000000e+00 |
| 3 | 8.310045850197242e-01 | 8.775825618903728e-01 | 4.657797687064857e-02 |
| 4 | 1.351555909752767e+00 | 1.357008100494576e+00 | 5.452190741808360e-03 |
| 5 | 1.749336069361392e+00 | 1.719053546698269e+00 | 3.028252266312248e-02 |
| 6 | 4.966761079937753e-01 | 5.403023058681398e-01 | 4.362619787436445e-02 |
| 7 | 1.004414403389051e+00 | 1.019727844472343e+00 | 1.531344108329225e-02 |
| 8 | 1.401401835870250e+00 | 1.381773290676036e+00 | 1.962854519421375e-02 |

* Центры масс элементов

|  |  |  |  |
| --- | --- | --- | --- |
| Элемент |  |  |  |
| 1 | 1.060853498257497e+00 | 1.110853079008153e+00 | 4.999958075065547e-02 |
| 2 | 1.276993816118990e+00 | 1.313337928359077e+00 | 3.634411224008716e-02 |
| 3 | 1.526772505906121e+00 | 1.563326749384475e+00 | 3.655424347835390e-02 |
| 4 | 1.690077530924497e+00 | 1.726320084758962e+00 | 3.624255383446506e-02 |
| 5 | 7.773650321341833e-01 | 8.383083767764716e-01 | 6.094334464228834e-02 |
| 6 | 1.062324966053847e+00 | 1.113081957573100e+00 | 5.075699151925273e-02 |
| 7 | 1.252457383004022e+00 | 1.290782047152794e+00 | 3.832466414877134e-02 |
| 8 | 1.500764604994803e+00 | 1.526064113972985e+00 | 2.529950897818223e-02 |

*Расчётная область 2*



*Результаты 2*

* Узлы сетки

|  |  |  |  |
| --- | --- | --- | --- |
| Узел |  |  |  |
| 0 | 9.999999999999997e-01 | 1.000000000000000e+00 | 3.330669073875470e-16 |
| 1 | 1.247403959254523e+00 | 1.247403959254523e+00 | 2.220446049250313e-16 |
| 2 | 1.479425538604203e+00 | 1.479425538604203e+00 | 2.220446049250313e-16 |
| 3 | 1.681638760023334e+00 | 1.681638760023334e+00 | 0.000000000000000e+00 |
| 4 | 1.841470984807897e+00 | 1.841470984807897e+00 | 0.000000000000000e+00 |
| 5 | 9.590634833109386e-01 | 9.689124217106447e-01 | 9.848938399706131e-03 |
| 6 | 1.207374395286980e+00 | 1.216316380965168e+00 | 8.941985678187381e-03 |
| 7 | 1.448727834936881e+00 | 1.448337960314848e+00 | 3.898746220332150e-04 |
| 8 | 1.659775971253269e+00 | 1.650551181733979e+00 | 9.224789519290510e-03 |
| 9 | 1.823225017615547e+00 | 1.810383406518541e+00 | 1.284161109700555e-02 |
| 10 | 8.148257833442044e-01 | 8.775825618903728e-01 | 6.275677854616835e-02 |
| 11 | 1.085130362950210e+00 | 1.124986521144896e+00 | 3.985615819468591e-02 |
| 12 | 1.352711241601590e+00 | 1.357008100494576e+00 | 4.296858892985922e-03 |
| 13 | 1.582067320254197e+00 | 1.559221321913707e+00 | 2.284599834049006e-02 |
| 14 | 1.754638658194818e+00 | 1.719053546698269e+00 | 3.558511149654864e-02 |
| 15 | 6.998618426225179e-01 | 7.316888688738209e-01 | 3.182702625130296e-02 |
| 16 | 9.560903493944704e-01 | 9.790928281283439e-01 | 2.300247873387351e-02 |
| 17 | 1.203913358209579e+00 | 1.211114407478024e+00 | 7.201049268444670e-03 |
| 18 | 1.419853846701487e+00 | 1.413327628897155e+00 | 6.526217804331935e-03 |
| 19 | 1.585629361036757e+00 | 1.573159853681717e+00 | 1.246950735503982e-02 |
| 20 | 5.143849549001237e-01 | 5.403023058681398e-01 | 2.591735096801606e-02 |
| 21 | 7.695048703449625e-01 | 7.877062651226627e-01 | 1.820139477770022e-02 |
| 22 | 1.012276526464760e+00 | 1.019727844472343e+00 | 7.451318007583030e-03 |
| 23 | 1.224802829512296e+00 | 1.221941065891474e+00 | 2.861763620822133e-03 |
| 24 | 1.394138503369882e+00 | 1.381773290676036e+00 | 1.236521269384583e-02 |

* Центры масс элементов

|  |  |  |  |
| --- | --- | --- | --- |
| Элемент |  |  |  |
| 1 | 1.055479292865973e+00 | 1.069380147763235e+00 | 1.390085489726234e-02 |
| 2 | 1.151592784847168e+00 | 1.162425919393975e+00 | 1.083313454680690e-02 |
| 3 | 1.301168729826128e+00 | 1.313337928359077e+00 | 1.216919853294929e-02 |
| 4 | 1.391852444265202e+00 | 1.401244350261684e+00 | 9.391905996481764e-03 |
| 5 | 1.529309781598118e+00 | 1.536952327449895e+00 | 7.642545851777038e-03 |
| 6 | 1.606946756626936e+00 | 1.614899589770296e+00 | 7.952833143360927e-03 |
| 7 | 1.721546582964050e+00 | 1.726320084758962e+00 | 4.773501794912205e-03 |
| 8 | 1.782111587482259e+00 | 1.790107589949502e+00 | 7.996002467242302e-03 |
| 9 | 9.530065432017844e-01 | 9.976799827941405e-01 | 4.467343959235615e-02 |
| 10 | 1.083856080516043e+00 | 1.110853079008153e+00 | 2.699699849210990e-02 |
| 11 | 1.215071999946260e+00 | 1.241637763389982e+00 | 2.656576344372241e-02 |
| 12 | 1.336271157275150e+00 | 1.349671509875862e+00 | 1.340035260071226e-02 |
| 13 | 1.461168798930889e+00 | 1.465252162480800e+00 | 4.083363549910679e-03 |
| 14 | 1.563523708814782e+00 | 1.563326749384475e+00 | 1.969594303077304e-04 |
| 15 | 1.665493983234095e+00 | 1.654619919789867e+00 | 1.087406344422748e-02 |
| 16 | 1.745879882354544e+00 | 1.738534749563680e+00 | 7.345132790864906e-03 |
| 17 | 8.235926584537310e-01 | 8.691241769772583e-01 | 4.553151852352733e-02 |
| 18 | 9.520154985629614e-01 | 1.000527392525072e+00 | 4.851189396211053e-02 |
| 19 | 1.081711356851420e+00 | 1.113081957573100e+00 | 3.137060072168030e-02 |
| 20 | 1.213918320920460e+00 | 1.239345823392782e+00 | 2.542750247232206e-02 |
| 21 | 1.325492815504219e+00 | 1.336696356663918e+00 | 1.120354115969890e-02 |
| 22 | 1.451544136185758e+00 | 1.453001062901394e+00 | 1.456926715635909e-03 |
| 23 | 1.529183509330814e+00 | 1.526064113972985e+00 | 3.119395357828569e-03 |
| 24 | 1.640778446495257e+00 | 1.628209063080599e+00 | 1.256938341465830e-02 |
| 25 | 6.612505559558681e-01 | 6.917057076683554e-01 | 3.045515171248725e-02 |
| 26 | 8.084856874539835e-01 | 8.383083767764716e-01 | 2.982268932248811e-02 |
| 27 | 9.126239154013975e-01 | 9.356634882641973e-01 | 2.303957286279978e-02 |
| 28 | 1.057426744689603e+00 | 1.077126807644181e+00 | 1.970006295457827e-02 |
| 29 | 1.146997571395545e+00 | 1.159277887355015e+00 | 1.228031595946999e-02 |
| 30 | 1.282856678141121e+00 | 1.290782047152794e+00 | 7.925369011672956e-03 |
| 31 | 1.346265059861222e+00 | 1.348645644664082e+00 | 2.380584802860630e-03 |
| 32 | 1.466540570369375e+00 | 1.465990047331999e+00 | 5.505230373765713e-04 |

*Вывод*

Оценивая погрешность численного решения на сетке с шагом посредством вычисления относительной нормы вектора погрешностей полученных значений функции в центрах масс треугольных элементов, получаем

Следовательно, порядок сходимости реализованного метода равен 1.

1. **Листинг разработанной программы**

#include <vector>

#include <string>

#include <fstream>

#include <set>

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <math.h>

struct NODE {

double x, y;

int global\_num = -1;

NODE(double x, double y, int global\_num) :

x(x), y(y), global\_num(global\_num) {}

NODE(double x, double y) :

x(x), y(y) {}

NODE() {}

bool operator<(const NODE& node) const {

return global\_num < node.global\_num;

}

};

struct ELEM {

std::vector<NODE> nodes;

NODE mass\_center;

int formula\_num = -1;

std::vector<std::vector<double>> local\_mat;

std::vector<double> local\_vec;

ELEM(std::vector<NODE> nodes, int formula\_num, int n) :

local\_mat(n, std::vector<double>(n, 0)), local\_vec(n, 0),

nodes(nodes), formula\_num(formula\_num) {}

ELEM(std::vector<NODE> nodes) :

nodes(nodes) {}

};

struct AREA {

std::vector<ELEM> grid;

std::vector<NODE> nodes;

std::vector<ELEM> first\_bound\_cond, second\_bound\_cond, third\_bound\_cond;

};

struct MAT {

int n;

std::vector<int> ig, jg;

std::vector<double> ggl, di;

MAT(int n) : n(n) {}

};

struct SLAE {

MAT mat;

std::vector<double> vec;

std::vector<double> sol;

double epsilon;

SLAE(MAT mat, int n, double epsilon) :

mat(mat), vec(n), sol(n), epsilon(epsilon) {}

SLAE(MAT mat, int n) :

mat(mat), vec(n), sol(n) {};

void clear\_sol() {

for (int i = 0; i < mat.n; i++)

sol[i] = 0;

}

};

void input(AREA& area, double& epsilon, int& max\_iter, std::string nodes\_file, std::string elems\_file,

std::string first\_bound\_cond, std::string second\_bound\_cond\_file,

std::string third\_bound\_cond\_file, std::string slae\_params\_file) {

std::ifstream in(nodes\_file);

double x, y;

for (int global\_num = 0; in >> x >> y; global\_num++) {

NODE node(x, y, global\_num);

area.nodes.push\_back(node);

}

in.close();

in.open(elems\_file);

int node1\_global\_num, node2\_global\_num, node3\_global\_num, formula\_num;

while (in >> node1\_global\_num >> node2\_global\_num >> node3\_global\_num >> formula\_num) {

ELEM elem({ area.nodes[node1\_global\_num],area.nodes[node2\_global\_num],area.nodes[node3\_global\_num] },

formula\_num, 3);

std::sort(elem.nodes.begin(), elem.nodes.end());

elem.mass\_center.x = (elem.nodes[0].x + elem.nodes[1].x + elem.nodes[2].x) / 3;

elem.mass\_center.y = (elem.nodes[0].y + elem.nodes[1].y + elem.nodes[2].y) / 3;

area.grid.push\_back(elem);

}

in.close();

in.open(first\_bound\_cond);

while (in >> node1\_global\_num >> node2\_global\_num >> formula\_num) {

ELEM edge({ area.nodes[node1\_global\_num], area.nodes[node2\_global\_num] }, formula\_num, 0);

area.first\_bound\_cond.push\_back(edge);

}

in.close();

in.open(second\_bound\_cond\_file);

while (in >> node1\_global\_num >> node2\_global\_num >> formula\_num) {

ELEM edge({ area.nodes[node1\_global\_num], area.nodes[node2\_global\_num] }, formula\_num, 2);

area.second\_bound\_cond.push\_back(edge);

}

in.close();

in.open(third\_bound\_cond\_file);

while (in >> node1\_global\_num >> node2\_global\_num >> formula\_num) {

ELEM edge({ area.nodes[node1\_global\_num], area.nodes[node2\_global\_num] }, formula\_num, 2);

area.third\_bound\_cond.push\_back(edge);

}

in.close();

in.open(slae\_params\_file);

in >> epsilon >> max\_iter;

}

double u(NODE node) {

return sin(node.x) + cos(node.y);

}

double calc\_detD(ELEM& elem) {

return (elem.nodes[1].x - elem.nodes[0].x) \* (elem.nodes[2].y - elem.nodes[0].y) -

(elem.nodes[2].x - elem.nodes[0].x) \* (elem.nodes[1].y - elem.nodes[0].y);

}

double calc\_mes\_edge(ELEM& edge) {

return sqrt(pow(edge.nodes[1].x - edge.nodes[0].x, 2) + pow(edge.nodes[1].y - edge.nodes[0].y, 2));

}

double mass\_center\_num\_val(std::vector<double>& slae\_sol, ELEM& elem) {

double elem\_detD = calc\_detD(elem);

ELEM S23({ elem.nodes[1],elem.nodes[2],elem.mass\_center });

ELEM S31({ elem.nodes[2], elem.nodes[0],elem.mass\_center });

ELEM S12({ elem.nodes[0],elem.nodes[1],elem.mass\_center });

double L1 = calc\_detD(S23) / elem\_detD;

double L2 = calc\_detD(S31) / elem\_detD;

double L3 = calc\_detD(S12) / elem\_detD;

return slae\_sol[elem.nodes[0].global\_num] \* L1 +

slae\_sol[elem.nodes[1].global\_num] \* L2 +

slae\_sol[elem.nodes[2].global\_num] \* L3;

}

void output(AREA& area, std::vector<double> slae\_sol, std::string res\_file) {

std::ofstream out(res\_file);

out << std::scientific << std::setprecision(15);

double u\_analit;

double err;

for (int i = 0; i < slae\_sol.size(); i++) {

u\_analit = u(area.nodes[i]);

out << i << " " << slae\_sol[i] << " " << u\_analit << " " << abs(slae\_sol[i] - u\_analit) << std::endl;

}

out << std::endl;

double u\_num;

for (int i = 0; i < area.grid.size(); i++) {

u\_analit = u(area.grid[i].mass\_center);

u\_num = mass\_center\_num\_val(slae\_sol, area.grid[i]);

out << i+1 << " " << u\_num << " " << u\_analit << " " << abs(u\_num - u\_analit) << std::endl;

}

}

void portrait(MAT & mat, AREA area) {

std::vector< std::set<int>> list(area.nodes.size());

for (int elem = 0; elem < area.grid.size(); elem++) {

for (int node1 = 2; node1 >= 0; node1--) {

for (int node2 = node1-1; node2 >= 0; node2--) {

list[area.grid[elem].nodes[node1].global\_num].insert(area.grid[elem].nodes[node2].global\_num);

}

}

}

mat.ig.resize(area.nodes.size() + 1);

mat.ig[0] = 0;

for (int i = 1; i < area.nodes.size() + 1; i++) {

mat.ig[i] = mat.ig[i - 1] + list[i - 1].size();

}

mat.ggl.resize(mat.ig.back());

mat.jg.resize(mat.ggl.size());

mat.di.resize(area.nodes.size());

for (int node = 0, k = 0; node < list.size(); node++) {

for (auto conn\_node = list[node].begin(); conn\_node != list[node].end(); ++conn\_node, k++)

mat.jg[k] = \*conn\_node;

}

}

double gamma(int formula\_num) {

switch (formula\_num) {

case(1):

return 2;

break;

case(2):

return 1;

break;

}

}

double lambda(int formula\_num, NODE node) {

switch (formula\_num) {

case(1):

return node.x;

break;

case(2):

return node.y;

break;

}

}

double f(int formula\_num, NODE node) {

switch (formula\_num) {

case(1):

return -cos(node.x) + node.x \* sin(node.x) + node.x \* cos(node.y) + 2 \* sin(node.x) + 2 \* cos(node.y);

break;

case(2):

return node.y \* sin(node.x) + sin(node.y) + node.y \* cos(node.y) + sin(node.x) + cos(node.y);

break;

}

}

double ug(int formula\_num, NODE node) {

switch (formula\_num) {

case(1):

return sin(node.x) + 1;

break;

}

}

double beta(int formula\_num) {

switch (formula\_num) {

case(1):

return 1;

break;

}

}

double ubeta(int formula\_num, NODE node) {

switch (formula\_num) {

case(1):

return sin(node.x) + cos(1) - node.y \* sin(node.y);

break;

}

}

double teta(int formula\_num, NODE node) {

switch (formula\_num) {

case(1):

return -node.x \* cos(node.x);

break;

case(2):

return node.x \* cos(node.x);

break;

case(3):

return -node.y \* cos(node.x);

break;

case(4):

return node.y \* cos(node.x);

}

}

void solve\_local(ELEM & elem) {

double detD = calc\_detD(elem);

std::vector<std::vector<double>> coefs =

{ {(elem.nodes[1].x \* elem.nodes[2].y - elem.nodes[2].x \* elem.nodes[1].y) / detD,

(elem.nodes[1].y - elem.nodes[2].y) / detD,

(elem.nodes[2].x - elem.nodes[1].x) / detD},

{(elem.nodes[2].x \* elem.nodes[0].y - elem.nodes[0].x \* elem.nodes[2].y) / detD,

(elem.nodes[2].y - elem.nodes[0].y) / detD,

(elem.nodes[0].x - elem.nodes[2].x) / detD},

{(elem.nodes[0].x \* elem.nodes[1].y - elem.nodes[1].x \* elem.nodes[0].y) / detD,

(elem.nodes[0].y - elem.nodes[1].y) / detD,

(elem.nodes[1].x - elem.nodes[0].x) / detD}

};

NODE mid\_node\_1(

(elem.nodes[0].x + elem.nodes[1].x) / 2.,

(elem.nodes[0].y + elem.nodes[1].y) / 2.

);

NODE mid\_node\_2{

(elem.nodes[1].x + elem.nodes[2].x) / 2.,

(elem.nodes[1].y + elem.nodes[2].y) / 2.,

};

NODE mid\_node\_3{

(elem.nodes[0].x + elem.nodes[2].x) / 2.,

(elem.nodes[0].y + elem.nodes[2].y) / 2.,

};

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

elem.local\_mat[i][j] =

(i == j ? gamma(elem.formula\_num) \* (abs(detD) / 12) :

gamma(elem.formula\_num) \* (abs(detD) / 24)) +

(abs(detD) / 6) \* (coefs[i][1] \* coefs[j][1] + coefs[i][2] \* coefs[j][2]) \*

(lambda(elem.formula\_num, mid\_node\_1)

+ lambda(elem.formula\_num, mid\_node\_2)

+ lambda(elem.formula\_num, mid\_node\_3));

}

}

for (int i = 0; i < 3; i++) {

for (int k = 0; k < 3; k++) {

elem.local\_vec[i] += i == k ?

f(elem.formula\_num, elem.nodes[k]) \* (abs(detD) / 12) :

f(elem.formula\_num, elem.nodes[k]) \* (abs(detD) / 24);

}

}

}

int find\_ind\_dichotomy(std::vector<int>&arr, int start, int end, int goal) {

int i;

while (start<=end) {

i = start + (end - start) / 2.;

if (arr[i] == goal)

return i;

if (arr[i] < goal)

start = i + 1;

else

end = i-1;

}

return -1;

}

void add\_to\_global(SLAE& slae, ELEM & elem) {

for (int i = 0; i < elem.local\_mat.size(); i++) {

slae.mat.di[elem.nodes[i].global\_num] += elem.local\_mat[i][i];

slae.vec[elem.nodes[i].global\_num] += elem.local\_vec[i];

int found = 0;

for (int j = 0; j < i; j++) {

int ggl\_ij = find\_ind\_dichotomy(slae.mat.jg,

slae.mat.ig[elem.nodes[i].global\_num]+found,

slae.mat.ig[elem.nodes[i].global\_num + 1]-1,

elem.nodes[j].global\_num);

slae.mat.ggl[ggl\_ij] += elem.local\_mat[i][j];

found++;

}

}

}

void first\_bound\_cond(SLAE& slae, ELEM& edge) {

for (int node = 0; node < 2;  node++) {

slae.mat.di[edge.nodes[node].global\_num] = 1;

slae.vec[edge.nodes[node].global\_num] = ug(edge.formula\_num, edge.nodes[node]);

for (int k = 0; k < slae.mat.ig[edge.nodes[node].global\_num + 1] - slae.mat.ig[edge.nodes[node].global\_num]; k++) {

slae.vec[slae.mat.jg[slae.mat.ig[edge.nodes[node].global\_num]+k]] +=

slae.vec[edge.nodes[node].global\_num] \* (-slae.mat.ggl[slae.mat.ig[edge.nodes[node].global\_num] + k]);

slae.mat.ggl[slae.mat.ig[edge.nodes[node].global\_num] + k] = 0;

}

int found = 0;

for (int i = edge.nodes[node].global\_num + 1; i < slae.mat.n; i++) {

int ggl\_inodegm = find\_ind\_dichotomy(slae.mat.jg,

slae.mat.ig[i]+found,

slae.mat.ig[i + 1] - 1,

edge.nodes[node].global\_num);

if (ggl\_inodegm != -1) {

slae.vec[i] += slae.vec[edge.nodes[node].global\_num] \* (-slae.mat.ggl[ggl\_inodegm]);

slae.mat.ggl[ggl\_inodegm] = 0;

found++;

}

}

}

}

void second\_bound\_cond(SLAE& slae, ELEM& edge) {

std::vector<std::vector<double>> coefs = {

{2.,1.},

{1.,2.}

};

double mes\_edge = calc\_mes\_edge(edge);

for (int i = 0; i < 2; i++) {

for (int k = 0; k < 2; k++) {

edge.local\_vec[i] += (mes\_edge / 6.) \* coefs[i][k] \* teta(edge.formula\_num, edge.nodes[k]);

}

}

add\_to\_global(slae, edge);

}

void third\_bound\_cond(SLAE& slae, ELEM& edge) {

double mes\_edge = calc\_mes\_edge(edge);

for (int i = 0; i < 2; i++) {

for (int j = 0; j < 2; j++) {

edge.local\_mat[i][j] = i == j ?

beta(edge.formula\_num) \* mes\_edge / 3. :

beta(edge.formula\_num) \* mes\_edge / 6.;

}

}

for (int i = 0; i < 2; i++) {

for (int k = 0; k < 2; k++) {

edge.local\_vec[i] += edge.local\_mat[i][k] \* ubeta(edge.formula\_num, edge.nodes[k]);

}

}

add\_to\_global(slae, edge);

}

MAT factor(MAT& mat) {

MAT res = mat;

double sum;

int found\_num1, found\_num2;

for (int i = 0; i < res.n; i++) {

sum = 0;

found\_num1 = found\_num2 = 0;

for (int j = 0; j < i; j++) {

int ggl\_ij = find\_ind\_dichotomy(res.jg,

res.ig[i]+found\_num1, res.ig[i + 1] - 1, j);

if (ggl\_ij != -1) {

for (int k = 0; k < j; k++) {

int ggl\_ik = find\_ind\_dichotomy(res.jg,

res.ig[i]+found\_num2, res.ig[i + 1] - 1, k);

int ggl\_jk = find\_ind\_dichotomy(res.jg,

res.ig[j]+found\_num2, res.ig[j + 1] - 1, k);

if (ggl\_ik != -1 and ggl\_jk != -1)

sum += res.ggl[ggl\_ik] \* res.ggl[ggl\_jk];

found\_num2++;

}

res.ggl[ggl\_ij] = (1 / res.di[j]) \* (res.ggl[ggl\_ij] - sum);

found\_num1++;

}

sum = 0;

}

found\_num1 = 0;

for (int k = 0; k < i; k++) {

int ggl\_ik = find\_ind\_dichotomy(res.jg,

res.ig[i]+found\_num1, res.ig[i + 1] - 1, k);

if (ggl\_ik != -1) {

sum += res.ggl[ggl\_ik] \* res.ggl[ggl\_ik];

found\_num1++;

}

}

res.di[i] = sqrt(res.di[i] - sum);

}

return res;

}

void solve\_L(SLAE& slae) {

int found\_num;

for (int i = 0; i < slae.mat.n; i++) {

slae.sol[i] = slae.vec[i];

found\_num = 0;

for (int j = 0; j < i; j++) {

int ggl\_ij = find\_ind\_dichotomy(slae.mat.jg,

slae.mat.ig[i]+ found\_num, slae.mat.ig[i + 1] - 1,

j);

if (ggl\_ij != -1) {

slae.sol[i] -= slae.mat.ggl[ggl\_ij] \* slae.sol[j];

found\_num++;

}

}

slae.sol[i] /=slae.mat.di[i];

}

}

void solve\_Lt(SLAE& slae) {

int found\_num;

for (int j = slae.mat.n - 1; j >= 0; j--) {

slae.sol[j] /= slae.mat.di[j];

found\_num = 0;

for (int i = j - 1; i >= 0; i--) {

int ggl\_ji = find\_ind\_dichotomy(slae.mat.jg,

slae.mat.ig[j], slae.mat.ig[j + 1] - 1-found\_num,

i);

if (ggl\_ji != -1) {

slae.sol[i] -= slae.mat.ggl[ggl\_ji] \* slae.sol[j];

found\_num++;

}

}

}

}

void solve\_SLAE\_direct(SLAE& slae) {

solve\_L(slae);

solve\_Lt(slae);

}

std::vector<double> mult\_mat\_CSR\_vec(MAT mat, std::vector<double> vec) {

std::vector<double> res(mat.n);

int found\_num;

for (int i = 0; i < mat.n; i++) {

found\_num = 0;

res[i] += mat.di[i] \* vec[i];

for (int j = 0; j < i; j++) {

int ggl\_ij = find\_ind\_dichotomy(mat.jg,

mat.ig[i]+found\_num, mat.ig[i + 1] - 1, j);

if (ggl\_ij != -1) {

res[i] += mat.ggl[ggl\_ij] \* vec[j];

res[j] += mat.ggl[ggl\_ij] \* vec[i];

found\_num++;

}

}

}

return res;

}

std::vector<double> vec\_sum(std::vector<double> vec1, std::vector<double> vec2, int sign) {

std::vector<double> res(vec1.size());

for (int i = 0; i < res.size(); i++) {

res[i] = sign == 1 ? vec1[i] + vec2[i] : vec1[i] - vec2[i];

}

return res;

}

double scal\_mult(std::vector<double> vec1, std::vector<double> vec2) {

double res = 0;

for (int i = 0; i < vec1.size(); i++)

res += vec1[i] \* vec2[i];

return res;

}

std::vector<double> mult\_scal\_vec(double scal, std::vector<double> vec) {

std::vector<double> res(vec.size());

for (int i = 0; i < vec.size(); i++)

res[i] = scal \* vec[i];

return res;

}

void solve\_SLAE\_CGM(SLAE& slae, int max\_iter) {

std::vector<double> r\_prev = vec\_sum(slae.vec, mult\_mat\_CSR\_vec(slae.mat, slae.sol),-1);

std::vector<double> r\_cur = r\_prev;

SLAE Mr\_prev\_slae(factor(slae.mat), slae.mat.n);

SLAE Mr\_cur\_slae = Mr\_prev\_slae;

Mr\_prev\_slae.vec = r\_prev;

solve\_SLAE\_direct(Mr\_prev\_slae);

std::vector<double> z = Mr\_prev\_slae.sol;

std::vector<double> Az;

double a, b;

double f\_norm = sqrt(scal\_mult(slae.vec, slae.vec));

double r\_cur\_norm;

int iter = 0;

while(true) {

iter++;

Az = mult\_mat\_CSR\_vec(slae.mat, z);

a = scal\_mult(Mr\_prev\_slae.sol, r\_prev) / scal\_mult(Az, z);

slae.sol = vec\_sum(slae.sol, mult\_scal\_vec(a, z), 1);

r\_cur = vec\_sum(r\_prev, mult\_scal\_vec(a, Az), -1);

r\_cur\_norm = sqrt(scal\_mult(r\_cur, r\_cur));

if (r\_cur\_norm / f\_norm <= slae.epsilon or iter == max\_iter)

break;

Mr\_cur\_slae.vec = r\_cur;

Mr\_cur\_slae.clear\_sol();

solve\_SLAE\_direct(Mr\_cur\_slae);

b = scal\_mult(Mr\_cur\_slae.sol, r\_cur) / scal\_mult(Mr\_prev\_slae.sol, r\_prev);

z = vec\_sum(Mr\_cur\_slae.sol, mult\_scal\_vec(b, z), 1);

r\_prev = r\_cur;

Mr\_prev\_slae.sol = Mr\_cur\_slae.sol;

}

}

int main() {

AREA area;

double epsilon;

int max\_iter;

input(area, epsilon, max\_iter, "nodes.txt", "elems.txt", "first\_bound\_cond.txt", "second\_bound\_cond.txt", "third\_bound\_cond.txt",

"slae\_params.txt");

MAT mat(area.nodes.size());

portrait(mat, area);

SLAE slae(mat, mat.n, epsilon);

for (ELEM& elem : area.grid) {

solve\_local(elem);

add\_to\_global(slae, elem);

}

for (ELEM& edge : area.third\_bound\_cond)

third\_bound\_cond(slae, edge);

for (ELEM& edge : area.second\_bound\_cond)

second\_bound\_cond(slae, edge);

for (ELEM& edge : area.first\_bound\_cond)

first\_bound\_cond(slae, edge);

solve\_SLAE\_CGM(slae, max\_iter);

std::vector<double> mass\_center\_num\_vals(area.grid.size());

for (int i=0; i<area.grid.size(); i++)

mass\_center\_num\_vals[i] = mass\_center\_num\_val(slae.sol, area.grid[i]);

output(area, slae.sol, "res.txt");

}

.