# Министерство науки и высшего образования Российской Федерации

## Федеральное государственное бюджетное образовательное учреждение высшего образования «Новосибирский государственный технический университет»

## C:\Users\kimev\AppData\Local\Temp\7zO0E6F35D7\4 ЦВЕТ (1).png

Кафедре прикладной математики

Курсовой проект

по дисциплине «Численные методы»

Вариант 43

|  |  |  |
| --- | --- | --- |
| https://psv4.userapi.com/c848132/u275429460/docs/d16/7a48be4e8485/Logo1.png?extra=xygsIQQwad0cqCKIxugjRIy9kd3JmRBfEKfWxcAvrfIKQ1CgI-Mr8IGFc60Gm-UtH3C0t8N-ws0mwmk1iKBnRdnqgFaqqJOX8cT3bxo_ADKOGBCCZ_VicmoU5cJzeVveeKnmjlfBib7DlanMmJKw6Dk2pw | Факультет: | ПМИ |
| Группа: | ПМ-72 |
| Студент: | Ким С.Е. |
| Преподаватель: | Патрушев И. И. |
| Персова М. Г. |

Новосибирск

2019

1. Задание

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

1. Постановка задачи

Эллиптическая краевая задача для функции определяется дифференциальным уравнением:

|  |  |
| --- | --- |
|  | (1) |

Заданным в некоторой области с границей , и краевыми условиями:

|  |  |
| --- | --- |
|  | (2) |
|  | (3) |
|  | (4) |

Для краевой задачи в декартовой системе координат уравнение представимо в виде:

|  |  |
| --- | --- |
|  | (5) |

1. Теоретическая часть
   1. Вариационная постановка

Выполним вариационную постановку методом Бубнова-Галёркина. Введём гильбертово пространство . Тогда скалярное произведение и норма определены:

|  |  |
| --- | --- |
|  | (6) |
|  | (7) |

В общем виде постановка Бубнова-Галёркина для операторного уравнения представима в виде:

|  |  |
| --- | --- |
|  | (8) |

Для уравнения постановка примет вид:

|  |  |
| --- | --- |
|  | (9) |

Далее применяя формулу Грина получим выражение:

|  |  |
| --- | --- |
|  | (10) |

где .

Подставим в формулу (10) вторые и третьи краевые условия (3), (4):

|  |  |
| --- | --- |
|  | (11) |

Так как первые краевые условия не определяют значение , слагаемое необходимо исключить из уравнения (11), взамен потребовав, чтобы пространство пробных функций содержало только функции, которые бы принимали бы нулевые значения на границе . Поэтому в качестве пространства пробных функций возьмем .

В пространстве выделим конечномерное подпространство, которое определяется как линейное пространство, натянутое на базисные функции .

Заменим функцию аппроксимирующей ее функцией , а функцию - функцией , то получим аппроксимацию уравнения Галеркина:

|  |  |
| --- | --- |
|  | (12) |

Так как представима в виде линейной комбинации:

|  |  |
| --- | --- |
|  | (13) |

Уравнение (12) эквивалентно следующей системе уравнений:

|  |  |
| --- | --- |
|  | (14) |

Так же представим в виде:

|  |  |
| --- | --- |
|  | (15) |

Причём компонент вектора весов могут быть фиксированы и определены из первого краевого условия . Подставим разложение (15) в систему (14) и получим СЛАУ для компонентвектора весов :

|  |  |
| --- | --- |
|  | (16) |

Таким образом СЛАУ может записана в виде , где:

|  |  |
| --- | --- |
|  | (17) |
|  |

* 1. Конечно элементная дискретизация

Чтобы реализовать МКЭ для данной задачи, разобьём область на прямоугольные конечные элементы .

Введем на них двумерные базисные биквадратичные функции:

|  |  |
| --- | --- |
|  | (18) |

Где – одномерные биквадратичные базисные функции.

|  |  |
| --- | --- |
|  | (19) |
|  |

Одномерные базисные функции получены из шаблонных базисных биквадратичных функций:

|  |  |
| --- | --- |
|  | (20) |
|  | (21) |

Введём матрицу массы:

|  |  |
| --- | --- |
|  | (22) |
|  | (23) |

Таким образом СЛАУ (17) будет иметь вид:

|  |  |
| --- | --- |
|  | (24) |
|  |

По заданию так же необходимо разложить по билинейным базисным функциям.

|  |  |
| --- | --- |
|  | (25) |

Где – одномерные билинейные базисные функции.

|  |  |
| --- | --- |
|  | (26) |
|  |

Одномерные базисные функции получены из шаблонных базисных билинейных функций:

|  |  |
| --- | --- |
|  | (27) |

После подстановки в СЛАУ (24) получаем:

|  |  |
| --- | --- |
|  | (28) |
|  |

* 1. Локальная матрица, локальные матрицы массы и жёсткости

Так как функция раскладывается по биквадратичному базису, локальная матрица массы и жёсткости имеет размерность .

Локальные массы и жёсткости можно получить из одномерных[[1]](#footnote-1) матриц массы и жёсткости, следующим образом:

|  |  |
| --- | --- |
|  | (29) |

Так как одномерные биквадратичные и билинейные базисные шаблоны одинаковые по , локальную одномерную матрицу жёсткости можно выразить:

|  |  |
| --- | --- |
|  | (30) |
|  |
|  | (31) |
|  |

Так же зададим матрицы:

– отображение матрицы относительно побочной диагонали;

– отображение матрицы относительно побочной диагонали;

– определяет значение параметра в угловых узлах конечного элемента

Тогда локальная двумерная матрица жёсткости будет выглядеть:

|  |  |
| --- | --- |
|  | (32) |

Матрица массы без учёта имеет вид:

|  |  |
| --- | --- |
|  | (33) |
|  |

Тогда, локальная двумерная матрица массы представима в виде:

|  |  |
| --- | --- |
|  | (34) |
|  |

И в итоге локальная матрица будет иметь вид:

|  |  |
| --- | --- |
|  | (35) |

А локальная правая часть:

|  |  |
| --- | --- |
|  | (36) |

Таким образом глобальная матрица будет иметь вид, без учёта краевых:

|  |  |
| --- | --- |
|  | (37) |
|  |

* 1. Учёт краевых условий
     1. Учёт первых краевых условий

Так как первое краевое условие фактически означает, что мы знаем значение функции в узле. По этой причине, чтобы не терять симметричность матрицы мы, поставим большое число на диагональный элемент, соответствующий этому узлу, и в правой части поставим значение первого краевого умноженное на это же большое число.

* + 1. Учёт вторых краевых условий

Рассмотрим краевые условия второго рода (3):

Для учёта вторых условий вычислим интеграл:

|  |  |
| --- | --- |
|  | (38) |

Разложим по базисным функциям:

|  |  |
| --- | --- |
|  | (39) |

Где значение функции в точке, где соответствующая базисная функция принимает 1.

И учёт вторых краевых будет представлен в виде:

|  |  |
| --- | --- |
|  | (40) |

* + 1. Учёт третьих краевых условий

Рассмотрим краевые условия третьего рода (4):

Тогда для учета третьего краевого условия нужно вычислить интегралы:

|  |  |
| --- | --- |
|  | (41) |
|  |

Будем считать, что параметры и постоянные, тогда параметр :

|  |  |
| --- | --- |
|  | (42) |

Который будем раскладывать по двум базисным функциям, определённым на этом ребре.

Тогда добавления к глобальной матрице будут представлены в виде:

|  |  |
| --- | --- |
|  | (43) |
|  |

* 1. Алгоритм формирования глобальной матрицы

Глобальную матрицу по заданию необходимо хранить в разряженном формате. Она состоит из элементов локальной матрицы.

Для начала нужно сформировать портрет матрицы, т.е. массивы и для разряженного формата. Для этого был выбран метод формирования с помощью списка смежностей.

Необходимо сформировать список смежности узлов друг с другом, с учетом того что портрет матрицы будет симметричным.

Добавим для в список смежности только те номера, которые меньше и находятся в тех же элементах что и . Затем с помощью этого списка смежности сформируем и , таким образом: элемент равен количеству элементов в списке смежности для и количеству элементов в списке смежности находящихся до ; в массив заносятся все элементы списка смежности.

Когда портрет сформировали, можно заносить локальные элементы в соответствии с глобальной нумерацией. Краевые условия задаются списком узлов для 1 краевых, списком узлов и номером функции для 2 краевых, и для третьих списком узлов, номером функции и значением коэффициента.

Полученную систему линейных алгебраических уравнений будем решать методом ЛОС.

Сетка храниться в виде двух списков, первый список узлов конечного элемента в глобальной нумерации, второй координаты вершин, в порядке слева на право, снизу-вверх.

1. Описание разработанной программы
   1. Структура данных, используемая для задания расчётной области, и конечной элементной сетки

Для задания расчетной области и конечной элементной сетки в программе используются следующие структуры:

1. Описания локальной области, содержит номера узлов и номер области

struct numberNode

{

int x[9];

int num\_sqr;

};

1. Координаты узлов

struct point

{

double x;

double y;

point();

};

1. Структура первых краевых условий

struct firstBoundCondition

{

int el;

double value;

};

1. Структура вторых краевых условий

struct secondBoundCondition

{

int nodes[3];

int numFunction;

};

1. Структура третих краевых условий

struct thirdBoundCondition

{

int nodes[3];

int numFunction;

double b;

};

1. Узлы хранятся в массиве xy со структурой point
2. Хранение конечных элементов происходит в массиве nvtr типа numberNode
3. Первые краевые условия хранятся в массиве nvk1, с типом firstBoundCondition
4. Вторые краевые условия хранятся в массиве nvk2, с типом secondBoundCondition
5. Первые краевые условия хранятся в массиве nvk3, с типом thirdBoundCondition

Для хранения, вывода результатов и внешнего ввода дробления сетки, расчётной области, конечной элементной сетки, и параметров для ЛОС используются файлы:

|  |  |  |
| --- | --- | --- |
| **Имя файла** | **Предназначение** | **Способ задания** |
| **x.txt** | Формирование сетки по оси | Количество значений  Перечисление |
| **y.txt** | Формирование сетки по оси | Количество значений  Перечисление |
| **hxhy.txt** | Дробление сетки, по итогу дробления получаться файлы xy.txt, nvtr.txt, nvk.txt | Водиться число – количество дроблений для интервала.  Водиться число – количество дроблений для интервала. |
| **xy.txt** | Узлы сетки | Первая строка – число узлов, последующие – их координаты. |
| **nvtr.txt** | Конечные элементы | Первая строка – число конечных элементов, далее номера узлов, номер области. |
| **nvk.txt** | Первые краевые | Номер узла, на котором задано условие и значение краевого условия в этом узле. |
| **nvk1.txt** | Вторые краевые | Три номера узла, задающих грань, и номер функции |
| **nvk2.txt** | Третьи краевые | Три номера узла, задающих грань, номер функции , значение |
| **options.txt** | Настройки для ЛОС | Количество итераций  Точность решения |
| **out.txt** | Результат работы программы | Выводятся координаты узла и значение в этом узле |

* 1. Структура основных модулей программы

1. FEM **-** Основной модуль программы, содержит описание и реализацию основного класса программы, основные и вспомогательные типы данных для представления расчетной области и конечной элементной сетки, все необходимые данные, такие как локальные матрицы.
   1. FEM(string direct); - процедура создания класса, чтение всех необходимых данных
   2. void solveSystem(); - процедура сборки глобальной матрицы, учета краевых условий, решение СЛАУ
   3. void output(string dir); - процедура вывода результата решения
   4. void initMatrix(); - процедура создания портрета
   5. double basicFunc(double x, double xpStart, double xpMid, double xpEnd, int num); - функция расчёта билинейной базисной функции в данных точках
   6. void initGM(); - процедура выполняющая инициализацию одномерных локальных матриц жесткости и массы
   7. int globalIndex(int elem, int num); - функция возвращающая глобальный номер по локальному и элементу
   8. double solveInPoint(double x, double y); - функция выдающая решение в заданной точке
   9. int getNumberElement(double x, double y); - функция возвращающая номер элемента в котором находиться данная точка
   10. double lambda(int elem, double x, double y); - функция расчета параметров уравнения в точке элемента
   11. double gamma(int elem, double x, double y); - функция расчета параметров уравнения в точке элемента
   12. double function(double x, double y); - функция вычисляющая значение правой части в данной точке
   13. double funcsKr2(int number, double x, double y); - функция вычисляющая значение параметра второго краевого условия в точке
   14. double funcsKr3(int number, double x, double y); - функция вычисляющая значение параметра третьего краевого условия в точке
   15. void calcFLocal(int elem); - процедура расчета локального вектора правой части
   16. void calcGM(int elem); - процедура расчета локальной двумерной матрицы жесткости и массы для данного элемента
   17. double bisquar(int def\_elem, int number, double x, double y); - функция расчета двумерного биквадратичной базисной функции для элемента.
   18. void addElem(int i, int j, double a); - процедура добавления к глобальной матрице в элемент
   19. void ukr1(int number); - процедуры учета краевых условий первого рода
   20. void ukr2(int number); - процедуры учета краевых условий второго рода
   21. void ukr3(int number); - процедуры учета краевых условий третьего рода
   22. double GLocal[9][9]; - матрица для хранения локальной матрицы жесткости
   23. double MLocal[9][9]; - матрица для хранения локальной матрицы массы
   24. double Gxy[3][3]; - матрица для хранения локальной одномерной матрицы жесткости
   25. double M[3][3]; - матрица для хранения локальной одномерной матрицы массы
   26. double Mxy[3][3]; - матрица для хранения локальной одномерной матрицы массы с учетом разложения параметра уравнения .
   27. double fLocal[9]; - локальный вектор правой части функции
   28. double bLocal[9]; **-** локальный вектор правой части функции разложенной по базисным функциям
   29. int\* ig, \* jg;
   30. double\* ggl, \* ggu, \* di;

- массивы для хранения матрицы в разряженном формате

* 1. double\* b; - вектор правой части
  2. int size\_gg; - количество элементов матрицы в массиве ggl или ggu
  3. int nk1; - количество первых краевых условий
  4. int nk2; - количество вторых краевых условий
  5. int nk3; - количество третьих краевых условий
  6. int nuz; - количество узлов
  7. int nel; - количество элементов

1. LosSolver - модуль программы, обеспечивающий решение СЛАУ в разряженном строчно-столбцовом формате методом локально оптимальной схемы
   1. LOS(int locn, int\* lig, int\* ljg, vectorD ld, vectorD ggu, vectorD ggl, vectorD lb); - процедура создания класса LOS
   2. void multyMatrixVector(vectorD x, vectorD res); - умножение матрицы на вектор , результат записывается в
   3. double scal(vectorD x, vectorD y); - скалярное произведение двух векторов
   4. double norm(vectorD a); - норма вектора
   5. void solve(); - процедура решения СЛАУ, решение записывается в вектор
2. fragmentationGrid - модуль программы, обеспечивающий генерацию сетки
   1. int \*ihx, \*ihy; - количество элементов на сколько нужно разбить промежутки
   2. int howx; - вертикальные асимптоты по
   3. int howy; - вертикальные асимптоты по
   4. int xsize; - переменная в которой храниться итоговое количество получившихся узлов по оси
   5. int ysize; - переменная в которой храниться итоговое количество получившихся узлов по оси
   6. double hx; double hy;переменная для хранения шага между промежутками
   7. double\* xw; - вспомогательный массив для формирования сетки по оси
   8. double\* yw; - вспомогательный массив для формирования сетки по оси
   9. vector<double> x; – все точки разбиения по
   10. vector<double> y; – все точки разбиения по
   11. void read(string dir); - процедура считывания входных данных
   12. double NodeValue(double x, double y); - аналитическая функция в точке, для генерации первых краевых условий на границах
   13. int K(int i); - функция необходимая для расчета глобальной нумерации
3. Тесты

Сетка для равномерных тестов имеет вид:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 20 | 21 | 22 | 23 | 24 |
| 3 | 15 | 16 | 17 | 18 | 19 |
| 2 | 10 | 11 | 12 | 13 | 14 |
| 1 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0 | 1 | 2 | 3 | 4 |
| y  x | 0 | 1 | 2 | 3 | 4 |

Сетка для неравномерных тестов имеет вид:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 | 10 | 11 | 12 | 13 | 14 |
| 3 | 5 | 6 | 7 | 8 | 9 |
| 0 | 0 | 1 | 2 | 3 | 4 |
| y  x | 0 | 2 | 3 | 5 | 6 |

Формула для расчёта относительной нормы:

|  |  |
| --- | --- |
|  | (44) |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  | |  | |  | |
| 4 | 4 | | 5 | | 6 | | 7 | | 8 | |
| 3 | 3 | | 4 | | 5 | | 6 | | 7 | |
| 2 | 2 | | 3 | | 4 | | 5 | | 6 | |
| 1 | 1 | | 2 | | 3 | | 4 | | 5 | |
| 0 | 0 | | 1 | | 2 | | 3 | | 4 | |
|  | 0 | | 1 | | 2 | | 3 | | 4 | |

|  |
| --- |
|  |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 4 | 4 | 5 | 6 | 7 | 8 | | 3 | 3 | 4 | 5 | 6 | 7 | | 2 | 2 | 3 | 4 | 5 | 6 | | 1 | 1 | 2 | 3 | 4 | 5 | | 0 | 0 | 1 | 2 | 3 | 4 | |  | 0 | 1 | 2 | 3 | 4 | |

|  |
| --- |
|  |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 4 | 0 | 0 | 0 | 0 | 0 | | 3 | 0 | 0 | 0 | 0 | 0 | | 2 | 0 | 0 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | |  | 0 | 1 | 2 | 3 | 4 | |

Относительная норма: 0.0

Комментарий: Так как данное уравнение полностью представимо в базисе, то погрешность практически отсутствует. (выводилось 12 знаков в экспоненциальном формате)

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  | |  | |  | |
| 4 | 4 | | 5 | | 6 | | 7 | | 8 | |
| 3 | 3 | | 4 | | 5 | | 6 | | 7 | |
| 2 | 2 | | 3 | | 4 | | 5 | | 6 | |
| 1 | 1 | | 2 | | 3 | | 4 | | 5 | |
| 0 | 0 | | 1 | | 2 | | 3 | | 4 | |
|  | 0 | | 1 | | 2 | | 3 | | 4 | |

|  |
| --- |
|  |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 4 | 4 | 5 | 6 | 7 | 8 | | 3 | 3 | 4 | 5 | 6 | 7 | | 2 | 2 | 3 | 4 | 5 | 6 | | 1 | 1 | 2 | 3 | 4 | 5 | | 0 | 0 | 1 | 2 | 3 | 4 | |  | 0 | 1 | 2 | 3 | 4 | |

|  |
| --- |
|  |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 4 | 0 | 0 | 0 | 0 | 0 | | 3 | 0 | 0 | 0 | 0 | 0 | | 2 | 0 | 0 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | |  | 0 | 1 | 2 | 3 | 4 | |

Относительная норма: 0.0

Комментарий: Так как уравнение и представимы в своих базисах, то погрешность практически отсутствует.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  | |  | |  | |
| 4 | 16 | | 17 | | 20 | | 25 | | 32 | |
| 3 | 9 | | 10 | | 13 | | 18 | | 25 | |
| 2 | 4 | | 5 | | 8 | | 13 | | 20 | |
| 1 | 1 | | 2 | | 5 | | 10 | | 17 | |
| 0 | 0 | | 1 | | 4 | | 9 | | 16 | |
|  | 0 | | 1 | | 2 | | 3 | | 4 | |

|  |
| --- |
|  |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 4 | 16 | 17 | 20 | 25 | 32 | | 3 | 9 | 10 | 13 | 18 | 25 | | 2 | 4 | 5 | 8 | 13 | 20 | | 1 | 1 | 2 | 5 | 10 | 17 | | 0 | 0 | 1 | 4 | 9 | 16 | |  | 0 | 1 | 2 | 3 | 4 | |

|  |
| --- |
|  |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 4 | 0 | 0 | 0 | 0 | 0 | | 3 | 0 | 0 | 0 | 0 | 0 | | 2 | 0 | 0 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 0 | 0 | | 0 | 0 | 0 | 0 | 0 | 0 | |  | 0 | 1 | 2 | 3 | 4 | |

Относительная норма: 0.0

Комментарий: Аналогично тесту 1.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | |  | |  | |  | |  | |
| 4 | 0 | | 64 | | 128 | | 192 | | 256 | |
| 3 | 0 | | 27.01070549 | | 54.01252712 | | 81.0109238 | | 108 | |
| 2 | 0 | | 8.009361632 | | 16.01171184 | | 24.01083645 | | 32 | |
| 1 | 0 | | 1.007259581 | | 2.008606122 | | 3.008564323 | | 4 | |
| 0 | 0 | | 0 | | 0 | | 0 | | 0 | |
|  | 0 | | 1 | | 2 | | 3 | | 4 | |

|  |
| --- |
|  |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 4 | 0 | 64 | 128 | 192 | 256 | | 3 | 0 | 27 | 54 | 81 | 108 | | 2 | 0 | 8 | 16 | 24 | 32 | | 1 | 0 | 1 | 2 | 3 | 4 | | 0 | 0 | 0 | 0 | 0 | 0 | |  | 0 | 1 | 2 | 3 | 4 | |

|  |
| --- |
|  |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 4 | 0E+00 | 0E+00 | 0E+00 | 0E+00 | 0E+00 | | 3 | 0E+00 | 1E-02 | 1E-02 | 1E-02 | 0E+00 | | 2 | 0E+00 | 9E-03 | 1E-02 | 1E-02 | 0E+00 | | 1 | 0E+00 | 7E-03 | 9E-03 | 9E-03 | 0E+00 | | 0 | 0E+00 | 0E+00 | 0E+00 | 0E+00 | 0E+00 | |  | 0 | 1 | 2 | 3 | 4 | |

Относительная норма: 8.0E-05

Комментарий: Уравнение не представимо базисными функциями, поэтому решение дает приблизительный результат (аппроксимированная функция).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 6 | 36 | 40 | 45 | 61 | 72 |
| 3 | 9 | 13 | 18 | 34 | 45 |
| 0 | 0 | 4 | 9 | 25 | 36 |
|  | 0 | 2 | 3 | 5 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 6 | 36 | 40 | 45 | 61 | 72 |
| 3 | 9 | 13 | 18 | 34 | 45 |
| 0 | 0 | 4 | 9 | 25 | 36 |
|  | 0 | 2 | 3 | 5 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| 6 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
|  | 0 | 2 | 3 | 5 | 6 |

Относительная норма: 0.0

Комментарий: Аналогично тесту 1. Только на неравномерной сетке.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 6 | 0 | 2 | 3 | 5 | 6 |
| 3 | 0 | 2 | 3 | 5 | 6 |
| 0 | 0 | 2 | 3 | 5 | 6 |
|  | 0 | 2 | 3 | 5 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 6 | 0 | 2 | 3 | 5 | 6 |
| 3 | 0 | 2 | 3 | 5 | 6 |
| 0 | 0 | 2 | 3 | 5 | 6 |
|  | 0 | 2 | 3 | 5 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| 6 | 0 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 |
|  | 0 | 2 | 3 | 5 | 6 |

Относительная норма: 0.0

Комментарий: Аналогично тесту 2. Только на неравномерной сетке.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 6 | 0 | 2 | 3 | 5 | 6 |
| 3 | 0 | 1.950951412 | 3.004707771 | 4.99018826 | 6 |
| 0 | 0 | 2 | 3 | 5 | 6 |
|  | 0 | 2 | 3 | 5 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 6 | 0 | 2 | 3 | 5 | 6 |
| 3 | 0 | 2 | 3 | 5 | 6 |
| 0 | 0 | 2 | 3 | 5 | 6 |
|  | 0 | 2 | 3 | 5 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| 6 | 0E+00 | 0E+00 | 0E+00 | 0E+00 | 0E+00 |
| 3 | 0E+00 | 5E-02 | 5E-03 | 1E-02 | 0E+00 |
| 0 | 0E+00 | 0E+00 | 0E+00 | 0E+00 | 0E+00 |
|  | 0 | 2 | 3 | 5 | 6 |

Относительная норма: 0.0033

Комментарий: Коэффициент диффузии не представим в билинейном базисе, поэтому решение не точное (аппроксимированная функция).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 4 | 0 | 0.5 | 0.75 | 1.25 | 1.5 |
| 3 | 0 | 0.666666667 | 1 | 1.666666667 | 2 |
| 1 | 0 | 2 | 3 | 5 | 6 |
|  | 0 | 2 | 3 | 5 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 4 | 0 | 0.5 | 0.75 | 1.25 | 1.5 |
| 3 | 0 | 0.67018938 | 1.005179574 | 1.672676283 | 2 |
| 1 | 0 | 2 | 3 | 5 | 6 |
|  | 0 | 2 | 3 | 5 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| 4 | 0E+00 | 0E+00 | 0E+00 | 0E+00 | 0E+00 |
| 3 | 0E+00 | 4E-03 | 5E-03 | 6E-03 | 0E+00 |
| 1 | 0E+00 | 0E+00 | 0E+00 | 0E+00 | 0E+00 |
|  | 0 | 2 | 3 | 5 | 6 |

Относительная норма: 0.00093

Комментарий: Была взята функция не представимая в биквадратичном базисе, для анализа влияния краевых условий на точность решения.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 4 | 0.00 | 0.50 | 0.75 | 1.25 | 1.50 |
| 3 | 0.00 | 0.68 | 1.00 | 1.55 | 2.00 |
| 1 | 0.00 | 2.00 | 3.00 | 5.00 | 6.00 |
|  | 0 | 2 | 3 | 5 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| 4 | 3E-17 | 0E+00 | 0E+00 | 0E+00 | 0E+00 |
| 3 | 7E-17 | 1E-02 | 1E-03 | 1E-01 | 0E+00 |
| 1 | 2E-17 | 0E+00 | 0E+00 | 0E+00 | 0E+00 |
|  | 0 | 2 | 3 | 5 | 6 |

Относительная норма: 0.0125

Комментарий: 2 краевые условия были добавлены на верхней и нижней грани области, при этом первые краевые на этих гранях были удалены.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 4 | 0.00 | 0.50 | 0.75 | 1.25 | 1.50 |
| 3 | 0.00 | 0.68 | 1.01 | 1.68 | 2.00 |
| 1 | 0.00 | 2.00 | 3.00 | 5.00 | 6.00 |
|  | 0 | 2 | 3 | 5 | 6 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| 4 | 3E-17 | 0E+00 | 0E+00 | 0E+00 | 0E+00 |
| 3 | 8E-17 | 1E-02 | 8E-03 | 9E-03 | 0E+00 |
| 1 | 2E-17 | 0E+00 | 0E+00 | 0E+00 | 0E+00 |
|  | 0 | 2 | 3 | 5 | 6 |

Относительная норма: 0.0019

Комментарий: 2 краевые условия были заменены 3-ми краевыми условиями.

Вывод: Третьи краевые условия дают результат точнее, чем вторые краевые, и приблизительные по точности к первым. Однако первые являются самыми точными.

Следующие тесты будут на дробление сетки. Дробление сетки происходило путём дробления области на 4 подобласти.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 6 | 0.333333333 | 0.666666667 | 1 | 1.333333333 | 1.666666667 |
| 4 | 0.5 | 1 | 1.5 | 2 | 2.5 |
| 2 | 1 | 2 | 3 | 4 | 5 |
|  | 2 | 4 | 6 | 8 | 10 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 6 | 0.33 | 0.67 | 1.00 | 1.33 | 1.67 |
| 4 | 0.50 | 1.01 | 1.51 | 2.01 | 2.50 |
| 2 | 1.00 | 2.00 | 3.00 | 4.00 | 5.00 |
|  | 2 | 4 | 6 | 8 | 10 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| 6 | 3E-08 | 3E-08 | 0E+00 | 3E-07 | 3E-07 |
| 4 | 0E+00 | 6E-03 | 1E-02 | 1E-02 | 0E+00 |
| 2 | 0E+00 | 0E+00 | 0E+00 | 0E+00 | 0E+00 |
|  | 2 | 4 | 6 | 8 | 10 |

Относительная норма: 0.00194

Комментарий: Для исследования уменьшения погрешности путём дробления сетки, была выбрана равномерная сетка и функция не представимая биквадратичным базисом.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 6 | 0.33 | 0.67 | 1.00 | 1.33 | 1.67 |
| 4 | 0.50 | 1.00 | 1.50 | 2.00 | 2.50 |
| 2 | 1.00 | 2.00 | 3.00 | 4.00 | 5.00 |
|  | 2 | 4 | 6 | 8 | 10 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| 6 | 3E-08 | 3E-08 | 0E+00 | 3E-07 | 3E-07 |
| 4 | 0E+00 | 4E-04 | 6E-04 | 8E-04 | 0E+00 |
| 2 | 0E+00 | 0E+00 | 0E+00 | 0E+00 | 0E+00 |
|  | 2 | 4 | 6 | 8 | 10 |

Относительная норма: 0.000128

Комментарий: Относительная норма уменьшилась в ~ 15.24 раза, при первом дроблении.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 6 | 0.33 | 0.67 | 1.00 | 1.33 | 1.67 |
| 4 | 0.50 | 1.00 | 1.50 | 2.00 | 2.50 |
| 2 | 1.00 | 2.00 | 3.00 | 4.00 | 5.00 |
|  | 2 | 4 | 6 | 8 | 10 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | |  |  |  |  |
| 6 | 3E-08 | 3E-08 | 0E+00 | 3E-07 | 3E-07 |
| 4 | 0E+00 | 3E-05 | 4E-05 | 5E-05 | 0E+00 |
| 2 | 0E+00 | 0E+00 | 0E+00 | 0E+00 | 0E+00 |
|  | 2 | 4 | 6 | 8 | 10 |

Относительная норма: 8.6E-06

Комментарий: Относительная норма уменьшилась в ~ 14.85 раза, при первом дроблении.

Вывод: Погрешность при дроблении сетки падает приблизительно в 16 раз.

1. Приложение

Код программы:

Fem.h

#pragma once

#include <string>

#include <vector>

#include "LOSSolver.h"

using namespace std;

struct numberNode

{

int x[9];

int num\_sqr;

};

struct point

{

double x;

double y;

point();

};

struct firstBoundCondition

{

int el;

double value;

};

struct secondBoundCondition

{

int nodes[3];

int numFunction;

};

struct thirdBoundCondition

{

int nodes[3];

int numFunction;

double b;

};

class FEM

{

private:

static const int countBasicFuncs = 9;

void initMatrix();

double basicFunc(double x, double xpStart, double xpMid, double xpEnd, int num);

void initGM();

int globalIndex(int elem, int num);

int getNumberElement(double x, double y);

double lambda(int elem, double x, double y);

double gamma(int elem, double x, double y);

double function(double x, double y);

double funcsKr2(int number, double x, double y);

double funcsKr3(int number, double x, double y);

void calcFLocal(int elem);

void calcGM(int elem);

double bisquar(int def\_elem, int number, double x, double y);

void addElem(int i, int j, double a);

void ukr1(int number);

void ukr2(int number);

void ukr3(int number);

numberNode\* nvtr;

point\* xy;

firstBoundCondition\* nvk1;

secondBoundCondition\* nvk2;

thirdBoundCondition\* nvk3;

double GLocal[9][9];

double MLocal[9][9];

double Gxy[3][3];

double M[3][3];

double Mxy[3][3];

double fLocal[9];

double bLocal[9];

int\* ig, \* jg;

double\* ggl, \* ggu, \* di;

int size\_gg;

int nk1;

int nk2;

int nk3;

double\* b;

int nuz, nel;

public:

FEM(string direct);

void solveSystem();

double solveInPoint(double x, double y);

void output(string dir);

~FEM();

};

Fem.cpp

#include "FEM.h"

point::point()

{

x = 0;

y = 0;

}

FEM::FEM(string direct)

{

string dir = direct + "/";

FILE\* f;

myfopen(&f, dir + "xy.txt", "r");

fscanf\_s(f, "%d", &nuz);

xy = new point[nuz];

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

{

fscanf\_s(f, "%lf%lf", &xy[i].x, &xy[i].y);

}

fclose(f);

myfopen(&f, dir + "nvtr.txt", "r");

fscanf\_s(f, "%d", &nel);

nvtr = new numberNode[nel];

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

{

for (int j = 0; j < 9; ++j) {

fscanf\_s(f, "%d", &nvtr[i].x[j]);

}

}

fclose(f);

myfopen(&f, dir + "nvk.txt", "r");

fscanf\_s(f, "%d", &nk1);

nvk1 = new firstBoundCondition[nk1];

for (int i = 0; i < nk1; ++i)

fscanf\_s(f, "%d%lf", &nvk1[i].el, &nvk1[i].value);

fclose(f);

myfopen(&f, dir + "nvk1.txt", "r");

fscanf\_s(f, "%d", &nk2);

nvk2 = new secondBoundCondition[nk2];

for (int i = 0; i < nk2; ++i)

{

fscanf\_s(f, "%d%d%d%d", &nvk2[i].nodes[0], &nvk2[i].nodes[1], &nvk2[i].nodes[2], &nvk2[i].numFunction);

}

fclose(f);

myfopen(&f, dir + "nvk2.txt", "r");

fscanf\_s(f, "%d", &nk3);

nvk3 = new thirdBoundCondition[nk3];

for (int i = 0; i < nk3; ++i)

fscanf\_s(f, "%d%d%d%d%lf", &nvk3[i].nodes[0], &nvk3[i].nodes[1], &nvk3[i].nodes[2], &nvk3[i].numFunction, &nvk3[i].b);

fclose(f);

b = new double[nuz];

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

b[i] = 0;

initMatrix();

initGM();

}

void FEM::solveSystem()

{

double g = gamma(0, 0, 0);

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

{

calcGM(i);

for (int k = 0; k < 9; ++k)

{

for (int j = 0; j < 9; ++j)

{

addElem(nvtr[i].x[k], nvtr[i].x[j], GLocal[k][j] + g \* MLocal[k][j]);

}

}

calcFLocal(i);

for (int k = 0; k < 9; ++k)

{

b[nvtr[i].x[k]] += bLocal[k];

}

}

for (int i = 0; i < nk2; ++i)

{

ukr2(i);

}

for (int i = 0; i < nk3; ++i)

{

ukr3(i);

}

for (int i = 0; i < nk1; ++i)

{

ukr1(i);

}

LOS solver(nuz, ig, jg, di, ggu, ggl, b);

solver.solve();

}

void FEM::output(string dir)

{

FILE\* f;

myfopen(&f, dir + "/out.txt", "w");

for (int i = 0; i < nuz; ++i) {

fprintf(f, "%lf %lf %.12le\n", xy[i].x, xy[i].y, b[i]);

}

fclose(f);

}

FEM::~FEM()

{

delete[]xy;

delete[]nvtr;

delete[]nvk1;

delete[]nvk2;

delete[]nvk3;

delete[]b;

delete[]ig;

delete[]jg;

delete[]ggl;

delete[]ggu;

delete[]di;

}

void FEM::initMatrix()

{

di = new double[nuz];

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

{

di[i] = 0;

}

ig = new int[nuz + 1];

vector<vector<int>> list(nuz);

int listsize = 0;

for (int elem = 0; elem < nel; elem++)

{

for (int i = 0; i < countBasicFuncs; ++i) {

int k = nvtr[elem].x[i];

for (int j = i + 1; j < countBasicFuncs; j++) {

int ind1 = k;

int ind2 = nvtr[elem].x[j];

if (ind2 < ind1) {

ind1 = ind2;

ind2 = k;

}

int iaddr = list[ind2].size();

if (iaddr == 0) {

listsize++;

list[ind2].push\_back(ind1);

}

else

{

int l = 0;

for (l = 0; l < iaddr && list[ind2][l] < ind1; l++);

if (l == iaddr)

{

listsize++;

list[ind2].push\_back(ind1);

}

else

{

if (list[ind2][l] > ind1) {

listsize++;

list[ind2].emplace(list[ind2].begin() + l, ind1);

}

}

}

}

}

}

jg = new int[listsize];

ig[0] = 0;

size\_gg = listsize;

int size\_sum = 0;

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

{

int size = list[i].size();

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

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

jg[size\_sum + j] = list[i][j];

size\_sum += size;

}

ggl = new double[listsize];

ggu = new double[listsize];

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

{

ggl[i] = 0;

ggu[i] = 0;

}

list.~vector();

}

double FEM::funcsKr2(int numFunc, double x, double y)

{

switch (numFunc)

{

case 1: {return x / (y \* y); }

case 2: {return -x / (y \* y); }

}

}

double FEM::funcsKr3(int numFunc, double x, double y)

{

switch (numFunc)

{

case 1: {return x / y - x / (y \* y); }

case 2: {return x / y + x / (y \* y); }

}

}

void FEM::ukr1(int number)

{

int elem = nvk1[number].el;

di[elem] = 1e+16;

b[elem] = nvk1[number].value \* 1e+16;

}

void FEM::ukr2(int number)

{

int\* uzel = nvk2[number].nodes;

double h = xy[uzel[2]].y - xy[uzel[0]].y;

if (h == 0)

{

h = xy[uzel[2]].x - xy[uzel[0]].x;

}

double buff[3];

for (int i = 0; i < 3; ++i) {

buff[i] = 0;

for (int j = 0; j < 3; j++)

buff[i] += funcsKr2(nvk2[number].numFunction, xy[uzel[j]].x, xy[uzel[j]].y) \* Mxy[i][j];

b[uzel[i]] += buff[i] \* h / 30.;

}

}

void FEM::ukr3(int number)

{

int\* uzel = nvk3[number].nodes;

double h = xy[uzel[2]].y - xy[uzel[0]].y;

double beta = nvk3[number].b;

if (h == 0) h = xy[uzel[2]].x - xy[uzel[0]].x;

double buff[3];

for (int i = 0; i < 3; ++i) {

buff[i] = 0;

for (int j = 0; j < 3; j++)

{

buff[i] += funcsKr3(nvk3[number].numFunction, xy[uzel[j]].x, xy[uzel[j]].y) \* Mxy[i][j];

}

b[uzel[i]] += buff[i] \* h \* beta / 30.;

}

for (int k = 0; k < 3; k++)

{

for (int j = 0; j < 3; j++)

{

addElem(nvk3[number].nodes[k], nvk3[number].nodes[j], Mxy[k][j] \* beta \* h / 30.0);

}

}

}

void FEM::initGM()

{

Gxy[0][0] = 11.; Gxy[0][1] = -12.; Gxy[0][2] = 1.;

Gxy[1][0] = -12.; Gxy[1][1] = 16.; Gxy[1][2] = -4.;

Gxy[2][0] = 1.; Gxy[2][1] = -4.; Gxy[2][2] = 3.;

Mxy[0][0] = 7.; Mxy[0][1] = 4.; Mxy[0][2] = -1.;

Mxy[1][0] = 4.; Mxy[1][1] = 16.; Mxy[1][2] = 0;

Mxy[2][0] = -1.; Mxy[2][1] = 0.; Mxy[2][2] = 1.;

M[0][0] = 4.; M[0][1] = 2.; M[0][2] = -1.;

M[1][0] = 2.; M[1][1] = 16.; M[1][2] = 2.;

M[2][0] = -1.; M[2][1] = 2.; M[2][2] = 4.;

}

double FEM::basicFunc(double x, double xpStart, double xpMid, double xpEnd, int num)

{

double hx1 = xpMid - xpStart;

double hx2 = xpEnd - xpStart;

double hx3 = xpEnd - xpMid;

switch (num)

{

case 0: {return((xpMid - x) \* (xpEnd - x)) / (hx1 \* hx2); }

case 1: {return((x - xpStart) \* (xpEnd - x)) / (hx1 \* hx3); }

case 2: {return((x - xpStart) \* (x - xpMid)) / (hx2 \* hx3); }

}

}

int FEM::globalIndex(int elem, int num)

{

return nvtr[elem].x[num];

}

double FEM::solveInPoint(double x, double y)

{

double u = 0;

int numElem = getNumberElement(x, y);

for (int i = 0; i < countBasicFuncs; ++i) {

u += b[globalIndex(numElem, i)] \* bisquar(numElem, i, x, y);

}

return u;

}

int FEM::getNumberElement(double x, double y)

{

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

{

int x0 = globalIndex(i, 0), x8 = globalIndex(i, 8);

if (xy[x0].x <= x && xy[x0].y <= y && xy[x8].x >= x && xy[x8].y >= y)

return i;

}

return -1;

}

double FEM::lambda(int elem, double x, double y)

{

return 1;

//return x + y;

//return x \* x;

}

double FEM::gamma(int elem, double x, double y)

{

return 1;

}

double FEM::function(double x, double y)

{

//TEST1

//return x + y;

//return x + y - 2;

//TEST2

//return x \* x + y \* y - 4;

//return x \* x + y \* y - 4 \* x - 2 \* y;

//TEST3

//return -y \* y \* y - 6 \* x \* x \* y - 9 \* y \* y \* x + x \* y \* y \* y;

//TEST4

//return -x;

//TEST5

return x / y - 2 \* x / (y \* y \* y);

}

void FEM::calcFLocal(int elem)

{

int node;

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

{

node = globalIndex(elem , i);

fLocal[i] = function(xy[node].x, xy[node].y);

}

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

{

bLocal[i] = 0;

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

{

bLocal[i] += MLocal[i][j] \* fLocal[j];

}

}

}

void FEM::calcGM(int elem)

{

int xy0 = globalIndex(elem, 0);

int xy2 = globalIndex(elem, 2);

int xy6 = globalIndex(elem, 6);

int xy8 = globalIndex(elem, 8);

double hx = xy[xy8].x - xy[xy0].x;

double hy = xy[xy8].y - xy[xy0].y;

double lk0k0 = lambda(elem, xy[xy0].x, xy[xy0].y);

double lk1k0 = lambda(elem, xy[xy2].x, xy[xy2].y);

double lk0k1 = lambda(elem, xy[xy6].x, xy[xy6].y);

double lk1k1 = lambda(elem, xy[xy8].x, xy[xy8].y);

int nui, mui, nuj, muj;

for (int i = 0; i < countBasicFuncs; ++i) {

nui = (i) % 3;

mui = (i) / 3;

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

{

nuj = (j) % 3;

muj = (j) / 3;

GLocal[i][j] = ((hy / hx) \* (lk0k0 \* Gxy[nui][nuj] \* M[mui][muj] + lk1k0 \* Gxy[2 - nui][2 - nuj] \* M[mui][muj]

+ lk0k1 \* Gxy[nui][nuj] \* M[2 - mui][2 - muj] + lk1k1 \* Gxy[2 - nui][2 - nuj] \* M[2 - mui][2 - muj])

+ (hx / hy) \* (lk0k0 \* Gxy[mui][muj] \* M[nui][nuj] + lk1k0 \* Gxy[2 - mui][2 - muj] \* M[nui][nuj]

+ lk0k1 \* Gxy[mui][muj] \* M[2 - nui][2 - nuj] + lk1k1 \* Gxy[2 - mui][2 - muj] \* M[2 - nui][2 - nuj])) / 360.;

MLocal[i][j] = hx \* hy \* (M[mui][muj] \* M[nui][nuj]) / 900.;

}

}

}

double FEM::bisquar(int elem, int num, double x, double y)

{

int xySt = globalIndex(elem, 0);

int xyMid = globalIndex(elem, 4);

int xyEnd = globalIndex(elem, 8);

double xpEnd = xy[xyEnd].x, xpSt = xy[xySt].x, xpMid = xy[xyMid].x;

double ypEnd = xy[xyEnd].y, ypSt = xy[xySt].y, ypMid = xy[xyMid].y;

switch (num)

{

case 0: {return basicFunc(x, xpSt, xpMid, xpEnd, 0) \* basicFunc(y, ypSt, ypMid, ypEnd, 0); }

case 1: {return basicFunc(x, xpSt, xpMid, xpEnd, 1) \* basicFunc(y, ypSt, ypMid, ypEnd, 0); }

case 2: {return basicFunc(x, xpSt, xpMid, xpEnd, 2) \* basicFunc(y, ypSt, ypMid, ypEnd, 0); }

case 3: {return basicFunc(x, xpSt, xpMid, xpEnd, 0) \* basicFunc(y, ypSt, ypMid, ypEnd, 1); }

case 4: {return basicFunc(x, xpSt, xpMid, xpEnd, 1) \* basicFunc(y, ypSt, ypMid, ypEnd, 1); }

case 5: {return basicFunc(x, xpSt, xpMid, xpEnd, 2) \* basicFunc(y, ypSt, ypMid, ypEnd, 1); }

case 6: {return basicFunc(x, xpSt, xpMid, xpEnd, 0) \* basicFunc(y, ypSt, ypMid, ypEnd, 2); }

case 7: {return basicFunc(x, xpSt, xpMid, xpEnd, 1) \* basicFunc(y, ypSt, ypMid, ypEnd, 2); }

case 8: {return basicFunc(x, xpSt, xpMid, xpEnd, 2) \* basicFunc(y, ypSt, ypMid, ypEnd, 2); }

};

return 0;

}

void FEM::addElem(int i, int j, double a)

{

if (i == j)

{

di[i] += a;

}

else

{

if (i < j)

{

int ind;

for (ind = ig[j]; ind < ig[j + 1]; ++ind)

{

if (jg[ind] == i) break;

}

ggu[ind] = ggu[ind] + a;

}

else

{

int ind;

for (ind = ig[i]; ind < ig[i + 1]; ++ind)

{

if (jg[ind] == j)

break;

}

ggl[ind] = ggl[ind] + a;

}

}

}

LOSSolver.h

#pragma once

#include <cstdio>

#include <cmath>

#define myfopen(file, path, mode) fopen\_s(file, (path).c\_str(), mode)

typedef double\* vectorD;

class LOS {

private:

double eps;

int \*ig, \*jg;

vectorD di, gu, gl;

vectorD b;

vectorD x;

vectorD mv;

vectorD z;

vectorD r;

vectorD p;

vectorD diag;

int n, maxiter;

public:

void multyMatrixVector(vectorD x, vectorD res);

double scal(vectorD x, vectorD y);

double norm(vectorD a);

void solve();

LOS(int locn, int\* lig, int\* ljg, vectorD ld, vectorD ggu, vectorD ggl, vectorD lb);

~LOS();

};

LOSSolver.cpp

#include "LOSSolver.h"

void LOS::multyMatrixVector(vectorD x, vectorD res)

{

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

int gi = ig[i], gi\_1 = ig[i + 1];

res[i] = di[i] \* x[i];

for (int j = gi; j < gi\_1; ++j) {

int column = jg[j];

res[i] += gl[j] \* x[column];

res[column] += gu[j] \* x[i];

}

}

}

double LOS::scal(vectorD a, vectorD b)

{

double s = 0.0;

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

s += a[i] \* b[i];

return s;

}

double LOS::norm(vectorD a)

{

return sqrt(scal(a, a));

}

LOS::LOS(int locn, int\* lig, int\* ljg, vectorD ld, vectorD ggu, vectorD ggl, vectorD lb)

{

FILE\* f;

fopen\_s(&f, "options.txt", "r");

fscanf\_s(f, "%d%lf", &maxiter, &eps);

n = locn;

ig = lig;

jg = ljg;

di = ld;

gl = ggl; gu = ggu;

b = lb;

mv = new double[n];

z = new double[n];

r = new double[n];

p = new double[n];

x = new double[n];

diag = new double[n];

}

LOS::~LOS()

{

delete[] mv;

delete[] z;

delete[] r;

delete[] p;

delete[] x;

delete[] diag;

}

void LOS::solve()

{

int count = 0;

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

{

x[i] = 1;

}

multyMatrixVector(x, mv);

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

{

r[i] = b[i] - mv[i];

z[i] = r[i];

}

multyMatrixVector(z, p);

double sr = scal(r, r);

while (sr > eps&& count <= maxiter)

{

double pp = scal(p, p);

double ak = scal(p, r) / pp;

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

{

x[i] = x[i] + ak \* z[i];

r[i] = r[i] - ak \* p[i];

}

multyMatrixVector(r, mv);

double bk = -scal(p, mv) / pp;

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

{

z[i] = r[i] + bk \* z[i];

p[i] = mv[i] + bk \* p[i];

}

sr = sqrt(scal(r, r));

++count;

}

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

{

b[i] = x[i];

}

}

fragmentationGrid.h

#pragma once

#include <vector>

#include <string>

using namespace std;

class fragmentationGrid

{

int \*ihx, \*ihy;

int howx;

int howy;

int xsize;

int ysize;

double hx; double hy;

double\* xw;

double\* yw;

vector<double> x;

vector<double> y;

void read(string dir);

double NodeValue(double x, double y);

int K(int i);

public:

void process(string dir);

};

fragmentationGrid.cpp

#include "fragmentationGrid.h"

void fragmentationGrid::read(string dir)

{

FILE\* f;

fopen\_s(&f, (dir + "/x.txt").c\_str(), "r");

fscanf\_s(f, "%d", &howx);

xw = new double[howx];

ihx = new int[howx - 1];

for (int i = 0; i < howx; ++i) {

fscanf\_s(f, "%lf", &xw[i]);

}

fclose(f);

fopen\_s(&f, (dir + "/y.txt").c\_str(), "r");

fscanf\_s(f, "%d", &howy);

yw = new double[howy];

ihy = new int[howy - 1];

for (int i = 0; i < howy; ++i) {

fscanf\_s(f, "%lf", &yw[i]);

}

fclose(f);

fopen\_s(&f, (dir + "/hxhy.txt").c\_str(), "r");

for (int i = 0; i < howx - 1; ++i) {

fscanf\_s(f, "%d", &ihx[i]);

ihx[i] \*= 2;

hx = (xw[i + 1] - xw[i]) / ihx[i];

for (int j = 0; j < ihx[i]; ++j)

{

x.push\_back(xw[i] + j \* hx);

}

}

x.push\_back(xw[howx - 1]);

for (int i = 0; i < howy - 1; ++i) {

fscanf\_s(f, "%d", &ihy[i]);

ihy[i] \*= 2;

hy = (yw[i + 1] - yw[i]) / ihy[i];

for (int j = 0; j < ihy[i]; j++)

{

y.push\_back(yw[i] + j \* hy);

}

}

y.push\_back(yw[howy - 1]);

xsize = x.size();

ysize = y.size();

fclose(f);

}

double fragmentationGrid::NodeValue(double x, double y)

{

//return x + y;

//return x \* x + y \* y;

//return x \* y \* y \* y;

return x / y;

}

void fragmentationGrid::process(string dir)

{

read(dir);

FILE\* f;

fopen\_s(&f, (dir + "/xy.txt").c\_str(), "w");

fprintf(f, "%d\n", ysize \* xsize);

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

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

fprintf(f, " %0.15lg %0.15lg\n", x[j], y[i]);

fclose(f);

fopen\_s(&f, (dir + "/nvtr.txt").c\_str(), "w");

fprintf(f, "%d\n", (xsize / 2) \* (ysize / 2));

for (int i = 0; i < (xsize / 2) \* (ysize / 2); i++) {

int k = K(i);

fprintf(f, "%d %d %d ", k, k + 1, k + 2);

fprintf(f, "%d %d %d ", k + 2 \* xsize / 2, k + 2 \* xsize / 2 + 1, k + 2 \* xsize / 2 + 2);

fprintf(f, "%d %d %d\n", k + 2 \* (2 \* xsize / 2), k + 2 \* (2 \* xsize / 2) + 1, k + 2 \* (2 \* xsize / 2) + 2);

}

fclose(f);

fopen\_s(&f, (dir + "/nvk.txt").c\_str(), "w");

fprintf(f, "%d\n", xsize \* 2 + ysize \* 2 - 4);

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

{

fprintf(f, "%d %le\n", i, NodeValue(x[i], y[0]));

}

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

{

fprintf(f, "%d %le\n", i + (2 \* xsize / 2) \* (ysize - 1), NodeValue(x[i], y[ysize - 1]));

}

for (int i = 1; i < ysize - 1; i++)

{

fprintf(f, "%d %le\n", i \* (2 \* xsize / 2), NodeValue(x[0], y[i]));

}

for (int i = 1; i < ysize - 1; i++)

{

fprintf(f, "%d %le\n", (i + 1) \* (2 \* xsize / 2) - 1, NodeValue(x[xsize - 1], y[i]));

}

fclose(f);

delete[]xw;

delete[]yw;

delete[]ihx;

delete[]ihy;

x.~vector();

y.~vector();

}

int fragmentationGrid::K(int i)

{

return 2 \* ((i) / (xsize / 2)) \* (xsize) + 2 \* (i % (xsize / 2));

}

1. Для удобства представления в программе одномерные матрицы массы и жёсткости представлены в виде двумерных матриц соответственно. [↑](#footnote-ref-1)