|  |  |  |  |
| --- | --- | --- | --- |
|  | | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | | |
|  | | | |
| Кафедра прикладной математики | | | |
|  | | | |
| Курсовой проект | | | |
| по дисциплине «Численные методы» | | | |
| **Метод конечных элементов для эллиптической задачи** | | | |
|  | | | |
|  | Факультет: | ПМИ |
| Группа: | ПМ-05 |
| Студент: | Пучков Дмитрий |
|  |  |
| Преподаватель: | Пересова Марина Геннадьевна |
|  |  |
|  | | | |
| Новосибирск | | | |
| 2022 | | | |

1. **Постановка задачи**

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

Эллиптическая краевая задача для функции определяется дифференциальным уравнением

заданным в области с границей и краевыми условиями

В нашей цилиндрической (r,z) системе координат:

1. **Теоретическая часть**
   1. **Вариационная постановка**

Запишем эквивалентную вариационную постановку в форме уравнения Галеркина.

Для этого потребуем, чтобы невязка исходного уравнения

была ортогональна пространству пробных функций , то есть

Воспользуемся формулой Грина и преобразуем интегралы по границам S2 иS3, воспользовавшись краевыми условиями. В качестве выберем - пространство функций, которые вместе со своей производной интегрируемы с квадратом на и на границе S1 удовлетворяют нулевым первым краевым условиям. При этом будем считать, что u принадлежит пространству , то есть множеству функций, имеющих суммируемые с квадратом первые производные и удовлетворяющих только первым краевым условиям на границе S1. Уравнение принимает вид:

* 1. **Аппроксимация на конечномерных пространствах и переход к локальным матрицам**

Получим конечноэлементную аппроксимацию уравнения Галеркина. Для этого нам понадобятся конечноэлементные пространства , , которые аппроксимируют и соответственно. Соответственно заменим *u*  на аппроксимирующую *uh* и *v0* на и получим:

Пусть функции из пространств и являются элементами одного и того же конечномерного пространства (пространство, аппроксимирующее *H1*). И пусть – финитные функции, являющиеся базисом *Vh*. Тогда и *uh*могут быть представлены в виде:

где – множество индексов i таких, что являются базисными функциями пространств , .

Подставляя вместо и *uh* их представления в виде линейной комбинации базисов с весами получим следующую СЛАУ для компонент вектора q с индексами :

В цилиндрической системе координат СЛАУ принимает вид:

Недостающие n-n0 уравнений для компонент вектора q с индексами могут быть получены из первого краевого условия :

Рассмотрим вид базисных функций на треугольнике. Для представления на треугольнике произвольной линейной функции необходимо три линейно не зависимые локальные функции. В качестве таких функций в нашем случае выступают барицентрические координаты треугольника. Каждая такая функция ассоциирована с определённой вершиной треугольника, пусть мы имеем {(r1, z1), (r2, z2), (r3, z3)}-треугольник, а L1(r, z), L2(r, z), L3(r, z) -его L – координаты, причем L1(r, z) = 1 в вершине треугольника (r1, z1), L2(r, z) = 1 в вершине треугольника (r2, z2), L3(r, z) = 1 в вершине треугольника (r3, z3). Будем считать, что вне данного треугольника локальные базисные функции равны 0. Вид барицентрических координат треугольника:

𝐿𝑖 = 𝛼0𝑖 + 𝛼1𝑖r + 𝛼2𝑖 z, 𝑖 = , где

Тогда базисные функции будут

𝜓1 = 𝐿1(r, z)

𝜓2 = 𝐿2(r, z)

𝜓3 = 𝐿3(r, z)

Также для барицентрических функций достаточно легко вычислить интеграл:

Локальная матрица будет представлять собой сумму матриц жёсткости и массы и будет иметь размерность 3x3 (по числу узлов на конечном элементе).

Компоненты локальных матриц жесткости и массы имеют вид:

При этом – усредненное значение , а – усредненное

* 1. **Аналитические выражения для вычисления локальных матриц**

Элементы матрицы жесткости:

J = r

Выразим как

Элементы матрицы массы.

Вычислять вручную каждый элемент не очень удобно, поэтому руководствуемся следующей логикой: подынтегральное выражение содержит в себе исключительно барицентрические функции в степенях ν1, ν2, ν3. Таким образом воспользовавшись формулой (\*) получим

Локальный вектор правой части найдем при помощи разложения f в виде линейного интерполянта. Получим

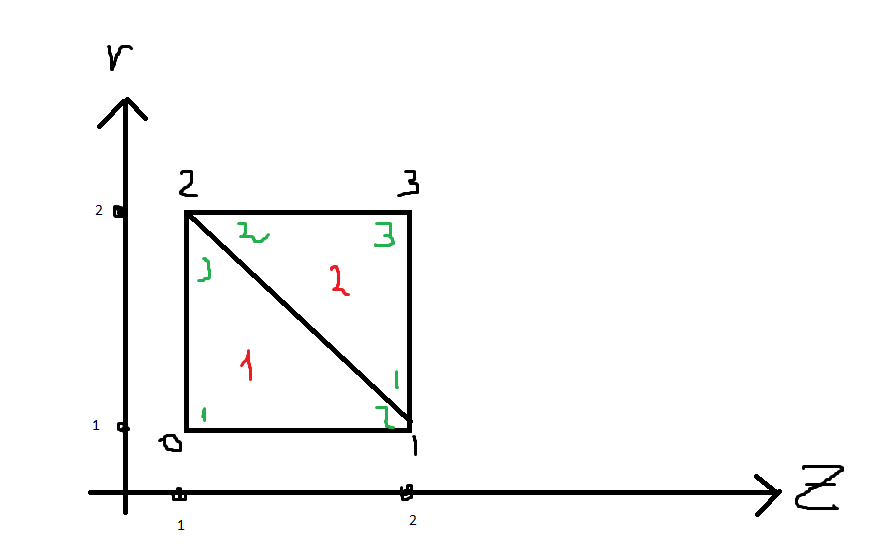
где равна матрица массы при а – значение функции f в точке конечного элемента

1. **Описание программы**

Структуры данных, используемые для задания расчётной области и конечноэлементной сетки выбраны из следующих соображений. Расчётная область – прямоугольник, зададим на нём прямоугольную сетку, каждый прямоугольник которой слева направо и сверху вниз поделим диагональю на два треугольника. Тогда нам, соответственно, для задания такой сетки необходимо знать количество узлов по r и z координатам, и сами массивы этих координат.

Программа состоит из 2 классов – один описывает параметры расчётной области: координаты сетки, количество узлов, а также в нем задаются параметры , и ug; второй класс описывает разреженную СЛАУ, формирует глобальную матрицу и вектор и решает ее методом сопряженных градиентов с диагональным предобуславливанием.

1. **Тестирование**

****

Первые краевые условия заданы на узлах 0, 1, 2, 3.

,

***1 тест***

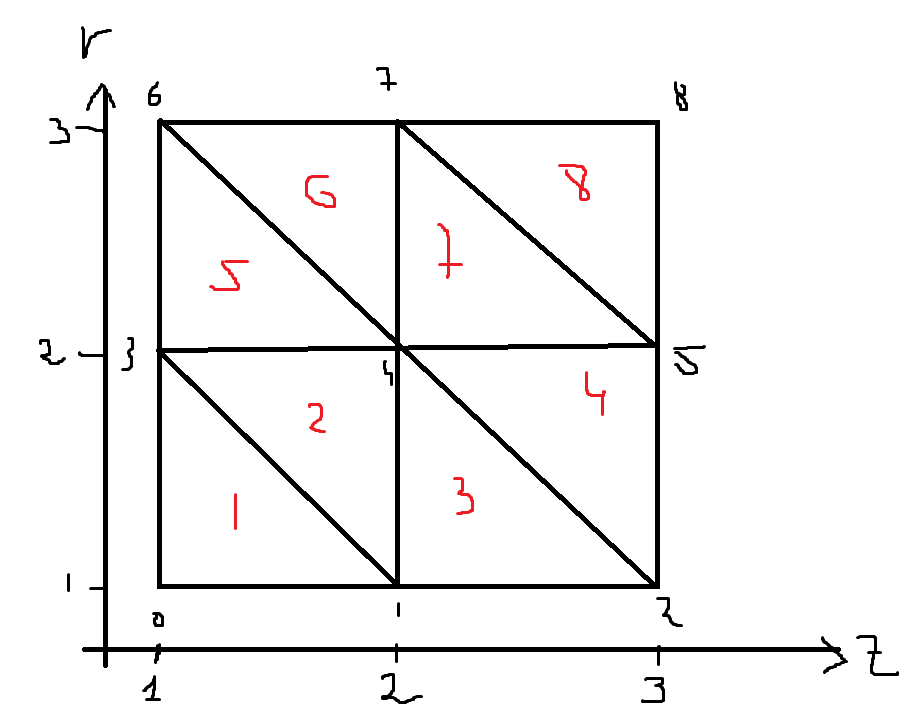
Проверка работы краевых условий.

Истинное значение функции: u\* = r+z,

f = r - 1/r +z.

Результаты:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | u | u\* | |u\* - u| |
| 1.000000e+00 | 1.000000e+00 | 2.000000e+00 | 2.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 2.000000e+00 | 3.000000e+00 | 3.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 1.000000e+00 | 3.000000e+00 | 3.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 2.000000e+00 | 4.000000e+00 | 4.000000e+00 | 0.000000e+00 |

******

Первые краевые условия заданы на узлах 0, 1, 2, 5, 8, 7, 6, 3.

,

***2 тест***

Проверка работы с константой.

Истинное значение функции: u\* = 1,

f = 1.

Результаты:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | u | u\* | |u\* - u| |
| 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 2.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 3.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 2.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 3.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 2.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 3.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |

***3 тест***

Проверим для линейной функции от r.

Истинное значение функции: u\* = r,

f = r – 1/r.

Результаты:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | u | u\* | |u\* - u| |
| 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 2.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 3.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 1.000000e+00 | 2.000000e+00 | 2.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 2.000000e+00 | 1.993827e+00 | 2.000000e+00 | 6.172840e-03 |
| 2.000000e+00 | 3.000000e+00 | 2.000000e+00 | 2.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 1.000000e+00 | 3.000000e+00 | 3.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 2.000000e+00 | 3.000000e+00 | 3.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 3.000000e+00 | 3.000000e+00 | 3.000000e+00 | 0.000000e+00 |

Погрешность объяснима неточностью вычисления интегралов правой части (в программе правая часть вычисляется как интерполянт, а не точное значение)

***4 тест***

Проверим для линейной функции от z.

Истинное значение функции: u\* = z,

f = z.

Результаты:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | u | u\* | |u\* - u| |
| 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 2.000000e+00 | 2.000000e+00 | 2.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 3.000000e+00 | 3.000000e+00 | 3.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 2.000000e+00 | 2.000000e+00 | 2.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 3.000000e+00 | 3.000000e+00 | 3.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 2.000000e+00 | 2.000000e+00 | 2.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 3.000000e+00 | 3.000000e+00 | 3.000000e+00 | 0.000000e+00 |

***5 тест***

Проверим для квадратичной функции.

Истинное значение функции: u\* = r2,

f = r2 – 4 .

Результаты:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | u | u\* | |u\* - u| |
| 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 2.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 3.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 1.000000e+00 | 4.000000e+00 | 4.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 2.000000e+00 | 4.000000e+00 | 4.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 3.000000e+00 | 4.000000e+00 | 4.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 1.000000e+00 | 9.000000e+00 | 9.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 2.000000e+00 | 9.000000e+00 | 9.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 3.000000e+00 | 9.000000e+00 | 9.000000e+00 | 0.000000e+00 |

***6 тест***

Проверим для квадратичной функции.

Истинное значение функции: u\* = z2,

f = z2 – 2 .

Результаты:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | u | u\* | |u\* - u| |
| 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 2.000000e+00 | 4.000000e+00 | 4.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 3.000000e+00 | 9.000000e+00 | 9.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 2.000000e+00 | 4.000000e+00 | 4.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 3.000000e+00 | 9.000000e+00 | 9.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 2.000000e+00 | 4.000000e+00 | 4.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 3.000000e+00 | 9.000000e+00 | 9.000000e+00 | 0.000000e+00 |

***7 тест***

***6 тест***

Вычислим кубическую функцию

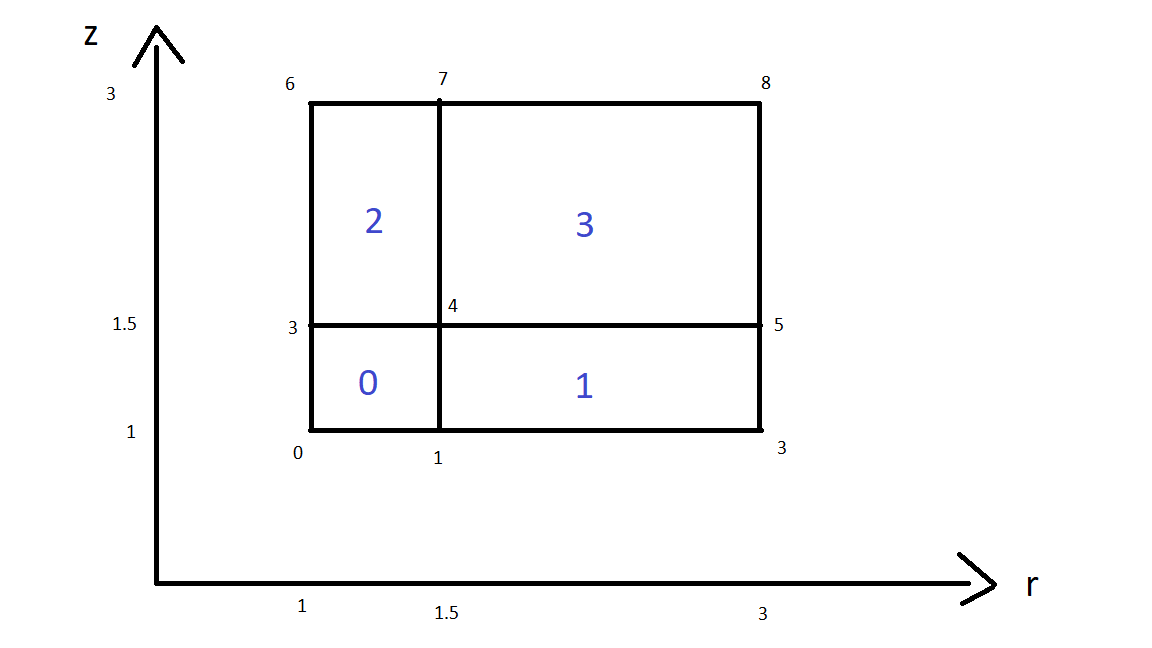
Истинное значение функции: u\* = rz,

f = zr – z/r.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | u | u\* | |u\* - u| |
| 1.5 | 1.5 | 2.24554 | 2.25 | 0.00446 |

***7 тест***

Неравномерная сетка.



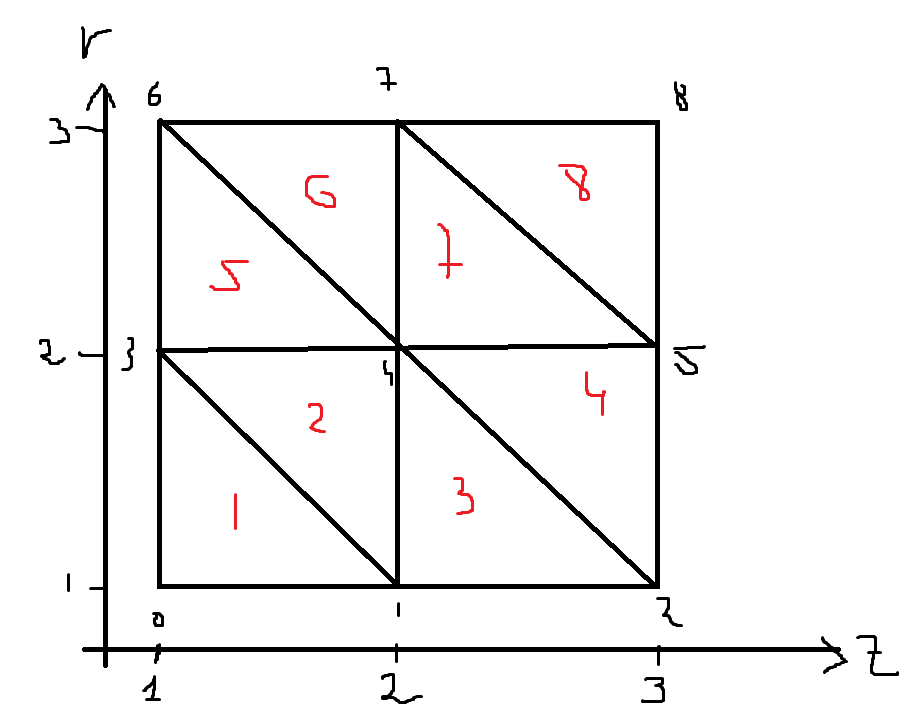
Первые граничные условия заданы на узлах 0, 1, 2, 3, 5, 6, 7, 8.

,

Проверим для билинейной функции.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | u | u\* | |u\* - u| |
| 1 | 1 | 0.841471 | 0.841471 | 0 |
| 2 | 1 | 0.909297 | 0.909297 | 0 |
| 3 | 1 | 0.14112 | 0.14112 | 0 |
| 1 | 2 | 0.909297 | 0.909297 | 0 |
| 2 | 2 | -0.532107 | -0.756802 | |0.224695| |
| 3 | 2 | -0.279415 | -0.279415 | 0 |
| 1 | 3 | 0.14112 | 0.14112 | 0 |
| 2 | 3 | -0.279415 | -0.279415 | 0 |
| 3 | 3 | 0.412118 | 0.412118 | 0 |

1. **Исследования**

******

Первые краевые условия заданы на узлах 0, 1, 2, 5, 8, 7, 6, 3.

,

Проверим для кубической функции порядок сходимости.

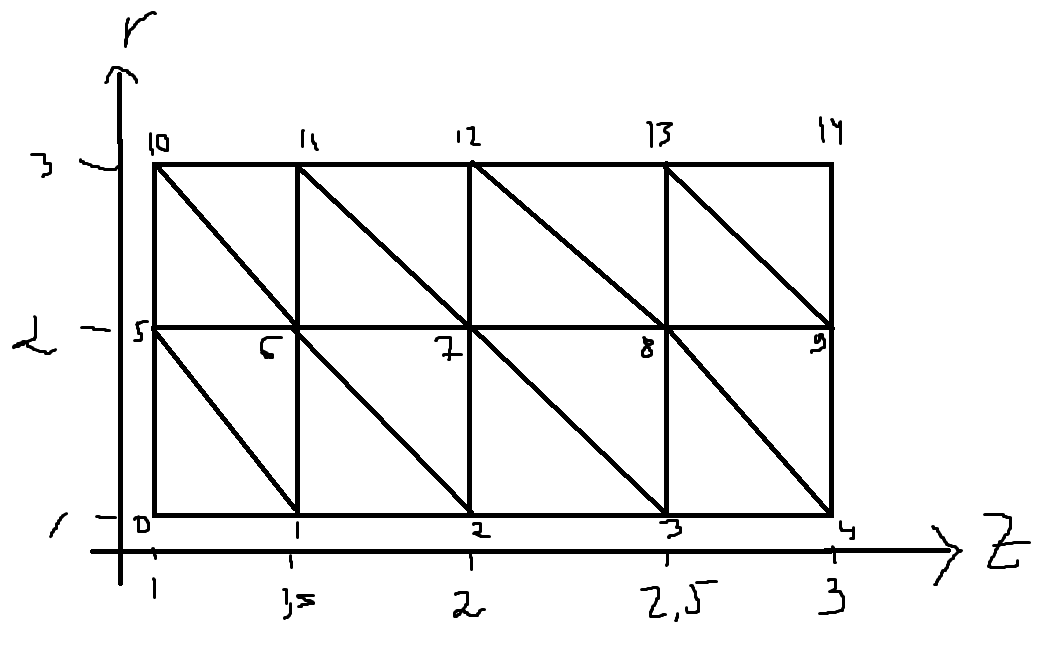
Истинное значение функции: u\* = z3,

f = z3 – 6z .

Результаты:

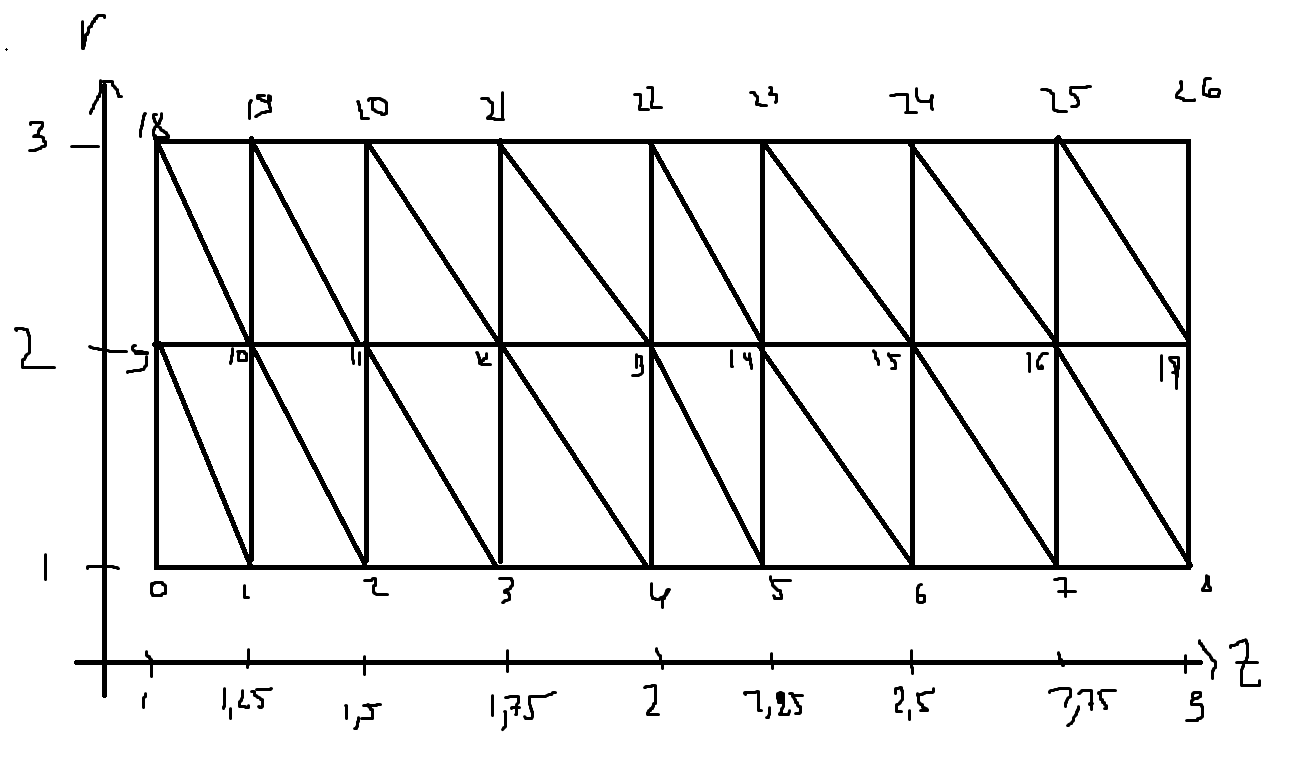
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | u | u\* | |u\* - u| |
| 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 2.000000e+00 | 8.000000e+00 | 8.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 3.000000e+00 | 2.700000e+01 | 2.700000e+01 | 0.000000e+00 |
| 2.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 2.000000e+00 | 8.055556e+00 | 8.000000e+00 | 5.555556e-02 |
| 2.000000e+00 | 3.000000e+00 | 2.700000e+01 | 2.700000e+01 | 0.000000e+00 |
| 3.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 2.000000e+00 | 8.000000e+00 | 8.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 3.000000e+00 | 2.700000e+01 | 2.700000e+01 | 0.000000e+00 |

Подробим сетку вдовое по OZ



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | u | u\* | |u\* - u| |
| 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 2.220446e-16 |
| 1.000000e+00 | 1.500000e+00 | 3.375000e+00 | 3.375000e+00 | 4.440892e-16 |
| 1.000000e+00 | 2.000000e+00 | 8.000000e+00 | 8.000000e+00 | 1.776357e-15 |
| 1.000000e+00 | 2.500000e+00 | 1.562500e+01 | 1.562500e+01 | 0.000000e+00 |
| 1.000000e+00 | 3.000000e+00 | 2.700000e+01 | 2.700000e+01 | 3.552714e-15 |
| 2.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 2.220446e-16 |
| 2.000000e+00 | 1.500000e+00 | 3.397648e+00 | 3.375000e+00 | 2.264793e-02 |
| 2.000000e+00 | 2.000000e+00 | 8.028801e+00 | 8.000000e+00 | 2.880084e-02 |
| 2.000000e+00 | 2.500000e+00 | 1.564765e+01 | 1.562500e+01 | 2.264793e-02 |
| 2.000000e+00 | 3.000000e+00 | 2.700000e+01 | 2.700000e+01 | 3.552714e-15 |
| 3.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 2.220446e-16 |
| 3.000000e+00 | 1.500000e+00 | 3.375000e+00 | 3.375000e+00 | 4.440892e-16 |
| 3.000000e+00 | 2.000000e+00 | 8.000000e+00 | 8.000000e+00 | 1.776357e-15 |
| 3.000000e+00 | 2.500000e+00 | 1.562500e+01 | 1.562500e+01 | 0.000000e+00 |
| 3.000000e+00 | 3.000000e+00 | 2.700000e+01 | 2.700000e+01 | 3.552714e-15 |

Подробим сетку вдовое по OZ

******

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| r | z | u | u\* | |u\* - u| |
| 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 1.250000e+00 | 1.953125e+00 | 1.953125e+00 | 0.000000e+00 |
| 1.000000e+00 | 1.500000e+00 | 3.375000e+00 | 3.375000e+00 | 0.000000e+00 |
| 1.000000e+00 | 1.750000e+00 | 5.359375e+00 | 5.359375e+00 | 0.000000e+00 |
| 1.000000e+00 | 2.000000e+00 | 8.000000e+00 | 8.000000e+00 | 0.000000e+00 |
| 1.000000e+00 | 2.250000e+00 | 1.139062e+01 | 1.139062e+01 | 0.000000e+00 |
| 1.000000e+00 | 2.500000e+00 | 1.562500e+01 | 1.562500e+01 | 0.000000e+00 |
| 1.000000e+00 | 2.750000e+00 | 2.079688e+01 | 2.079688e+01 | 7.105427e-15 |
| 1.000000e+00 | 3.000000e+00 | 2.700000e+01 | 2.700000e+01 | 0.000000e+00 |
| 2.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 2.000000e+00 | 1.250000e+00 | 1.960224e+00 | 1.953125e+00 | 7.099274e-03 |
| 2.000000e+00 | 1.500000e+00 | 3.386461e+00 | 3.375000e+00 | 1.146125e-02 |
| 2.000000e+00 | 1.750000e+00 | 5.373192e+00 | 5.359375e+00 | 1.381674e-02 |
| 2.000000e+00 | 2.000000e+00 | 8.014560e+00 | 8.000000e+00 | 1.456038e-02 |
| 2.000000e+00 | 2.250000e+00 | 1.140444e+01 | 1.139062e+01 | 1.381674e-02 |
| 2.000000e+00 | 2.500000e+00 | 1.563646e+01 | 1.562500e+01 | 1.146125e-02 |
| 2.000000e+00 | 2.750000e+00 | 2.080397e+01 | 2.079688e+01 | 7.099274e-03 |
| 2.000000e+00 | 3.000000e+00 | 2.700000e+01 | 2.700000e+01 | 0.000000e+00 |
| 3.000000e+00 | 1.000000e+00 | 1.000000e+00 | 1.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 1.250000e+00 | 1.953125e+00 | 1.953125e+00 | 0.000000e+00 |
| 3.000000e+00 | 1.500000e+00 | 3.375000e+00 | 3.375000e+00 | 0.000000e+00 |
| 3.000000e+00 | 1.750000e+00 | 5.359375e+00 | 5.359375e+00 | 0.000000e+00 |
| 3.000000e+00 | 2.000000e+00 | 8.000000e+00 | 8.000000e+00 | 0.000000e+00 |
| 3.000000e+00 | 2.250000e+00 | 1.139062e+01 | 1.139062e+01 | 0.000000e+00 |
| 3.000000e+00 | 2.500000e+00 | 1.562500e+01 | 1.562500e+01 | 0.000000e+00 |
| 3.000000e+00 | 2.750000e+00 | 2.079688e+01 | 2.079688e+01 | 7.105427e-15 |
| 3.000000e+00 | 3.000000e+00 | 2.700000e+01 | 2.700000e+01 | 0.000000e+00 |

Сравним погрешности:

Отношения погрешности e1/e2 = 1,96~2 e2/e3 = 1,97 ~ 2 Получили первый порядок сходимости, что совпадает с теорией ||u-uh||э ≤ Chm

m = 1 тк базисные функции линейные

Порядок аппроксимации и порядок сходимости в МКЭ эквивалентные понятия.

1. **Текст программы**

#include <iostream>

#include <fstream>

#include <math.h>

#include <vector>

using namespace std;

class mesh //сетка и все параметры задачи

{

public:

long count\_r, count\_z; //количество узлов по r,z

double \*r, \*z; //массивы координат

mesh(string file\_name); //конструктор mesh

~mesh();

double lambda(); //выдают значение лямбда

double gamma(double r, double z);

double tetta(double r, double z);

double f(double r, double z);

double ug(double r, double z);

double u(double r, double z);

};

mesh::mesh(string file\_name)

{

// считываем количество узлов и их координаты

fstream in;

long i;

in.open(file\_name);

in >> count\_r >> count\_z;

r = new double[count\_r];

z = new double[count\_z];

for (i = 0; i < count\_r; i++)

in >> r[i];

for (i = 0; i < count\_z; i++)

in >> z[i];

in.close();

}

mesh::~mesh()

{

delete[] r;

delete[] z;

}

double mesh::lambda()

{

return 1.0;

}

double mesh::gamma(double r, double z)

{

return 1.0;

}

double mesh::f(double r, double z)

{

return z\*z\*z-6\*z;

}

double mesh::ug(double r, double z)

{

return z\*z\*z;

}

double mesh::tetta(double r, double z)

{

if(z <= 1.1)

return -1;

else

return 1;

}

double mesh::u(double r, double z)

{

return z\*z\*z;

}

class sparse\_sistem //разряженная система

{

private:

long dim; //разреженно-строчная матрица

vector<long> ig, jg;

vector<double> di, gl;

vector<double> b; //вектор правой части

public:

sparse\_sistem(class mesh &MESH); //конструктор

~sparse\_sistem();

void make\_slau(class mesh &MESH); //сборка матрицы

void add(double value, long i, long j); //вклад в матрицу

void add(double value, long i); //вклад в вектор

void boundary1(long i, double value); //учёт первого краевого с симметризацией матрицы

void matrixvectormult(vector<double> \*a, vector<double> \*b); //A\*a=b

void inversediagonalvectormult(vector<double> \*a, vector<double> \*b); //D^(-1)\*a=b

double dotproduct(vector<double> \*a, vector<double> \*b); //(a,b)

void MSG(vector<double> \*result); //сопряжённые градиенты

long fact(long a); // a!

double integrate\_L(long nu[], double det); // вычисляет интеграл по базисным функциям nu - массив с степенями

double calc\_m(long i, long j, double r[], double z[], double det);

double calc\_b(long i, double r[], double z[], double det, mesh &MESH);

};

sparse\_sistem::sparse\_sistem(mesh &MESH)

{

// строим портрет матрицы

long i, j, k, num, \*tmp, \*\*tmp\_n;

// (?) tmp - массив с количеством внедиагональных элементов построчно

// tmp\_n - двмерный массив где первое измерение - номер строки, второе - номер столбцов с ненулевыми элементами

dim = MESH.count\_r \* MESH.count\_z;

tmp = new long[dim];

tmp\_n = new long \*[dim];

for (i = 0; i < dim; i++)

tmp\_n[i] = new long[3];

tmp[0] = 0;

for (i = 1; i < MESH.count\_r - 1; i++)

for (j = 1; j < MESH.count\_z - 1; j++)

{

num = i \* MESH.count\_z + j; // (?) num = номер узла

tmp[num] = 3;

tmp\_n[num][0] = num - MESH.count\_z;

tmp\_n[num][1] = tmp\_n[num][0] + 1;

tmp\_n[num][2] = num - 1;

}

for (i = 1; i < MESH.count\_z; i++)

{

tmp[i] = 1;

tmp\_n[i][0] = i - 1;

}

for (i = 1; i < MESH.count\_z - 1; i++)

{

num = MESH.count\_z \* (MESH.count\_r - 1) + i;

tmp[num] = 3;

tmp\_n[num][0] = num - MESH.count\_z;

tmp\_n[num][1] = tmp\_n[num][0] + 1;

tmp\_n[num][2] = num - 1;

}

for (i = 1; i < MESH.count\_r; i++)

{

num = MESH.count\_z \* i;

tmp[num] = 2;

tmp\_n[num][0] = num - MESH.count\_z;

tmp\_n[num][1] = tmp\_n[num][0] + 1;

}

for (i = 1; i < MESH.count\_r; i++)

{

num = MESH.count\_z \* (i + 1) - 1;

tmp[num] = 2;

tmp\_n[num][0] = num - MESH.count\_z;

tmp\_n[num][1] = num - 1;

}

for (i = 0, j = 0; i < dim; i++)

{

j += tmp[i];

}

di.resize(dim);

gl.resize(j);

b.resize(dim);

ig.resize(dim + 1);

jg.resize(j);

ig[0] = 0;

for (i = 0, j = 0; i < dim; i++)

{

di[i] = 0.0;

b[i] = 0.0;

ig[i + 1] = ig[i] + tmp[i];

for (k = 0; k < tmp[i]; k++, j++)

{

jg[j] = tmp\_n[i][k];

gl[j] = 0.0;

}

}

delete[] tmp;

for (i = 0; i < dim; i++)

delete[] tmp\_n[i];

delete[] tmp\_n;

ofstream out;

out.open("matr.txt");

out << "ig" << endl;

for (i = 0; i < dim + 1; i++)

out << i << " " << ig[i] << endl;

out << endl;

out << "jg" << endl;

for (i = 0; i < ig[dim]; i++)

out << i << " " << jg[i] << endl;

out << endl;

//out.clear();//????????

out.close();

}

sparse\_sistem::~sparse\_sistem()

{

ig.clear();

jg.clear();

di.clear();

gl.clear();

b.clear();

}

void sparse\_sistem::make\_slau(mesh &MESH)

{

long i, j;

double a11, a12, a21, a22, a31, a32;

double det;

double h, sumr;

double r[3] = {0, 0, 0}, z[3] = {0, 0, 0}, n[3] = {0, 0, 0};

for (i = 0; i < MESH.count\_r - 1; i++)//генерация вкладов с элементов

for (j = 0; j < MESH.count\_z - 1; j++)

{

//вклад с нижнего треугольника

n[0] = i \* MESH.count\_z + j;

n[1] = n[0] + 1;

n[2] = n[0] + MESH.count\_z;

r[0] = MESH.r[i];

r[1] = MESH.r[i];

r[2] = MESH.r[i + 1];

z[0] = MESH.z[j];

z[1] = MESH.z[j + 1];

z[2] = MESH.z[j];

det = (z[1]-z[0])\*(r[2]-r[0]) - (z[2]-z[0])\*(r[1]-r[0]); // |detD|

a11 = (r[1]-r[2])/det;

a12 = (z[2]-z[1])/det;

a21 = (r[2]-r[0])/det;

a22 = (z[0]-z[2])/det;

a31 = (r[0]-r[1])/det;

a32 = (z[1]-z[0])/det;

sumr = MESH.lambda() \* (r[0]+r[1]+r[2])\*fabs(det)/12;

add(sumr \* (a11 \* a11 + a12 \* a12) + MESH.gamma(r[0], z[0]) \* calc\_m(1, 1, r, z, det), n[0], n[0]);

add(sumr \* (a11 \* a21 + a22 \* a12) + MESH.gamma(r[0], z[0]) \* calc\_m(2, 1, r, z, det), n[1], n[0]);

add(sumr \* (a21 \* a21 + a22 \* a22) + MESH.gamma(r[0], z[0]) \* calc\_m(2, 2, r, z, det), n[1], n[1]);

add(sumr \* (a31 \* a11 + a32 \* a12) + MESH.gamma(r[0], z[0]) \* calc\_m(3, 1, r, z, det), n[2], n[0]);

add(sumr \* (a31 \* a21 + a32 \* a22) + MESH.gamma(r[0], z[0]) \* calc\_m(3, 2, r, z, det), n[2], n[1]);

add(sumr \* (a31 \* a31 + a32 \* a32) + MESH.gamma(r[0], z[0])\*calc\_m(3, 3, r, z, det), n[2], n[2]);

add(calc\_b(1, r, z, det, MESH), n[0]);

add(calc\_b(2, r, z, det, MESH), n[1]);

add(calc\_b(3, r, z, det, MESH), n[2]);

//Вклад с верхнего треугольника

n[0] = i \* MESH.count\_z + j + 1;

n[2] = n[0] + MESH.count\_z;

n[1] = n[2] - 1;

r[0] = MESH.r[i];

r[1] = MESH.r[i + 1];

r[2] = MESH.r[i + 1];

z[0] = MESH.z[j + 1];

z[1] = MESH.z[j];

z[2] = MESH.z[j + 1];

det = (z[1] - z[0]) \* (r[2] - r[0]) - (z[2] - z[0]) \* (r[1] - r[0]); // |detD|

a11 = (r[1] - r[2]) / det;

a12 = (z[2] - z[1]) / det;

a21 = (r[2] - r[0]) / det;

a22 = (z[0] - z[2]) / det;

a31 = (r[0] - r[1]) / det;

a32 = (z[1] - z[0]) / det;

sumr = MESH.lambda() \* (r[0] + r[1] + r[2]) \* fabs(det) / 12;

add(sumr \* (a11 \* a11 + a12 \* a12) + MESH.gamma(r[0], z[0]) \* calc\_m(1, 1, r, z, det), n[0], n[0]);

add(sumr \* (a11 \* a21 + a22 \* a12) + MESH.gamma(r[0], z[0]) \* calc\_m(2, 1, r, z, det), n[1], n[0]);

add(sumr \* (a21 \* a21 + a22 \* a22) + MESH.gamma(r[0], z[0]) \* calc\_m(2, 2, r, z, det), n[1], n[1]);

add(sumr \* (a31 \* a11 + a32 \* a12) + MESH.gamma(r[0], z[0]) \* calc\_m(3, 1, r, z, det), n[2], n[0]);

add(sumr \* (a31 \* a21 + a32 \* a22) + MESH.gamma(r[0], z[0]) \* calc\_m(3, 2, r, z, det), n[2], n[1]);

add(sumr \* (a31 \* a31 + a32 \* a32) + MESH.gamma(r[0], z[0]) \* calc\_m(3, 3, r, z, det), n[2], n[2]);

add(calc\_b(1, r, z, det, MESH), n[0]);

add(calc\_b(2, r, z, det, MESH), n[1]);

add(calc\_b(3, r, z, det, MESH), n[2]);

}

//Учёт первого краевого

// Верхняя граница

for (i = 0; i < MESH.count\_z; i++)

boundary1((MESH.count\_r - 1) \* (MESH.count\_z) + i, MESH.ug(MESH.r[MESH.count\_r - 1], MESH.z[i]));

// Нижняя граница

for (i = 0; i < MESH.count\_z; i++)

boundary1(i, MESH.ug(MESH.r[0], MESH.z[i]));

//Левая граница

for (i = 0; i < MESH.count\_r; i++)

boundary1(MESH.count\_z \* i, MESH.ug(MESH.r[i], MESH.z[0]));

// Правая граница

for (i = 1; i < MESH.count\_r+1; i++)

boundary1(MESH.count\_z \* i - 1, MESH.ug(MESH.r[i-1], MESH.z[MESH.count\_z-1]));

ofstream out;

out.open("slau.txt");

out << "IG\n";

for (i = 0; i < this->dim + 1; i++)

out << this->ig[i] << " ";

out << "\nJG\n";

for (i = 0; i < this->ig[this->dim]; i++)

out << this->jg[i] << " ";

out << "\nDI\n";

for (i = 0; i < this->dim; i++)

out << this->di[i] << " ";

out << "\nGL\n";

for (i = 0; i < this->ig[this->dim]; i++)

out << this->gl[i] << " ";

out << "\nB\n";

for (i = 0; i < this->dim; i++)

out << this->b[i] << " ";

out.close();

}

void sparse\_sistem::add(double value, long i, long j)

{

bool flag;

long M, m, k;

flag = true;

if (i == j)

{

this->di[i] += value;

flag = false;

}

else

{

M = (i > j ? i : j);

m = (i < j ? i : j);

for (k = this->ig[M]; flag && (k < this->ig[M + 1]); k++)

if (this->jg[k] == m)

{

this->gl[k] += value;

flag = false;

}

}

if (flag)

{

cout << "Error add" << i << "\t" << j << "\n";

exit(2);

}

}

void sparse\_sistem::add(double value, long i)

{

this->b[i] += value;

}

void sparse\_sistem::boundary1(long i, double value)

{

long j, k;

this->di[i] = 1.0;

this->b[i] = value;

for (j = this->ig[i]; j < this->ig[i + 1]; j++)

{

this->b[this->jg[j]] -= this->gl[j] \* value;

this->gl[j] = 0.0;

}

for (j = i + 1; j < this->dim; j++)

{

for (k = this->ig[j]; k < this->ig[j + 1]; k++)

if (this->jg[k] == i)

{

this->b[j] -= this->gl[k] \* value;

this->gl[k] = 0.0;

}

}

}

void sparse\_sistem::matrixvectormult(vector<double> \*a, vector<double> \*b)

{

long i, j;

for (i = 0; i < this->dim; i++)

(\*b)[i] = 0.0;

for (i = 0; i < this->dim; i++)

{

(\*b)[i] += (\*a)[i] \* this->di[i];

for (j = this->ig[i]; j < this->ig[i + 1]; j++)

{

(\*b)[i] += (\*a)[this->jg[j]] \* this->gl[j];

(\*b)[this->jg[j]] += (\*a)[i] \* this->gl[j];

}

}

}

void sparse\_sistem::inversediagonalvectormult(vector<double> \*a, vector<double> \*b)

{

long i;

for (i = 0; i < this->dim; i++)

(\*b)[i] = (\*a)[i] / this->di[i];

}

double sparse\_sistem::dotproduct(vector<double> \*a, vector<double> \*b)

{

long i;

double res;

res = 0.0;

for (i = 0; i < this->dim; i++)

res += (\*a)[i] \* (\*b)[i];

return res;

}

long sparse\_sistem::fact(long a)

{

long f = 1;

for(long i =1; i <= a; i++) f \*= i;

return f;

}

void sparse\_sistem::MSG(vector<double> \*result)

{

long i, j, max\_iter;

double eps, nn, nb, rr, rr1, alpfa, betta;

//double \*x, \*r, \*p, \*Ap, \*Dr;

vector<double> x, r, p, Ap, Dr;

eps = 1.e-9;

max\_iter = 200;

x.resize(dim);

r.resize(dim);

p.resize(dim);

Ap.resize(dim);

Dr.resize(dim);

for (i = 0; i < this->dim; i++)

x[i] = 0.0;

nb = sqrt(this->dotproduct(&b, &b));

this->matrixvectormult(&x, &r);

for (i = 0; i < this->dim; i++)

r[i] = this->b[i] - r[i];

this->inversediagonalvectormult(&r, &p);

this->inversediagonalvectormult(&r, &Dr);

j = 0;

rr = this->dotproduct(&Dr, &r);

cout << rr << endl;

nn = 1.0;

while (nn > eps && j < max\_iter)

{

this->matrixvectormult(&p, &Ap);

alpfa = rr / this->dotproduct(&Ap, &p);

for (i = 0; i < this->dim; i++)

{

x[i] = x[i] + alpfa \* p[i];

r[i] = r[i] - alpfa \* Ap[i];

}

this->inversediagonalvectormult(&r, &Dr);

rr1 = this->dotproduct(&Dr, &r);

betta = rr1 / rr;

for (i = 0; i < this->dim; i++)

p[i] = Dr[i] + betta \* p[i];

rr = rr1;

nn = sqrt(this->dotproduct(&r, &r)) / nb;

cout << j++ << "\t" << alpfa << "\t" << betta << "\t" << nn << endl;

}

for (i = 0; i < this->dim; i++)

(\*result)[i] = x[i];

x.clear();

r.clear();

p.clear();

Ap.clear();

Dr.clear();

}

double sparse\_sistem::integrate\_L(long nu[], double det)

{

double chisl = 1, znamen = 2;

for (long i = 0; i < 3; i++)

{

chisl \*= fact(nu[i]);

znamen += nu[i];

}

znamen = fact(znamen);

return (chisl\*fabs(det))/(znamen\*2);

}

double sparse\_sistem::calc\_m(long i, long j, double r[], double z[], double det)

{

long nu[3];

double sum = 0;

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

{

nu[0] = 0;

nu[1] = 0;

nu[2] = 0;

nu[i-1]++;

nu[j-1]++;

nu[m]++;

sum += r[m]\*integrate\_L(nu, det);

}

return sum;

}

double sparse\_sistem::calc\_b(long i, double r[], double z[], double det, mesh &MESH)

{

double sum = 0;

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

sum += MESH.f(r[k], z[k]) \* calc\_m(i, k+1, r, z, det);

return sum;

}

int main()

{

long i, j;

vector<double> res, anal, raz;

class mesh MESH("mesh.txt");

class sparse\_sistem SISTEM(MESH);

res.resize(MESH.count\_r \* MESH.count\_z);

anal.resize(MESH.count\_r \* MESH.count\_z);

raz.resize(MESH.count\_r \* MESH.count\_z);

SISTEM.make\_slau(MESH);

SISTEM.MSG(&res);

ofstream out;

out.open("res.xls");

out << "r\t" << "z\t" << "u\*(r,z)\t" << "u(r,z)\t" << "(u\*-u)(r,z)\n";

for (i = 0; i < MESH.count\_r; i++)

for (j = 0; j < MESH.count\_z; j++)

{

anal[i \* MESH.count\_z + j] = MESH.u(MESH.r[i], MESH.z[j]);

raz[i \* MESH.count\_z + j] = fabs(res[i \* MESH.count\_z + j] - anal[i \* MESH.count\_z + j]);

out<<scientific << MESH.r[i] << "\t" << MESH.z[j] << "\t" << res[i \* MESH.count\_z + j] << "\t" << anal[i \* MESH.count\_z + j] << "\t" << raz[i \* MESH.count\_z + j] << endl;

}

out << endl << sqrt(SISTEM.dotproduct(&raz, &raz));

out.close();

res.clear();

anal.clear();

raz.clear();

return 0;

}