|  |  |  |
| --- | --- | --- |
| Министерство науки и высшего образования  Российской Федерации | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования | | |
| «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
|  | | |
| Курсовой проект | | |
| по дисциплине «Численные методы» | | |
|  | | |
| **МКЭ для краевой задачи** | | |
|  | | |
|  | Факультет: | ПМИ |
| Группа: | ПМ-81 |
| Вариант: | 43 |
| Студенты: | Ефремов Артур |
|  |  |
|  |  |
| Преподаватели: | Патрушев Илья Игоревич  Задорожный Александр Геннадьевич  Рояк Михаил Эммануилович |
|  | | |
| Новосибирск | | |
| 2021 | | |

1. **Постановка задачи**
   1. **Условие задачи**

Вариант №43. МКЭ для двумерной краевой задачи для гиперболического уравнения в декартовой системе координат. Базисные функции биквадратичные на прямоугольниках. Краевые условия всех типов. Коэффициент диффузии разложить по билинейным базисным функциям. Матрицу СЛАУ сгенерировать в разреженном строчном формате. Для решения СЛАУ использовать МСГ или ЛОС с неполной факторизацией. Четырехслойная неявная схема по времени.

* 1. **Решаемое уравнение**
     1. **В общем виде**
     2. **В декартовой системе координат**
  2. **Краевые условия**
  3. **Расчетная область**

Расчетная область задается с помощью описания подобластей прямоугольной формы.

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

Потребуем, чтобы невязка дифференциального уравнения была ортогональна некоторому пространству пробных функций .

Используя формулу Грина, получим:

Так как , преобразуем интегралы по границам и , воспользовавшись краевыми условиями 2 и 3:

Исключаем так как краевыми условиями не определяется значение . Тогда потребуем, чтобы содержало только - функции, которые принимают нулевые значения на границе . Тогда в качестве выберем - пространство функций, имеющие суммируемые с квадратом производные и равных нулю на границе .

Таким образом, получим вариационное уравнение вида

* 1. **Конечноэлементная дискретизация и переход к локальным матрицам**
     1. **Конечноэлементная дискретизация**

Разобьем Ω на подобласти, получим:

Функцию будем искать в виде разложения по базисным функциям с соответствующими весами :

Итак, используя базисные функции, принимающих нулевые значения во всех узлах сетки кроме одного, СЛАУ для вектора весов может быть записана в матричном виде:

где компоненты матрицы A и вектора b определяются соотношениями:

Матрица жесткости:

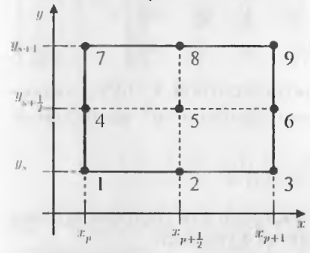
Матрица массы:

* + 1. **Построение базисных функций**

Биквадратичной называется функция вида:

Каждая биквадратичная функция фактически определяется значениями девяти коэффициентов. Поэтому на конечном элементе мы можем представить любую функцию в виде линейной комбинации некоторых девяти линейно-независимых биквадратичных функций.

Рассмотрим следующий способ определения явного вида биквадратичных базисных функций на прямоугольном конечном элементе. Представим каждую базисную функцию в виде произведений одномерных квадратичных базисных функций координат x и y:



Введем следующие обозначения:

Тогда локальные базисные функции на будут иметь следующий вид:

* + 1. **Локальные матрицы**

На локальных элементах базисные функции и можно выбрать следующими:

где или .

Так как коэффициент диффузии необходимо разложить по билинейным базисным функциям, выберем их в виде

На конечном элементе локальная матрица жесткости имеет следующий вид:

C учетом замены и разложением коэффициента диффузии по билинейному базису, локальная матрица жесткости принимает следующий вид:

Матрицу же массы удобнее всего вычислить через компоненты локальных матриц одномерных квадратичных элементов:

Локальный вектор правой части с учетом того что функция на конечном элементе представлена в виде биквадратичного интерполянта

может быть вычислен с помощью соотношения

где - вектор, составленный из значений правой части дифференциального уравнения в узлах элемента, – матрица массы:

Локальная матрица ребра длины с заданным на нем краевым условием третьего рода имеет вид:

а локальный вектор этого ребра при представлении на в виде разложения по одномерным квадратичным локальным базисным функциям имеет вид:

Локальный вектор ребра длины с заданным на нем краевым условием второго рода при представлении параметра на вычисляется аналогично и имеет вид:

Краевые условия первого рода учитываются после полной сборки глобальной матрицы и правой части путем фиксации соответствующих весов , при решении СЛАУ. Таким образом, из сгенерированной СЛАУ можно исключить уравнения с теми номерами, которые являются уравнениями узлов, лежащих на границе с краевыми условиями первого рода, а весам с этими номерами присвоить значения первого краевого условия в соответствующих узлах сетки. В работе реализован способ с занулением соответствующей строки глобальной матрицы системы, установкой единицы на главной диагонали и значения точного решения в соответствующей компоненте вектора правой части.

* 1. **Построение схемы по времени**

В четырехслойной неявной схеме по времени искомая функция может быть представлена в следующем виде:

где функции являются кубическими полиномами Лагранжа и имеют следующий вид:

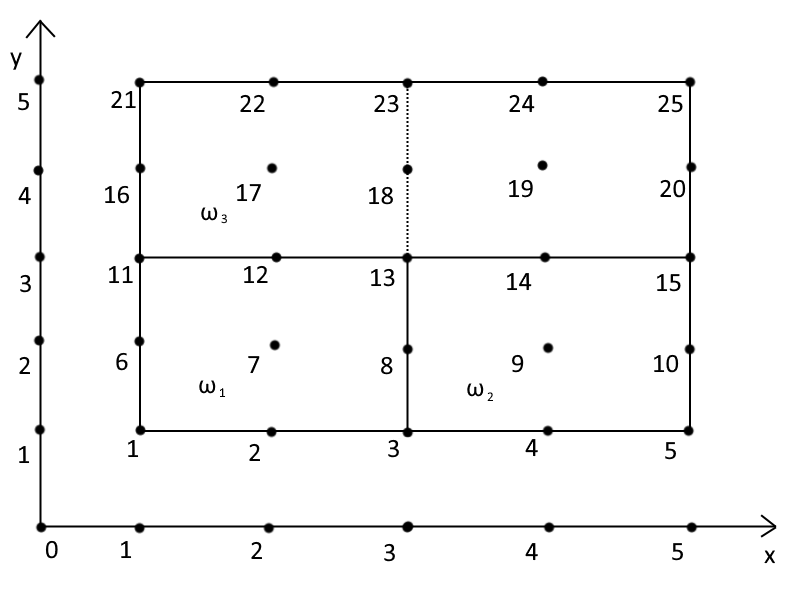
Вычислим производные функций по t:

Вычислим вторые производные функций по t:

Тогда задача сводится к решению следующей системы:

1. **Описание разработанных программ** 
   1. **Структуры данных, используемые для задания расчетной области и конечноэлементной сетки**

Разберем задание расчетной области и конечноэлементной сетки на следующем примере:



Расчетная область и конечноэлементная сетка задаются в файле **“regions.txt”** в следующем формате:

3

1 3 5

2 2

3

1 3 5

2 2

3

0 1 0 1

1 2 0 1

0 2 1 2

Число **n** в первой строке соответствует количеству координат, необходимых для описания границ подобластей (регионов) по оси X, во второй строке идут **n** координат границ регионов по оси X (координатные линии). В третьей строке идут **n – 1** чисел, задающих количество разбиений сетки между координатными линиями. Разбиение строится и заносится в вектор **x\_nodes.**

Затем в строках 4, 5, 6 по схожему алгоритму описывается информация для построения сетки по Y, заполнения массива **y\_nodes.**

В строке 7 стоит число **m**, задающее количество регионов. В следующих **m** строках идут 4 индекса координат для левой, правой, нижней и верхней границы **i-ой** области, где , (**m** в примере равно 3).

Для хранения индексов границ регионов в программе используется двумерный массив **regions,** размерность **m x 4,** где **m** – количество регионов, на местах **m[0], m[1]** строят индексы границ региона в массиве **x\_nodes**, на местах **m[2], m[3]** строят индексы границ региона в массиве **y\_nodes**. Хранение индексов таким образом позволяет добиться целочисленного сравнения при расчете подобласти, в которую попадает конечных элемент, зная индексы его центрального узла в массивах **x\_nodes** и **y\_nodes,** а также позволяет не выделять память под хранение координат каждого узла сетки, что позволяет сэкономить большое количество памяти, особенно при значительном увеличении числа дробления сетки.

Сетка по времени считывается из файла **“time.txt”** в котором хранится два числа: конечная временная точка и количество разбиений сетки. Сетка хранится в массиве **time\_grid.**

* 1. **Структура основных модулей программы, в том числе генерация портрета СЛАУ, вычисление локальных матриц, генерация глобальных матриц, решение СЛАУ.**
     1. **Вычисление локальных матриц**
        1. **Матрица массы**

Формулу:

из

можно привести к виду:

где

и еще упростив, получим:

где

Матрица не зависит от размера конечного элемента, поэтому может быть вычислена заранее. Для удобства вычисления матрицы был написан скрипт на языке python:

import numpy as np

import math

def mu(i):

    return i % 3

def nu(i):

    return math.floor(i / 3)

M = np.array(

    [[4,2,-1],

    [2,16,2],

    [-1,2,4]])

M1 = np.zeros((9,9))

for i in range(9):

    for j in range(9):

        M1[i][j] = M[mu(i)][mu(j)] \* M[nu(i)][nu(j)]

def out(file\_name, mat):

    with open(file\_name, "w") as f:

     for i in range(9):

         f.write(str(mat[i][i]))

         f.write(" ")

     f.write("\n")

     for i in range(9):

for j in range(i):

             f.write(str(mat[i][j]))

             f.write(" ")

out("M.txt", M1)

Скрипт также осуществляет вывод диагоналей и нижних треугольников матриц в файл, для последующего чтения этих файлов основной программой.

* + - 1. **Матрица жесткости**

Формулу

можно привести к виду

где

Каждую из матриц и можно разделить на сумму 4 матриц и , которые будут отличаться лишь базисными функциями, отвечающими за разложение коэффициента диффузии в формулах для расчета интегралов. Представленные таким образом матрицы и не зависят от размера конечного элемента, поэтому могут быть вычислены заранее. Для удобства расчетов интегралов был написан скрипт на языке python:

from scipy.integrate import dblquad

import numpy as np

import math

def psi1(x):

    return 1 - x

def psi2(x):

    return x

flin = [psi1, psi2]

def phi1(x):

    return 2 \* (x - 0.5) \* (x - 1)

def phi2(x):

    return -4 \* x \* (x - 1)

def phi3(x):

    return 2 \* x \* (x - 0.5)

def dphi1dx(x):

    return 4 \* x - 3

def dphi2dx(x):

    return -8 \* x + 4

def dphi3dx(x):

    return 4 \* x – 1

fquad = [phi1, phi2, phi3]

dfquaddx = [dphi1dx, dphi2dx, dphi3dx]

Gl = np.zeros((4, 9, 9))

Gr = np.zeros((4, 9, 9))

for s in range(4):

    for i in range(9):

        for j in range(9):

            f = lambda x, y: flin[s % 2](x) \* flin[s // 2](y) \* fquad[i // 3](x) \* dfquaddx[i % 3](y) \* fquad[j // 3](x) \* dfquaddx[j % 3](y)

            Gl[s][i][j] = dblquad(f, 0, 1, 0, 1)[0]

            f = lambda x, y: flin[s % 2](x) \* flin[s // 2](y) \* dfquaddx[i // 3](x) \* fquad[i % 3](y) \* dfquaddx[j // 3](x) \* fquad[j % 3](y)

            Gr[s][i][j] = dblquad(f, 0, 1, 0, 1)[0]

def out(file\_name, mat):

    with open(file\_name, "w") as f:

        for i in range(9):

            f.write(str(mat[i][i]))

            f.write(" ")

        f.write("\n")

        for i in range(9):

            for j in range(i):

                f.write(str(mat[i][j]))

                f.write(" ")

out("data/Gl0.txt", Gl[0] \* 90)

out("data/Gl1.txt", Gl[1] \* 90)

out("data/Gl2.txt", Gl[2] \* 90)

out("data/Gl3.txt", Gl[3] \* 90)

out("data/Gr0.txt", Gr[0] \* 90)

out("data/Gr1.txt", Gr[1] \* 90)

out("data/Gr2.txt", Gr[2] \* 90)

out("data/Gr3.txt", Gr[3] \* 90)

Скрипт также осуществляет вывод диагоналей и нижних треугольников матриц в файл, для последующего чтения этих файлов основной программой.

* + 1. **Генерация глобальной матрицы**

Генерация глобальной матрицы происходит в цикле по конечным элементам. Для узлов каждого конечного элемента вычисляется соответствующий им индекс в глобальной нумерации узлов по следующей формуле:

где – количество конечных элементов по оси X, – номер конечного элемента, считая слева направо, снизу вверх.

Тогда для 9 узлов можно определить формулы их индексов в глобальной нумерации:

Координаты же узлов конечного элемента считаются по следующим формулам:

где и – массивы с координатами сетки по X и по Y соответственно.

Номер области, в которую попал конечный элемент легко найти – благодаря структуре хранения информации по каждом регионе (см. 3.1).

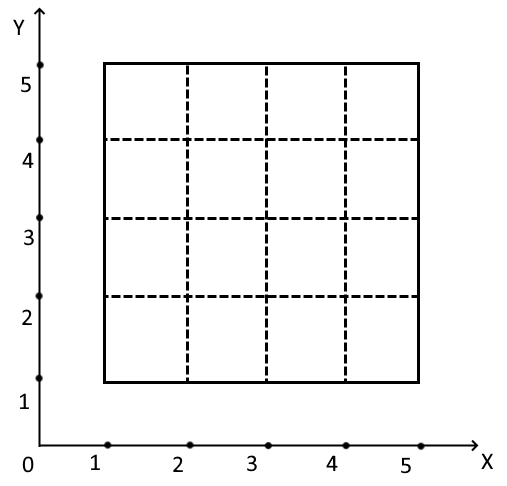
* + 1. **Генерация портрета глобальной матрицы**

В цикле по конечным элементам вычисляются индексы узлов конечных элементов, формируется вспомогательный массив **help** размера 9 на 9, где для каждого элемента **helpij** лежит пара чисел – i и j в глобальной нумерации. Затем для каждого элемента из массива **help**, если в профиле i – той строки в портрете глобальной матрицы нет элемента, для которого jg равен j, добавим его, увеличим значения всех элементов начиная с i в массиве ig на 1.

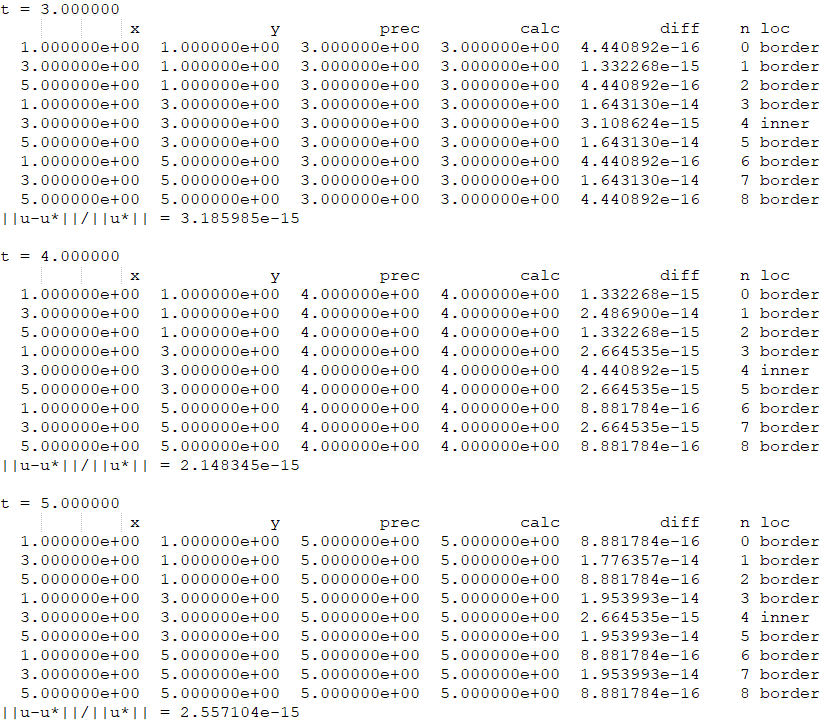
* + 1. **Решение СЛАУ**

Решение СЛАУ осуществляется с помощью МСГ с неполной диагональной факторизацией.

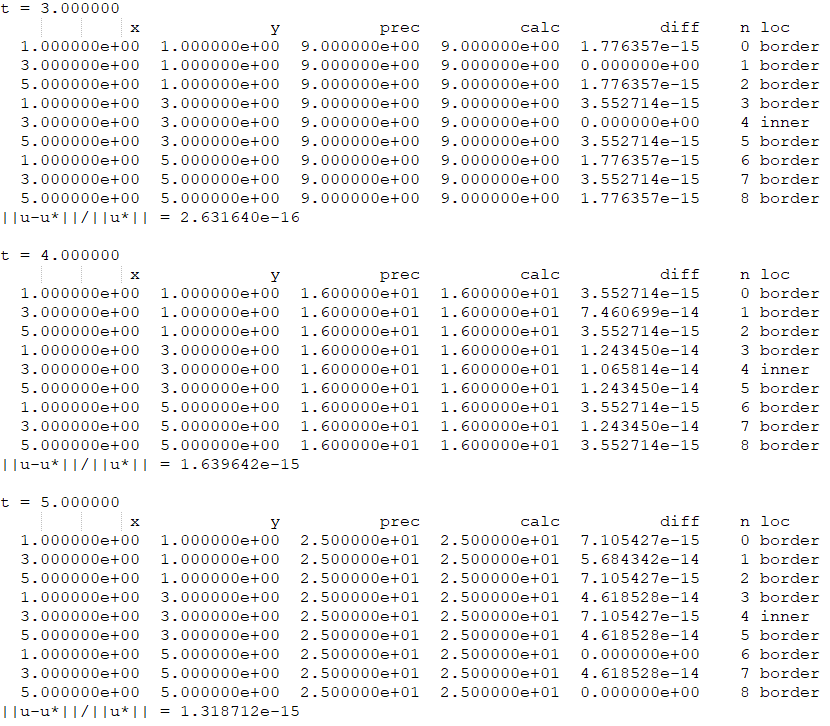
1. **Описание тестирования программы**
2. **Исследование порядка аппроксимации**
   1. **Исследование порядка аппроксимации по времени**

****

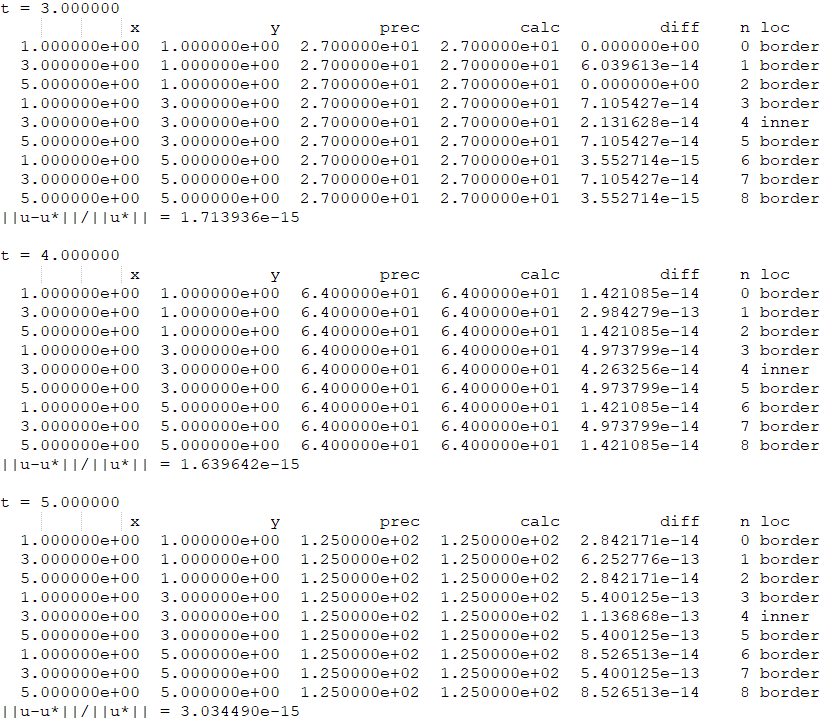
* + 1. **Тест 1**



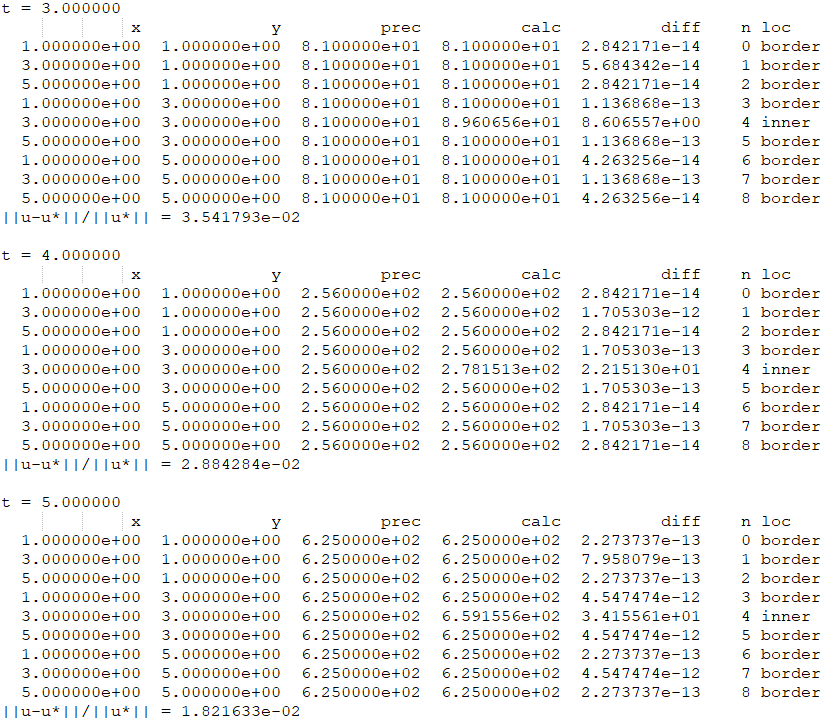
* + 1. **Тест 2**



* + 1. **Тест 3**



* + 1. **Тест 4**

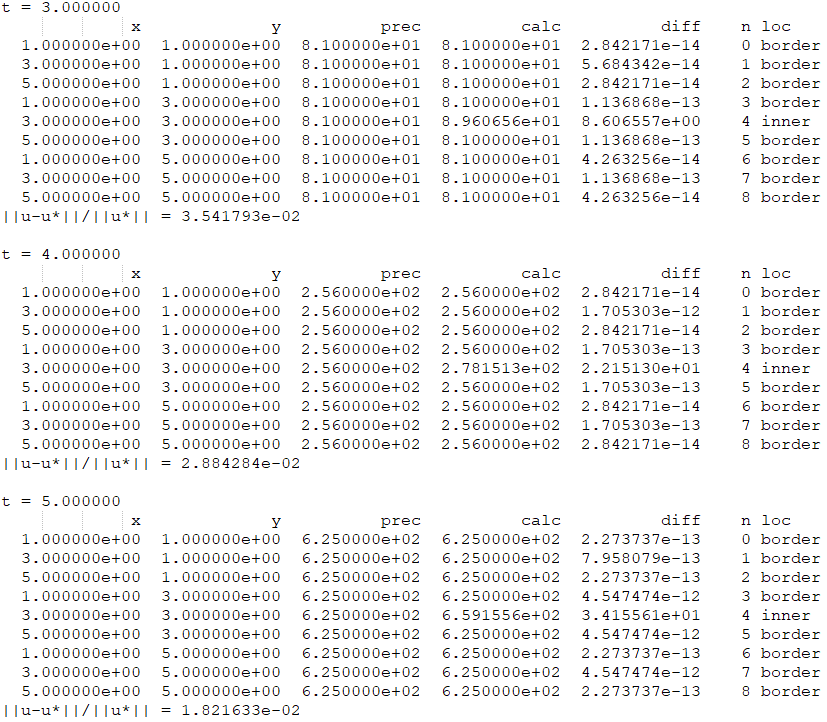


При увеличении степени t в искомой функции, начиная с , происходит увеличение погрешности.

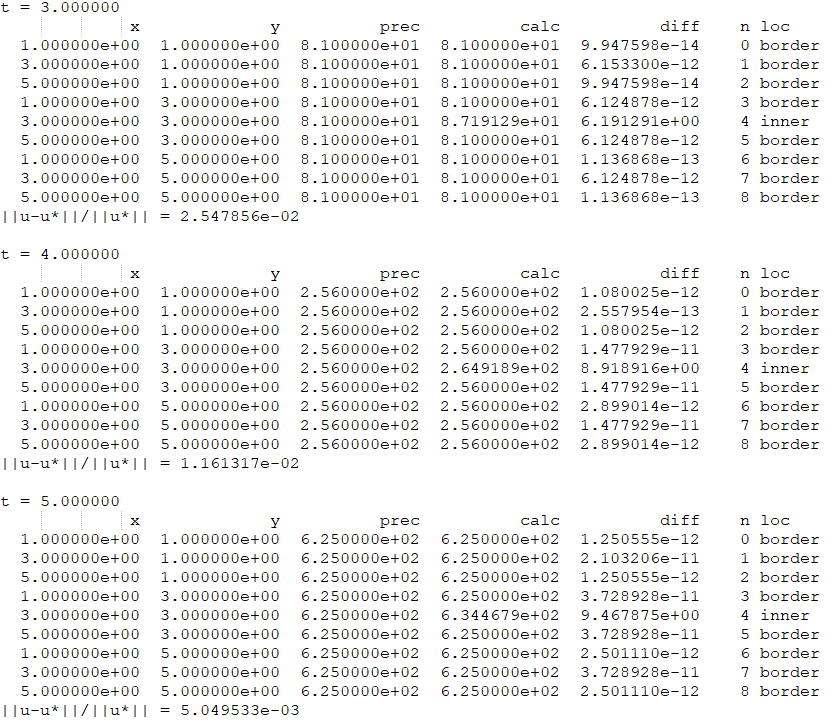
1. **Исследование порядка сходимости**
   1. **Исследование порядка сходимости по времени**

Для удобства будем отображать только временные слои, которые были на изначальной временной сетке.

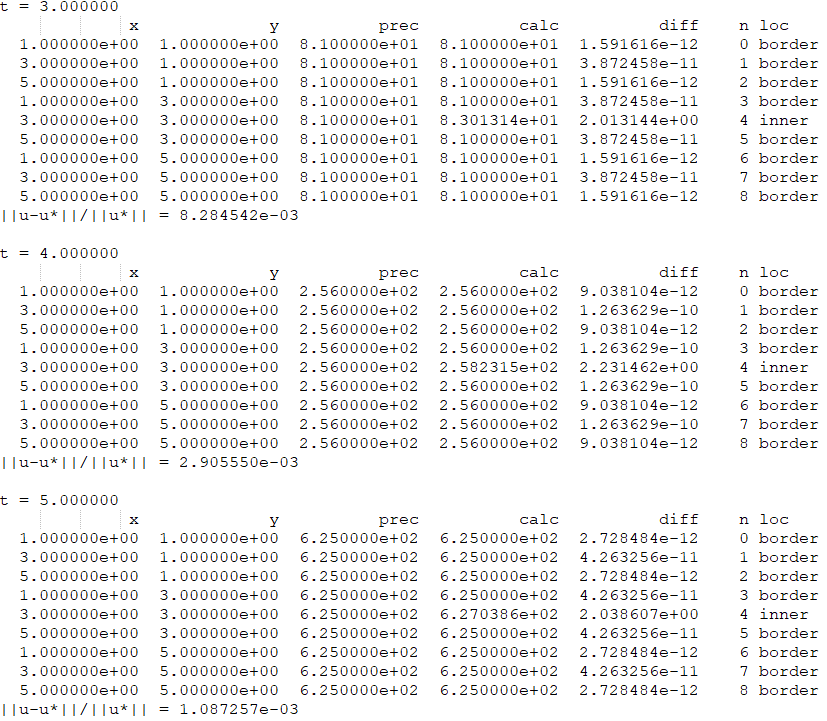
* + 1. **Тест 1**



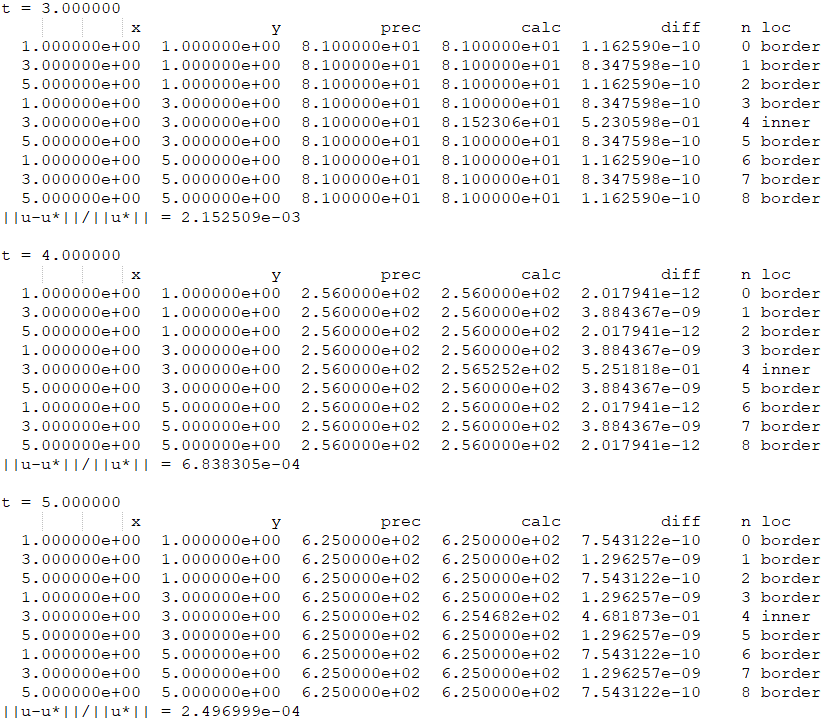
* + 1. **Тест 2**



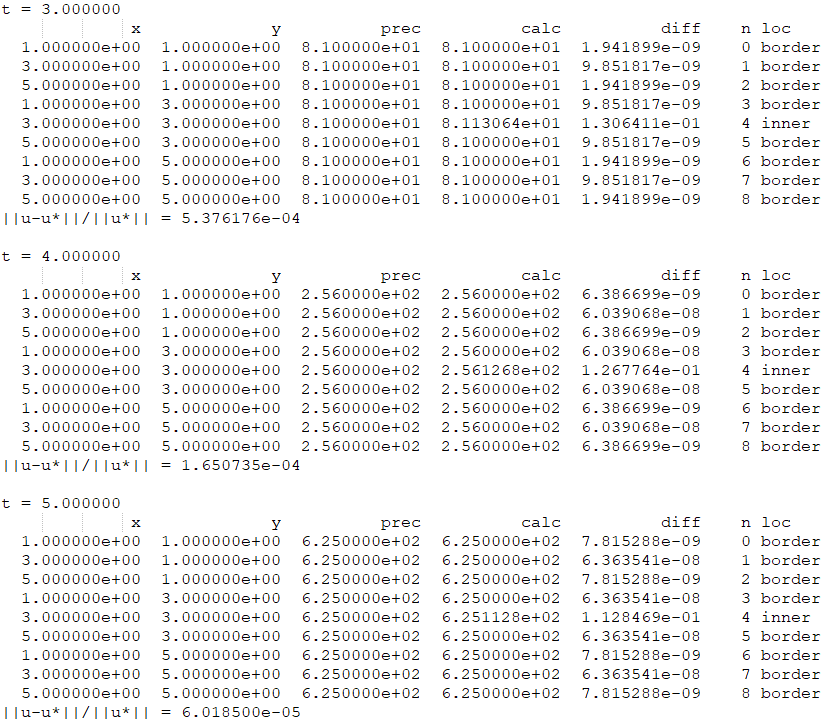
* + 1. **Тест 3**



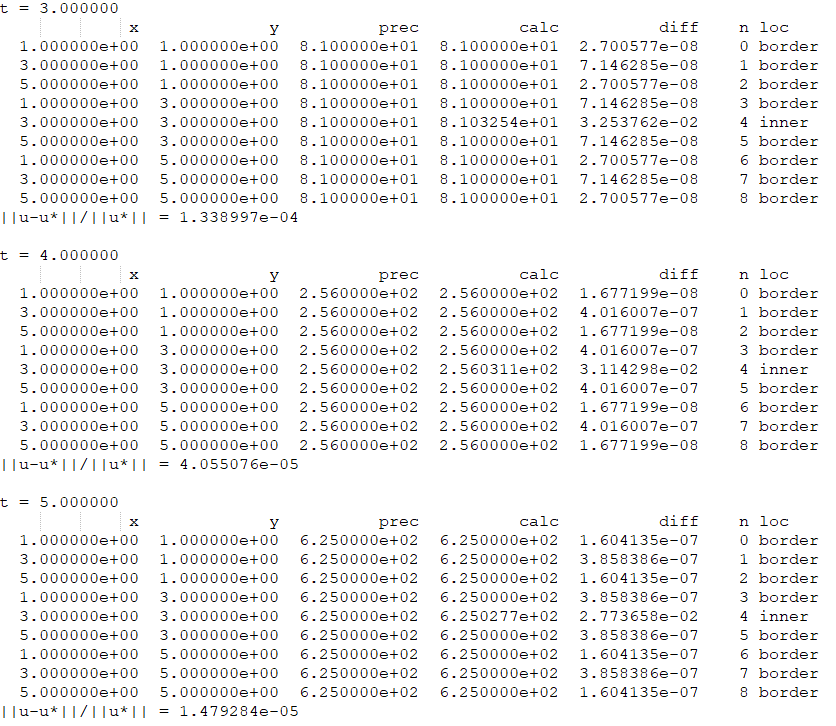
* + 1. **Тест 4**



* + 1. **Тест 5**



* + 1. **Тест 6**



По результатам тестов можно составить таблицу для сравнения погрешностей во внутренней точке области:



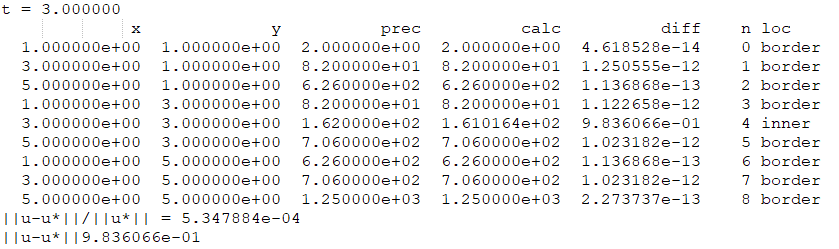
и для сравнения нормы погрешности по всем точкам:



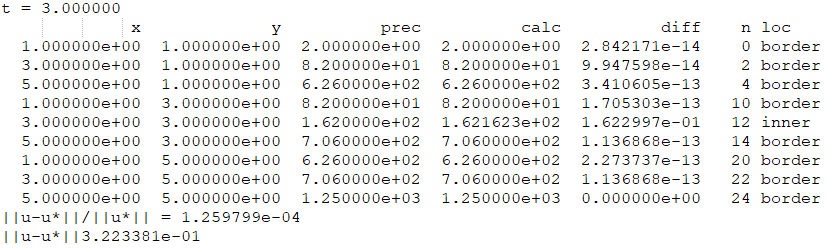
* 1. **Исследование порядка сходимости по пространству**

Для удобства будем отображать только узлы, которые были на изначальной пространственной сетке.

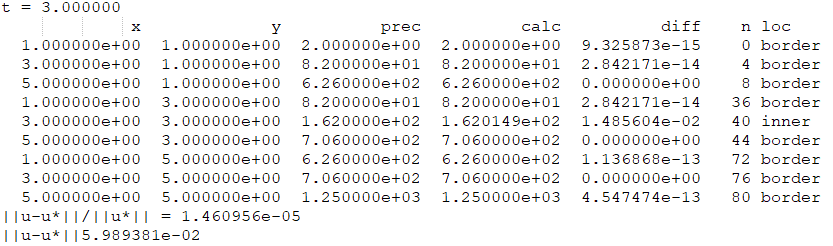
* + 1. **Тест 1**



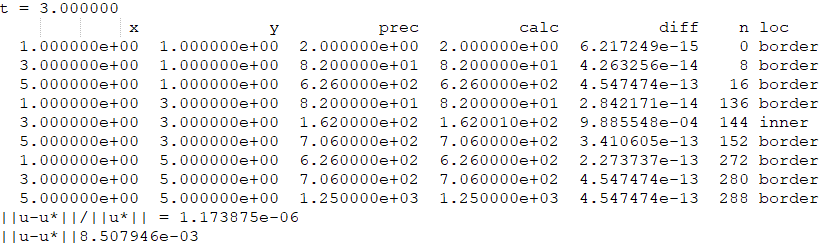
* + 1. **Тест 2**



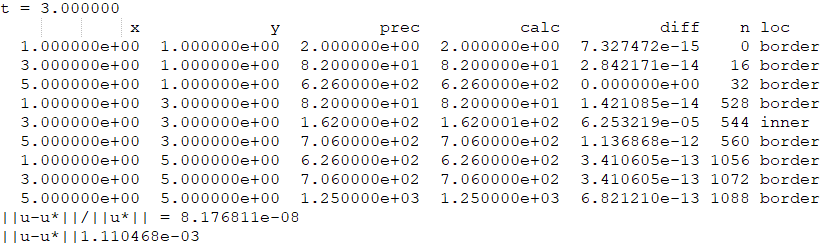
* + 1. **Тест 3**



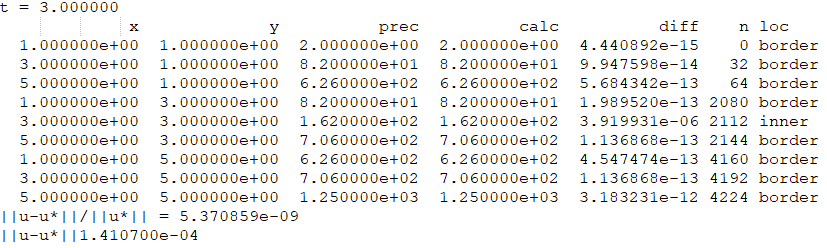
* + 1. **Тест 4**



* + 1. **Тест 5**



* + 1. **Тест 6**



По результатам тестов можно составить таблицу для сравнения погрешностей во внутренней точке области:



и для сравнения нормы погрешности по всем точкам:

