Белорусский государственный университет

Факультет прикладной математики и информатики

Разностные схемы для уравнений Пуассона и теплопроводности

Отчет по лабораторной работе

студента 3 курса 3 группы

Аквуха Джеймса

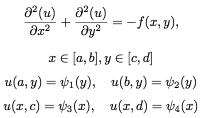
Преподаватель:

Будник А.М.

**1. Уравнение Пуассона**

**1.1 Постановка задачи**

Поставлена задача Дирихле для уравнения Пуассона в прямоугольной области:



Требуется найти значения функции на равномерной сетке узлов в прямоугольнике .

**1.2 Описание метода**

Исходное дифференциальное уравнение записывается на сетке на пятиточечном шаблоне:



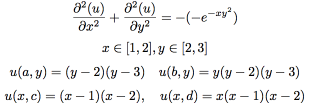
Данное уравнение можно решить с помощью метода Зейделя:



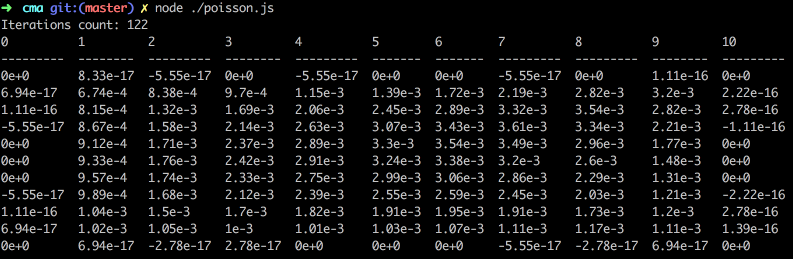
Двигаясь слева направо, сверху вниз получаем разностный метод решения задачи Дирихле с погрешностью . Итерации проводятся пока норма невязки изменения решения не окажется меньше .

**1.3. Результаты**

Условия задачи Дирихле:



Точное решения находилось с помощью Wolfram Mathematica.



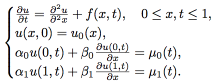
**1.4. Вывод**

Как видно из результата, построенный метод обладает погрешностью не более . Недостатком данного метода является то, что для обеспечения точности в 3 знака после запятой потребовалось совершить 122 итерации.

**2. Уравнение теплопроводности**

**2.1 Постановка задачи**

Поставлена граничная задача для дифференциального уравнения в частных производных.



Требуется найти приближенное значение функции на равномерной сетке узлов в прямоугольнике с погрешностью не хуже .

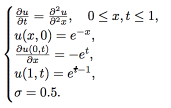
**2.2 Описание метода**

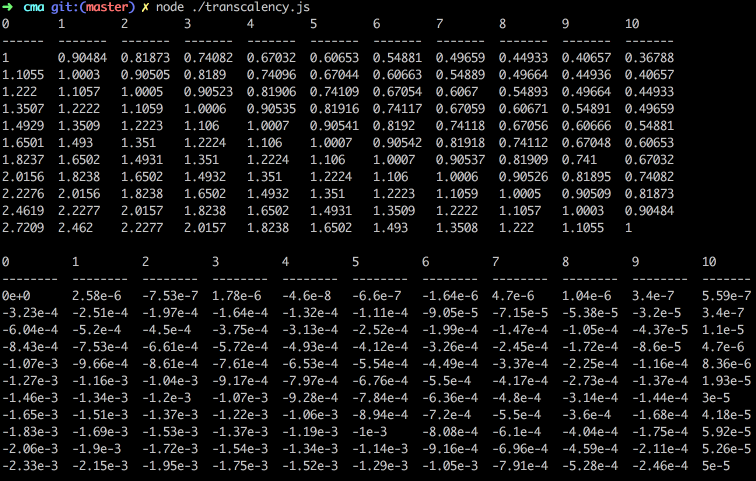
Записав условия на шеститочечном шаблоне, получим:

Первый ряд известен из условий, остальные вычисляются последовательно.

**2.3. Результаты**

Условия:





**2.4. Вывод**

Погрешность полученного метода не превышает допустимую.

**3. Листинги**

**// poisson.js**

**require('console.table')  
const {cloneDeep} = require('lodash')  
const {max, pow, abs} = Math  
const {eval, subtract, transpose} = require('mathjs')  
  
const array = n => [...Array(n).keys()].map(\_ => 0)  
const range = (a, b, n = 1) => [...Array(n + 1).keys()].map(i => a + (b - a) / n \* i)  
  
const f = (x, y) => eval('e^(-x y^2)', {x, y})  
const top = (x, y) => x \* (x - 1) \* (x - 2)  
const right = (x, y) => y \* (y - 2) \* (y - 3)  
const bottom = (x, y) => (x - 1) \* (x - 2)  
const left = (x, y) => (y - 2) \* (y - 3)  
  
const EPS = 1E-5  
const N = 10  
const vx = range(1, 2, N)  
const vy = range(3, 2, N)  
const hx = abs(vx[1] - vx[0])  
const hy = abs(vy[1] - vy[0])  
const matrix = array(N + 1).map(\_ => array(N + 1))  
  
for (let i = 0; i < N + 1; ++i) {  
 matrix[0][i] = top(vx[i], vy[0])  
 matrix[N][i] = bottom(vx[i], vy[N])  
 matrix[i][0] = left(vx[0], vy[i])  
 matrix[i][N] = right(vx[N], vy[i])  
}  
  
let eps = 1  
let iters\_count = 0  
let c = 1 / (2 / pow(hx, 2) + 2 / pow(hy, 2))  
while (eps > EPS && ++iters\_count) {  
 eps = 0  
 let prev\_matrix = cloneDeep(matrix)  
 for (let i = 1; i < N; ++i) {  
 for (let j = N - 1; j > 0; --j) {  
 matrix[i][j] = c \* (f(vx[i], vy[j]) +   
 (prev\_matrix[i + 1][j] + matrix[i - 1][j]) / pow(hx, 2) +  
 (prev\_matrix[i][j + 1] + matrix[i][j - 1]) / pow(hy, 2))  
 eps = max(eps, abs(matrix[i][j] - prev\_matrix[i][j]))  
 }  
 }  
}  
  
const solution =   
[[0., -0.099, -0.192, -0.273, -0.336, -0.375, -0.384, -0.357, -0.288,   
-0.171, 0.], [-0.09, -0.14483, -0.20528, -0.26218, -0.30918, -0.3414,   
-0.35521, -0.34869, -0.32286, -0.28523, -0.261], [-0.16, -0.18609,   
-0.22251, -0.26147, -0.29735, -0.32648, -0.3472, -0.36061, -0.37192,   
-0.3933, -0.448], [-0.21, -0.21743, -0.23755, -0.26414, -0.29261,   
-0.32037, -0.34717, -0.37573, -0.4128, -0.47044, -0.567], [-0.24,   
-0.23625, -0.24642, -0.26525, -0.28897, -0.31572, -0.34602, -0.38324,   
-0.43419, -0.5096, -0.624], [-0.25, -0.2413, -0.24686, -0.2618,   
-0.28268, -0.308, -0.33853, -0.37769, -0.4318, -0.51023, -0.625],   
[-0.24, -0.2322, -0.23818, -0.25273, -0.27233, -0.29549, -0.32291,   
-0.35771, -0.40557, -0.47479, -0.576], [-0.21, -0.20949, -0.22119,   
-0.23893, -0.2588, -0.27912, -0.30035, -0.32511, -0.35836, -0.40763,   
-0.483], [-0.16, -0.17485, -0.19854, -0.22338, -0.24517, -0.26219,   
-0.27463, -0.28445, -0.2956, -0.31469, -0.352], [-0.09, -0.13204,   
-0.17529, -0.21125, -0.23659, -0.25012, -0.25186, -0.24288, -0.22551,   
-0.20428, -0.189], [0., -0.09, -0.16, -0.21, -0.24, -0.25, -0.24,   
-0.21, -0.16, -0.09, 0.]]  
  
console.log('Iterations count:', iters\_count)  
console.table(subtract(solution, matrix).map(row => row.map(v => Number(v.toPrecision(3)).toExponential())))**

**// poisson\_wolfram.nb**

**sol = NDSolveValue[{D[u[x, y], x, x] + D[u[x, y], y, y] ==**

**Exp[-x y^2], u[1, y] == (y - 2)\*(y - 3),**

**u[2, y] == y\*(y - 2)\*(y - 3), u[x, 2] == (x - 1)\*(x - 2),**

**u[x, 3] == x\*(x - 1)\*(x - 2)}, u, {x, 1, 2}, {y, 2, 3},**

**PrecisionGoal -> 10]**

**Plot3D[sol[x, y], {x, 1, 2}, {y, 2, 3}]**

**Table[Round[sol[x, y], .00001], {y, 3, 2, -.1}, {x, 1, 2, .1}]**

**MatrixForm[%]**

**// transcalency.js**

**require('console.table')  
const {cloneDeep} = require('lodash')  
const {max, pow, abs, exp} = Math  
const {eval, lusolve, subtract, transpose} = require('mathjs')  
  
const array = n => [...Array(n).keys()].map(\_ => 0)  
const range = (a, b, n = 1) => [...Array(n + 1).keys()].map(i => a + (b - a) / n \* i)  
  
const f = (x, t) => 0  
const u0 = x => exp(-x)  
const alpha0 = 0  
const beta0 = 1  
const mu0 = t => -exp(t)  
const alpha1 = 1  
const beta1 = 0  
const mu1 = t => exp(t - 1)  
const sigma = .5  
  
const EPS = 1E-5  
const N = 10  
const vx = range(0, 1, N)  
const vt = range(0, 1, N)  
const h = abs(vx[1] - vx[0])  
  
const matrix = array(N + 1).map(\_ => array(N + 1))  
matrix[0] = vx.map(u0)  
  
for (let j = 1; j <= N; ++j) {  
 const A = array(N + 1).map(\_ => array(N + 1))  
 const b = array(N + 1)  
 const u\_prev = matrix[j - 1]  
  
 A[0][0] = beta0 / h - alpha0 + (beta0 \* h) / (2 \* h)  
 A[0][1] = -beta0 / h  
 b[0] = - mu0(vx[j]) + (beta0 \* h) / (2 \* h) \* (u\_prev[0] + h \* f(vx[0], vt[j]))  
  
 A[N][N - 1] = -beta1 / h  
 A[N][N] = alpha1 + beta1 / h + beta1 \* h / (2 \* h)  
 b[N] = mu1(vt[j]) + beta1 \* h / (2 \* h) \* (u\_prev[N] + h \* f(vx[N], vt[j]))  
  
 for (let i = 1; i < N; ++i) {  
 A[i][i - 1] = -sigma/pow(h, 2)  
 A[i][i] = 1 / h + 2 \* sigma / pow(h, 2)  
 A[i][i + 1] = -sigma/pow(h, 2)  
 b[i] = u\_prev[i] / h + (1 - sigma) / pow(h, 2) \*   
 (u\_prev[i + 1] - 2 \* u\_prev[i] + u\_prev[i - 1]) + f(vx[i], vt[j - 1])  
 }  
  
 matrix[j] = transpose(lusolve(A, b))[0]  
}  
  
const prettify = row => row.map(v => Number(v.toPrecision(5)))  
console.table(matrix.map(prettify))  
  
const solution = [[1., 0.90484, 0.81873, 0.74082, 0.67032, 0.60653, 0.54881, 0.49659,   
 0.44933, 0.40657, 0.36788], [1.10518, 1.00001, 0.90485, 0.81874,   
 0.74083, 0.67033, 0.60654, 0.54882, 0.49659, 0.44933,   
 0.40657], [1.22143, 1.1052, 1.00002, 0.90486, 0.81875, 0.74084,   
 0.67034, 0.60655, 0.54882, 0.4966, 0.44934], [1.34989, 1.22143,   
 1.1052, 1.00003, 0.90486, 0.81875, 0.74084, 0.67034, 0.60654,   
 0.54882, 0.49659], [1.49187, 1.3499, 1.22144, 1.1052, 1.00003,   
 0.90486, 0.81875, 0.74084, 0.67034, 0.60654, 0.54882], [1.64879,   
 1.49189, 1.34992, 1.22146, 1.10522, 1.00004, 0.90487, 0.81876,   
 0.74085, 0.67034, 0.60655], [1.82223, 1.64882, 1.49192, 1.34994,   
 1.22148, 1.10524, 1.00006, 0.90489, 0.81878, 0.74086,   
 0.67035], [2.01391, 1.82226, 1.64885, 1.49194, 1.34997, 1.2215,   
 1.10526, 1.00007, 0.9049, 0.81878, 0.74086], [2.22576, 2.01395,   
 1.8223, 1.64888, 1.49197, 1.34999, 1.22152, 1.10527, 1.00008,   
 0.90491, 0.81879], [2.45985, 2.22576, 2.01396, 1.8223, 1.64889,   
 1.49197, 1.34999, 1.22151, 1.10526, 1.00007, 0.90489], [2.71853,   
 2.45983, 2.22575, 2.01394, 1.82229, 1.64887, 1.49195, 1.34997,   
 1.22149, 1.10524, 1.00005]]  
  
  
console.table(subtract(solution, matrix).map(row => row.map(v => Number(v.toPrecision(3)).toExponential())))**

**// transcalency\_wolfram.nb**

**sol = NDSolveValue[{D[u[x, t], t] == D[u[x, t], x, x],**

**u[x, 0] == Exp[-x], (D[u[x, t], x] /. x -> 0) == -Exp[t],**

**u[1, t] == Exp[t - 1]}, u, {x, 0, 1}, {t, 0, 1},**

**PrecisionGoal -> 10]**

**Plot3D[sol[x, t], {x, 0, 1}, {t, 0, 1}]**

**Table[Round[sol[x, t], .00001], {t, 0, 1, .1}, {x, 0, 1, .1}]**

**MatrixForm[%]**