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:

$$\frac{\partial^2(u)}{\partial x^2} + \frac{\partial^2(u)}{\partial y^2} = -f(x,y),$$

$$x \in [a,b], y \in [c,d]$$

$$u(a,y) = \psi_1(y), \quad u(b,y) = \psi_2(y)$$

$$u(x,c) = \psi_3(x), \quad u(x,d) = \psi_4(x)$$

Требуется найти значения функции на равномерной сетке узлов в прямоугольнике $[a,\ b] * [c,\ d]$.

1.2 Описание метода

Исходное дифференциальное уравнение записывается на сетке на пятиточечном шаблоне:

$$\frac{y_{i+1,j} - 2y_{ij} + y_{i-1,j}}{h_x^2} + \frac{y_{i,j+1} - 2y_{ij} + y_{i,j-1}}{h_y^2} = -f_{ij}, \ \ i = \overline{1, N_x - 1}, j = \overline{1, N_y - 1}$$

Данное уравнение можно решить с помощью метода Зейделя:

$$y_{ij}^{k+1} = (\frac{2}{h_x^2} + \frac{2}{h_y^2})^{-1}(f_{ij} + \frac{y_{i+1,j}^k + y_{i-1,j}^{k+1}}{h_x^2} + \frac{y_{i,j+1}^k + y_{i,j-1}^{k+1}}{h_y^2}), \ \ i = \overline{1, N_x - 1}, j = \overline{1, N_y - 1}$$

Двигаясь слева направо, сверху вниз получаем разностный метод решения задачи Дирихле с погрешностью $O({h_x}^2 + {h^2}_y)$. Итерации проводятся пока норма невязки изменения решения не окажется меньше $\varepsilon = O(min({h_x}^3, {h^3}_y))$.

1.3. Результаты

Условия задачи Дирихле:

$$\frac{\partial^2(u)}{\partial x^2} + \frac{\partial^2(u)}{\partial y^2} = -(-e^{-xy^2})$$

$$x \in [1,2], y \in [2,3]$$

$$u(a,y) = (y-2)(y-3) \quad u(b,y) = y(y-2)(y-3)$$

$$u(x,c) = (x-1)(x-2), \quad u(x,d) = x(x-1)(x-2)$$

Точное решения находилось с помощью Wolfram Mathematica.

```
→ cma git:(master) ✗ node ./poisson.js
Iterations count: 122
```

0	1	2	3	4	5	6	7	8	9	10
0e+0	8.33e-17	-5.55e-17	0e+0	-5.55e-17	0e+0	0e+0	-5.55e-17	0e+0	1.11e-16	0e+0
6.94e-17	6.74e-4	8.38e-4	9.7e-4	1.15e-3	1.39e-3	1.72e-3	2.19e-3	2.82e-3	3.2e-3	2.22e-16
1.11e-16	8.15e-4	1.32e-3	1.69e-3	2.06e-3	2.45e-3	2.89e-3	3.32e-3	3.54e-3	2.82e-3	2.78e-16
-5.55e-17	8.67e-4	1.58e-3	2.14e-3	2.63e-3	3.07e-3	3.43e-3	3.61e-3	3.34e-3	2.21e-3	-1.11e-16
0e+0	9.12e-4	1.71e-3	2.37e-3	2.89e-3	3.3e-3	3.54e-3	3.49e-3	2.96e-3	1.77e-3	0e+0
0e+0	9.33e-4	1.76e-3	2.42e-3	2.91e-3	3.24e-3	3.38e-3	3.2e-3	2.6e-3	1.48e-3	0e+0
0e+0	9.57e-4	1.74e-3	2.33e-3	2.75e-3	2.99e-3	3.06e-3	2.86e-3	2.29e-3	1.31e-3	0e+0
-5.55e-17	9.89e-4	1.68e-3	2.12e-3	2.39e-3	2.55e-3	2.59e-3	2.45e-3	2.03e-3	1.21e-3	-2.22e-16
1.11e-16	1.04e-3	1.5e-3	1.7e-3	1.82e-3	1.91e-3	1.95e-3	1.91e-3	1.73e-3	1.2e-3	2.78e-16
6.94e-17	1.02e-3	1.05e-3	1e-3	1.01e-3	1.03e-3	1.07e-3	1.11e-3	1.17e-3	1.11e-3	1.39e-16
0e+0	6.94e-17	-2.78e-17	2.78e-17	0e+0	0e+0	0e+0	-5.55e-17	-2.78e-17	6.94e-17	0e+0

1.4. Вывод

Как видно из результата, построенный метод обладает погрешностью не более $O(h_x^2 + h^2_y)$.
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.

$$\begin{cases} \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial^2 x} + f(x,t), \quad 0 \le x,t \le 1, \\ u(x,0) = u_0(x), \\ \alpha_0 u(0,t) + \beta_0 \frac{\partial u(0,t)}{\partial x} = \mu_0(t), \\ \alpha_1 u(1,t) + \beta_1 \frac{\partial u(1,t)}{\partial x} = \mu_1(t). \end{cases}$$

Требуется найти приближенное значение функции на равномерной сетке узлов в прямоугольнике $[a,\ b] * [c,\ d]$ с погрешностью не хуже $O(h^2 + \tau)$.

2.2 Описание метода

Записав условия на шеститочечном шаблоне, получим:

$$-\frac{\sigma}{h^2} y_{i-1,j} + (\frac{1}{\tau} + \frac{2\sigma}{h^2})\, y_{i,j} - \frac{\sigma}{h^2} y_{i+1,j} = \frac{y_{i,j-1}}{\tau} + \frac{1-\sigma}{h^2}(y_{i+1,j-1} - 2_{i,j-1} + y_{i-1,j-1}) + f_{ij-1},\ i = \overline{1,\ N-1}$$

$$(\frac{\beta_0}{h} - \alpha_0 + \frac{\beta_0 h}{2\tau})\, y_{0,j} - \frac{\beta_0}{h} y_{1,j} = -\mu_{0,j} + \frac{\beta_0 h}{2\tau}(y_{0,j-1} + \tau f_{0,j})$$

$$(-\frac{\beta_1}{h} y_{N-1,j} + (\alpha_1 + \frac{\beta_1}{h} + \frac{\beta_1 h}{2\tau})\, y_{N,j} = -\mu_{1,j} + \frac{\beta_1 h}{2\tau}(y_{N,j-1} + \tau f_{N,j})$$

Первый ряд $y_{i,0}$ известен из условий, остальные вычисляются последовательно.

2.3. Результаты

Условия:

$$\begin{cases} \frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial^2 x}, \quad 0 \le x, t \le 1, \\ u(x,0) = e^{-x}, \\ \frac{\partial u(0,t)}{\partial x} = -e^t, \\ u(1,t) = e^{t-1}, \\ \sigma = 0.5. \end{cases}$$

```
➜  cma git:(master) ✗ node ./transcalency.js
```

0	1	2	3	4	5	6	7	8	9	10
1	0.90484	0.81873	0.74082	0.67032	0.60653	0.54881	0.49659	0.44933	0.40657	0.36788
1.1055	1.0003	0.90505	0.8189	0.74096	0.67044	0.60663	0.54889	0.49664	0.44936	0.40657
1.222	1.1057	1.0005	0.90523	0.81906	0.74109	0.67054	0.6067	0.54893	0.49664	0.44933
1.3507	1.2222	1.1059	1.0006	0.90535	0.81916	0.74117	0.67059	0.60671	0.54891	0.49659
1.4929	1.3509	1.2223	1.106	1.0007	0.90541	0.8192	0.74118	0.67056	0.60666	0.54881
1.6501	1.493	1.351	1.2224	1.106	1.0007	0.90542	0.81918	0.74112	0.67048	0.60653
1.8237	1.6502	1.4931	1.351	1.2224	1.106	1.0007	0.90537	0.81909	0.741	0.67032
2.0156	1.8238	1.6502	1.4932	1.351	1.2224	1.106	1.0006	0.90526	0.81895	0.74082
2.2276	2.0156	1.8238	1.6502	1.4932	1.351	1.2223	1.1059	1.0005	0.90509	0.81873
2.4619	2.2277	2.0157	1.8238	1.6502	1.4931	1.3509	1.2222	1.1057	1.0003	0.90484
2.7209	2.462	2.2277	2.0157	1.8238	1.6502	1.493	1.3508	1.222	1.1055	1

0	1	2	3	4	5	6	7	8	9	10
0e+0	2.58e-6	-7.53e-7	1.78e-6	-4.6e-8	-6.6e-7	-1.64e-6	4.7e-6	1.04e-6	3.4e-7	5.59e-7
-3.23e-4	-2.51e-4	-1.97e-4	-1.64e-4	-1.32e-4	-1.11e-4	-9.05e-5	-7.15e-5	-5.38e-5	-3.2e-5	3.4e-7
-6.04e-4	-5.2e-4	-4.5e-4	-3.75e-4	-3.13e-4	-2.52e-4	-1.99e-4	-1.47e-4	-1.05e-4	-4.37e-5	1.1e-5
-8.43e-4	-7.53e-4	-6.61e-4	-5.72e-4	-4.93e-4	-4.12e-4	-3.26e-4	-2.45e-4	-1.72e-4	-8.6e-5	4.7e-6
-1.07e-3	-9.66e-4	-8.61e-4	-7.61e-4	-6.53e-4	-5.54e-4	-4.49e-4	-3.37e-4	-2.25e-4	-1.16e-4	8.36e-6
-1.27e-3	-1.16e-3	-1.04e-3	-9.17e-4	-7.97e-4	-6.76e-4	-5.5e-4	-4.17e-4	-2.73e-4	-1.37e-4	1.93e-5
-1.46e-3	-1.34e-3	-1.2e-3	-1.07e-3	-9.28e-4	-7.84e-4	-6.36e-4	-4.8e-4	-3.14e-4	-1.44e-4	3e-5
-1.65e-3	-1.51e-3	-1.37e-3	-1.22e-3	-1.06e-3	-8.94e-4	-7.2e-4	-5.5e-4	-3.6e-4	-1.68e-4	4.18e-5
-1.83e-3	-1.69e-3	-1.53e-3	-1.37e-3	-1.19e-3	-1e-3	-8.08e-4	-6.1e-4	-4.04e-4	-1.75e-4	5.92e-5
-2.06e-3	-1.9e-3	-1.72e-3	-1.54e-3	-1.34e-3	-1.14e-3	-9.16e-4	-6.96e-4	-4.59e-4	-2.11e-4	5.26e-5
-2.33e-3	-2.15e-3	-1.95e-3	-1.75e-3	-1.52e-3	-1.29e-3	-1.05e-3	-7.91e-4	-5.28e-4	-2.46e-4	5e-5

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[%]
```