Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Курсовой проект
по курсу «Численные методы»

Студент: А. А. Садаков
Группа: М8О-406Б-19

Москва, 2023

Лабораторная работа №5

Задача: Используя явную и неявную конечно-разностные схемы, а также схему Кранка - Николсона, решить начально-краевую задачу для дифференциального уравнения параболического типа. Осуществить реализацию трех вариантов аппроксимации граничных условий, содержащих производные: двухточечная аппроксимация с первым порядком, трехточечная аппроксимация со вторым порядком, двухточечная аппроксимация со вторым порядком. В различные моменты времени вычислить погрешность численного решения путем сравнения результатов с приведенным в задании аналитическим решением $U(x,t)$. Исследовать зависимость погрешности от сеточных параметров τ, h.

Вариант: 5

Начально-краевая задача:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \dfrac{\partial^2 u}{\partial x^2} + \sin(\pi x), \\ u(0,t) = 0, \\ u(1,t) = 0, \\ u(x,0) = 0 \end{cases}$$

Точное решение:

$$U(x,t) = \frac{1}{\pi^2}(1 - \exp(-\pi^2 t))\sin(\pi x)$$

Вывод программы для шагов $h_x = 0.01$ и $h_t = 0.01$:

=====5.1=====

Введите начально-краевую задачу:

Введите размер шага для "x"и для "t":

Введите функцию для сравнения:

Размер таблицы функции U: 100x1000

Point 2 Order 1

Средняя погрешность явного метода: nan

Средняя погрешность неявного метода: 0.002

Средняя погрешность метода Кранка-Николаса: 0.002

Point 2 Order 2
Средняя погрешность явного метода: nan
Средняя погрешность неявного метода: 0.002
Средняя погрешность метода Кранка-Николаса: 0.002
Point 3 Order 2
Средняя погрешность явного метода: nan
Средняя погрешность неявного метода: 0.002
Средняя погрешность метода Кранка-Николаса: 0.002

Вывод программы для шагов $h_x = 0.001$ и $h_t = 0.01$:

=====5.1=====
Введите начально-краевую задачу:
Введите размер шага для "x"и для "t":
Введите функцию для сравнения:
Размер таблицы функции U: 1000x1000
Point 2 Order 1
Средняя погрешность явного метода: nan
Средняя погрешность неявного метода: 0.000
Средняя погрешность метода Кранка-Николаса: 0.000
Point 2 Order 2
Средняя погрешность явного метода: nan
Средняя погрешность неявного метода: 0.000
Средняя погрешность метода Кранка-Николаса: 0.000
Point 3 Order 2
Средняя погрешность явного метода: nan
Средняя погрешность неявного метода: 0.000
Средняя погрешность метода Кранка-Николаса: 0.000

Явный метод расходится, так как является условно устойчивым и зависит от размеров шагов сетки.

Исходный код

5-1.cpp:

```cpp
#include "5-1.hpp"

double Point2Order1 (const std::vector<double> &coeff, double h, double u1, double f,
    uint64_t i) {
    double alpha = coeff[0], beta = coeff[1];
    double ans = 0;
    if (i == 0) {
        ans += f - alpha * u1 / h;
        ans /= beta - alpha / h;
    } else {
        ans += f + alpha * u1 / h;
        ans /= beta + alpha / h;
    }
    return ans;
}

double Point2Order2 (const std::vector<double> &ux, const std::vector<double> &coeff,
    double h, double t, double u0, double u1, double f, uint64_t i) {
    double alpha = ux[0], beta = ux[1];
    double a = coeff[0], b = coeff[1], c = coeff[2];
    double ans = 0;
    if (i == 0) {
        ans += h / t * u0 - f * (2 * a - b * h) / alpha + 2 * u1 * a / h;
        ans /= 2 * a / h + h / t - c * h - (beta / alpha) * (2 * a - b * h);
    } else {
        ans += h / t * u0 + f * (2 * a + b * h) / alpha + 2 * u1 * a / h;
        ans /= 2 * a / h + h / t - c * h + (beta / alpha) * (2 * a + b * h);
    }
    return ans;
}

double Point3Order2 (const std::vector<double> &coeff, double h, double u1, double u2,
    double f, uint64_t i) {
    double alpha = coeff[0], beta = coeff[1];
    double ans = 0;
    if (i == 0) {
        ans += f - alpha * (4 * u1 - u2) / (2 * h);
        ans /= beta - 3 * alpha / (2 * h);
    } else {
        ans += f + alpha * (4 * u1 - u2) / (2 * h);
        ans /= beta + 3 * alpha / (2 * h);
    }
    return ans;
}

```

```cpp
void FiniteDifferenceExplicitApprox (std::vector<std::vector<double>> &u, double X0,
    double xh, double th, const std::vector<double> &coeff, const std::function<double(
    double, double)> &f, uint64_t id) {
    double a = coeff[0], b = coeff[1], c = coeff[2];
    double sigmaA = a * th / (xh * xh), sigmaB = b * th / (2 * xh), sigmaC = c * th;
    uint64_t idx[] = {1, u[id].size() - 3};
    for (uint64_t i = 0; i < 2; ++i) {
        for (uint64_t j = id; j < id + 1; ++j) {
            for (uint64_t k = idx[i]; k < idx[i] + 2; ++k) {
                double tmp = 0;
                tmp += sigmaA * (u[j - 1][k + 1] - 2 * u[j - 1][k] + u[j - 1][k - 1]);
                tmp += sigmaB * (u[j - 1][k + 1] - u[j - 1][k]);
                tmp += sigmaC * u[j - 1][k];
                tmp += f(X0 + k * xh, j * th) * th;
                u[j][k] = tmp + u[j - 1][k];
            }
        }
    }
}

void ExplNonExplIteration (double theta, std::vector<std::vector<double>> &u, double X0,
    double xh, double th, const std::vector<double> &coeff, const std::function<double(
    double, double)> &f, uint64_t i) {
    double a = coeff[0], b = coeff[1], c = coeff[2];
    double sigmaA = a * th / (xh * xh), sigmaB = b * th / (2 * xh), sigmaC = c * th;

    uint64_t n = u[0].size() - 2;
    Matrix<double> M(n, n);
    std::vector<double> ans(n);

    M(0, 0) = -1 - (2 * sigmaA - sigmaB + sigmaC) * theta;
    M(0, 1) = (sigmaA + sigmaB) * theta;
    ans[0] = -u[i - 1][1] - (sigmaA + sigmaB + sigmaC) * u[i][0] * theta;
    ans[0] += -(sigmaA * (u[i - 1][2] - 2 * u[i - 1][1] + u[i - 1][0]) + sigmaB * (u[i -
        1][2] - u[i - 1][1]) + sigmaC * u[i - 1][1]) * (1 - theta);
    ans[0] += -f(X0 + xh, i * th) * th;

    for (uint64_t j = 1; j < n - 1; ++j) {
        M(j, j - 1) = sigmaA * theta;
        M(j, j) = -1 - (2 * sigmaA - sigmaB + sigmaC) * theta;
        M(j, j + 1) = (sigmaA + sigmaB) * theta;
        ans[j] = -u[i - 1][j + 1];
        ans[j] += -(sigmaA * (u[i - 1][j + 2] - 2 * u[i - 1][j + 1] + u[i - 1][j]) + sigmaB
             * (u[i - 1][j + 2] - u[i - 1][j + 1]) + sigmaC * u[i - 1][j + 1]) * (1 - theta
            );
        ans[j] += -f(X0 + (j + 1) * xh, i * th) * th;
    }

```

```
84      M(n - 1, n - 2) = sigmaA * theta;
85      M(n - 1, n - 1) = -1 - (2 * sigmaA - sigmaB + sigmaC) * theta;
86      ans[n - 1] = -u[i - 1][n] - (sigmaA + sigmaB + sigmaC) * u[i][n + 1] * theta;
87      ans[n - 1] += -(sigmaA * (u[i - 1][n + 1] - 2 * u[i - 1][n] + u[i - 1][n - 1]) + sigmaB
            * (u[i - 1][n + 1] - u[i - 1][n]) + sigmaC * u[i - 1][n]) * (1 - theta);
88      ans[n - 1] += -f(X0 + xh * n, i * th) * th;
89
90      if (theta == 0.0) {
91          for (uint64_t j = 0; j < ans.size(); ++j) {
92              ans[j] *= -1;
93          }
94      } else {
95          ans = RUNsolveSLAE(M, ans);
96      }
97
98      for (uint64_t j = 0; j < ans.size(); ++j) {
99          u[i][j + 1] = ans[j];
100     }
101 }
102
103 void ExplNonExpl (double theta, std::vector<std::vector<double>> &u, const std::vector<std
        ::vector<double>> &ux, double X0, double xh, double th, const std::vector<double> &
        coeff, const std::function<double(double, double)> &f, ApproxLevel left, ApproxLevel
        right) {
104     if (theta < 0 || theta > 1) {
105         throw std::logic_error("ExplNonExpl: theta must be in range [0, 1]");
106     }
107     for (uint64_t i = 1; i < u.size(); ++i) {
108         FiniteDifferenceExplicitApprox(u, X0, xh, th, coeff, f, i);
109         uint64_t start = 0, end = u[i].size() - 1;
110         switch (left) {
111             case ApproxLevel::POINT2_ORDER1:
112                 u[i][start] = Point2Order1(ux[0], xh, u[i][start + 1], u[i][start], 0);
113                 break;
114             case ApproxLevel::POINT2_ORDER2:
115                 u[i][start] = Point2Order2(ux[0], coeff, xh, th, u[i - 1][start], u[i][start
                        + 1], u[i][start], 0);
116                 break;
117             case ApproxLevel::POINT3_ORDER2:
118                 u[i][start] = Point3Order2(ux[0], xh, u[i][start + 1], u[i][start + 2], u[i
                        ][start], 0);
119                 break;
120             default:
121                 break;
122         }
123         switch (right) {
124             case ApproxLevel::POINT2_ORDER1:
125                 u[i][end] = Point2Order1(ux[1], xh, u[i][end - 1], u[i][end], 0);
```

```
126                 break;
127             case ApproxLevel::POINT2_ORDER2:
128                 u[i][end] = Point2Order2(ux[1], coeff, xh, th, u[i - 1][end], u[i][end - 1],
                        u[i][end], 0);
129                 break;
130             case ApproxLevel::POINT3_ORDER2:
131                 u[i][end] = Point3Order2(ux[1], xh, u[i][end - 1], u[i][end - 2], u[i][end],
                        0);
132                 break;
133             default:
134                 break;
135         }
136         ExplNonExplIteration(theta, u, X0, xh, th, coeff, f, i);
137     }
138 }
139
140 std::vector<std::vector<double>> SolveIBVP (const Task &task, double timeLimit, double xh,
        double th, Method method, ApproxLevel approx) {
141     double X1 = task.X[0], X2 = task.X[1];
142     std::vector<std::vector<double>> u(uint64_t(timeLimit / th), std::vector<double>(
            uint64_t((X2 - X1) / xh), 0));
143     ApproxLevel left = ApproxLevel::NONE, right = ApproxLevel::NONE;
144     if (task.ux[0][0] != 0) {
145         left = approx;
146     }
147     if (task.ux[1][0] != 0) {
148         right = approx;
149     }
150     auto f = [&] (double x, double t) -> double {
151         return task.trees[0]({0, 0, 0, x, t});
152     };
153     auto fx0 = [&] (double t) -> double {
154         return task.trees[1](t);
155     };
156     auto fxl = [&] (double t) -> double {
157         return task.trees[2](t);
158     };
159     auto ft0 = [&] (double x) -> double {
160         return task.trees[3](x);
161     };
162     for (uint64_t i = 0; i < u[0].size(); ++i) {
163         u[0][i] = ft0(i * xh);
164     }
165     for (uint64_t i = 1; i < u.size(); ++i) {
166         u[i].front() = fx0(i * th);
167         u[i].back() = fxl(i * th);
168     }
169     switch (method) {
```

```
        case Method::EXPLICIT:
            ExplNonExpl(0, u, task.ux, X1, xh, th, task.coeff, f, left, right);
            break;
        case Method::NOT_EXPLICIT:
            ExplNonExpl(1, u, task.ux, X1, xh, th, task.coeff, f, left, right);
            break;
        case Method::KRANK_NICOLAS:
            ExplNonExpl(0.5, u, task.ux, X1, xh, th, task.coeff, f, left, right);
            break;
        default:
            break;
    }
    return u;
}
```

Выводы

Точность приближённого решения сильно зависит от размера шага h_x и мало зависит от шага h_t. Из-за этого приходится ставить большой шаг h_t и маленький h_x для сохранения быстроты решения из-за чего явный метод расходится.