НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Механико-математический факультет
Кафедра: Математика и компьютерные науки

Тлепбергенова Дарья Дулатовна

Отчет по вычислительному практикуму

Решение уравнения теплопроводности с помощью неявного метода Эйлера. Вариант 12.

3 курс, группа 16121

Преподаватель:
Махоткин Олег Александрович

Новосибирск, 2018 г.

1. Постановка задачи.

Дано уравнение теплопроводности в виде:

$$\begin{cases} \frac{\partial u}{\partial t} = a^2 \dfrac{\partial^2 u}{\partial x^2} + f(x,t) \\ 0 \leq x \leq 1, 0 \leq t \leq 1 \\ u(x,0) = \mu(x) \\ u(0,t) = \mu_1(t) \\ u(1,t) = \mu_2(t) \end{cases} \tag{1}$$

Для $\mu(x) = -4x^4 + 2x^2$,
$\mu_1(t) = t^2 - t$,
$\mu_2(t) = 1 + t + t^2 - te^x$,
$f(x,t) = x + 2t - e^x + a(12x^2 - 4 + te^x)$,
$u(x,t) = -x^4 + 2x^2 + tx + t^2 - te^x$,
$a = 0.021$

Выполнить следующие пункты:

1) Исследовать данную схему на точность и устойчивость.

2) Проверить, что $u(x,t)$ является решением краевой задачи.

3) Написать программу для решения уравнения теплопроводности методом конечных разностей.

4) Выводить на экран значения относительной погрешности разностного решения в заданных контрольных точках по t. Использовать любую из трех основных норм вектора.

2. Описание вычислительного метода.

Перейдем к дискретной постановке задачи: разобьем наши промежутки по x и по t на N_x и N_t равных частей соответственно. Получим систему:

$$\left\{ \begin{array}{l} \frac{u_k^{j+1} - u_k^j}{\tau} = a^2 \frac{u_{k+1}^{j+1} - 2u_k^{j+1} + u_{k-1}^{j+1}}{h^2} + f_k^{j+1} \\ \left\{ x_k = kh : h = \frac{1}{N_x}, k = 0..N_x \right\} \\ \left\{ t_j = j\tau : \tau = \frac{1}{N_t}, j = 0..N_t \right\} \\ u_k^0 = \mu(x_k) \\ u_0^t = \mu_1(t_j) \\ u_{N_x}^t = \mu_2(t_j) \end{array} \right. \tag{2}$$

3. Исследование данной схемы на точность и устойчивость.

Погрешность аппроксимации:

$$\psi_k^j = f(x_i, t_{n+1}) - \frac{u(x_k, t_{j+1} - u(x_k, t_j)}{\tau} + a\frac{u(x_{k+1}, t_{j+1}) - 2u(x_k, t_{j+1}) + u(x_{k-1}, t_{j+1})}{h^2} =$$

Разложим в Ряд Тейлора в x_i, где t'$\in [t_j, t_{j+1}]$:

$$= f - \frac{\partial u}{\partial t} + \frac{\tau}{2}\frac{\partial^2 u}{\partial t^2} + a\frac{\partial^2 u}{\partial x^2} + \frac{a^2 h^2}{24}(\frac{\partial^4 u}{\partial x^4} + \frac{\partial^4 u}{\partial x^4}) = O(\tau + h^2)$$

Погрешность решения:

$$L_{\tau,h}\xi^{\tau,h} = \psi^{\tau,h}$$
$$\xi_k^j = u_k^j - u(x_k, t_j)$$
$$(1 + \frac{2a^2\tau}{h^2})\xi_k^{j+1} = \xi_k^j + \frac{a^2\tau}{h^2}(\xi_{k-1}^{j+1} + \xi_{k+1}^{j+1}) + \tau\psi_k^j$$

Возьмем модуль:

$$(1 + \frac{2a^2\tau}{h^2})|\xi_k^{j+1}| \le |\xi_k^j| + \frac{a^2\tau}{h^2}(|\xi_{k-1}^{j+1}| + |\xi_{k+1}^{j+1}|) + \tau|\psi_k^j| \le$$
$$\le \max_{k=0..N_x}|\xi_k^j| + \frac{a^2\tau}{h^2}(\max_{k=0..N_x}|\xi_{k-1}^{j+1}| + \max_{k=0..N_x}|\xi_{k+1}^{j+1}|) + \tau\max_{k,j=0..N_x,N_t}|\psi_k^j|$$

Обозначим через $\delta^j = \max_{k=0..N_x}|\xi_k^j|$получим неравенство:

$$\delta^j \le \delta^{j+1} + \tau\max_{k,j=0..N_x,N_t}|\psi_k^j|$$

Учитывая то, что $\delta^0 = \max_{k=0..N_x}\mu(x_k)$ получаем:

$$\delta^j \le \max_{k=0..N_x}\mu(x_k) + j\tau\max_{k,j=0..N_x,N_t}|\psi_k^j|$$
$$\Rightarrow \max_{k=0..N_x}|\xi_k^j| = O(\tau + h^2)$$

4. Проверим, что $u(x,t)$ - решение системы.

Подставим в систему (1):

$$\frac{\partial u}{\partial t} = x + 2t - e^x = a(-12x^2 + 4 - te^x) + x + 2t - e^x + a(12x^2 - 4 + te^x)$$
$$\frac{\partial u}{\partial t} = -4x^3 + 4x + t - te^x$$

Проверим краевые условия:

$$u(x,0) = -x^4 + 2x^2 = \mu(x)$$
$$u(0,t) = t^2 - t = \mu_1(t)$$
$$u(1,t) = -1 + 2 + t + t^2 - te = \mu_2(t)$$

Значит функция $u(x,t)$ является решением системы (1)

5. Описание алгорима.

- Main class
  - Создаем поля для разбиения сетки, коэффициента для второй производной (main class для простоты замены начальных данных)
  - Создаем функции краевых условий и решения (main class для простоты замены начальных данных)
  - В main функции обращаемся к методу решения уравнения теплопроводности и запускаем визуализацию на питоне сначала для решения, потом для точного решения
- HeatEquation class
  - создаем поле аналога числа Куранта (для упрощения формул вычислений)
  - создаем поля для шага по t, x
  - функция для решения уравнения теплопроводности:
    * создаем двумерный массив для записи решения
    * заполняем его первую строку и первый и последний столбец начальными значениями
    * для оставшихся строк решение находится построчно методом прогонки через предыдущее:
    * функция метода прогонки с постоянными коэффициентами:

- в данном случае диагональные элементы одинаковы и над/поддиагональные тоже, по этому заводим для них 2 переменные
- также заводим два массива из метода прогонки и находим для них коэффициенты с последнего до первого (в силу диагонального преобладания)
- далее находим координаты искомого вектора, начиная с первого.
- возвращаем этот вектор.

* после нахождения всех коэффициентов матрицы решений, записываем ее в текстовый файл для графического вывода, подсчитываем максимальную ошибку (через максимум модуля) и печатаем ее.

6. Код программы (на Java).

6.1. Класс HeatEquation

```
package ru.nsu.mmf.g16121.ddt.math;

import java.io.FileNotFoundException;
import java.io.PrintWriter;

import static ru.nsu.mmf.g16121.ddt.main.Main.*;

public class HeatEquations {

    private static final double stepX = (rightBound - leftBound) /
            NUMBERS_COUNT_OF_GRID_BY_X;
    private static final double stepT = (rightBound - leftBound) /
            NUMBERS_COUNT_OF_GRID_BY_T;

    private static final double COURANT_NUMBER_ANALOGUE =
            2.0 * COEFFICIENT_AT_SECOND_DERIVATIVE * stepT /
            Math.pow(stepX, 2);

    /**
     * @param rightPart - this is array of the right part of linear
     *                  equation system Au = rightPart, where A is
     *                  triDiagonal matrix with coeff:
     *                  diag - is coefficient of the matrix A on the
     *                  diagonal;
     *                  overDiagonal - the coefficient of the matrix
     *                  And the overdiagonal and subdiagonal;
```

```
 *                  <p>
 * @return solution matrix of a linear system.
 */

private static double[] sweepMethodWithConstCoef(
double[] rightPart) {
    double diag = 1.0 + COURANT_NUMBER_ANALOGUE;
    double overDiagonal = -COURANT_NUMBER_ANALOGUE * 0.5;

    double[] u = new double[rightPart.length];

    double[] alpha = new double[rightPart.length - 1];
    double[] beta = new double[rightPart.length - 1];

    alpha[rightPart.length - 2] = -overDiagonal / diag;
    beta[rightPart.length - 2] = rightPart[rightPart.length - 1] /
    diag;

    for (int i = rightPart.length - 3; i >= 0; i--) {
        alpha[i] = -(overDiagonal / (diag + overDiagonal *
        alpha[i + 1]));
        beta[i] = ((rightPart[i + 1] - beta[i + 1] * overDiagonal)
        /(diag + overDiagonal * alpha[i + 1]));
    }

    u[0] = ((rightPart[0] - overDiagonal * beta[0]) / (diag +
            overDiagonal * alpha[0]));
    for (int i = 1; i < rightPart.length; i++) {
        u[i] = alpha[i - 1] * u[i - 1] + beta[i - 1];
    }
    return u;
}

/**
 * Tn this method (<>writeInTxt</>) we write points surface in
 * txt file: x - in 1st column, t - in 2d column,
 * exact value of the func - 3d column and our func value in
 * 4d column (for gnuplot)
 */

private static void writeForGnuplot(double[][] u) throws
FileNotFoundException {
    PrintWriter writer = new
    PrintWriter("functions_for_Gnuplot.txt");
```

```
    double x;
    double t = leftBound;
    for (int i = 0; i <= NUMBERS_COUNT_OF_GRID_BY_T; i++) {
        x = leftBound;
        for (int j = 0; j <= NUMBERS_COUNT_OF_GRID_BY_X; j++) {
            writer.println(x + "\t" + t + "\t" + u(x, t) + "\t"
            + u[i][j]);
            x += stepX;
        }
        t += stepT;
    }
    writer.close();
}

private static void writeResultForPython(double[][] u) throws
FileNotFoundException {
    PrintWriter writer = new PrintWriter("result.txt");

    writer.print("[[" + (int) leftBound + ", " + (int) rightBound
            + ", " + (NUMBERS_COUNT_OF_GRID_BY_X + 1) + "],");
    writer.print("[" + (int) leftBound + ", " + (int) rightBound
            + ", " + (NUMBERS_COUNT_OF_GRID_BY_T + 1) + "],");

    double x;
    double t = leftBound;

    writer.print("[");
    for (int i = 0; i <= NUMBERS_COUNT_OF_GRID_BY_T; i++) {
        x = leftBound;
        writer.print("[");
        for (int j = 0; j < NUMBERS_COUNT_OF_GRID_BY_X; j++) {
            writer.print(u[i][j] + ",");
            x += stepX;
        }
        writer.print(u[i][NUMBERS_COUNT_OF_GRID_BY_X]);
        if (i == NUMBERS_COUNT_OF_GRID_BY_T) {
            writer.print("]");
        } else {
            writer.print("],");
        }
        t += stepT;
    }
    writer.print("]]");
```

```
    writer.close();
}

private static void writeMainFuncForPython() throws
FileNotFoundException {
    PrintWriter writer = new PrintWriter("mainFunc.txt");

    writer.print("[[" + (int) leftBound + ", " + (int) rightBound
            + ", " + (NUMBERS_COUNT_OF_GRID_BY_X + 1) + "],");
    writer.print("[" + (int) leftBound + ", " + (int) rightBound
            + ", " + (NUMBERS_COUNT_OF_GRID_BY_T + 1) + "],");

    double x;
    double t = leftBound;

    writer.print("[");
    for (int i = 0; i <= NUMBERS_COUNT_OF_GRID_BY_T; i++) {
        x = leftBound;
        writer.print("[");
        for (int j = 0; j < NUMBERS_COUNT_OF_GRID_BY_X; j++) {
            writer.print(u(x, t) + ",");
            x += stepX;
        }
        writer.print(u(x, t));
        if (i == NUMBERS_COUNT_OF_GRID_BY_T) {
            writer.print("]");
        } else {
            writer.print("],");
        }
        t += stepT;
    }
    writer.print("]]");
    writer.close();
}

private static double maxError(double[][] u) {

    double x = leftBound;
    double t = leftBound;
    double max = Math.abs(u[0][0] - u(x, t));
    for (int i = 0; i <= NUMBERS_COUNT_OF_GRID_BY_T; i++) {
        x = leftBound;
        for (int j = 0; j <= NUMBERS_COUNT_OF_GRID_BY_X; j++) {
            double error = Math.abs(u(x, t) - u[i][j]);
```

```
            if (error > max) {
                max = error;
            }
            x += stepX;
        }
        t += stepT;
    }
    return max;
}

/**
 * Tn this method (<>solveHeatEquation</>) solve the heat
 * equation solves the heat equation using the Euler method,
 * with initial conditions, and writes data to a text file
 * to display the result.
 */

public static void solveHeatEquation()
throws FileNotFoundException {
    double[][] u = new double[NUMBERS_COUNT_OF_GRID_BY_T + 1]
            [NUMBERS_COUNT_OF_GRID_BY_X + 1];

    //The first row of the matrix is filled by the initial data.
    double x = leftBound;
    for (int i = 0; i <= NUMBERS_COUNT_OF_GRID_BY_X; i++) {
        u[0][i] = mu(x);
        x += stepX;
    }

    //The first and second columns are filled with source data.
    double t = leftBound;
    for (int j = 0; j <= NUMBERS_COUNT_OF_GRID_BY_T; j++) {
        u[j][0] = mu1(t);
        u[j][NUMBERS_COUNT_OF_GRID_BY_X] = mu2(t);
        t += stepT;
    }

    //build the right part for the sweep method
    t = leftBound;
    for (int j = 0; j <= NUMBERS_COUNT_OF_GRID_BY_T - 1; j++) {
        x = leftBound + stepX;
        double[] rightPart =
        new double[NUMBERS_COUNT_OF_GRID_BY_X - 1];
        for (int i = 0; i <= NUMBERS_COUNT_OF_GRID_BY_X - 2; i++) {
```

```
                rightPart[i] = u[j][i + 1] + stepT * f(x, t + stepT);
                x += stepX;
            }
            rightPart[0] += COURANT_NUMBER_ANALOGUE * 0.5 *
            mu1(t + stepT);
            rightPart[NUMBERS_COUNT_OF_GRID_BY_X - 2] +=
            COURANT_NUMBER_ANALOGUE * 0.5 * mu2(t + stepT);

            //fill the rest of the matrix
            System.arraycopy(sweepMethodWithConstCoef(rightPart),
                    0, u[j + 1], 1, NUMBERS_COUNT_OF_GRID_BY_X - 1);
            t += stepT;
        }

        //write in the txt for display the result
        writeForGnuplot(u);
        writeMainFuncForPython();
        writeResultForPython(u);
        System.out.println("Max error = " + maxError(u));
    }
}
```

6.2. Класс Main

```
package ru.nsu.mmf.g16121.ddt.main;

import ru.nsu.mmf.g16121.ddt.math.HeatEquations;

import java.io.IOException;

public class Main {
    public static final double leftBound = 0;
    public static final double rightBound = 1;

    public static final int NUMBERS_COUNT_OF_GRID_BY_X = 10;
    public static final int NUMBERS_COUNT_OF_GRID_BY_T = 20;

    public static final double
            COEFFICIENT_AT_SECOND_DERIVATIVE = 0.021;

    public static double f(double x, double t) {
        return (x + 2.0 * t - Math.exp(x) +
                COEFFICIENT_AT_SECOND_DERIVATIVE * (12.0 *
                        Math.pow(x, 2) - 4.0 + t * Math.exp(x)));
```

```
    }

    public static double mu(double x) {
        return -Math.pow(x, 4) + 2.0 * Math.pow(x, 2);
    }

    public static double mu1(double t) {

        return Math.pow(t, 2) - t;
    }

    public static double mu2(double t) {
        return 1 + t + Math.pow(t, 2) - t * Math.E;
    }

    public static double u(double x, double t) {
        return -Math.pow(x, 4) + 2.0 * Math.pow(x, 2)
                + t * x + Math.pow(t, 2) - t * Math.exp(x);
    }

    public static void main(String[] args) throws IOException {
        HeatEquations.solveHeatEquation();

        Runtime.getRuntime().exec("python3 vizualization.py");
        Runtime.getRuntime().exec("python3 vizualization2.py");
    }
}
```

7. Графический вывод (Тесты)

При $h = 5$ и $\tau = 5$

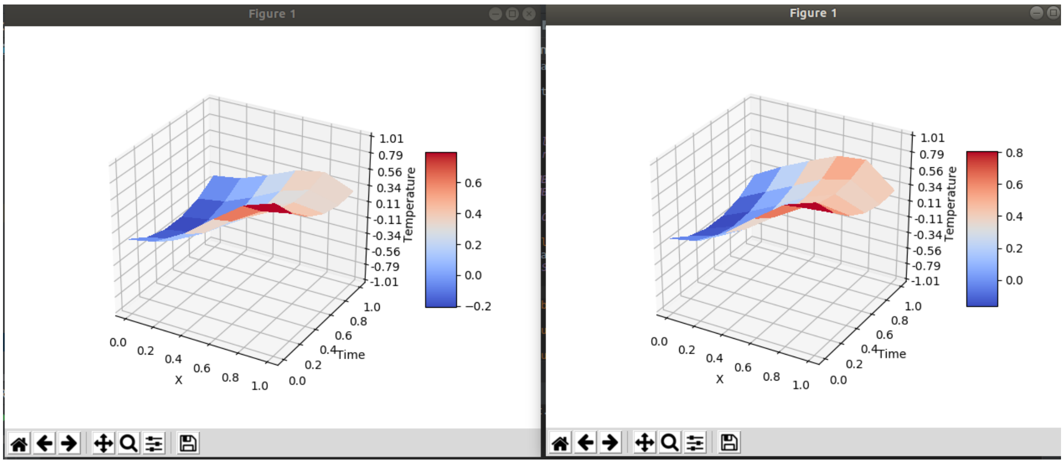
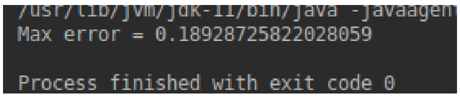


Рис. 1. Слева - решение системы неявным методом Эйлера, справа - точное решение

Тогда максимальная ошибка:



```
/usr/lib/jvm/jdk-11/bin/java -javaagen
Max error = 0.18928725822028059

Process finished with exit code 0
```

Теперь увеличим h в 2 раза, а τ в 4 раза:
При $h = 10$ и $\tau = 20$:

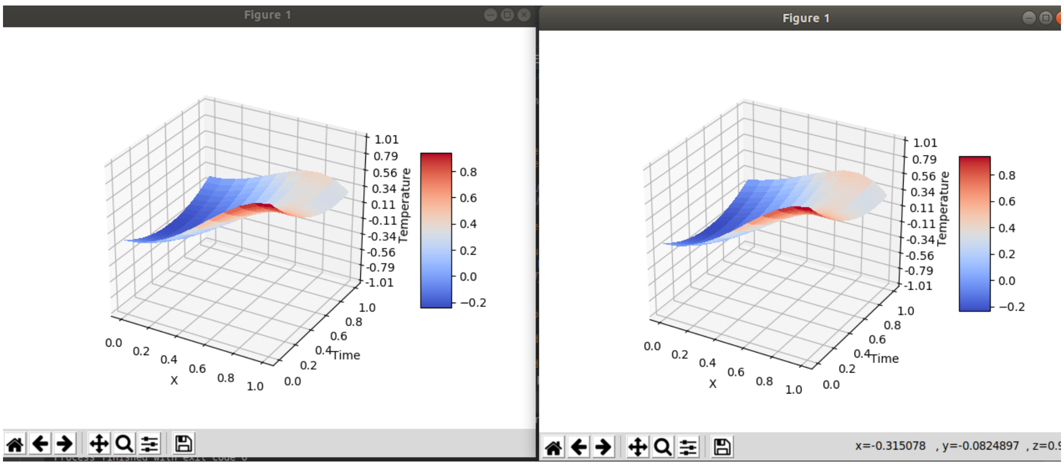


Рис. 2. Слева - решение системы неявным методом Эйлера, справа - точное решение

Тогда максимальная ошибка должна уменьшится в 4 раза: а по факту уменьшилась примерно 3,9 раз

```
Max error = 0.048956582324791054

Process finished with exit code 0
```

Теперь выберем большие h, τ, чтобы убедиться в том, что поверхности совпадут:
При $h = 500$ и $\tau = 500$

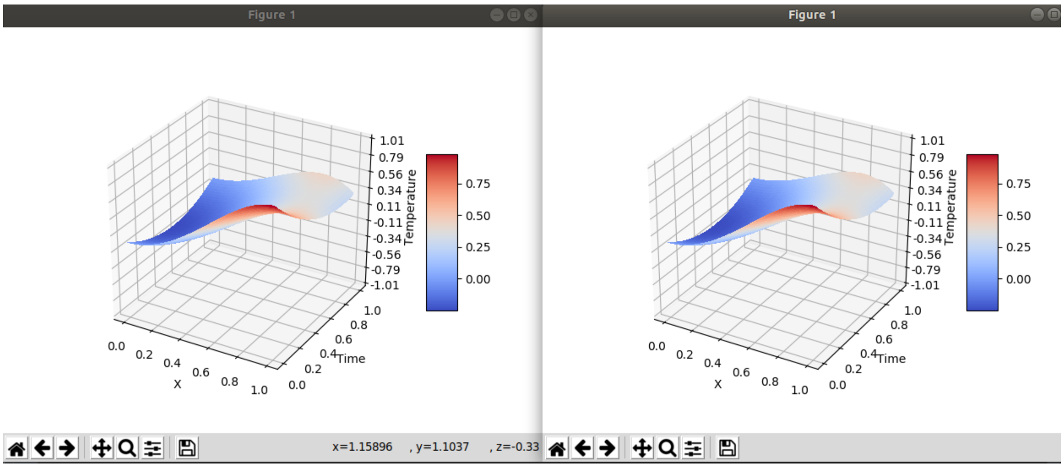


Рис. 3. Слева - решение системы неявным методом Эйлера, справа - точное решение

Как видно на рисунке, графики практически идентичны и почти непрерывны для человеческого взгляда.

8. Выводы.

Таким образом мы установили, что неявный метод Эйлера является устойчивым, и не зависит от τ, h или a как, например, явный метод Эйлера, но, с другой стороны при малых τ, h погрешность решения достаточно велика и, следовательно, решение не достаточно точное.

Также убедились на практике, что данный метод при достаточно больших τ, h наше дискретное решение практически не отличимо от непрерывного, что значительно упрощает решения многих видов уравнений.

Мы увидели, что теоретическая погрешность, которую мы посчитали до прогонки решения, практически совпадает с действительной погрешностью, а значит мы можем выбрать нужную нам точность решения заранее, что не мало важно для методов вычислений.