МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Нижегородский государственный университет им. Н.И. Лобачевского»
Национальный исследовательский университет

ЛАБОРАТОРНАЯ РАБОТА

«ЧИСЛЕННОЕ РЕШЕНИЕ НАЧАЛЬНО-КРАЕВОЙ ЗАДАЧИ ДЛЯ ИНТЕГРО-ДИФФЕРЕНЦИАЛЬНОГО УРАВНЕНИЯ В ЧАСТНЫХ ПРОИЗВОДНЫХ»

Выполнил:
Антипин Александр Сергеевич 381706-2
___________ Подпись

Проверил:
Ассистент кафедры ДУМЧА:
Морозов Кирилл Евгеньевич

___________ Подпись

Нижний Новгород
2020

Оглавление

Введение

Для изучения начально-краевой задачи рассмотрим математическую модель диффузии тепла в твердых телах, для простоты, уравнение нагревания тонкого однородного стержня длиной l под воздействием внешних сил (например, пропускание электрического тока или помещение его в электромагнитное поле). Это уравнение имеет фундаментальное значение во многих областях науки. Например, в статистике уравнение теплопроводности связано с изучением броуновского движения через уравнение Фоккера-Планка. Само же уравнение диффузии является более общей версией уравнения теплопроводности и относится главным образом к изучению процессов химической диффузии.

Будем предполагать, что имеется функция y, которая описывает температуру части стержня в (x, t). Эта функция будет меняться со временем, так как тепло рассеивается в пространстве. Одним из интересных свойств уравнения теплопроводности является принцип максимума, который гласит, что максимальное значение y не может быть больше, чем начальное и граничное значения. То есть, по сути, утверждать, что температура происходит либо от источника, либо от предыдущих состояний тела во времени. Это свойство параболических уравнений в частных производных, и его нетрудно доказать математически.

Постановка задачи

Рассмотрим в области $[0,\ l]\times[0,\ T]$ неоднородное уравнение теплопроводности в однородной среде, где $y(x,\ t)$ дважды непрерывно дифференцируема по x и один раз по t.

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

Оно удовлетворяет однородным граничным условиям второго рода.

$$y'_x(0,t) = y'_x(l,t) = 0$$

А также начальному условию $y(x,0) = \varphi(x)$. Где a – константа, функция $\varphi(x) > 0$ задает начальное распределение температуры и удовлетворяет условию $\int_o^l \varphi(x)dx = l$. Непрерывная функция $u(x,\ t)$ – управление с обратной связью, которое представляется в одном из вариантов:

$$u(x,t) = b(x)y(x,t)$$

$$u(x,t) = b(x)y(x,t) - y(x,t)\int_0^l b(x)y(x,t)dt$$

где $b(x)$ – управляющая функция, непрерывная на отрезке $[0,\ l]$.

Требуется построить графики функции $\varphi(x)$ и $y(x,\ T)$.

План действий

Возьмем в место функции $\varphi(x)$ следующую функцию:

$$\varphi(x) = \frac{1}{l} + \varphi_1 \cos\left(\frac{\pi x}{l}\right) + \varphi_2 \cos\left(\frac{2\pi x}{l}\right)$$

а вместо функции $b(x)$:

$$b(x) = b_0 + b_1 \cos\left(\frac{\pi x}{l}\right) + b_2 \cos\left(\frac{2\pi x}{l}\right)$$

Для решения задачи нам необходимо составить неявную разностную схему. Из предыдущего пункта следует, что ее первый слой можно заполнить значениями, которые получаются из функции $\varphi(x)$.

Для заполнения последующих слоев нам необходимо посчитать интеграл в точке. Это можно сделать по формуле Симпсона:

$$I_j = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + \ldots + 2y_{n-2} + 4y_{n-1} + y_n)$$

После этого составляем неявную разностную схему по формуле:

$$\frac{y_i^{j+1} - y_i^j}{\tau} = \frac{y_{i-1}^{j+1} - 2y_i^{j+1} + y_{i+1}^{j+1}}{h^2} + x_i, i = 1,2, \ldots, N-1, j = 1,2, \ldots, M-1$$

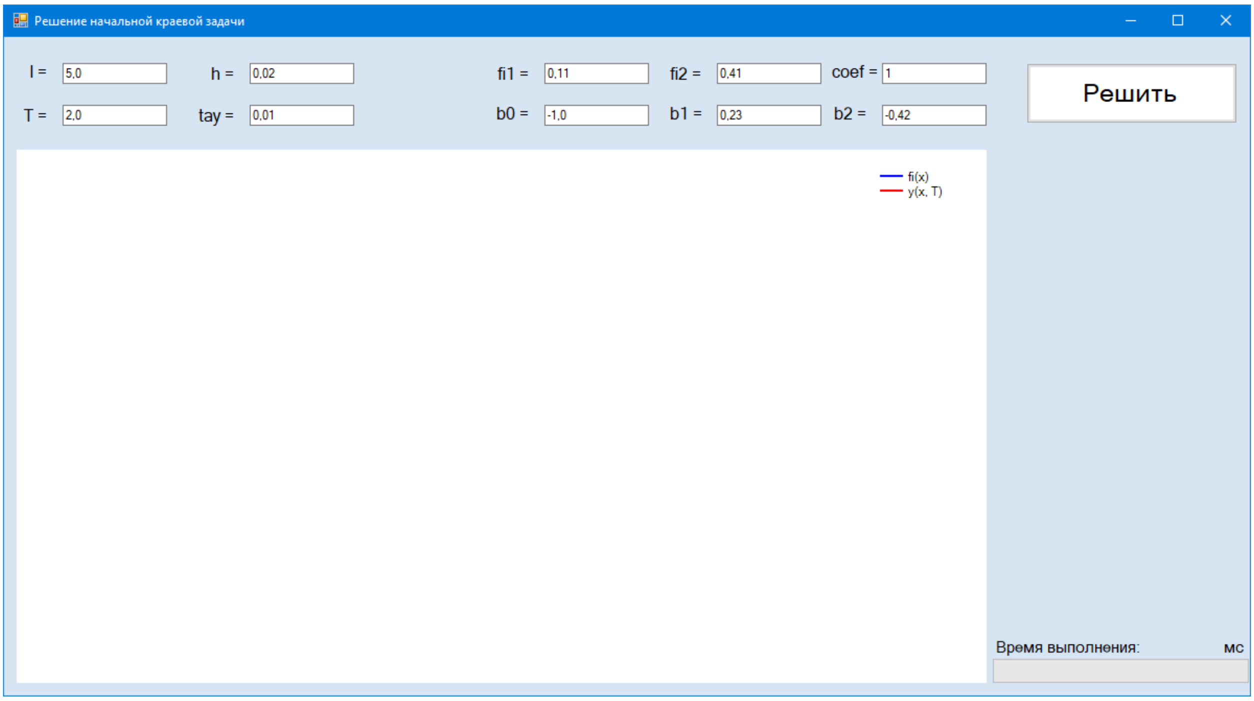
Первое и конечное значения представим 0. В итоге у нас получится трехдиагональная матрица, которую можно решить методом прогонки.

Выбор языка

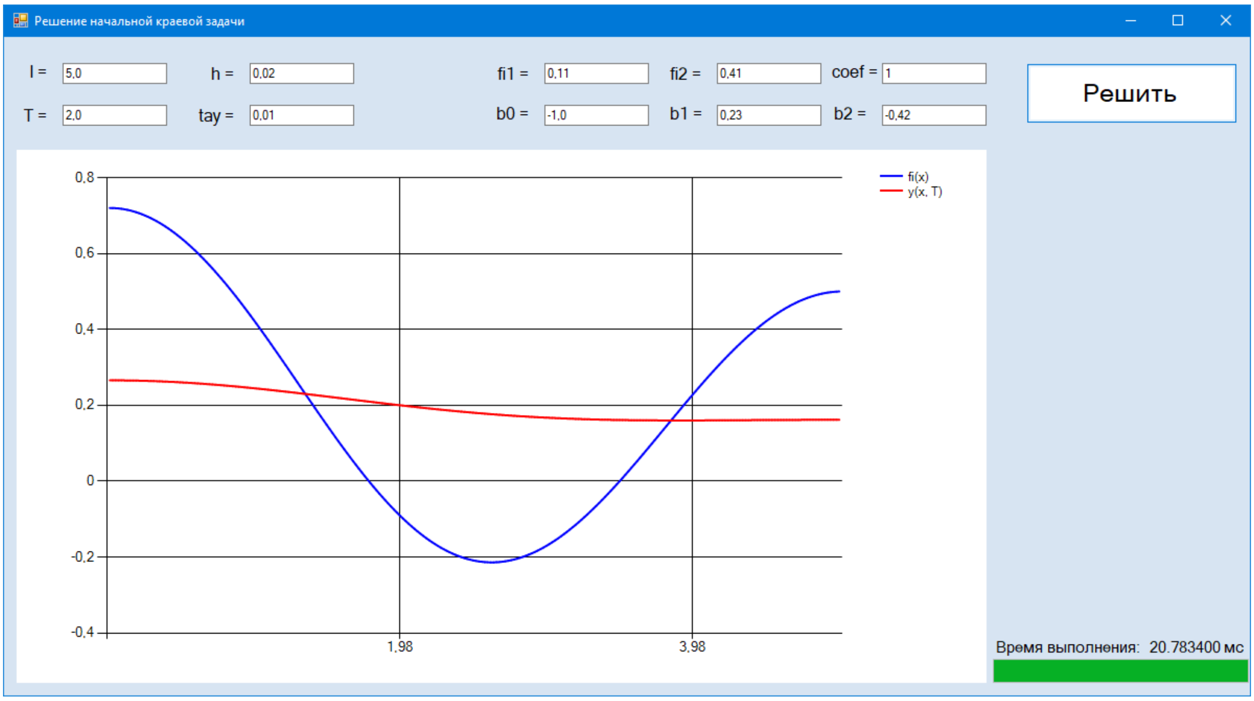
Для создания программы использовался язык высокого уровня C++ с фреймворком Windows Forms для обеспечения визуального представления.

Программная реализация

При запуске программы пользователя встречает форма следующего вида:

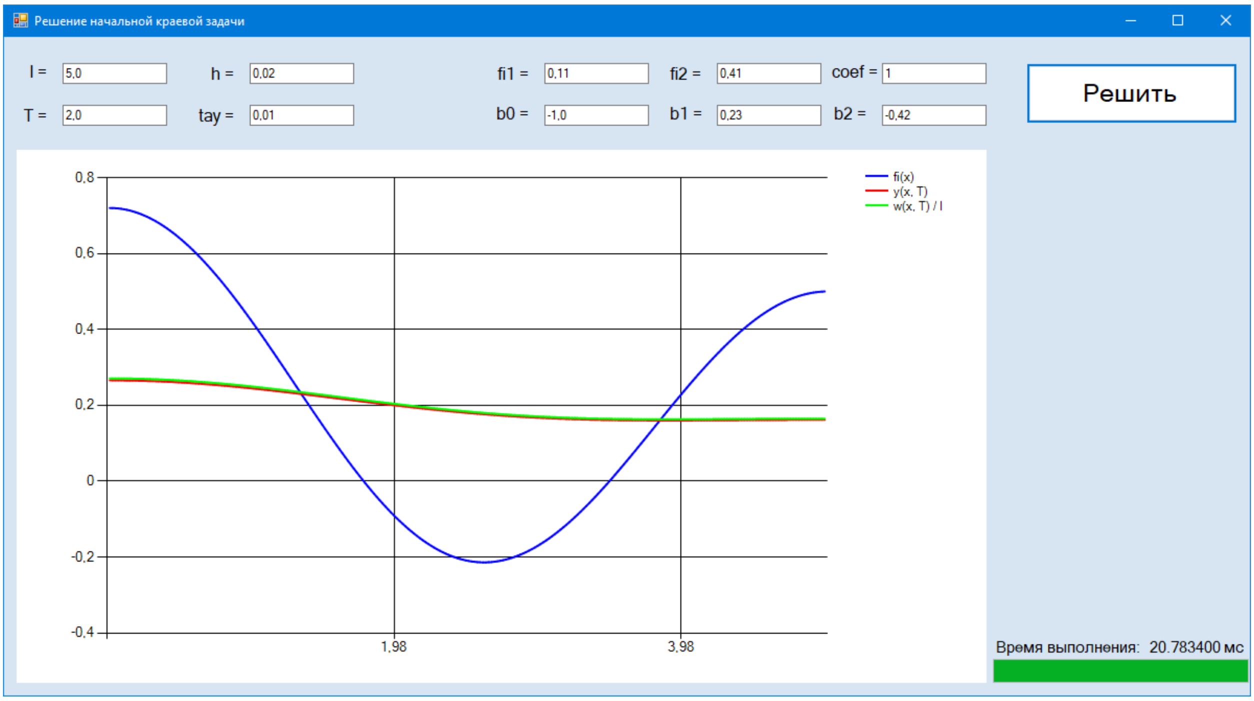


на которой в верхние текстовые полях можно вписать коэффициенты краевой задачи (по умолчанию они заполнены как в примере). Справа находится кнопка «Решить», по нажатию на которую происходит построение графиков функции.



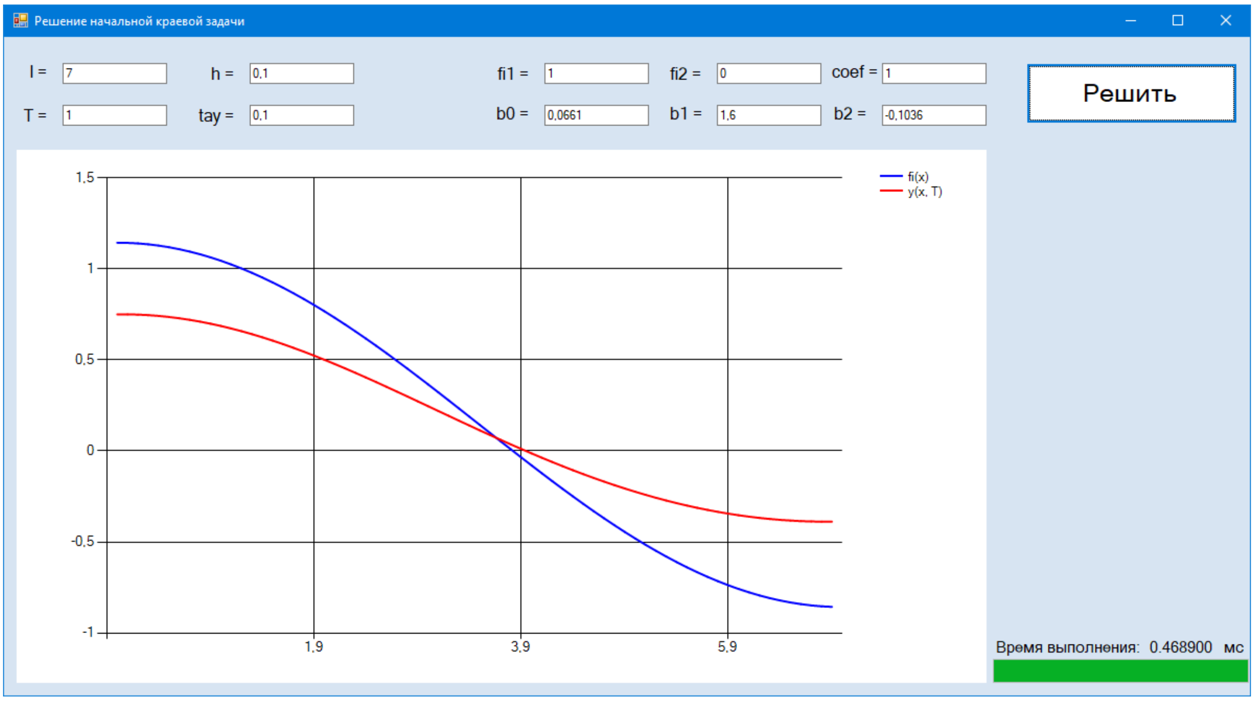
Справа внизу находится строка прогресса, а над ней выводится время подсчета функции.

По нажатию на горячую клавишу «R» зеленым цветом выводится дополнительный график функции $\frac{w(x,t)}{\int_0^l w(x,t)dx}$ – решение части А.

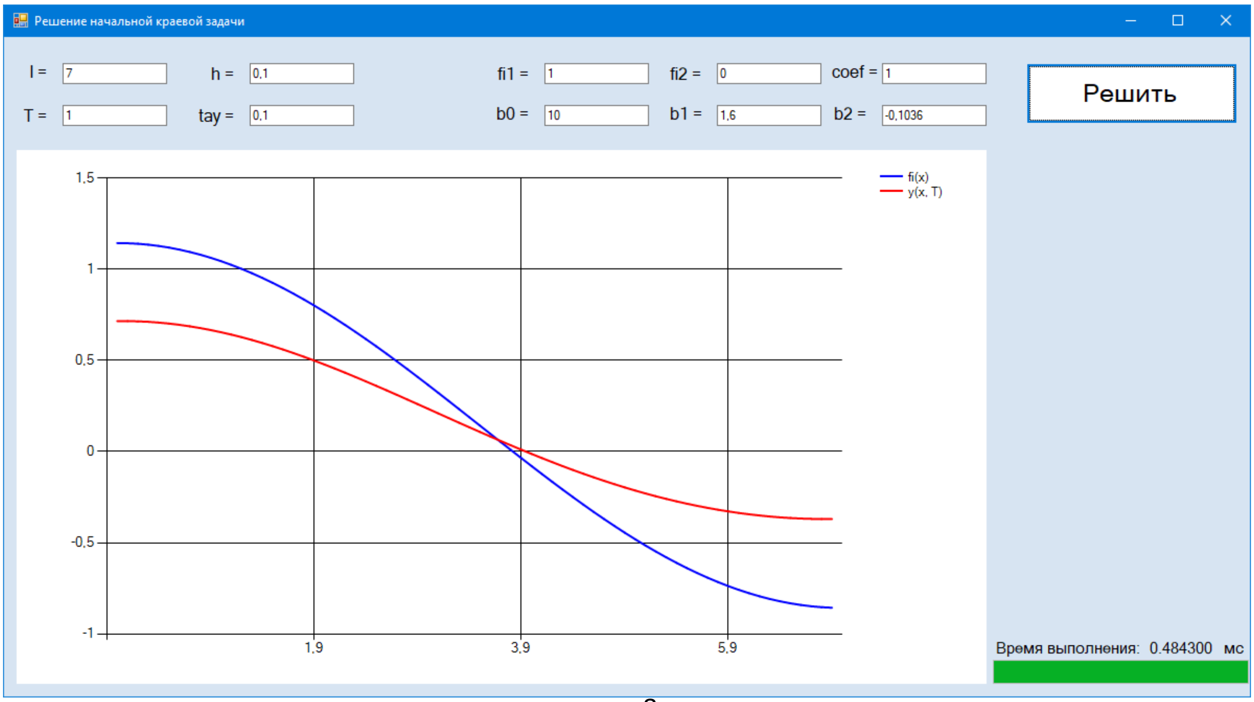


Проверка корректности

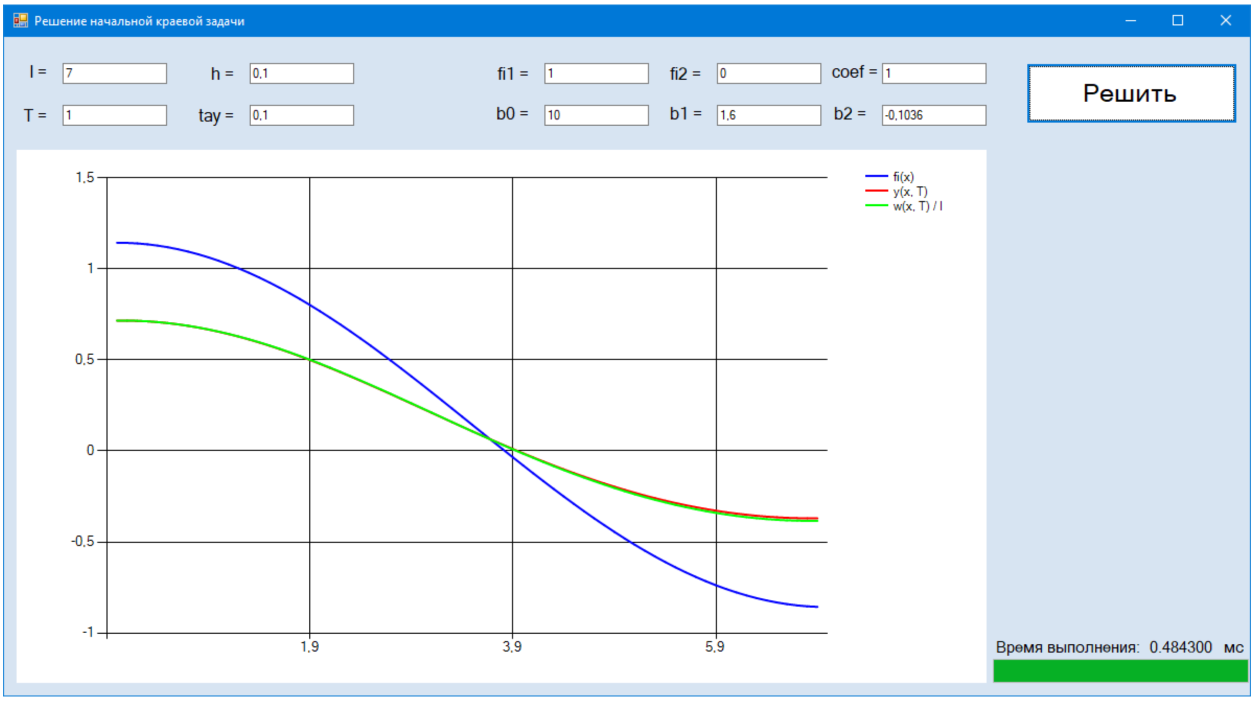
Еще раз запустим программу и убедимся в правильности ее работы:



1. На своих концах графики функций имеют горизонтальные касательные;
2. Площадь фигуры, которая образуется до пересечения красного и синего графиков, равна площади фигуры, которая образуется после их пересечения;



3. Если изменить параметр *b0*, то графики функций останутся на прежних местах;



4. Зеленый график практически полностью совпадает с красным.

Вывод

В процессе выполнения лабораторной работы была решена начально-краевая задача для интегро-дифференциального уравнения нагревания стержня. Была написана программа на языке высокого уровня C++ с дружественным интерфейсом, которая выводит графическую информацию на экран. Работы программы была успешно протестирована.

Список использованной литературы

- А.А.Самарский. Введение в численные методы М.: Наука, 1982.
- http://math.phys.msu.ru/data/374/tema_2.pdf
- https://ru.wikipedia.org/wiki/Уравнение_теплопроводности
- Учебно-методическое пособие - Лабораторная работа «Численное решение начально-краевой задачи для интегро-дифференциального уравнения в частных производных».

Листинг

```
// структура с константами
constants Constants;

// функция fi
double fi(const double x, double fi2) {
    return 1.0 / Constants.l + Constants.fi1 * cos(M_PI * x / Con-
stants.l)
        + fi2 * cos(2.0 * M_PI * x / Constants.l);
}

// функция b
double b(const double x, double b0, double b2) {
    return b0 + Constants.b1 * cos(M_PI * x / Constants.l)
        + b2 * cos(2.0 * M_PI * x / Constants.l);
}

// интегрирование методом Симпсона для части A
double IntegralA(std::vector<std::vector<double>>& grid, size_t j) {
    double integral = grid[j][0];

    for (int i = 1; i < Constants.stepL - 1; i += 2) {
        integral += 4.0 * grid[j][i];
        integral += 2.0 * grid[j][i + 1];
    }

    integral += grid[j][Constants.stepL - 1];
    integral = integral * Constants.h / 3.0;
    return integral;
}

// интегрирование методом Симпсона для части B
double IntegralB(std::vector<std::vector<double>>& grid,
std::vector<double>& b_array, size_t j) {
    double integral = grid[j][0] * b_array[0];

    for (int i = 1; i < Constants.stepL - 1; i += 2) {
        integral += 4.0 * grid[j][i] * b_array[i];
        integral += 2.0 * grid[j][i + 1] * b_array[i + 1];
    }

    integral += grid[j][Constants.stepL - 1] * b_array[Constants.stepL
- 1];
    integral = integral * Constants.h / 3.0;
    return integral;
}

// Метод прогонки
void TridiagonalMatrixAlgorithm(std::vector<double>& a,
std::vector<double>& b, std::vector<double>& c,
    std::vector<double>& f, std::vector<double>& x) {
    size_t size = x.size();
    std::vector<double> A(size), B(size);

    A[0] = -c[0] / b[0];
    B[0] = f[0] / b[0];
```

```
    for (size_t i = 1; i < size; ++i) {
        if (i == size - 1) {
            A[i] = -c[2] / (a[2] * A[i - 1] + b[2]);
            B[i] = (f[i] - a[2] * B[i - 1]) / (a[2] * A[i - 1] +
b[2]);
            break;
        }
        A[i] = -c[1] / (a[1] * A[i - 1] + b[1]);
        B[i] = (f[i] - a[1] * B[i - 1]) / (a[1] * A[i - 1] + b[1]);
    }

    x[size - 1] = B[size - 1];

    for (int i = size - 2; i >= 0; i--)
        x[i] = A[i] * x[i + 1] + B[i];
}

void Thermal(std::vector<std::vector<double>>& gridB,
std::vector<std::vector<double>>& gridA, std::vector<double>& Ares,
    double _l, double _T, double _h, double _tay, double _b0, double
_b1, double _b2, double _fi1, double _fi2, int coef,
    System::Windows::Forms::ProgressBar^ pb) {
    // инициализация констант
    Constants.l = _l; Constants.T = _T;
    Constants.h = _h; Constants.tay = _tay;
    Constants.b0 = _b0; Constants.b1 = _b1; Constants.b2 = _b2;
    Constants.fi1 = _fi1; Constants.fi2 = _fi2;
    Constants.stepL = _l / _h; Constants.stepT = _T / _tay;
    pb->Value = 0;
    pb->Maximum = Constants.stepL;
    gridB.resize(Constants.stepT);
    gridA.resize(Constants.stepT);
    gridB[0].resize(Constants.stepL);
    gridA[0].resize(Constants.stepL);

    // правая часть основного уравнения из A и B
    std::vector<double> fB(Constants.stepL), fA(Constants.stepL);
    std::vector<double> b_array(Constants.stepL);
    std::vector<double> x(Constants.stepL);

    // вычисление значений функций, зависящих от x
    for (size_t i = 0; i < Constants.stepL; ++i) {
        gridB[0][i] = fi(i * Constants.h, Constants.fi2);
        gridA[0][i] = fi(i * Constants.h, Constants.fi2);
        b_array[i] = b(i * Constants.h, Constants.b0, Constants.b2);
    }

    std::vector<double> a(3), b(3), c(3);

    // заполнение векторов коэффициентов метода прогонки
    a[0] = 0.0; b[0] = 1.0; c[0] = -1.0;

    a[1] = Constants.tay * coef * coef / Constants.h / Constants.h;
    b[1] = -2.0 * Constants.tay * coef * coef / Constants.h / Con-
stants.h - 1.0;
    c[1] = Constants.tay * coef * coef / Constants.h / Constants.h;
```

```
    a[2] = -1.0; b[2] = 1.0; c[2] = 0.0;

    // нахождение значений сетки
    for (size_t j = 1; j < Constants.stepT; ++j) {
        gridB[j].resize(Constants.stepL);
        gridA[j].resize(Constants.stepL);
        double integral = IntegralB(gridB, b_array, j - 1);
        for (size_t i = 1; i < Constants.stepL - 1; i++) {
            fB[i] = -gridB[j - 1][i] * (Constants.tay * Constants.tay
* b_array[i]
                - integral * Constants.tay * Constants.tay + 1.0);
            fA[i] = -gridA[j - 1][i] * (Constants.tay * Constants.tay
* b_array[i] + 1.0);
        }
        fB[0] = fB[Constants.stepL - 1] = 0;
        fA[0] = fA[Constants.stepL - 1] = 0;

        // применение метод прогонки для части B
        TridiagonalMatrixAlgorithm(a, b, c, fB, x);
        for (size_t i = 0; i < Constants.stepL; i++)
            gridB[j][i] = x[i];

        // применение метод прогонки для части A
        TridiagonalMatrixAlgorithm(a, b, c, fA, x);
        for (size_t i = 0; i < Constants.stepL; i++)
            gridA[j][i] = x[i];
        pb->PerformStep();
    }

    Ares.resize(Constants.stepL);
    double Aintegral = IntegralA(gridA, Constants.stepT - 1);
    for (size_t i = 0; i < Constants.stepL; i++)
        Ares[i] = gridA[Constants.stepT - 1][i] / Aintegral;
    pb->Value = Constants.stepL;
}


private: System::Void button1_Click(System::Object^ sender, Sys-
tem::EventArgs^ e) {
     Parse();
     // очистка предыдущих графиков
     chart1->Series["fi(x)"]->Points->Clear();
     chart1->Series["y(x, T)"]->Points->Clear();
     chart1->Series["w(x, T) / I"]->Points->Clear();
     chart1->Series["w(x, T) / I"]->Enabled = false;

     // заполнение констант
     std::vector<std::vector<double>> gridB, gridA;
     std::vector<double> Ares;
     double l, T, h, tay, b0, b1, b2, fi1, fi2;
     int coef;

     l = Convert::ToDouble(this->textBox1->Text);
     T = Convert::ToDouble(this->textBox2->Text);
     h = Convert::ToDouble(this->textBox3->Text);
     tay = Convert::ToDouble(this->textBox4->Text);
     b0 = Convert::ToDouble(this->textBox5->Text);
     b1 = Convert::ToDouble(this->textBox6->Text);
```

```
	b2 = Convert::ToDouble(this->textBox7->Text);
	fi1 = Convert::ToDouble(this->textBox8->Text);
	fi2 = Convert::ToDouble(this->textBox9->Text);
	coef = Convert::ToInt32(this->textBox10->Text);

	//выполнение основной функции и замер времени
	std::chrono::time_point<std::chrono::system_clock> start, end;

	start = std::chrono::system_clock::now();
	Thermal(gridB, gridA, Ares, l, T, h, tay, b0, b1, b2, fi1, fi2,
coef, progressBar1);
	end = std::chrono::system_clock::now();
	std::chrono::duration<double, std::milli> time(end - start);
	this->label11->Text = gcnew
String(std::to_string(time.count()).c_str());
	// построение графиков
	for (size_t i = 0; i < gridB[static_cast<size_t>(T / tay) -
	1].size(); ++i) {
		chart1->Series["fi(x)"]->Points->AddXY(i * h, gridB[0][i]);
		chart1->Series["y(x, T)"]->Points->AddXY(i * h,
gridB[static_cast<size_t>(T / tay) - 1][i]);
		chart1->Series["w(x, T) / I"]->Points->AddXY(i * h,
Ares[i]);
	}
}
```