Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЁТ

По лабораторной работе №3

По курсу: «Моделирование»

Тема: «Программно-алгоритмическая реализация моделей на основе ОДУ второго порядка с краевыми условиями II и III рода»

Студент: Керимов А. Ш.

Группа: ИУ7-64Б

Оценка (баллы): ______________

Преподаватель: Градов В. М.

Москва

2020

Цель работы. Получение навыков разработки алгоритмов решения краевой задачи при реализации моделей, построенных на ОДУ второго порядка.

Исходные данные

1. Задана математическая модель.

   Уравнение для функции $T(x)$

   $$\frac{\mathrm{d}}{\mathrm{d}x}\left(k(x)\frac{\mathrm{d}T}{\mathrm{d}x}\right) - \frac{2}{R}\alpha(x)T + \frac{2T_0}{R}\alpha(x) = 0. \tag{1}$$

   Краевые условия

   $$\begin{cases} x = 0, & -k(0)\dfrac{\mathrm{d}T}{\mathrm{d}x} = F0, \\ x = l, & -k(l)\dfrac{\mathrm{d}T}{\mathrm{d}x} = \alpha_N(T(l) - T_0). \end{cases} \tag{2}$$

2. Функции $k(x)$, $\alpha(x)$ заданы своими константами

   $$\begin{aligned} k(x) &= \frac{a}{x-b}, \\ \alpha(x) &= \frac{c}{x-d}. \end{aligned} \tag{3}$$

   Константы a, b следует найти из условий $k(0) = k_0$, $k(l) = k_N$, а константы c, d из условий $\alpha(0) = \alpha_0$, $\alpha(l) = \alpha(N)$.

3. Разностная схема с разностным краевым условием при $x = 0$. Получено в Лекции №7 (7.14), (7.15), и может быть использовано в данной работе. Самостоятельно надо получить интегро-интерполяционным методом разностный аналог краевого условия при $x = l$, точно так же, как это было сделано применительно к краевому условию при $x = 0$ в Лекции №7 (7.15). Для этого надо проинтегрировать на отрезке $[x_{N-\frac{1}{2}}, x_{N+\frac{1}{2}}]$ выписанное выше уравнение (1) с учётом (7.9) из Лекции №7 и учесть, что поток $F_N = \alpha_N(y_N - T_0)$, а $F_{N-\frac{1}{2}} = \chi_{N-\frac{1}{2}}\dfrac{y_{N-1} - y_N}{h}$.

4. Значения параметров для отладки (все размерности согласованы)

   $$\begin{aligned} k_0 &= 0{,}4 && \text{Вт/см К}, \\ k_N &= 0{,}1 && \text{Вт/см К}, \\ \alpha_0 &= 0{,}05 && \text{Вт/см}^2\text{ К}, \\ \alpha_N &= 0{,}01 && \text{Вт/см}^2\text{ К}, \\ l &= 10 && \text{см}, \\ T_0 &= 300 && \text{К}, \\ R &= 0{,}5 && \text{см}, \\ F_0 &= 50 && \text{Вт/см}^2. \end{aligned}$$

Физическое содержание задачи

Сформулированная математическая модель описывает температурное поле $T(x)$ вдоль цилиндрического стержня радиуса R и длиной l, причём $R << l$ и температуру можно принять постоянной по радиусу цилиндра. Ось x направлена вдоль оси цилиндра и начало координат совпадает с левым торчцем стержня. Слева при $x = 0$ цилиндр нагружается тепловым потоком F_0. Стержень обдувается воздухом, температура которого равна T_0. В результате происходит съем тепла с цилиндрической поверхности и поверхности правого торца при $x = l$. Функции $k(x)$, $\alpha(x)$ являются, соответственно, коэффициентами теплопроводности материала стержня и теплоотдачи при обдуве.

Результаты работы

1. **Разностный аналог краевого условия при $x = l$ и его краткий вывод интегро-интерполяционным методом**

   Обозначим $F = -k(x)\dfrac{\mathrm{d}T}{\mathrm{d}x}$, $p(x) = \dfrac{2}{R}\alpha(x)$, $f(x) = \dfrac{2T_0}{R}\alpha(x)$. Тогда (1) примет вид:

$$-\frac{\mathrm{d}F}{\mathrm{d}x} - p(x)T + f(x) = 0. \tag{4}$$

   Проинтегрируем на отрезке $[x_{N-\frac{1}{2}}, x_{N+\frac{1}{2}}]$:

$$-\int_{x_{N-\frac{1}{2}}}^{x_N} \frac{\mathrm{d}F}{\mathrm{d}x}\,\mathrm{d}x - \int_{x_{N-\frac{1}{2}}}^{x_N} p(x)T\,\mathrm{d}x + \int_{x_{N-\frac{1}{2}}}^{x_N} f(x)\,\mathrm{d}x = 0. \tag{5}$$

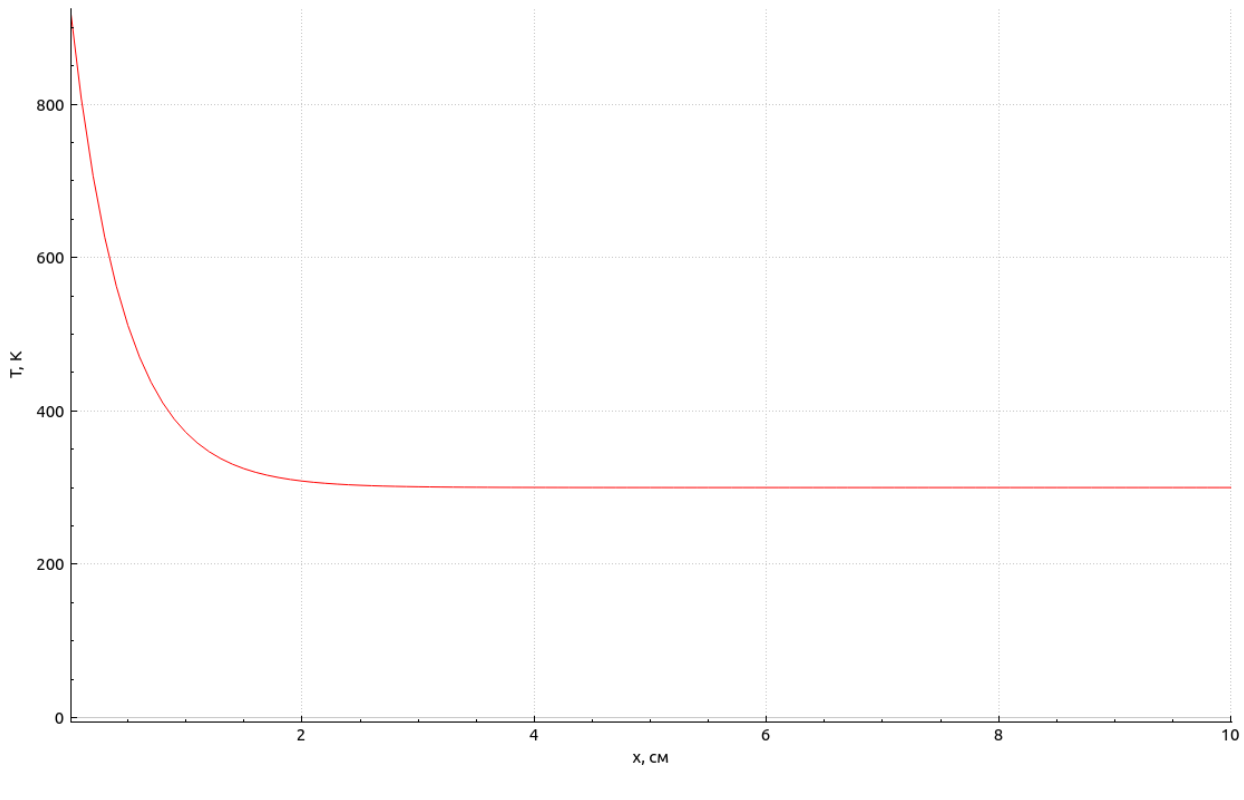
   Второй и третий интеграл вычислим методом трапеций:

$$F_{N-\frac{1}{2}} - F_N - \frac{p_{N-\frac{1}{2}}y_{N-\frac{1}{2}} + p_N y_N}{2}\cdot\frac{h}{2} + \frac{f_{N-\frac{1}{2}} + f_N}{2}\cdot\frac{h}{2} = 0. \tag{6}$$

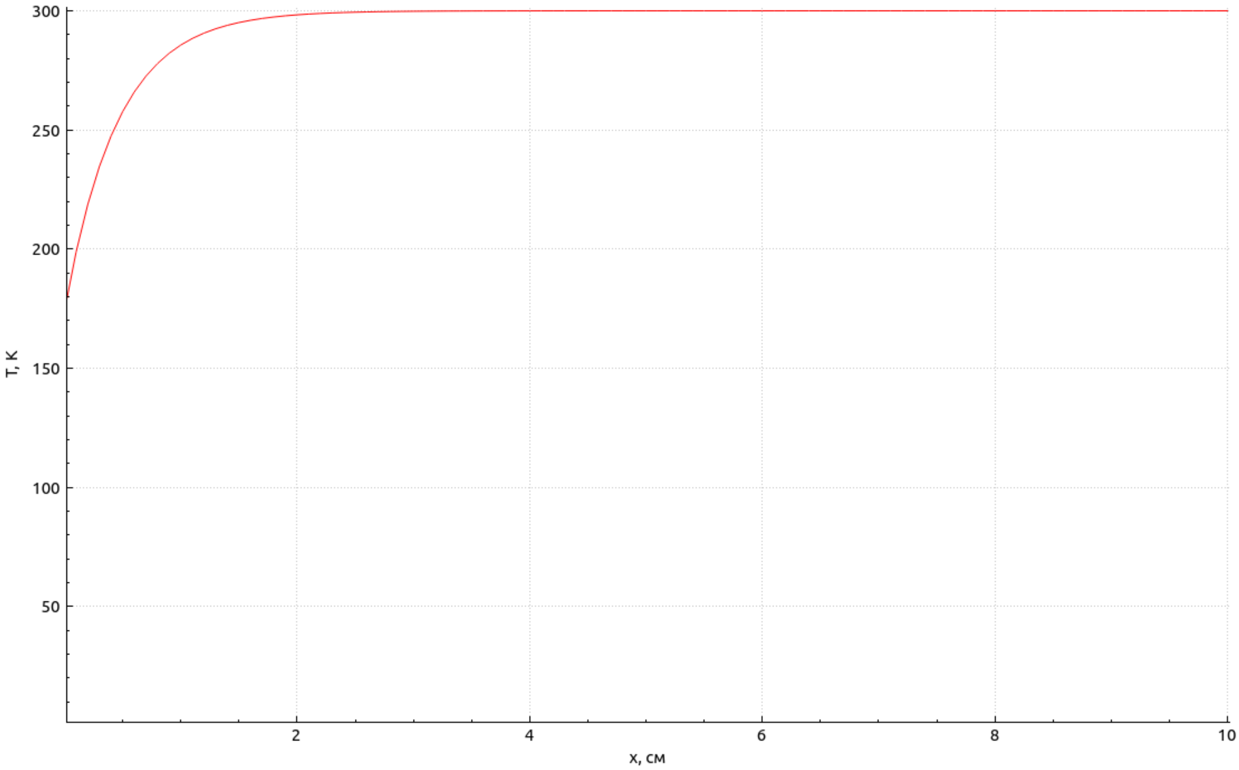
   Подставляя (7.12) Лекции №7 и краевое условие потока $F_N = \alpha_N(y_N - T_0)$ в (6):

$$y_N\left(\frac{-\chi_{N-\frac{1}{2}}}{h} - \alpha_N - \frac{p_{N-\frac{1}{2}}h}{8} - \frac{p_N h}{4}\right) + y_{N-1}\left(\frac{\chi_{N-\frac{1}{2}}}{h} - \frac{p_{N-\frac{1}{2}}h}{8}\right) = -\alpha_N T_0 - \frac{h}{4}\left(f_{N-\frac{1}{2}} + f_N\right) \tag{7}$$

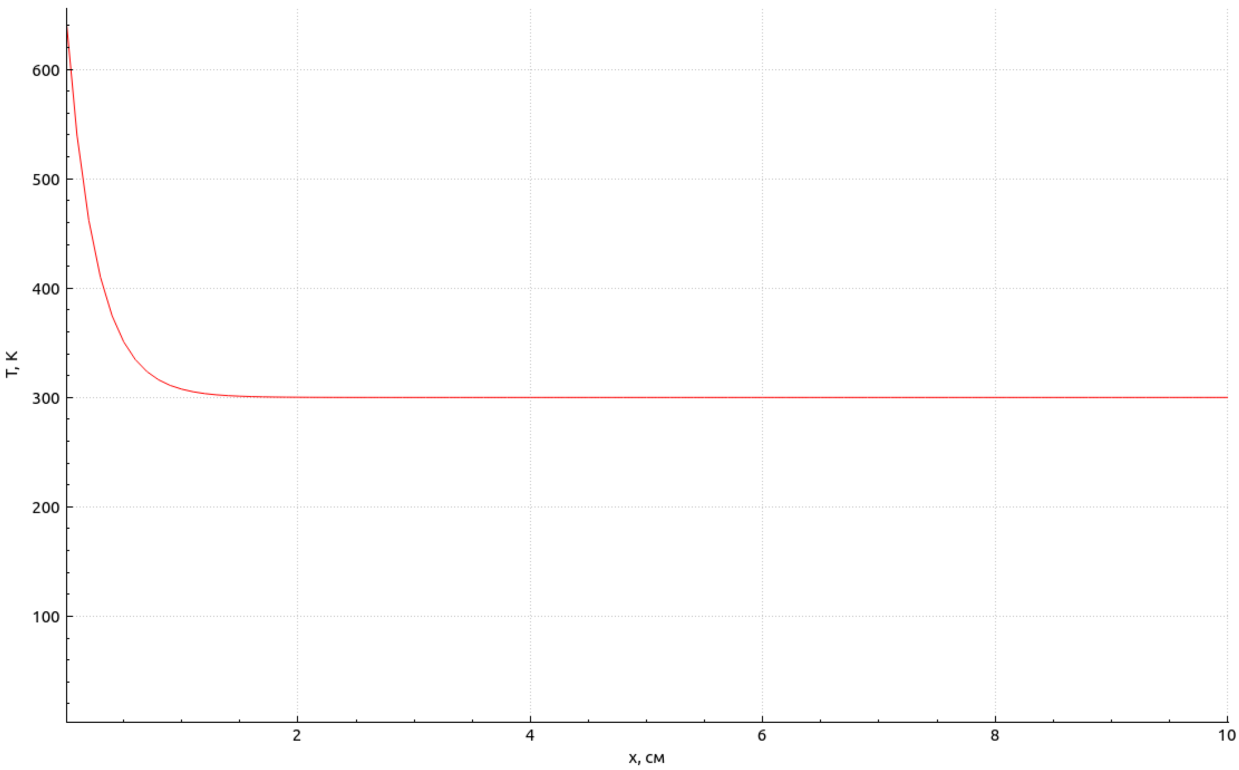
2. **График зависимости $T(x)$ при заданных выше параметрах**



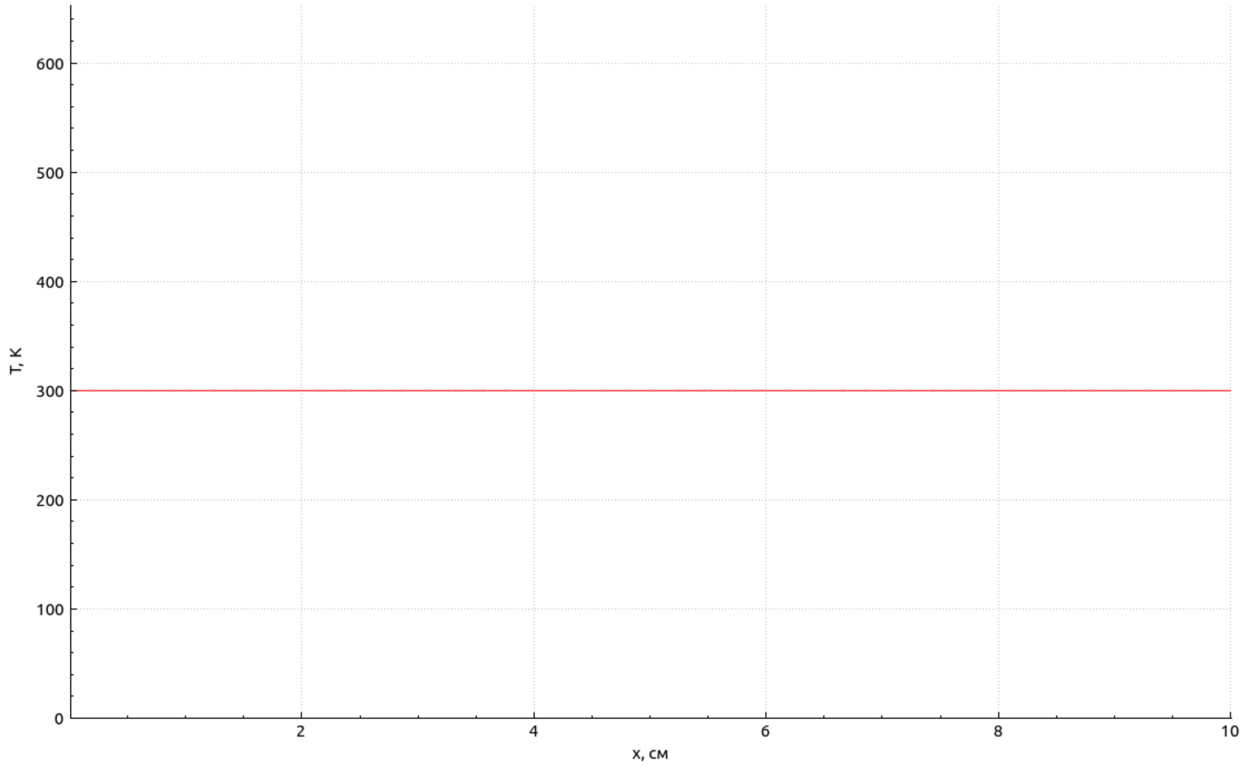
3. **График зависимости $T(x)$ при $F_0 = -10$ Вт/см^2**



4. **График зависимости $T(x)$ при увеличенных в 3 раза значениях $\alpha(x)$**



5. **График зависимости $T(x)$ при $F_0 = 0$**



Вопросы

1. **Какие способы тестирования программы можно предложить?**

   - Задать отрицательный тепловой поток. Стержень будет охлаждаться с левого торца, а значит $T(x)$ от 0 до l будет увеличиваться.
   - При большей теплоотдачи стержня его температура должна снизиться.
   - При нулевом тепловом потоке температура стержня должна быть неизменной и равняться температуре окружающей среды.

2. **Получите простейший разностный аналог нелинейного краевого условия при**

$$x = l, \quad -k(x)\frac{\mathrm{d}T}{\mathrm{d}x} = \alpha_N \left(T(l) - T_0\right) + \varphi(T). \tag{8}$$

   Аппроксимируем производную односторонней разностью

$$-k_l \frac{T_l - T_{l-1}}{h} = \alpha_N(T_l - T_0) + \varphi(T_l). \tag{9}$$

   Отсюда

$$-(k_l + \alpha_N h)T_l + k_l T_{l-1} = \varphi(T_l)h - \alpha_N h T_0 \tag{10}$$

3. **Опишите алгоритм применения метода прогонки, если при $x = 0$ краевое условие линейное (как в настоящей работе), а при $x = l$, как в п. 2.**

$$\begin{cases} x = 0, \quad -k(0)\dfrac{\mathrm{d}T}{\mathrm{d}x} = F0, \\ x = l, \quad -k(l)\dfrac{\mathrm{d}T}{\mathrm{d}x} = \alpha_N(T(l) - T_0) + \varphi(T). \end{cases} \tag{11}$$

Будем использовать левую прогонку, основная прогоночная формула:

$$y_n = \xi_{n-1} y_{n-1} + \eta_{n-1} \tag{12}$$

Принимая простейшую (первого порядка точности) аппроксимацию краевого условия при $x = 0$, получим его разностный аналог:

$$-k_0 \frac{T_1 - T_0}{h} = F_0 \quad \Rightarrow \quad -k_0 T_1 + k_0 T_0 = F_0 h. \tag{13}$$

Сравнивая с (12) при $n = 1$:

$$\begin{cases} \xi_0 = 1, \\ \eta_0 = -\dfrac{F_0 h}{k_0}. \end{cases} \tag{14}$$

Аналогично, разностная аппроксимация правого краевого условия имеет вид:

$$-(k_l + \alpha_N h) T_l + k_l T_{l-1} = \varphi(T_l) h - \alpha_N h T_0, \tag{15}$$

$$-(k_l + \alpha_N h) T_l + k_l \frac{T_l - \eta_{l-1}}{\xi_{l-1}} = \varphi(T_l) h - \alpha_N h T_0. \tag{16}$$

Отсюда получаем уравнение для определения T_0:

$$\varphi(T_l) h - \left(k_l + \alpha_N h - \frac{k_l}{\xi_{l-1}} \right) T_l = \frac{k_l \eta_{l-1}}{\xi_{l-1}} - \alpha_N h T_0. \tag{17}$$

4. **Опишите алгоритм определения единственного значения сеточной функции y_p в одной заданной точке p. Использовать встречную прогонку, т. е. комбинацию правой и левой прогонок** (Лекция №8). **Краевые условия линейные.**

   Пусть $i = p$, где $0 < p < N$. Тогда в области $0 \leqslant i \leqslant p + 1$ прогоночные коэффициенты α_i, β_i (правая прогонка):

$$\alpha_{i+1} = \frac{C_i}{B_i - \alpha_i A_i}, \quad \beta_{i+1} = \frac{A_i \beta_i + D_i}{C_i - \alpha_i A_i} \tag{18}$$

   А в области $p \leqslant i \leqslant N$ прогоночные коэффициенты ξ_i, η_i (левая прогонка):

$$\xi_i = \frac{C_i}{B_i - \xi_{i+1} A_i}, \quad \eta_i = \frac{A_i \eta_{i+1} + D_i}{B_i - \xi_{i+1} A_i}. \tag{19}$$

   Тогда при $i = p$:

$$y_p = \alpha_{p+1} y_{p+1} + \beta_{p+1}, \;\; y_{p+1} = \xi_{p+1} y_p + \eta_{p+1}, \tag{20}$$

   и тогда:

$$y_p = \frac{\beta_{p+1} + \alpha_{p+1} \eta_{p+1}}{1 - \alpha_{p+1} \xi_{p+1}}. \tag{21}$$

Листинг

Листинг 1: solve.hpp

```
#ifndef SOLVE_HPP_
#define SOLVE_HPP_

#include <QVector>

class Parameters {
public:
	double k0;
	double kN;
	double alpha0;
	double alphaN;
	double F0;
};

using Container = QVector<double>;

class Dependency {
public:
	Container x;
	Container T;
};

Dependency solve(const Parameters& parameters);

#endif  // SOLVE_HPP_
```

Листинг 2: solve.cpp

```
#include "solve.hpp"

class Solver {
public:
	explicit Solver(const Parameters& parameters) :
			k0_     (parameters.k0),
			kN_     (parameters.kN),
			alpha0_(parameters.alpha0),
			alphaN_(parameters.alphaN),
			F0_     (parameters.F0),
			b_      (l_ * kN_ / (kN_ - k0_)),
			a_      (-k0_ / b_),
			d_      (l_ * alphaN_ / (alphaN_ - alpha0_)),
			c_      (-alpha0_ * d_) {}

	Dependency solve() const {
		Container a(1), b(1), c(1), d(1), xs;
		for (double x = 0.0; x <= l_; x += h_) {
			a.push_back(x_nmh(x) / h_);
			c.push_back(x_nph(x) / h_);
			b.push_back(a.back() + c.back() + p(x) * h_);
			d.push_back(f(x) * h_);
			xs.push_back(x);
		}

		// compiler will optimize calculations
		const double k0 = x_nph(0) + h_ * h_ * (p(0) + p(h_)) / 16 + h_ * h_ * p(0) / 4;
		const double m0 = h_ * h_ * (p(0) + p(h_)) / 16 - x_nph(0);
		const double p0 = h_ * F0_ + h_ * h_ * ((f(0) + f(h_)) / 2 + f(0)) / 4;

		const double kN = -x_nmh(l_) / h_ - alphaN_ - h_ * (p(l_) + p(l_ - h_)) / 16 - h_ *
			p(l_) / 4;
		const double mN = x_nmh(l_) / h_ - h_ * (p(l_) + p(l_ - h_)) / 16;
		const double pN = -(alphaN_ * T0_ + h_ * ((f(l_) + f(l_ - h_)) / 2 + f(l_)) / 4);

		// forward sweep
		Container xi(a.size() + 1);
		Container eta(a.size() + 1);
		xi[1]  = -m0 / k0;
		eta[1] =  p0 / k0;
		for (int i = 1; i < a.size(); ++i) {
			const double det = b[i] - a[i] * xi[i];
			xi[i + 1]  = c[i] / det;
			eta[i + 1] = (a[i] * eta[i] + d[i]) / det;
```

```
			}

			// backward substitution
			Container ys(a.size());
			ys.back() = (pN - mN * eta.back()) / (kN + mN * xi.back());
			for (int i = ys.size() - 2; i >= 0; --i) {
				ys[i] = xi[i + 1] * ys[i + 1] + eta[i + 1];
			}

			return {xs, ys};
		}

private:
		const double k0_;
		const double kN_;
		const double alpha0_;
		const double alphaN_;
		const double F0_;

		static constexpr double l_  = 10;
		static constexpr double T0_ = 300;
		static constexpr double R_  = 0.5;

		const double b_;
		const double a_;
		const double d_;
		const double c_;

		static constexpr double h_ = 0.1;

		double k(double x) const {
			return a_ / (x - b_);
		}

		double alpha(double x) const {
			return c_ / (x - d_);
		}

		double p(double x) const {
			return 2 * alpha(x) / R_;
		}

		double f(double x) const {
			return 2 * alpha(x) * T0_ / R_;
		}

		double x_nph(double xn) const {
			const double k_curr = k(xn);
			const double k_next = k(xn + h_);
			return 2 * k_curr * k_next / (k_curr + k_next);
		}

		double x_nmh(double xn) const {
			const double k_curr = k(xn);
			const double k_prev = k(xn - h_);
			return 2 * k_curr * k_prev / (k_curr + k_prev);
		}
};

Dependency solve(const Parameters& parameters) {
		return Solver(parameters).solve();
}
```