Министерство образования и науки РФ
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

Отчёт по лабораторной работе №1
по дисциплине
«Математические модели
систем с распределёнными параметрами»

Выполнил студент гр. 3530904/00104 Смирнов Е. А.

Руководитель С. П. Воскобойников

Санкт-Петербург
2022

Оглавление

Постановка задачи

Номер Варианта СР-3.

Задание: используя интегро-интерполяционный метод (метод баланса), разработать программу для моделирования стационарного распределения температуры в полом цилиндре, описываемого математической моделью вида

$$-\left[\frac{1}{r}\frac{d}{dr}\left(rk(r)\frac{du}{dr}\right)-q(r)u\right]=f(r),\, r\in[R_L,\ R_R],\, R_L>0,$$
$$0<C_1\le k(r)\le C_2,\, 0\le q(r)$$

с граничными условиями

$$1)\ u_{r=R_L}=\ \vartheta_1$$
$$2)\ -k(r)\frac{du}{dr}\bigg|_{\,r=R_R}=\ \chi_2 u|_{\,r=R_R}-\vartheta_2$$

Построение дискретной модели

1. Дискретизация

Введем число N – число разбиений

$$r_0<\ r_1<\cdots<\ r_N\,, r_i\ \in[R_l,R_R], r_0=R_l\,, r_N=R_R$$

2. Интегро-интерполяционный метод

А) основная сетка

$$h_i=r_i-r_{i-1}\,, i=1,2,\dots N$$

Б) вспомогательная сетка

- Разбиваем *каждый получившийся интервал в первом пункте пополам*

$$r_{i-\frac{1}{2}}=\frac{r_i+\ r_{i-1}}{2}\,, i=1,2,\dots N$$

- Записываем шаг вспомогательной сетки ($\hbar$)

$$\hbar_i=\begin{cases}\dfrac{h_{i+1}}{2}\,, i=0\\ \dfrac{h_i+\ h_{i+1}}{2}, i=1,2\dots N-1\\ \dfrac{h_i}{2}, i=N\end{cases}$$

3. Аппроксимация основного уравнения

1. умножим обе части исходного уравнения на r (интегрируем с весом r)

$$-\left[\frac{d}{dr}\left(rk(r)\frac{du}{dr}\right)-rq(r)u\right]=rf(r)$$

$$-\int_{r_{i-\frac{1}{2}}}^{r_{i+\frac{1}{2}}} \left[\frac{d}{dr}\left(rk(r)\frac{du}{dr}\right) - rq(r)u\right]dr = \int_{r_{i-\frac{1}{2}}}^{r_{i+\frac{1}{2}}} rf(r)dr\,, i = 1,2,\dots,N-1$$

$$-\left[rk(r)\frac{du}{dr}\bigg|_{r=r_{i+1/2}} - rk(r)\frac{du}{dr}\bigg|_{r=r_{i-\frac{1}{2}}} - \int_{r_{i-\frac{1}{2}}}^{r_{i+\frac{1}{2}}} rq(r)u(r)dr\right] = \int_{r_{i-\frac{1}{2}}}^{r_{i+\frac{1}{2}}} rf(r)dr,$$

$$i = 1,2,\dots,N-1$$

2. аппроксимируем (формула центральных разностей)

$$k(r)\frac{du}{dr}\bigg|_{r=r_{i-\frac{1}{2}}} \approx k\left(r_{i-\frac{1}{2}}\right)\frac{u_i - u_{i-1}}{2\frac{h_i}{2}} = k_{i-\frac{1}{2}}\frac{u_i - u_{i-1}}{h_i}$$

3. аппроксимируем (формула средних прямоугольников)

$$\int_{r_{i-\frac{1}{2}}}^{r_{i+\frac{1}{2}}} r\varphi(r)dr \approx \hbar_i\, r_i\, \varphi_i$$

4. Запишем получившуюся аппроксимацию уравнения для $i = 1,2,\dots,N-1$

$$-\left[r_{i+\frac{1}{2}}k_{i+\frac{1}{2}}\frac{v_{i+1} - v_i}{h_{i+1}} - r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}\frac{v_i - v_{i-1}}{h_i} - \hbar_i\, r_i\, q_i v_i\right] = \hbar_i\, r_i\, f_i$$

$$, i = 1,2,\dots,N-1$$

4. Аппроксимация граничных условий

1) условие 1. $u_{r=R_L} = \vartheta_1$

$$\boldsymbol{v_i = \vartheta_1, i = 0}$$

2) условие 2. - $k(r)\frac{du}{dr}\Big|_{r=R_R} = \chi_2 u|_{r=R_r} - \vartheta_2$

Проинтегрируем основное уравнение по вспомогательной сетке:

$$-\int_{r_{i-1/2}}^{r_i} \left[\frac{d}{dr}\left(rk(r)\frac{du}{dr}\right) - rq(r)u\right]dr = \int_{r_{i-1/2}}^{r_i} rf(r)dr\ , i = N$$

$$-\left[rk(r)\frac{du}{dr}\Big|_{r=r_i} - rk(r)\frac{du}{dr}\Big|_{r=r_{i-\frac{1}{2}}} - \int_{r_{i-\frac{1}{2}}}^{r_i} rq(r)u(r)dr\right] = \int_{r_{i-\frac{1}{2}}}^{r_i} rf(r)dr$$

Используя формулы центральных разностей и правых прямоугольников и, подставляя вместо производной - заданное выражение получаем:

$$-\left[-r_i(\chi_2 v_i - \vartheta_2) - r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}\frac{v_i - v_{i-1}}{h_i} - \hbar_i\, r_i\, q_i v_i\right] = \hbar_i\, r_i\, f_i\,, i = N$$

5. Разностная схема

$$v_i = \vartheta_1, \qquad i = 0$$

$$-\left[r_{i+\frac{1}{2}}k_{i+\frac{1}{2}}\frac{v_{i+1}-v_i}{h_{i+1}} - r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}\frac{v_i - v_{i-1}}{h_i} - \hbar_i\, r_i\, q_i v_i\right] = \hbar_i\, r_i\, f_i, \quad i = 1,2,\dots,N-1$$

$$-\left[-r_i(\chi_2 v_i - \vartheta_2) - r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}\frac{v_i - v_{i-1}}{h_i} - \hbar_i\, r_i\, q_i v_i\right] = \hbar_i\, r_i\, f_i, \qquad i = N$$

6. Матричное уравнение

Составим алгебраическую систему уравнений с трех диагональной матрицей.

Для $\boldsymbol{i = 0}$

Введем следующие обозначения: $\boldsymbol{c_i} = 1$, $\boldsymbol{b_i} = 0$, $\boldsymbol{g_i} = \vartheta_1$

Для $\boldsymbol{i = 1, 2, \dots, N-1}$

Приведем подобные члены:

$$\boldsymbol{v_{i-1}}\left(-\frac{r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}}{h_i}\right) + \boldsymbol{v_i}\left(\frac{r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}}{h_i} + \frac{r_{i+\frac{1}{2}}k_{i+\frac{1}{2}}}{h_{i+1}} + \hbar_i\, r_i\, q_i\right) + \boldsymbol{v_{i+1}}\left(-\frac{r_{i+\frac{1}{2}}k_{i+\frac{1}{2}}}{h_{i+1}}\right) = \hbar_i\, r_i\, f_i$$

Введем следующие обозначения:

$$\boldsymbol{a_i} = \left(-\frac{r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}}{h_i}\right),$$

$$\boldsymbol{c_i} = \left(\frac{r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}}{h_i} + \frac{r_{i+\frac{1}{2}}k_{i+\frac{1}{2}}}{h_{i+1}} + \hbar_i\, r_i\, q_i\right),$$

$$\boldsymbol{b_i} = \left(-\frac{r_{i+\frac{1}{2}}k_{i+\frac{1}{2}}}{h_{i+1}}\right),$$

$$\boldsymbol{g_i} = \hbar_i\, r_i\, f_i$$

Для $\boldsymbol{i = N}$

Приведем подобные члены:

$$\boldsymbol{v_{i-1}}\left(-\frac{r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}}{h_i}\right) + \boldsymbol{v_i}\left(\frac{r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}}{h_i} + \hbar_i\, r_i\, q_i + r_i\chi_2\right) = \hbar_i\, r_i\, f_i + r_i\vartheta_2$$

Введем следующие обозначения:

$$\boldsymbol{a_i} = \left(-\frac{r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}}{h_i}\right)$$

$$\boldsymbol{c_i} = \left(\frac{r_{i-\frac{1}{2}}k_{i-\frac{1}{2}}}{h_i} + \hbar_i\, r_i\, q_i + r_i\chi_2\right),$$

$$\boldsymbol{g_i} = \hbar_i\, r_i\, f_i + r_i\vartheta_2$$

Получаем систему уравнений

$$\begin{aligned} \mathrm{c_i}\mathbf{v_i} + \mathbf{b_i}\mathbf{v_{i+1}} &= \mathbf{g_i}, & \mathbf{i} &= \mathbf{0} \\ \mathbf{a_i}\mathbf{v_{i-1}} + \mathrm{c_i}\mathbf{v_i} + \mathbf{b_i}\mathbf{v_{i+1}} &= \mathbf{g_i}, & \mathbf{i} &= \mathbf{1}, \mathbf{2}, \dots, \mathbf{N-1} \\ \mathbf{a_i}\mathbf{v_{i-1}} + \mathbf{c_i}\mathbf{v_i} &= \mathbf{g_i}, & \mathbf{i} &= \mathbf{N} \end{aligned}$$

Запишем матрицу $\boldsymbol{A} = \begin{bmatrix} c_0 & b_0 & & & & \\ a_1 & c_1 & b_1 & & & \\ & a_2 & c_2 & b_2 & & \\ & & . & . & . & \\ & & & & c_{N-1} & b_{N-1} \\ & & & & a_N & c_N \end{bmatrix}$

Размерность матрицы А – (N+1) (N+1).
Теперь разностную схему *запишем в виде* $A\boldsymbol{v} = \boldsymbol{g}$, где

$$\boldsymbol{v} = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ . \\ v_{N-1} \\ v_N \end{bmatrix} \text{ и } \boldsymbol{g} = \begin{bmatrix} g_0 \\ g_1 \\ g_2 \\ . \\ g_{N-1} \\ g_N \end{bmatrix} \quad \text{v, g} \in R^{(N+1)}$$

Метод прогонки

Алгебраическую систему $A\boldsymbol{v} = \boldsymbol{g}$ с трех диагональной матрицей будем решать методом прогонки. Решение запишем в виде $\boldsymbol{v_i} = \boldsymbol{\alpha_{i+1} v_{i+1}} + \boldsymbol{\beta_{i+1}}$, где α и β - прогоночные коэффициенты

Этап 1. Найдем коэффициенты α и β из первого уравнения. Выразим v_0:

$$i = 0: \ v_0 = -\frac{b_0}{c_0} v_{i+1} + \frac{g_0}{c_0}$$

$$\alpha_1 = -\frac{b_0}{c_0} \ и \ \beta_1 = \frac{g_0}{c_0}$$

Этап 2. Для $i = 1,2, \dots, N-1$ сначала выразим $v_{i-1} = \alpha_i v_i + \beta_i$ и подставим в (N-1) уравнение одновременно выразив v_i. Получаем:

$$v_i = -\frac{b_i}{(a_i * \alpha_i + c_i)} v_{i+1} + \frac{g_i - a_i * \beta_i}{(a_i * \alpha_i + c_i)}$$

$$\alpha_{i+1} = -\frac{b_i}{(a_i * \alpha_i + c_i)} \text{ и } \beta_{i+1} = \frac{g_i - a_i * \beta_i}{(a_i * \alpha_i + c_i)}$$

Считаем коэффициенты – прямой ход.

Этап 3. Для $i = N$ можем найти компоненту $v_N = \frac{g_N - a_N * \beta_N}{(a_N * \alpha_N + c_N)}$

Этап 4. Вычисляем остальные компоненты – обратный ход.

$$v_i = \alpha_{i+1} v_{i+1} + \beta_{i+1}$$

Этап 5. Протестируем метод на входной матрице *A*

$$A = \begin{pmatrix} -3 & 4 & 0 & 0 \\ -2 & 5 & 1 & 0 \\ 0 & 5 & 9 & 1 \\ 0 & 0 & 3 & 6 \end{pmatrix} \ и \ g = \begin{pmatrix} -3 \\ -2 \\ 9 \\ -2 \end{pmatrix}$$

В результате вычислительная машина выдала точные результаты.

Ожидаемые результаты	Полученные результаты
3	3.000e+00
2	2,000e+00
0	0,000e+00
-1	-1,000e+00

Анализ порядка аппроксимации

1. Невязка для уравнения

$$\varepsilon = g - Au$$

$$\varepsilon = \hbar_i\, r_i\, f_i + \left[r_{i+\frac{1}{2}} k_{i+\frac{1}{2}} \frac{u_{i+1}-u_i}{h_{i+1}} - r_{i-\frac{1}{2}} k_{i-\frac{1}{2}} \frac{u_i-u_{i-1}}{h_i} - \hbar_i\, r_i\, q_i u_i \right] \text{[0]}$$

Так как, точного решения мы не знаем, тогда разложим по степеням h *u* и *k*:

$$u_{i+1} = u(r_i + h) = u_i + h\frac{du_i}{dr} + \frac{h^2}{2}\frac{d^2u_i}{dr^2} + \frac{h^3}{6}\frac{d^3u_i}{dr^3} + \frac{h^4}{24}\frac{d^4u_i}{dr^4} + O(h^5) \text{ [1]}$$

$$\frac{\boldsymbol{u_{i+1}-u_i}}{\boldsymbol{h}} = \frac{du_i}{dr} + \frac{h}{2}\frac{d^2u_i}{dr^2} + \frac{h^2}{6}\frac{d^3u_i}{dr^3} + \frac{h^3}{24}\frac{d^4u_i}{dr^4} + O(h^4) \text{ [2]}$$

$$u_{i-1} = u(r_i - h) = u_i - h\frac{du_i}{dr} + \frac{h^2}{2}\frac{d^2u_i}{dr^2} - \frac{h^3}{6}\frac{d^3u_i}{dr^3} + \frac{h^4}{24}\frac{d^4u_i}{dr^4} + O(h^5) \text{ [3]}$$

$$\frac{\boldsymbol{u_i-u_{i-1}}}{\boldsymbol{h}} = \frac{du_i}{dr} - \frac{h}{2}\frac{d^2u_i}{dr^2} + \frac{h^2}{6}\frac{d^3u_i}{dr^3} - \frac{h^3}{24}\frac{d^4u_i}{dr^4} + O(h^4) \text{ [4]}$$

$$\boldsymbol{k_{i+\frac{1}{2}}} = k\left(r_i + \frac{h}{2}\right) = k_i + \frac{h}{2}\frac{dk_i}{dr} + \frac{h^2}{8}\frac{d^2k_i}{dr^2} + \frac{h^3}{48}\frac{d^3k_i}{dr^3} + O(h^4) \text{ [5]}$$

$$\boldsymbol{k_{i-\frac{1}{2}}} = k\left(r_i - \frac{h}{2}\right) = k_i - \frac{h}{2}\frac{dk_i}{dr} + \frac{h^2}{8}\frac{d^2k_i}{dr^2} - \frac{h^3}{48}\frac{d^3k_i}{dr^3} + O(h^4) \text{ [6]}$$

Умножим [2] на [5] и [4] на [6] – получим выражения, которые можно будет подставить в [0].

$$\boldsymbol{\varepsilon_i = h\left[rf + \frac{d}{dr}\left(rk\frac{du}{dr}\right) - rqu\right]_{r=r_i} + h^3\left[\frac{1}{12}rk\frac{d^4u}{dr^4} + \frac{1}{6}\frac{d^3u}{dr^3}\frac{d(rk)}{dr} + \frac{1}{8}\frac{d^2(rk)u}{dr^2}\frac{d^2u}{dr^2} + \frac{1}{24}\frac{d^3(rk)}{dr^3}\frac{du}{dr}\right]_{r=r_i} + O(h^4)}$$

2. Левое граничное условие

- условие первого порядка. Оно аппроксимируется точно, следовательно, не выносит никакой погрешности.

3. Правое граничное условие

$$-k(r)\frac{du}{dr}\bigg|_{r=R_R} = \chi_2 u|_{r=R_r} - \vartheta_2, \chi_2 > 0$$

$$-\left[-(\chi_2 v_i - \vartheta_2) - k_{i-1/2}\frac{v_i - v_{i-1}}{h} - \frac{h}{2}q_i v_i\right] = \frac{h}{2}f_i \text{ , } i=N$$

Подставим раннее полученное разложение по степеням h – $k_{i-\frac{1}{2}}$ и $\frac{u_i-u_{i-1}}{h}$, после группировки слагаемых получаем:

$$\varepsilon_i = h^0\left[-r(\chi_2 u_i - \vartheta_2) - rk_i\frac{du}{dr}\right]_{r=r_i} + h\left[rf + \frac{d}{dr}\left(rk\frac{du}{dr}\right) - rqu\right]_{r=r_i} - h^2\left[\frac{1}{6}\frac{d^3u_i}{dr^3}rk_i + \frac{1}{4}\frac{d(rk_i)u_i}{dr}\frac{d^2u_i}{dr^2} + \frac{1}{8}\frac{d^2(rk_i)}{dr^2}\frac{du_i}{dr}\right]_{r=r_i} + O(h^3)$$

4. Порядок аппроксимации

Для правого граничного условия - разница степеней при h составляет $p = 2 - 0 = 2$. Для уравнения - разница степеней при h: $p = 3 - 1 = 2$

Тесты

Для тестирования написанной программы были найдены тестовые функции с помощью *метода частных решений*.

<u>Тест 1 Константный тест</u>

Введём: u* = 1, k* = 1, q* = 1, RL = 1, RR = 11, $\boldsymbol{\chi_2 = 2}$,

Вычисляем граничные условия $\boldsymbol{\vartheta_1 * = 1}$ и $\boldsymbol{\vartheta_2 = 2}$,

Находим f = 1

N	Тест 1
4	1,110e-16
8	3,331e-16
16	2,220e-16
32	6,661e-16
64	8,882e-16
128	1,221e-15
512	1,732e-14
1024	6,128e-14
2048	1,433e-13
4096	2,714e-13
8192	1,096e-12

В данном тесте погрешность будет зависеть только от вычислительной ошибки(ошибка округления). Аппроксимация не влияет на погрешность, так как $\varepsilon = 0$ (Пункт 6). Тест показал хорошие результаты невязки на первой же итерации при числе разбиений N = 4. Однако, с ростом N погрешность растет. Это связано с накоплением вычислительной ошибки и увеличением числа обусловленности матрицы.

<u>Тест 2 Линейный тест</u>

Введём: u* = $\boldsymbol{r}$, k* = 1, q* = 2, RL = 2, RR = 5, $\boldsymbol{\chi_2 = 1}$,

Вычисляем граничные условия $\boldsymbol{\vartheta_1 * = 2}$ и $\boldsymbol{\vartheta_2 = 6}$

Находим f =$\boldsymbol{2r - 1/r}$

N	Тест 2
4	8,882e-16
8	8,882e-16
16	1,776e-15
32	3,553e-15
64	7,994e-15

128	2,576e-14
512	1,776e-13
1024	3,810e-13
2048	4,974e-13
4096	4,358e-12
8192	1,882e-11

Данные тест предназначен для того, чтобы заметить ошибки, которые возможно были упущены в тест 1, так как класс константных функций не имеет большого доверия при тестировании. Погрешность округления присутствует; увеличивается из-за роста числа обусловленности матрицы.

Тест 3 Тест с погрешностью аппроксимации
Введём: u* = r^3, k* = r^3, q* = 2*r, RL = 2, RR = 5, $\chi_2 = 1$,
Вычисляем граничные условия $\vartheta_1 = 8$ и $\vartheta_2 = 3 * 5^3 * 5^2 + 5^3 = 9500$
Находим f =$-16r^4$

N	Тест 3
4	3,208e+01
8	8,233e+00
16	2,072e+00
32	5,189e-01
64	1,298e-01
128	3,245e-02
512	2,028e-03
1024	5,071e-04
2048	1,268e-04
4096	3,168e-05
8192	7,902e-06

С увеличением числа разбиений, точность повышается – погрешность уменьшается. При увеличении шага в два раза, погрешность уменьшается в 4 раза. Третий тест выдал погрешность при N=4 равную 3,208e+01. С такой погрешностью нужно осторожно принимать решения в зависимости от задачи.

Вывод

Используя интегро-интерполяционный метод (метод баланса) была разработана программа для моделирования распределения температуры в полом цилиндре, описываемого математической моделью с заданными граничными условиями, а также проведен ряд тестов. Было проведено 3 теста. Первый и второй тест демонстрируют накопление вычислительной ошибки из-за округлений и увеличения числа обусловленности матрицы. С увеличением числа разбиений увеличивается погрешность.

Третий тест демонстрируют изменение погрешности аппроксимации. С увеличением числа разбиений погрешность уменьшается, так как точность аппроксимации растет для вектора решения алгебраической системы.

Код программы

```
import java.util.*

val scanner = Scanner(System.`in`)

fun main() {

    //    test()

    print("N    = "); val N = scanner.nextInt()
    print("RL    = "); val RL = scanner.nextDouble()
    print("RR    = "); val RR = scanner.nextDouble()
    print("v1    = "); val v1 = scanner.nextDouble()
    print("v2    = "); val v2 = scanner.nextDouble()
    print("Xi2    = "); val Xi2 = scanner.nextDouble()

    val u: (Double) -> Double = { r: Double -> r }
    val k: (Double) -> Double = { 1.0 }
    val q: (Double) -> Double = { r: Double -> 2.0 }
    val f: (Double) -> Double = { r: Double -> 2.0 * r -1.0/r}

    val solver = Solver(N, RL, RR, v1, v2, Xi2)
    solver.init(u, k, q, f)
    val v = solver.progonka()
    solver.printEps()
    Solver.printVector(solver.r, 'r')
    Solver.printVector(v, 'v')
}

/**
 * **Организует математические вычисления в одном объекте**
 *
 * **0 шаг**
 *
 * Создается объект для системы, в конструкторе задаются параметры: [N], [RL],
[RR], [v1], [v2], [Xi2]
 *
 * **1 шаг**
 *
 * Инициализируются все поля и методы, а именно:
 * * задаются функции вычисляющие k, f, u, q, k_h
 * * вызывается метод [initNet] - инициализирует параметры для основной и
вспомогательной сетки
 * * вызывается метод [initMatrix] - инициализирует коэффициенты для трех-
диагональной матрицы A(a, b, c) и вектор g для СЛАУ Av=g
 *
 * **2 шаг**
 *
 * Вызывается метод [progonka], который решает СЛАУ методом прогонки для трех
диагональной матрицы A
 *
 * **3 шаг**
 *
 * Вызывается метод [computeErrorAndShow], считает невязку[[u]-[v]] для каждого
i-ого разбиения и выбирает максимальную - погрешность всего уравнения
 *
 * @param N число разбиений > 0
 * @param RL начало промежутка > 0
 * @param RR конец промежутка
```

```kotlin
 * @param v1 первое граничное условие
 * @param v2 часть второго граничного условия
 * @param Xi2 часть второго граничного условия
 *
 * @property h веткор "шаг" основной сетки
 * @property h_h вектор "шаг" вспомогательной сетки
 * @property r
 * @property r_h
 *
 * @property k_h fun k_h(**x**: Number):Double внутренняя функция уравнений
 * **x** - аргумент функции r[[RL];[RR]]
 * @property q fun q(**x**: Number):Double внутренняя функция уравнений
 * **x** - аргумент функции r[[RL];[RR]]
 * @property u fun u(**x**: Number):Double внутренняя функция уравнений
 * **x** - аргумент функции r[[RL];[RR]]
 * @property f fun f(**x**: Number):Double внутренняя функция уравнений
 * **x** - аргумент функции r[[RL];[RR]]
 */

class Solver(var N: Int, val RL: Double, val RR: Double, val v1: Double, val
v2: Double, val Xi2: Double) {
    val a: Array<Double>
    val b: Array<Double>
    val c: Array<Double>

    var v: Array<Double>
    var g: Array<Double>

    private val h: Array<Double>
    private val h_h: Array<Double>
    val r: Array<Double>
    private val r_h: Array<Double>

    lateinit var k: ((Double) -> Double)
    lateinit var q: ((Double) -> Double)
    lateinit var u: ((Double) -> Double)
    lateinit var f: ((Double) -> Double)

    init {
        if (N < 0) N = 0

        a = Array(N + 1) { 0.0 }
        b = Array(N + 1) { 0.0 }
        c = Array(N + 1) { 0.0 }

        v = Array(N + 1) { 0.0 }
        g = Array(N + 1) { 0.0 }

        h = Array(N+1){0.0}
        h_h = Array(N+1){0.0}
        r = Array(N+1){0.0}
        r_h = Array(N+1){0.0}
    }

    fun init(u_: (Double) -> Double, k_: (Double) -> Double, q_: (Double) ->
Double, f_: (Double) -> Double ) {
        u = u_
        k = k_
        q = q_
        f = f_
```

```
        initNet()
        initMatrix()
    }

    /**
     * Заполняет значения векторов r и h для основной сетки и вспомогательной.
     *
     * Массивы r и h нужны для ИИМ (интегро-интерполяционного метода)
     */
    private fun initNet() {
        var step = RL
        val inc: Double = (RR - RL) / N

        // основная сетка
        r[0] = step
        for (i in 1..N) {
            step += inc
            r[i] = step
            h[i] = inc
        }

        // вспомогательная сетка
        h_h[0] = h[1]/2.0
        h_h[N] = h[N]/2.0
        r_h[N] = (r[N] + r[N - 1]) / 2.0
        for (i in 1 until N) {
            r_h[i] = (r[i] + r[i - 1]) / 2.0
            h_h[i] = (h[i] + h[i + 1]) / 2.0
        }

        println("Инициализация сетки успешна")
    }

    /**
     * Заполняет матричное уравнение Av=g: матрицу A, вектор g.
     */
    private fun initMatrix() {

        // граничное условие слева - первая строчка СЛАУ
        c[0] = 1.0
        b[0] = 0.0
        g[0] = v1

        // temp variables
        var k_m: Double  // k1/2-1
        var k_p: Double  // k1/2+1
        var r_m: Double  // r1/2-1
        var r_p: Double  // к1/2+1

        // основная часть СЛАУ
        for (i in 1 until N) {
            k_m = k(r[i]-h[i]/2)
            k_p = k(r[i]+h[i]/2)
            r_m = r_h[i]
            r_p = r_h[i+1]

            a[i] = -((r_m*k_m)/h[i])
            c[i] = ((r_m*k_m)/h[i] + (r_p*k_p)/h[i+1] + h_h[i]*r[i]*q(r[i]))
            b[i] = -((r_p*k_p)/h[i+1])
            g[i] = h_h[i]*r[i]*f(r[i])
        }
```

```
        // граничное условие справа - последняя строчка СЛАУ

        a[N] = -((r_h[N]*k(r[N]-h[N]/2))/h[N])
        c[N]  = ((r_h[N]*k(r[N]-h[N]/2))/h[N] + h_h[N]*r[N]*q(r[N]) +
r[N]*Xi2);
        g[N] = h_h[N]*r[N]*f(r[N]) + r[N]*v2

        println("Инициализация матрицы A и вектора g успешны")
    }

    /**
     * Находит решение матричного уравнения Av=g методом 'прогонки' (прямой и
обратный ход)
     *
     * Решение ищется в виде v(i)=α(i+1)*v(i+1)+β(i+1)
     * @param A матрица размера N*N
     * @param g вектор размера N
     *
     * @return v вектор решений
     */
    fun progonka(): Array<Double> {
        // прогоночные коэффициенты alpha и beta
        val alpha: Array<Double> = Array(g.size) { 0.0 }
        val beta: Array<Double> = Array(g.size) { 0.0 }

        // прямой ход - считаем коэффициенты alpha и beta
        alpha[1] = -b[0] / c[0]
        beta[1] = g[0] / c[0]

        for (i in 1 until N) {
            alpha[i + 1] = -(b[i]) / (alpha[i] * a[i] + c[i])
            beta[i + 1] = (g[i] - a[i] * beta[i]) / (a[i] * alpha[i] + c[i])
        }

        // обратный ход - находим веткор решений v
        v[N] = ((g[N] - a[N] * beta[N]) / (a[N] * alpha[N] + c[N]))
        for (i in N-1 downTo 0 step 1) {
            v[i] = alpha[i + 1] * v[i + 1] + beta[i + 1]
        }

        println("Расчет нахождения решения СЛАУ методом прогонки закончен")
        return v
    }

    /**
     * Считает невязку [u-v] для каждой точки.
     * Находит максимальное отклонение по модулю, что и считается за
погрешность всего решения
     */
    fun printEps() {
        var maxEps= -1.0
        var eps: Double

        for(i in 0..N) {
            eps = kotlin.math.abs(u(r[i]) - v[i])
            if (eps > maxEps) {
                maxEps = eps
            }
        }
        println(String.format("eps = %.3e \n\n", maxEps))
```

```
    }

    companion object {
        fun printVector(v: Array<Double>, ch: Char) {
            print("$ch = ")
            v.forEach { i -> print(String.format("%2.6f; ", i)) }
            println()
        }
    }
}
```

Код тестирования

```
fun test() {
    val n = listOf(2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192,
16000, 32000, 64000)
    for (i in n) test1(i)
    for (i in n) test2(i)
    for (i in n) test3(i)
}

fun test1(N:Int) {
    println("Test 1 N = $N")
    /* Вычислительная error constant */
    val RL = 1.0
    val RR = 11.0

    val v1 = 1.0
    val Xi2 = 2.0
    val v2 = 2.0

    val u: (Number) -> Double   = {1.0}
    val k: (Number) -> Double   = {1.0}
    val q: (Number) -> Double   = {1.0}
    val f: (Number) -> Double   = {1.0}

    val solver = Solver(N, RL, RR, v1, v2, Xi2)
    solver.init(u, k, q, f)
    solver.progonka()
    solver.printEps()
}

fun test2(N:Int) {
    /* Вычислительная error linear */
    println("Test 3 N = $N")
    val RL = 2.0
    val RR = 5.0

    val v1 = 2.0
    val Xi2 = 1.0
    val v2 = 6.0

    val u: (Double) -> Double = { r: Double -> r }
    val k: (Double) -> Double = { 1.0 }
    val q: (Double) -> Double = { r: Double -> 2.0 }
    val f: (Double) -> Double = { r: Double -> 2.0 * r -1.0/r}
```

```
    val solver = Solver(N, RL, RR, v1, v2, Xi2)
    solver.init(u, k, q, f)
    solver.progonka()
    solver.printEps()
}

fun test3(N:Int) {
    /* Approximation error General test */
    println("Test 3 N = $N")
    val RL = 2.0
    val RR = 5.0

    val v1 = 8.0
    val Xi2 = 1.0
    val v2 = 9500.0

    val u: (Double) -> Double  = { r: Double -> r*r*r }
    val k: (Double) -> Double  = { r: Double -> r*r*r }
    val q: (Double) -> Double  = { r: Double -> 2.0*r }
    val f: (Double) -> Double  = { r: Double -> -16.0*r*r*r*r}

    val solver = Solver(N, RL, RR, v1, v2, Xi2)
    solver.init(u, k, q, f)
    solver.progonka()
    solver.printEps()
}
```