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»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

По курсу: "Моделирование"

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

Группа ИУ7-63Б

Студент Сукочева А.

Преподаватель Градов В.М.

Москва — 2021 г.

0.1 Постановка задачи

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

0.1.1 Исходные данные

Задана математическая модель:

$$\frac{d}{dx}(\lambda(T)\frac{dT}{dx}) - 4 \cdot k(T) \cdot n_p^2 \cdot \sigma \cdot (T^4 - T_0^4) = 0$$

Краевые условия:

$$\begin{cases} x = 0, -\lambda(T(0))\frac{dT}{dx} = F_0. \\ x = l, -\lambda(T(l))\frac{dT}{dx} = \alpha(T(l) - T_0) \end{cases}$$

Функции $\lambda(T)$ и $k(T)$ заданы таблицей.

T, K	λ, Вт/(см К)		T, K	k, см$^{-1}$
300	1.36 10^{-2}		293	2.0 10^{-2}
500	1.63 10^{-2}		1278	5.0 10^{-2}
800	1.81 10^{-2}		1528	7.8 10^{-2}
1100	1.98 10^{-2}		1677	1.0 10^{-1}
2000	2.50 10^{-2}		2000	1.3 10^{-1}
2400	2.74 10^{-2}		2400	2.0 10^{-1}

Рис. 1: Таблица функций $\lambda(T)$ и $k(T)$

Заданы начальные параметры:
$n_p = 1.4$ - коэффициент преломления
$l = 0.2$ см - толщина слоя
$T_0 = 300$K - температура окружающей среды
$\sigma = 5.668 \cdot 10^{-12}$ Вт / $(cm^2 \cdot K^4)$ - постоянная Стефана - Больцмана
$F_0 = 100$ Вт / cm^2 - поток тепла
$\alpha = 0.05$ Вт / $(cm^2 \cdot K)$ - коэффициент теплоотдачи

Выход из итераций организовать по температуре и по балансу энергии:

$$max|\frac{y_n^s - y_n^{s-1}}{y_n^s}| <= \varepsilon_1$$

для всех $n = 0, 1, ...N.$ и

$$max|\frac{f_1^s - f_2^s}{f_1^s}| <= \varepsilon_1$$

где

$$f_1 = F_0 - \alpha(T(l) - T_0)$$

$$f_2 = 4n_p^2\sigma \int_0^1 k(T(x))(T^4(x) - T_0^4)dx$$

Физическое содержание задачи.

Сформулированная математическая модель описывает температурное поле T(x) в плоском слое с внутренними стоками тепловой энергии. Можно представить, что это стенка из полупрозрачного материала, например, кварца или сапфира, нагружаемая тепловым потоком на одной из поверхностей (у нас - слева). Другая поверхность (справа) охлаждается потоком воздуха, температура которого равна T_0. Например, данной схеме удовлетворяет цилиндрическая оболочка, ограничивающая разряд в газе, т.к. при больших диаметрах цилиндра стенку можно считать плоской. При высоких температурах раскаленный слой начинает объемно излучать. Зависимость от температуры излучательной способности материала очень резкая. При низких температурах стенка излучает очень слабо. Функции lambda(T),k(T) являются, соответственно, коэффициентами теплопроводности и оптического поглощения материала стенки

0.2 Реализация

```
class Params:
        n_p = 1.4
        l = 0.2
        T0 = 300
        sigma = 5.668*1e-12
        F0 = 100
        alpha = 0.05
        h = 1e-4

        fst_table = ((300, 500, 800, 1100, 2000, 2400),
                        (1.36 * pow(10, -2), 1.63 * pow(10, -2), 1.81 *
                           pow(10, -2),
                        1.98 * pow(10, -2), 2.50 * pow(10, -2), 2.74 * pow(10,
                           -2)))

        snd_table = ((293, 1278, 1528, 1677, 2000, 2400),
                (2.0 * pow(10, -2), 5.0 * pow(10, -2), 7.8 * pow(10, -2),
                1.0 * pow(10, -1), 1.3 * pow(10, -1), 2.0 * pow(10, -1)),)

params = Params()

class Functions:
        @staticmethod
        def Interpolate(x_pts, y_pts, order=1):
                return InterpolatedUnivariateSpline(x_pts, y_pts, k=order)
        @staticmethod
        def p(k_t, t, n):
                return 0
        @staticmethod
        def f(k_t, t, n):
                return 4 * params.Np * params.Np + params.sigma * k_t(t[n]) *
                   (pow(t, 4) - pow(params.T0, 4))
        @staticmethod
        def x_right(l_t, t, n):
                return (l_t(t[n]) + l_t(t[n + 1])) / 2
        @staticmethod
        def x_left(l_t, t, n):
                return (l_t(t[n]) + l_t(t[n - 1])) / 2
        @staticmethod
        def p_right(k_t, t, n):
                return (Functions.p(k_t, t, n) + Functions.p(k_t, t, n + 1)) /
                   2
        @staticmethod
        def p_left(k_t, t, n):
                return (Functions.p(k_t, t, n) + Functions.p(k_t, t, n - 1)) /
                   2
        @staticmethod
        def f_right(k_t, t, n):
                return (Functions.f(k_t, t, n) + Functions.f(k_t, t, n + 1)) /
                   2
        @staticmethod
        def f_left(k_t, t, n):
                return (Functions.f(k_t, t, n) + Functions.f(k_t, t, n - 1)) /
                   2
        @staticmethod
        def A(l_t, t, n):
                return (l_t(t[n]) + l_t(t[n - 1])) / 2 / params.h
        @staticmethod
        def B(l_t, k_t, t, n):
```

```
            return Functions.A(l_t, t, n) + Functions.C(l_t, t, n)
    @staticmethod
    def C(l_t, t, n):
            return (l_t(t[n]) + l_t(t[n + 1])) / 2 / params.h
    @staticmethod
    def D(k_t, t, n):
            return -4 * k_t(t[n]) * params.Np * params.Np * params.sigma *
                (pow(t[n], 4) - pow(params.T0, 4)) * params.h


def GetRightConditions(l_t, k_t, t):
    K0 = Functions.x_right(l_t, t, 0) + pow(params.h, 2) / 8 *
        Functions.p_right(k_t, t, 0) + pow(params.h, 2) / 4 *
        Functions.p(k_t, t, 0)
    M0 = pow(params.h, 2) / 8 * Functions.p_right(k_t, t, 0) -
        Functions.x_right(l_t, t, 0)
    P0 = params.h * params.F0 + pow(params.h, 2) / 4 *
        (Functions.f_right(k_t, t, 0) + Functions.f_left(k_t, t, 0))
    return K0, M0, P0


def GetLeftConditions(k_t, l_t, t, n):
    Kn = Functions.x_left(l_t, t, n) / params.h - params.alpha - params.h
        * Functions.p(k_t, t, n) / 4 - params.h * Functions.p_left(k_t, t,
        n) / 8
    Mn = Functions.x_left(l_t, t, n) / params.h - params.h *
        Functions.p_left(k_t, t, n) / 8
    Pn = -(params.alpha * params.T0 + (Functions.f_right(k_t, t, n) +
        Functions.f_left(k_t, t, n)) / 4 * params.h)
    return Kn, Mn, Pn


def SaveImg(xs, ys, name_x, name_y, file_name):
    fig, ax = plt.subplots()
    ax.plot(xs, ys)
    ax.grid()
    ax.set_xlabel(name_x)
    ax.set_ylabel(name_y)
    plt.savefig(file_name, bbox_inches="tight")

def Main():
    l_t = Functions.Interpolate(params.fst_table[0], params.fst_table[1])
    k_t = Functions.Interpolate(params.snd_table[0], params.snd_table[1])
    t = [0 for _ in range(int(1 / params.h) + 2)]
    K0, M0, P0 = GetRightConditions(l_t, k_t, t)
    xi_list = [0]
    eta_list = [0]
    x_list = list()
    x = 0
    n = 0
    while x + params.h < 1:
            x_list.append(x)
            xi_list.append(Functions.C(l_t, t, n) / (Functions.B(l_t, k_t,
                t, n) - Functions.A(l_t, t, n) * xi_list[n]))
            eta_list.append((Functions.D(k_t, t, n) + Functions.A(l_t, t,
                n) * xi_list[n]) / (Functions.B(l_t, k_t, t, n) -
                Functions.A(l_t, t, n) * xi_list[n]))
            n += 1
            x += params.h
    x_list.extend([x + params.h, x + params.h * 2])
    Kn, Mn, Pn = GetLeftConditions(k_t, l_t, t, n)
    t[n] = (Pn - Mn * xi_list[n]) / (Kn + Mn * xi_list[n])
    for i in range(n - 1, -1, -1):
```

```
                t[i] = xi_list[i + 1] * t[i + 1] + eta_list[i + 1]
        SaveImg(x_list, t, "l", "T", "img_1.png")


if __name__ == "__main__":
        Main()
```

0.3 Экспериментальная часть

0.3.1 1. Представить разностный аналог краевого условия при $x = l$ и его краткий вывод интегро-интерполяционным методом.

№1.

$$\begin{cases} \frac{d}{dx}\left(\lambda(x)\frac{dT}{dx}\right) - 4k(T)n_p^2\sigma(T^4 - T_0^4) = 0 \\ x = 0, \quad -\lambda(T(0))\frac{dT}{dx} = F_0 \\ x = l, \quad -\lambda(T(l))\frac{dT}{dx} = \alpha(T(l) - T_0) \end{cases}$$

Приведем к виду:

$$\frac{d}{dx}\left(k(u,x)\frac{du}{dx}\right) - p(x,u)u + f(x,u) = 0$$

где $p(x,u) = 0$

$$f(x,u) = f(u) = -4k(T)n_p^2\sigma(T^4 - T_0^4)$$

Обозначение: $F = -\lambda(x)\frac{dT}{dx}$

$$\int_{x_{N-1/2}}^{x_N} \frac{dF}{dx}dx + \int_{x_{N-1/2}}^{x_N} f(x)\,dx = 0$$

$$-(F_N - F_{N-1/2}) + \frac{h}{4}(f_N + f_{N-1/2}) = 0$$

Учитывая:

$$F_N = \alpha_N(y_N - T_0) \qquad F_{N-1/2} = \chi_{N-1/2}\frac{y_{N-1} - y_N}{h}$$

Получаем:

$$\alpha_N(y_N - T_0) - \frac{h}{4}(f_N + f_{N-1/2}) = \chi_{N-1/2}\frac{y_{N-1} - y_N}{h}$$

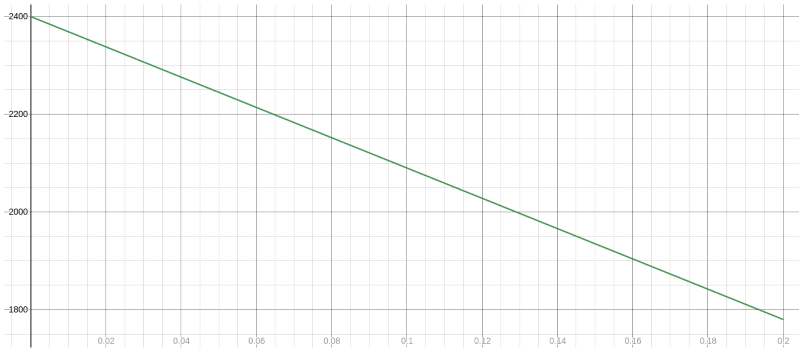
$$-T_0\alpha_N - \frac{h}{4}(f_N + f_{N-1/2}) = y_{N-1}\left(\frac{\chi_{N-1/2}}{h}\right) + y_N\left(-\alpha_N - \frac{\chi_{N-1/2}}{h}\right)$$

Полученное краевое условие:

$$K_N y_{N-1} + M_N y_N = P_N$$

0.3.2 2. График зависимости температуры $T(x)$ координаты x при заданных выше параметрах.

Выяснить, как сильно зависят результаты расчета T(x) и необходимое для этого количество итераций от начального распределения температуры и шага сетки.



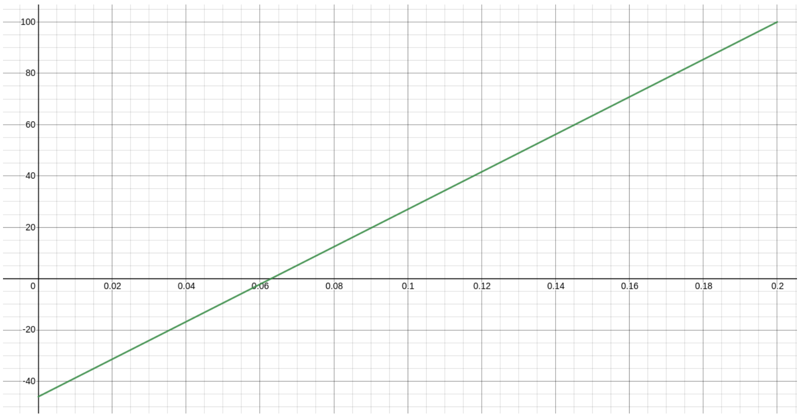
Кол-во итераций: 23

Изменим параметры: $T_0 = 2000$.

Кол-во итераций: 9

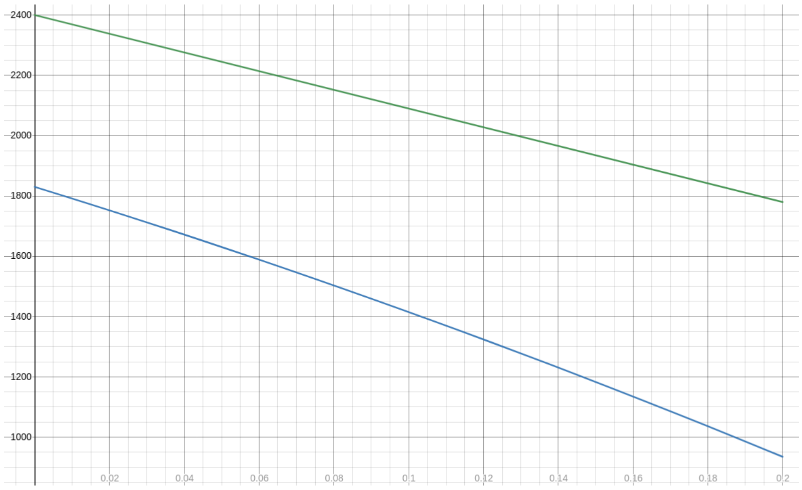
При увеличении T_0 получается более выраженная выпуклость кривых. При этом количество итераций значительно снижается.

0.3.3 3. График зависимости $T(x)$ при $F_0 = -10\frac{B}{cm^2}$.

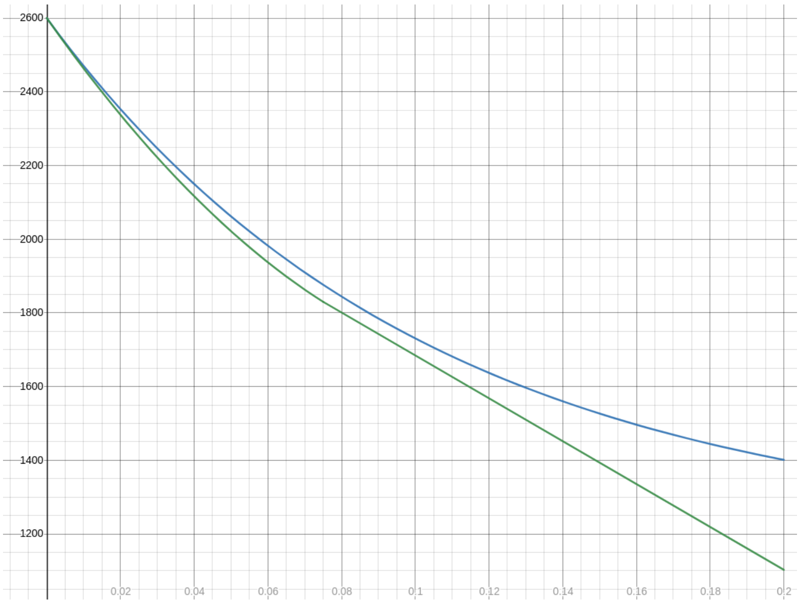


0.3.4 4. График зависимости $T(x)$ при увеличенных значениях α (например, в 3 раза). Сравнить с п. 2.

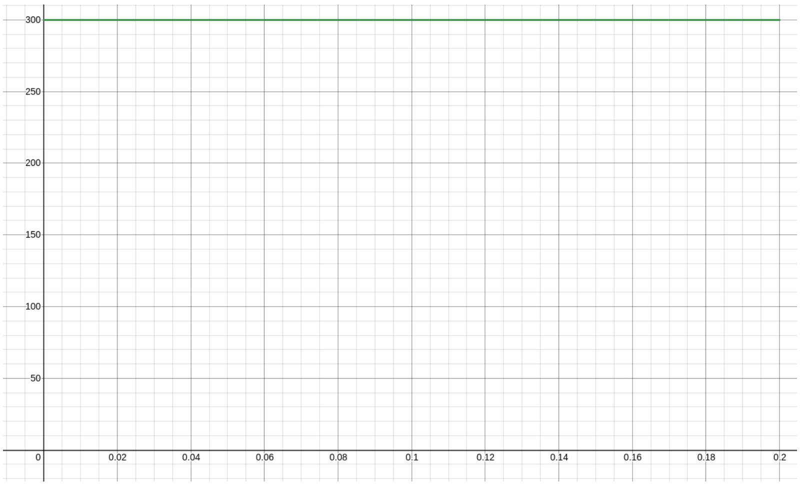
Синяя кривая при увеличенном в 3 раза коэффициенте теплоотдачи.



Зеленая кривая при увеличенном в 3 раза коэффициенте теплоотдачи.



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



0.3.6 6. Для указанного в задании исходного набора параметров привести данные по балансу энергии.

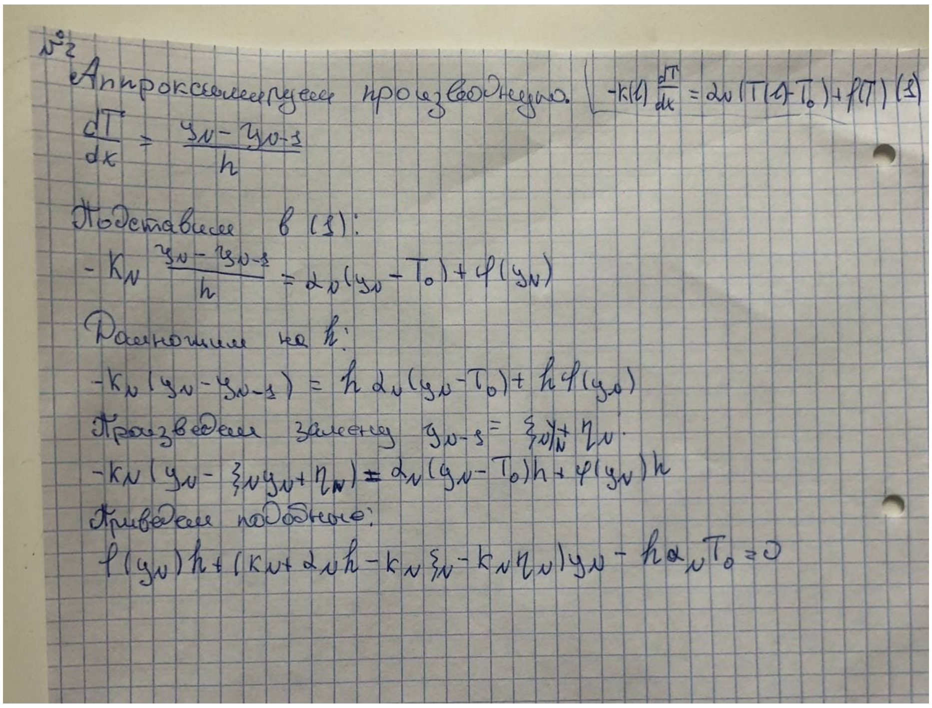
- Точность выхода $\varepsilon_1 = 1\text{e-}5$ (по температуре)
- Точность выхода $\varepsilon_2 = 1\text{e-}3$ (по балансу)

0.4 Ответы на вопросы

1. Какие способы тестирования программы можно предложить?

В данной программе можно предложить тестирование при помощи изменения значения параметра F_0. Также при $F_0 > 0$ происходит охлаждение пластины, при $F_0 < 0$ нагревание. При увеличении теплосъема и неизменном потоке F_0 уровень температур T(x) должен снижаться, а градиент увеличиваться.

2. Получите простейший разностный аналог нелинейного краевого условия при $x = l$.



№2

Аппроксимируем производную. $-k(l)\frac{dT}{dx} = \alpha_N(T(l)-T_0)+f(T)$ (1)

$\frac{dT}{dx} = \frac{y_N - y_{N-1}}{h}$

Подставим в (1):

$-k_N\frac{y_N-y_{N-1}}{h} = \alpha_N(y_N - T_0)+\varphi(y_N)$

Домножим на h:

$-k_N(y_N - y_{N-1}) = h\alpha_N(y_N-T_0)+h\varphi(y_N)$

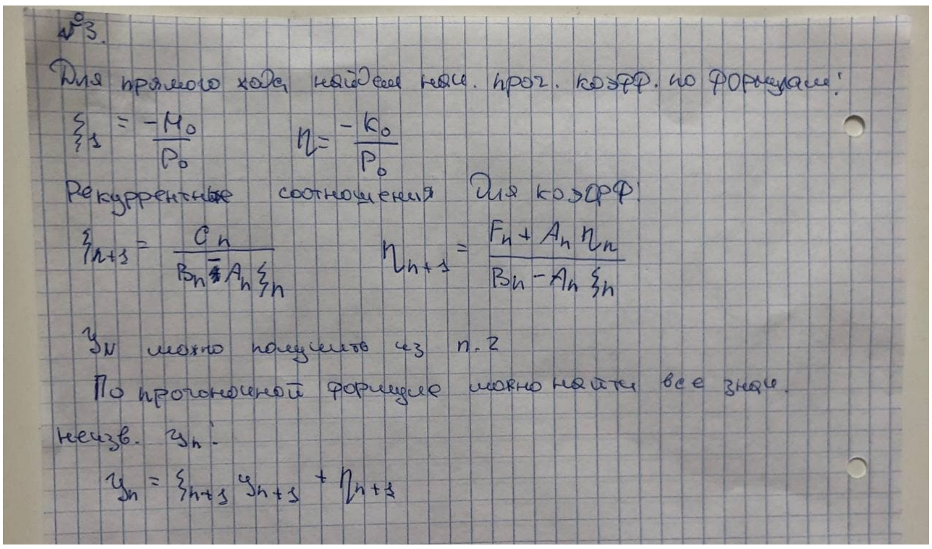
Произведем замену $y_{N-1} = \xi_N y_N + \eta_N$:

$-k_N(y_N - \xi_N y_N + \eta_N) = \alpha_N(y_N-T_0)h + \varphi(y_N)h$

Приведем подобные:

$\varphi(y_N)h + (k_N + \alpha_N h - k_N\xi_N - k_N\eta_N)y_N - h\alpha_N T_0 = 0$

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



№3.

Для прямого хода найдем нач. прог. коэфф. по формулам:

$\xi_1 = \frac{-M_0}{P_0}$ $\eta = \frac{-K_0}{P_0}$

Рекуррентные соотношения для коэфф:

$\xi_{n+1} = \frac{C_n}{B_n - A_n\xi_n}$ $\eta_{n+1} = \frac{F_n + A_n\eta_n}{B_n - A_n\xi_n}$

y_N можно получить из п. 2

По прогоночной формуле можно найти все знач. неизв. y_n:

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

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.

4. Вычисляем нач. прог. коэфф. Для правой прогонки:

$$\xi_1 = \frac{-M_0}{P_0} \qquad \eta_1 = \frac{-K_0}{P_0}$$

Для левой прогонки:

$$\alpha_{N-1} = \frac{-M_N}{K_N} \qquad \beta_{N-1} = \frac{-P_N}{K_N}$$

Рек. соотнош. для коэфф. Для правой прогонки:

$$\xi_{n+1} = \frac{C_n}{B_n - A_n \xi_n} \qquad \eta_{n+1} = \frac{F_n + A_n \eta_n}{B_n - A_n \xi_n}$$

Для левой прогонки:

$$\alpha_{n-1} = \frac{A_n}{B_n - C_n \alpha_n} \qquad \beta_{n-1} = \frac{F_n + C_n \beta_n}{B_n - C_n \alpha_n}$$

Рек. соотн. для левой и правой прогонки:

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

$$y_n = \alpha_{n-1} y_{n+1} + \beta_{n-1}$$

Выразим y_p:

$$\begin{cases} y_{p-1} = \xi_p y_p + \eta_p \\ y_p = \alpha_{p-1} y_{p-1} + \beta_{p-1} \end{cases} \Rightarrow y_p = \frac{\xi_{n+1}\beta_n + \eta_{n+1}}{1 - \xi_{n+1}\alpha_n}$$