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»

Лабораторная работа № 4

Дисциплина: Моделирование

Тема: Программно-алгоритмическая реализация моделей на основе ОДУ в частных производных с краевыми условиями I и III рода.

Студент Тимонин А. С.

Группа ИУ7-62Б

Оценка (баллы) _______________

Преподаватель Градов В.М.

Москва.
2020 г.

Тема

Программно-алгоритмическая реализация моделей на основе ОДУ в частных производных с краевыми условиями I и III рода.

Цель работы

Получение навыков разработки алгоритмов решения смешанной краевой задачи при реализации моделей, построенных на квазилинейном уравнении параболического типа.

Задача

1.Задана математическая модель.

Уравнение для функции $T(x,t)$

$$c\,(T)\frac{\partial T}{\partial t}=\frac{\partial}{\partial x}\left(k(T)\frac{\partial T}{\partial x}\right)-\frac{2}{R}\alpha(x)T+\frac{2T_0}{R}\alpha(x) \qquad (1)$$

Краевые условия

$$\begin{cases} t=0, \quad T(x,0)=T_0, \\ x=0, \quad -k(T(0))\,\dfrac{\partial T}{\partial x}=F_0, \\ x=l, \quad -k(T(l))\,\dfrac{\partial T}{\partial x}=\alpha_N\,(T(l)-T_0) \end{cases}$$

В обозначениях уравнения (14.1) лекции №14

$$p(x)=\frac{2}{R}\alpha(x), \qquad f(u)\equiv f(x)=\frac{2T_0}{R}\alpha(x)$$

2. Разностная схема с разностным краевым условием при $x=0$ получена в Лекции №14 (14.6),(14.7) и может быть использована в данной работе. Самостоятельно надо получить интегро-интерполяционным методом разностный аналог краевого условия при $x=l$, точно так же, как это сделано при

$x=0$ (формула (14.7)). Для этого надо проинтегрировать на отрезке [$x_{N-1/2}$, x_N] выписанное выше уравнение (1) и учесть, что:

$$\widehat{F}_N = \alpha_N\,(\widehat{y}_N - T_0)$$

$$\widehat{F}_{N-1/2} = \widehat{\chi}_{N-1/2}\,\frac{\widehat{y}_{N-1} - \widehat{y}_N}{h}$$

3. Значения параметров для отладки (все размерности согласованы)

$k(T) = a_1(b_1 + c_1\,T^{m_1})$, Вт/см К,

$c(T) = a_2 + b_2\,T^{m_2} - \dfrac{c_2}{T^2}$, Дж/см^{3}К.

a_1=0.0134, b_1=1, c_1=4.35 10^{-4}, m_1=1,

a_2=2.049, b_2=0.563 10^{-3}, c_2=0.528 10^5, m_2=1.

$\alpha(x) = \dfrac{c}{x-d}$,

$\alpha_0 = 0.05$ Вт/см^2 К,

$\alpha_N = 0.01$ Вт/см^2 К,

$l =$ 10 см,

$T_0 = 300$К,

$R = 0.5$ см,

$F(t) = 50$ Вт/см^2.

Физическое содержание задачи

1. Сформулированная в данной работе математическая модель описывает **нестационарное** температурное поле $T(x,t)$, зависящее от координаты x и меняющееся во времени.

2. Свойства материала стержня привязаны к температуре, т.е. теплоемкость и коэффициент теплопроводности $c(T), k(T)$ зависят от T.

3. При $x=0$ цилиндр нагружается тепловым потоком $F(t)$, в общем случае зависящим от времени.

Результат работы программы

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

Проинтегрируем уравнение $c(u)\frac{\partial u}{\partial t}=-\frac{\partial F}{\partial x}-p(x)u+f(u)$ на отрезке $[x_{N-1/2}; x_N]$ и на временном интервале $[t_m; t_{m+1}]$.

С учетом $F=-k(x)\frac{\partial u}{\partial x}$:

$$\int_{x_{N-1/2}}^{x_N} dx \int_{t_m}^{t_{m+1}} c(u)\frac{\partial u}{\partial t}\partial t = -\int_{t_m}^{t_{m+1}} dt \int_{x_{N-1/2}}^{x_N} \frac{\partial F}{\partial x} dx -$$

$$-\int_{x_{N-1/2}}^{x_N} dx \int_{t_m}^{t_{m+1}} p(x)u dt + \int_{x_{N-1/2}}^{x_N}\int_{t_m}^{t_{m+1}} f(u)dt$$

Приближенно вычислим интегралы по времени:

$$\int_{x_{N-1/2}}^{x_N} \hat{c}\,(\hat{u}-u)dx = -\int_{t_m}^{t_{m+1}} \left(F_N - F_{N-1/2}\right)dt - \int_{x_{N-1/2}}^{x_N} \hat{u}p\tau dx + \int_{x_{N-1/2}}^{x_N} \hat{f}\tau dx$$

Первый интеграл вычисляем методом правых прямоугольников, остальные методом трапеций:

$$\frac{h}{4}\left[\widehat{c_N}\left(\widehat{y_N}-y_N\right)+c_{\widehat{N-1/2}}\left(y_{\widehat{N-1/2}}-y_{N-1/2}\right)\right] = -\left(\widehat{F_N}-F_{\widehat{N-1/2}}\right)\tau -$$

$$-\frac{h\tau}{4}\left(p_N\widehat{y_N}-p_{N-1/2}y_{\widehat{N-1/2}}\right)+\frac{h\tau}{4}\left(f_{\widehat{N-1/2}}-\widehat{f_N}\right)$$

Подставляя в данное уравнение:

$$\widehat{F_N}=F(t_{m+1})=\ \alpha\left(\widehat{y_N}-\beta\right),\ \alpha=\alpha_N, \beta=T_0,$$

$$y_{\widehat{N-1/2}}=\frac{\widehat{y_N}+\widehat{y_{N-1}}}{2},$$

$$y_{N-1/2}=\frac{y_N+y_{N-1}}{2}.$$

Найдем разностный аналог краевого условия:

$$\left[\alpha\tau + \frac{h}{4}\widehat{c_N} + \frac{h}{8}\widehat{c_{N-1/2}} + \frac{\widehat{\chi_{N-1/2}}\tau}{h} + \frac{p_N\tau h}{4} + \frac{p_{N-1/2}\tau h}{8}\right]\widehat{y_N} + \\ + \left[\frac{h}{8}\widehat{c_{N-1/2}} + \frac{\widehat{\chi_{N-1/2}}\tau}{h} + \frac{p_{N-1/2}\tau h}{8}\right]\widehat{y_{N-1}} = \\ = \alpha\beta\tau + \frac{h\tau}{4}\left(\widehat{f_{N-1/2}} - \widehat{f_N}\right) + \frac{h}{4}\widehat{c_N}y_N + \frac{h}{4}\widehat{c_{N-1/2}}\frac{y_N + y_{N-1}}{2}$$

2. График зависимости температуры $T(x,t_m)$ от координаты x при нескольких фиксированных значениях времени t_m при заданных выше параметрах. Обязательно представить распределение $T(x,t)$ в момент времени, соответствующий установившемуся режиму, когда поле перестает меняться с некоторой точностью, т.е. имеет место выход на стационарный режим.

Верхняя кривая соответствует установившемуся режиму (выход на стационарный режим происходит на 62 секунде).

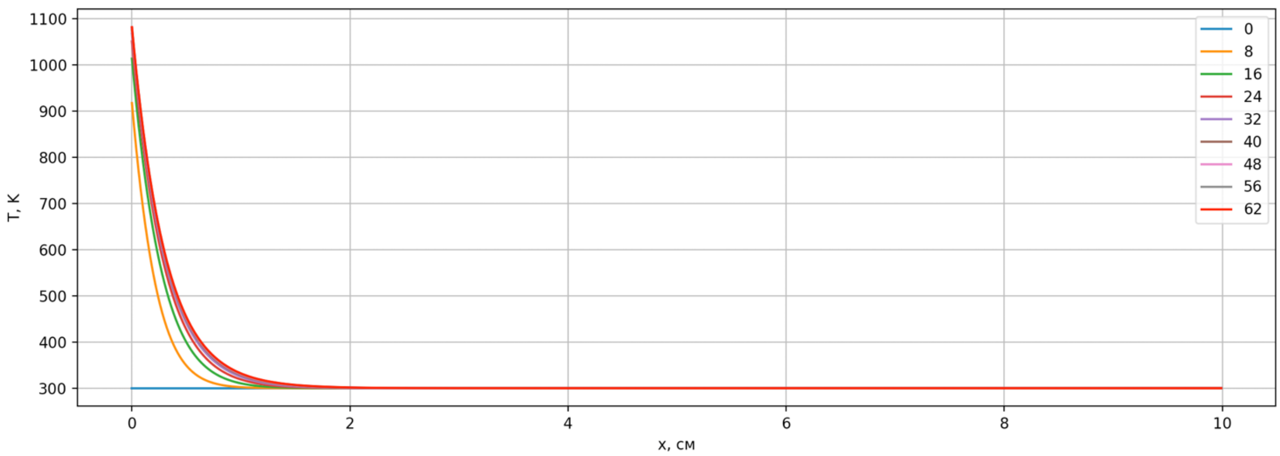

Рисунок 1. График зависимости $T(x,t_m)$

3. График зависимости $T(x_n,t)$ при нескольких фиксированных значениях координаты x_n. Обязательно представить случай n=0, т.е. $x = x_0 = 0$.

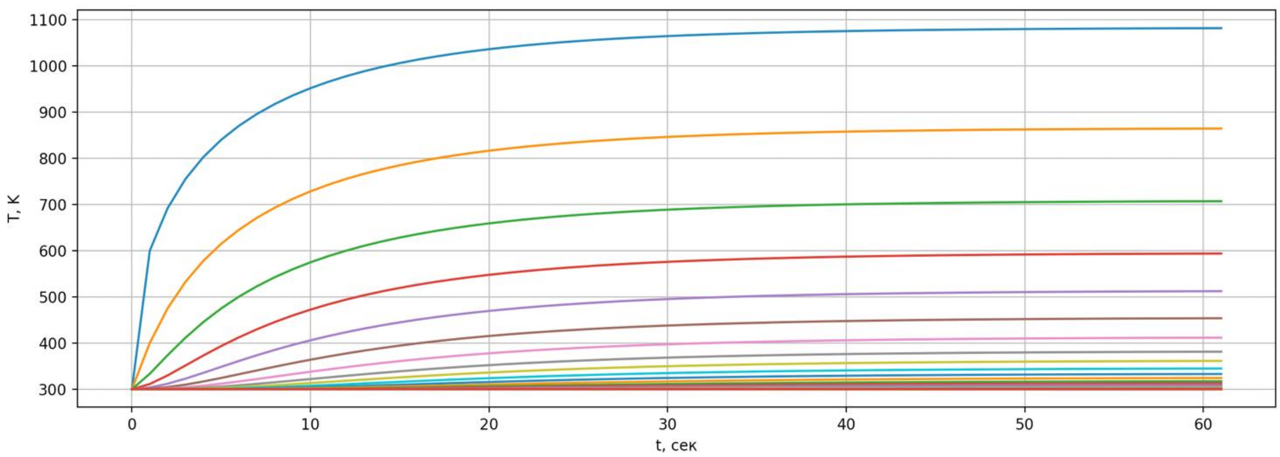

Рисунок 2. График зависимости $T(x_n, t)$

Листинг 1. Реализация программы

```
import matplotlib.pyplot
from math import fabs
import numpy

a_0 = 0.05
a_N = 0.01
l = 10
T_0 = 300
R = 0.5
F_0 = 50
h = 0.01
t = 1

a_1 = 0.0134
b_1 = 1
c_1 = 4.35e-4
m_1 = 1

a_2 = 2.049
b_2 = 0.563e-3
c_2 = 0.528e5
m_2 = 1

d = a_N * l / (a_N - a_0)
c = -(a_0 * d)

def getK(T):
    return a_1 * (b_1 + c_1 * (T ** m_1))

def getC(T):
    return a_2 + b_2 * T ** m_2 - c_2 / T ** 2

def getP(x):
    return 2 * c / (R * (x - d))

def getF(x):
    return 2 * c * T_0 / (R * (x - d))
```

```python
def plusHalf(x, h, f):
    return (f(x) + f(x + h)) / 2

def minusHalf(x, h, f):
    return (f(x) + f(x - h)) / 2

def An(T):
    return minusHalf(T, t, getK) * t / h

def Dn(T):
    return plusHalf(T, t, getK) * t / h

def Bn(x, T):
    return An(T) + Dn(T) + getC(T) * h + getP(x) * h * t

def Fn(x, T):
    return getF(x) * h * t + getC(T) * T * h


def leftCondition(p_T):
    k_0 = h / 8 * plusHalf(p_T[0], t, getC) + h / 4 * getC(p_T[0]) + \
         plusHalf(p_T[0], t, getK) * t / h + \
         t * h / 8 * getP(h / 2) + t * h / 4 * getP(0)

    m_0 = h / 8 * plusHalf(p_T[0], t, getC) - \
         plusHalf(p_T[0], t, getK) * t / h + t * h * getP(h / 2) / 8

    p_0 = h / 8 * plusHalf(p_T[0], t, getC) * (p_T[0] + p_T[1]) + \
         h / 4 * getC(p_T[0]) * p_T[0] + F_0 * t + \
         t * h / 8 * (3 * getF(0) + getF(h))
    return k_0, m_0, p_0

def rightCondition(p_T):
    k_n = h / 8 * minusHalf(p_T[-1], t, getC) + \
         h / 4 * getC(p_T[-1]) + \
         minusHalf(p_T[-1], t, getK) * t / h + \
         t * a_N + t * h / 8 * getP(l - h / 2) + t * h / 4 * getP(l)

    m_n = h / 8 * minusHalf(p_T[-1], t, getC) - \
         minusHalf(p_T[-1], t, getK) * t / h + \
         t * h * getP(l - h / 2) / 8

    p_n = h / 8 * minusHalf(p_T[-1], t, getC) * (p_T[-1] +  p_T[-2]) + \
          h / 4 * getC(p_T[-1]) * p_T[-1] + \
         t * a_N * T_0 + t * h / 4 * (getF(l) + getF(l - h / 2))
    return k_n, m_n, p_n

def isEndIter(prev, cur):
    for i, j in zip(prev, cur):
        return not fabs((i - j) / j) > 1e-4

def isEndRun(prev, cur):
    max = fabs((prev[0] - cur[0]) / cur[0])
    for i, j in zip(prev, cur):
        tmp = fabs(i - j) / j
        max = tmp if tmp > max else max
    return max < 1

def run(p_T):
```

```
    k_0, m_0, p_0 = leftCondition(p_T)
    k_n, m_n, p_n = rightCondition(p_T)
    eps = [0, -m_0 / k_0]
    eta = [0, p_0 / k_0]
    n = 1

    for x in numpy.arange(h, l, h):
        eps_x = Dn(p_T[n]) / (Bn(x, p_T[n]) - An(p_T[n]) * eps[n])
        eta_x = (Fn(x, p_T[n]) + An(p_T[n]) * eta[n]) / \
                (Bn(x, p_T[n]) - An(p_T[n]) * eps[n])
        n += 1
        eps.append(eps_x)
        eta.append(eta_x)

    y = [0 for _ in range(n + 1)]
    y[n] = (p_n - m_n * eta[n]) / (k_n + m_n * eps[n])
    for i in range(n - 1, -1, -1):
        y[i] = eps[i + 1] * y[i + 1] + eta[i + 1]
    return y

if __name__ == "__main__":
    x = [i for i in numpy.arange(0, l, h)]
    n = len(x)
    T = [T_0 for _ in range(n + 1)]
    n_T = [0 for _ in range(n + 1)]
    res = [T]
    t_total = 0
    time = [0]

    while True:
        tmp_T = T
        while True:
            n_T = run(tmp_T)
            if isEndRun(tmp_T, n_T):
                break
            tmp_T = n_T
        res.append(n_T)
        t_total += t
        time.append(t_total)
        if isEndIter(T, n_T):
            break
        T = n_T

    res_time = []
    for i in range(0, len(res), 5):
        matplotlib.pyplot.plot(x, res[i][:-1])
        res_time.append(time[i])
    matplotlib.pyplot.plot(x, res[-1][:-1], color='red')
    res_time.append(time[-1])

    matplotlib.pyplot.legend(res_time)
    matplotlib.pyplot.xlabel("x, cm")
    matplotlib.pyplot.ylabel("T, K")
    matplotlib.pyplot.grid(True)
    matplotlib.pyplot.show()

    te = [i for i in range(0, t_total, t)]
    i = 0
    while (i < l / 3):
```

```
        xfix = [temp[int(i / h)] for temp in res]
        matplotlib.pyplot.plot(te, xfix[:-1])
        i += 0.1

    matplotlib.pyplot.xlabel("t, cek")
    matplotlib.pyplot.ylabel("T, K")
    matplotlib.pyplot.grid(True)
    matplotlib.pyplot.show()
```

Ответы на вопросы

Приведите результаты тестирования программы. Учесть опыт выполнения лабораторной работы №3.

1.Пусть $F(t) = const = F_0 = 50$ Вт / см^2

Все параметры текущей задачи совпадают с параметрами задачи лабораторной работы №3 (вместо $k(t)$ используется $k(x)$), поэтому поле не меняется с течением времени и совпадает с температурным распределением $T(x)$ из лабораторной работы №3.

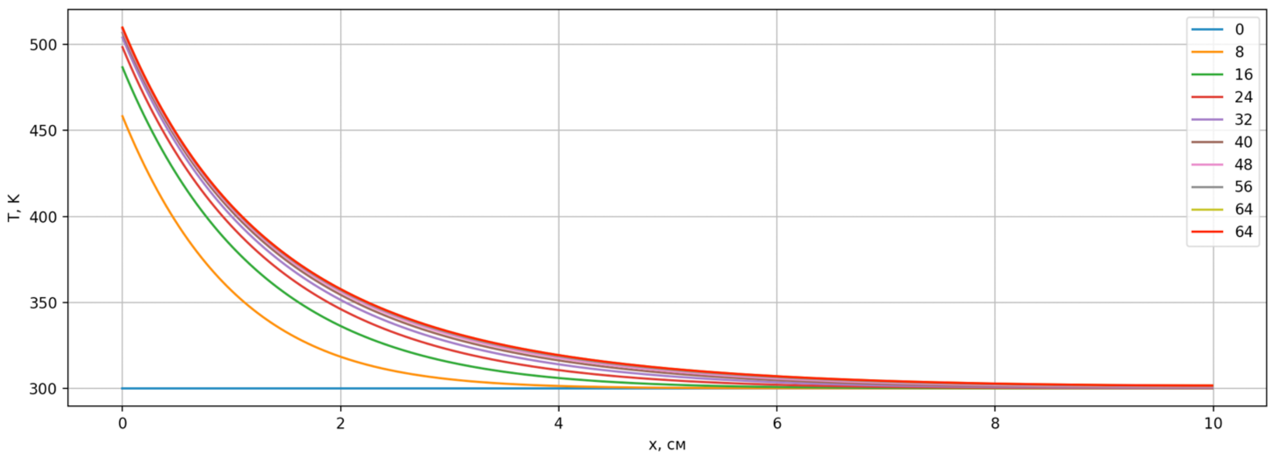

Рисунок 3. График зависимости $T(x, t_m)$ при параметрах 3 лабораторной работы.

2. Протестируем программу при отрицательных значениях F_0. Тепловой поток охлаждает стержень и чем дальше (ближе) находится участок стержня, тем меньшую (большую) температуру он будет иметь. При этом температура не должна быть ниже (выше) температуры окружающей среды.

Пусть $F(t) = F_0 = -10$ Вт / см^2.

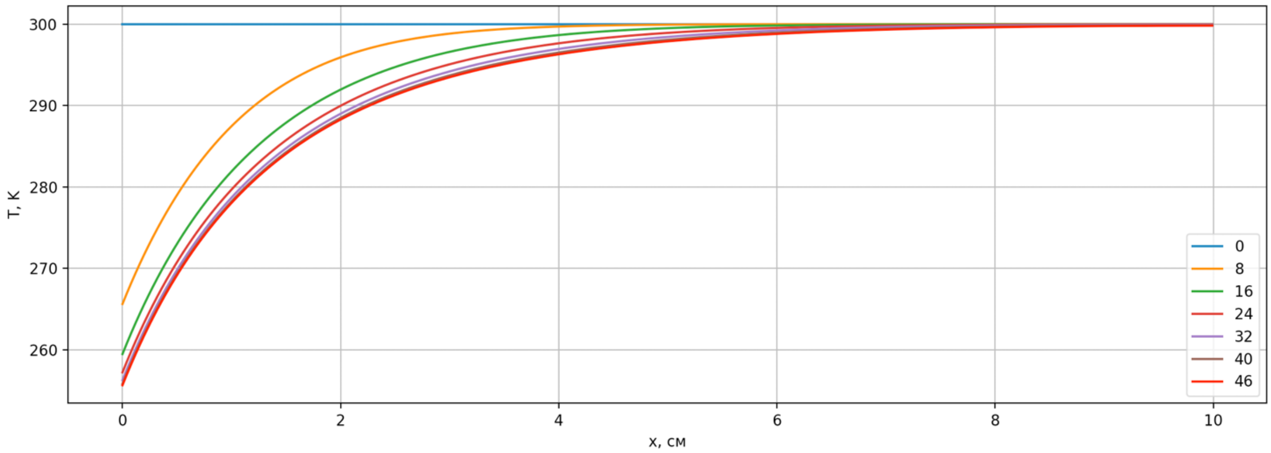

Рисунок 4. График зависимости $T(x,t_m)$ при $F_0 = -10$ Вт / см^2.

Чем дальше от теплового потока находится участок стержня, тем температура данного участка будет ближе к температуре окружающей среды.

3. Пусть коэффициент теплоотдачи увеличен в три раза $3\alpha(x)$.

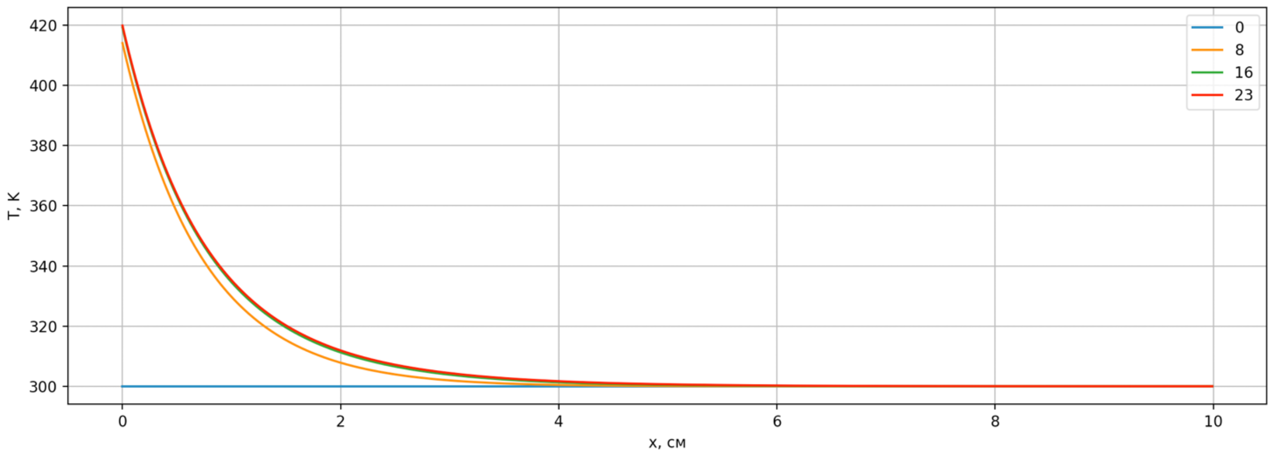

Рисунок 4. График зависимости $T(x,t_m)$ при $3\alpha(x)$.

Чем больше (меньше) значение теплоотдачи, тем больше (меньше) тепла отдает конкретный участок стержня, а значит тем меньше (больше) будет составлять его температура.

Выполните линеаризацию уравнения (14.8) по Ньютону, полагая для простоты, что все коэффициенты зависят только от одной переменной $\bar{y}_n$. Приведите линеаризованный вариант уравнения и опишите алгоритм его решения. Воспользуйтесь процедурой вывода, описанной в лекции №8.

По условию все коэффициенты зависят только от одной переменной $\widehat{y_n}$:

$\widehat{A_n} = \widehat{A_n}(\widehat{y_n})$, $\widehat{B_n} = \widehat{B_n}(\widehat{y_n})$, $\widehat{D_n} = \widehat{D_n}(\widehat{y_n})$, $\widehat{F_n} = \widehat{F_n}(\widehat{y_n})$.

Проведем линеаризацию по Ньютону по неизвестному $\widehat{y_n}$:

$$\left(\widehat{A_n}\widehat{y_{n-1}} - \widehat{B_n}\widehat{y_n} + \widehat{D_n}\widehat{y_{n+1}} + \widehat{F_n}\widehat{y_n}\right)|_{s-1} +$$
$$+\frac{\delta(\widehat{A_n}\widehat{y_{n-1}} - \widehat{B_n}\widehat{y_n} + \widehat{D_n}\widehat{y_{n+1}} + \widehat{F_n}\widehat{y_n})}{\delta\widehat{y_{n-1}}}|_{s-1}\Delta\widehat{y_{n-1}^s} +$$
$$+\frac{\delta(\widehat{A_n}\widehat{y_{n-1}} - \widehat{B_n}\widehat{y_n} + \widehat{D_n}\widehat{y_{n+1}} + \widehat{F_n}\widehat{y_n})}{\delta\widehat{y_n}}|_{s-1}\Delta\widehat{y_n^s} +$$
$$+\frac{\delta(\widehat{A_n}\widehat{y_{n-1}} - \widehat{B_n}\widehat{y_n} + \widehat{D_n}\widehat{y_{n+1}} + \widehat{F_n}\widehat{y_n})}{\delta\widehat{y_{n+1}}}|_{s-1}\Delta\widehat{y_{n+1}^s} = 0$$

Получим:

$$\left(\widehat{A_n}\widehat{y_{n-1}} - \widehat{B_n}\widehat{y_n} + \widehat{D_n}\widehat{y_{n+1}} + \widehat{F_n}\right)|_{s-1} + \widehat{A_n}|_{s-1}\Delta\widehat{y_{n-1}^s} +$$
$$+\left(\frac{\delta\widehat{A_n}}{\delta\widehat{y_n}}\widehat{y_{n-1}} - \frac{\delta\widehat{B_n}}{\delta\widehat{y_n}}\widehat{y_n} - \widehat{B_n} + \frac{\delta\widehat{D_n}}{\delta\widehat{y_n}}\widehat{y_{n+1}} + \frac{\delta\widehat{F_n}}{\delta\widehat{y_n}}\right)|_{s-1}\Delta\widehat{y_n^s} + \widehat{D_n}|_{s-1}\Delta\widehat{y_{n+1}^s} = 0$$

Это уравнение решается методом прогонки, в результате чего находятся $\Delta\widehat{y_n^s}$ а затем находятся значения искомой функции в узлах на s-итерации $\widehat{y_n^s} = \widehat{y_n^{s-1}} + \Delta\widehat{y_n^s}$. Итерационный процесс заканчивается при выполнении условия $\max\left|\frac{\Delta\widehat{y_n^s}}{\widehat{y_n^s}}\right| \leq \varepsilon$, для всех $n = 0, 1, \ldots N$.

Заключение

В ходе лабораторной работы были получены навыки по разработке алгоритмов решения краевой задачи при реализации моделей на основе ОДУ в частных производных с краевыми условиями I и III рода.