

Санкт-Петербургский государственный политехнический университет
Институт компьютерных наук и кибербезопасности
«Высшая школа программной инженерии»

Лабораторная работа 2

по дисциплине «Разработка программного обеспечения для
моделирования физических процессов»

Выполнил

студент
гр.5130904/10101



Абраамян А. М.

Руководитель

Воскобойников С. П.

«___» _____ 202__ г.

Санкт-Петербург
2024

Оглавление

Оглавление

Постановка задачи.....	3
Дискретная модель.....	3
Коэффициенты.....	5
Решение системы ОДУ.....	6
Явный метод ломаных Эйлера.....	6
Неявный метод ломаных Эйлера.....	7
Метод неявного Эйлера с тридиагональной матрицей.....	7
Обзор метода неявного Эйлера.....	7
Тридиагональная матрица.....	8
Метод Томаса.....	8
Метод Томаса для неявного метода Эйлера.....	9
Жёсткость системы.....	10
Тестирование.....	10
Пример 1.....	10
Пример 2.....	12
Вывод.....	13
Код.....	13

Постановка задачи

Используя интегро-интерполяционный метод (метод баланса), разработать программу для моделирования нестационарного распределения температуры в полом цилиндра, описываемого математической моделью вида:

$$\frac{\partial u}{\partial t} = \frac{1}{r} \frac{\partial}{\partial r} \left(rk(r, t) \frac{\partial u}{\partial r} \right) - q(r, t)u + f(r, t), \quad r \in [R_L, R_R], \quad R_L > 0, \quad t \in [0, T]$$

$$0 < c_1 \leq k(r, t) \leq c_2, \quad 0 \leq q(r, t)$$

начальным условием вида $u|_{t=0} = \varphi(r)$ и граничными условиями вида:

$$u|_{r=R_L} = v_1(t) \quad -k \frac{\partial u}{\partial r} \Big|_{r=R_R} = \chi_2 u|_{r=R_R} - v_2(t)$$

$$\chi_2 \geq 0$$

Для построения и тестирования модели будет использоваться язык C++.

Дискретная модель

Введём обозначения:

N - число разбиений интервала $[R_l, R_r]$

$$h_i = r_i - r_{i-1}$$

$$r_{i-1/2} = (r_i + r_{i-1})/2$$

$$\bar{h}_i = \begin{cases} \frac{h_i + 1}{2}, & i = 0 \\ \frac{h_i + h_{i+1}}{2}, & i = 1, 2, \dots, N-1 \\ \frac{h_i}{2}, & i = N \end{cases}$$

Домножим уравнение на r :

$$r \frac{\partial u}{\partial t} = \frac{\partial}{\partial r} \left(rk \frac{\partial u}{\partial r} \right) - rqu + rf$$

Проинтегрируем уравнение для промежутка, не включая границы:

$$\int_{r_{i-0.5}}^{r_{i+0.5}} r \frac{\partial u}{\partial t} dt = \int_{r_{i-0.5}}^{r_{i+0.5}} \left(\frac{\partial}{\partial r} \left(rk \frac{\partial u}{\partial r} \right) \right) dr - \int_{r_{i-0.5}}^{r_{i+0.5}} r q u dr + \int_{r_{i-0.5}}^{r_{i+0.5}} r f dr \quad i = 1, 2, \dots, N-1$$

Интегро-дифференциальное тождество:

$$\int_{r_{i-0.5}}^{r_{i+0.5}} r \frac{\partial u}{\partial t} dt = rk \frac{\partial u}{\partial r} \Big|_{r=r_{i+0.5}} - rk \frac{\partial u}{\partial r} \Big|_{r=r_{i-0.5}} - \int_{r_{i-0.5}}^{r_{i+0.5}} r q u dr + \int_{r_{i-0.5}}^{r_{i+0.5}} r f dr$$

По формуле центральных разностей:

$$rk \frac{\partial u}{\partial r} \Big|_{r=r_{i-0.5}} \approx r_{i-0.5} k_{i-0.5} \frac{v_i - v_{i-1}}{h_i}$$

$$rk \frac{\partial u}{\partial r} \Big|_{r=r_{i+0.5}} \approx r_{i+0.5} k_{i+0.5} \frac{v_{i+1} - v_i}{h_{i+1}}$$

По формуле **средних** прямоугольников:

$$\int_{r_{i-0.5}}^{r_{i+0.5}} \varphi(r) dr \approx h_i \varphi_i$$

Разностная схема:

$$\frac{dv_i}{dt} = r_{i+0.5} k_{i+0.5} \frac{v_{i+1} - v_i}{h_i r_i h_{i+1}} - r_{i-0.5} k_{i-0.5} \frac{v_i - v_{i-1}}{h_i r_i h_i} - q_i v_i + f_i \quad i = 1, 2, \dots, N-1$$

Аппроксимация граничного условия справа:

$$\int_{r_{i-0.5}}^{r_i} r \frac{\partial u}{\partial t} dt = \int_{r_{i-0.5}}^{r_i} \left(\frac{\partial}{\partial r} \left(rk \frac{\partial u}{\partial r} \right) \right) dr - \int_{r_{i-0.5}}^{r_i} r q u dr + \int_{r_{i-0.5}}^{r_i} r f dr \quad i = N$$

Теперь используем формулу **правых** прямоугольников для аппроксимации интеграла:

$$\int_{r_{i-0.5}}^{r_i} \varphi(r) dr \approx h_i \varphi_i$$

$$h_i r_i \frac{dv_i}{dt} = rk \frac{\partial u}{\partial r} \Big|_{r=r_i} - rk \frac{\partial u}{\partial r} \Big|_{r=r_{i-0.5}} - h_i r_i q_i v_i + h_i r_i f_i$$

Используя наше граничное условие справа:

$$\frac{dv_i}{dt} = - \left(\frac{\chi_2 v_i - v_2}{h_i} \right) - r_{i-0.5} k_{i-0.5} \frac{v_i - v_{i-1}}{h_i r_i h_i} - q_i v_i + f_i \quad i = N$$

Приведём подобные слагаемые для разностных схем:

$$\frac{dv_i}{dt} = v_{i-1} \left(\frac{r_{i-0.5} k_{i-0.5}}{h_i r_i h_{i+1}} \right) + v_i \left(\frac{-r_{i-0.5} k_{i-0.5}}{h_i r_i h_{i+1}} - \frac{r_{i+0.5} k_{i+0.5}}{h_i r_i h_{i+1}} - q_i \right) + v_{i+1} \left(\frac{r_{i+0.5} k_{i+0.5}}{h_i r_i h_{i+1}} \right) + f_i$$

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

$$\frac{dv_i}{dt} = v_{i-1} \left(\frac{r_{i-0.5} k_{i-0.5}}{h_i r_i h_{i+1}} \right) + v_i \left(\frac{-r_{i-0.5} k_{i-0.5}}{h_i r_i h_{i+1}} - \frac{\chi_2}{h_i} - q_i \right) + \frac{\gamma_2}{h_i} + f_i \quad i = N$$

Поскольку нам дано граничное условие слева первого рода, то мы поступим следующим образом. Используем его для уравнение $i = 1$. Буквально возьмем и подставим v_0 вместо v_{i-1} . Таким образом наш коэффициент просто перейдет a просто перейдет в коэффициент g как константа, а матрица станет на одну строчку короче чем была прежде.

Коэффициенты

Теперь данную систему можно представить в виде:

$$\frac{dv}{dt} = Av + g \quad (1)$$

Где A - трёхдиагональная матрица вида:

$$\begin{pmatrix} c_1 & b_1 & & & & & 0 \\ a_2 & c_2 & b_2 & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \ddots & \ddots & \ddots \\ & & & & & a_{n-1} & c_{n-1} & b_{n-1} \\ 0 & & & & & & a_n & c_n \end{pmatrix}$$

Коэффициенты соответственно следующие:

Для $i=0$ коэффициентов у нас не будет.

$$\begin{aligned}
b_i &= \frac{r_{i+0.5} k_{i+0.5}}{\hbar_i r_i h_{i+1}} & c_i &= \frac{-r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}} - \frac{r_{i+0.5} k_{i+0.5}}{\hbar_i r_i h_{i+1}} - q_i \\
g_i &= f_i + \frac{r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}} * v_{i-1} & i &= 1 \\
a_i &= \frac{r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}} & b_i &= \frac{r_{i+0.5} k_{i+0.5}}{\hbar_i r_i h_{i+1}} \\
c_i &= \frac{-r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}} - \frac{r_{i+0.5} k_{i+0.5}}{\hbar_i r_i h_{i+1}} - q_i & g_i &= f_i \quad i=2, \dots, N-1 \\
a_i &= \frac{r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}} & g_i &= \frac{y_2}{\hbar_i} + f_i \quad c_i = \frac{-r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}} - \frac{\chi_2}{\hbar_i} - q_i \quad i=N
\end{aligned}$$

Решить систему численным методом - найти вектор v и сравнить его с точным решением u .

Решение системы ОДУ

Чтобы решить систему дифференциальных уравнений, введём дискретизацию по времени и проинтегрируем:

$$\begin{cases} \int_{t_n}^{t_{n+1}} \frac{\partial v_i}{\partial t} dt = \int_{t_n}^{t_{n+1}} (A(t_n) v + g(t_n)) dt \\ v(t_{n+1}) = v(t_n) + \int_{t_n}^{t_{n+1}} (A(t_n) v + g(t_n)) dt \end{cases}$$

Добавим начальное условие и получим систему:

$$\begin{cases} v(t_{n+1}) = v(t_n) + \int_{t_n}^{t_{n+1}} (A(t_n) v + g(t_n)) dt \\ v(t_0) = \varphi(r) \end{cases}$$

Явный метод ломаных Эйлера

Аппроксимируем интеграл по формуле левых прямоугольников

$\int_a^b f(x) dx \approx f(a)(b-a)$, получаем:

$$\begin{aligned}
H &= (t_{n+1} - t_n) \\
v(t_{n+1}) &= v(t_n) + H A(t_n) v(t_n) + H g(t_n) \\
v(t_{n+1}) &= (E + H A(t_n)) v(t_n) + H g(t_n)
\end{aligned}$$

$$v(t_0) = \varphi(r)$$

Неявный метод ломаных Эйлера

Аппроксимируем интеграл по формуле правых прямоугольников

$$\int_a^b f(x) dx \approx f(b)(b-a), \text{ получаем:}$$

$$\left\{ \begin{array}{l} H = (t_{n+1} - t_n) \\ v(t_{n+1}) = (E - HA(t_{n+1}))^{-1} (v(t_n) + Hg(t_{n+1})) \\ v(t_0) = \varphi(r) \end{array} \right.$$

Это привычная запись неявного метода Эйлера. Однако надо понимать, что вычисление обратной матрицы очень трудоемкая операция, которую можно легко избежать зная что матрица A трехдиагональная

Метод неявного Эйлера с тридиагональной матрицей

Метод неявного Эйлера (или метод "Backward Euler") используется для решения дифференциальных уравнений, особенно жестких систем. Это неявный метод, который требует решения системы линейных уравнений для получения решения на каждом шаге.

Обзор метода неявного Эйлера

Для дифференциального уравнения вида:

$$\frac{du}{dt} = Au + b$$

где:

- $u(t)$ — вектор состояния системы в момент времени t ,
- A — матрица, описывающая систему,
- b — вектор внешних сил или источников.

Метод неявного Эйлера обновляет решение на каждом шаге по формуле:

$$u(t+\Delta t) = u(t) + \Delta t \cdot (Au(t+\Delta t) + b(t+\Delta t))$$

Переписываем это уравнение, чтобы выразить решение для следующего шага:

$$(I - \Delta t A)u(t + \Delta t) = u(t) + \Delta t b(t + \Delta t)$$

где:

- I — единичная матрица,
- Δt — шаг по времени.

Получается система линейных уравнений, которую нужно решить для вектора $u(t + \Delta t)$.

Тридиагональная матрица

Если матрица A является тридиагональной, это означает, что все элементы матрицы вне главной диагонали, а также двух соседних диагоналей (над и под главной), равны нулю. Тридиагональная матрица имеет вид:

$$A = \begin{bmatrix} b_1 & c_1 & 0 & 0 & \cdots & 0 \\ a_2 & b_2 & c_2 & 0 & \cdots & 0 \\ 0 & a_2 & b_2 & c_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & b_n \end{bmatrix}$$

Система линейных уравнений для метода неявного Эйлера становится:

$$(I - \Delta t A)u(t + \Delta t) = u(t) + \Delta t b(t + \Delta t)$$

Это линейная система с тридиагональной матрицей, которую можно решить с использованием метода **Томаса**.

Метод Томаса

Метод Томаса — это специализированная форма метода Гаусса, оптимизированная для тридиагональных матриц. Система линейных уравнений для тридиагональной матрицы A имеет вид:

$$Ax = d$$

где A — тридиагональная матрица, x — вектор неизвестных, и d — вектор правых частей.

Система уравнений:

$$b_1 * x_1 + c_1 * x_2 = d_1$$

$$a_2 * x_1 + b_2 * x_2 + c_2 * x_3 = d_2$$

...

$$a_n * x_{n-1} + b_n * x_n = d_n$$

Шаги метода Томаса

1. Прямой ход (Forward Elimination)

Необходимо модифицировать матрицу системы, чтобы избавиться от поддиагональных элементов:

$$\begin{aligned}\hat{b}_i &= b_i - a_{i-1} \cdot \hat{c}_{i-1} \\ \hat{d}_i &= d_i - a_{i-1} \cdot \hat{d}_{i-1}\end{aligned}$$

где b_i — это модифицированные элементы главной диагонали, а d_i — обновленные элементы вектора правых частей.

2. Обратный ход (Back Substitution)

После того как система станет верхней треугольной, мы можем решить её методом обратного хода:

$$\begin{aligned}x_n &= \frac{\hat{d}_n}{\hat{b}_n} \\ x_{n-1} &= \frac{\hat{d}_{n-1} - c_{n-1} \cdot x_n}{\hat{b}_{n-1}} \\ &\vdots \\ x_1 &= \frac{\hat{d}_1}{\hat{b}_1}\end{aligned}$$

Метод Томаса для неявного метода Эйлера

Когда матрица A тридиагональная, для решения системы линейных уравнений для метода неявного Эйлера, мы используем алгоритм Томаса. Шаги следующие:

1. Формируем тридиагональную матрицу $A_{new} = I - \Delta t A$.

2. Решаем систему $A_{new}u(t+\Delta t)=u(t)+\Delta tb(t+\Delta t)$ с использованием метода Томаса.

Жёсткость системы

При решении явным методом нужно учитывать, что система может быть жёсткой. В этом случае накладывается ограничение на шаг интегрирования τ . Это ограничение зависит от обусловленности матрицы, которое зависит от шага разбиения интервала по r .

$$\tau < \frac{2}{\max_i |\lambda_i|}$$

$$\max_i |\lambda_i| < \|A\| \sim \frac{1}{h^2}$$

$$\tau < h^2, \quad \tau < \frac{2}{\|A\|}$$

При превышении этого ограничения погрешность станет накапливаться с очень большой скоростью (решение неустойчиво).

Тестирование

Пример 1

Первый тест будет выглядеть как на рисунке. Предварительно ожидается что компьютер без проблем справится с поставленной задачей. Зададим $T=1$.

$$\begin{aligned}
 K &= 1 \\
 q &= 3 \\
 u &= t + 2z \\
 R_1 &= 1 \\
 R_2 &= 10 \\
 \bar{u}_1 &= 2 + t \\
 x_2 &= 3 \\
 \bar{u}_2 &= 62 + 3t \\
 \varphi &= 2 \cdot z
 \end{aligned}$$

$$\frac{\partial u}{\partial t} = \frac{1}{z} \frac{\partial}{\partial z} \left(z \cdot k \cdot \frac{\partial u}{\partial z} \right) - q u + \phi, \quad u|_{z=R_1} = \bar{u}_1(t)$$

$$-k \frac{\partial u}{\partial z} \Big|_{z=R_2} = x_2 u|_{z=R_2} - \bar{u}_2(t)$$

$$\begin{aligned}
 \bar{u}_2 \\
 -1 \cdot 2 &= 3 \cdot (t + 2 \cdot 10) - \bar{u}_2 \\
 \bar{u}_2 &= 2 + 60 + 3t \\
 \bar{u}_2 &= 62 + 3t
 \end{aligned}$$

$$\begin{aligned}
 \phi \\
 \phi &= \frac{1}{z} 2 - 3(t + 2z) + \phi \\
 \phi &= \frac{2}{z} - 3t - 6z + \phi \\
 \phi &= 3t + 6z - \frac{2}{z} + \phi
 \end{aligned}$$

В результате программы получаем следующие столбцы:

Method Name	Название метода
R Steps Count	Количество разбиений по R
R Step Size	Получившийся шаг по R
T Steps Count	Количество разбиений по T
T Step Size	Получившийся шаг по T
Sum Inaccuracy	Суммарная ошибка
Max Inaccuracy	Максимальная ошибка
Sum Inaccuracy (2nd Column)	Суммарная погрешность второй колонки
Max Inaccuracy (2nd Column)	Максимальная погрешность второй колонки
Sum Inaccuracy (Middle Col)	Суммарная погрешность средней колонки
Max Inaccuracy (Middle Col)	Максимальная погрешность средней колонки
Sum Inaccuracy (Last Col)	Суммарная погрешность последней колонки
Max Inaccuracy (Last Col)	Максимальная погрешность последней колонки
Main matrix build time (us)	Среднее время построения матрицы
Integration time (us)	Среднее время интегрирования ОДНОГО ШАГА

Второй тест выглядит несколько серьезнее первого, и забегаая наперед скажу, что программа не смогла уложиться в разумные значения погрешностей.

						10.0001		10.0002		10.0003		10.0004		10.0005		10.0006		10.0007		10.0008		10.0009		10.0010		10.0011		10.0012		10.0013		10.0014		10.0015		10.0016		10.0017		10.0018		10.0019		10.0020		10.0021		10.0022		10.0023		10.0024		10.0025		10.0026		10.0027		10.0028		10.0029		10.0030		10.0031		10.0032		10.0033		10.0034		10.0035		10.0036		10.0037		10.0038		10.0039		10.0040		10.0041		10.0042		10.0043		10.0044		10.0045		10.0046		10.0047		10.0048		10.0049		10.0050		10.0051		10.0052		10.0053		10.0054		10.0055		10.0056		10.0057		10.0058		10.0059		10.0060		10.0061		10.0062		10.0063		10.0064		10.0065		10.0066		10.0067		10.0068		10.0069		10.0070		10.0071		10.0072		10.0073		10.0074		10.0075		10.0076		10.0077		10.0078		10.0079		10.0080		10.0081		10.0082		10.0083		10.0084		10.0085		10.0086		10.0087		10.0088		10.0089		10.0090		10.0091		10.0092		10.0093		10.0094		10.0095		10.0096		10.0097		10.0098		10.0099		10.0100		10.0101		10.0102		10.0103		10.0104		10.0105		10.0106		10.0107		10.0108		10.0109		10.0110		10.0111		10.0112		10.0113		10.0114		10.0115		10.0116		10.0117		10.0118		10.0119		10.0120		10.0121		10.0122		10.0123		10.0124		10.0125		10.0126		10.0127		10.0128		10.0129		10.0130		10.0131		10.0132		10.0133		10.0134		10.0135		10.0136		10.0137		10.0138		10.0139		10.0140		10.0141		10.0142		10.0143		10.0144		10.0145		10.0146		10.0147		10.0148		10.0149		10.0150		10.0151		10.0152		10.0153		10.0154		10.0155		10.0156		10.0157		10.0158		10.0159		10.0160		10.0161		10.0162		10.0163		10.0164		10.0165		10.0166		10.0167		10.0168		10.0169		10.0170		10.0171		10.0172		10.0173		10.0174		10.0175		10.0176		10.0177		10.0178		10.0179		10.0180		10.0181		10.0182		10.0183		10.0184		10.0185		10.0186		10.0187		10.0188		10.0189		10.0190		10.0191		10.0192		10.0193		10.0194		10.0195		10.0196		10.0197		10.0198		10.0199		10.0200		10.0201		10.0202		10.0203		10.0204		10.0205		10.0206		10.0207		10.0208		10.0209		10.0210		10.0211		10.0212		10.0213		10.0214		10.0215		10.0216		10.0217		10.0218		10.0219		10.0220		10.0221		10.0222		10.0223		10.0224		10.0225		10.0226		10.0227		10.0228		10.0229		10.0230		10.0231		10.0232		10.0233		10.0234		10.0235		10.0236		10.0237		10.0238		10.0239		10.0240		10.0241		10.0242		10.0243		10.0244		10.0245		10.0246		10.0247		10.0248		10.0249		10.0250		10.0251		10.0252		10.0253		10.0254	
Factor Region	5	1.0	0.001	137938464264778813777877																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																													

Лучшее чего удалось добиться — это уменьшение погрешности до значения 1.5 с помощью неявного метода Эйлера при разбиении **R** интервала на 10 частей и интервала **T** на 10000 частей.

В целом такая точность может быть допустимой если нужны лишь приблизительные вычисления.

Целиком с таблицей можно ознакомиться по ссылке:

<https://github.com/Hryapusek/rust-tridiagonal-matrix-vector/blob/cp3-try-correct-version/lab2/result2.txt>

Вывод

Освоили необычный способ численного решения дифференциального уравнения второго порядка с различными граничными условиями, увидели зависимости результата от различных комбинаций настраиваемых параметров и освоили неявный метод Эйлера в комбинации с трёхдиагональным методом.

Исходный код доступен по ссылке: <https://github.com/Hryapusek/rust-tridiagonal-matrix-vector/tree/cp3-try-correct-version>

Код

```
#pragma once

#include <interface/i_euler_explicit_method.hpp>

#include <Eigen/Dense>

class DefaultEulerExplicitMethod : public IEulerExplicitMethod
{
public:
    auto integrate(
        Eigen::VectorX<Number_t> const& start_v,
        Eigen::MatrixX<Number_t> const& A,
        Eigen::SparseVector<Number_t> const& g,
        std::vector<Number_t> const& points
    ) -> Eigen::SparseMatrix<Number_t> override;

    auto name() -> std::string override { return "Euler Explicit"; }
};
```

```

#pragma once

#include <interface/i_euler_implicit_method.hpp>
#include <Eigen/Dense>

class DefaultEulerImplicitMethod : public IEulerImplicitMethod
{
public:
    auto integrate(
        Eigen::VectorX<Number_t> const& start v,
        Eigen::MatrixX<Number_t> const& A,
        Eigen::SparseVector<Number_t> const& g,
        std::vector<Number_t> const& points
    ) -> Eigen::SparseMatrix<Number_t> override;

    auto name() -> std::string override { return "Euler Implicit"; }
};

#pragma once

#include <memory>
#include <Eigen/Dense>
#include <interface/i_main_matrix_calculator.hpp>
#include <input_parameters.hpp>

class DefaultMainMatrixCalculator : public IMainMatrixCalculator
{
public:
    explicit DefaultMainMatrixCalculator(
        std::shared_ptr<InputParameters> params,
        std::vector<Number_t> r_points
    )
    : params_(params)
    , r_points_(std::move(r_points))
    {}

    auto calc_a(size_t r_index, Number_t t) const -> Number_t override;
    auto calc_b(size_t r_index, Number_t t) const -> Number_t override;
    auto calc_c(size_t r_index, Number_t t) const -> Number_t override;
    auto calc_g(size_t r_index, Number_t t) const -> Number_t override;

    auto r_points() const -> std::vector<Number_t> const& { return r_points ; }

    auto params() const -> std::shared_ptr<InputParameters> const& { return params_ ; }

protected:
    std::vector<Number_t> r_points_ ;
    std::shared_ptr<InputParameters> params_ ;
};

#pragma once

#include <interface/i_base_integrate.hpp>
#include <Eigen/Dense>

class RKFMethod : public IBaseIntegrate
{
public:
    auto integrate(
        Eigen::VectorX<Number_t> const& start v,
        Eigen::MatrixX<Number_t> const& A,
        Eigen::SparseVector<Number_t> const& g,
        std::vector<Number_t> const& points
    ) -> Eigen::SparseMatrix<Number_t> override;

    auto name() -> std::string override { return "RKF Method"; }
};

```

```

#pragma once

#include <string>
#include <defines.hpp>
#include <Eigen/Sparse>

class IBaseIntegrate
{
public:
    virtual auto integrate(
        Eigen::VectorX<Number_t> const& start v,
        Eigen::MatrixX<Number_t> const& A,
        Eigen::SparseVector<Number_t> const& g,
        std::vector<Number_t> const& points
    ) -> Eigen::SparseMatrix<Number_t> = 0;

    virtual auto name() -> std::string { return "Unknown"; }
};

#pragma once

#include <interface/i_base_integrate.hpp>

class IEulerExplicitMethod : public IBaseIntegrate
{};

#pragma once

#include <interface/i_base_integrate.hpp>

class IEulerImplicitMethod : public IBaseIntegrate
{};

#pragma once

#include <cstdio>

using Number_t = double;

class IMainMatrixCalculator
{
public:
    using Number_t = double;
    virtual auto calc_a(size_t r_index, Number_t t) const -> Number_t = 0;
    virtual auto calc_b(size_t r_index, Number_t t) const -> Number_t = 0;
    virtual auto calc_c(size_t r_index, Number_t t) const -> Number_t = 0;
    virtual auto calc_g(size_t r_index, Number_t t) const -> Number_t = 0;
};

#pragma once

#include <functional>

using Number_t = double;

// First argument is r, second is t
using R_T_Function_type = std::function<double(double, double)>;

using T_Function_type = std::function<double(double)>;
using R_Function_type = std::function<double(double)>;

#pragma once

#include <defines.hpp>

struct InputParameters {
    Number_t Rl;
    Number_t Rr; // [Rl, Rr]
    Number_t T; // [0, T]
};

```

```

// First type condition
T_Function_type v1; // u(rL) = v1(t)

// Third type condition
Number_t hi2; // -k * du/dr = hi2 * u(rR) - v2(t)
R_Function_type phi;
T_Function_type v2;

// Just input functions
R_T_Function_type k;
R_T_Function_type q;
R_T_Function_type f;
};

#include <vector>
#include <cstdio>

#include <contract/contract.hpp>

#include <defines.hpp>

auto split_interval(const Number_t& left, const Number_t& right, size_t num_intervals) ->
std::vector<Number_t>;

// Calculate the length of an interval `index-1` to `index`
auto calc_h(const std::vector<Number_t>& intervals, size_t index) -> Number_t;

// Calculate the cross h of an interval
auto calc_cross_h(const std::vector<Number_t>& intervals, size_t index) -> Number_t;

/// @return middle point between `index` and `index - 1`
auto middle_point(const std::vector<Number_t>& intervals, size_t index) -> Number_t;

#include <default_impl/euler_explicit_method.hpp>

#include <Eigen/Dense>

#include <contract/contract.hpp>

auto DefaultEulerExplicitMethod::integrate(
Eigen::VectorX<Number_t> const& start_v,
Eigen::MatrixX<Number_t> const& A,
Eigen::SparseVector<Number_t> const& g,
std::vector<Number_t> const& points
) -> Eigen::SparseMatrix<Number_t>
{
    auto result = Eigen::MatrixX<Number_t>(
A.rows(),
points.size()
); // Adjust columns based on intervals size

    contract(fun)
    {
        precondition(A.rows() == A.cols(), "A must be a square matrix");
        precondition(A.rows() == start_v.rows(), "A and start_v must have the same number of rows");
        precondition(A.rows() == g.rows(), "A and g must have the same number of rows");
        postcondition(
            result.cols() == points.size(),
            "result must have the same number of columns as intervals"
        );
    };

    result.col(0) = start_v.sparseView();

    auto E = Eigen::SparseMatrix<Number_t>(A.rows(), A.rows());
    E.setIdentity();
    auto H = points.at(1) - points.at(0); // Step size
    for(size_t i = 1; i < 2; ++i) { // Loop over intervals, not A.cols()
        // for(size_t i = 1; i < points.size(); ++i) {
        result.col(i) = (E + H * A) * result.col(i - 1) + H * g; // Euler update
    }
}

```



```

return result.sparseView();
}

#include <default_impl/euler_implicit_method.hpp>

#include <vector>
#include <stdexcept>

#include <contract/contract.hpp>

#include <Eigen/Dense>
#include <Eigen/Sparse>

// Helper function to solve a tridiagonal system using the Thomas algorithm
void solve_tridiagonal(
    const Eigen::VectorXd& a, // Lower diagonal (size n-1)
    const Eigen::VectorXd& b, // Main diagonal (size n)
    const Eigen::VectorXd& c, // Upper diagonal (size n-1)
    const Eigen::VectorXd& d, // Right-hand side (size n)
    Eigen::VectorXd& x // Solution (size n)
)
{
    size_t n = b.size();
    Eigen::VectorXd cp(n); // Temporary vector for forward substitution
    Eigen::VectorXd dp(n); // Temporary vector for the right-hand side adjustment

    // Forward elimination
    cp(0) = c(0) / b(0);
    dp(0) = d(0) / b(0);

    for (size_t i = 1; i < n - 1; ++i) { // Loop from 1 to n-2 for the main part
        double m = 1.0 / (b(i) - a(i - 1) * cp(i - 1)); // a(i-1) for lower diagonal
        cp(i) = c(i) * m;
        dp(i) = (d(i) - a(i - 1) * dp(i - 1)) * m;
    }

    // Last step for the forward elimination
    dp(n - 1) = (d(n - 1) - a(n - 2) * dp(n - 2)) / (b(n - 1) - a(n - 2) * cp(n - 2));

    // Backward substitution
    x(n - 1) = dp(n - 1);
    for (size_t i = n - 2; i != static_cast<size_t>(-1); --i) {
        x(i) = dp(i) - cp(i) * x(i + 1);
    }
}

auto DefaultEulerImplicitMethod::integrate(
    Eigen::VectorX<Number_t> const& start_v,
    Eigen::MatrixX<Number_t> const& A,
    Eigen::SparseVector<Number_t> const& g,
    std::vector<Number_t> const& points
) -> Eigen::SparseMatrix<Number_t>
{
    Eigen::MatrixX<Number_t> result(
        A.rows(), points.size()); // Adjust the columns to match intervals

    contract(fun)
    {
        precondition(A.rows() == A.cols(), "A must be a square matrix");
        precondition(A.rows() == start_v.rows(), "A and start_v must have the same number of rows");
        precondition(A.rows() == g.rows(), "A and g must have the same number of rows");
        postcondition(
            result.cols() == points.size(),
            "result must have the same number of columns as intervals"
        );
    };

    result.col(0) = start_v.sparseView();

    auto E = Eigen::SparseMatrix<Number_t>(A.rows(), A.rows());
    E.setIdentity();

```

```

for (size_t i = 1; i < points.size(); ++i) { // Iterate over intervals
    auto H = points.at(i) - points.at(i - 1); // Step size
    Eigen::SparseMatrix<Number_t> M = E - H * A; // Matrix for the implicit step

    // Prepare the right-hand side of the equation
    Eigen::SparseVector<Number_t> rhs = result.col(i - 1) + H * g;

    // We will now extract the tridiagonal parts of M to solve the system
    size_t n = M.rows();
    // Extract diagonals from sparse matrix M
    Eigen::VectorXd a(n - 1); // Lower diagonal
    Eigen::VectorXd b(n); // Main diagonal
    Eigen::VectorXd c(n - 1); // Upper diagonal

    // Fill diagonals by iterating through the sparse matrix
    for (int k = 0; k < n; ++k) {
        b(k) = M.coeff(k, k); // Main diagonal
        if (k > 0) {
            a(k - 1) = M.coeff(k, k - 1); // Lower diagonal
        }
        if (k < n - 1) {
            c(k) = M.coeff(k, k + 1); // Upper diagonal
        }
    }

    // Prepare the right-hand side (rhs) vector
    Eigen::VectorXd rhs_full(n);
    for (size_t k = 0; k < n; ++k) {
        rhs_full(k) = rhs.coeff(k); // Copy rhs values
    }

    // Now solve the tridiagonal system
    Eigen::VectorXd solution(n);
    solve_tridiagonal(a, b, c, rhs_full, solution);

    // Update the result for this timestep
    result.col(i) = solution.sparseView();
}

return result.sparseView();
}

#include <default_impl/main_matrix_calculator.hpp>

#include <cassert>

#include <contract/contract.hpp>

#include <interval_splitter.hpp>

auto DefaultMainMatrixCalculator::calc_a(size_t r_index, Number_t t) const -> Number_t
{
    contract(fun)
    {
        precondition(
            r_index != 0,
            "You should not calculate anything for index == 0 - you already have v function"
        );
        precondition(r_index != 1, "You should not calculate a for index == 1");
        precondition(r_index < r_points.size(), "index out of range");
    };
    auto mid_point = middle_point(r_points_, r_index);
    auto k_value = params_->k(mid_point, t);
    auto h_value = calc_h(r_points_, r_index);

    return mid_point * k_value / h_value;
}

auto DefaultMainMatrixCalculator::calc_b(size_t r_index, Number_t t) const -> Number_t
{
    contract(fun)
    {

```

```

precondition(
  r_index != 0,
  "You should not calculate anything for index == 0 - you already have v function"
);
precondition(
  r_index != r_points_.size() - 1,
  "You should not calculate b for index == intervals_.size() - 1"
);
precondition(r_index < r_points_.size(), "index out of range");
};
auto up = middle_point(r_points_, r_index + 1)
* params_>k(middle_point(r_points_, r_index + 1), t);
auto down = calc_h(r_points_, r_index + 1);
return up / down;
}

/* clang-format off */
auto DefaultMainMatrixCalculator::calc_c(size_t r_index, Number_t t) const -> Number_t
{
  contract(fun) {
    precondition(r_index != 0, "You should not calculate anything for index == 0 - you already have v
function");
    precondition(r_index < r_points_.size(), "index out of range");
  };
  if (r_index == r_points_.size() - 1) {
    return - middle_point(r_points_, r_index) * params_>k(middle_point(r_points_, r_index), t)
/ calc_h(r_points_, r_index)
- params_>hi2
- params_>q(r_points_[r_index], t)
* calc_cross_h(r_points_, r_index);
  } else {
    auto m_point_index = middle_point(r_points_, r_index);
    auto m_point_index_plus_1 = middle_point(r_points_, r_index + 1);
    auto k_index = params_>k(m_point_index, t);
    auto k_index_plus_1 = params_>k(m_point_index_plus_1, t);
    auto h_index_plus_1 = calc_h(r_points_, r_index + 1);
    auto cross_h_index = calc_cross_h(r_points_, r_index);
    auto r = r_points_[r_index];
    auto q_index = params_>q(r_index, t);

    return -m_point_index * k_index / h_index_plus_1
-m_point_index_plus_1 * k_index_plus_1 / h_index_plus_1
-q_index * cross_h_index;
  }
  assert(false);
}

/* clang-format on */

/* clang-format off */
auto DefaultMainMatrixCalculator::calc_g(size_t r_index, Number_t t) const -> Number_t {
  contract(fun) {
    precondition(r_index != 0, "You should not calculate anything for index == 0 - you already have v
function");
  };
  if (r_index == 1) {
    return params_>f(r_points_[r_index], t)
* calc_cross_h(r_points_, r_index-1)
+
middle_point(r_points_, r_index)
* params_>k(middle_point(r_points_, r_index), t)
/ calc_h(r_points_, r_index)
* params_>v1(t);
  } else if (r_index == r_points_.size() - 1) {
    return params_>f(r_points_[r_index], t) * calc_cross_h(r_points_, r_index)
+ params_>v2(t);
  } else {
    return params_>f(r_points_[r_index], t) * calc_cross_h(r_points_, r_index);
  }
  assert(false);
}

```

```

/* clang-format on */

#include <default_impl/rkf_method.hpp>

#include <Eigen/Sparse>

auto RKFMethod::integrate(
Eigen::VectorX<Number_t> const& start_v,
Eigen::MatrixX<Number_t> const& A,
Eigen::SparseVector<Number_t> const& g,
std::vector<Number_t> const& points
) -> Eigen::SparseMatrix<Number_t>
{
    auto result = Eigen::MatrixX<Number_t>(A.rows(), A.cols());
    result.col(0) = start_v.sparseView();

    for(size_t i = 1; i < points.size(); ++i) {
        auto H = points.at(i) - points.at(i - 1);

        Eigen::MatrixX<Number_t> u = result.col(i - 1);

        Eigen::MatrixX<Number_t> k1 = (H * (A * u + g));
        Eigen::MatrixX<Number_t> k2 = (H * (A * (u + 0.5 * k1) + g));
        Eigen::MatrixX<Number_t> k3 = (H * (A * (u + 0.5 * k2) + g));
        Eigen::MatrixX<Number_t> k4 = (H * (A * (u + k3) + g));

        result.col(i) = u + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
    }

    return result.sparseView();
}

#include <interval_splitter.hpp>

auto split_interval(Number_t const& left, Number_t const& right, size_t num_intervals) ->
std::vector<Number_t>
{
    std::vector<Number_t> intervals;

    contract(fun)
    {
        precondition(num_intervals > 0, "invalid number of intervals");
    };

    auto interval_size = (right - left) / num_intervals;
    for(size_t i = 0; i < num_intervals; ++i) {
        intervals.push_back(left + interval_size * i);
    }
    intervals.push_back(right);
    return intervals;
}

// Calculate the length of an interval `index-1` to `index`
auto calc_h(std::vector<Number_t> const& points, size_t index) -> Number_t
{
    contract(fun)
    {
        precondition(index < points.size(), "index out of range");
        precondition(index > 0, "h can not be calculated for the first interval");
    };

    return (points.at(index) - points.at(index - 1));
}

// Calculate the cross h of an interval
auto calc_cross_h(std::vector<Number_t> const& points, size_t index) -> Number_t
{
    if(index == 0) {
        return calc_h(points, 1) / 2;
    }
    else if(index == points.size() - 1) {
        return calc_h(points, index) / 2;
    }
}

```

```

}
else {
return (calc_h(points, index) + calc_h(points, index + 1)) / 2;
}
}

/// @return middle point between `index` and `index - 1`
auto middle_point(std::vector<Number_t> const& points, size_t index) -> Number_t
{
return (points.at(index) + points.at(index - 1)) / 2;
}

#include <iostream>
#include <iomanip> // For std::setw, std::fixed, std::setprecision, etc.
#include <chrono>

#include <Eigen/Dense>
#include <Eigen/Sparse>

#include <defines.hpp>
#include <input_parameters.hpp>
#include <default_impl/main_matrix_calculator.hpp>
#include <interval_splitter.hpp>
#include <default_impl/euler_explicit_method.hpp>
#include <default_impl/euler_implicit_method.hpp>
#include <default_impl/rkf_method.hpp>

auto build_main_matrix(
DefaultMainMatrixCalculator const& calc,
double t,
Eigen::MatrixX<Number_t>& main_matrix
) -> Eigen::MatrixX<Number_t>
{
auto const size = calc.r_points().size() - 1;
contract(fun) { precondition(main_matrix.rows() == size and main_matrix.cols() == size); };

for(size_t row = 0; row < size; ++row) {
main_matrix(row, row) = calc.calc_c(row + 1, t);

if(row + 1 < size) {
main_matrix(row, row + 1) = calc.calc_b(row + 1, t);
}

if(row > 0) {
main_matrix(row, row - 1) = calc.calc_a(row + 1, t);
}
}

return main_matrix;
}

auto build_g_vector(DefaultMainMatrixCalculator const& calc, Number_t t)
-> Eigen::SparseVector<Number_t>
{
auto g = Eigen::SparseVector<Number_t>(calc.r_points().size() - 1);
for(size_t row = 0; row < calc.r_points().size() - 1; ++row) {
g.insert(row) = calc.calc_g(row + 1, t);
}
return g;
}

auto integrate(
DefaultMainMatrixCalculator const& calc,
std::vector<Number_t> const& r_points,
std::vector<Number_t> const& t_points,
Eigen::VectorX<Number_t> const& start_v,
IBaseIntegrate& method
) -> std::tuple<Eigen::MatrixX<Number_t>, double, double>
{
auto result = Eigen::MatrixXd(Eigen::MatrixXd::Zero(r_points.size() - 1, t_points.size()));
result.col(0) = start_v;

```

```

std::cout << std::setprecision(4);
Eigen::MatrixX<Number_t> main_matrix(calc.r_points().size() - 1, calc.r_points().size() - 1);
Eigen::VectorX<Number_t> g;
double total_main_matrix_time = 0;
double total_integrate_time = 0;
for(size_t i = 1; i < t_points.size(); ++i) {
std::cerr << "Percentage done: " << (double)i / t_points.size() * 100 << " ";
std::cerr.flush();
auto start = std::chrono::high_resolution_clock::now();

if(dynamic_cast<IEulerImplicitMethod*>(&method)) {
build_main_matrix(calc, t_points.at(i), main_matrix);
g = build_g_vector(calc, t_points.at(i));
}
else {
build_main_matrix(calc, t_points.at(i - 1), main_matrix);
g = build_g_vector(calc, t_points.at(i - 1));
}

auto main_matrix_time = std::chrono::duration_cast<std::chrono::microseconds>(
std::chrono::high_resolution_clock::now() - start
)
.count();
total_main_matrix_time += main_matrix_time;
start = std::chrono::high_resolution_clock::now();
auto points = std::vector<Number_t>(t_points.cbegin() + i - 1, t_points.cbegin() + i + 1);
auto integrated_result =
method.integrate(result.col(i - 1), main_matrix.sparseView(), g.sparseView(), points);
auto integrate_time = std::chrono::duration_cast<std::chrono::microseconds>(
std::chrono::high_resolution_clock::now() - start
)
.count();
total_integrate_time += integrate_time;
result.col(i) = integrated_result.col(1);
std::cerr << "Time for main matrix: " << main_matrix_time
<< "us, Time for integration: " << integrate_time << "us\r";
std::cerr.flush();
}
std::cerr << "\n";

return std::make_tuple(
result,
total_main_matrix_time / (t_points.size() - 1),
total_integrate_time / (t_points.size() - 1)
);
}

void print_table_header()
{
// Set column widths for consistent formatting
std::cout << std::setw(20) << std::left << "Method Name" << std::setw(15) << "R Steps Count"
<< std::setw(12) << "R Step Size" << std::setw(15) << "T Steps Count" << std::setw(12)
<< "T Step Size" << std::setw(20) << "Sum Inaccuracy" << std::setw(20)
<< "Max Inaccuracy" << std::setw(30) << "Sum Inaccuracy (2nd Column)" << std::setw(30)
<< "Max Inaccuracy (2nd Column)" << std::setw(30) << "Sum Inaccuracy (Middle Col)"
<< std::setw(30) << "Max Inaccuracy (Middle Col)" << std::setw(30)
<< "Sum Inaccuracy (Last Col)" << std::setw(30) << "Max Inaccuracy (Last Col)"
<< std::setw(40) << "Main matrix build time (us)" << std::setw(40)
<< "Integration time (us)" << std::endl;

// Print a separator line
std::cout << std::string(260, '-') << std::endl;
}

void print_row(
R_T_Function_type original_func,
std::vector<Number_t> const& r_points,
std::vector<Number_t> const& t_points,
Eigen::VectorX<Number_t> const& start_v,
Eigen::MatrixXd const& result,
IBaseIntegrate& method,
double build_main_matrix_time,
double integration_time

```

```

}
{
size_t t_steps_count = t_points.size() - 1;
size_t r_steps_count = r_points.size() - 1;

Number_t t_step_size = (t_points.back() - t_points.front()) / t_steps_count;
Number_t r_step_size = (r_points.back() - r_points.front()) / r_steps_count;

// Initialize inaccuracies for columns and total
Number_t sum_inaccuracy_total = 0, max_inaccuracy_total = 0;
Number_t sum_inaccuracy_2nd_col = 0, max_inaccuracy_2nd_col = 0;
Number_t sum_inaccuracy_middle_col = 0, max_inaccuracy_middle_col = 0;
Number_t sum_inaccuracy_last_col = 0, max_inaccuracy_last_col = 0;
Number_t sum_inaccuracy_each_cell = 0;

// Iterate over all cells of the result matrix
for(size_t i = 1; i < r_points.size(); ++i) { // Skip the first row (r = 0)
for(size_t j = 0; j < t_points.size(); ++j) {
Number_t r = r_points[i];
Number_t t = t_points[j];
Number_t actual_value = original_func(r, t);
Number_t result_value = result(i - 1, j); // Note that we offset by 1 for r = 0

Number_t inaccuracy = std::abs(actual_value - result_value);
sum_inaccuracy_each_cell += inaccuracy;

// Sum and Max inaccuracies for the whole matrix
sum_inaccuracy_total += inaccuracy;
max_inaccuracy_total = std::max(max_inaccuracy_total, inaccuracy);

// Column-wise inaccuracies
if(j == 1) { // Second column
sum_inaccuracy_2nd_col += inaccuracy;
max_inaccuracy_2nd_col = std::max(max_inaccuracy_2nd_col, inaccuracy);
}
if(j == t_points.size() / 2) { // Middle column
sum_inaccuracy_middle_col += inaccuracy;
max_inaccuracy_middle_col = std::max(max_inaccuracy_middle_col, inaccuracy);
}
if(j == t_points.size() - 1) { // Last column
sum_inaccuracy_last_col += inaccuracy;
max_inaccuracy_last_col = std::max(max_inaccuracy_last_col, inaccuracy);
}
}
}

// Print the row in the desired format
std::cout << std::setw(20) << std::left << method.name() // Method name
<< std::setw(15) << r_steps_count // R Steps Count
<< std::setw(12) << r_step_size // R Step Size
<< std::setw(15) << t_steps_count // T Steps Count
<< std::setw(12) << t_step_size // T Step Size
<< std::setw(20) << std::fixed << std::setprecision(6)
<< sum_inaccuracy_total // Sum Inaccuracy (Total)
<< std::setw(20) << max_inaccuracy_total // Max Inaccuracy (Total)
<< std::setw(30) << sum_inaccuracy_2nd_col // Sum Inaccuracy (2nd Column)
<< std::setw(30) << max_inaccuracy_2nd_col // Max Inaccuracy (2nd Column)
<< std::setw(30) << sum_inaccuracy_middle_col // Sum Inaccuracy (Middle Column)
<< std::setw(30) << max_inaccuracy_middle_col // Max Inaccuracy (Middle Column)
<< std::setw(30) << sum_inaccuracy_last_col // Sum Inaccuracy (Last Column)
<< std::setw(30) << max_inaccuracy_last_col // Max Inaccuracy (Last Column)
<< std::setw(40) << build_main_matrix_time << std::setw(40) << integration_time
<< std::endl;
}

void do_all(std::shared_ptr<InputParameters> params, R T Function type expected_func)
{
print_table_header(); // Print header

static constexpr auto t_interval_counts = {100, 500, 1'000, 2'000, 5'000, 10'000};
static constexpr auto r_interval_counts = {5, 10, 20, 50, 100};
static std::vector<std::shared_ptr<IBaseIntegrate>> methods =

```

```

{std::make_shared<DefaultEulerExplicitMethod>(),
 std::make_shared<DefaultEulerImplicitMethod>(),
 std::make_shared<RKMethod>()});

for(auto const r_count : r_interval_counts) {
for(auto const t_count : t_interval_counts) {
for(auto& method : methods) {
auto t_points = split_interval(0, params->T, t_count);
auto r_points = split_interval(params->Rl, params->Rr, r_count);

DefaultMainMatrixCalculator calc(params, r_points);

Eigen::SparseVector<Number_t> start_v(r_points.size() - 1);
for(auto i = 0; i < r_points.size() - 1; ++i) {
start_v.insert(i) = params->phi(r_points.at(i + 1));
}

auto result = integrate(calc, r_points, t_points, start_v, *method);

print_row(
expected_func,
r_points,
t_points,
start_v,
std::get<0>(result),
*method,
std::get<1>(result),
std::get<2>(result)
);
std::cerr << "Finished for r: " << r_count << " t: " << t_count << "\n";
std::cerr.flush();
std::cout.flush();
}
std::cout << std::string(10, '-') << std::endl;
}
std::cout << std::string(50, '-') << std::endl;
}
}

void basic_example()
{
std::shared_ptr<InputParameters> params = std::make_shared<InputParameters>();
params->Rl = 1;
params->Rr = 10;
params->T = 1;
params->v1 = [](double t) { return 2 + t; };
params->hi2 = 3;
params->phi = [](double r) { return 2 * r; };
params->v2 = [](double t) { return 62 + 3 * t; };

params->k = [](double r, double t) { return 1.0; };
params->q = [](double r, double t) { return 3.0; };
params->f = [](double r, double t) { return 3 * t + 6 * r + 1 - 2 / r; };

auto expected_func = [](double r, double t) { return t + 2 * r; };

do_all(params, expected_func);
}

void hard_example()
{
std::shared_ptr<InputParameters> params = std::make_shared<InputParameters>();
params->Rl = 1;
params->Rr = 10;
params->T = 1;
params->v1 = [](double t) { return 5 + t * t; };
params->hi2 = 5;
params->phi = [](double r) { return 5 * r * r; };
params->v2 = [](double t) { return 5*(500 + t * t) + (20 + t*t)*(100 + t * t); };

params->k = [](double r, double t) { return 2 * r + t * t; };
params->q = [](double r, double t) { return 3 * r * r + t; };

```



```

params->f = [](double r, double t) {
return 2 * t - 40 - 20 * t * t + 15 * r * r * r * r + 3 * r * r * t * t + 5 * r * r * t
+ t * t * t;
};

auto expected_func = [](double r, double t) { return 5 * r * r + t * t; };

do_all(params, expected_func);
}

int main()
{
// basic_example();
hard_example();
return 0;
}

cmake_minimum_required(VERSION 3.16.3)

project(lab2 LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 20)

add_subdirectory(external/src/eigen)

add_library(${PROJECT_NAME} STATIC
external/src/contract/src/contract.cpp
src/default_impl/euler_explicit_method.cc
src/default_impl/euler_implicit_method.cc
src/default_impl/main_matrix_calculator.cc
src/default_impl/rkf_method.cc
src/interval_splitter.cc
)

target_include_directories(${PROJECT_NAME}
PUBLIC
include/public
external/src/contract/include
)

target_link_libraries(${PROJECT_NAME}
PUBLIC
Eigen3::Eigen
)

add_executable(${PROJECT_NAME}-main src/main.cc)
target_link_libraries(${PROJECT_NAME}-main ${PROJECT_NAME})

include(FetchContent)
FetchContent_Declare(
googletest
GIT_REPOSITORY https://github.com/google/googletest.git
GIT_TAG v1.14.0
)
FetchContent_MakeAvailable(googletest)

add_executable(${PROJECT_NAME}-test
tests/test.cc
tests/explicit-eugen-test.cc
tests/implicit-eugen-test.cc
)

target_link_libraries(${PROJECT_NAME}-test
${PROJECT_NAME}
gtest
)

```