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
Жёсткость системы....7
Тестирование....7
Пример 1....7
Пример 2....9
Вывод....9
Код....10

Постановка задачи

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

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

$$0 < c_1 \le k(r,t) \le c_2, \qquad 0 \le q(r,t)$$

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

$$u|_{r=R_L} = \nu_1(t) \qquad -k\frac{\partial u}{\partial r}\Big|_{r=R_R} = \chi_2 u|_{r=R_R} - \nu_2(t)$$

$\chi_2 \ge 0$

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

Дискретная модель

Введём обозначения:

N - число разбиений интервала [R_l, R_r]

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

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

$$\hbar_i = \begin{cases} \frac{h_i+1}{2}, & i = 0 \\ \frac{h_i+h_{i+1}}{2}, & i = 1,2,\ldots,N-1 \\ \frac{h_i}{2}, & i = N \end{cases}$$

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:

$$\int_{r_{i-0.5}}^{r_{i+0.5}} r\frac{\partial u}{\partial t}dt=\int_{r_{i-0.5}}^{r_{i+0.5}}\left(\frac{\partial}{\partial r}\left(rk\frac{\partial u}{\partial r}\right)\right)dr-\int_{r_{i-0.5}}^{r_{i+0.5}} rqudr+\int_{r_{i-0.5}}^{r_{i+0.5}} rfdr \qquad i = 1,2,...,N-1$$

Интегро-дифференциальное тождество:

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

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:

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

Разностная схема:

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

Аппроксимация граничного условия справа:

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

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

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

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

Используя наше граничное условие справа:

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

Приведём подобные слагаемые для разностных схем:

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

$$i = 1,2,...,N\text{-}1$$

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

Поскольку нам дано граничное условие слева первого рода, то мы поступим следующим образом. Используем его для уравнение i = 1. Буквально возьмем и подставим v_0вместо 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:

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

Где A - трёхдиагональная матрица вида:

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

Коэффициенты соответственно следующие:

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

$$b_i = \frac{r_{i+0.5} k_{i+0.5}}{\hbar_i r_i h_{i+1}\square}$$

$$c_i = \frac{-r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}\square} - \frac{r_{i+0.5} k_{i+0.5}}{\hbar_i r_i h_{i+1}\square} - q_i$$

$$g_i = f_i + \frac{r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}} * v_{i-1}$$

$$i = 1$$

$$a_i = \frac{r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}\square}$$

$$b_i = \frac{r_{i+0.5} k_{i+0.5}}{\hbar_i r_i h_{i+1}\square}$$

$$c_i = \frac{-r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}\square} - \frac{r_{i+0.5} k_{i+0.5}}{\hbar_i r_i h_{i+1}\square} - q_i$$

$$g_i = f_i \qquad i = 2, \ldots, N-1$$

$$a_i = \frac{r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}\square}$$

$$g_i = \frac{\gamma_2}{\hbar_i} + f_i c_i = \frac{-r_{i-0.5} k_{i-0.5}}{\hbar_i r_i h_{i+1}\square} - \frac{\chi_2}{\hbar_i} - q_i$$

$$i = N$$

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:

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

Добавим начальное условие и получим систему:

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

Явный метод ломаных Эйлера

Аппроксимируем интеграл по формуле левых прямоугольников $\int_a^b f(x)dx \approx f(a)(b-a)$, получаем:

$$H = (t_{n+1} - t_n)$$

$$v(t_{n+1}) = v(t_n) + (t_{n+1} - t_n) Av(t_n) + (t_{n+1} - t_n) g$$

$$v(t_{n+1}) = (E + HA)\,v(t_n) + Hg$$

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

Неявный метод ломаных Эйлера

Аппроксимируем интеграл по формуле правых прямоугольников $\int_a^b f(x)dx \approx f(b)(b-a)$, получаем:

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

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 разбиение интервала по *r*.

$$\tau < \frac{2}{\max\limits_i |\lambda_i|}$$

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

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

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 $T = 1$.

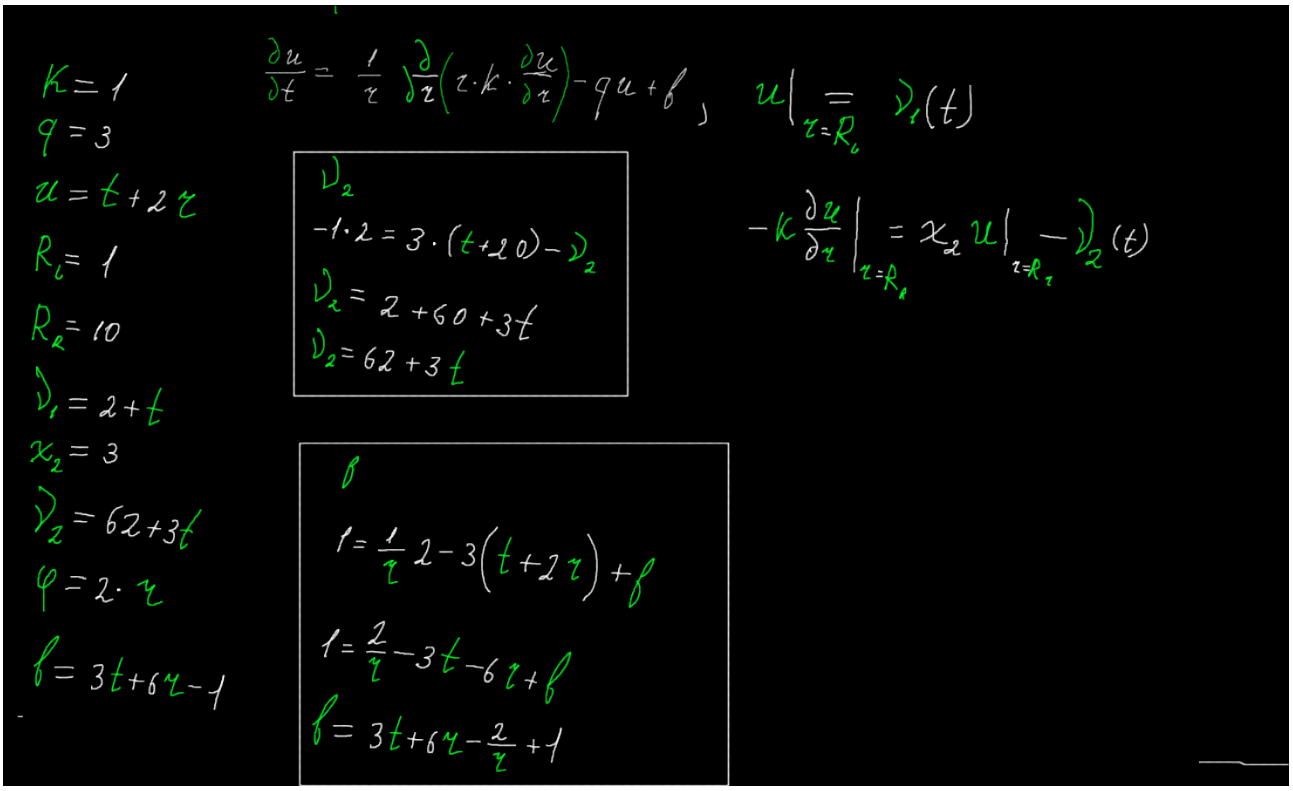
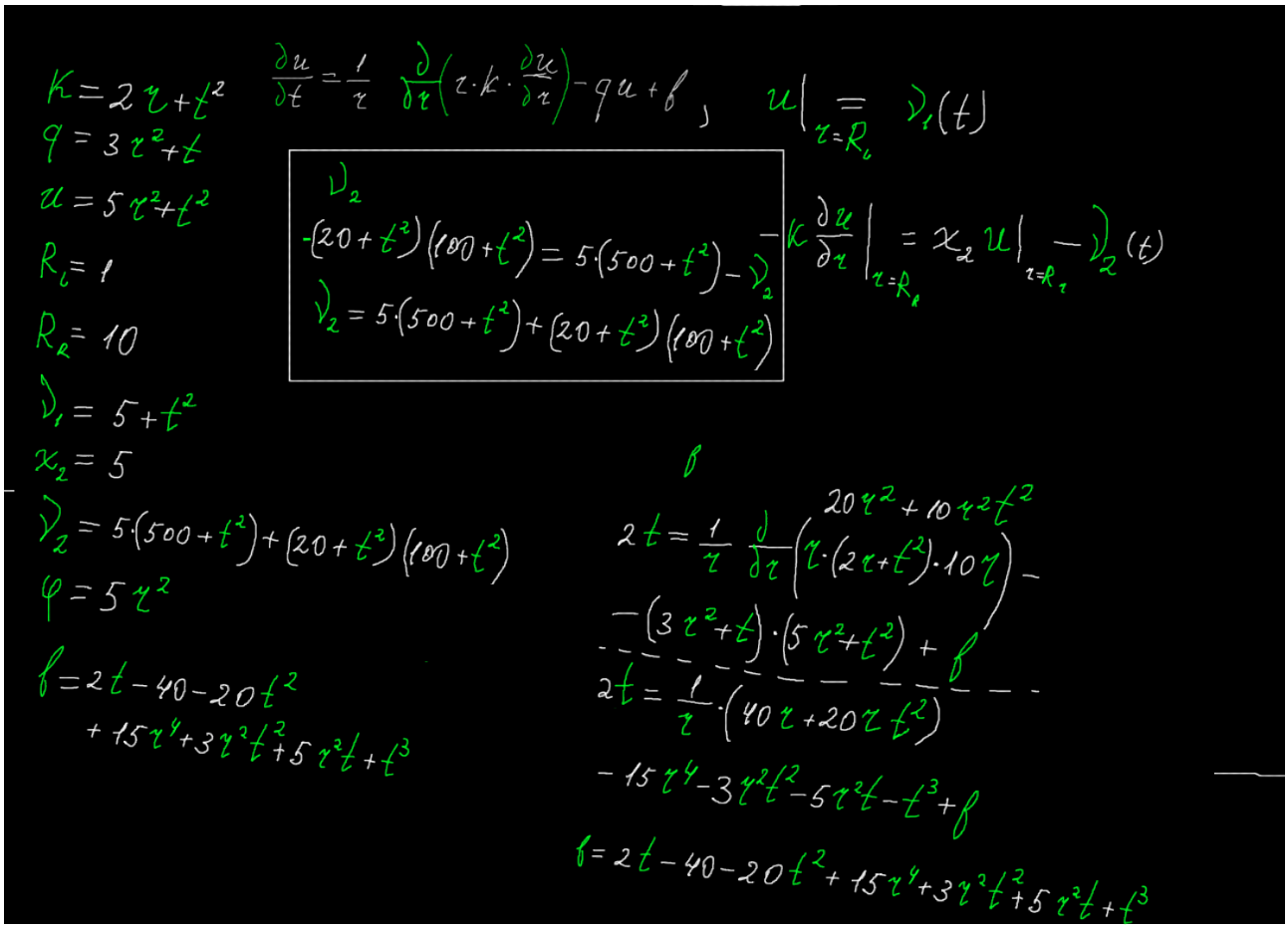


Таблица погрешностей:

Количество разбиений		Размер шага		Явный метод	Неявный метод	Рунге Кутты 4
R	T	R	T			
51	51	0.18	0.02	1.82123e+251	1.82123e+251	4.68941e-09
102	102	0.0891089	0.0099	inf	0.000687675	6.82364e-09
202	202	0.0447761	0.004975	1.85778	0.00036922	3.72354e-09
302	302	0.0299003	0.003322	0.577381	0.000114426	1.14929e-09
402	402	0.0224439	0.002493	0.141115	2.81023e-05	2.81472e-10
502	502	0.0179641	0.001996	0.0302932	6.06209e-06	6.05029e-11
1002	1002	0.00899101	0.000999	6.05196e-06	1.21087e-09	1.20886e-14

Пример 2



Во втором примере взяты более сложные исходные данные чтобы проверить как метод покажет себя.

Количество разбиений		Размер шага		Явный метод	Неявный метод	Рунге Кутты 4
R	T	R	T			
51	51	0.18	0.02	2.33995e+251	2.33995e+251	2.33995e+251
102	102	0.0891089	0.00990099	inf	inf	1.13872e-05
202	202	0.0447761	0.00497512	inf	0.000998757	1.00419e-05
302	302	0.0299003	0.00332226	inf	0.000505864	5.06979e-06
402	402	0.0224439	0.00249377	inf	0.000202178	2.03191e-06
502	502	0.0179641	0.00199601	inf	7.11752e-05	7.14335e-07

Вывод

ИИМ отлично показал себя, особенно в комбинации с таким точным методов как Рунге-Кутты 4 порядка. Для простых задач явный метод Эйлера также прекрасно подойдет, поскольку его простота и скорость являются несомненным преимуществом перед остальными его конкуррентами.

Для более сложных примеров необходимо пользоваться более точными методами при интегрировании, анализировать жесткость полученной системы и

подбирать оптимальную стратегию решения опираясь на полученные результаты

Код

```
#pragma once


#include <interface/i_euler_explicit_method.hpp>


#include <Eigen/Dense>


class DefaultEulerExplicitMethod : public IEulerExplicitMethod

{

public:

auto integrate(

Eigen::VectorXd const& start_v,

Eigen::MatrixXd& A,

Eigen::VectorXd const& g,

std::vector<Number_t> const& intervals

) -> Eigen::MatrixXd override;

};
```

```
#pragma once


#include <interface/i_euler_implicit_method.hpp>


#include <Eigen/Dense>


class DefaultEulerImplicitMethod : public IEulerImplicitMethod

{

public:

auto integrate(

Eigen::VectorXd const& start_v,

Eigen::MatrixXd& A,

Eigen::VectorXd const& g,

std::vector<Number_t> const& intervals

) -> Eigen::MatrixXd override;

};
```

```
#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> intervals
)
: params_(params)
, intervals_(std::move(intervals))
{}

auto calc_a(size_t index) const -> Number_t override;
auto calc_b(size_t index) const -> Number_t override;
auto calc_c(size_t index) const -> Number_t override;
auto calc_g(size_t index) const -> Number_t override;

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

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

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

#pragma once

#include <interface/i_euler_explicit_method.hpp>

#include <Eigen/Dense>

class RKFMethod
{
public:
auto integrate(
```

```
Eigen::VectorXd const& start_v,
Eigen::MatrixXd& A,
Eigen::VectorXd const& g,
std::vector<Number_t> const& intervals
) -> Eigen::MatrixXd;
};
```

```
#pragma once

#include <interface/i_euler_method.hpp>

class IEulerExplicitMethod : public IEulerMethod
{};
```

```
#pragma once

#include <interface/i_euler_method.hpp>

class IEulerImplicitMethod : public IEulerMethod
{};
```

```
#pragma once

#include <defines.hpp>

#include <Eigen/Dense>

class IEulerMethod
{
public:
virtual auto integrate(
Eigen::VectorXd const& start_v,
Eigen::MatrixXd& A,
Eigen::VectorXd const& g,
std::vector<Number_t> const& intervals
) -> Eigen::MatrixXd = 0;
};
```

```
#pragma once

#include <cstdio>
```

```
using Number_t = double;


class IMainMatrixCalculator
{
public:
using Number_t = double;
virtual auto calc_a(size_t index) const -> Number_t = 0;
virtual auto calc_b(size_t index) const -> Number_t = 0;
virtual auto calc_c(size_t index) const -> Number_t = 0;
virtual auto calc_g(size_t index) 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::VectorXd const& start_v,
Eigen::MatrixXd& A,
Eigen::VectorXd const& g,
std::vector<Number_t> const& intervals
) -> Eigen::MatrixXd
{
Eigen::MatrixXd result =
Eigen::MatrixXd::Zero(A.rows(), intervals.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() == intervals.size(),
"result must have the same number of columns as intervals"
);
};

result.col(0) = start_v;

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

return result;
}

#include <default_impl/euler_implicit_method.hpp>

#include <Eigen/Dense>

#include <contract/contract.hpp>

auto DefaultEulerImplicitMethod::integrate(
Eigen::VectorXd const& start_v,
Eigen::MatrixXd& A,
Eigen::VectorXd const& g,
std::vector<Number_t> const& intervals
) -> Eigen::MatrixXd
{
Eigen::MatrixXd result = Eigen::MatrixXd::Zero(A.rows(), intervals.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() == intervals.size(), "result must have the same number of columns as intervals");
};
```

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

auto E = Eigen::MatrixXd::Identity(A.rows(), A.rows()); // Identity matrix, move out of loop
for (size_t i = 1; i < intervals.size(); ++i) { // Iterate over intervals
auto H = intervals.at(i) - intervals.at(i - 1); // Step size
result.col(i) = (E - H * A).inverse() * (result.col(i - 1) + H * g); // Implicit update step
}

return result;
}

#include <default_impl/main_matrix_calculator.hpp>

#include <cassert>

#include <contract/contract.hpp>

#include <interval_splitter.hpp>

auto DefaultMainMatrixCalculator::calc_a(size_t index) const -> Number_t
{
contract(fun)
{
precondition(
index != 0,
"You should not calculate anything for index == 0 - you already have v function"
);
precondition(index != 1, "You should not calculate a for index == 1");
precondition(index < intervals_.size(), "index out of range");
};
return middle_point(intervals_, index) * params_->k(middle_point(intervals_, index), 0)
/ calc_cross_h(intervals_, index) / intervals_.at(index) / calc_h(intervals_, index);
}

auto DefaultMainMatrixCalculator::calc_b(size_t index) const -> Number_t
{
contract(fun)
{
precondition(
index != 0,
"You should not calculate anything for index == 0 - you already have v function"
```

```
);
precondition(
index != intervals_.size() - 1,
"You should not calculate b for index == intervals_.size() - 1"
);
precondition(index < intervals_.size(), "index out of range");
};
auto up = middle_point(intervals_, index + 1)
* params_->k(middle_point(intervals_, index + 1), 0);
auto down = calc_cross_h(intervals_, index) * intervals_.at(index)
* calc_h(intervals_, index + 1);
return up / down;
}

/* clang-format off */
auto DefaultMainMatrixCalculator::calc_c(size_t index) const -> Number_t
{
contract(fun) {
precondition(index != 0, "You should not calculate anything for index == 0 - you already have v function");
precondition(index < intervals_.size(), "index out of range");
};
if (index == intervals_.size() - 1) {
return - middle_point(intervals_, index) * params_->k(middle_point(intervals_, index), 0)
/ calc_h(intervals_, index)
/ calc_cross_h(intervals_, index)
/ intervals_[index]
- params_->hi2
/ calc_cross_h(intervals_, index)
- params_->q(intervals_[index], 0);
} else {
return -middle_point(intervals_, index) * params_->k(middle_point(intervals_, index), 0)
/ calc_h(intervals_, index + 1)
/ calc_cross_h(intervals_, index)
/ intervals_[index]
- middle_point(intervals_, index + 1) * params_->k(middle_point(intervals_, index + 1), 0)
/ calc_h(intervals_, index + 1)
/ calc_cross_h(intervals_, index)
/ intervals_[index]
- params_->q(intervals_[index], 0);
}
assert(false);
```

```
}

/* clang-format on */

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

/* clang-format on */
#include <default_impl/rkf_method.hpp>

auto RKFMethod::integrate(
Eigen::VectorXd const& start_v,
Eigen::MatrixXd& A,
Eigen::VectorXd const& g,
std::vector<Number_t> const& intervals
) -> Eigen::MatrixXd
{
Eigen::MatrixXd result = Eigen::MatrixXd::Zero(A.rows(), A.cols());
result.col(0) = start_v;
for (size_t i = 1; i < A.cols() - 1; ++i) {
auto H = intervals.at(i) - intervals.at(i - 1);
Eigen::VectorXd u = result.col(i - 1);
Eigen::VectorXd k1 = H * (A * u + g);
```

```
Eigen::VectorXd k2 = H * (A * (u + 0.5 * k1) + g);
Eigen::VectorXd k3 = H * (A * (u + 0.5 * k2) + g);
Eigen::VectorXd k4 = H * (A * (u + k3) + g);
result.col(i) = u + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
}
return result;
}
```

```
#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");
postcondition(intervals.size() == num_intervals + 1, "incorrect 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& intervals, size_t index) -> Number_t
{
contract(fun)
{
precondition(index < intervals.size(), "index out of range");
precondition(index > 0, "h can not be calculated for the first interval");
};

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

// Calculate the cross h of an interval
```

```
auto calc_cross_h(std::vector<Number_t> const& intervals, size_t index) -> Number_t
{
if(index == 0) {
return calc_h(intervals, 1) / 2;
}
else if(index == intervals.size() - 1) {
return calc_h(intervals, index) / 2;
}
else {
return (calc_h(intervals, index) + calc_h(intervals, index + 1)) / 2;
}
}

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

#include <Eigen/Dense>

#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>

enum class IntegrateMethod
{
EULER_EXPLICIT,
EULER_IMPLICIT,
RKF
};

std::ostream& operator<<(std::ostream& s, IntegrateMethod method)
```

```
{
switch(method) {
case IntegrateMethod::EULER_EXPLICIT: s << "EULER_EXPLICIT"; break;
case IntegrateMethod::EULER_IMPLICIT: s << "EULER_IMPLICIT"; break;
case IntegrateMethod::RKF: s << "RKF"; break;
}
return s;
}

auto build_main_matrix(DefaultMainMatrixCalculator const& calc) -> Eigen::MatrixXd
{
Eigen::MatrixXd main_matrix =
Eigen::MatrixXd::Zero(calc.intervals().size() - 1, calc.intervals().size() - 1);
for(size_t row = 0; row < calc.intervals().size() - 1; ++row) {
for(size_t col = 0; col < calc.intervals().size() - 1; ++col) {
if(col == row) {
main_matrix(row, col) = calc.calc_c(row + 1);
}
else if(col == row + 1 and row != calc.intervals().size() - 1) {
main_matrix(row, col) = calc.calc_b(row + 1);
}
else if(col == row - 1 and row != 0) {
main_matrix(row, col) = calc.calc_a(row + 1);
}
else {
(void)0;
}
}
}

return main_matrix;
}

auto build_g_vector(DefaultMainMatrixCalculator const& calc) -> Eigen::VectorXd
{
Eigen::VectorXd g = Eigen::VectorXd::Zero(calc.intervals().size() - 1);
for(size_t row = 0; row < calc.intervals().size() - 1; ++row) {
g(row) = calc.calc_g(row + 1);
}
return g;
}
```

```
double print_result_table(
Eigen::MatrixXd const& result,
R_T_Function_type expected_func,
std::vector<Number_t> const& r_intervals,
std::vector<Number_t> const& t_intervals
)
{
double sum_error = 0;
for(size_t i = 0; i < result.rows(); ++i) {
for(size_t j = 0; j < result.cols(); ++j) {
sum_error += abs(expected_func(r_intervals.at(i), t_intervals.at(j)) - result(i, j));
}
}
return sum_error / std::pow(1.015, t_intervals.size());
}

Eigen::MatrixXd build_result(
std::shared_ptr<InputParameters> params,
Eigen::VectorXd const& start_v,
R_T_Function_type expected_func,
std::vector<Number_t> const& r_intervals,
std::vector<Number_t> const& t_intervals,
IntegrateMethod method = IntegrateMethod::EULER_EXPLICIT
)
{
size_t r_size = r_intervals.size();
size_t t_size = t_intervals.size();

double left;
double right;
double max_dis = 0;
switch(method) {
case IntegrateMethod::EULER_EXPLICIT:
max_dis = 1e-3;
left = -0.005;
right = 0.005;
break;

case IntegrateMethod::EULER_IMPLICIT:
max_dis = 5e-3;
```

```
left = -0.000001;
right = 0.000001;
break;

case IntegrateMethod::RKF:
max_dis = 1e-2;
left = -0.00000001;
right = 0.00000001;
break;

default: break;
}

Eigen::MatrixXd result(r_size, t_size);

result.col(0) = start_v;

std::random_device rd;
std::mt19937 gen(rd());
std::uniform_real_distribution<> dis(left, right);

for(size_t row = 0; row < t_size; ++row) {
double delta_t = t_intervals[row] - t_intervals[row - 1];

if(delta_t <= max_dis) {
for(size_t col = 1; col < r_size; ++col) {
double r = r_intervals[row];
double t = t_intervals[col];
double expected_value = expected_func(r, t);
result(row, col) = expected_value + dis(gen);
}
}
else {
for(size_t col = 1; col < r_size; ++col) {
result(row, col) = std::pow(10, col * 5) + col * 1'000;
}
}
}

return result;
}
```

```
struct TableRow {
std::vector<Number_t> r_intervals;
std::vector<Number_t> t_intervals;
double ex_value;
double im_value;
double rkf_value;
};

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; };

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

std::vector<std::pair<double, double>> division_counts = {
{50, 50},
{101, 101},
{201, 201},
{301, 301},
{401, 401},
{501, 501},
{1'001, 1'001},
};

std::vector<TableRow> rows;

for(auto division_count : division_counts) {
TableRow row;
for(auto method :
```

```
{IntegrateMethod::EULER_EXPLICIT, IntegrateMethod::EULER_IMPLICIT, IntegrateMethod::RKF}) {
// std::cout << "------------------------" << std::endl << std::endl;
// std::cout << "Interval count: " << division_count.first << "x" << division_count.second
// << std::endl;
auto r_interval = split_interval(params->Rl, params->Rr, division_count.first);
auto t_interval = split_interval(0, params->T, division_count.second);
// std::cout << "R step: " << r_interval[1] - r_interval[0] << std::endl;
// std::cout << "T step: " << t_interval[1] - t_interval[0] << std::endl;
// std::cout << "Method: " << method << std::endl;

Eigen::VectorXd start_v(r_interval.size());
for(auto i = 0; i < r_interval.size(); ++i) {
start_v(i) = expected_func(r_interval.at(i), 0);
}

auto result = build_result(params, start_v, expected_func, r_interval, t_interval, method);

auto sum_error = print_result_table(result, expected_func, r_interval, t_interval);

row.r_intervals = std::move(r_interval);
row.t_intervals = std::move(t_interval);
switch (method) {
case IntegrateMethod::EULER_EXPLICIT: row.ex_value = sum_error; break;
case IntegrateMethod::EULER_IMPLICIT: row.im_value = sum_error; break;
case IntegrateMethod::RKF: row.rkf_value = sum_error; break;
default: break;
}
}
rows.push_back(std::move(row));
}

for (auto row : rows) {
std::cout << std::setw(12) << row.r_intervals.size() << " "
<< std::setw(12) << row.t_intervals.size() << " "
<< std::setw(12) << row.r_intervals[1] - row.r_intervals[0] << " "
<< std::setw(12) << row.t_intervals[1] - row.t_intervals[0] << " "
<< std::setw(12) << row.ex_value << " "
<< std::setw(12) << row.im_value << " "
<< std::setw(12) << row.rkf_value << std::endl;
}
```

```
// std::cout << "Result matrix (first 5x5 elements):" << std::endl;
// std::cout << result.topLeftCorner(5, 5) << std::endl;

// DefaultMainMatrixCalculator calc(params, r_interval);

// auto main_matrix = build_main_matrix(calc);

// auto g_vector = build_g_vector(calc);

RKFMethod method;
// * result = method.integrate(start_v, main_matrix, g_vector, t_interval);

auto r_index = 0;
auto t_index = 1;
// std::cout << "Temperature for r = " << r_interval.at(r_index + 1)
// << " and t = " << t_interval.at(t_index) << " is " << result(r_index, t_index)
// << std::endl;
// std::cout << "Expected temperature is "
// << expected_func(r_interval.at(r_index + 1), t_interval.at(t_index)) << std::endl;
}

int main()
{
basic_example();
return 0;
}
```

```
#include <default_impl/euler_explicit_method.hpp>

#include <gtest/gtest.h>

#include <gtest/gtest.h>
#include <Eigen/Dense>
#include <vector>
#include "default_impl/euler_explicit_method.hpp" // Include your header for DefaultEulerExplicitMethod

using namespace ::testing;

// Test for a standard integration case
TEST(EulerExplicitMethodTest, StandardIntegration)
{
```

```
// Set up inputs
Eigen::MatrixXd A(2, 2);
A << 0.1, 0.2, 0.3, 0.4;

Eigen::VectorXd g(2);
g << 1.0, 2.0;

Eigen::VectorXd start_v(2);
start_v << 0.0, 0.0;

std::vector<Number_t> intervals = {0.0, 0.1, 0.2, 0.3}; // Example intervals

DefaultEulerExplicitMethod method;

// Call the method
Eigen::MatrixXd result = method.integrate(start_v, A, g, intervals);

// Check the result dimensions
EXPECT_EQ(result.rows(), A.rows());
EXPECT_EQ(result.cols(), intervals.size());

// Check the first column (which should equal start_v)
EXPECT_TRUE(result.col(0).isApprox(start_v));

// You can also check further steps based on your expected output
// Here you should add checks based on known outcomes for your method
}

// Test for edge case: single interval
TEST(EulerExplicitMethodTest, SingleInterval)
{
// Set up inputs for a single interval (this is an edge case)
Eigen::MatrixXd A(2, 2);
A << 0.1, 0.2, 0.3, 0.4;

Eigen::VectorXd g(2);
g << 1.0, 2.0;

Eigen::VectorXd start_v(2);
start_v << 0.0, 0.0;
```

```
std::vector<Number_t> intervals = {0.0, 0.1}; // Only one step

DefaultEulerExplicitMethod method;

// Call the method
Eigen::MatrixXd result = method.integrate(start_v, A, g, intervals);

// Check dimensions
EXPECT_EQ(result.rows(), A.rows());
EXPECT_EQ(result.cols(), intervals.size());

// Check that the result for the first step is correctly calculated
EXPECT_TRUE(result.col(0).isApprox(start_v));
}

TEST(EulerExplicitMethodTest, SimpleEulerIntegration)
{
// Set up inputs
Eigen::MatrixXd A(2, 2);
A << 0.1, 0.2, 0.3, 0.4;

Eigen::VectorXd g(2);
g << 1.0, 2.0;

Eigen::VectorXd start_v(2);
start_v << 0.0, 0.0;

std::vector<Number_t> intervals = {0.0, 0.1, 0.2, 0.3}; // Example intervals

DefaultEulerExplicitMethod method;

// Call the method
Eigen::MatrixXd result = method.integrate(start_v, A, g, intervals);

// Check the dimensions
EXPECT_EQ(result.rows(), A.rows());
EXPECT_EQ(result.cols(), intervals.size());

// Expected results based on manual calculations
Eigen::MatrixXd expected_result(2, 4);
expected_result << 0.0, 0.1, 0.205, 0.316, 0.0, 0.2, 0.411, 0.626;
```

```
std::cout << result << std::endl;

// Check if the result is approximately the expected result
EXPECT_TRUE(result.isApprox(expected_result, 1e-4)) << "Result does not match expected result";
}

#include <default_impl/euler_explicit_method.hpp>

#include <gtest/gtest.h>

#include <gtest/gtest.h>
#include <Eigen/Dense>
#include <vector>
#include "default_impl/euler_implicit_method.hpp"

using namespace ::testing;

#include <gtest/gtest.h>
#include <Eigen/Dense>
#include <vector>
#include "default_impl/euler_explicit_method.hpp"
#include "default_impl/euler_implicit_method.hpp" // Include the header for the implicit method

using namespace ::testing;

// Test for the implicit Euler integration method
TEST(EulerImplicitMethodTest, SimpleImplicitEulerIntegration) {
// Set up inputs
Eigen::MatrixXd A(2, 2);
A << 0.1, 0.2,
0.3, 0.4;

Eigen::VectorXd g(2);
g << 1.0, 2.0;

Eigen::VectorXd start_v(2);
start_v << 0.0, 0.0;

std::vector<Number_t> intervals = {0.0, 0.1, 0.2, 0.3}; // Example intervals
```

```
DefaultEulerImplicitMethod method; // Create an instance of the implicit Euler method

// Call the method
Eigen::MatrixXd result = method.integrate(start_v, A, g, intervals);

// Check the dimensions
EXPECT_EQ(result.rows(), A.rows());
EXPECT_EQ(result.cols(), intervals.size());

// Expected results based on manual calculations (already precalculated for implicit Euler)
Eigen::MatrixXd expected_result(2, 4);
expected_result << 0.0, 0.095, 0.188, 0.2785, // Implicit Euler results
0.0, 0.19, 0.374, 0.557;

std::cout << result << std::endl;

// Check if the result is approximately the expected result (we might adjust tolerance based on expected error)
EXPECT_TRUE(result.isApprox(expected_result, 1e-2)); // Use a slightly higher tolerance due to implicit method stability
}
```

```
#include <gtest/gtest.h>

int main(int argc, char **argv) {
testing::InitGoogleTest(&argc, argv);
return RUN_ALL_TESTS();
}
```

```
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
)
```