МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Кубанский государственный университет»

Факультет компьютерных технологий и прикладной математики

Кафедра математического моделирования

**ОТЧЁТ О ПРОХОЖДЕНИИ УЧЕБНОЙ ПРАКТИКИ**

(практике по получению первичных профессиональных умений и навыков)

Выполнил\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ К. А. Ненашев

Направление подготовки 01.03.02 Прикладная математика и информатика

Курс 2

Руководитель учебной практики

канд. физ.-мат. наук, доцент кафедры

математического моделирования \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.Е. Рубцов

Краснодар

2024г.

**СОДЕРЖАНИЕ**

1 Постановка задачи………………………………………………………………. 3

2 Описание метода Эйлера и расчётные формулы для вычисления *yi = y(xi)*… 4

3 Аналитическое решение задачи Коши………………………………………… 5

4 Описание программы…………………………………………………………… 7

5 Результаты вычислений………………….……………………………………... 9

Список используемых источников………………………………………………. 11

Приложение………………………………………………………………………... 12

**1 Постановка задачи**

Дано дифференциальное уравнение *,*

Методом ломанных Эйлера получить приближённое решение задачи Коши для заданного дифференциального уравнения. Начальное условие . Вычисления произвести при помощи программы, разработанной на языке высокого уровня, для различных значений N. В программе предусмотреть ввод N.

Получить аналитически точное решение задачи Коши.

В одной системе координат построить графики точного и приближённого решений. Вычислить максимальную невязку. Для построения графика использовать графические возможности выбранного языка программирования.

**2 Описание метода Эйлера и расчётные формулы**

В основе метода Эйлера (метод ломаных) лежит идея графического построения решения дифференциальных уравнений, однако этот метод дает одновременно и способ нахождения искомой функции в численной (табличной) форме.

Пусть дано дифференциальное уравнение:

с начальными условиями:

Выбрав достаточно малый шаг h, строится система равноотстоящих точек

В методе Эйлера приближенные значения вычисляются последовательно по формулам:

При этом искомая интегральная кривая , проходящая через точку , заменяется ломанной с вершинами . Каждое звено этой ломанной, называемой ломанной Эйлера, имеет направление, совпадающее с направлением той интегральной кривой уравнения , которая проходит через точку .

**3 Аналитическое решение задачи Коши**

,

Вычтем с двух сторон

Получаем уравнение Бернулли, делим на 2x

, Замена

Пусть: , следовательно:

Пусть: , умножим на него

Положим: , тогда

Из взятия производной произведения следует, что:

тогда

Теперь можно перейти к интегрированию по

Поделим на

Обратная подстановка , Получим

Значит итоговые функции:

Учитывая начальные условия решим поставленную задачу Коши:

Это уравнение не имеет решения и не может соблюсти начальное условие, следовательно этого решения не существует.

Значит и остаётся лишь одно итоговое решение:

**4 Описание программы**

Программа написана на языке Rust с использованием библиотеки Plotly для построения графиков и включает в себя 1 типаж, 2 структуры с реализацией типажа и 6 функций

- Функция fn derivative(x: f64, y: f64) → f64  
Вычисляет значение y’ в точке

- Функция fn by\_hand(x: f64) → f64  
Вычисляет значение функции y в точке x

- Функция fn span\_gen(span: (f64, f64), h: f64) → Vec<f64>  
Вычисляет вектор всех значений оси x из span.0 в span.1 с шагом h

- Функция fn euler<F>(f: &F, x\_n: f64, y\_n: f64, h: f64) -> f64  
Вычисляет значение y’ в точке n+1, применяя f.

- Типаж Solver<F1, F2>   
Описывает типаж, который задаёт типу функцию  
fn solve(f: &F1, xs: &Vec<f64>, y\_0: f64, solver: &F2, h: f64) -> Vec<f64>, которая в свою очередь возвращает все значения оси y в заданных точках xs.

- Структура NumericalSolver  
Реализует типаж Solver<F1, F2> и имеет ассоциированную  
fn solve(f: &F1, xs: &Vec<f64>, y\_0: f64, solver: &F2, h: f64) -> Vec<f64>  
Которая в свою очередь возвращает все значения y при заданных точках xs и указанном способе численного решения solver (в нашем случае Эйлера) и указанной функции, задающей значение производной в точке f.

- Структура AnalyticalSolver  
Реализует типаж Solver<F1, F2> и имеет ассоциированную  
fn solve(f: &F1, xs: &Vec<f64>, \_y\_0: f64, \_solver: &(), \_h: f64) -> Vec<f64>  
Которая в свою очередь возвращает все значения y при заданных точках в заданном решении дифференциального уравнения f.

- Функция fn main() → Result<(), Box<dyn std::error::Error>>  
Задаёт основное поведение программы. В нём происходит вычисление значений и построение графиков.

В начале работы программы пользователю выводится сообщение с просьбой ввести число N, которое будет использоваться как размер шага. Затем программа считывает введенную строку в переменную input и преобразует ее в число типа i32.

Далее программа вычисляет размер шага, преобразуя N в значение f32. Затем генерируется xs, вектор чисел, каждый элемент которого будет подставляться в функцию для вычисления соответствующих значений оси y. Также генерируется xs\_exact, вектор значений оси x для построения графика аналитического решения с высокой точностью для сравнения на графике.

В переменную ys\_e записываются значения, полученные методом Эйлера на оси y в точках xs. В переменную ys\_a записываются значения, полученные аналитическим решением в точках xs. Эта переменная будет использоваться для вычисления максимальной погрешности по отношению к решению, полученному методом Эйлера. В переменную ys\_aex записываются значения для точного графика аналитического решения в точках xs\_exact.

Затем в переменную name записывается путь к файлу с генерируемой картинкой графика. Переменная root инициализирует бэкенд для отрисовки растровой графики. С помощью root.fill(&WHITE) задается белый цвет фона. Затем, в переменной chart, создается двухмерная координатная плоскость, и на следующем шаге отрисовывается сетка.

FIX!!!!!!!!!!!!1С помощью метода .draw\_series мы передаем серию кортежей (x,y), полученных после применения zip и map, которые представляют собой точки, и между каждой парой точек проводится линия. С помощью метода .label задается название кривой “Euler’s approximation”. С помощью метода .legend задается цвет кривой и ее “Легенда” (надпись на картинке с графиком). Затем снова с помощью метода .draw\_series мы рисуем вторую ветвь параболы. После этого на графике будут отображены две ветви параболы, которые являются приближением, полученным методом Эйлера.

Затем будет проделано то же самое, но уже для значений, полученных аналитическим способом.

После этого происходит финальная конфигурация графика и его отрисовка. В конце в переменную mx\_error будет записано значение максимальной погрешности, после чего напечатано.

**5 Результаты вычислений**

Предоставленные ниже изображения являются результатом работы программы при входных данных.

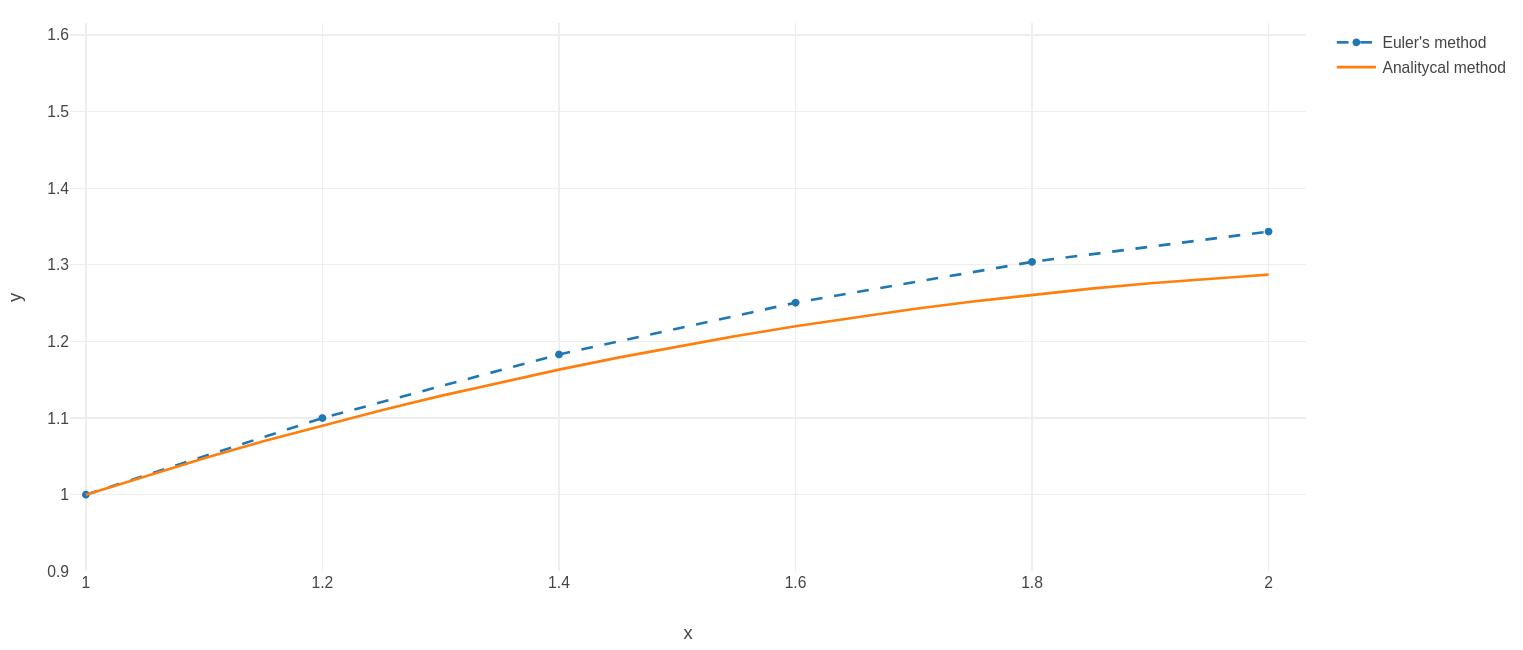
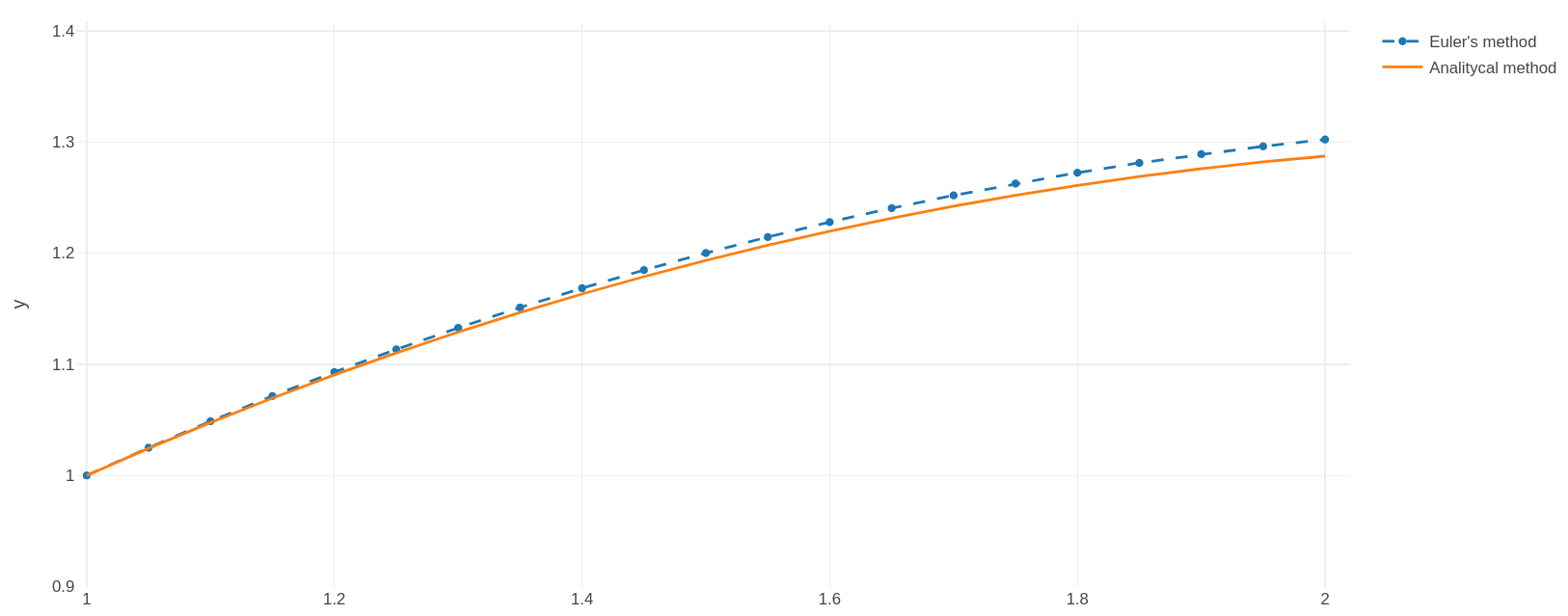
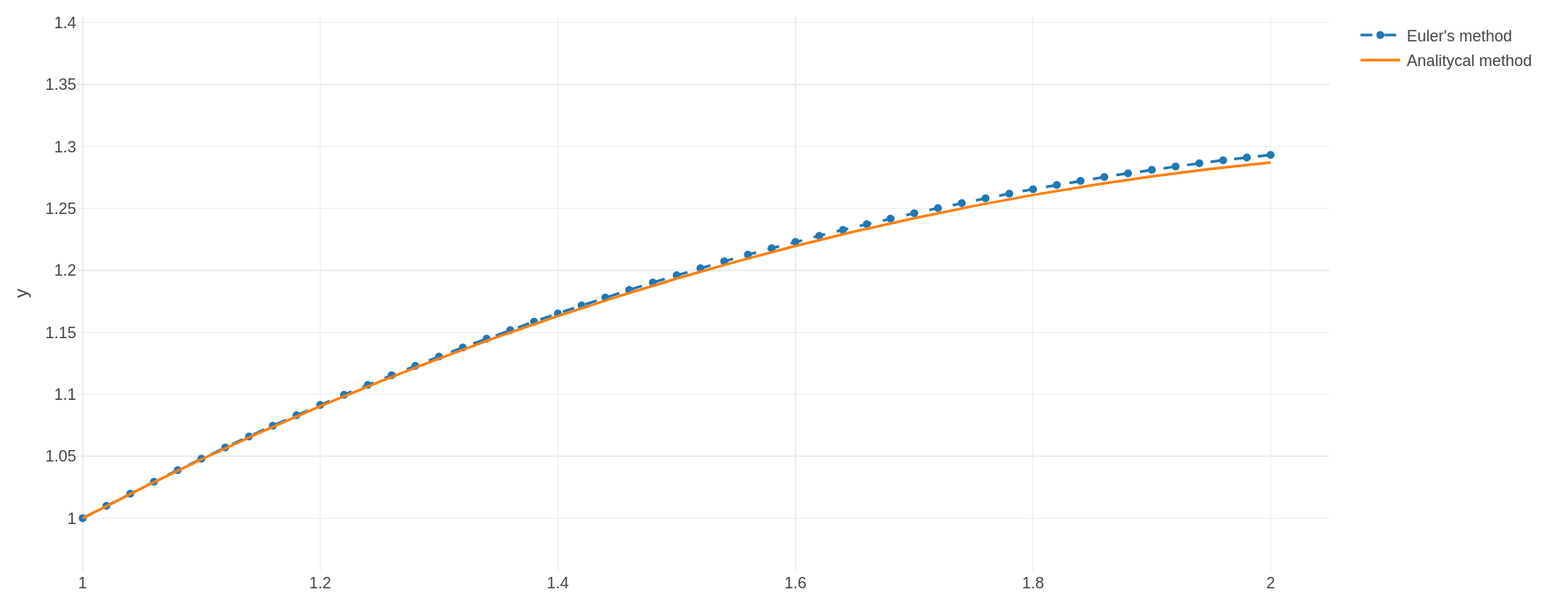


Рисунок 1 — График приблизительного и точного решения при n = 5

Рисунок 2 — График приблизительного и точного решения при n = 20

Рисунок 3 — График приблизительного и точного решения при n = 50

**СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

1. Понтрягин, Л.С. Дифференциальные уравнения и их приложения / Л.С. Понтрягин. – М.: Едиториал УРСС, 2018. – 208 c. – ISBN 978-5-354-01586-3.
2. Краткое руководство по LibreOffice [Электронный ресурс] – URL: https://libreoffice.readthedocs.io/ru/latest/ (07.05.2024)
3. Язык программирования Rust — Стив Клабник и Кэрол Николс [ Электронный ресурс ] - URL: <https://doc.rust-lang.ru/book/> (07.05.2024)

**ПРИЛОЖЕНИЕ**

**Код программы**

use plotly::{

common::{Mode, Line, DashType},

layout::{Axis, Layout},

Plot, Scatter

};

use std::io;

const A: f64 = 1.0;

const B: f64 = 2.0;

// 4xyy' - 3y^2 + x^2 = 0

// =>

// y' = ...

// for numerical

fn derivative(x: f64, y: f64) -> f64 {

((3. \* y) / (4. \* x)) - (x / (4. \* y))

}

// y(x) = sqrt(-x^2 + c\_1\*x^(3/2))

// where c\_1 = 2

fn by\_hand(x: f64) -> f64 {

(-x.powf(2.) + 2.\*x.powf(3./2.)).sqrt()

}

// Generate range

fn span\_gen(span: (f64, f64), h: f64) -> Vec<f64> {

let mut l\_span = vec![];

let mut cnt: f64 = span.0;

let tolerance = 1e-6; // Adjust tolerance as needed

while cnt < span.1 + tolerance {

l\_span.push(cnt);

cnt += h;

}

l\_span

}

// Euler's method

fn euler<F>(f: &F, x\_n: f64, y\_n: f64, h: f64) -> f64

where F:

Fn(f64, f64) -> f64

{

y\_n + h \* f(x\_n, y\_n)

}

/// F1 is a function itself

/// F2 is a solver function which applies F1

trait Solver<F1, F2>

{

fn solve(f: &F1, xs: &Vec<f64>, y\_0: f64, solver: &F2, h: f64) -> Vec<f64>;

}

struct NumericalSolver;

impl<F1, F2> Solver<F1, F2> for NumericalSolver

where F1: Fn(f64, f64) -> f64,

F2: Fn(&F1, f64, f64, f64) -> f64

{

fn solve(f: &F1, xs: &Vec<f64>, y\_0: f64, solver: &F2, h: f64) -> Vec<f64> {

let mut ys = vec![0.; xs.len()];

ys[0] = y\_0;

for i in 0..xs.len() - 1 {

ys[i + 1] = solver(f, xs[i], ys[i], h);

}

ys

}

}

struct AnalyticalSolver;

impl<F1> Solver<F1, ()> for AnalyticalSolver

where F1: Fn(f64) -> f64

{

fn solve(f: &F1, xs: &Vec<f64>, \_y\_0: f64, \_solver: &(), \_h: f64) -> Vec<f64> {

let mut ys = vec![0.; xs.len()];

for i in 0..xs.len() {

ys[i] = f(xs[i]);

}

ys

}

}

fn main() -> Result<(), Box<dyn std::error::Error>> {

println!("Please enter a step size:");

let mut input = String::new();

io::stdin().read\_line(&mut input)

.expect("Failed to read line");

let number: i32 = input.trim().parse()

.expect("Please enter a valid number");

let step: f64 = (B - A) / (number as f64);

let xs = span\_gen((A, B), step);

let xs\_exact = span\_gen((A, B), 0.05);

// Euler's values

let ys\_e = NumericalSolver::solve(&derivative, &xs, 1., &euler, step);

// Analytical values

let ys\_a = AnalyticalSolver::solve(&by\_hand, &xs, 1., &(), step);

// Exact Analytical

let ys\_aex = AnalyticalSolver::solve(&by\_hand, &xs\_exact, 1., &(), 0.05);

let mut plot = Plot::new();

let trace\_euler = Scatter::new(xs.clone(), ys\_e.clone())

.mode(Mode::LinesMarkers)

.line(Line::new().dash(DashType::Dash))

.name("Euler's method");

let trace\_analytical = Scatter::new(xs\_exact.clone(), ys\_aex.clone())

.mode(Mode::Lines)

.name("Analitycal method");

let layout = Layout::new()

.title("Euler's aproximation".into())

.x\_axis(Axis::new().title("x".into()).range(vec![0., 2.1]))

.y\_axis(Axis::new().title("y".into()).range(vec![0.9, 2.]))

.height(600)

.width(1200);

plot.set\_layout(layout);

plot.add\_trace(trace\_euler);

plot.add\_trace(trace\_analytical);

plot.show();

let mx\_error = ys\_e.iter().zip(ys\_a.iter())

.map(|(&x, &y)| f64::abs(x - y))

.max\_by(|a, b| a.partial\_cmp(b).unwrap() )

.expect("Something went wrong with finding maximum error");

println!("Max error: {}", mx\_error);

Ok(())

}