**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Факультет прикладной математики и информатики

Стаскевич Виталий Дмитриевич

Вариант 7

Отчет по лабораторной работе №4

**«Численные методы решения задачи Коши»**

студента 2 курса 13 группы

**Преподаватель**

Горбачёва Ю. Н.

Минск 2024

Постановка задачи

Решить задачу Коши для обыкновенного дифференциального уравнения первого порядка на отрезке с шагом h = 0.1 методами, указанными в варианте задания. Оценить погрешность численного решения с шагом h = 0.1 с помощью правила Рунге (для одношаговых методов). Сравнить полученные численные решения с точным решением . В одной системе координат построить график функции и график одного из полученных численных решений

Вариант 7:

,

Методы:

1. Неявный метод трапеций;
2. Явный метод средних прямоугольников
3. Предиктор-корректорный метод Адамса 2-го порядка.

Константы и функции:

# const double u0 = 0.5;

# double h = 0.1;

# const double a = 1.0;

# const double b = 2.0;

# const double eps= pow(10,-6);

# double f(double x, double u)

# {

# return (u\*u+u\*x)/(x\*x);

# }

# double fprime(double x, double u)

# {

# return (x+2.0\*u)/(x\*x);

# }

# double exactSolution(double x)

# {

# return x/(2-log(x));

# }

Неявный метод трапеций

# vector <point> implicitTrapezoidalMethod()

# {

# vector<point> solution;

# double x = a;

# double y = u0;

# solution.push\_back({x,y});

# while (x < b)

# {

# double y\_next = y;

# double x\_next = x + h;

# while(true)

# {

# double numerator = y\_next - y - ((h / 2.0) \* (f(x, y) + f(x\_next, y\_next)));

# double denominator = 1 - (h / 2.0)\*fprime(x\_next,y\_next);

# double delta = numerator/denominator;

# y\_next -= delta;

# if(abs(delta)<eps) break;

# }

# y = y\_next;

# x = x\_next;

# solution.push\_back({x, y});

# }

# return solution;

# }

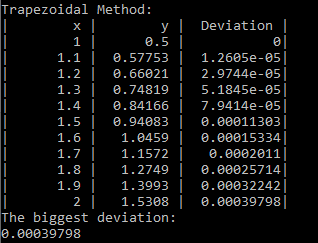
Неявный метод трапеций в общем виде выглядит следующим образом:

Для решения полученного неявного уравнения на практике используется итерационный метод Ньютона, который в общем виде выглядит следующим образом:

В нашем случае для неявного метода трапеций принимает следующий вид:

Данный метод является методом второго порядка точности, то есть

Результаты и погрешность:



Явный метод средних прямоугольников

# vector <point> midpointMethod()

# {

# vector<point> solution;

# double x = a;

# double y = u0;

# solution.push\_back({x,y});

# while (x < b)

# {

# double k1 = h \* f(x, y);

# double k2 = h \* f(x + h / 2, y + k1 / 2);

# y += k2;

# x += h;

# solution.push\_back({x, y});

# }

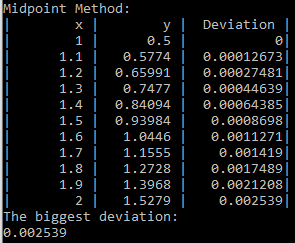
# return solution;

# }

Явный метод средних прямоугольников выражается следующей системой:

Данный метод является методом второго порядка точности, то есть

Результаты и погрешность:



Предиктор-корректорный метод Адамса 2-го порядка

# vector <point> adamsBashforthMethod()

# {

# vector<point> solution;

# double x = a;

# double y = u0;

# solution.push\_back({x,y});

# ///RK2

# double k1 = h \* f(x, y);

# double k2 = h \* f(x + h / 2.0, y + k1 / 2.0);

# y += k2;

# x += h;

# solution.push\_back({x, y});

# ///ABM

# while (x < b)

# {

# double x\_prev = solution[solution.size() - 2].x;

# double y\_prev = solution[solution.size() - 2].y;

# double f\_n = f(x, y);

# double f\_n\_prev = f(x\_prev, y\_prev);

# y += (h / 2.0) \* (3.0 \* f\_n - f\_n\_prev);

# x += h;

# solution.push\_back({x, y});

# }

# return solution;

# }

Предиктор-корректорный метод Адамса 2-го порядка является многошаговым методом, поэтому мы сначала находим значение с помощью явного метода Адамса 2-го порядка

# ///RK2

# double k1 = h \* f(x, y);

# double k2 = h \* f(x + h / 2.0, y + k1 / 2.0);

# y += k2;

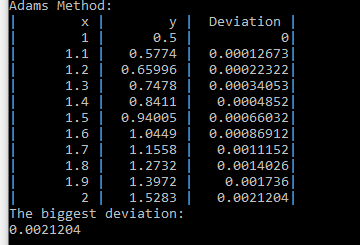
# x += h;

# solution.push\_back({x, y});

Затем уточняем полученное решение с помощью неявного метода Адамса 2-го порядка, где — полученный из явного метода Адамса:

Данный метод является методом второго порядка точности, то есть

Результаты и погрешность:

**

Правило Рунге

# struct RungeErrors{

# double MPM;

# double ITM;

# double ABM;

# };

# RungeErrors Runge(vector<point>& MPM,

# vector<point>& ITM,

# vector<point>& ABM){

# h/=2;

# vector <point> resultMPM2 = midpointMethod();

# vector <point> resultITM2 = implicitTrapezoidalMethod();

# vector <point> resultABM2 = adamsBashforthMethod();

# double maxMPM = INT\_MIN;

# double maxITM = INT\_MIN;

# double maxABM = INT\_MIN;

# for(int i = 0; i<resultMPM2.size(); i+=2)

# {

# if(abs((MPM[i/2].y-resultMPM2[i].y)/3)>maxMPM) maxMPM = abs(MPM[i/2].y-resultMPM2[i].y)/3;

# if(abs((ITM[i/2].y-resultITM2[i].y)/3)>maxITM) maxITM = abs(ITM[i/2].y-resultITM2[i].y)/3;

# if(abs((ABM[i/2].y-resultABM2[i].y)/3)>maxABM) maxABM = abs(ABM[i/2].y-resultABM2[i].y)/3;

# }

# return {maxMPM, maxITM, maxABM};

# }

Данный код реализует правило Рунге для расчета апостериорной погрешности:

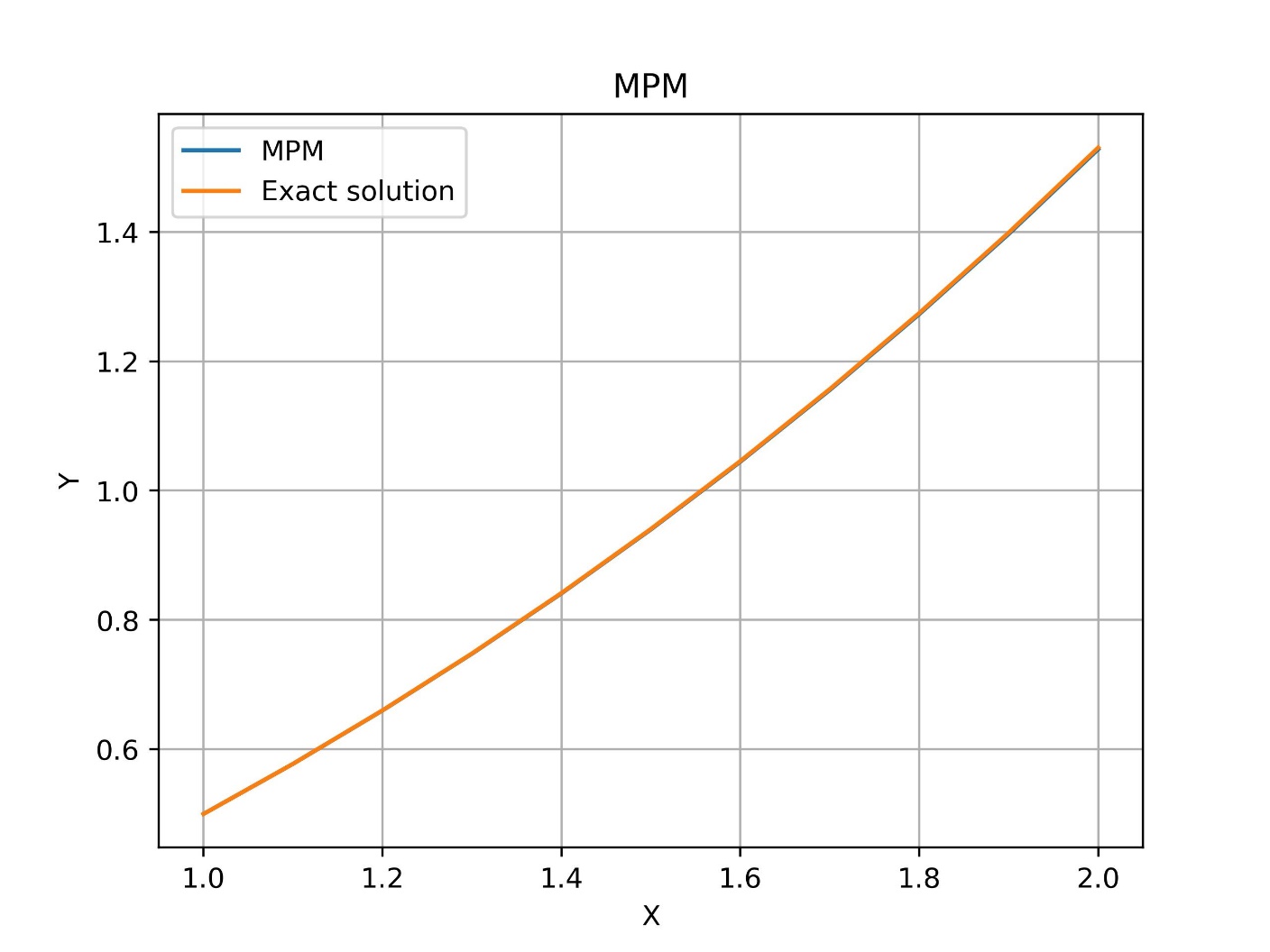
В этой формуле p – степень точности метода (в нашем случае p=2 для всех методов)

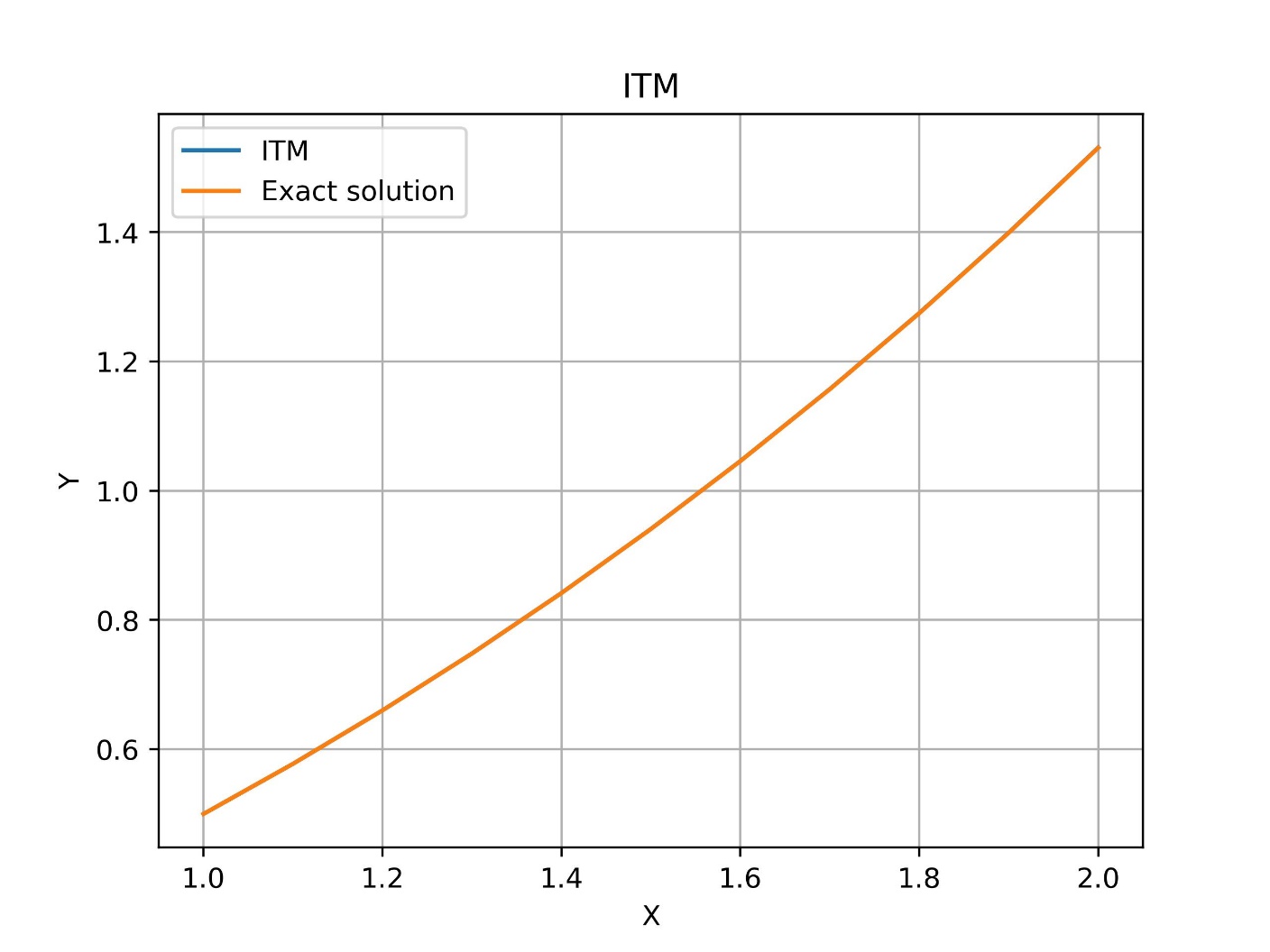
Таблица результатов

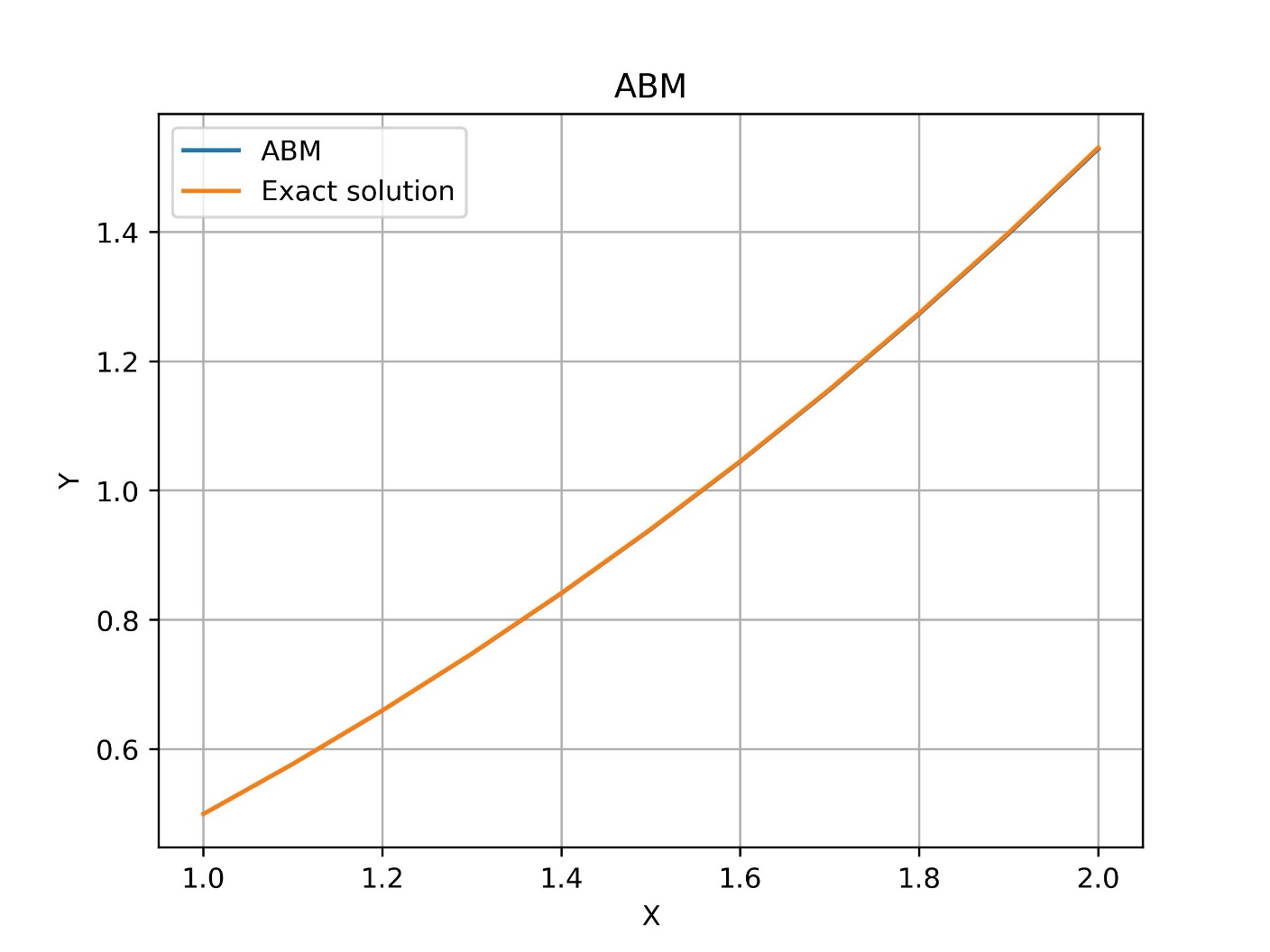
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| i | x\_i | Exact solution | Numerical solution for 0.1 step | | |
| MPM | ITM | ABM |
| u(x\_i) | y\_i | y\_i | y\_i |
| 0 | 1 | 0.5 | 0.5 | 0.5 | 0.5 |
| 1 | 1.1 | 0.577522 | 0.577395 | 0.577534 | 0.577395 |
| 2 | 1.2 | 0.660183 | 0.659908 | 0.660213 | 0.65996 |
| 3 | 1.3 | 0.748143 | 0.747697 | 0.748195 | 0.747802 |
| 4 | 1.4 | 0.841585 | 0.840941 | 0.841664 | 0.8411 |
| 5 | 1.5 | 0.940713 | 0.939843 | 0.940826 | 0.940053 |
| 6 | 1.6 | 1.04575 | 1.04463 | 1.04591 | 1.04488 |
| 7 | 1.7 | 1.15696 | 1.15554 | 1.15716 | 1.15584 |
| 8 | 1.8 | 1.27459 | 1.27285 | 1.27485 | 1.27319 |
| 9 | 1.9 | 1.39897 | 1.39685 | 1.39929 | 1.39723 |
| 10 | 2 | 1.53039 | 1.52786 | 1.53079 | 1.52827 |
| max (abs(u(x\_i)-y\_i)) | | | 0.00253896 | 0.000397981 | 0.00212043 |
| Runge Method error assessment | | | 0.000624163 | 9.95683e-05 | 0.000533855 |

Графики

# import matplotlib.pyplot as plt # Function to plot data from a file import numpy as np def plot\_from\_file(filename): with open(filename, 'r') as file: lines = file.readlines() x = [] y = [] # Process data for line in lines: values = line.split() x.append(float(values[0])) y.append(float(values[1])) with open('exact.txt', 'r') as file: lines2 = file.readlines() x\_exact = [] y\_exact = [] for line in lines2: values = line.split() x\_exact.append(float(values[0])) y\_exact.append(float(values[1])) # Plot plt.figure() # Create a new figure plt.xlabel('X') plt.ylabel('Y') plt.title(filename.removesuffix('.txt')) plt.grid(True) plt.plot(x, y, label = filename.removesuffix('.txt')) plt.plot(x\_exact, y\_exact, label='Exact solution') plt.legend() plt.savefig(filename.removesuffix('.txt')+'.jpg', dpi = 400) plt.show() # Array of filenames filenames1 = ['ABM.txt','ITM.txt','MPM.txt'] # Add more filenames as needed # Plot from each file for filename in filenames1: plot\_from\_file(filename)







Выводы

В данной работе были выполнены следующие задачи:

* Реализованы:
  + Неявный метод трапеций
  + Явный метод средних прямоугольников
  + Предиктор-корректорный метод Адамса 2-го порядка.
* Построили 3 графика для визуальной проверки полученного решения
* Построен метод касательных Ньютона для решения нелинейных уравнений
* Реализовано правило Рунге для апостериорной оценки погрешности методов
* Построили таблицу полученных решений

На основе выполненной работы можно сделать следующие выводы:

* Данные методы являются эффективными методами нахождения приблизительного решения ОДУ с задачей Коши. С увеличением степени точности метода возрастает и его эффективность, также эффективность метода увеличивается при использовании неявных методов.
* Убедили, что по правилу Рунге можно лишь приблизительно найти ожидаемую погрешность метода.