**Министерство науки и высшего образования РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ИС**

отчет

**по лабораторной работе №9**

**по дисциплине «Конструирование программ»**

Тема: Методы интегрирования обыкновенных дифференциальных уравнений

|  |  |  |
| --- | --- | --- |
| Студент гр. 8363 |  | Нерсисян А.С. |
| Преподаватель |  | Копыльцов А.В. |

Санкт-Петербург

2020

**Цель работы.**

Написать программу, которая решает заданное дифференциальное уравнение первого порядка методом Эйлера и Рунге – Кутты четвертого порядка на указанном отрезке с указанным шагом и оценивает погрешность интегрирования по правилу Рунге.

**Основные теоретические положения.**

Существует большое число методов приближенного решения дифференциальных уравнений, основанных на самых различных идеях. Численные методы дают приближенное решение  в виде таблицы значений  в точках .

Простейшим методом численного интегрирования дифференциального уравнения первого порядка , удовлетворяющего начальному условию , является метод Эйлера. В нем величины  вычисляются по формуле



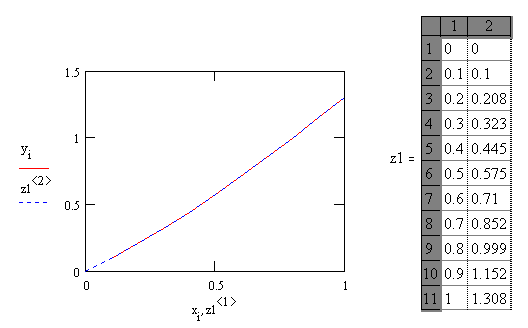
Метод Эйлера относится к группе одношаговых, в которых для расчета точки  требуется информация только о последней вычисленной точке . Геометрическая интерпретация метода изложена в подразд. 7.4.

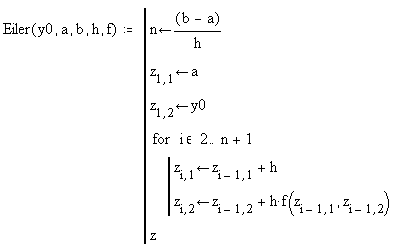
В среде Mathcad имеется тринадцать встроенных функций решения дифференциальных уравнений и систем ( задача Коши, краевая задача, уравнения в частных производных). Самая употребительная из них - **rkfixed**, в которую заложен метод Рунге – Кутты четвертого порядка с постоянным шагом. Подпрограммы для метода Эйлера нет из-за его низкой точности. Формулы метода Эйлера настолько просты, что вычисления по ним можно организовать с помощью дискретной переменной.

Рассмотрим **пример.** Решим задачу Коши для дифференциального уравнения  при 

Введем начало программы

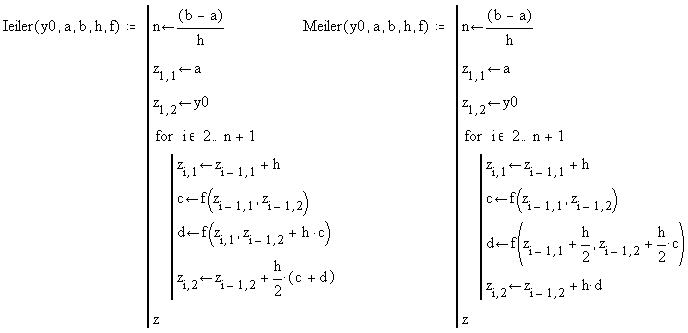


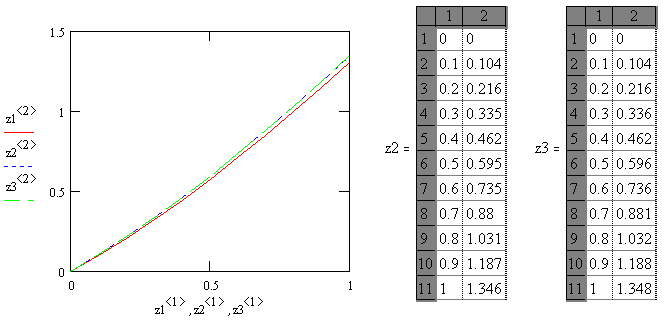
Аналогичного результата можно достигнуть, введя подпрограмму, реализующую формулы (7.4.1) метода Эйлера:





Минимальная переделка подпрограммы позволяет запрограммировать исправленный и модифицированный метод Эйлера по формулам (7.4.4) и (7.4.6).



Обратимся теперь к средствам пакета Mathcad. Для решения обыкновенных «неособенных» дифференциальных уравнений здесь используются две функции **rkfixed** и **Rkadapt**, реализующие метод Рунге – Кутты четвертого порядка с постоянным и переменным шагом. Набор параметров у этих подпрограмм одинаков: , где    
- начало и конец интервала интегрирования, - число точек и, следовательно, шаг , - правая часть дифференциального уравнения. Несмотря на то что при решении дифференциального уравнения функция **Rkadapt** использует переменный шаг, она тем не менее представляет ответ для  точек, находящихся на одинаковом расстоянии друг от друга, равном .

Вводим следующую часть программы:

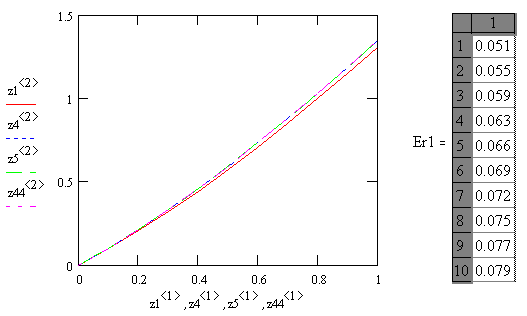








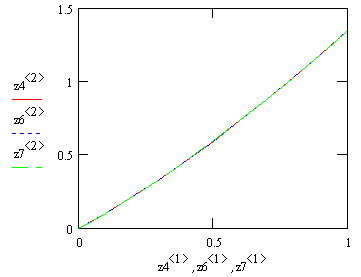


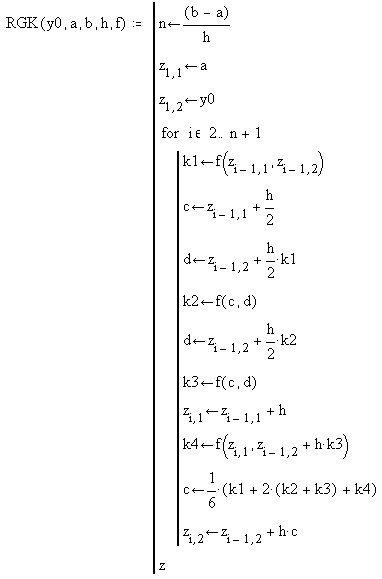
Обращение к **rkfixed** с  и  сделано для оценки погрешности интегрирования по правилу Рунге (7.5.2). - оценка погрешности.

На данном учебном примере, конечно, невозможно оценить выгоды использования функции **Rkadapt** вместо **rkfixed**. В более сложных случаях **Rkadapt** решает уравнение более точно и быстро. Даже в этом примере точность решения по **Rkadapt** выше, чем по **rkfixed**. Это видно из следующей таблицы:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 |
|  | 0.1040989 | 0.2161356 | 0.3357322 | 0.4625076 | 0.5960572 |
|  | 0.1040990 | 0.2161359 | 0.3357326 | 0.4625081 | 0.5960578 |
|  | 0.1040990 | 0.2161359 | 0.3357326 | 0.4625081 | 0.5960571 |
|  | 0.6 | 0.7 | 0.8 | 0.9 | 1.0 |
|  | 0.7359363 | 0.8816484 | 1.0326377 | 1.1882891 | 1.3479326 |
|  | 0.7359370 | 0.8816491 | 1.0326386 | 1.1882900 | 1.3479335 |
|  | 0.7359371 | 0.8816492 | 1.0326386 | 1.1882901 | 1.3479336 |

В заключение приведем подпрограмму **RGK**, реализующую формулы (7.4.13) метода Рунге – Кутты четвертого порядка. Далее, как и в предыдущем случае, произведена оценка погрешности по правилу Рунге и даны графики полученных решений. Программа метода Рунге – Кутты четвертого порядка **(RGK)** приведена после графиков проинтегрированных функций.





**Экспериментальные результаты.**

**Задание № 1**

Решить заданное дифференциальное уравнение первого порядка методом Эйлера и Рунге – Кутты четвертого порядка на отрезке с шагом и оценить погрешность интегрирования по правилу Рунге.:

**Дано:** Вариант 11

**Обработка результатов эксперимента.**

**Задание № 1. решение:**

#include <iostream>

#include <conio.h>

using namespace std;

double cos(double x)

{

double y = 1, prev, a = 1, fact = 1;

int i=1;

const double E = 0.0001;

do

{

prev = y;

a \*= x\*x;

for (int j = 2 \* (i - 1) + 1; j <= 2 \* i; ++j)

fact \*= j;

if (i % 2) y -= a / fact;

else y += a / fact;

++i;

} while (abs(y - prev) > E);

return y;

}

double func(double x, double y)

{

return cos(y) / (1.25 + x) - 0.5\*y\*y;

}

int main()

{

setlocale(LC\_ALL, "rus");

double x1 = 0, x2 = 1, h = 0.1, x = x1 + h, y = 0, prev;

cout << "Метод Эйлера" << endl;

cout << "x = 0; y = 0" << endl;

do

{

y += h\*func(x, y);

cout << "x = " << x << "; y = " << y << endl;

x += h;

} while (x <= x2);

cout << "Метод Рунге - Кутты" << endl;

double A[4][3];

A[0][0] = x1;

A[0][1] = 0;

cout << "x = 0; y = 0" << endl;

do

{

A[2][0] = A[1][0] = A[0][0] + h / 2;

A[3][0] = A[0][0] + h;

A[0][2] = func(A[0][0], A[0][1]);

A[1][1] = A[0][1] + h / 2 \* A[0][2];

A[1][2] = func(A[1][0], A[1][1]);

A[2][1] = A[0][1] + h / 2 \* A[1][2];

A[2][2] = func(A[2][0], A[2][1]);

A[3][1] = A[0][1] + h \* A[2][2];

A[3][2] = func(A[3][0], A[3][1]);

A[0][1] += h\*(A[0][2] + 2 \* A[1][2] + 2 \* A[2][2] + A[3][2])/6;

A[0][0] += h;

cout << "x = " << A[0][0] << "; y = " << A[0][1] << endl;

} while (A[0][0] <= x2-h);

\_getch();

return 0;

}

**Выводы.**

В ходе выполнения данной лабораторной была написана программа, которая решает заданное дифференциальное уравнение первого порядка методом Эйлера и Рунге – Кутты четвертого порядка на указанном отрезке с указанным шагом и оценивает погрешность интегрирования по правилу Рунге.