**Министерство науки и высшего образования РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ИС**

отчет

**по лабораторной работе №10**

**по дисциплине «Конструирование программ»**

Тема: Численное интегрирование систем дифференциальных уравнений первого порядка

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

Санкт-Петербург

2020

**Цель работы.**

Написать программу, которая решает задачу Коши для системы дифференциальных уравнений на заданном отрезке.

**Основные теоретические положения.**

Задача Коши для отдельного дифференциального уравнения решается сравнительно редко. Чаще приходится интегрировать систему обыкновенных дифференциальных уравнений. Дифференциальные уравнения высших порядков также легко сводятся к системе уравнений. Если задано уравнение  с начальными условиями , то стандартная замена переменных  приводит это уравнение к системе  дифференциальных уравнений первого порядка:



с начальными условиями 

Интегрирование систем дифференциальных уравнений в пакете Mathcad проводится теми же функциями, которые описаны в предыдущей лабораторной работе, поскольку задача Коши для уравнений сводится при использовании этих же функций к решению задачи Коши для систем.

Численное решение этой задачи состоит в построении таблицы приближенных значений  решения  на отрезке  в узлах сетки . Пусть



где - решение системы, - вектор начальных условий, - вектор правых частей системы. Тогда исходная система дифференциальных уравнений первого прядка (7.11.1) в векторной форме перепишется в виде



Рассмотрим **пример.** Пусть 



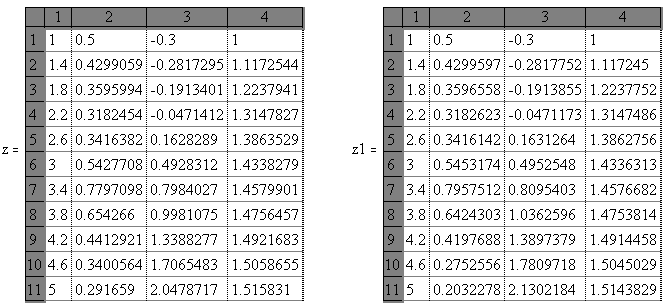
Воспользуемся всеми встроенными программами интегрирования систем обыкновенных дифференциальных уравнений, которыми располагает система Mathcad. Введем программу вычислений.





Последняя функция **Bulstoer** решает задачу Коши на отрезке  методом Булирша – Штера. Он является методом рациональной экстраполяции, главным достоинством которого является то, что для достижения высокой точности не требуется многократного перевычисления правых частей интегрируемых уравнений. Это особенно удобно, когда правые части уравнений сложны. Основная идея метода рациональной экстраполяции заключается в следующем. Сначала находится некоторое приближенное решение рассматриваемых уравнений в точках , например, по методу Эйлера; затем рассчитывается улучшенное приближение путем экстраполяции рациональными функциями, например, многочленами по специальным вычислительным схемам.

Для сложных систем, насчитывающих десятки дифференциальных уравнений, основной выигрыш при применении метода Булирша – Штера, помимо точности, заключается в заметном сокращении времени вычислений.

Приведем результаты вычислений по всем трем используемым программам и графики решений.

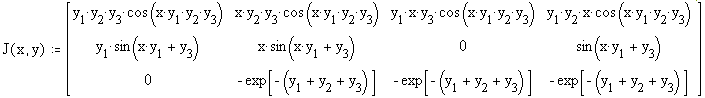
Видно, что функция **rkfixef** по  на конце интервала интегрирования дает уже неудовлетворительный результат, что объясняется слишком большим шагом интегрирования 

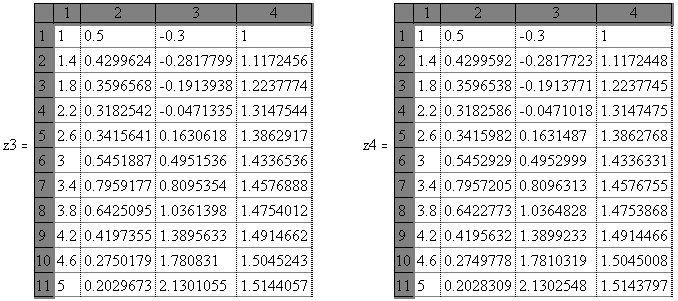
Рассмотрим еще один метод интегрирования так называемых «жестких» систем дифференциальных уравнений. Эти системы характерны тем, что, несмотря на медленное изменение функций, определяющих решение, расчеты приходится вести с очень мелким шагом. Все попытки увеличить шаг и тем самым уменьшить время решения задачи приводят лишь к катастрофически большому росту погрешности. Термин «жесткий» происходит из механики, где численное решение некоторых систем дифференциальных уравнений требует разного шага интегрирования по разным искомым функциям. Численное решение «жестких» задач требует применения специальных неявных методов.

В пакете Mathcad подобные системы решаются с помощью функций **Stiffb** и **Stiffr**, которые имеют те же параметры, что и функция **rkfixed**. Кроме того, задается информация о скорости изменения вектора правых частей уравнений (7.11.2), то есть матрица Якоби правых частей:



Введем следующую часть программы:





Функция **Stiffb** использует алгоритм Булирша – Штера, функция **Stiffr** – алгоритм Розенброка, подробно описанный в специальной литературе.

Приведем, наконец, текст подпрограммы, реализующий метод Рунге – Кутты четвертого порядка с постоянным шагом для систем дифференциальных уравнений с параметрами, полностью аналогичными параметрам функции **rkfixed**:



Сравнение матриц  показывает, что результаты расчета по подпрограммам **rkfixed** и **RGKsyst** практически одинаковы.

**Экспериментальные результаты.**

**Задание № 1**

С помощью любой из разобранных в лабораторной работе подпрограмм решить задачу Коши для системы дифференциальных уравнений на заданном отрезке:

**Дано:** Вариант 11

**Обработка результатов эксперимента.**

**Задание № 1. решение:**

#include <iostream>

#include <conio.h>

using namespace std;

double exp(double x)

{

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

int i = 1;

const double E = 0.00001;

do

{

prev = y;

a \*= x;

fact \*= i;

++i;

y += a / fact;

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

return y;

}

double arctg(double x)

{

double y = x, prev, a = x;

int i = 2;

const double E = 0.00001;

do

{

prev = y;

a \*= x\*x;

if (i % 2) y += a / (2 \* i - 1);

else y -= a / (2 \* i - 1);

++i;

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

return y;

}

double sin(double x)

{

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

int i = 2;

const double E = 0.00001;

do

{

prev = y;

a \*= x\*x;

fact \*= i\*(i + 1);

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

else y -= a / fact;

++i;

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

return y;

}

double f1(double x, double y1, double y2, double y3)

{

return arctg(x\*y1\*y3);

}

double f2(double x, double y1, double y2, double y3)

{

return sin(arctg(y1\*y3));

}

double f3(double x, double y1, double y2, double y3)

{

return exp(-y1\*y3\*y2);

}

int main()

{

setlocale(LC\_ALL, "rus");

double x1 = 1, x2 = 4, h = 0.1, y1 = 0, y2 = -0.3, y3 = 1, x = x1;

cout << "Метод Эйлера" << endl;

cout << "x = 1; y1 = 0; y2 = -0.3; y3 = 1" << endl;

do

{

x += h;

y1 += h\*f1(x, y1, y2, y3);

y2 += h\*f2(x, y1, y2, y3);

y3 += h\*f3(x, y1, y2, y3);

cout << "x = " << x << "; y1 = " << y1 << "; y2 = " << y2 << "; y3 = " << y3 << endl;

} while (x <= x2);

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

double k[3][4];

y1 = 0;

y2 = -0.3;

y3 = 1;

x = 1;

cout << "x = 1; y1 = 0; y2 = -0.3; y3 = 1" << endl;

do

{

k[0][0] = f1(x, y1, y2, y3);

k[1][0] = f2(x, y1, y2, y3);

k[2][0] = f3(x, y1, y2, y3);

x += h / 2;

k[0][1] = f1(x, y1 + h\*k[0][0] / 2, y2 + h\*k[1][0] / 2, y3 + h\*k[2][0] / 2);

k[1][1] = f2(x, y1 + h\*k[0][0] / 2, y2 + h\*k[1][0] / 2, y3 + h\*k[2][0] / 2);

k[2][1] = f3(x, y1 + h\*k[0][0] / 2, y2 + h\*k[1][0] / 2, y3 + h\*k[2][0] / 2);

k[0][2] = f1(x, y1 + h\*k[0][1] / 2, y2 + h\*k[1][1] / 2, y3 + h\*k[2][1] / 2);

k[1][2] = f2(x, y1 + h\*k[0][1] / 2, y2 + h\*k[1][1] / 2, y3 + h\*k[2][1] / 2);

k[2][2] = f3(x, y1 + h\*k[0][1] / 2, y2 + h\*k[1][1] / 2, y3 + h\*k[2][1] / 2);

x += h / 2;

k[0][3] = f1(x, y1 + h\*k[0][2], y2 + h\*k[1][2], y3 + h\*k[2][2]);

k[1][3] = f2(x, y1 + h\*k[0][2], y2 + h\*k[1][2], y3 + h\*k[2][2]);

k[2][3] = f3(x, y1 + h\*k[0][2], y2 + h\*k[1][2], y3 + h\*k[2][2]);

y1 += h \* (k[0][0] + 2 \* k[0][1] + 2 \* k[0][2] + k[0][3]) / 6;

y2 += h \* (k[1][0] + 2 \* k[1][1] + 2 \* k[1][2] + k[1][3]) / 6;

y3 += h \* (k[2][0] + 2 \* k[2][1] + 2 \* k[2][2] + k[2][3]) / 6;

cout << "x = " << x << "; y1 = " << y1 << "; y2 = " << y2 << "; y3 = " << y3 << endl;

} while (x < x2 - h);

\_getch();

return 0;

}

**Выводы.**

В ходе выполнения данной лабораторной была написана программа, которая решает задачу Коши для системы дифференциальных уравнений на заданном отрезке.