От автора

В варианте задания дан пример титульной страницы, где данный отчет следует под номером два. В свою очередь, мною предыдущая практическая работа была разделена на две, одна из которых уже имеет этот номер (№ 2). Таким образом, не трудно заметить, что сбита нумерация. Чтобы продолжить естественную закономерность, этот отчет и следующий за ним, обозначим как третий (№ 3) и четвертый (№ 4) соответственно. То есть продолжим тенденцию в нумерации и разбиении одной практической работы на несколько подработ.

Стоит отметить, что предыдущий отчет был разделен по весомым причинам. По моему мнению, два разных метода решения поставленной задачи и две разные программы должны быть в разных практических работах.

Оглавление

Цели и постановка задачи

Цель работы

В данной работе требуется освоить методы Рунге-Кутта второго и четвертого порядка точности, применяемые для численного решения задачи Коши для обыкновенного дифференциального уравнения первого порядка, разрешенного относительно производной. Или для системы ОДУ первого порядка, разрешенных относительно производных.

Постановка задачи

Дано обыкновенное дифференциальное уравнение первого порядка, разрешенное относительно производной вида:

$$\frac{dy}{dx} = f(x, y),\ x_0 < x, \tag{1}$$

с дополнительным начальным условием, заданным в точке $x = x_0$:

$$y(x_0) = y_0. \tag{2}$$

Причем правая часть уравнения (1) такая, что гарантирует единственность и существование решения задачи Коши(1)(2), то есть:

1. $f(x, y)$ непрерывна в $\Pi = \{(x, y) : |x - x_0| \leqslant T,\ |y - y_0| \leqslant A,\ T > 0,\ A > 0\}$,
2. $f(x, y)$ удовлетворяет условию Липшица в Π по y ($f(x, y) \in Lip(\Pi)$):

$$|f(x, y_1) - f(x, y_2)| \leqslant L\,|y_1 - y_2|\,,\ \forall (x, y_1),\ (x, y_2) \in \Pi,$$

3. $|f(t, y)| \leqslant M, \quad \forall\,(t, y) \in \Pi, M > 0.$

В том случае, если рассматривается не одно дифференциальное уравнение вида (1), а система ОДУ первого порядка, разрешенных относительно производных неизвестных функций, то соответствующая задача Коши имеет вид (на примере двух дифференциальных уравнений):

$$\begin{cases} \dfrac{dy_1}{dx} = f_1(x, y_1, y_2) \\ \dfrac{dy_2}{dx} = f_2(x, y_1, y_2) \end{cases} \quad x < x_0 \tag{3}$$

с дополнительными начальными условиями в точке $x = x_0$:

$$y_1(x_0) = y_1^{(0)}, \quad y_2(x_0) = y_2^{(0)} \tag{4}$$

Также предполагается, что правые части уравнений из системы (3) удовлетворяют пунктам $1-3$, то есть гарантируют существование и единственность решения задачи Коши(3)(4) для системы ОДУ первого порядка, разрешенных относительно неизвестных функций.

Вариантом задания предусмотрены следующие задачи Коши:

1.
$$\frac{dy}{dx} = (y - y^2)x, \ y = y(x) \tag{5}$$
$$y(0) = 3, \tag{6}$$

где $\quad y(x) = \dfrac{1}{1 - \dfrac{2}{3} e^{\dfrac{-x^2}{2}}}$ – точное решение задачи Коши(5)(6).

2.
$$\begin{cases} \dfrac{dy_1}{dx} = \sqrt{x^2 + 1,1 \cdot y_1} + y_2 \\ \dfrac{dy_2}{dx} = \cos(2,1 \cdot y_2) + y_1 \end{cases} \tag{7}$$

$$y_1(0) = 0,5, \quad y_2(0) = 1. \tag{8}$$

Численное решение задачи Коши

Дальнейшая теория методов решения задачи Коши будет описана для обыкновенных дифференциальных уравнений первого порядка, разрешенных относительно производной. Понятно, что эту теорию можно провести для систем ОДУ первого порядка, разрешенных относительно производных неизвестных функций, аналогичным образом.

Метод Эйлера

Итак, нам нужно найти решение задачи Коши(1)(2) на отрезке $[x_0, x_0 + l]$ длины l. Рассмотрим задачу Коши в эквивалетном виде (для простоты дальнейших объяснений):

$$u' = f(x, u), \tag{9}$$

$$u(x_0) = u_0. \tag{10}$$

Причем предполагается, что $f(x, u)$ удовлетворяет тем же условиям $(1 - 3)$ для существования и единственности задачи Коши на рассматриваемом отрезке.

Пусть $n \in \mathbb{Z}$ – некоторое число. Возьмем шаг $h = \dfrac{l}{n}$ и образуем на отрезке сетку

$$x_i = x_0 + ih,\ 0 \leqslant i \leqslant n. \tag{11}$$

Сопоставим задаче (9)(10) на отрезке разностную задачу

$$\frac{y_{i+1} - y_i}{h} = f(x_i, y_i),\ 0 \leqslant i \leqslant n - 1, \tag{12}$$

$$y_0 = u_0. \tag{13}$$

Здесь производная $u'(x)$ из уравнения (9) заменена правой разностной производной. Также остается неизменным начальное условие (10).

Уравнение (12) является разностным уравнением первого порядка, которое принято называть ***схемой Эйлера***. Его можно переписать в виде рекуррентного соотношения:

$$y_{i+1} = y_i + hf(x_i, y_i),\ 0 \leqslant i \leqslant n - 1. \tag{14}$$

Это позволяет последовательно рассчитать все значения сеточной функции $\{y_i\}$, решив тем самым задачу (10)(11). Такую разностную схему называют *явной*.

Точность решения исходной задачи Коши

Рассмотрим решение задачи (9)(10) в точках сетки (11), образовав из функции непрерывного аргумента сеточную функцию $u_i = u(x_i)$, и сравним ее с рассчитанной сеточной функцией y_i. Для этого образуем две сеточные функции – z, ψ:

$$z_i = y_i - u_i, 0 \leqslant i \leqslant n, \tag{15}$$

$$\psi_i = \frac{(u_{i+1} - u_i)}{h} - f(x_i, u_i), 0 \leqslant i \leqslant n-1. \tag{16}$$

Первая функция характеризует разницу между рассчитанными числами y_i и решением $u(x)$ задачи Коши(9)(10) в точках сетки x_i. В соответствии с этим сеточную функцию называют *погрешностью решения*.

Функция ψ (16) получается в результате подстановки решения дифференциального уравнения (9) в разностное уравнение (12). Если бы уравнения совпадали, то мы бы получили нуль. Сеточную функцию ψ, характеризующую степень близости дифференциального и разностного уравнений, называют *погрешностью аппроксимации* схемы на решении.

Если установить связь между сеточными функциями z и ψ, можно получить следующее выражение:

$$||z||_c \ \leqslant \ le^{CL}||\psi||_c \tag{17}$$

где

$$||\psi||_c = \max_{0 \leqslant i \leqslant n-1} |\psi_i|, \ |\psi_i| \leqslant ||\psi||_c$$

$C > 0, \ C \in \mathbb{R}$, т.ч. $\left|\dfrac{\partial f}{\partial u}(x,u)\right| \ \leqslant \ C$ в интересующей нас области изменения аргументов, l – длина отрезка, на котром рассматривается решение исходной задачи (9)(10).

Выражение (17) определяет оценку погрешности решения через оченку погрешности аппроксимации схемы с коэффициентом, который не зависит от шага h. Поэтому выражение (17) дает возможность получить важный результат: *чем лучше разностное уравнение аппроксимирует дифференциальное, тем меньше погрешность решения*.

Если оценить норму погрешности аппроксимации уравнения $||\psi||_c$, можно получить следующие выражения:

$$||\psi||_c \leqslant \frac{M}{2}h, \ ||z||_c \leqslant \frac{Ml}{2}e^{Cl}h, \tag{18}$$

где $M > 0$, т.ч. $|u(x)| \ \leqslant \ M, \ x_0 \leqslant x \leqslant x_0 + l$.

Неравенства (18) показывают, что при $h \to 0$ погрешность аппроксимации уравнения и с ней неравенством (17) погрешность решения стремятся к нулю со скоростью h. В связи с этим метод Эйлера называют *методом первого порядка точности* относитльно h.

Метод Рунге-Кутта

Если поставить вопрос о повышении точности разностного метода Эйлера, можно получить рекуррентное соотношение для схемы Эйлера:

$$\frac{y_{i+1} - y_i}{h} = f(x_i, y_i) + \frac{1}{2}\left\{\frac{\partial f}{\partial x}(x_i, y_i) + \frac{\partial f}{\partial y}(x_i, y_i) f(x_i, y_i)\right\} h, \tag{19}$$

которое получается при разложении функции $u(x_i + h)$ из уравнения (9) по формуле Тейлора.

Главная идея метода Рунге-Кутта состоит в том, чтобы *приближенно заменить ее на сумму значений функции f в двух разных точках с точностью до членов порядка h^2*. С этой целью положим

$$f(x_i, y_i) + \frac{1}{2}\left\{\frac{\partial f}{\partial x}(x_i, y_i) + \frac{\partial f}{\partial y}(x_i, y_i)\right\} h = \beta f(x_i, y_i) + \alpha f(x_i + \gamma h, y_i + \delta h) + O\left(h^2\right), \quad (20)$$

где $\alpha,\ \beta,\ \gamma,\ \delta$ – четыре свободных параметра, которые нужно подобрать так, чтобы правая часть равнялась левой с нужной степенью точности.

Разложим функцию $f(x_i + \gamma h, y_i + \delta h)$ по степеням h по формуле Тейлора:

$$f(x_i + \gamma h, y_i + \delta h) = f(x_i, y_i) + \left\{\gamma \frac{\partial f}{\partial x}(x_i, y_i) + \delta \frac{\partial f}{\partial y}(x_i, y_i)\right\} h + O\left(h^2\right). \quad (21)$$

Подставим разложение (21) в формулу (20) и приравняем слева и справа члены, не содержащие h и соержащие h в первой. В результате получим для четырех параметров три уравнения:

$$\alpha + \beta = 1,\ \alpha\gamma = \frac{1}{2},\ \alpha\delta = \frac{1}{2} f(x_i, y_i).$$

Эти уравнения позволяют выразить параметры $\beta,\ \gamma,\ \delta$ через α:

$$\beta = 1 - \alpha,\ \gamma = \frac{1}{2\alpha} f(x_i, y_i).$$

Заменяя с помощью (20) левую часть уравнения (19) и отбрасывая члены порядка $O(h^2)$, получаем одно параметрическое семейство разностных схем Рунге-Кутта:

$$\frac{y_{i+1} - y_i}{h} = (1 - \alpha) f(x_i, y_i) + \alpha \left(x_i + \frac{h}{2\alpha},\ y_i + \frac{h}{2\alpha} f(x_i, y_i)\right). \quad (22)$$

Это уравнение можно записать в более удобном для вычислений виде рекуррентного соотношения:

$$\boxed{y_{i+1} = y_i + \left[(1 - \alpha) f(x_i, y_i) + \alpha f\left(x_i + \frac{h}{2\alpha},\ y_i + \frac{h}{2\alpha} f(x_i, y_i)\right)\right] h.} \quad (23)$$

Наиболее удобные разностные схемы этого семейства соответствуют двум значениям параметра α: $\left[\begin{array}{l} \alpha = \frac{1}{2} \\ \alpha = 1 \end{array}\right.$. При $\alpha = \frac{1}{2}$ рекуррентная формула (23) принимает вид:

$$y_{i+1} = y_i + \frac{h}{2}\left(f(x_i, y_i) + f\left(x_i + h,\ y_i + h f(x_i, y_i)\right)\right). \quad (24)$$

Она определяет следующую процедуру расчета y_{i+1}. Сначала делается шаг по схеме h по схеме Эйлера и вычисляется величина

$$\tilde{y}_{i+1} = y_i + f(x_i, y_i) h. \quad (25)$$

Затем находится значение функции f в точке $(x_{i+1},\ \tilde{y}_{i+1})$, составляется полусумма $\dfrac{f(x_i, y_i) + f(x_{i+1},\ \tilde{y}_{i+1})}{2}$ и проводится окончательный расчет величины

$$y_{i+1} = y_i + \frac{f(x_i, y_i) + f(x_i + 1, \tilde{y}_i + 1)}{2} h. \tag{26}$$

Такая схема вычислений называется ***«предиктор – корректор»***, или, буквально, «предсказание – исправление» («счет – пересчет»). Вычисление $\tilde{y}_{i+1}$ по схеме Эйлера (25) – это грубое предсказание результата. Вторичный расчет (26), сделанный на основании первого, является уточнением результата, его коррекцией.

При $\alpha = 1$ рекуррентная формула (23) принимает вид:

$$y_{i+1} = y_i + f\left(x_i + \frac{h}{2},\ y_i + \frac{h}{2} f(x_i, y_i)\right). \tag{27}$$

Следует отметить, что процедура расчета приближенного решения задачи Коши(9)(10) по схеме (22) по сравнению со схемой Эйлера усложняется: теперь на каждом шаге функцию $f(x, u)$ приходится считать не один, а два раза. Однако такое усложнение оказывается оправданным благодаря более высокой точности метода.

Как в методе Эйлера можно получить оценки *для погрешности* решения и *погрешности аппроксимации* схемы на решении:

$$||\psi||_c \leqslant Mh^2,\ ||z||_c \leqslant Mle^{Cl}h^2, \tag{28}$$

где константы M, C, l определяют те же понятия, что и в методе Эйлера.

При $h \to 0$ погрешность аппроксимации уравнения и, как следствие, погрешность решения стремятся к нулю со скоростью h^2. Это означает, разностное уравнение (22), по схеме Рунге-Кутта, имеет ***второй порядкок точности*** относительно h.

Также можно построить разностную схему метода Рунге-Кутта ***четвертого порядка точности***:

$$\boxed{\frac{y_{i+1} - y_i}{h} = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)}, \tag{29}$$

где

$$k_1 = f(x_i, y_i),\ k_2 = f\left(x_i + \frac{h}{2},\ y_i + \frac{h}{2}k_1\right),\ k_3 = f\left(x_i + \frac{h}{2},\ y_i + \frac{h}{2}k_2\right),\ k_4 = f(x_i + h, y_i + hk_3). \tag{30}$$

Если в схеме второго порядка точности на каждом шаге функцию $f(x, y)$ приходилось вычислять два раза, то здесь – четыре раза. Однако это усложнение схемы расчета окупается высокой точностью.

Аналогичные рекурретные соотношения для систем ОДУ

Рекуррентное соотношение Рунге-Кутта для систем *второго порядка*:

$$\begin{aligned} y_1^{i+1} &= y_1^i + \left[(1-\alpha) f_1(x_i, y_1^i, y_2^i) + \alpha f_1\left(x_i + \frac{h}{2\alpha},\ y_1^i + \frac{h}{2\alpha} f_1(x_i, y_1^i, y_2^i),\ y_2^i + \frac{h}{2\alpha} f_2(x_i, y_1^i, y_2^i)\right)\right] h \\ y_2^{i+1} &= y_2^i + \left[(1-\alpha) f_2(x_i, y_1^i, y_2^i) + \alpha f_2\left(x_i + \frac{h}{2\alpha},\ y_1^i + \frac{h}{2\alpha} f_1(x_i, y_1^i, y_2^i),\ y_2^i + \frac{h}{2\alpha} f_2(x_i, y_1^i, y_2^i)\right)\right] h \end{aligned} \tag{31}$$

Рекуррентное соотношение Рунге-Кутта для систем *четвертого порядка*:

$$\begin{aligned}
\frac{y_{i+1}-y_i}{h} &= \frac{1}{6}(k_1+2k_2+2k_3+k_4)\\
\frac{y_{i+1}-y_i}{h} &= \frac{1}{6}(m_1+2m_2+2m_3+m_4)
\end{aligned} \tag{32}$$

где

$$\begin{aligned}
k_1 &= f_1(x_i,\ y_1^i,\ y_2^i)\\
m_1 &= f_2(x_i,\ y_1^i,\ y_2^i)\\
k_2 &= f_1\left(x_i+\frac{h}{2},\ y_1^i+\frac{h}{2}k_1,\ y_2^i+\frac{h}{2}m_1\right)\\
m_2 &= f_2\left(x_i+\frac{h}{2},\ y_1^i+\frac{h}{2}k_1,\ y_2^i+\frac{h}{2}m_1\right)\\
k_3 &= f_1\left(x_i+\frac{h}{2},\ y_1^i+\frac{h}{2}k_2,\ y_2^i+\frac{h}{2}m_2\right)\\
m_3 &= f_2\left(x_i+\frac{h}{2},\ y_1^i+\frac{h}{2}k_2,\ y_2^i+\frac{h}{2}m_2\right)\\
k_4 &= f_1(x_i+h,\ y_1^i+hk_3,\ y_2^i+m_3)\\
m_4 &= f_2(x_i+h,\ y_1^i+hk_3,\ y_2^i+m_3)
\end{aligned} \tag{33}$$

Описание алгоритма программы

В варианте задания требуется реализовать метод Рунге-Кутта решениия ОДУ и системы ОДУ. Для этих целей программа разделена на 2 части. Часть с чесленным решением задачи Коши для системы ОДУ второго порядка, разрешенных относительно производных неизвестных функций на ходится после $Jump2$.

Программа построена так, чтобы пользователь мог получить решение данной задачи (предварительно задав сами уравнения в функциях $ff1$ и $ff2$ (для одного уравнения – f)), используя только командную строку. Как именно работать с командной строкой и как понять какие возможности пользователю дает программа, можно прочитать при вызове функции $print\_help\_info$. Для ее вызова, нужно или вбить неправильные входные данные в аргументах командной стороки, или вызвать непосредственно строкой $help$ (или просто h).

Состав программы для ОДУ и для систем ОДУ аналогичен. Поэтому разберем конструкцию программы только для ОДУ первого порядка, разрешенного относительно производной.

Описание алгоритма программы для ОДУ

Программа реализована на основе формул (23), (29). Так как в этих формулах следующее значение y выражается через предыдущее, то нет необходимости задействовать сложные структуры памяти, ограничимся просто переменной.

Из-за состава входных данных может получится большой вывод требуемых выходных данных. Чтобы избежать этого, в программе реализована часть кода, которая в зависимости от размера отрезка и шага выводит лишь ту часть выходных данных, которая помещается в таблицу длиной 15. Например, для шага $h = 0.25$ и отрезка длиной $l = 5$ будет печататься каждое второе y_{i+1}.

Так же в программе на стандартный поток вывода выводится погрешность решения.

Программа в зависимости от аргументов командной строки строит таблицу с необходимыми значениями. Поэтому там строго разделены метод Рунге-Кутта второго и четвертого порядка. Также пользователь может посмотреть работу этих двух методов одновременно (для этого используется $Jump$).

Хоть сложность программы состоит в реализации двух формул, такой большой объем кода обосновывается в комбинации режимов вывода данных и, вследствие этого, повторения одних и тех же частей кода.

Код программы

runge_kutta_method.c

```
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <string.h>

enum
{
    order_is_two = 2,
    order_if_four = 4,
};

double
ff1(double x, double y1, double y2)
{
    return y2 + 2*exp(x);
}
double
ff2(double x, double y1, double y2)
{
    return y1 + x*x;
}
double
xx1(double x)
{
    return exp(x)*(x + 1) - 2 - x*x + 0.5*exp(x) + 0.5*exp(-x);
}
double
yy1(double x)
{
    return exp(x)*x -2*x + 0.5*exp(x) - 0.5*exp(-x);
}

double
f(double x, double y)
{
    return (y - y*y)*x;
    //return (0.5*y + x);
    //return (1.0 + y*y*sin(2*x))/(2.0*y*cos(x)*cos(x));
}

double
f1(double x, double y1, double y2)
{
    return sqrt(x*x + 1.1*y1) + y2;
}

double
f2(double x, double y1, double y2)
{
    return cos(2.1*y2) + y1;
}

double
u(double x)
{
    return 1.0/(1 - 2.0*(exp(-0.5*x*x))/3.0);
```

```c
    //return -2.0*(x + 2) + 4 * exp(x/2.0);
    //return sqrt(x+1.0)/cos(x);
}

void
print_info(void)
{
    printf("_________________________________________________________\n");
    printf("\n\t\tRunge-Kutta method\n");
    printf("_________________________________________________________\n");
}

void
print_help_info(void)
{
    printf("---------------------------------------------------------\n");
    printf("\t\thelp information\n");
    printf("---------------------------------------------------------\n");
    printf("first argument in command line is mode of solving - \n");
    printf("is it one differential equation or system of them. Enter 1 or 2.\n");
    printf("in command line enter accuracy order of method, step and alpha\n");
    printf("accuracy order may be '2' or '4'\n");
    printf("if you want to see 2 methods simultaneously, enter '3'\n");
    printf("step is 'h' in original method by Runge and Kutt\n");
    printf("alpha is parametr of approximation\n");
    printf("it's usually 1 or 0.5. But you can enter yours value\n");
    printf("After that enter size of section ([x0; x0 + l]) \n\n");
}

int
main(int argc, char **argv)
{
    print_info();
    if (argc < 5 || !strcmp(argv[1], "h") || !strcmp(argv[1], "help")) {
        print_help_info();
        return -1;
    }
    int mode = 0;
    if (!sscanf(argv[1], "%d", &mode) || (mode != 1 && mode !=2)) {
        print_help_info();
        return -1;
    }
    int how;
    if (!sscanf(argv[2], "%d", &how) || (how != order_is_two && how != order_if_four && how != 3)) {
        print_help_info();
        return -1;
    }
    double step;
    if (!sscanf(argv[3], "%lf", &step) || step <= 0) {
        fprintf(stderr, "incorrect input value of step\n");
        fprintf(stderr, "enter 'help' or 'h' in comand line\n");
        fprintf(stderr, "to see help-infotmation about input data\n\n");
        return -1;
    }
    double alpha;
    if (!sscanf(argv[4], "%lf", &alpha) || alpha == 0) {
        fprintf(stderr, "inorrect input value of alpha\n");
        fprintf(stderr, "enter 'help' or 'h' in command line\n");
        fprintf(stderr, "to see help-infotmation about input data\n\n");
        return -1;
    }
    int flag = 0;
    double size = 0;
    if (mode == 1) {
        printf("_________________________________________________________\n");
        printf("\n\tOne differential equation\n");
        printf("_________________________________________________________\n");
    } else {
        printf("_________________________________________________________\n");
        printf("\n\tSystem of differential equations\n");
        printf("_________________________________________________________\n\n");
    }
```

```c
    printf("enter size of segment [x0; x0 + l], x0 = 0, l = ");
    if (!scanf("%lf", &size)) {
        fprintf(stderr, "incorrect size of segment\n");
        fprintf(stderr, "enter 'help' or 'h' in command line\n");
        fprintf(stderr, "to see help-infotmation about input data\n\n");
        return -1;
    }
    if (mode == 2) {
        goto Jump2;
    }
    double x0 = 0;
    double y0 = 3;
    double x = x0;
    double y = y0 + f(x0, y0) * step;
    double b = size;
    double amt = size/step;
    if (amt <= 15.0) {
        flag = 1;
    }
    amt /= 15.0;
    amt = ceil(amt);
    double count = 0;

    int n = 13; // size of table column
    printf("step = %.10g, alpha = %.10g\n\n", step, alpha);

    if (how == 3) {
        goto Jump;
    }

    if (!flag) {
        printf("too big size of result table\n");
        printf("you can see result only on x0 + step * %.10g dots\n", step*amt);
    }
    printf("------------------------------------------------------------------------------\n");
    printf("%-*s | %-*s | %-*s \n", n, "x", n, "y", n , "u");
    printf("------------------------------------------------------------------------------\n");
    printf("%-*.10g | %-*.10g | %-*.10g \n", n, x0, n, y0, n, y0);
    if (how == order_is_two) {
        while (x < b) {
            double c = step/(2*alpha);
            double m = f(x, y);
            y += ((1 - alpha)*m + alpha*f(x + c, y + c*m))*step;
            x += step;
            ++count;
            if (count ==  amt) {
                printf("%-*.10g | %-*.10g | %-*.10g\n", n, x, n, y, n, u(x));
                count = 0;
            }
        }
    } else if (how == order_if_four) {
        while (x < b) {
            double k1 = f(x, y);
            double k2 = f(x + step/2.0, y + (step/2.0)*k1);
            double k3 = f(x + step/2.0, y + (step/2.0)*k2);
            double k4 = f(x + step, y + step*k3);
            y += ((k1 + 2*k2 + 2*k3 + k4)/6.0)*step;
            x += step;
            ++count;
            if (count ==  amt) {
                printf("%-*.10g | %-*.10g | %-*.10g \n", n, x, n, y, n, u(x));
                count = 0;
            }
        }
    }
    return 0;

Jump:
    if (!flag) {
        printf("too big size of result table\n");
        printf("you can see result only on x0 + step * %.10g dots\n", step*amt);
    }
```

```
    printf("---------------------------------------------------------------------------------------------\n");
    printf("%-*s | %-*s | %-*s | %-*s | %-*s | %-*s |\n", n, "x", n, "y1", n, "y2", n, "u", n, "z1", n, "z2");
    printf("---------------------------------------------------------------------------------------------\n");
    printf("%-*s | %-*s | %-*s |\n", n, " ", n, "2 order", n, "4 order");
    printf("---------------------------------------------------------------------------------------------\n");
    printf("%-*.10g | %-*.10g | %-*.10g | %-*.10g \n", n, x0, n, y0, n, y0, n, y0);
    double y2 = y;

    while (x < b) {
        double c = step/(2*alpha);
        double m = f(x, y);
        y += ((1 - alpha)*m + alpha*f(x + c, y + c*m))*step;
        double k1 = f(x, y2);
        double k2 = f(x + step/2.0, y2 + (step/2.0)*k1);
        double k3 = f(x + step/2.0, y2 + (step/2.0)*k2);
        double k4 = f(x + step, y2 + step*k3);
        y2 += ((k1 + 2*k2 + 2*k3 + k4)/6.0)*step;
        x += step;
        ++count;
        if (count ==  amt) {
            printf("%-*.10g | %-*.10g | %-*.10g | %-*.10g | %-*lf | %-*lf | \n", n, x, n, y, n, y2, n, u(x), n, u(x) - y, n, u(x) - y2);
            count = 0;
        }
    }
    return 0;

Jump2:
    x0 = 0;
    double y10 = 0;
    double y20 = 0;
    x = x0;
    double y1 = y10;
    y2 =  y20;

    b = size;
    amt = size/step;
    if (amt <= 15.0) {
        flag = 1;
    }
    amt /= 15.0;
    amt = ceil(amt);
    count = 0;

    n = 13; // size of table column
    printf("step = %.10g, alpha = %.10g\n\n", step, alpha);

    if (how == 3) {
        goto Jump3;
    }

    if (!flag) {
        printf("too big size of result table\n");
        printf("you can see result only on x0 + step * %.10g dots\n", step*amt);
    }
    printf("-----------------------------------------------------------------------------\n");
    printf("%-*s | %-*s | %-*s | %-*s | %-*s\n", n, "x", n, "y1", n , "y2", n, "u1", n, "u2");
    printf("-----------------------------------------------------------------------------\n");
    printf("%-*.10g | %-*.10g | %-*.10g | %-*.10g | %-*.10g\n", n, x0, n, y10, n, y20, n, xx1(x0), n, yy1(x0));
    if (how == order_is_two) {
        while (x < b) {

            double c = step/(2*alpha);
            double pm1 = ff1(x, y1, y2);
            double pm2 = ff2(x, y1, y2);
            y1 += ((1 - alpha)*pm1 + alpha*ff1(x + c, y1 + c*pm1, y2 + c*pm2))*step;
            y2 += ((1 - alpha)*pm2 + alpha*ff2(x + c, y1 + c*pm1, y2 + c*pm2))*step;
            x += step;
            ++count;
            if (count ==  amt) {
                printf("%-*.10g | %-*.10g | %-*.10g | %-*.10g | %-*.10g\n", n, x, n, y1, n, y2, n, xx1(x), n, yy1(x));
```

```
                count = 0;
            }
        }
    } else {
        while (x < b) {
            double k1 = ff1(x, y1, y2);
            double m1 = ff2(x, y1, y2);
            double k2 = ff1(x + step/2.0, y1 + (step/2.0)*k1, y2 + (step/2.0)*m1);
            double m2 = ff2(x + step/2.0, y1 + (step/2.0)*k1, y2 + (step/2.0)*m1);
            double k3 = ff1(x + step/2.0, y1 + (step/2.0)*k2, y2 + (step/2.0)*m2);
            double m3 = ff2(x + step/2.0, y1 + (step/2.0)*k2, y2 + (step/2.0)*m2);
            double k4 = ff1(x + step, y1 + step*k3, y2 + step*m3);
            double m4 = ff2(x + step, y1 + step*k3, y2 + step*m3);
            y1 += ((k1 + 2*k2 + 2*k3 + k4)/6.0)*step;
            y2 += ((m1 + 2*m2 + 2*m3 + m4)/6.0)*step;
            x += step;
            ++count;
            if (count ==  amt) {
                printf("%-*.10g | %-*.10g | %-*.10g | %-*.10g | %-*.10g\n", n, x, n, y1, n, y2, n, xx1(x), n, yy1(x));
                count = 0;
            }
        }
    }
    return 0;

Jump3:
    if (!flag) {
        printf("too big size of result table\n");
        printf("you can see result only on x0 + step * %.10g dots\n", step*amt);
    }
    printf("-------------------------------------------------------------------------------\n");
    printf("%-*s | %-*s | %-*s | %-*s | %-*s | %-*s | %-*s \n", n, "x", n, "y1", n, "y2", n, "y3", n, "y4", n, "u1", n, "u2");
    printf("-------------------------------------------------------------------------------\n");
    printf("%-*s | %-*s | %-*s |\n", n, " ", 2*n + 3, "2 order", 2*n + 3, "4 order");
    printf("-------------------------------------------------------------------------------\n");
    printf("%-*.10g | %-*.10g | %-*.10g | %-*.10g | %-*.10g | %-*.10g | %-*.10g\n", n, x0, n, y10, n, y20, n, y10, n, y20, n, xx1(x0), n, yy1(x0));
    double y3 = y1;
    double y4 = y2;

    while (x < b) {
        double c = step/(2*alpha);
        double pm1 = ff1(x, y1, y2);
        double pm2 = ff2(x, y1, y2);
        y1 += ((1 - alpha)*pm1 + alpha*ff1(x + c, y1 + c*pm1, y2 + c*pm2))*step;
        y2 += ((1 - alpha)*pm2 + alpha*ff2(x + c, y1 + c*pm1, y2 + c*pm2))*step;
        double k1 = ff1(x, y3, y4);
        double m1 = ff2(x, y3, y4);
        double k2 = ff1(x + step/2.0, y3 + (step/2.0)*k1, y4 + (step/2.0)*m1);
        double m2 = ff2(x + step/2.0, y3 + (step/2.0)*k1, y4 + (step/2.0)*m1);
        double k3 = ff1(x + step/2.0, y3 + (step/2.0)*k2, y4 + (step/2.0)*m2);
        double m3 = ff2(x + step/2.0, y3 + (step/2.0)*k2, y4 + (step/2.0)*m2);
        double k4 = ff1(x + step, y3 + step*k3, y4 + step*m3);
        double m4 = ff2(x + step, y3 + step*k3, y4 + step*m3);
        y3 += ((k1 + 2*k2 + 2*k3 + k4)/6.0)*step;
        y4 += ((m1 + 2*m2 + 2*m3 + m4)/6.0)*step;
        x += step;
        ++count;
        if (count ==  amt) {
            printf("%-*.10g | %-*.10g | %-*.10g | %-*.10g | %-*.10g | %-*.10g | %-*.10g\n", n, x, n, y1, n, y2, n, y3, n, y4, n, xx1(x), n, yy1(x));
            count = 0;
         }
    }
    return 0;
}
```

Тесты, проверяющие правильность работы программы

Тесты для ОДУ первого порядка, разрешенного относительно первой производной

1. Тест из учебника «Вводные лекции по численным методам» Задача Коши: $y' = \frac{1}{2}y + x,\ u(0) = 0$. И $u(x) = -2(x+2) + 4e^{\frac{1}{2}x}$ – точное решение этой задачи Коши.

$z_1,\ z_2$ – погрешность решения. Графики представлены на рисунке 1.

x	y1	y2	u	z1	z2
	2 order	4 order			
0	0	0	0	0	0
0.25	0.03125	0.03259277344	0.03259381227	0.001344	0.000001
0.5	0.1330566406	0.1360993125	0.1361016668	0.003045	0.000002
0.75	0.3147907257	0.3199616568	0.3199656585	0.005175	0.000004
1	0.587067619	0.5948790368	0.5948850828	0.007817	0.000006
1.25	0.9619125371	0.972975266	0.9729838297	0.011071	0.000009
1.5	1.452947796	1.467988422	1.468000066	0.015052	0.000012
1.75	2.075604925	2.095485781	2.095501176	0.019896	0.000015
2	2.847364954	2.873107378	2.873127314	0.025762	0.000020

Также как в этом тесте, далее $y,\ u,\ z_1,\ z_2$ определяют тот же смысл.

2. Тест из сборника задач по дифференциальным уравнениям А.Ф. Филипова (№ 192):

$$y' = \frac{1 + y^2 \sin(2x)}{2y\cos^2 x}$$

$$u = \frac{\sqrt{x+1}}{\cos^2(x)}$$

$$u(0) = 1$$

Графики представлены на рисунке 2.

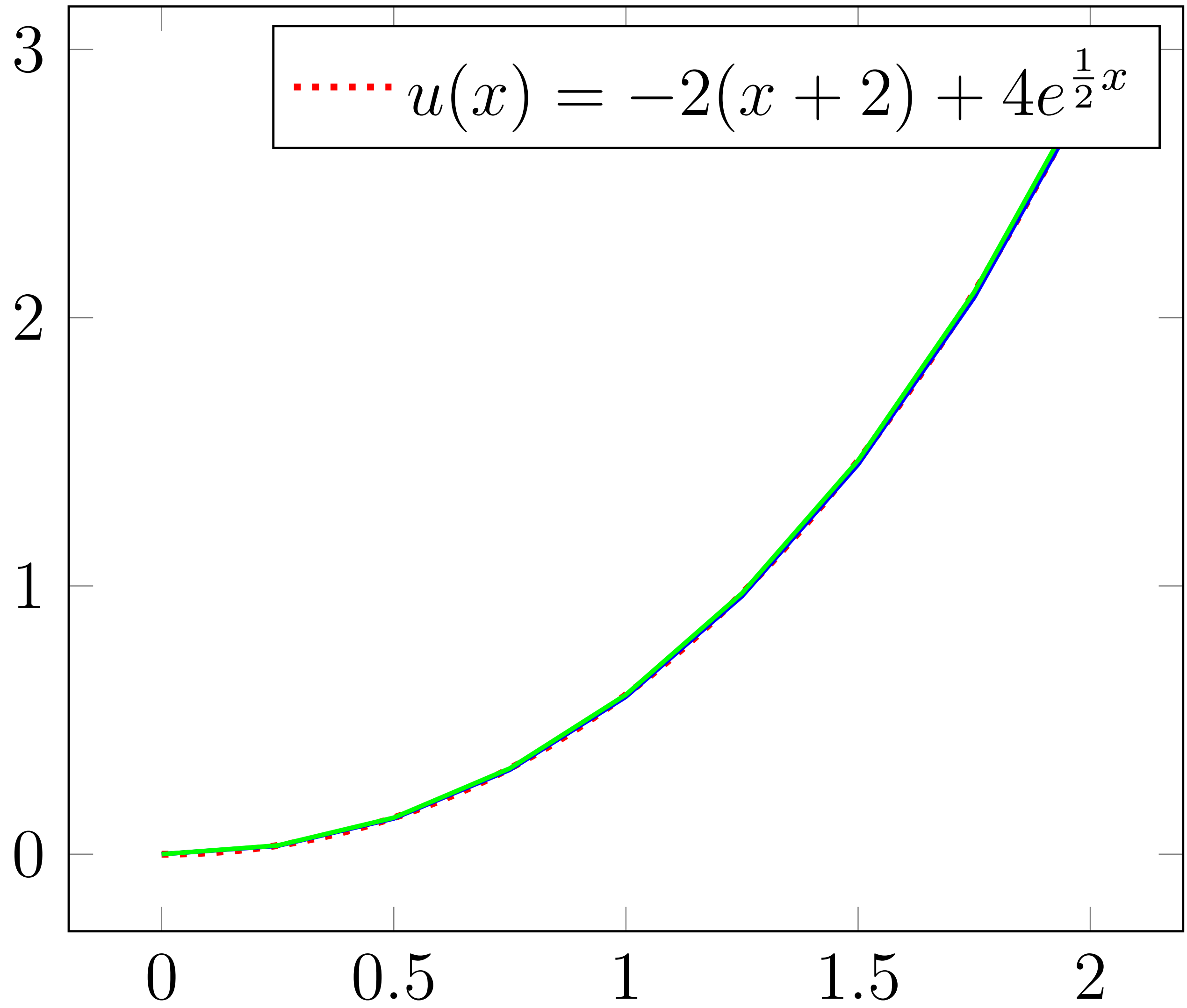

Рис. 1: График искомой функции и график, полученный при помощи метода Рунге-Кутта второго (синий) и четвертого (зеленый) порядка с шагом $h = 0.25$

x	y1	y2	u	z1	z2
	2 order	4 order			
0	1	1	1		
0.1	1.102321852	1.102091476	1.054074832	-0.048247	-0.048017
0.2	1.164961519	1.164483372	1.117725203	-0.047236	-0.046758
0.3	1.240396575	1.239638588	1.193480452	-0.046916	-0.046158
0.4	1.331906746	1.330818691	1.284622804	-0.047284	-0.046196
0.5	1.443977176	1.442484375	1.395589343	-0.048388	-0.046895
0.6	1.582939186	1.580931762	1.532602061	-0.050337	-0.048330
0.7	1.758045063	1.755359449	1.70471831	-0.053327	-0.050641
0.8	1.983375096	1.97976106	1.925689488	-0.057686	-0.054072
0.9	2.281448785	2.276507217	2.2174755	-0.063973	-0.059032
1	2.690634588	2.683690658	2.617448689	-0.073186	-0.066242
1.1	3.282015324	3.27183142	3.194775277	-0.087240	-0.077056

3. Тест из варианта задания. Задача Коши (5)(6).
Графики представлены на рисунке 3.

x	y1	y2	u	z1	z2
	2 order	4 order			
0	3	3	3		
0.2	2.885092612	2.885716496	2.885717914	0.000625	0.000001
0.4	2.599924829	2.600174655	2.60017768	0.000253	0.000003
0.6	2.257699197	2.256554643	2.256556011	-0.001143	0.000001
0.8	1.94087025	1.938359628	1.938357735	-0.002513	-0.000002
1	1.682091393	1.678853183	1.678848879	-0.003243	-0.000004
1.2	1.483754682	1.480393527	1.480388196	-0.003366	-0.000005
1.4	1.336826772	1.333707541	1.333702145	-0.003125	-0.000005
1.6	1.230252008	1.227538356	1.227533378	-0.002719	-0.000005
1.8	1.154252356	1.15198848	1.151984104	-0.002268	-0.000004
2	1.101004659	1.099174827	1.099171087	-0.001834	-0.000004

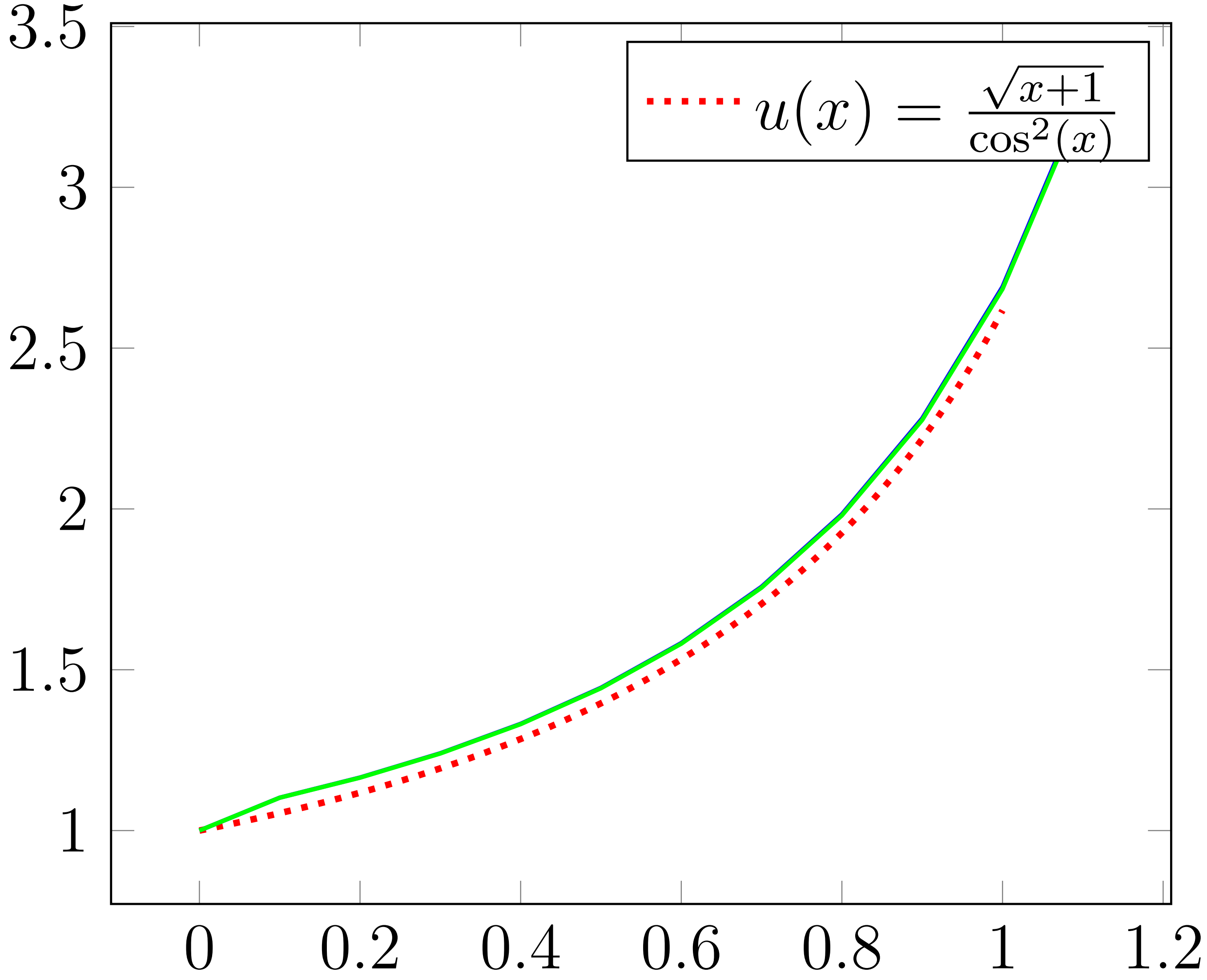

Рис. 2: График искомой функции и график, полученный при помощи метода Рунге-Кутта второго (синий) и четвертого (зеленый) порядка с шагом $h = 0.1$

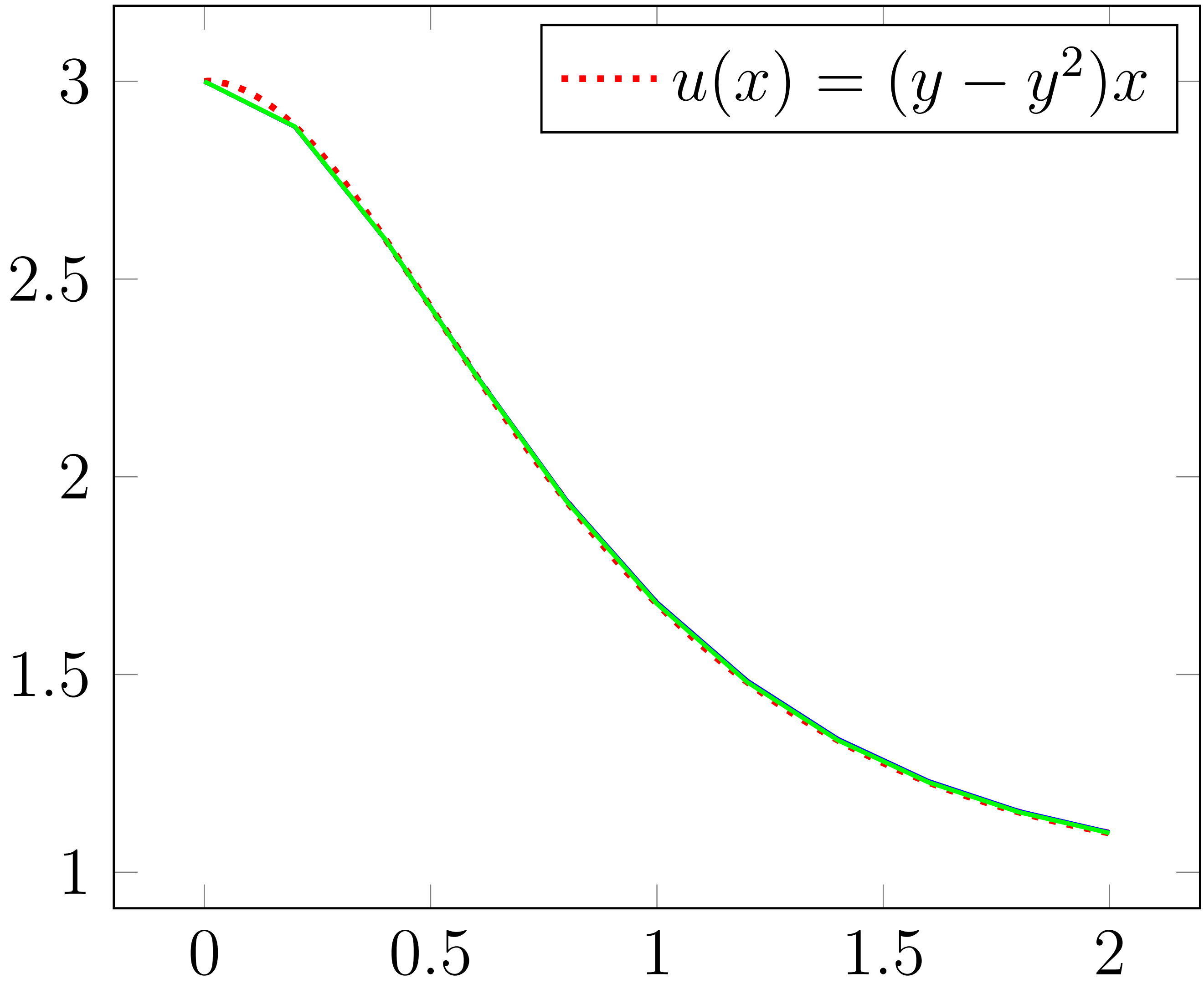

Рис. 3: График искомой функции и график, полученный при помощи метода Рунге-Кутта второго (синий) и четвертого (зеленый) порядка с шагом $h = 0.1$

Тесты для системы ОДУ первого порядка, разрешенных относительно производных неизвестных функций

Выводы

Литература

[1] Костомаров Д.П., Фаворский А.П. *Вводные лекции по численным методам.* Изд-во Логос, 2004.

[2] Столяров А.В. *Сверстай диплом красиво:LaTeX за три дня.* Изд-во Макс Пресс, 20010.

[3] Чернов А.В. *Стиль форматирования программ.*[1]

[1] https://ejudge.ru/study/3sem/style.shtml