МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

---

**Компьютерный практикум по курсу**

**«ВВЕДЕНИЕ В ЧИСЛЕННЫЕ МЕТОДЫ»**

**ЗАДАНИЕ № 2**

**«Численные методы решения дифференциальных уравнений»**

# ОТЧЕТ

**о выполненном задании**

студента 203 учебной группы факультета ВМК МГУ

Дорожкина Дениса Сергеевича

Москва, 2015 г.

# Часть 1

## Вариант задания

Подвариант 1:  1-6, приложение 2-14

## Цель работы

Освоить методы Рунге-Кутта  второго и четвёртого порядка точности, применимые для численного решения задачи Коши для дифференциального уравнения (или системы диыыеренциальных уравнений) первого порядка.

## Постановка задачи

Рассматривается обыкновенное дифференциальное уравнение первого порядка, разрешенное относительно производной и имеющее вид:

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

с дополнительным начальным условием, заданным в точке x = x0:

$$y(x_0) = y_0. \qquad (2)$$

Предполагается, что правая часть уравнения (1) функция f = f(x,y) такова, что гарантирует существование и единственность решения задачи Коши (1)-(2).

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

$$\begin{cases} \dfrac{dy_1}{dx} = f_1(x, y_1, y_2), \\ \dfrac{dy_2}{dx} = f_2(x, y_1, y_2),\ x > x_0. \end{cases} \qquad (3)$$

Дополнительные (начальные) условия задаются в точке x = x0:

$$y_1(x_0) = y_1^{(0)},\ y_2(x_0) = y_2^{(0)}. \qquad (4)$$

Также предполагается, что правые части уравнений из (3) заданы так, что это гарантирует существование и единственность решения задачи Коши (3)-(4), но уже для системы обыкновенных дифференциальных уравнений первого порядка в форме, разрешенной относительно производных неизвестных функций.

**Необходимо:**

1) Решить задачу Коши (1)-(2) (или (3)-(4)) наиболее известными и широко используемыми на практике методами Рунге-Кутта второго и четвертого порядка точности, аппроксимировав дифференциальную задачу соответствующей разностной схемой(на равномерной сетке); полученное конечно-разностное уравнение (или уравнения в случае системы), представляющие фактически некоторую рекуррентную формулу, просчитать численно;
2) Найти численное решение задачи и построить его график;
3) Найденное численное решение сравнить с точным решением дифференциального уравнения.

# Метод решения

Метод Рунге-Кутты является наиболее распространенным методом решения обыкновенных дифференциальных уравнений.
Метод Рунге-Кутты четвертого порядка:

$$y_{i+1} = y_i + h*(k_1 + 2k_2 + 2k_3 + k_4)/6$$
$$k_1 = f(x_i , y_i)$$
$$k_2 = f(x_i + h/2, y_i + k_1/2)$$
$$k_3 = f(x_i + h/2, y_i + k_2/2)$$
$$k_4 = f(x_i + h, y_i + k_3)$$

Метод Рунге-Кутты второго порядка:

$$y_{i+1} = y_i + h*k_2$$
$$k_1 = f(x_i , y_i)$$
$$k_2 = f(x_i + h/2, y_i + k_1/2)$$

Метод явный, итерационный. Метод Рунге-Кутта четвертого порядка требует на каждом шаге четырехкратного вычисления правой части уравнения, а второго порядка всего двукратного.

# Алгоритм решения

- На вход в программу сначала подается число (1 или 2), которое показывает количество уравнений в системе.
- На вход в программу сначала подается число (2 или 4), которое показывает какой из методов Рунге-Кутта будет использован(второго или четвертого порядка).
- Вводятся 2 числа – отрезок, на котором будут производится вычисления.
- Вводится число – количество итераций в методе.
- Считается ответ при помощи итерационного метода Рунге-Кутта выбранного порядка.

# Исходный код

```cpp
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <algorithm>

using namespace std;

double f(double x, double y) // фунцкция из примера
{
    return (x - x * x) * y;
}

double f1(double x, double u, double v) // функция из примера
{
    return pow(M_E, -u * u - v * v) + 2 * x;
}

double f2(double x, double u, double v) // функция из примера
{
    return 2 * u * u + v;
}

double **vectors; // массив генерируемых итерациионно значений x, y1, ...
double h;// шаг
int iter_count;// число итераций

void RK_2(int num) { // метод Рунге-Кутта второго порядка
    double *k1 = (double *)malloc((num + 1) * sizeof(double));
    double *k2 = (double *)malloc((num + 1) * sizeof(double));
    if (num == 1) { // 1 уравнение
        for (int i = 1; i <= iter_count; i++) {
            vectors[0][i] = vectors[0][i - 1] + h;
            for (int k = 1; k < num + 1; k++) {
                k1[k] = f(vectors[0][i - 1], vectors[k][i - 1]);
                k2[k] = f(vectors[0][i - 1] + h / 2, vectors[k][i - 1] + h /
2 * k1[k]);
            }
            for (int k = 1; k < num + 1; k++) {
                double delta_y = h * k2[k];
                vectors[k][i] = vectors[k][i - 1] + delta_y;
            }
        }
    } else if (num == 2) { // 2 уравнения
        double *l1 = (double *)malloc((num + 1) * sizeof(double));
        double *l2 = (double *)malloc((num + 1) * sizeof(double));
        for (int i = 1; i <= iter_count; i++) {
            vectors[0][i] = vectors[0][i - 1] + h;
            for (int k = 1; k < num; k++) {
                k1[k] = f1(vectors[0][i - 1], vectors[k][i - 1], vectors[k +
1][i - 1]);
                l1[k] = f2(vectors[0][i - 1], vectors[k][i - 1], vectors[k +
1][i - 1]);
                k2[k] = f1(vectors[0][i - 1] + h / 2, vectors[k][i - 1] + h /
2 * k1[k], vectors[k + 1][i - 1] + h / 2 * l1[k]);
                l2[k] = f2(vectors[0][i - 1] + h / 2, vectors[k][i - 1] + h /
2 * k1[k], vectors[k + 1][i - 1] + h / 2 * l1[k]);
            }
            for (int k = 1; k < num; k++) {
                double delta_y = h / 2 * (k1[k] + k2[k]);
                vectors[k][i] = vectors[k][i - 1] + delta_y;
                delta_y = h / 2 * (l1[k] + l2[k]);
```

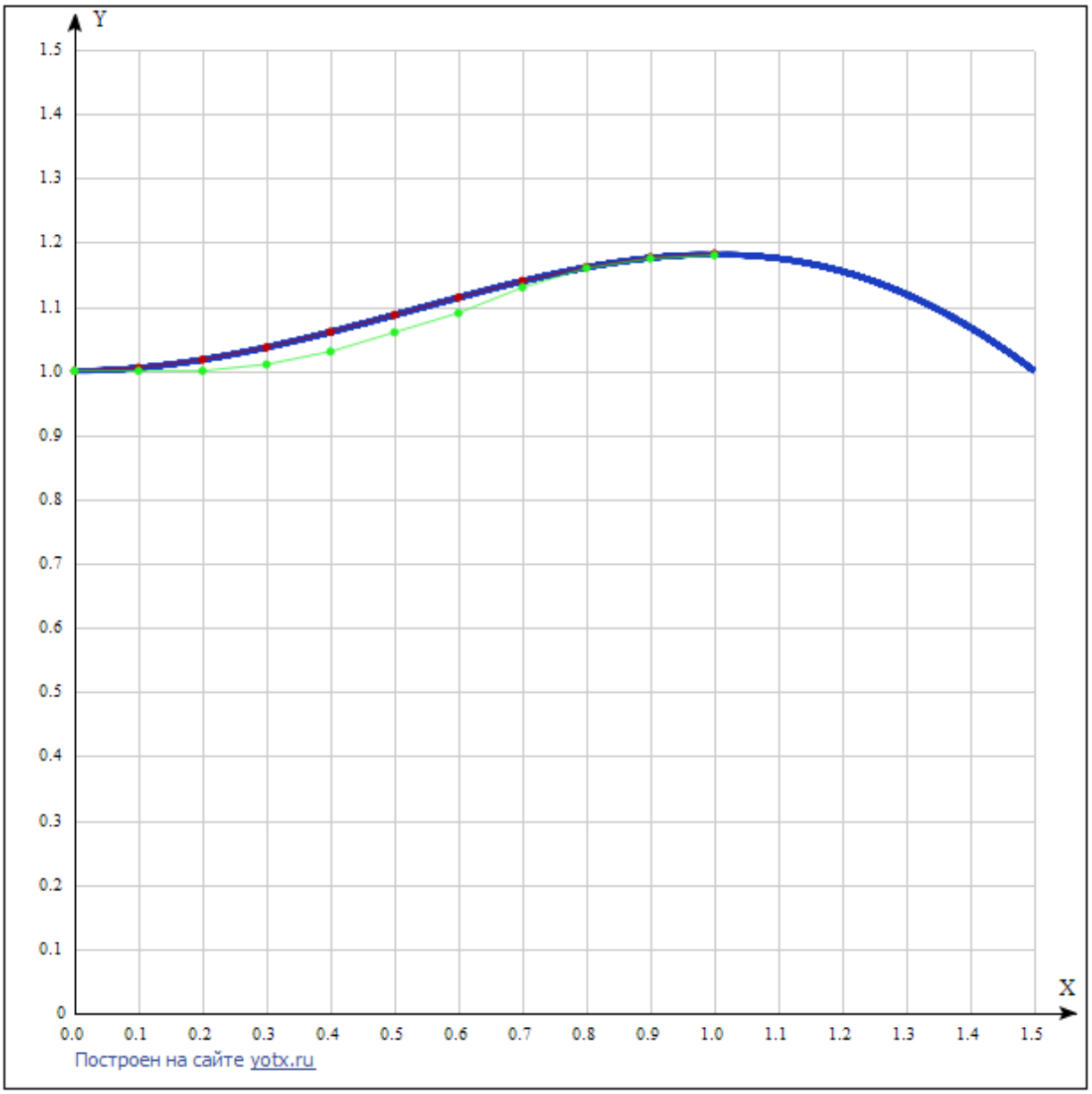
```
                vectors[k + 1][i] = vectors[k + 1][i - 1] + delta_y;
            }
        free(l1);
        free(l2);
        }
    }
    free(k1);
    free(k2);
}

void RK_4(int num) { // Метод Рунге-Кутта четвертого порядка
    double *k1 = (double *)malloc((num + 1) * sizeof(double));
    double *k2 = (double *)malloc((num + 1) * sizeof(double));
    double *k3 = (double *)malloc((num + 1) * sizeof(double));
    double *k4 = (double *)malloc((num + 1) * sizeof(double));
    if (num == 1) { // 1 уравнение
        for (int i = 1; i <= iter_count; i++) {
            vectors[0][i] = vectors[0][i - 1] + h;
            for (int k = 1; k < num + 1; k++) {
                k1[k] = f(vectors[0][i - 1], vectors[k][i - 1]);
                k2[k] = f(vectors[0][i - 1] + h / 2, vectors[k][i - 1] + h /
2 * k1[k]);
                k3[k] = f(vectors[0][i - 1] + h / 2, vectors[k][i - 1] + h /
2 * k2[k]);
                k4[k] = f(vectors[0][i - 1] + h, vectors[k][i - 1] + h *
k3[k]);
            }
            for (int k = 1; k < num + 1; k++) {
                double delta_y = h / 6 * (k1[k] + 2 * k2[k] + 2 * k3[k] +
k4[k]);
                vectors[k][i] = vectors[k][i - 1] + delta_y;
            }
        }

    } else if (num == 2) { // 2 уравнения
        double *l1 = (double *)malloc((num + 1) * sizeof(double));
        double *l2 = (double *)malloc((num + 1) * sizeof(double));
        double *l3 = (double *)malloc((num + 1) * sizeof(double));
        double *l4 = (double *)malloc((num + 1) * sizeof(double));
        for (int i = 1; i <= iter_count; i++) {
            vectors[0][i] = vectors[0][i - 1] + h;
            for (int k = 1; k < num; k++) {
                k1[k] = f1(vectors[0][i - 1], vectors[k][i - 1], vectors[k +
1][i - 1]);
                l1[k] = f2(vectors[0][i - 1], vectors[k][i - 1], vectors[k +
1][i - 1]);
                k2[k] = f1(vectors[0][i - 1] + h / 2, vectors[k][i - 1] + h /
2 * k1[k], vectors[k + 1][i - 1] + h / 2 * l1[k]);
                l2[k] = f2(vectors[0][i - 1] + h / 2, vectors[k][i - 1] + h /
2 * k1[k], vectors[k + 1][i - 1] + h / 2 * l1[k]);
                k3[k] = f1(vectors[0][i - 1] + h / 2, vectors[k][i - 1] + h /
2 * k2[k], vectors[k + 1][i - 1] + h / 2 * l2[k]);
                l3[k] = f2(vectors[0][i - 1] + h / 2, vectors[k][i - 1] + h /
2 * k2[k], vectors[k + 1][i - 1] + h / 2 * l2[k]);
                k4[k] = f1(vectors[0][i - 1] + h, vectors[k][i - 1] + h *
k3[k], vectors[k + 1][i - 1] + h * l3[k]);
                l4[k] = f2(vectors[0][i - 1] + h, vectors[k][i - 1] + h *
k3[k], vectors[k + 1][i - 1] + h * l3[k]);
            }
            for (int k = 1; k < num; k++) {
                double delta_y = h / 6  * (k1[k] + 2 * k2[k] + 2 * k3[k] +
k4[k]);
                vectors[k][i] = vectors[k][i - 1] + delta_y;
                delta_y = h / 6 * (l1[k] + 2 * l2[k]+ 2 * l3[k] + l4[k]);
```

```
                vectors[k + 1][i] = vectors[k + 1][i - 1] + delta_y;
            }
        }
        free(l1);
        free(l2);
        free(l3);
        free(l4);
    }
    free(k1);
    free(k2);
    free(k3);
    free(k4);
}

void print(int n) // вывод ответа
{
    cout << "i: ";
    for (int i = 0; i < iter_count + 1; i++) {
        cout << i << " ";
    }
    cout << "\n";
    cout << "x: " ;
    for (int i = 0; i < iter_count + 1; i++) {
        cout << vectors[0][i] <<     " ";
    }
    cout << "\n";
    for (int k = 1; k < n + 1; k++ ){
        cout << "y[" << k << "]:" ;
        for (int i = 0; i < iter_count + 1; i++) {
            cout << vectors[k][i] << " ";
        }
        cout << "\n";
    }
}


int main()
{
    int mod; // количество уравнений
    int method; // порядок в методе Рунге-Кутта
    double left, right; // границы вычисления
    cout << "Пожалуйста выберите тип входных данных:(1) - одно уравнение, (2)
- система из двух уравнений\n";
    cin >> mod;
    cout << "Пожалуйста выберите порядок метода Рунге-Кутта:(2) - второй, (4)
- четвертый\n";
    cin >> method;
    cout << "Пожалуйста введите отрезок, на котором будет искаться решение в
формате \"a b\"\n";
    cin >> left >> right;
    cout << "Пожалуйста введите количество итераций\n";
    cin >> iter_count;
    h = (right - left) / iter_count;
    vectors = (double **)malloc((mod + 1) * sizeof(double *));
    for (int i = 0; i <= mod; i++) {
        vectors[i] = (double *)malloc((iter_count + 1) * sizeof(double));
    }
    cout << "Введите начаньные условия\n";
    for (int i = 0; i < mod + 1; i++) {
        cin >> vectors[i][0];
    }
    if (method == 2) {
        RK_2(mod);
        print(mod);
```

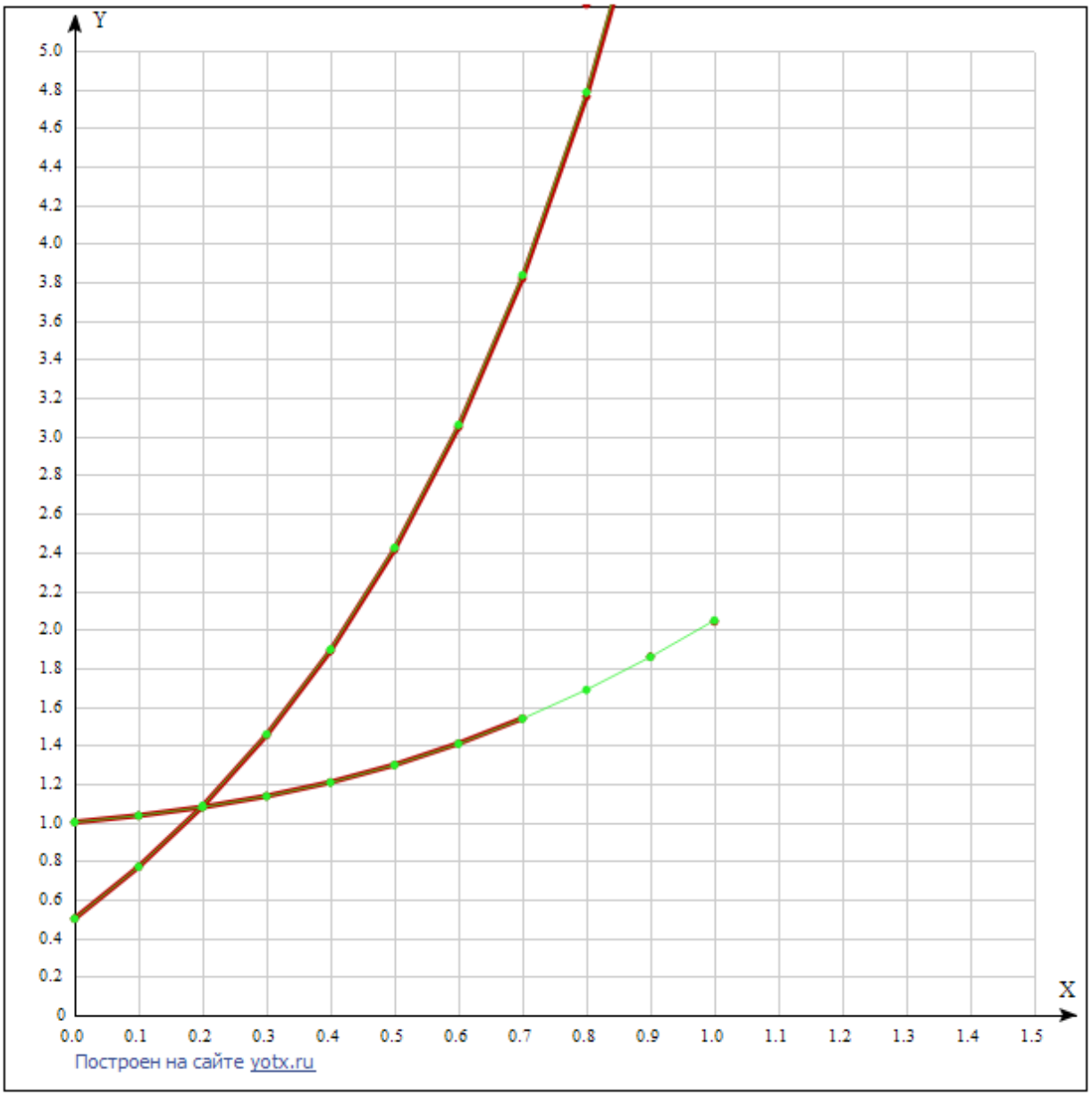
```
    } else if (method == 4) {
        RK_4(mod);
        print(mod);
    } else {
        cout << "Введите правильный метод";
        exit(0);
    }
    return 0;
}
```

# Тестирование

Сверка ответов осуществлялась при помощи сервисов:
http://www.yotx.ru/ и http://www.wolframalpha.com/
Пример 1: (1 уравнение)
Синий – точное решение
Красный – 4 порядок
Зеленый – 2 порядок



Пример 2: (система из 2х уравнений)
Красный – 4 порядок
Зеленый – 2 порядок




# Выводы:

Метод Рунге-Кутта позволяет численно решить систему ОДУ 1-го порядка с заданной наперед точностью. Метод 4-го порядка дает по точности результат лучше, чем метод 2-го порядка, но производит больше вычислений.

# Литература:

Костомаров Д.П., Фаворский А.П.. Вводные лекции по численным методам.
http://www.mathprofi.ru/metody_eilera_i_runge_kutty.html
Костомаров Д.П. Введение в численные методы. Методическое пособие для 2 курса.

# Часть 2

## Вариант задания

Подвариант 2: 10

## Цель работы

Освоить метод прогонки решения краевой задачи для дифференциального уравнения второго порядка.

## Постановка задачи

Рассмотрим линейное дифференциальное уравнение

$$y'' + p(x)\cdot y' + q(x)\cdot y = f(x) \qquad (1)$$

на интервале [a,b] с дополнительными условиями в граничных точках

$$A0 \cdot y(a) + B0 \cdot y'(a) = C0$$

$$A1 \cdot y(b) + B1 \cdot y'(b) = C1 \qquad (2)$$

1) Решить краевую задачу (1)–(2) методом конечных разностей, аппроксимировав ее разностной схемой второго порядка точности (на равномерной сетке); полученную систему конечно-разностных уравнений решить методом прогонки;
2) Найти разностное решение задачи;
3) Найденное разностное решение сравнить с точным решением дифференциального уравнения(подобрать специальные тесты, где аналитические решения находятся в классе элементарных функций, при можно использовать ресурсы on-line системы http://www.wolframalpha.com.

## Метод решения

Дискретизируем задачу, т.е. вводим сетку по переменной x:

$$h := \frac{b-a}{n}, \qquad i := 0 \ldots n$$

$$x_i = a + i \cdot h;$$

$$y_i := y(x_i);$$

1. Заменяем исходное уравнение (1) конечно-разностным во внутренних узлах:

$$\frac{y_{i+1}-2y_i+y_{i-1}}{h^2}+p_i\cdot\frac{y_{i+1}-y_{i-1}}{2h}+q_i\cdot y_i=f_i$$

Это уравнение приводим к каноническому трехдиагональному виду

$$a_i\cdot y_{i-1}-b_i\cdot y_i+c_i\cdot y_{i+1}=d_i\ , \qquad (3)$$

где

$$a_i=\frac{1}{h^2}-\frac{p_i}{2h},\quad b_i=\frac{2}{h^2}-q_i,\quad c_i=\frac{1}{h^2}+\frac{p_i}{2h},\quad d_i=-f_i$$

2. Линейную систему (3) решаем методом прогонки. Ищем в виде

$$y_{j-1}:=y_j\cdot\xi_j+\eta_j$$

$$j=n,\ n-1,\ldots 2 \qquad (4)$$

где $\xi_j$ , $\eta_j$ - прогоночные коэффициенты, которые необходимо предварительно вычислить. Решение реализуется в два этапа.

3.1. Прямой ход – от левого края заданного интервала [a,b] до правого в узлах сетки вычисляются $\xi_j$ , $\eta_j$ , $j = 1, \ldots, n$

3.2. Обратный ход – от правого края до левого по формуле (4) в тех же узлах находится искомое решение.

Прямой ход реализуется рекуррентными формулами для прогоночных коэффициентов, которые получаются из (3) и (4) при k=1..n-1:

$$\begin{bmatrix}\xi_{k+1}\\ \eta_{k+1}\end{bmatrix}:=\begin{bmatrix}\dfrac{c_k}{b_k-a_k\cdot\xi_k}\\ \dfrac{\eta_k\cdot a_k-d_k}{b_k-a_k\cdot\xi_k}\end{bmatrix},$$

Начальные значения коэффициентов для этих рекуррентных формул можно найти из (4) и левого краевого условия (2):

$$\xi_1 = -\frac{B0}{A0*h-B0} \qquad \eta_1 = \frac{C0*h}{A0*h-B0}$$

После того как получены прогоночные коэффициенты, можно приступать к обратному ходу по формуле (4), но предварительно необходимо найти значение $y_n$. Из (4) и правого краевого условия (2) получаем

$$y_n = \frac{B1*\eta_n + C1*h}{B1*(1-\xi_n)+A1*h}$$

а, теперь, собственно обратный ход - искомую функцию находим по рекуррентной формуле

# Алгоритм решения

- Вводится число – количество итераций в методе.
- Считается ответ при помощи метода Прогонки.

# Исходный код

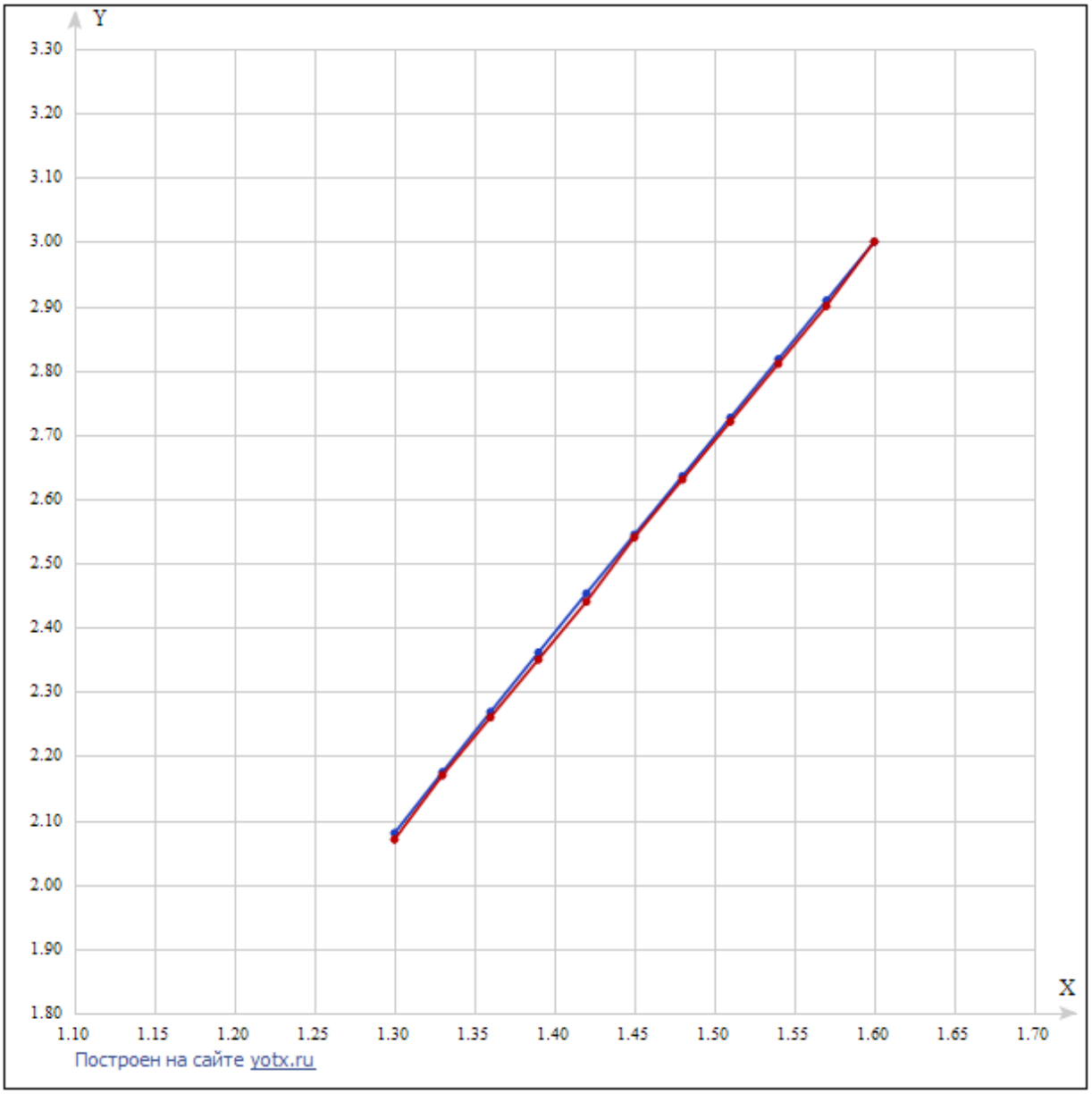
```
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <limits.h>
#include <algorithm>

using namespace std;

int iter_count; // количество итераций
//k1 * y(l) + k2 * y'(l) = a
//l1 * y(r) + l2 * y'(r) = b
double k1 = 2, k2 = -1, a = 1, l1 = 1, l2 = 0, b = 3; // параметры системы
(из примера)
double l = 1.3, r = 1.6; // параметры системы (из примера)
double h;// шаг

double p(double x) // из примера
{
    return 1.5;
}

double q(double x) // из примера
{
```

```
    return -x;
}

double f(double x) // из примера
{
    return 0.5;
}


void progonka(double *arr_x, double *arr_y) // метод прогонки
{
    double *arr_p = (double *)malloc((iter_count + 1) * sizeof(double));
    double *arr_q = (double *)malloc((iter_count + 1) * sizeof(double));
    double *arr_r = (double *)malloc((iter_count + 1) * sizeof(double));
    double *arr_f = (double *)malloc((iter_count + 1) * sizeof(double));
    double *arr_alpha = (double *)malloc((iter_count + 1) * sizeof(double));
    double *arr_betta = (double *)malloc((iter_count + 1) * sizeof(double));
    // прямой проход
    // вычисление коефицентов
    for (int i = 0; i <= iter_count; i++) {
        arr_x[i] = l + h * i;
        arr_p[i] = 1 - p(arr_x[i]) * h / 2;
        arr_q[i] = 1 + p(arr_x[i]) * h / 2;
        arr_r[i] = 2 - q(arr_x[i]) * h * h;
        arr_f[i] = h * h * f(arr_x[i]);
    }
    // вычисление альфа и бетта
    arr_alpha[1] = k2 / (k2 - k1 * h);
    arr_betta[1] = - (a * h) / (k2 - k1 * h);
    for (int i = 1; i < iter_count; i++) {
        arr_alpha[i + 1] = arr_q[i] / (arr_r[i] - arr_alpha[i] * arr_p[i]);
        arr_betta[i + 1] = (arr_p[i] * arr_betta[i] - arr_f[i]) / (arr_r[i] -
arr_alpha[i] * arr_p[i]);
    }
    // обратный проход
    arr_y[iter_count] = (l2 * arr_betta[iter_count] + b * h) / (l2 + h * l1 -
l2 * arr_alpha[iter_count]);
    for (int i = iter_count - 1; i >= 0; i--) {
        arr_y[i] = arr_alpha[i + 1] * arr_y[i + 1] + arr_betta[i + 1];
    }
}


int main()
{
    cout << "Пожалуйста введите количество итераций\n";
    cin >> iter_count;
    h = (r - l) / iter_count;
    double *arr_x = (double *)malloc((iter_count + 1) * sizeof(double));
    double *arr_rez = (double *)malloc((iter_count + 1) * sizeof(double));
    progonka(arr_x, arr_rez);
    cout << "x: " ;
    for (int i = 0; i < iter_count + 1; i++) {
        cout << arr_x[i] << " ";
    }
    cout << "\n";
    cout << "y: " ;
    for (int i = 0; i < iter_count + 1; i++) {
        cout << arr_rez[i] << " ";
    }
    cout << "\n";
    return 0;
}
```

# Тестирование

Сверка ответов осуществлялась при помощи сервисов:
http://www.yotx.ru/  и http://www.wolframalpha.com/

Синий – точное решение
Красный – полученное решение




# Выводы:

Метод прогонки решения краевой задачи позволяет численно решить дифференциальное уравнение второго порядка на отрезке [a; b] с наперед заданной точностью.

# Литература:

Костомаров Д.П., Фаворский А.П.. Вводные лекции по численным методам.
Костомаров Д.П. Введение в численные методы. Методическое пособие для 2 курса.