**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**

студентки 203 учебной группы факультета ВМК МГУ

Шавалиевой Ралины

Москва, 2015 г.

**ПРАКТИЧЕСКАЯ РАБОТА №2(2)**

**Вариант.**

Краевая задача – 7.

**Цель работы.**

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

**Постановка задачи.**

Рассмотрим линейное дифференциальное уравнение



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

*A0 ∙ y(a) + B0 ∙ y'(a) = C0*

*A1 ∙ y(b) + B1 ∙ y'(b) = C1*  (2)

**Цели и задачи практической работы.**

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-ru.com>.

**Алгоритм решения.**

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

, *i:=0... n*

;

;

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



Это уравнение приводим к каноническому трехдиагональному виду



где



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



j = n, n - 1,...2 (4)

где ξj , ηj - 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 ξj , ηj , j = 1, . . .,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:

,

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

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

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



**Описание программы.**

В самом начале программы определяются константы: k - число точек в сетке на отрезке, a – координата левого конца отрезка, b – координата правого конца отрезка. Также необходимо написать функции f(x), p(x), q(x) и real\_solve(x), где последняя функция – решение данного уравнения.

Затем программа просит пользователя ввести коэффициенты *A0, B0, C0, A1, B1, C1*.

*A0 ∙ y(a) + B0 ∙ y'(a) = C0*

*A1 ∙ y(b) + B1 ∙ y'(b) = C1*

Программа написана в соответствии с приведенным выше алгоритмом и содержит комментарии в достаточном для ее понимая количестве.

Правильность работы программы подтверждается системой тестов, которые были проверены с помощью специализированного программного обеспечения (http://www.wolframalpha.com).

**Выводы.**

Мной был изучен метод прогонки решения краевой задачи для дифференциального уравнения второго порядка.

**Приложение 1**

**Текст программы**

#include <stdlib.h>

#include <stdio.h>

#include <math.h>

// изменяемые параметры

#define k 10 // кол-во шагов

#define a 0.4 // левая граница рассматриваемого отрезка

#define b 0.7 // правая граница рассматриваемого отрезка

double f(double);

double real\_solve(double);

double p(double);

double q(double);

double \*sweep\_method(double A0, double B0, double C0, double A1, double B1, double C1);

void printf\_solve(double \*solve);

// заполняются для каждого теста

double f(double x)

{

return -1; // пример

}

double real\_solve(double x)

{

return x; // пример

}

double p(double x)

{

return -3; // пример

}

double q(double x)

{

return -1 / x; // пример

}

int

main(void)

{

double A0, B0, C0, A1, B1, C1;

printf("Введите начальные условия (A0, B0, C0, A1, B1, C1)\n");

scanf("%lf%lf%lf%lf%lf%lf", &A0, &B0, &C0, &A1, &B1, &C1);

double \*solve;

solve = sweep\_method(A0, B0, C0, A1, B1, C1);

//вывод полученных и действильных значений

printf\_solve(solve);

free(solve);

return 0;

}

// прямой и обратный ход метода прогонки

double \*sweep\_method(double A0, double B0, double C0, double A1, double B1, double C1)

{

double \*y = calloc(k + 1, sizeof(\*y));

// прогоночные коэффициенты

double \*u = calloc(k, sizeof(\*u));

double \*v = calloc(k, sizeof(\*v));

double h = (double)(b - a) / k; // шаг

// начальные значения прогоночных коэффициенов

u[0] = - B0 / (A0 \* h - B0);

v[0] = C0 \* h / (A0 \* h - B0);

// прямой ход

double ai, bi, ci, di, xi = a + h;

for (int i = 0; i < k - 1; i++) {

// константы канонического трехдиагонального вида

ai = 1 / (h \* h) - p(xi) / (2 \* h);

bi = 2 / (h \* h) - q(xi);

ci = 1 / (h \* h) + p(xi) / (2 \* h);

di = f(xi);

// прогоночные коэффициенты

u[i + 1] = ci / (bi - ai \* u[i]);

v[i + 1] = (v[i] \* ai - di) / (bi - ai \* u[i]);

xi += h;

}

// обратный ход

y[k] = (B1 \* v[k - 1] + C1 \* h) / (B1 \* (1 - u[k - 1]) + A1 \* h);

for (int i = k; i > 0; i--) {

y[i - 1] = y[i] \* u[i - 1] + v[i - 1];

}

free(u);

free(v);

return y;

}

// вывод ответа

void printf\_solve(double \*solve)

{

double xi = a;

double h = (double)(b - a) / k;

for (int i = 0; i < k + 1; i++) {

printf("x = %lf %\*.\*lf %\*.\*lf\n", xi, 10, 6, solve[i], 10, 6, real\_solve(xi));

xi += h;

}

}

**Приложение 2**

**Система тестов, подтверждающая решения СЛАУ**

1. **Тесты проверенные с помощью http://www.wolframalpha.com.**

Для всех тестов я определяла k = 10.

1. Уравнение:

y’’(x) + y’(x) = 1 (1)

с дополнительными условиями:

y’(0) = 0;

y(1) = 1; (2)

double f(double x)

{

return x + 1;

}

double real\_solve(double x)

{

return x;

}

double p(double x)

{

return 1;

}

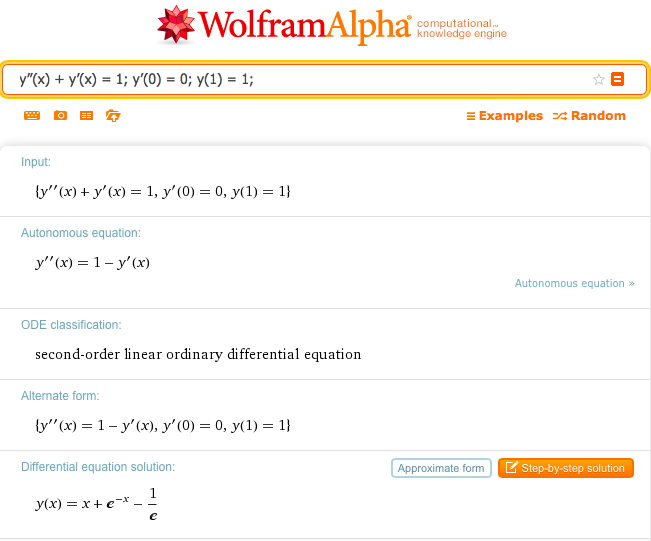
double q(double x)

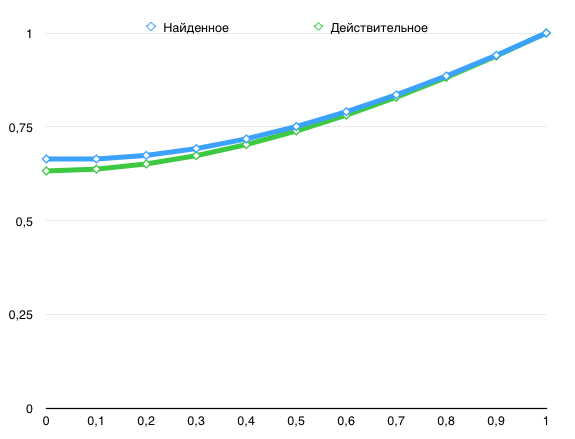
{

return 0;

}

Ответ, полученный с помощью <http://www.wolframalpha.com> :



****Входные аргументы: 0 1 0 1 0 1

Ответ программы:

**x = 0.000000 0.664049 0.632121**

**x = 0.100000 0.664049 0.636958**

**x = 0.200000 0.673573 0.650851**

**x = 0.300000 0.691713 0.672939**

**x = 0.400000 0.717650 0.702441**

**x = 0.500000 0.750640 0.738651**

**x = 0.600000 0.790013 0.780932**

**x = 0.700000 0.835159 0.828706**

**x = 0.800000 0.885529 0.881450**

**x = 0.900000 0.940626 0.938690**

**x = 1.000000 1.000000 1.000000**

1. Уравнение:

y’’(x) + y’(x) + y(x) = x + 1 (1)

с дополнительными условиями:

y(0) = 0;

y(1) = 1; (2)

double f(double x)

{

return x + 1;

}

double real\_solve(double x)

{

return x;

}

double p(double x)

{

return 1;

}

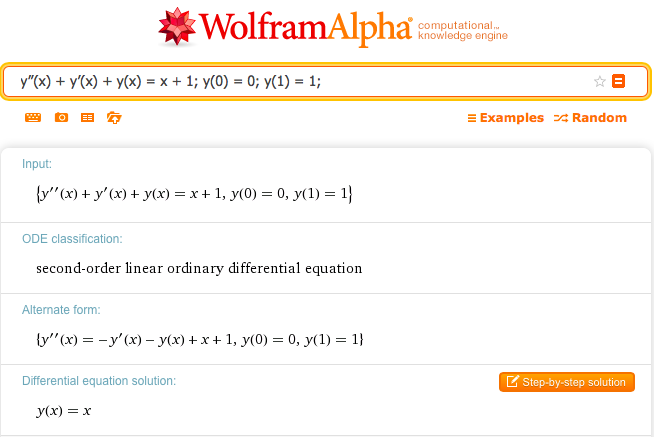
double q(double x)

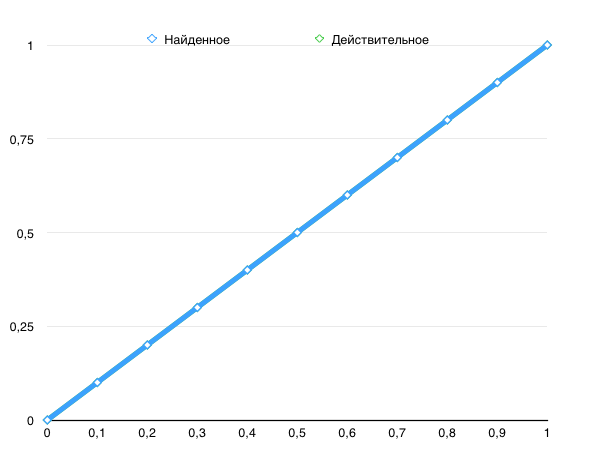
{

return 1;

}

Ответ, полученный с помощью <http://www.wolframalpha.com> :



****Входные аргументы: 1 0 0 1 0 1

Ответ программы:

**x = 0.000000 0.000000 0.000000**

**x = 0.100000 0.100000 0.100000**

**x = 0.200000 0.200000 0.200000**

**x = 0.300000 0.300000 0.300000**

**x = 0.400000 0.400000 0.400000**

**x = 0.500000 0.500000 0.500000**

**x = 0.600000 0.600000 0.600000**

**x = 0.700000 0.700000 0.700000**

**x = 0.800000 0.800000 0.800000**

**x = 0.900000 0.900000 0.900000**

**x = 1.000000 1.000000 1.000000**

1. Уравнение:

y’’(x) + y(x) = 1 (1)

с дополнительными условиями:

y(0) = 0

y’(pi / 2) = 0 (2)

double f(double x)

{

return 1;

}

double real\_solve(double x)

{

return (-exp(M\_PI/2-x)\*M\_PI+exp(M\_PI/2)\*(M\_PI-2\*x)+2\*x)/(2-2\*exp(M\_PI/2));

}

double p(double x)

{

return 1;

}

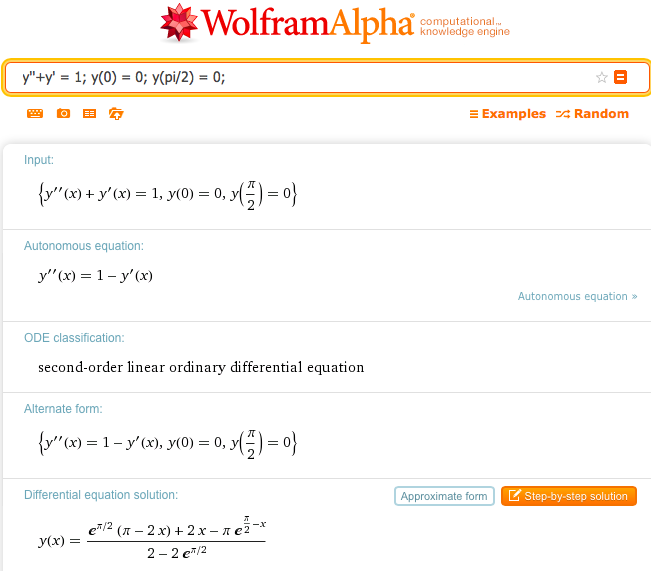
double q(double x)

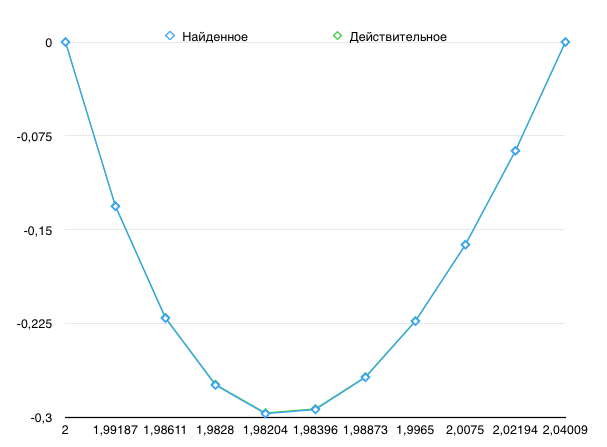
{

return 0;

}

Ответ, полученный с помощью <http://www.wolframalpha.com> :



****Входные аргументы: 1 0 0 1 0 0

Ответ программы:

**x = 0.000000 0.000000 -0.000000**

**x = 0.157080 -0.131485 -0.131181**

**x = 0.314159 -0.220944 -0.220460**

**x = 0.471239 -0.274496 -0.273926**

**x = 0.628319 -0.297372 -0.296787**

**x = 0.785398 -0.294039 -0.293491**

**x = 0.942478 -0.268314 -0.267841**

**x = 1.099557 -0.223458 -0.223085**

**x = 1.256637 -0.162258 -0.162002**

**x = 1.413717 -0.087094 -0.086964**

**x = 1.570796 0.000000 -0.000000**

1. **Тест, заданный уравнением (7) из приложения к заданию.**

Уравнение :

y''(x) - 3y'(x) – (1/ x) \* y(x) = 1 (1)

с дополнительными условиями:

y(0.4) = 2

y(0.7) - 2y'(0.7) = 0.7 (2)

WolframAlpha выдает ответ:

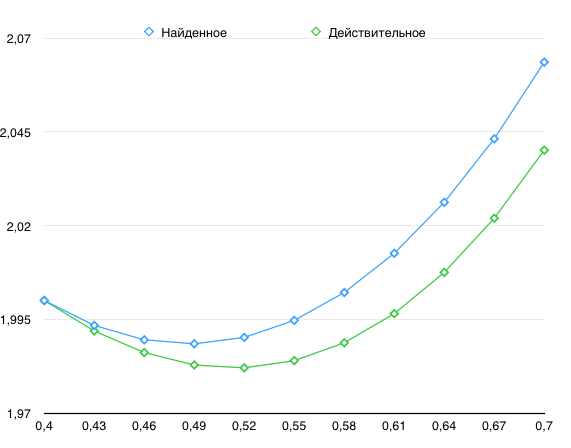
y(x) = (2.18628 + 1.0472 i) + 0.0551615 e^(3x) - 0.333333x + 0.333333e^(3x) Ei(-3x) –

-0.333333log(-3x)

Для проверки я заменяла х на конкретное число (шла по сетке) и проверяла значение данного выражения для этого х. Получились такие результаты:

**x = 0.400000 2.00000**

**x = 0.430000 1.99187**

**x = 0.460000 1.98611**

**x = 0.490000 1.98280**

**x = 0.520000 1.98204**

**x = 0.550000 1.98396**

**x = 0.580000 1.98873**

**x = 0.610000 1.99650**

**x = 0.640000 2.00750**

**x = 0.670000 2.02194**

**x = 0.700000 2.04009**

Входные аргументы: 1 0 2 1 -2 0.7

Ответ программы:

**x = 0.400000 2.000000**

**x = 0.430000 1.993358**

**x = 0.460000 1.989517**

**x = 0.490000 1.988447**

**x = 0.520000 1.990158**

**x = 0.550000 1.994695**

**x = 0.580000 2.002135**

**x = 0.610000 2.012587**

**x = 0.640000 2.026191**

**x = 0.670000 2.043118**

**x = 0.700000 2.063571**

