**Московский авиационный институт**

**(Национальный исследовательский университет)**

Факультет прикладной математики и информационных технологий

Кафедра вычислительной математики и программирования

**Отчет по лабораторным работам**

по курсу «Численные Методы»

Студент: Иларионов Д.А.

Группа: М8О-308Б

Преподаватель: Сластушенский Ю.В.

**Лабораторная работа 1.1**

1. Тема ЛР:

Реализовать алгоритм LU - разложения матриц (с выбором главного элемента) в виде программы. Используя разработанное программное обеспечение, решить систему линейных алгебраических уравнений (СЛАУ). Для матрицы СЛАУ вычислить определитель и обратную матрицу.

1. Вариант : **7**



1. Алгоритм:

Для данной ЛР используется алгоритм LU разложения матриц. Задается исходная матрица и вектор решений. После чего из этой матрицы формируются 2 матрицы – матрица L и матрица U. Главная диагональ матрицы L, значения всех элементов на ней = 1. Остальные элементы считаются в зависимости от элементов на диагонали исходной матрицы, а элементы над главной диагональю = 0. В итоге матрица L – нижняя треугольная. Матрица U образуется из исходной вычитанием определенных элементов, она в свою очередь получается верхней треугольной. Эти матрицы позволяют быстро найти определитель, решить саму систему, и так далее. Например, определитель – всего лишь произведение элементов на главной диагонали U матрицы.

1. Среда разработки:

Visual Studio 2019 , язык – C++

1. Реализация

Использую библиотеку дробных чисел, которую я делал для какой-то ЛР по дискретному анализу в прошлом семестре. Вводится матрица и вектор B. После чего строится матрица L, а матрица U образуется из исходной. Идет вычисление ответа, определителя, обратных матриц. После чего, выводится все на экран.

1. Код (С++)

#include<iostream>

#include<vector>

#include "TDrob.h"

#include "WorkMatrix.h"

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

cout << "Введите размер матрицы\n";

int N;

cin >> N;

vector<vector<TDrob> >matrix(N, vector<TDrob>(N, 0));

vector<vector<TDrob> >matrixL(N, vector<TDrob>(N, 0));

vector<TDrob>solve(N, 0);

vector<TDrob>answerY(N, 0);

vector<TDrob>answer(N, 0);

vector<vector<TDrob> >matrixUObr(N, vector<TDrob>(N, 0));

vector<vector<TDrob> >matrixLObr(N, vector<TDrob>(N, 0));

vector<vector<TDrob> >matrixObr(N, vector<TDrob>(N, 0));

TDrob det = 1;

cout << "Введите матрицу A\n";

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

cin >> matrix[i][j];

}

}

cout << "Введите вектор B\n";

for (int i = 0; i < N; ++i) {

cin >> solve[i];

}

for (int i = 0; i < N; ++i) {

matrixL[i][i] = 1;

}

for (int r = 0; r < N - 1; ++r) {

for (int i = r+1; i < N; ++i) {

TDrob koef = matrix[i][r] / matrix[r][r];

matrixL[i][r] = koef;

for (int j = 0; j < N; ++j) {

matrix[i][j] -= matrix[r][j] \* koef;

}

//solve[i] -= solve[r] \* koef;

}

}

answerY[0] = solve[0];

for (int i = 1; i < N; ++i) {

TDrob value = 0;

for (int j = 0; j < i; ++j) {

value += answerY[j] \* matrixL[i][j];

}

answerY[i] = solve[i] - value;

}

answer[N - 1] = answerY[N - 1] / matrix[N - 1][N - 1];

for (int i = N - 2; i >= 0; --i) {

TDrob value = 0;

for (int j = N - 1; j > i; --j) {

value += answer[j] \* matrix[i][j];

}

answer[i] = (answerY[i] - value) / matrix[i][i];

}

for (int i = 0; i < N; ++i) {

det \*= matrix[i][i];

}

for (int i = 0; i < N; ++i) {

matrixLObr[i][i] = 1 / matrixL[i][i];

matrixUObr[i][i] = 1 / matrix[i][i];

}

for (int i = 1; i < N; ++i) {

for (int j = 0; j < i; ++j) {

TDrob value = 0;

for (int k = j; k < i; ++k) {

value -= matrixL[i][k] \* matrixLObr[k][j];

}

matrixLObr[i][j] = value / matrixL[i][i];

}

}

for (int i = N - 2; i >= 0; --i) {

for (int j = N - 1; j > i; --j) {

TDrob value = 0;

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

value -= matrix[i][k] \* matrixUObr[k][j];

}

matrixUObr[i][j] = value / matrix[i][i];

}

}

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

for (int k = 0; k < N; ++k) {

matrixObr[i][j] += matrixUObr[i][k] \* matrixLObr[k][j];

}

}

}

cout << "Ответ:\n";

for (int i = 0; i < N; ++i) {

cout << answer[i] << "\n";

}

cout << "Определитель:\n";

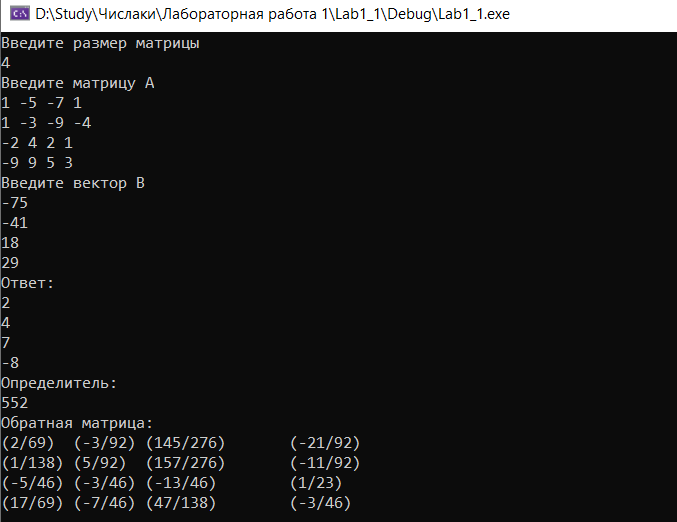
cout << det << "\n";

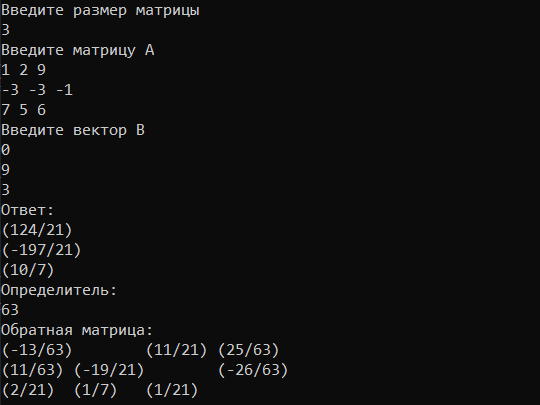
cout << "Обратная матрица:\n";

PrintMatrix(matrixObr);

}

1. Тесты (скриншоты – мой вариант + рандомный тест)





1. Данная лабораторная работа выполнена: **10 марта 2020.**

**Лабораторная работа 1.2**

1. Тема ЛР:

Реализовать метод прогонки в виде программы, задавая в качестве входных данных ненулевые элементы матрицы системы и вектор правых частей. Используя разработанное программное обеспечение, решить СЛАУ с трехдиагональной матрицей.

1. Вариант : **7**



1. Алгоритм:

Метод прогонки позволяет решать только системы линейных уравнений с тремя диагоналями. То есть, в каждой строке не больше трех элементов, которые идут подряд, а с каждой строкой их позиции сдвигаются вправо на 1. Остальные элементы равны нулю. Работает он не сложно. У нас есть цикл длительностью размерности матрицы, а также набор значений альфа и бета, которые нам пригодятся для решения СЛАУ методом прогонки. Каждый раз вычисляем коэффициенты A0, B0, C0, F0, по формулам находим альфы и беты, каждые последующие находятся в зависимости от предыдущих. Последний элемент решения равен последней бете, затем обратным циклом вычисляем предыдущие по формуле.

1. Среда разработки:

Visual Studio 2019 , язык – C++

1. Реализация

Тоже использовал ту свою библиотеку дробных чисел. Вводятся матрица, вектор B. Далее, цикл, который вычисляет значения, альфы, беты. После чего, обратным циклом находится ответ. Все довольно просто. Однако работает только с определенными матрицами.

1. Код (С++)

#include<iostream>

#include<vector>

#include "TDrob.h"

#include "WorkMatrix.h"

const int N = 5;

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

cout << "Введите матрицу 5х5\n";

vector<vector<TDrob> >matrix(N, vector<TDrob>(N, 0));

vector<TDrob>vectorB(N, 0);

vector<TDrob>vectorX(N, 0);

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

cin >> matrix[i][j];

}

}

cout << "Введите вектор B\n";

for (int i = 0; i < N; ++i) {

cin >> vectorB[i];

}

vector<TDrob>alphas(N, 0);

vector<TDrob>betas(N, 0);

//Прямой ход прогонки

for (int i = 0; i < N; ++i) {

TDrob A0, C0, B0, F0;

if (i - 1 < 0) {

A0 = 0;

}

else {

A0 = matrix[i][i - 1];

}

C0 = -1 \* matrix[i][i];

if (i + 1 < N) {

B0 = matrix[i][i + 1];

}

else {

B0 = 0;

}

F0 = vectorB[i];

if (i == 0) {

alphas[i] = B0 / C0;

betas[i] = -(F0 / C0);

}

else if (i == N - 1) {

alphas[i] = 0;

betas[i] = (betas[i - 1] \* A0 - F0) / (C0 - alphas[i - 1] \* A0);

}

else {

alphas[i] = B0 / (C0 - alphas[i - 1] \* A0);

betas[i] = (betas[i - 1] \* A0 - F0) / (C0 - alphas[i - 1] \* A0);

}

}

vectorX[N - 1] = betas[N - 1];

for (int i = 2; i <= N; ++i) {

vectorX[N - i] = alphas[N - i] \* vectorX[N - i + 1] + betas[N - i];

}

cout << "Ответ:\n";

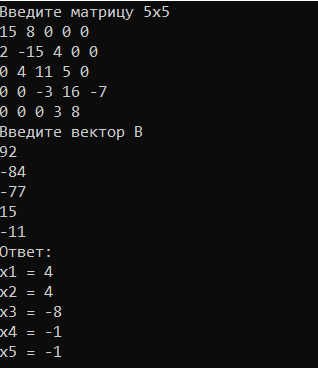
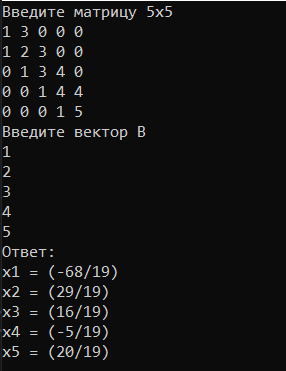
for (int i = 0; i < N; ++i) {

cout << "x" << (i + 1) << " = " << vectorX[i] << "\n";

}

}

1. Тесты (скриншоты – мой вариант + рандомный тест)

1. Данная лабораторная работа выполнена: **15 апреля 2020.**

**Лабораторная работа 1.3**

1. Тема ЛР:

Реализовать метод простых итераций и метод Зейделя в виде программ, задавая в качестве входных данных матрицу системы, вектор правых частей и точность вычислений. Используя разработанное программное обеспечение, решить СЛАУ. Проанализировать количество итераций, необходимое для достижения заданной точности.

1. Вариант : **7**



1. Алгоритм:

Метод простых итераций и метод Зейделя очень похожи, можно даже сказать, что они почти аналогичны. Мы берем нулевой вектор и подставляем его в систему. Каждый раз решаем ее, а ответы кладем в этот же вектор, и так повторяем, пока разница между векторами не станет очень маленькой (определяется погрешностью). Метод Зейделя – практически то же самое, только мы обновляем значения вектора после каждой строчки, а не после решения полностью.

1. Среда разработки:

Visual Studio 2019 , язык – C++

1. Реализация

Так как в данной ЛР используется погрешность, то с дробным типом чисел будет не очень удобно работать, решил отказаться от них, вместо этого используя обычный тип double. Определены две процедуры, которые практически аналогичны, принимают значения по ссылке, ничего не возвращаются, но при этом изменяются, так как по ссылке. Принимает пустой вектор решений, он меняется со временем. Процедура метода Зейделя отличается лишь тем, что она обновляет значения после решений каждого уравнения, а не всей системы. Также принимает значение максимальной погрешностью, и сравнивает значения по LU методу с этим, и когда макс погрешность < заданной, то итерации прекращаются. Как мы видим, метод Зейделя работает быстрее и эффективнее.

1. Код (С++)

#include<iostream>

#include<vector>

#include<cmath>

#include<cstdlib>

using namespace std;

void LUMethod(vector<vector<double> >matrix, vector<double>solve, int N, vector<double>& answer) {

vector<vector<double> >matrixL(N, vector<double>(N, 0));

vector<double>answerY(N, 0);

for (int i = 0; i < N; ++i) {

matrixL[i][i] = 1;

}

for (int r = 0; r < N - 1; ++r) {

for (int i = r + 1; i < N; ++i) {

double koef = matrix[i][r] / matrix[r][r];

matrixL[i][r] = koef;

for (int j = 0; j < N; ++j) {

matrix[i][j] -= matrix[r][j] \* koef;

}

}

}

answerY[0] = solve[0];

for (int i = 1; i < N; ++i) {

double value = 0;

for (int j = 0; j < i; ++j) {

value += answerY[j] \* matrixL[i][j];

}

answerY[i] = solve[i] - value;

}

answer[N - 1] = answerY[N - 1] / matrix[N - 1][N - 1];

for (int i = N - 2; i >= 0; --i) {

double value = 0;

for (int j = N - 1; j > i; --j) {

value += answer[j] \* matrix[i][j];

}

answer[i] = (answerY[i] - value) / matrix[i][i];

}

}

void Iterations(vector<vector<double> >matrix, vector<double>solve, int N, vector<double> answer1, vector<double>& answer2, double& maxPogr, int& iterations) {

bool pogrGot = false;

bool pogrNew = false;

vector<double>answerX(N, 0);

while (!pogrGot) {

for (int i = 0; i < N; ++i) {

answerX[i] = solve[i] / matrix[i][i];

for (int j = 0; j < N; ++j) {

if (i != j) {

answerX[i] -= matrix[i][j] / matrix[i][i] \* answer2[j];

}

}

}

for (int i = 0; i < N; ++i) {

answer2[i] = answerX[i];

}

++iterations;

//Check pogr

pogrNew = true;

for (int i = 0; i < N; ++i) {

if (abs(answer2[i] - answer1[i]) > maxPogr) {

pogrNew = false;

}

}

pogrGot = pogrNew;

}

}

void Zeidell(vector<vector<double> >matrix, vector<double>solve, int N, vector<double> answer1, vector<double>& answer2, double& maxPogr, int& iterations) {

bool pogrGot = false;

bool pogrNew = false;

while (!pogrGot) {

for (int i = 0; i < N; ++i) {

answer2[i] = solve[i] / matrix[i][i];

for (int j = 0; j < N; ++j) {

if (i != j) {

answer2[i] -= matrix[i][j] / matrix[i][i] \* answer2[j];

}

}

}

++iterations;

//Check pogr

pogrNew = true;

for (int i = 0; i < N; ++i) {

if (abs(answer2[i] - answer1[i]) > maxPogr) {

pogrNew = false;

}

}

pogrGot = pogrNew;

}

}

int main() {

setlocale(LC\_ALL, "Russian");

cout << "Введите размер матрицы\n";

int N;

cin >> N;

vector<vector<double> >matrix(N, vector<double>(N, 0));

vector<double>solve(N, 0);

vector<double>answer1(N, 0);

vector<double>answer2(N, 0);

double maxPogr;

int iterations = 0;

int iterations2 = 0;

vector<double>answer3(N, 0);

cout << "Введите матрицу A\n";

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

cin >> matrix[i][j];

}

}

cout << "Введите вектор B\n";

for (int i = 0; i < N; ++i) {

cin >> solve[i];

}

cout << "Введите максимальную погрешность\n";

cin >> maxPogr;

LUMethod(matrix, solve, N, answer1);

Iterations(matrix, solve, N, answer1, answer2, maxPogr, iterations);

Zeidell(matrix, solve, N, answer1, answer3, maxPogr, iterations2);

cout << "\nОтвет (Метод LU):\n";

for (int i = 0; i < N; ++i) {

cout << answer1[i] << "\n";

}

cout << "\nОтвет (Метод простых итераций):\n";

for (int i = 0; i < N; ++i) {

cout << answer2[i] << "\n";

}

cout << "Количество итераций: " << iterations << "\n";

cout << "\nОтвет (Метод Зейделя):\n";

for (int i = 0; i < N; ++i) {

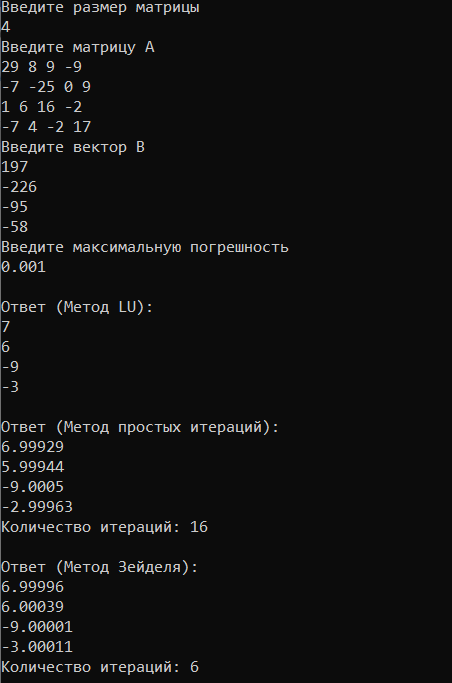
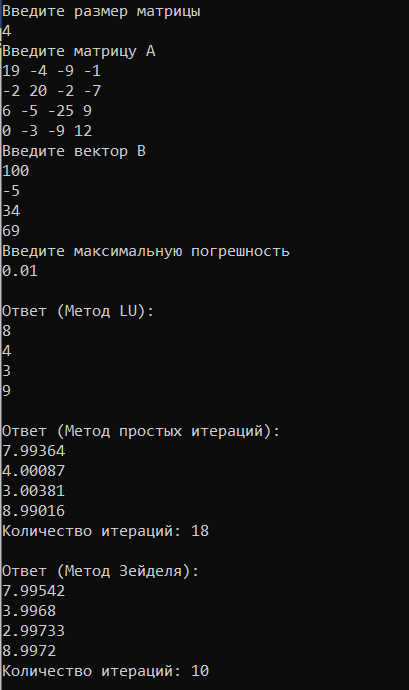
cout << answer3[i] << "\n";

}

cout << "Количество итераций: " << iterations2 << "\n";

}

1. Тесты (скриншоты – мой вариант + 1)

1. Данная лабораторная работа выполнена: **16 апреля 2020.**

**Лабораторная работа 1.4**

1. Тема ЛР:

Реализовать метод вращений в виде программы, задавая в качестве входных данных матрицу и точность вычислений. Используя разработанное программное обеспечение, найти собственные значения и собственные векторы симметрических матриц. Проанализировать зависимость погрешности вычислений от числа итераций.

1. Вариант : **7**



1. Алгоритм:

Довольно сложный метод, заключающийся в том, что мы так же, как и в прошлой ЛР делаем его итерационно. В этот раз мы умножаем матрицу на матрицу поворота. Выбираем угол, в зависимости от максимального элемента в матрице. Элементы на диагонали = 1. Элементы на позициях, ii, ij, ji, jj, где ij – позиция макс. элемента – синусы и косинусы угла. Остальные = 0. Транспонируем матрицу поворота, умножаем на исходную, затем умножаем на матрицу поворота. Делаем так, пока погрешность не станет слишком маленькой. На диагонали главной – собственные значения. Долго пытался разобраться, как найти собственные векторы. Для этого завел еще одну матрицу, столбцы которой и будут СВ. Изначально на главной диагонали 1, остальные = 0. Во время вычислений мы просто много раз умножаем эту матрицу на матрицу поворота, делим на мин. значение, для того, чтобы СВ выглядели более красиво и имели единицу.

1. Среда разработки:

Visual Studio 2019 , язык – C++

1. Реализация

Думаю, тут особо нечего писать. Все выполняется в main(). Цикл используется do while(), так удобнее для данной ЛР, чтобы условие проверялось в конце цикла. А условие – проверка на погрешность. После чего выводятся СЗ и СВ. Матрицы должны быть симметрическими.

1. Код (С++)

#include<iostream>

#include<vector>

#include<cmath>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

int i, j;

int n = 3;

vector<vector<double> >A(n, vector<double>(n, 0));

cout << "Введите матрицу 3х3: \n";

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

cin >> A[i][j];

}

}

double f;

double eps = 0.3;

cout << "Введите макс. погрешность: ";

cin >> eps;

cout << "\n\n";

double t = 0;

int iter = 0;

vector<vector<double> >V(n, vector<double>(n, 0));

for (int i = 0; i < n; ++i) {

V[i][i] = 1;

}

do {

int maxi = 0, maxj = 0;

double max = 0;

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

if (i < j && abs(A[i][j]) > max) {

max = abs(A[i][j]);

maxi = i;

maxj = j;

}

}

}

f = 0.5 \* atan(2 \* A[maxi][maxj] / (A[maxi][maxi] - A[maxj][maxj]));

vector<vector<double> >U(n, vector<double>(n, 0));

U[maxi][maxj] = -sin(f);

U[maxj][maxi] = sin(f);

U[maxi][maxi] = cos(f);

U[maxj][maxj] = cos(f);

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

if (i != j && i != maxi && j != maxj && i != maxj && j != maxi) {

U[i][j] = 0;

}

}

}

for (int m = 0; m < n; m++) {

if (m != maxi && m != maxj) {

U[m][m] = 1;

}

}

vector<vector<double> > matrixtrans(n, vector<double>(n, 0));

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

matrixtrans[j][i] = U[i][j];

}

}

vector<vector<double> > C(n, vector<double>(n, 0));

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

for (int k = 0; k < n; k++) {

C[i][j] += matrixtrans[i][k] \* A[k][j];

}

}

}

vector<vector<double> >A1(n, vector<double>(n, 0));

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

for (int k = 0; k < n; k++) {

A1[i][j] += C[i][k] \* U[k][j];

}

A[i][j] = A1[i][j];

}

}

double b = 0;

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

if (i != j) {

b += 0.5 \* pow(A1[i][j], 2);

}

}

}

t = pow(b, 0.5);

iter++;

vector<vector<double> >newV(n, vector<double>(n, 0));

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

for (int k = 0; k < n; ++k) {

newV[i][j] += V[i][k] \* U[k][j];

}

}

}

for (int i = 0; i < n; ++i) {

for (int j = 0; j < n; ++j) {

V[i][j] = newV[i][j];

}

}

} while (t > eps);

for (int i = 0; i < n; ++i) {

double minVal = 999999999999;

for (int j = 0; j < n; ++j) {

if (abs(V[j][i]) < abs(minVal)) {

minVal = V[j][i];

}

}

for (int j = 0; j < n; ++j) {

V[j][i] /= minVal;

}

}

vector<double> lyamda(n);

for (int i = 0; i < n; ++i) {

lyamda[i] = A[i][i];

}

cout << "Собственные значения: ";

for (int i = 0; i < n; ++i) {

cout << lyamda[i];

if (i != n - 1) {

cout << ", ";

}

}

cout << "\n\nСобственные вектора: \n";

for (int i = 0; i < n; i++) {

cout << "\nC" << i << "^T = {";

for (int j = 0; j < n; ++j) {

cout << V[j][i];

if (j != n - 1) {

cout << ", ";

}

}

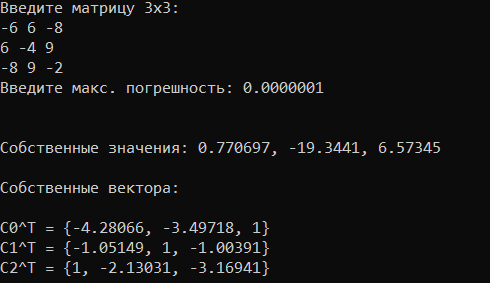
cout << "}";

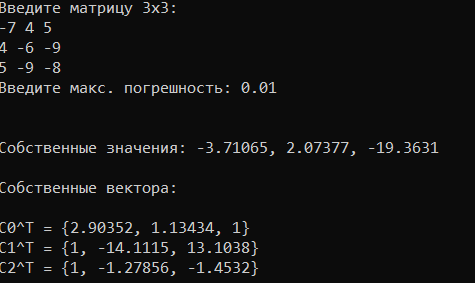
}

cout << "\n\n";

}

1. Тесты (скриншоты – мой вариант + 1)





1. Данная лабораторная работа выполнена: **23 апреля 2020.**

**Лабораторная работа 1.5**

1. Тема ЛР:

Реализовать алгоритм QR – разложения матриц в виде программы. На его основе разработать программу, реализующую QR – алгоритм решения полной проблемы собственных значений произвольных матриц, задавая в качестве входных данных матрицу и точность вычислений. С использованием разработанного программного обеспечения найти собственные значения матрицы.

1. Вариант : **7**



1. Алгоритм:

Данный метод отличается от предыдущего тем, что могут быть найдены комплексные корни, а также матрица не должна быть обязательно симметрической. Из матрицы создаются две новые – Q и R матрицы. Изначально на главной диагонали матрицы Q единицы, а остальные – нули. В каждой итерации берем вектор. Элементы до значения итерации в нем = 0. Если элемент вектора равен итерации, то выполняем следующее. Ищем квадратичную сумму элементов под главной диагональю. Берем корень и ставим знак элемента на [i][i] позиции. После этого элемента = значению в матрице на [j][i] позиции. Ищем квадратичную сумму элементов вектора. Создаем матрицу из вектора, где элемент на позиции [i][j] равен произведению [i] и [j] элементов из вектора. Получаем матрицу H. Для ее получения из единичной матрицы (где на главной диагонали единицы) вычитаем 2 матрицы из того вектора, деленных на ту квадратичную сумму его элементов. Меняем матрицу Q, умножая ее на матрицу H. Изменяем текущую матрицу, умножая ее на матрицу H слева. Когда погрешность будет очень маленькой, матрица R и будет текущей матрицей. Мы можем найти обратную матрицу, умножив матрицу R на матрицу Q. Проверяем элементы в первом столбце. 2 и 3 должны быть равны 0, а на диагонали – первое СЗ. Если элементы больше погрешности, повторяем нахождение Q и R, только уже для обратной этой матрицы, пока они не станут меньше погрешности. Первое СЗ – действительное. Второе и третье – комплексные, находим их обычным решением через дискриминант.

1. Среда разработки:

Visual Studio 2019 , язык – C++

1. Реализация

Очень сложная лаба, пришлось долго разбираться с алгоритмом. Да еще и быстро реализовать класс комплексных чисел, так как 2 и 3 корни тут комплексные. Выполняется алгоритм много раз, но иногда могут попадаться очень маленькие значения, где должны быть нули, поэтому при этом они обращаются в 0. После алгоритма нахожу СЗ и вывожу их.

1. Код (С++)

#include<iostream>

#include<vector>

#include "imag.h"

#include <iomanip>

const int N = 3;

using namespace std;

double signF(double a) {

if (a < 0) {

return -1;

}

else {

return 1;

}

}

const double minNum = 1e-12;

void getQR(vector<vector<double> > matrixA, vector<vector<double> >& matrQ, vector<vector<double> >& matrR) {

vector<vector<double> >matrixE(N, vector<double>(N, 0));

vector<vector<double> >matrixCur(N, vector<double>(N, 0));

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

if (i != j) {

matrQ[i][j] = 0;

}

}

}

for (int i = 0; i < N; ++i) {

matrixE[i][i] = 1;

matrQ[i][i] = 1;

}

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

matrixCur[i][j] = matrixA[i][j];

}

}

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

vector<double> vectorV(N, 0);

for (int j = 0; j < N; ++j) {

if (j < i) {

vectorV[j] = 0;

}

else if (j == i) {

double sumGet = 0;

for (int k = j; k < N; ++k) {

sumGet += matrixCur[k][j] \* matrixCur[k][j];

}

vectorV[j] = matrixCur[i][i] + signF(matrixCur[i][i]) \* sqrt(sumGet);

}

else {

vectorV[j] = matrixCur[j][i];

}

}

double vectorNum = 0;

vector<vector<double> > vectorMatr(N, vector<double>(N, 0));

vector<vector<double> > MatrH(N, vector<double>(N, 0));

vector<vector<double> > MatrA(N, vector<double>(N, 0));

for (int i = 0; i < N; ++i) {

vectorNum += vectorV[i] \* vectorV[i];

}

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

vectorMatr[i][j] = vectorV[i] \* vectorV[j];

}

}

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

MatrH[i][j] = matrixE[i][j] - (2 \* vectorMatr[i][j] / vectorNum);

}

}

vector<vector<double> > MatrRes(N, vector<double>(N, 0));

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

for (int k = 0; k < N; ++k) {

MatrRes[i][j] += matrQ[i][k] \* MatrH[k][j];

}

}

}

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

if (abs(MatrRes[i][j]) > minNum) {

matrQ[i][j] = MatrRes[i][j];

}

else {

matrQ[i][j] = 0;

}

}

}

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

for (int k = 0; k < N; ++k) {

MatrA[i][j] += MatrH[i][k] \* matrixCur[k][j];

}

}

}

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

matrixCur[i][j] = MatrA[i][j];

}

}

}

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

if (abs(matrixCur[i][j]) > minNum) {

matrR[i][j] = matrixCur[i][j];

}

else {

matrR[i][j] = 0;

}

}

}

}

int main() {

setlocale(LC\_ALL, "Russian");

cout << setprecision(5);

//Разложение QR

vector<vector<double> >matrix(N, vector<double>(N, 0));

cout << "Введите матрицу 3x3: \n";

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

cin >> matrix[i][j];

}

}

double eps;

cout << "Введите макс. погрешность (точность): \n";

cin >> eps;

vector<vector<double> >matrQ(N, vector<double>(N, 0));

vector<vector<double> >matrR(N, vector<double>(N, 0));

getQR(matrix, matrQ, matrR);

bool trigger = true;

int iters = 1;

while (trigger) {

vector<vector<double> >invA(N, vector<double>(N, 0));

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

for (int k = 0; k < N; ++k) {

invA[i][j] += matrR[i][k] \* matrQ[k][j];

}

}

}

if (abs(invA[1][0]) < eps && abs(invA[2][0]) < eps) {

trigger = false;

}

if (trigger) {

getQR(invA, matrQ, matrR);

++iters;

}

}

vector<vector<double> >ansMatrix(N, vector<double>(N, 0));

for (int i = 0; i < N; ++i) {

for (int j = 0; j < N; ++j) {

for (int k = 0; k < N; ++k) {

ansMatrix[i][j] += matrR[i][k] \* matrQ[k][j];

}

}

}

vector<Imag> answers(0);

answers.push\_back(Imag(ansMatrix[0][0]));

//Решение квадратного уравнения

double A = 1;

double B = -ansMatrix[1][1] - ansMatrix[2][2];

double C = ansMatrix[1][1] \* ansMatrix[2][2] - ansMatrix[1][2] \* ansMatrix[2][1];

Imag iA = Imag(A);

Imag iB = Imag(B);

Imag iC = Imag(C);

Imag D = iB \* iB - 4 \* iA \* iC;

Imag Answer2 = (-iB + Imag::sqrt(D)) / (2 \* iA);

Imag Answer3 = (-iB - Imag::sqrt(D)) / (2 \* iA);

answers.push\_back(Answer2);

answers.push\_back(Answer3);

cout << "\nСобственные значения:\n";

for (int i = 0; i < answers.size(); ++i) {

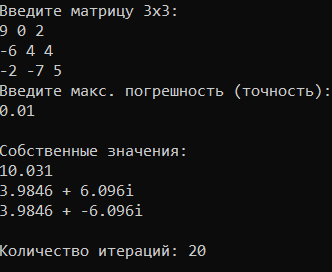
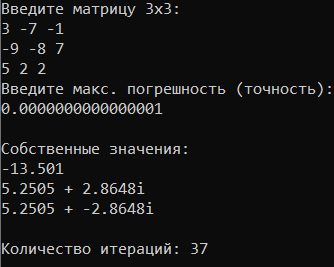
cout << answers[i] << "\n";

}

cout << "\nКоличество итераций: " << iters << "\n";

}

1. Тесты (скриншоты – мой вариант + 1)

1. Данная лабораторная работа выполнена: **25 апреля 2020.**

**Лабораторная работа 2.1**

1. Тема ЛР:

Реализовать методы простой итерации и Ньютона решения нелинейных уравнений в виде программ, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения найти положительный корень нелинейного уравнения (начальное приближение определить графически). Проанализировать зависимость погрешности вычислений от количества итераций.

1. Вариант : **7**

.

1. Алгоритм:

Метод простой итерации тут схож с методом в ЛР 1.3, только здесь используется для одного нелинейного уравнения. Я нашел начальное приближение графически на сайте. Заметил, что функцию Phi лучше выразить через x^2, ибо я пробовал через 2^x, но условия не удовлетворяли, а Q получался больше 1. Нашел производные исходной функции и функции Phi и ввел их как функции в программу. Метод простых итераций довольно прост. Находим начальное приближение, приравниваем X ему. Вставляем в функцию фи, ибо она равна как раз X и заново присваиваем, пока eps > погрешности. Eps вычисляется через спец. функцию, где как раз используется Q, который должен быть < 1 , а Q вычисляется через начальное приближение как максимум на нем в функции производной Phi. Метод Ньютона похож, но работает намного! быстрее и эффективнее. Выбираем за начальное приближение a или b, нам без разницы, главное, чтобы условие выполнялось: произведение производной 1 и 2 порядков в этой точке > 0. Тогда мы просто каждый раз из этой точки вычитаем значение функции в ней деленное на производную в ней, пока разница не станет слишком маленькой.

1. Среда разработки:

Visual Studio 2019 , язык – C++

1. Реализация

По сравнению с предыдущей лабой, кажется простой. Задал функцию в программе, нашел собственноручно ее производные, а также функцию Phi и ее производную. После чего, уже действовал по алгоритму и решил уравнение методом итераций и методом Ньютона.

1. Код (С++)

#include<iostream>

#include<cmath>

#include<algorithm>

#include<vector>

using namespace std;

double function(double x) {

return pow(2, x) + pow(x, 2) - 2; //=0

}

double der1(double x) {

// (2^x + x^2 - 2)' = ln2 \* 2^x + 2x

return log(2) \* pow(2, x) + 2 \* x;

}

double der2(double x) {

// (ln2 \* 2^x + 2x)' = ln2 \* ln2 \* 2^x + 2

return log(2) \* log(2) \* pow(2, x) + 2;

}

double phi(double x) {

return sqrt(2 - pow(2, x)); //= x [0.6, 0.7]

//x^2 = 2 - 2^x

//x = sqrt(2 - 2^x)

}

double phi\_der(double x) {

return pow(2, x) \* log(2) / (2 \* sqrt(2 - pow(2, x)));

//(sqrt(2 - 2^x))' = 2^x \* ln2 / (2 \* sqrt(2 - 2^x))

}

bool getEPS(double q, double newX, double prevX, double eps) {

return ((q / (1 - q)) \* abs(newX - prevX)) <= eps;

}

int main() {

setlocale(LC\_ALL, "Russian");

//Метод простых итераций

double eps;

cout << "Введите точность E: ";

cin >> eps;

cout << "\n\nМетод простых итераций: \n\n";

double a = 0.6;

double b = 0.7;

double q = abs(max(phi\_der(a), phi\_der(b)));

double newX = (a + b) / 2;

double prevX = -999999999;

int iterations = 0;

while (!getEPS(q, newX, prevX, eps)) {

prevX = newX;

newX = phi(prevX);

iterations++;

}

cout << "Ответ: x = " << newX << "\n";

cout << "Количество итераций: " << iterations << "\n";

//Метод Ньютона

cout << "\n\nМетод Ньютона: \n\n";

if (der1(a) \* der2(a) > 0) {

newX = a;

}

else if (der1(b) \* der2(b) > 0) {

newX = b;

}

prevX = -999999999;

iterations = 0;

while (!(abs(newX - prevX) < eps)) {

prevX = newX;

newX = prevX - (function(prevX) / der1(prevX));

++iterations;

}

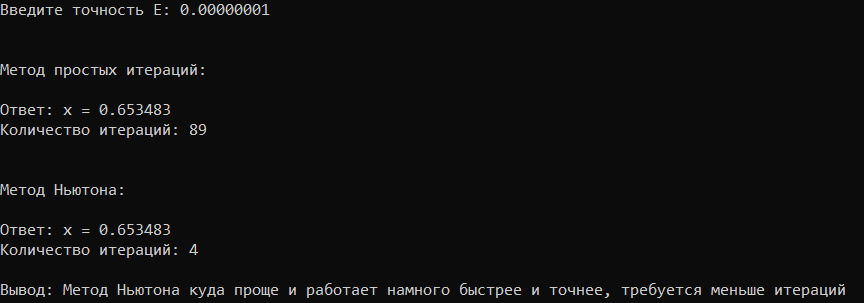
cout << "Ответ: x = " << newX << "\n";

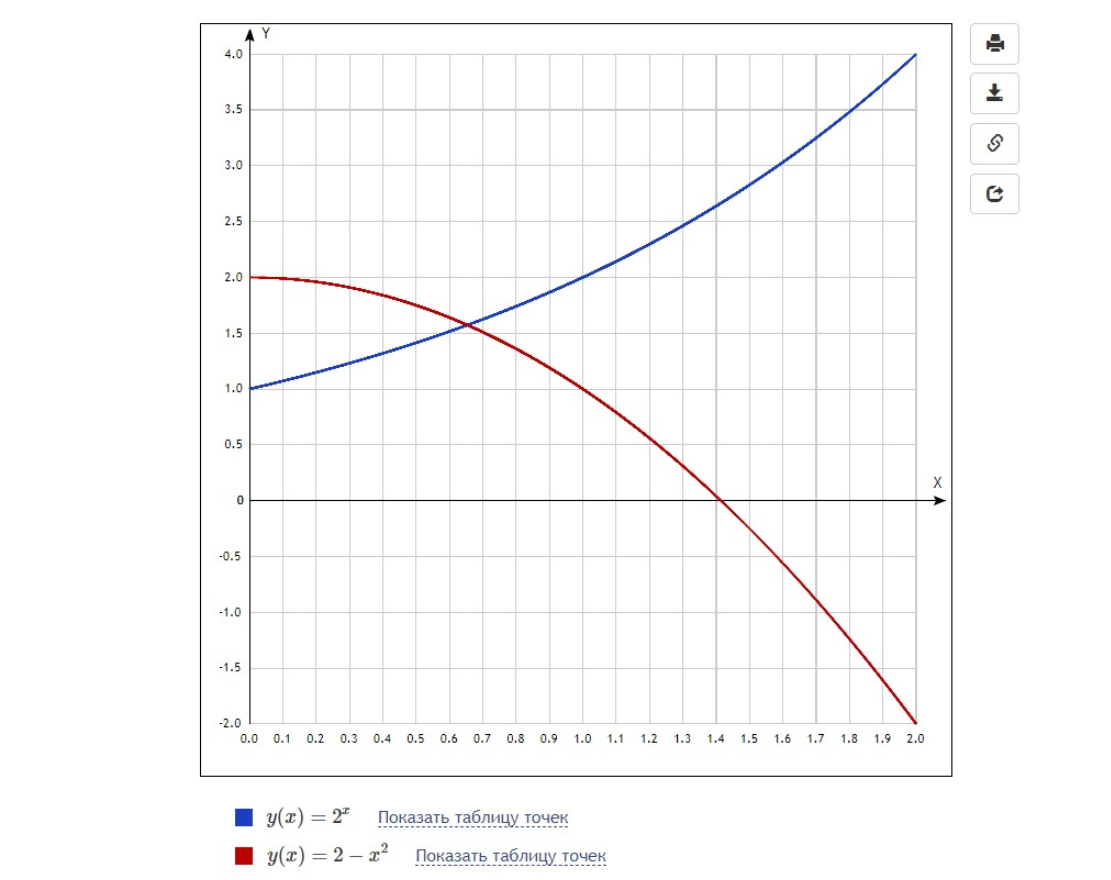
cout << "Количество итераций: " << iterations << "\n\n";

cout << "Вывод: Метод Ньютона куда проще и работает намного быстрее и точнее, требуется меньше итераций\n";

}

1. Тест (скриншот – мой вариант + начальное приближение)





1. Данная лабораторная работа выполнена: **29 апреля 2020.**

**Лабораторная работа 2.2**

1. Тема ЛР:

Реализовать методы простой итерации и Ньютона решения систем нелинейных уравнений в виде программного кода, задавая в качестве входных данных точность вычислений. С использованием разработанного программного обеспечения решить систему нелинейных уравнений (при наличии нескольких решений найти то из них, в котором значения неизвестных являются положительными); начальное приближение определить графически. Проанализировать зависимость погрешности вычислений от количества итераций.

1. Вариант : **7**

 *a* = 2.

1. Алгоритм:

Можно сказать, что лаба очень похожа на предыдущая. Отличие в том, что тут надо решить систему из 2 нелинейный уравнений и по 2 переменным. Так же, найти приближение, в этот раз – точка, а не отрезок. Реализовал функции в коде. Нашел произодные их по обеим переменным. Нашел Phi1 и Phi2, их производные тоже. Определители матриц, где столбцы – производные по переменной и функции. Находится начальное приближение. Вычисляется Q , как макс. сумма производных. В методе простых итерации так же переприсваиваются переменные через функции Phi1 и Phi2. В методе Ньютона вычитается частное определителей, пока оно не станет слишком маленьким. В конце концов выводится ответ и количество итераций.

1. Среда разработки:

Visual Studio 2019 , язык – C++

1. Реализация

Не думаю, что тут есть смысл что-то писать. Я нашел производные, функции, Phi, написал код, а точнее, я взял код прошлой лабы и переделал. Я сделал ее примерно за час, если не быстрее, так как на основе предыдущей лабы, задача оказалась проще.

1. Код (С++)

#include<iostream>

#include<cmath>

#include<algorithm>

#include<vector>

using namespace std;

const double E = 2.71828182846;

double f1(double x1, double x2) {

return pow(x1, 2) + pow(x2, 2) - 4;

}

double f2(double x1, double x2) {

return x1 - pow(E, x2) + 2;

}

double df1dx1(double x1, double x2) {

return 2 \* x1;

}

double df1dx2(double x1, double x2) {

return 2 \* x2;

}

double df2dx1(double x1, double x2) {

return 1;

}

double df2dx2(double x1, double x2) {

return -pow(E, x2);

}

double phi1(double x1, double x2) {

return sqrt(4 - pow(x2, 2));

//x1^2 = 4 - x2^2 //не сч отриц => x = sqrt(4 - x2^2)

}

double phi2(double x1, double x2) {

return log(x1 + 2);

//e^x2 = 2 + x1 => x2 = ln(x1 + 2)

}

double dphi1dx1(double x1, double x2) {

return 0;

}

double dphi1dx2(double x1, double x2) {

return -x2 / sqrt(4 - pow(x2, 2));

}

double dphi2dx1(double x1, double x2) {

return 1 / (x1 + 2);

}

double dphi2dx2(double x1, double x2) {

return 0;

}

double find\_der(int i, int j, double x1, double x2) {

if (i == 1) {

if (j == 1) {

return dphi1dx1(x1, x2);

}

else {

return dphi1dx2(x1, x2);

}

}

else {

if (j == 1) {

return dphi2dx1(x1, x2);

}

else {

return dphi2dx2(x1, x2);

}

}

}

bool cond(double xPrev, double xNext, double eps, double q) {

return (q / (1 - q)) \* abs(xNext - xPrev) <= eps;

}

double detJ(double x1, double x2) {

return df1dx1(x1, x2) \* df2dx2(x1, x2) - df1dx2(x1, x2) \* df2dx1(x1, x2);

}

double detA1(double x1, double x2) {

return f1(x1, x2) \* df2dx2(x1, x2) - df1dx2(x1, x2) \* f2(x1, x2);

}

double detA2(double x1, double x2) {

return df1dx1(x1, x2) \* f2(x1, x2) - f1(x1, x2) \* df2dx1(x1, x2);

}

int main() {

setlocale(LC\_ALL, "Russian");

double eps;

cout << "Введите точность E: ";

cin >> eps;

cout << "\nМетод итераций: \n\n";

//Метод Итераций

double p\_x1 = 1.5;

double p\_x2 = 1.25;

double q = max((find\_der(1, 1, p\_x1, p\_x2) + find\_der(1, 2, p\_x1, p\_x2)), (find\_der(2, 1, p\_x1, p\_x2) + find\_der(2, 2, p\_x1, p\_x2)));

//cout << q; q ~ 0.28 < 1 => все ОК!

double x1Prev = -999999999;

double x1Next = p\_x1;

double x2Prev = -999999999;

double x2Next = p\_x2;

int iterations = 0;

bool maxIter = false;

while (!cond(x1Prev, x1Next, eps, q) || !cond(x2Prev, x2Next, eps, q)) {

x1Prev = x1Next;

x2Prev = x2Next;

x1Next = phi1(x1Prev, x2Prev);

x2Next = phi2(x1Prev, x2Prev);

iterations++;

if (iterations >= 5000) {

maxIter = true;

break;

}

}

cout << "x1 = " << x1Next << "\n";

cout << "x2 = " << x2Next << "\n";

if (!maxIter) {

cout << "Количество итераций: " << iterations << "\n";

}

else if (maxIter) {

cout << "Количество итераций: " << iterations << " (Макс!)\n";

cout << "Невозможно добиться соответствующей точности.\n";

}

//Метод Итераций

cout << "\nМетод Ньютона: \n\n";

x1Prev = -999999999;

x1Next = p\_x1;

x2Prev = -999999999;

x2Next = p\_x2;

iterations = 0;

while (!(max((x1Next - x1Prev), (x2Next - x2Prev)) <= eps)) {

x1Prev = x1Next;

x2Prev = x2Next;

x1Next = x1Prev - (detA1(x1Prev, x2Prev) / detJ(x1Prev, x2Prev));

x2Next = x2Prev - (detA2(x1Prev, x2Prev) / detJ(x1Prev, x2Prev));

iterations++;

}

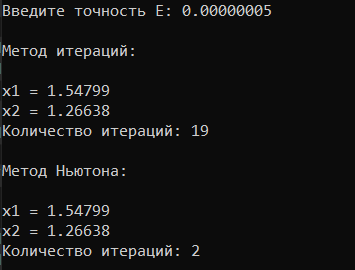
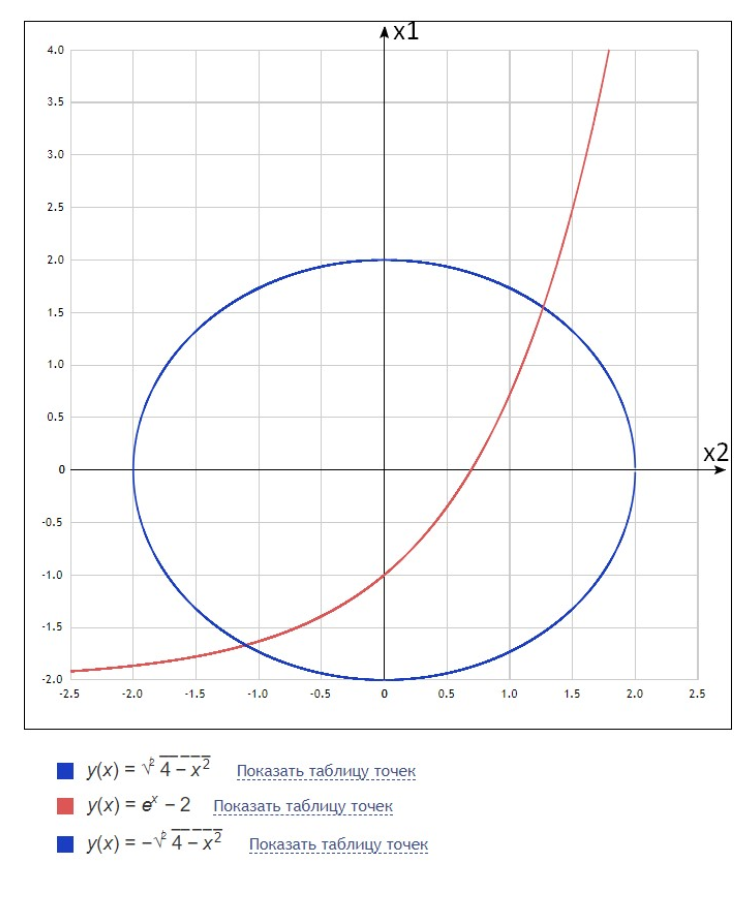
cout << "x1 = " << x1Next << "\n";

cout << "x2 = " << x2Next << "\n";

cout << "Количество итераций: " << iterations << "\n";

}

1. Тест (скриншот – мой вариант + начальное приближение)

1. Данная лабораторная работа выполнена: **30 апреля 2020.**

**Лабораторная работа 3.1**

1. Тема ЛР:

Используя таблицу значений  функции , вычисленных в точках  построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки .Вычислить значение погрешности интерполяции в точке .

1. Вариант : **7**

, a)  ; б) ; .

1. Алгоритм:

Интерполяционные многочлены позволяют приблизить функцию в заданных к точкам к заданной используя лишь многочлены. Многочлен Лагранжа нахожу так, что перемножаю разницы между точками и вывожу их в коэффициенты. Затем делю значения этих точек на эти коэффициенты и получаются коэф. для многочлена. Затем просто вывожу его и сверяю с настоящим значением этой функции в точке X\*, нахожу погрешность. Многочлен Ньютона строится так, что его коэффициенты – разницы между значениями функции деленные на разницу между точками. В конце так же нахожу погрешность.

1. Среда разработки:

Visual Studio 2019 , язык – C++

1. Реализация

Нахожу оба многочлена, вывожу их через цикл, проверяю на знак, для того, чтобы многочлен выводился красивее.

1. Код (С++)

#include<iostream>

#include<vector>

#include<cmath>

#include<algorithm>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

cout << "Вариант 7, функция sqrt(x)\n";

cout << "Интерполяционный многочлен Лагранджа:\n";

int N = 4;

cout << "Введите точки:\n";

vector<double>points\_L(N, 0);

for (int i = 0; i < N; ++i) {

cin >> points\_L[i];

}

vector<double>values\_L(N, 0);

vector<double>Ws\_L(N, 0);

for (int i = 0; i < N; ++i) {

values\_L[i] = sqrt(points\_L[i]);

}

for (int i = 0; i < N; ++i) {

double base = 1;

for (int j = 0; j < N; ++j) {

if (i != j) {

base \*= (points\_L[i] - points\_L[j]);

}

}

Ws\_L[i] = base;

}

vector<double>FdivW(N, 0);

for (int i = 0; i < N; ++i) {

FdivW[i] = values\_L[i] / Ws\_L[i];

}

cout << "\nИнтерполяционный многочлен Лагранджа:\n\n";

bool firstWas = false;

for (int i = 0; i < N; ++i) {

if (firstWas) {

if (FdivW[i] > 0) {

cout << " + " << FdivW[i];

}

else if (FdivW[i] < 0) {

cout << " - " << abs(FdivW[i]);

}

else {

continue;

}

}

else {

if (FdivW[i] != 0) {

cout << FdivW[i];

firstWas = true;

}

else {

continue;

}

}

for (int j = 0; j < N; ++j) {

if (i != j) {

cout << "(x";

if (points\_L[j] > 0) {

cout << " - " << points\_L[j];

}

else if (points\_L[j] < 0) {

cout << " + " << -points\_L[j];

}

cout << ")";

}

}

}

double X\_star = 3.0;

cout << "\n\nX\* = " << X\_star << "\n\n";

double L\_value = 0;

for (int i = 0; i < N; ++i) {

double basI = FdivW[i];

for (int j = 0; j < N; ++j) {

if (i != j) {

basI \*= (X\_star - points\_L[j]);

}

}

L\_value += basI;

}

cout << "L" << (N - 1) << "(X\*) = " << L\_value << "\n";

cout << "Sqrt(" << X\_star << ") = " << sqrt(X\_star) << "\n\n";

cout << "Абсолютная погрешность = " << abs(sqrt(X\_star) - L\_value) << "\n";

cout << "\nИнтерполяционный многочлен Ньютона:\n";

cout << "Введите точки:\n";

vector<double>points\_N(N, 0);

for (int i = 0; i < N; ++i) {

cin >> points\_N[i];

}

vector<double>values\_N(N, 0);

vector<vector<double>>getVals\_N(N, vector<double>(0));

for (int i = 0; i < N; ++i) {

values\_N[i] = sqrt(points\_N[i]);

getVals\_N[0].push\_back(values\_N[i]);

}

int iter = 0;

while (getVals\_N[iter].size() > 1) {

int M = getVals\_N[iter].size();

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

getVals\_N[iter + 1].push\_back((getVals\_N[iter][i + 1] - getVals\_N[iter][i]) / (points\_N[iter + i + 1] - points\_N[i]));

}

iter++;

}

cout << "\nИнтерполяционный многочлен Ньютона:\n\n";

firstWas = false;

for (int i = 0; i < N; ++i) {

if (firstWas) {

if (getVals\_N[i][0] > 0) {

cout << " + " << getVals\_N[i][0];

}

else if (getVals\_N[i][0] < 0) {

cout << " - " << abs(getVals\_N[i][0]);

}

else {

continue;

}

}

else {

if (getVals\_N[i][0] != 0) {

cout << getVals\_N[i][0];

firstWas = true;

}

else {

continue;

}

}

for (int j = 0; j < i; ++j) {

cout << "(x";

if (points\_N[j] > 0) {

cout << " - " << points\_N[j];

}

else if (points\_N[j] < 0) {

cout << " + " << -points\_N[j];

}

cout << ")";

}

}

cout << "\n\nX\* = " << X\_star << "\n\n";

double N\_value = 0;

for (int i = 0; i < N; ++i) {

double basI = getVals\_N[i][0];

for (int j = 0; j < i; ++j) {

basI \*= (X\_star - points\_N[j]);

}

N\_value += basI;

}

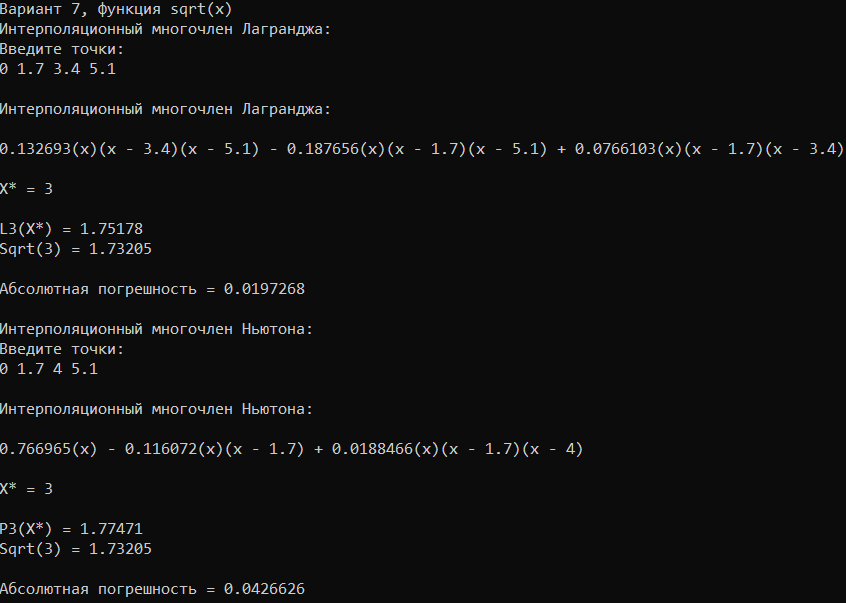
cout << "P" << (N - 1) << "(X\*) = " << N\_value << "\n";

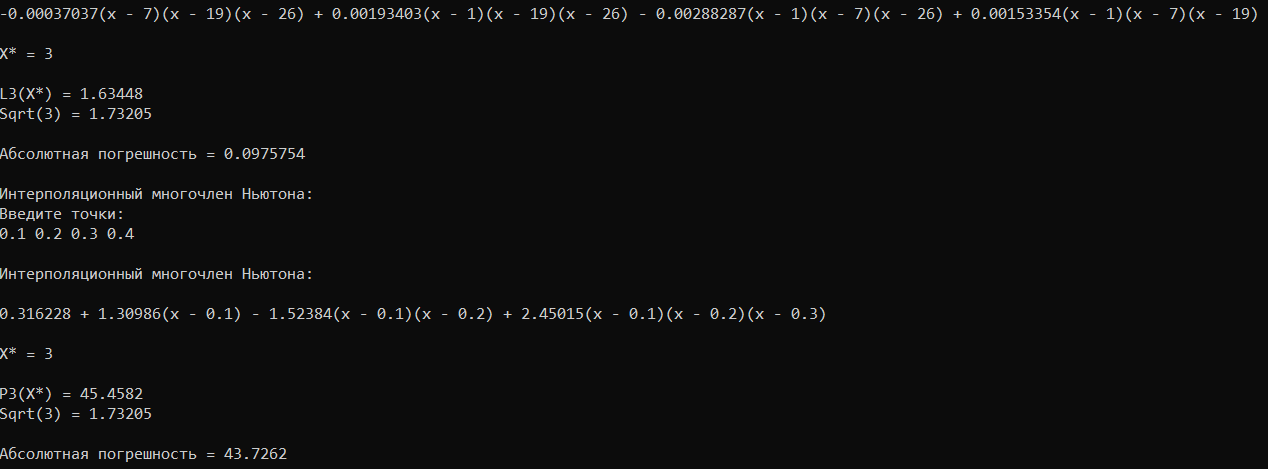
cout << "Sqrt(" << X\_star << ") = " << sqrt(X\_star) << "\n\n";

cout << "Абсолютная погрешность = " << abs(sqrt(X\_star) - N\_value) << "\n";

}

1. Тесты (Мой вариант с разными точками)





1. Данная лабораторная работа выполнена: **30 апреля 2020.**

**Лабораторная работа 3.2**

1. Тема ЛР:

Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при  и . Вычислить значение функции в точке .

1. Вариант : **7**

3.0

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
|  | 0.0 | 1.7 | 3.4 | 5.1 | 6.8 |
|  | 0.0 | 1.3038 | 1.8439 | 2.2583 | 2.6077 |

1. Алгоритм:

Сплайны позволяют “сгладить” функцию с заданными точками, сделать переходы более плавными, предположить, как бы выглядела функция. Программа содержит поле с графиком, показатели, которые можно выставить (аппроксимация – плавность графика и скорость его построения). При нажатии кнопки begin график строится по точкам. Для построения кубического сплайна находятся специальные коэффициенты, после чего, они вставляются в особо заданную функцию, находится значение в этой точке. Для коэффициентов C, например, вообще нужно решить матрицу, поэтому пришлось переписать код из 1.2 лабы на js.

1. Среда разработки:

Adobe Animate CC , язык - Javascript

1. Реализация

Так как в данной ЛР нужно построить график, мне пришлось выбрать ПО, где возможно визуализировать программу. Для этого я выбрал Adobe Animate CC, так как имею опыт работы с ним, и считаю, что он довольно удобен. Язык программирования – javascript. Построение графика идет по кадрам, добавляются точки, соединяются линиями. Линия – отрезок, длина которого зависит от расстояния между 2 точками, а поворот задается тангенсом. Можно регулировать скорость, аппроксимацию. Также находится сразу значение в точке. Все просчитывается предварительно после нажатия сразу, а при нажатии на кнопку, уже идет прорисовка. В архиве есть видео, где показана работа программы для этой ЛР и нескольких других. Запуск программы идет обычным открытием файла index.html. Часть кода сгенерирована программой, ниже представлен код, который я писал сам. Но при обычном открытии в браузере кнопки не работают, а в консоли вылезает ошибка “ createjs.min.js:13 Uncaught An error has occurred. This is most likely due to security restrictions on reading canvas pixel data with local or cross-domain images.”. Я не знаю, с чем это связано, это так со всеми проектами в данной программе. Однако, если загрузить файлы на какой-нибудь хостинг, все будет работать. Также, все отлично работает, если открыть файл в браузере “Microsoft Edge”, поэтому, рекомендую использовать его. Либо можно загрузить файлы на хостинг, тогда все будет работать в любом браузере. Или запустить в самой программе, так работает.

1. Код (JS + Canvas)

*//x = 0 : 120.25*

*//y = 0 : 513.75*

*//x = 1 : 242.25 (122/1)*

*//y = 1 : 391.75 (-122/1)*

**var** shift = **false**;

**var** control = **false**;

**function** transfX(x) {

**var** newX = 120.25 + (x \* 122);

**return** newX;

}

**function** transfY(y) {

**var** newY = 513.75 - (y \* 122);

**return** newY;

}

**var** xStar = 3;

**var** xSRes = 0;

**var** pointsX = [0 , 1.7, 3.4, 5.1, 6.8];

**var** pointsY = [0, 1.3038, 1.8439, 2.2583, 2.6077];

**var** NEWx = [];

**var** NEWy = [];

**var** approx = 10;

**var** speed = 10;

**var** begined = **false**;

**var** lifeTime = 0;

**function** f(a, b, c, d, x) {

**return** a + b \* x + c \* Math.pow(x, 2) + d \* Math.pow(x, 3);

}

**function** get\_a(f) {

**var** a = [0];

**for** (**var** i = 0 ; i < f.length - 1 ; ++i) {

        a.push(f[i]);

    }

**return** a;

}

**function** get\_b(f, h, c) {

**var** b = [0];

**var** n = f.length - 1;

**for** (**var** i = 1 ; i < n ; ++i) {

        b.push((f[i] - f[i - 1]) / h[i] - 1/3 \* h[i] \* (c[i + 1] + 2 \* c[i]));

    }

    b.push((f[n] - f[n - 1]) / h[n] - 2/3 \* h[n] \* c[n]);

**return** b;

}

**function** get\_c(x, f, h) {

**var** n = f.length;

**var** newN = n - 2;

**var** a = [0];

**for** (**var** i = 3 ; i < n ; ++i) {

        a.push(h[i - 1]);

    }

**var** b = [];

**for** (**var** i = 2 ; i < n ; ++i) {

        b.push(2 \* (h[i - 1] + h[i]));

    }

**var** c = [];

**for** (**var** i = 2 ; i < n - 1; ++i) {

        c.push(h[i]);

    }

    c.push(0);

**var** d = [];

**for** (**var** i = 2 ; i < n ; ++i) {

        d.push(3 \* ((f[i] - f[i - 1]) / h[i] - ((f[i - 1] - f[i - 2]) / h[i - 1])));

    }

**var** TDmatrix = [];

**for** (**var** i = 0 ; i < newN ; ++i) {

**var** thisVect = [];

**for** (**var** j = 0 ; j < i-1 ; ++j) {

            thisVect.push(0);

        }

**if** (i - 1 >= 0) {

            thisVect.push(a[i]);

        }

        thisVect.push(b[i]);

**if** (i + 1 < newN) {

            thisVect.push(c[i]);

        }

**for** (**var** j = i+2 ; j < newN ; ++j) {

            thisVect.push(0);

        }

        TDmatrix.push(thisVect);

    }

**var** x = progonka(TDmatrix, d);

**var** res = [0, 0];

**for** (**var** i = 0 ; i < x.length ; ++i) {

        res.push(x[i]);

    }

**return** res;

}

**function** get\_d(h, c) {

**var** d = [0];

**var** N = c.length - 1;

**for** (**var** i = 1 ; i < N ; ++i) {

        d.push((c[i + 1] - c[i]) / (3 \* h[i]));

    }

    d.push(-c[N] / (3 \* h[N]));

**return** d;

}

**function** find\_interval(points, x) {

**for** (**var** i = 0 ; i < points.length - 1 ; ++i) {

**if** (points[i] <= x && x <= points[i + 1]) {

**return** i;

        }

    }

}

spline\_interpolation(pointsX, pointsY, xStar);

**function** spline\_interpolation(points, values, x) {

**var** h = [0];

**for** (**var** i = 1 ; i < points.length ; ++i) {

        h.push(points[i] - points[i - 1]);

    }

**var** c = get\_c(points, values, h);

**var** a = get\_a(values);

**var** b = get\_b(values, h, c);

**var** d = get\_d(h, c);

**var** i = find\_interval(points, x);

    xSRes = f(a[i + 1], b[i + 1], c[i + 1], d[i + 1], x - points[i]);

**this**.exportRoot.f\_Res.text = "" + Math.round(xSRes\*1000000)/1000000;

    drawThis(points, values, a, b, c, d, approx);

}

**function** drawThis(points, values, a, b, c, d, aprox) {

    NEWx = [];

    NEWy = [];

**var** N = points.length - 1;

**for** (**var** i = 0 ; i < N ; ++i) {

**var** x1 = [];

**var** y1 = [];

**for** (**var** j = 0 ; j < aprox ; ++j) {

            x1.push(points[i] + (points[i+1] - points[i])\*(j/aprox));

            y1.push(f(a[i + 1], b[i + 1], c[i + 1], d[i + 1], x1[j] - points[i]));

        }

**for** (**var** j = 0 ; j < aprox ; ++j) {

            NEWx.push(x1[j]);

            NEWy.push(y1[j]);

        }

    }

    NEWx.push(points[N]);

    NEWy.push(values[N]);

}

**var** frame = 0;

**var** curPt = 0;

**this**.addEventListener("tick", Otrisovka.bind(**this**));

**function** Otrisovka() {

**if** (begined == **true**) {

        frame += speed/60;

**if** (frame > 1) {

**var** amount = Math.min(Math.floor(frame), NEWx.length - curPt - 1);

            frame -= amount;

**for** (**var** i = 0 ; i < amount ; ++i) {

**var** join = **new** lib.line();

                stage.addChild(join);

                join.x = transfX(NEWx[i+curPt]);

                join.y = transfY(NEWy[i+curPt]);

                join.endX = transfX(NEWx[i+curPt+1]);

                join.endY = transfY(NEWy[i+curPt+1]);

                join.yGO = -9;

                join.decr = 0;

                join.life = lifeTime;

                join.gotoAndStop(0);

                join.len = Math.sqrt(Math.pow((join.endY - join.y), 2) + Math.pow((join.endX - join.x), 2));

                join.scaleX = join.len;

                join.rotation = Math.atan2((join.endY - join.y), (join.endX - join.x)) \* 180 / Math.PI;

                join.visible = **true**;

                join.alpha = 1;

                join.addEventListener('tick', check\_life);

            }

            curPt += amount;

        }

    }

**this**.aproxx.text = "Approximation: " + Math.round(approx);

**this**.speedD.text = "Speed: " + Math.round(speed);

**if** (speed > 100000) {

        speed = 100000;

    }

**if** (approx > 200) {

        approx = 200;

    }

**if** (speed < 1) {

        speed = 1;

    }

**if** (approx < 1) {

        approx = 1;

    }

    speed = Math.round(speed);

    approx = Math.round(approx);

}

**function** check\_life(e) {

**var** object = e.currentTarget;

**var** gravity = 0.55;

**if** (object.life != lifeTime) {

        object.gotoAndStop(1);

        object.yGO += gravity;

        object.y += object.yGO;

        object.alpha \*= 0.95;

        object.scaleY += Math.pow(Math.max(0,(16 - object.scaleY)), 0.6);

        object.decr += 0.25;

        object.scaleY \*= (object.scaleY/(object.scaleY+object.decr));

    }

**if** (object.alpha <= 0.05) {

        object.alpha = 0;

        object.visible = **false**;

        object.removeEventListener('tick', check\_life);

        stage.removeChild(object);

    }

}

**this**.plusApr.addEventListener("click", addApr.bind(**this**));

**function** addApr() {

**if** (shift == **true** && control == **true**) {

        approx += 10;

        approx \*= 1.5

    }

**else** **if** (shift == **true**) {

        approx += 10;

    }

**else** **if** (control == **true**) {

        approx \*= 1.5;

    }

**else** {

        approx += 1;

    }

}

**this**.minusApr.addEventListener("click", remApr.bind(**this**));

**function** remApr() {

**if** (shift == **true** && control == **true**) {

**if** (approx > 10) {

            approx -= 10;

            approx /= 1.5

        }

    }

**else** **if** (shift == **true**) {

**if** (approx > 10) {

            approx -= 10;

        }

    }

**else** **if** (control == **true**) {

        approx /= 1.5;

    }

**else** {

        approx -= 1;

    }

}

window.addEventListener("keydown", doKeyDown.bind(**this**));

**function** doKeyDown(e) {

**if** (e.keyCode == 16) {

        shift = **true**;

    }

**if** (e.keyCode == 17) {

        control = **true**;

    }

}

window.addEventListener("keyup", doKeyUP.bind(**this**));

**function** doKeyUP(e) {

    shift = **false**;

    control = **false**;

}

**this**.plusSpd.addEventListener("click", addSpd.bind(**this**));

**function** addSpd() {

**if** (shift == **true** && control == **true**) {

        speed += 10;

        speed \*= 1.5

    }

**else** **if** (shift == **true**) {

        speed += 10;

    }

**else** **if** (control == **true**) {

        speed \*= 1.5;

    }

**else** {

        speed += 1;

    }

}

**this**.minusSpd.addEventListener("click", remSpd.bind(**this**));

**function** remSpd() {

**if** (shift == **true** && control == **true**) {

**if** (speed > 10) {

            speed -= 10;

            speed /= 1.5

        }

    }

**else** **if** (shift == **true**) {

**if** (speed > 10) {

            speed -= 10;

        }

    }

**else** **if** (control == **true**) {

        speed /= 1.5;

    }

**else** {

        speed -= 1;

    }

}

**this**.beginBt.addEventListener("click", startGame.bind(**this**));

**function** startGame() {

*//make pts*

**for** (**var** i = 0 ; i < pointsX.length ; ++i) {

**var** bigP = **new** lib.bigPoint();

        stage.addChild(bigP);

        bigP.x = transfX(pointsX[i]);

        bigP.y = transfY(pointsY[i]);

        bigP.visible = **true**;

        bigP.alpha = 1;

    }

    lifeTime += 1;

    frame = 0;

    curPt = 0;

    spline\_interpolation(pointsX, pointsY, xStar);

    begined = **true**;

}

**function** progonka(matrix, vectorB) {

**var** vectorX = [];

**var** N = vectorB.length;

**var** alphas = [];

**var** betas = [];

**for** (**var** i = 0 ; i < vectorB.length ; ++i) {

        alphas.push(0);

        betas.push(0);

    }

*//Прямой ход прогонки*

**for** (**var** i = 0; i < N; ++i) {

**var** A0, C0, B0, F0;

**if** (i - 1 < 0) {

            A0 = 0;

        }

**else** {

            A0 = matrix[i][i - 1];

        }

        C0 = -1 \* matrix[i][i];

**if** (i + 1 < N) {

            B0 = matrix[i][i + 1];

        }

**else** {

            B0 = 0;

        }

        F0 = vectorB[i];

**if** (i == 0) {

            alphas[i] = B0 / C0;

            betas[i] = -(F0 / C0);

        }

**else** **if** (i == N - 1) {

            alphas[i] = 0;

            betas[i] = (betas[i - 1] \* A0 - F0) / (C0 - alphas[i - 1] \* A0);

        }

**else** {

            alphas[i] = B0 / (C0 - alphas[i - 1] \* A0);

            betas[i] = (betas[i - 1] \* A0 - F0) / (C0 - alphas[i - 1] \* A0);

        }

    }

    vectorX[N - 1] = betas[N - 1];

**for** (**var** i = 2; i <= N; ++i) {

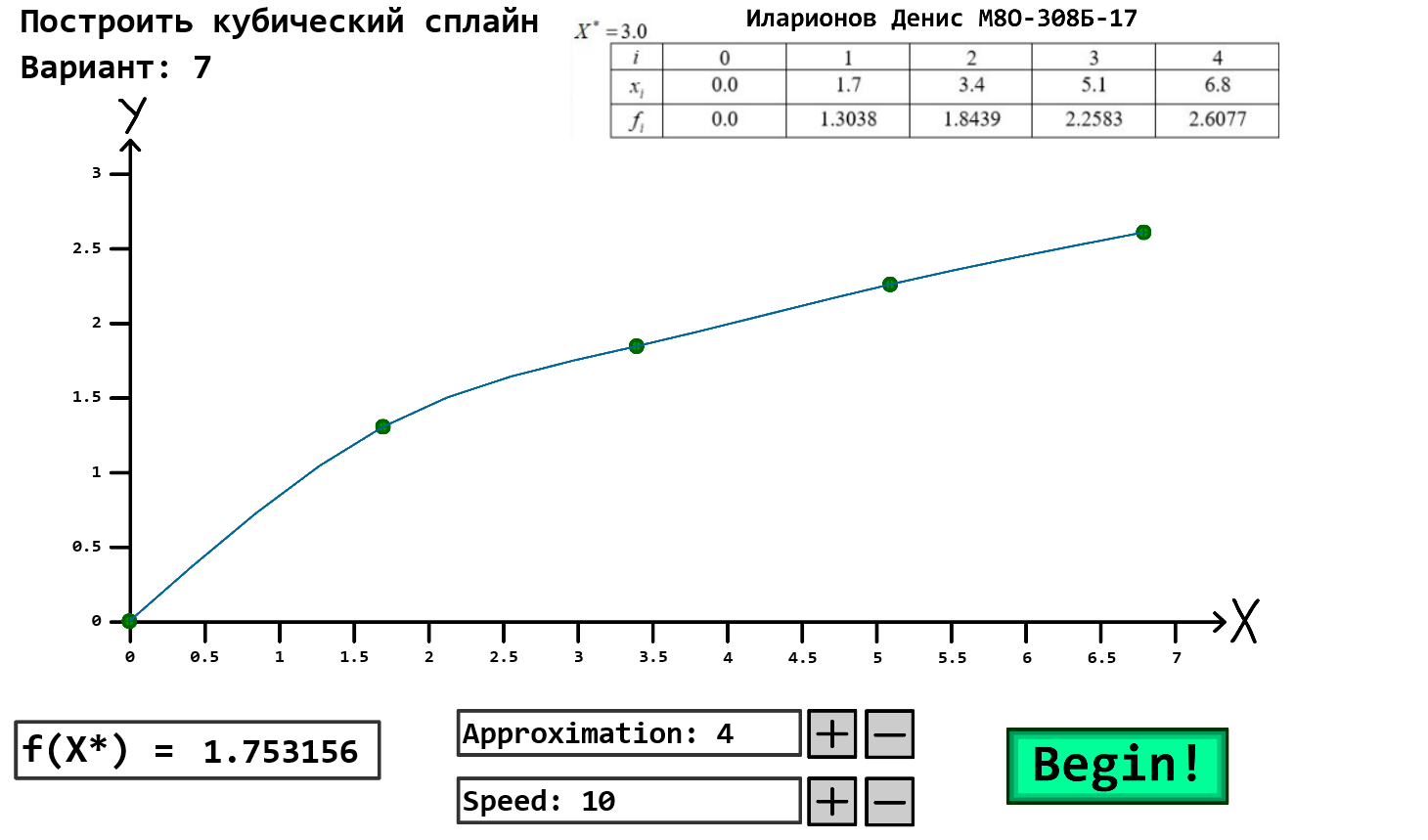
        vectorX[N - i] = alphas[N - i] \* vectorX[N - i + 1] + betas[N - i];

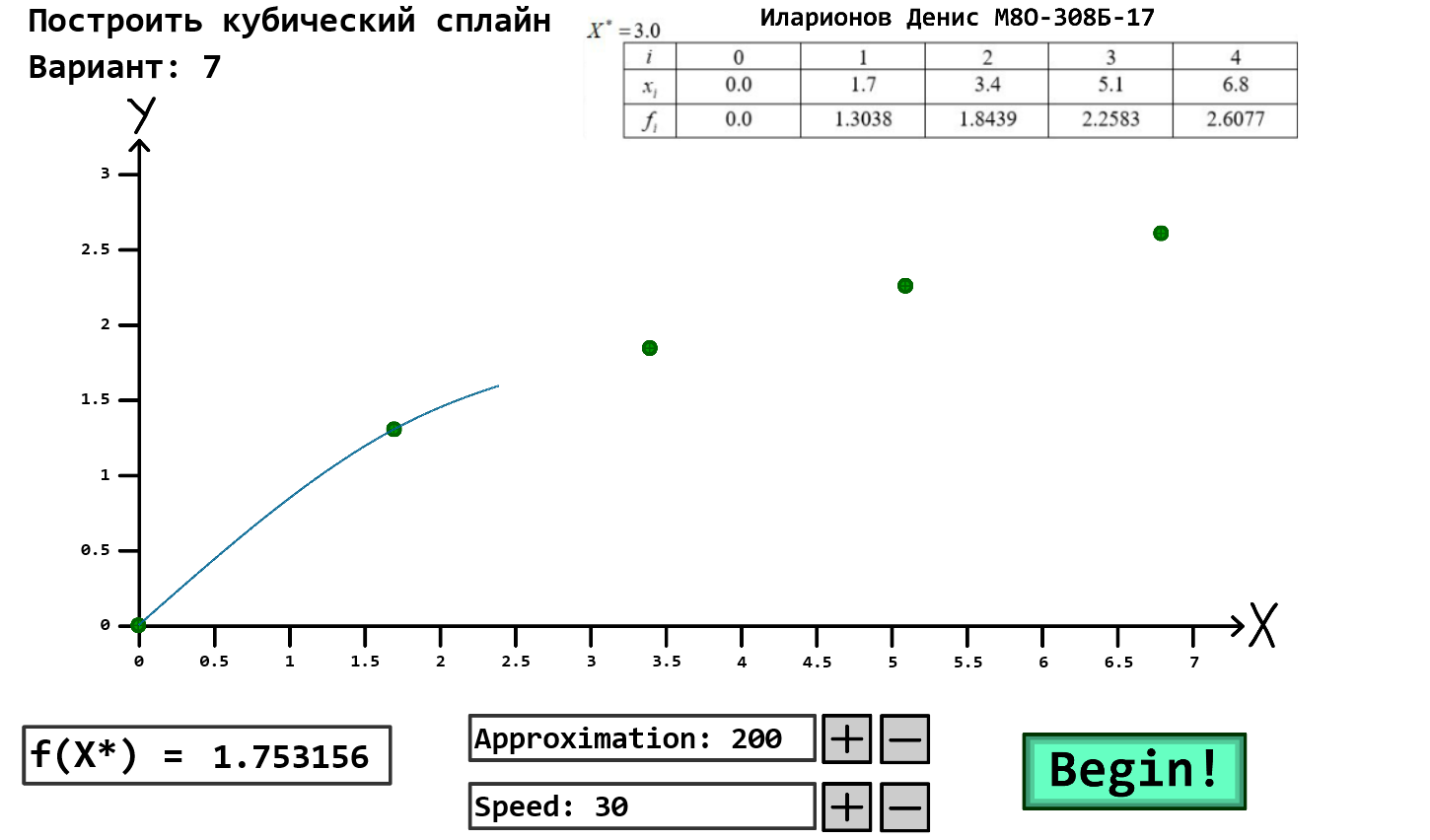
    }

**return** vectorX;

}

1. Тесты (Скриншоты моей программы)





1. Данная лабораторная работа выполнена: **4 мая 2020.**

**Лабораторная работа 3.3**

1. Тема ЛР:

Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены a) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

1. Вариант : **7**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
|  | 0.0 | 0.2 | 0.4 | 0.6 | 0.8 | 1.0 |
|  | 1.0 | 1.0032 | 1.0512 | 1.2592 | 1.8192 | 3.0 |

1. Алгоритм:

Построение приближающих многочленов – классная вещь. Она позволяет из таблицы с точками сделать похожую функцию. Иногда, даже может предсказать что-то. Для этого, нам нужно решить системы уравнений. Решал я их вручную, переписывать другие методы было лень что-то.. Чем больше степень приближающего многочлена, тем он ближе к табличной функции. Но вычисляется такой многочлен сложнее. Если для нахождения многочлена 1 степени нам надо решить систему из 2 уравнений до 2 степени, то для нахождения 2 степени – уже до 4 степени. И в конце я уже нашел погрешности.

1. Среда разработки:

Adobe Animate CC , язык - Javascript

1. Реализация

Аналогично предыдущей лабораторной. Пользователь задает аппроксимацию и скорость. Затем, уже идут вычисление и отрисовка. Отрисовка идет двух графиков одновременно, можно посмотреть, как они отличаются от заданных точек.

1. Код (JS + Canvas)

*//x = 0 : 119.2*

*//y = 0 : 562.65*

*//x = 1 : 363.4 (244.2/1)*

*//y = 1 : 440.5 (-122.1/1)*

**var** approx = 10;

**var** speed = 10;

**var** begined = **false**;

**var** shift = **false**;

**var** control = **false**;

**var** lifeTime = 0;

**function** transfX(x) {

**var** newX = 119.2 + (x \* 244.2);

**return** newX;

}

**function** transfY(y) {

**var** newY = 562.65 - (y \* 122.1);

**return** newY;

}

**var** pointsX = [0 , 0.2, 0.4, 0.6, 0.8, 1];

**var** pointsY = [1, 1.0032, 1.0512, 1.2592, 1.8192, 3];

**var** N = 5;

**var** coef1 = [0,0];

**var** coef2 = [0,0,0];

*//Нахождение многочл. 1 степени*

**var** a0first = N+1;

**var** a1first = 0;

**var** a0second = 0;

**var** a1second = 0;

**for** (**var** i = 0 ; i < N+1 ; ++i) {

    a1first += pointsX[i];

    a0second += pointsX[i];

    a1second += pointsX[i] \* pointsX[i];

}

**var** ansfirst = 0;

**var** anssecond = 0;

**for** (**var** i = 0; i < N+1 ; ++i) {

    ansfirst += pointsY[i];

    anssecond += pointsY[i] \* pointsX[i];

}

**function** ch(num) {

**if** (num >= 0) {

**return** " + " + num;

    }

**else** **return** " - " + Math.abs(num);

}

**var** a1third, ansthird;

a1third = a1second - (a0second / a0first) \* a1first;

ansthird = anssecond - (a0second / a0first) \* ansfirst;

coef1[1] = ansthird / a1third;

coef1[0] = (ansfirst - (a1first \* coef1[1]))/a0first;

**this**.ans1.text = Math.round(coef1[1]\*100000)/100000 + "x" + ch(Math.round(coef1[0]\*100000)/100000);

**var** sqError1 = 0;

**for** (**var** i = 0 ; i < N+1 ; ++i) {

    sqError1 += Math.pow((coef1[1]\*pointsX[i] + coef1[0]) - pointsY[i], 2);

}

**this**.er\_sq1.text = Math.round(sqError1\*100000)/100000;

*//Нахождение многочл. 2 степени*

**var** b0first = N + 1;

**var** b1first = 0, b2first = 0;

**var** b0second = 0, b1second = 0, b2second = 0;

**var** b0third = 0, b1third = 0, b2third = 0;

ansfirst = 0;

anssecond = 0;

ansthird = 0;

**for** (**var** i = 0 ; i < N + 1; ++i) {

    b1first += pointsX[i];

    b0second += pointsX[i];

    b2first += Math.pow(pointsX[i], 2);

    b1second += Math.pow(pointsX[i], 2);

    b0third += Math.pow(pointsX[i], 2);

    b2second += Math.pow(pointsX[i], 3);

    b1third += Math.pow(pointsX[i], 3);

    b2third += Math.pow(pointsX[i], 4);

    ansfirst += pointsY[i];

    anssecond += pointsY[i] \* pointsX[i];

    ansthird += pointsY[i] \* pointsX[i] \* pointsX[i];

}

**var** b1fourth, b2fourth, ansfourth = 0;

**var** b1fifth, b2fifth, ansfifth = 0;

b1fourth = b1second - b1first \* (b0second / b0first);

b2fourth = b2second - b2first \* (b0second / b0first);

ansfourth = anssecond - ansfirst \* (b0second / b0first);

b1fifth = b1third - b1first \* (b0third / b0first);

b2fifth = b2third - b2first \* (b0third / b0first);

ansfifth = ansthird - ansfirst \* (b0third / b0first);

b2fifth = b2fifth - b2fourth \* (b1fifth / b1fourth);

ansfifth = ansfifth - ansfourth \* (b1fifth / b1fourth);

b1fifth = 0;

coef2[2] = ansfifth/b2fifth;

coef2[1] = (ansfourth - (b2fourth \* coef2[2]))/b1fourth;

coef2[0] = (ansfirst - (b2first \* coef2[2]) - (b1first \* coef2[1]))/b0first;

**this**.ans2.text = Math.round(coef2[2]\*100000)/100000 + "x^2" + ch(Math.round(coef2[1]\*100000)/100000) + "x" +ch(Math.round(coef2[0]\*100000)/100000);

**var** sqError2 = 0;

**for** (**var** i = 0 ; i < N+1 ; ++i) {

    sqError2 += Math.pow((coef2[2]\*pointsX[i]\*pointsX[i] + coef2[1]\*pointsX[i] + coef2[0]) - pointsY[i], 2);

}

**this**.er\_sq2.text = Math.round(sqError2\*100000)/100000;

**var** newXPts = [];

**var** newYPts1 = [];

**var** newYPts2 = [];

**function** pointGen(approx) {

    newXPts = [];

    newYPts1 = [];

    newYPts2 = [];

**for** (**var** i = 0 ; i < N ; ++i) {

**for** (**var** j = 0 ; j < approx ; ++j) {

**var** newXPt = (pointsX[i] + (pointsX[i + 1] - pointsX[i])\*(j/approx));

            newXPts.push(newXPt);

            newYPts1.push((coef1[1]\*newXPt + coef1[0]));

            newYPts2.push((coef2[2]\*newXPt\*newXPt + coef2[1]\*newXPt + coef2[0]));

        }

    }

    newXPts.push(pointsX[N]);

    newYPts1.push((coef1[1]\*pointsX[N] + coef1[0]));

    newYPts2.push((coef2[2]\*pointsX[N]\*pointsX[N] + coef2[1]\*pointsX[N]+ coef2[0]));

}

**var** frame = 0;

**var** curPt = 0;

**this**.addEventListener("tick", Otrisovka.bind(**this**));

**function** Otrisovka() {

**if** (begined == **true**) {

        frame += speed/60;

**if** (frame > 1) {

**var** amount = Math.min(Math.floor(frame), newXPts.length - curPt - 1);

            frame -= amount;

**for** (**var** i = 0 ; i < amount ; ++i) {

**var** join = **new** lib.line();

**var** join2 = **new** lib.line();

                stage.addChild(join);

                stage.addChild(join2);

                join.x = transfX(newXPts[i+curPt]);

                join2.x = transfX(newXPts[i+curPt]);

                join.y = transfY(newYPts1[i+curPt]);

                join2.y = transfY(newYPts2[i+curPt]);

                join.endX = transfX(newXPts[i+curPt+1]);

                join2.endX = transfX(newXPts[i+curPt+1]);

                join.endY = transfY(newYPts1[i+curPt+1]);

                join2.endY = transfY(newYPts2[i+curPt+1]);

                join.yGO = -9;

                join.decr = 0;

                join.life = lifeTime;

                join.gotoAndStop(0);

                join2.yGO = -9;

                join2.decr = 0;

                join2.life = lifeTime;

                join2.gotoAndStop(1);

                join.len = Math.sqrt(Math.pow((join.endY - join.y), 2) + Math.pow((join.endX - join.x), 2));

                join2.len = Math.sqrt(Math.pow((join2.endY - join2.y), 2) + Math.pow((join2.endX - join2.x), 2));

                join.scaleX = join.len;

                join2.scaleX = join2.len;

                join.rotation = Math.atan2((join.endY - join.y), (join.endX - join.x)) \* 180 / Math.PI;

                join2.rotation = Math.atan2((join2.endY - join2.y), (join2.endX - join2.x)) \* 180 / Math.PI;

                join.visible = **true**;

                join.alpha = 1;

                join2.visible = **true**;

                join2.alpha = 1;

                join.addEventListener('tick', check\_life);

                join2.addEventListener('tick', check\_life);

            }

            curPt += amount;

        }

    }

**this**.aproxx.text = "Approximation: " + Math.round(approx);

**this**.speedD.text = "Speed: " + Math.round(speed);

**if** (speed > 100000) {

        speed = 100000;

    }

**if** (approx > 100) {

        approx = 100;

    }

**if** (speed < 1) {

        speed = 1;

    }

**if** (approx < 1) {

        approx = 1;

    }

    speed = Math.round(speed);

    approx = Math.round(approx);

}

**function** check\_life(e) {

**var** object = e.currentTarget;

**var** gravity = 0.55;

**if** (object.life != lifeTime) {

        object.gotoAndStop(2);

        object.yGO += gravity;

        object.y += object.yGO;

        object.alpha \*= 0.95;

        object.scaleY += Math.pow(Math.max(0,(16 - object.scaleY)), 0.6);

        object.decr += 0.25;

        object.scaleY \*= (object.scaleY/(object.scaleY+object.decr));

    }

**if** (object.alpha <= 0.05) {

        object.alpha = 0;

        object.visible = **false**;

        object.removeEventListener('tick', check\_life);

        stage.removeChild(object);

    }

}

**this**.plusApr.addEventListener("click", addApr.bind(**this**));

**function** addApr() {

**if** (shift == **true** && control == **true**) {

        approx += 10;

        approx \*= 1.5

    }

**else** **if** (shift == **true**) {

        approx += 10;

    }

**else** **if** (control == **true**) {

        approx \*= 1.5;

    }

**else** {

        approx += 1;

    }

}

**this**.minusApr.addEventListener("click", remApr.bind(**this**));

**function** remApr() {

**if** (shift == **true** && control == **true**) {

**if** (approx > 10) {

            approx -= 10;

            approx /= 1.5

        }

    }

**else** **if** (shift == **true**) {

**if** (approx > 10) {

            approx -= 10;

        }

    }

**else** **if** (control == **true**) {

        approx /= 1.5;

    }

**else** {

        approx -= 1;

    }

}

window.addEventListener("keydown", doKeyDown.bind(**this**));

**function** doKeyDown(e) {

**if** (e.keyCode == 16) {

        shift = **true**;

    }

**if** (e.keyCode == 17) {

        control = **true**;

    }

}

window.addEventListener("keyup", doKeyUP.bind(**this**));

**function** doKeyUP(e) {

    shift = **false**;

    control = **false**;

}

**this**.plusSpd.addEventListener("click", addSpd.bind(**this**));

**function** addSpd() {

**if** (shift == **true** && control == **true**) {

        speed += 10;

        speed \*= 1.5

    }

**else** **if** (shift == **true**) {

        speed += 10;

    }

**else** **if** (control == **true**) {

        speed \*= 1.5;

    }

**else** {

        speed += 1;

    }

}

**this**.minusSpd.addEventListener("click", remSpd.bind(**this**));

**function** remSpd() {

**if** (shift == **true** && control == **true**) {

**if** (speed > 10) {

            speed -= 10;

            speed /= 1.5

        }

    }

**else** **if** (shift == **true**) {

**if** (speed > 10) {

            speed -= 10;

        }

    }

**else** **if** (control == **true**) {

        speed /= 1.5;

    }

**else** {

        speed -= 1;

    }

}

**this**.beginBt.addEventListener("click", startGame.bind(**this**));

**function** startGame() {

*//make pts*

**for** (**var** i = 0 ; i < pointsX.length ; ++i) {

**var** bigP = **new** lib.bigPoint();

        stage.addChild(bigP);

        bigP.x = transfX(pointsX[i]);

        bigP.y = transfY(pointsY[i]);

        bigP.visible = **true**;

        bigP.alpha = 1;

    }

    lifeTime += 1;

    frame = 0;

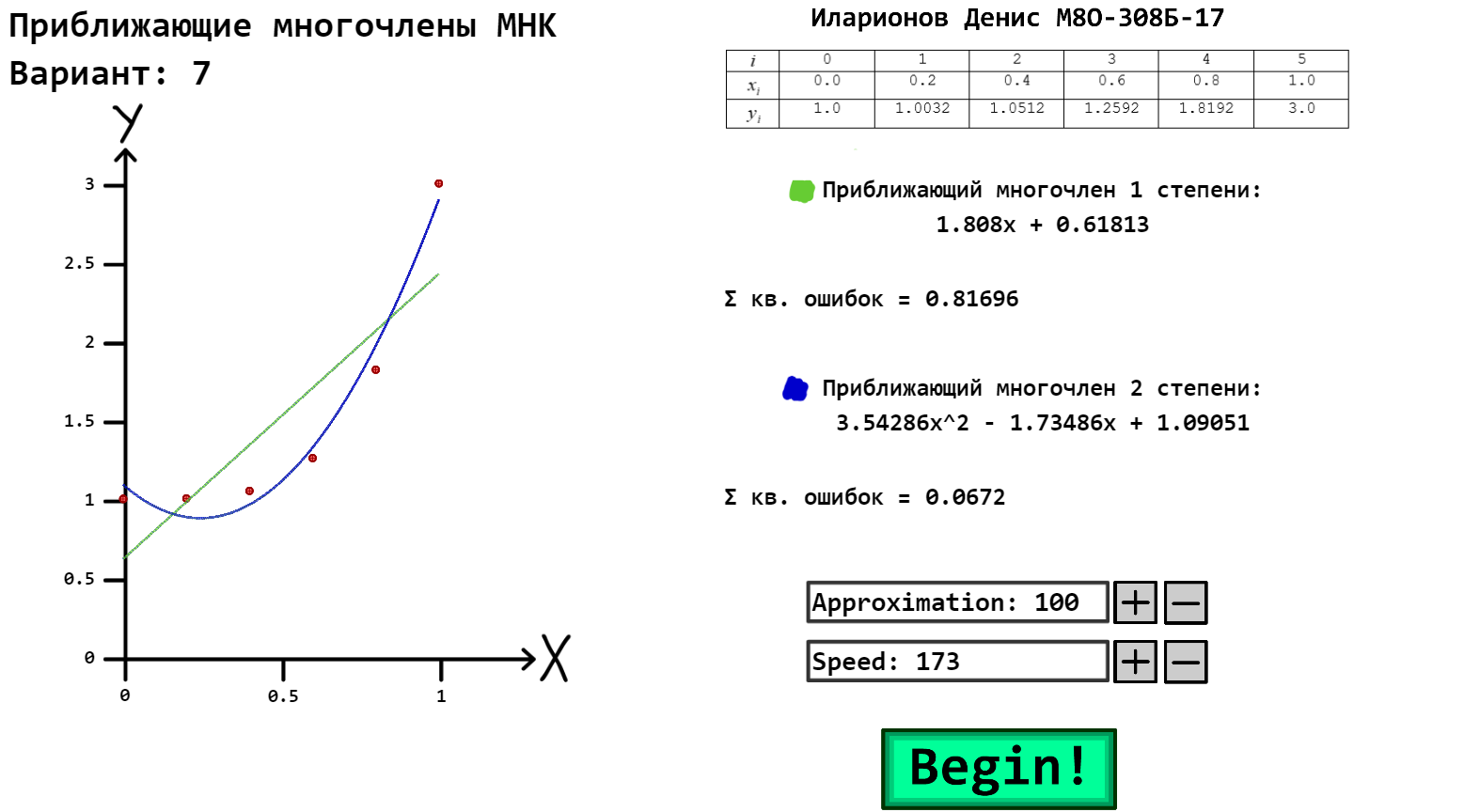
    curPt = 0;

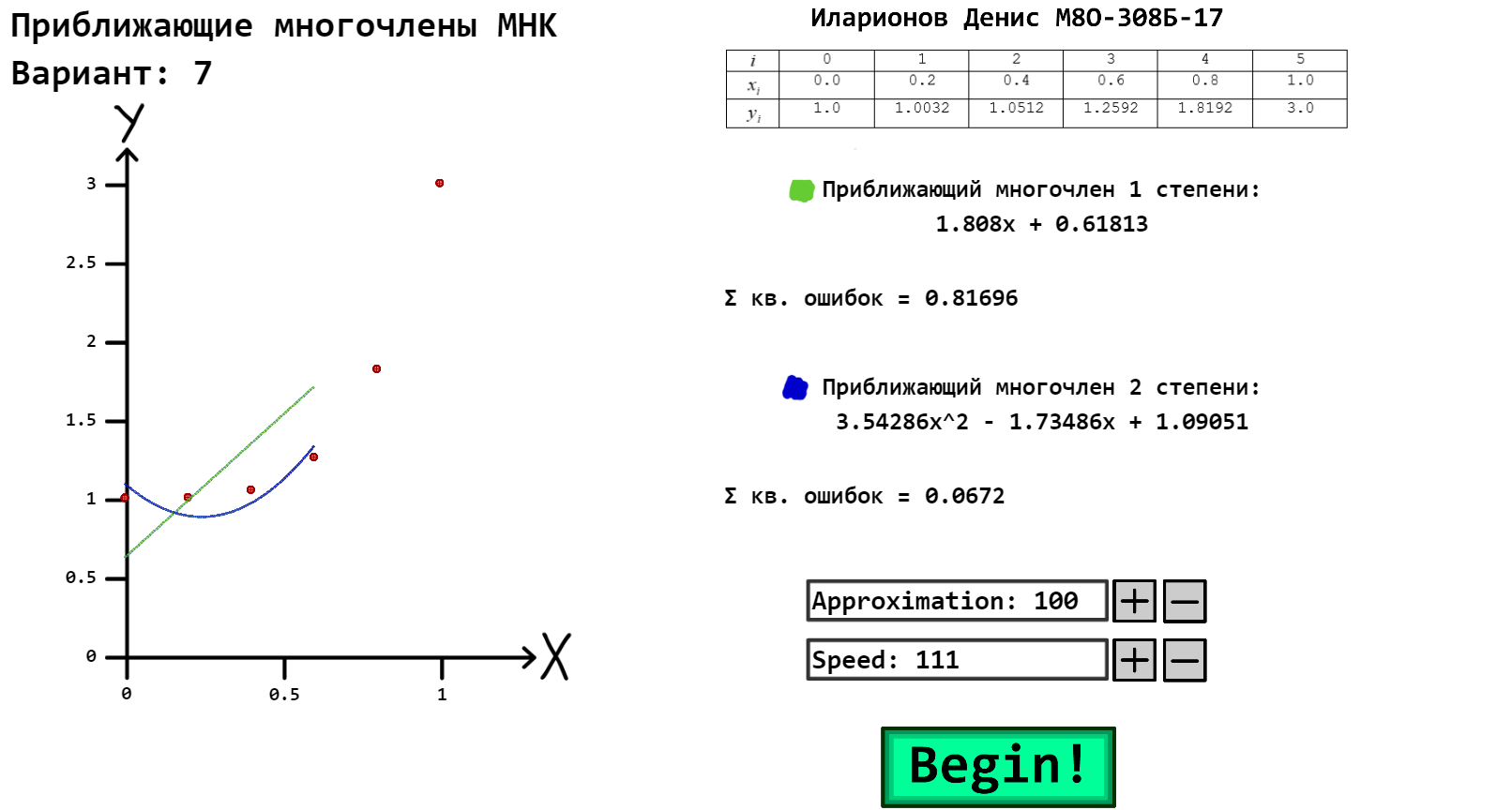
    pointGen(approx);

    begined = **true**;

}

1. Тесты (Скриншоты моей программы)





1. Данная лабораторная работа выполнена: **4 мая 2020.**

**Лабораторная работа 3.4**

1. Тема ЛР:

Вычислить первую и вторую производную от таблично заданной функции в точке .

1. Вариант : **7**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 |
|  | -0.2 | 0.0 | 0.2 | 0.4 | 0.6 |
|  | 1.7722 | 1.5708 | 1.3694 | 1.1593 | 0.9273 |

 0.2

1. Алгоритм:

Довольно простая лаба. И нахождение производных тут не является сложным. Мы просто ищем точку в таблице. Находим левую и правую производные. (производные при вычитании из точки левой, и из правой точки точку). И по формулам вычисляем первую и вторую производные.

1. Среда разработки:

Visual Studio 2019 , язык – C++

1. Реализация

Так как тут не нужна визуализация, вернулся к консольному приложению и написал все на C++

1. Код (С++)

#include<iostream>

#include<cmath>

#include<algorithm>

#include<vector>

using namespace std;

int main() {

setlocale(LC\_ALL, "Russian");

int N;

cout << "Введите количество точек: ";

cin >> N;

vector<double>Xs(N, 0);

vector<double>Ys(N, 0);

double XStar;

cout << "\nВведите точки X:\n";

for (int i = 0; i < N; ++i) {

cin >> Xs[i];

}

cout << "\nВведите точки Y:\n";

for (int i = 0; i < N; ++i) {

cin >> Ys[i];

}

cout << "\nВведите точку X\*:\n";

cin >> XStar;

double firstDer;

double secondDer;

//Вычисление производных

int num = -1;

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

if (XStar == Xs[i]) {

num = i;

}

}

if (num == -1) {

cout << "\nУказаной точки нет в таблице, либо вычисление невозможно.\n";

return -1;

}

double left1Der, right1Der;

left1Der = (Ys[num] - Ys[num - 1]) / (Xs[num] - Xs[num - 1]);

right1Der = (Ys[num + 1] - Ys[num]) / (Xs[num + 1] - Xs[num]);

firstDer = left1Der + (right1Der - left1Der) / (Xs[num + 1] - Xs[num - 1]) \* (2 \* XStar - Xs[num - 1] - Xs[num]);

secondDer = 2 \* (right1Der - left1Der) / (Xs[num + 1] - Xs[num - 1]);

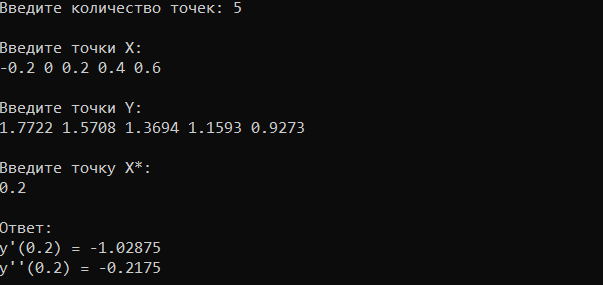
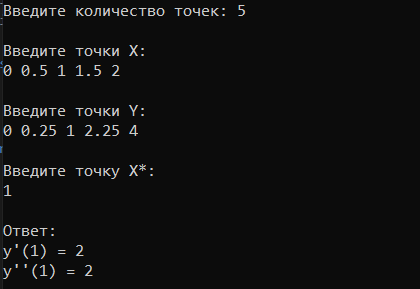
cout << "\nОтвет:\n";

cout << "y'(" << XStar << ") = " << firstDer << "\n";

cout << "y''(" << XStar << ") = " << secondDer << "\n";

}

1. Тесты (Мой вариант и функция x^2)

1. Данная лабораторная работа выполнена: **5 мая 2020.**

**Лабораторная работа 3.5**

1. Тема ЛР:

Вычислить определенный интеграл , методами прямоугольников, трапеций, Симпсона с шагами . Оценить погрешность вычислений, используя Ме­тод Рунге-Ромберга:

1. Вариант : **7**

, ;

1. Алгоритм:

Нужно решить интеграл тремя способами. Тоже довольно просто. Значение – сумма элементов, для каждого способа разные формулы. Прибавляем к значению итоговому после каждого шага. Например, для метода прямоугольников + y((2 \* x + h1) / 2) \* h1; Где h1 – шаг, а x – точка. y(x) – значение подынтегральной функции в точке x. Для методов трапеций и Симпсона формулы немного другие, а также отличаются первых/последний и/или четных/нечетных шагов. Погрешность так же находится по формуле, нам нужно два решения с шагами, где один больше другого в несколько раз. Будет как отношение разницы между значениями с разным шагом к степени разницы шагов – 1.

1. Среда разработки:

Visual Studio 2019 , язык – C++

1. Реализация

Написал программу консольным приложением. Для разных шагов решил ее. Можно было сократить и решить одним циклом, а не два раза писать для каждого шага, но мне что-то было лень.

1. Код (С++)

#include<iostream>

#include<cmath>

#include<algorithm>

#include<vector>

using namespace std;

double y(double x) {

return 1 / (3 \* pow(x, 2) + 4 \* x + 2);

}

int main() {

setlocale(LC\_ALL, "Russian");

int N;

cout << "Вариант 7, Формула:\n\n";

cout << "y = 1 / (3x^2 + 4x + 2)\n";

double X0, Xk, h1, h2;

cout << "\nВведите точку X0: ";

cin >> X0;

cout << "Введите точку Xk: ";

cin >> Xk;

cout << "Введите шаг h1: ";

cin >> h1;

cout << "Введите шаг h2: ";

cin >> h2;

double rect1 = 0;

double rect2 = 0;

double trap1 = 0;

double trap2 = 0;

double simpson1 = 0;

double simpson2 = 0;

int s\_iter = 0;

//решаем для h1

double x = X0;

trap1 += y(x) / 2 \* h1;

simpson1 += y(x) \* h1 / 3;

while (x + h1 <= Xk) {

rect1 += y((2 \* x + h1) / 2) \* h1;

if ((x + h1 != Xk)) {

trap1 += y(x + h1) \* h1;

}

else trap1 += y(x + h1) / 2 \* h1;

if ((x + h1 != Xk)) {

if (s\_iter % 2 == 0) {

simpson1 += 4 \* y(x + h1) \* h1 / 3;

s\_iter++;

}

else {

simpson1 += 2 \* y(x + h1) \* h1 / 3;

s\_iter++;

}

}

else {

simpson1 += y(x + h1) \* h1 / 3;

}

x += h1;

}

//решаем для h2

x = X0;

trap2 += y(x) / 2 \* h2;

simpson2 += y(x) \* h2 / 3;

s\_iter = 0;

while (x + h2 <= Xk) {

rect2 += y((2 \* x + h2) / 2) \* h2;

if ((x + h2 != Xk)) {

trap2 += y(x + h2) \* h2;

}

else trap2 += y(x + h2) / 2 \* h2;

if ((x + h2 != Xk)) {

if (s\_iter % 2 == 0) {

simpson2 += 4 \* y(x + h2) \* h2 / 3;

s\_iter++;

}

else {

simpson2 += 2 \* y(x + h2) \* h2 / 3;

s\_iter++;

}

}

else {

simpson2 += y(x + h2) \* h2 / 3;

}

x += h2;

}

//Погрешности и метод Рутте

double k = h1 / h2;

double rungeValue = 0;

double exact = 1.857418687;

double pogr\_rect1 = abs(rect1 - rect2) / (pow(k, 2) - 1);

double pogr\_rect2 = abs(rect2 - exact) / (pow(k, 2) - 1);

double pogr\_trap1 = abs(trap1 - trap2) / (pow(k, 2) - 1);

double pogr\_trap2 = abs(trap2 - exact) / (pow(k, 2) - 1);

double pogr\_simp1 = abs(simpson1 - simpson2) / (pow(k, 4) - 1);

double pogr\_simp2 = abs(simpson2 - exact) / (pow(k, 4) - 1);

cout << "\n";

cout << "Значение интеграла методом прямоугольников с шагом h1 = " << h1 << " : " << rect1 << "\n";

cout << "Значение интеграла методом прямоугольников с шагом h2 = " << h2 << " : " << rect2 << "\n\n";

cout << "Значение интеграла методом трапеций с шагом h1 = " << h1 << " : " << trap1 << "\n";

cout << "Значение интеграла методом трапеций с шагом h2 = " << h2 << " : " << trap2 << "\n\n";

cout << "Значение интеграла методом Симпсона с шагом h1 = " << h1 << " : " << simpson1 << "\n";

cout << "Значение интеграла методом Симпсона с шагом h2 = " << h2 << " : " << simpson2 << "\n\n";

cout << "Погрешность зн. интеграла методом прямоугольников с шагом h1 = " << h1 << " : " << pogr\_rect1 << "\n";

cout << "Погрешность зн. интеграла методом прямоугольников с шагом h2 = " << h2 << " : " << pogr\_rect2 << "\n\n";

cout << "Погрешность зн. интеграла методом трапеций с шагом h1 = " << h1 << " : " << pogr\_trap1 << "\n";

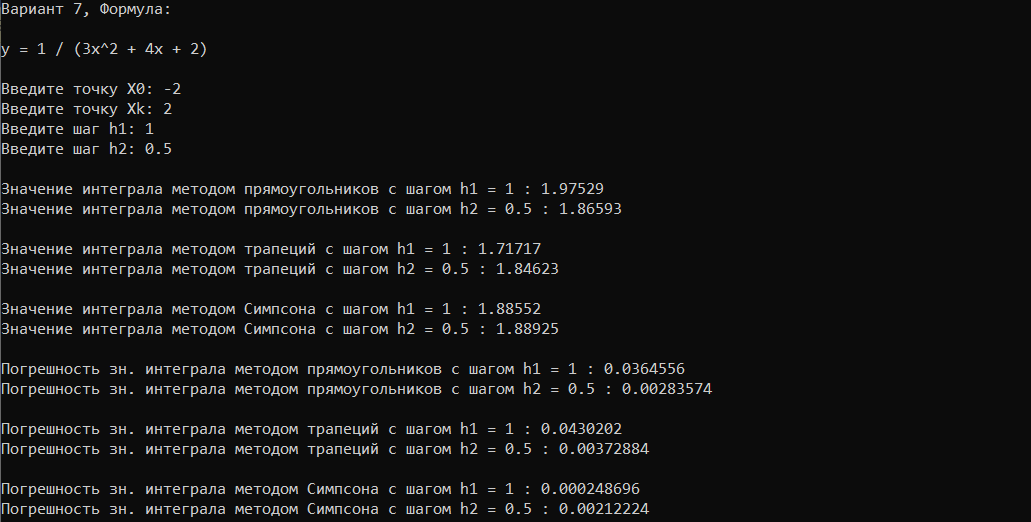
cout << "Погрешность зн. интеграла методом трапеций с шагом h2 = " << h2 << " : " << pogr\_trap2 << "\n\n";

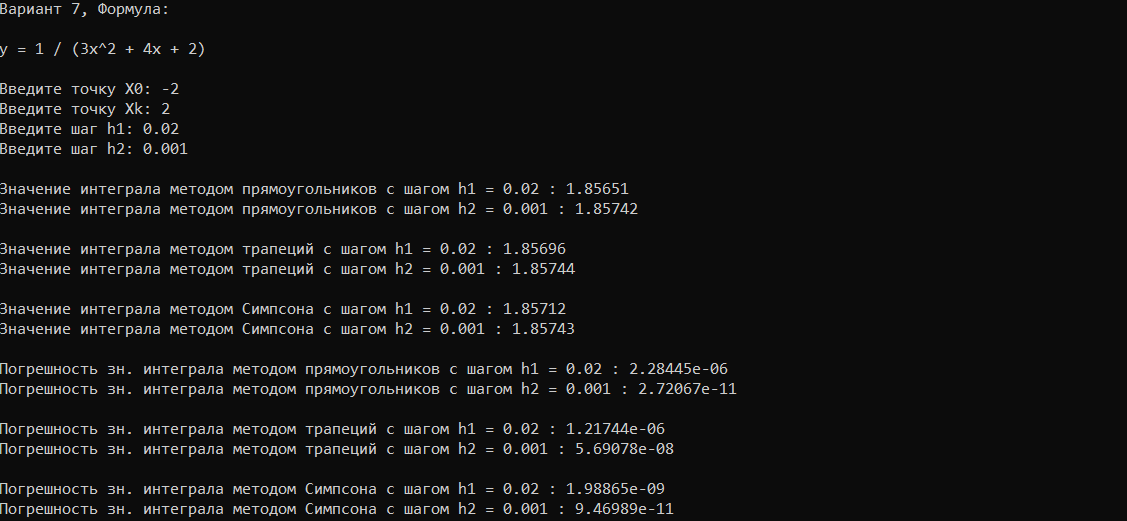
cout << "Погрешность зн. интеграла методом Симпсона с шагом h1 = " << h1 << " : " << pogr\_simp1 << "\n";

cout << "Погрешность зн. интеграла методом Симпсона с шагом h2 = " << h2 << " : " << pogr\_simp2 << "\n\n";

}

1. Тесты (Мой вариант с разными шагами)





1. Данная лабораторная работа выполнена: **5 мая 2020**

**Лабораторная работа 4.1**

1. Тема ЛР:

Реализовать методы Эйлера, Рунге-Кутты и Адамса 4-го порядка в виде программ, задавая в качестве входных данных шаг сетки . С использованием разработанного программного обеспечения решить задачу Коши для ОДУ 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

1. Вариант : **7**

|  |  |
| --- | --- |
| , |  |

1. Алгоритм:

Решение ДУ 3 способами. Задается само ДУ, условия, отрезок. Я это вводил все в программу. Также дается функция, которая является решением, для нахождения потом погрешностей. Метод Эйлера работает итерационно. Создается новая переменная z, к которой прибавляется шаг, умноженный на значение при подставлении точек. Z – производная от y первого порядка. Она добавляется к точке y, умноженная на шаг. В итоге заполняются точки. Метод Рунге-Кутты на основе неких коэффициентов, находятся дельты y и z (производной) по этим коэффициентам (K1-4, L1-4), добавляются в массивы. Но выводится и z, так как первые 4 точки для метода Адамса нельзя найти, в итоге мы их берем из метода Рунге-Кутты. Он находит значения просто подставляя определенные коэффициенты и берет 4 предыдущие точки. Для x используется само ДУ, для y – функция g, которая просто возвращает производную. Затем идет подсчет погрешностей, аналогично тому, как это было в предыдущих лабах.

1. Среда разработки:

Adobe Animate CC , язык - Javascript

1. Реализация

Можно было решить это просто выводя таблицу в консольном приложении. Но мне это показалось скучно, решил сделать визуализацию всего этого. К тому же, смотреть такую таблицу будет неудобно. Поэтому, я прибегнул к Adobe Animate и запрограммировал все это. Пользователь вводит шаг и скорость. Затем, поочереди строятся графики каждым методом. Можно включить/отключить видимость некоторых графиков и основной функции, которая уже изначально построена. Для нахождения погрешностей, система перерешивается с шагом в 2 раза меньше, но визуально изменений нет, она потом снова перерешивается с нужным шагом и уже находятся погрешности. При нажатии на кнопку Begin идет визуализация. При наведении на точку, отображается ее значение в каждом из методов, точное значение, а также погрешности.

1. Код (JS + Canvas)

*//x = 0 : 128,2*

*//y = 0 : 570,55*

*//x = 1 : 471 (342.8/1)*

*//y = 1 : 509,25 (-61.3)*

**function** transfX(x) {

**var** newX = 128.2 + (x \* 342.8);

**return** newX;

}

**function** transfY(y) {

**var** newY = 570.55 - (y \* 61.3);

**return** newY;

}

**var** control = **false**;

**var** shift = **false**;

*//function (1 + x) \* e^(x^2)*

*//x c [0, 1]*

**var** begined = **false**;

**var** lifeTime = 1;

**var** frame = 0;

**var** closestDist = 9999999;

**var** closestI = 0;

**function** Koshi(x, y, y\_der) {

**return** 4\*x\*y\_der - (4 \* x\*x - 2)\*y

}

**function** g(x, y, z) {

**return** z;

}

**function** func(x) {

**return** (1 + x) \* Math.exp(Math.pow(x, 2));

}

**var** y0 = 1;

**var** y\_der = 1;

**var** pointsX = [];

**var** pointsY = [];

**var** mousePosX;

**var** mousePosY;

**var** h = 0.1;

**var** speed = 10; *//2 per second*

**var** EulerX = [];

**var** EulerY = [];

**var** RungeX = [];

**var** RungeY = [];

**var** RungeZ = [];

**var** AdamsX = [];

**var** AdamsY = [];

**var** EulerErrs1 = [];

**var** EulerErrs2 = [];

**var** RungeErrs1 = [];

**var** RungeErrs2 = [];

**var** AdamsErrs1 = [];

**var** AdamsErrs2 = [];

**var** iter = 0;

**var** graph = 1;

**var** showExact = **true**;

**var** showEuler = **true**;

**var** showRunge = **true**;

**var** showAdams = **true**;

**function** Euler(a, b, h, y0, y\_der) {

**var** N = 0;

**var** x = [];

**var** y = [y0];

**for** (**var** i = a ; i <= b ; i += h) {

        i = Math.round(i\*10000)/10000;

        x.push(i);

        ++N;

    }

    z = y\_der;

**for** (**var** i = 0 ; i < N ; ++i) {

        z += h \* Koshi(x[i], y[i], z);

**var** y\_i = y[i] + h \* g(x[i], y[i], z);

        y.push(y\_i);

    }

    EulerX = x;

    EulerY = y;

}

**function** RungeKutta(a, b, h, y0, y\_der) {

**var** N = 0;

**var** x = [];

**var** y = [y0];

**for** (**var** i = a ; i <= b ; i += h) {

        i = Math.round(i\*10000)/10000;

        x.push(i);

        ++N;

    }

**var** z = [y\_der];

**for** (**var** i = 0 ; i < N ; ++i) {

**var** K1 = h \* g(x[i], y[i], z[i]);

**var** L1 = h \* Koshi(x[i], y[i], z[i]);

**var** K2 = h \* g(x[i] + 0.5 \* h, y[i] + 0.5 \* K1, z[i] + 0.5 \* L1);

**var** L2 = h \* Koshi(x[i] + 0.5 \* h, y[i] + 0.5 \* K1, z[i] + 0.5 \* L1);

**var** K3 = h \* g(x[i] + 0.5 \* h, y[i] + 0.5 \* K2, z[i] + 0.5 \* L2);

**var** L3 = h \* Koshi(x[i] + 0.5 \* h, y[i] + 0.5 \* K2, z[i] + 0.5 \* L2);

**var** K4 = h \* g(x[i] + h, y[i] + K3, z[i] + L3);

**var** L4 = h \* Koshi(x[i] + h, y[i] + K3, z[i] + L3);

**var** delta\_y = (K1 + 2 \* K2 + 2 \* K3 + K4) / 6;

**var** delta\_z = (L1 + 2 \* L2 + 2 \* L3 + L4) / 6;

        y.push(y[i] + delta\_y);

        z.push(z[i] + delta\_z);

    }

    RungeX = x;

    RungeY = y;

    RungeZ = z;

}

**function** Adams(a, b, h, y0, y\_der) {

**var** N = 0;

**var** x = [];

**var** y = [];

**var** z = [];

**for** (**var** i = a ; i <= b ; i += h) {

        i = Math.round(i\*10000)/10000;

        x.push(i);

        ++N;

    }

**for** (**var** i = 0 ; i < Math.min(N, 4) ; ++i) {

        y.push(RungeY[i]);

        z.push(RungeZ[i]);

    }

**for** (**var** i = 3 ; i < N ; ++i) {

**var** z\_i = z[i] + h \* (55 \* Koshi(x[i], y[i], z[i]) -

                    59 \* Koshi(x[i - 1], y[i - 1], z[i - 1]) +

                    37 \* Koshi(x[i - 2], y[i - 2], z[i - 2]) -

                    9 \* Koshi(x[i - 3], y[i - 3], z[i - 3])) / 24;

        z.push(z\_i);

**var** y\_i = y[i] + h \* (55 \* g(x[i], y[i], z[i]) -

                    59 \* g(x[i - 1], y[i - 1], z[i - 1]) +

                    37 \* g(x[i - 2], y[i - 2], z[i - 2]) -

                    9 \* g(x[i - 3], y[i - 3], z[i - 3])) / 24;

        y.push(y\_i);

    }

    AdamsX = x;

    AdamsY = y;

}

**function** exErrors(a, b, h) {

**var** N = Math.ceil((b - a) / h);

    EulerErrs1 = [];

    RungeErrs1 = [];

    AdamsErrs1 = [];

**for** (**var** i = 0 ; i < N+1 ; ++i) {

        EulerErrs1.push(Math.abs(EulerY[i] - func(EulerX[i])));

        RungeErrs1.push(Math.abs(RungeY[i] - func(RungeX[i])));

        AdamsErrs1.push(Math.abs(AdamsY[i] - func(AdamsX[i])));

    }

}

**function** RRombergErrors(a, b, h) {

**var** N = Math.ceil((b - a) / h);

    EulerErrs2 = [];

    RungeErrs2 = [];

    AdamsErrs2 = [];

    Euler(0, 1, h/2, y0, y\_der);

    RungeKutta(0, 1, h/2, y0, y\_der);

    Adams(0, 1, h/2, y0, y\_der);

**var** newEulerX1 = [];

**var** newEulerY1 = [];

**var** newRungeX1 = [];

**var** newRungeY1 = [];

**var** newAdamsX1 = [];

**var** newAdamsY1 = [];

**for** (**var** i = 0 ; i < EulerX.length ; ++i) {

        newEulerX1.push(EulerX[i]);

        newEulerY1.push(EulerY[i]);

        newRungeX1.push(RungeX[i]);

        newRungeY1.push(RungeY[i]);

        newAdamsX1.push(AdamsX[i]);

        newAdamsY1.push(AdamsY[i]);

    }

    Euler(0, 1, h, y0, y\_der);

    RungeKutta(0, 1, h, y0, y\_der);

    Adams(0, 1, h, y0, y\_der);

**var** newEulerX = EulerX;

**var** newEulerY = [];

**var** newRungeX = RungeX;

**var** newRungeY = [];

**var** newAdamsX = AdamsX;

**var** newAdamsY = [];

**for** (**var** i = 0, j = 0; i < N+1 && j < 2\*N+1 ; ++j) {

**if** (Math.round(EulerX[i]\*100000) == Math.round(newEulerX1[j]\*100000)) {

            newEulerY.push(newEulerY1[j]);

            newRungeY.push(newRungeY1[j]);

            newAdamsY.push(newAdamsY1[j]);

            ++i;

        }

    }

*//k = 2*

**for** (**var** i = 0 ; i < N+1 ; ++i) {

        EulerErrs2.push(Math.abs(EulerY[i] - newEulerY[i]));

        RungeErrs2.push(Math.abs(RungeY[i] - newRungeY[i]) / 15);

        AdamsErrs2.push(Math.abs(AdamsY[i] - newAdamsY[i]));

    }

}

**function** draw(X, Y, type, iter) {

**var** i = iter;

**if** (i < X.length && i > 0) {

**var** point = **new** lib.bigPoint();

            stage.addChild(point);

            point.x = transfX(X[i]);

            point.y = transfY(Y[i]);

            point.type = type;

**var** join = **new** lib.line();

            stage.addChild(join);

            join.x = transfX(X[i-1]);

            join.y = transfY(Y[i-1]);

            join.endX = transfX(X[i]);

            join.endY = transfY(Y[i]);

            join.yGO = -9;

            join.decr = 0;

            join.life = lifeTime;

            point.yGO = -9;

            point.decr = 0;

            point.life = lifeTime;

            point.count = i;

            join.type = type;

**if** (type == "Euler") {

                join.gotoAndStop(1);

                point.gotoAndStop(0);

            }

**else** **if** (type == "Runge") {

                join.gotoAndStop(2);

                point.gotoAndStop(1);

            }

**else** {

                join.gotoAndStop(3);

                point.gotoAndStop(2);

            }

            join.len = Math.sqrt(Math.pow((join.endY - join.y), 2) + Math.pow((join.endX - join.x), 2));

            join.scaleX = join.len;

            join.rotation = Math.atan2((join.endY - join.y), (join.endX - join.x)) \* 180 / Math.PI;

            join.visible = **true**;

            join.alpha = 1;

            point.visible = **true**;

            point.alpha = 1;

            join.addEventListener('tick', liveLine);

            point.addEventListener('tick', livePoint);

        }

**else** **if** (i == 0) {

**var** point = **new** lib.bigPoint();

            stage.addChild(point);

            point.x = transfX(X[0]);

            point.y = transfY(Y[0]);

            point.type = type;

            point.yGO = -9;

            point.decr = 0;

            point.life = lifeTime;

            point.count = i;

**if** (type == "Euler") {

                point.gotoAndStop(0);

            }

**else** **if** (type == "Runge") {

                point.gotoAndStop(1);

            }

**else** {

                point.gotoAndStop(2);

            }

            point.visible = **true**;

            point.alpha = 1;

            point.addEventListener('tick', livePoint);

        }

}

**for** (**var** i = 0 ; i <= 300 ; ++i) {

**var** x = i/300;

**var** y = func(x);

    pointsX.push(x);

    pointsY.push(y);

}

*//Прорисовка ориг. функции*

**for** (**var** i = 0 ; i < 300 ; ++i) {

**var** join = **new** lib.line();

    stage.addChild(join);

    join.x = transfX(pointsX[i]);

    join.y = transfY(pointsY[i]);

    join.endX = transfX(pointsX[i+1]);

    join.endY = transfY(pointsY[i+1]);

    join.gotoAndStop(0);

    join.len = Math.sqrt(Math.pow((join.endY - join.y), 2) + Math.pow((join.endX - join.x), 2));

    join.scaleX = join.len;

    join.scaleY = 1.5;

    join.rotation = Math.atan2((join.endY - join.y), (join.endX - join.x)) \* 180 / Math.PI;

    join.visible = **true**;

    join.alpha = 1;

    join.addEventListener('tick', configureF);

}

**function** configureF(e) {

**var** object = e.currentTarget;

**if** (showExact) {

        object.visible = **true**;

    }

**else** {

        object.visible = **false**;

    }

}

**function** liveLine(e) {

**var** object = e.currentTarget;

**var** gravity = 0.55;

**if** (object.life != lifeTime) {

        object.gotoAndStop(4);

        object.yGO += gravity;

        object.y += object.yGO;

        object.alpha \*= 0.95;

        object.scaleY += Math.pow(Math.max(0,(16 - object.scaleY)), 0.6);

        object.decr += 0.25;

        object.scaleY \*= (object.scaleY/(object.scaleY+object.decr));

    }

**if** (object.alpha <= 0.05) {

        object.alpha = 0;

        object.visible = **false**;

        object.removeEventListener('tick', liveLine);

        stage.removeChild(object);

    }

**if** (object.type == "Euler") {

**if** (showEuler) {

            object.visible = **true**;

        }

**else** {

            object.visible = **false**;

        }

    }

**else** **if** (object.type == "Runge") {

**if** (showRunge) {

            object.visible = **true**;

        }

**else** {

            object.visible = **false**;

        }

    }

**else** {

**if** (showAdams) {

            object.visible = **true**;

        }

**else** {

            object.visible = **false**;

        }

    }

}

**function** livePoint(e) {

**var** object = e.currentTarget;

**var** gravity = 0.55;

**if** (object.life != lifeTime) {

        object.gotoAndStop(3);

        object.yGO += gravity;

        object.y -= object.yGO;

        object.alpha \*= 0.95;

    }

**if** (object.alpha <= 0.05) {

        object.alpha = 0;

        object.visible = **false**;

        object.removeEventListener('tick', livePoint);

        stage.removeChild(object);

    }

**if** (object.type == "Euler") {

**if** (showEuler) {

            object.visible = **true**;

        }

**else** {

            object.visible = **false**;

        }

    }

**else** **if** (object.type == "Runge") {

**if** (showRunge) {

            object.visible = **true**;

        }

**else** {

            object.visible = **false**;

        }

    }

**else** {

**if** (showAdams) {

            object.visible = **true**;

        }

**else** {

            object.visible = **false**;

        }

**if** (mousePosX >= 128 && mousePosX <= 492 && graph > 3) {

**if** (object.count == closestI) {

**this**.exportRoot.miniUI.x = object.x + 200;

**this**.exportRoot.miniUI.y = mousePosY - 80;

            }

        }

    }

}

**this**.plusH.addEventListener("click", addH.bind(**this**));

**function** addH() {

    h += 0.01;

}

**this**.minusH.addEventListener("click", remH.bind(**this**));

**function** remH() {

**if** (h > 0.01) {

        h -= 0.01;

    }

}

window.addEventListener("keydown", doKeyDown.bind(**this**));

**function** doKeyDown(e) {

**if** (e.keyCode == 16) {

        shift = **true**;

    }

**if** (e.keyCode == 17) {

        control = **true**;

    }

}

window.addEventListener("keyup", doKeyUP.bind(**this**));

**function** doKeyUP(e) {

    shift = **false**;

    control = **false**;

}

**this**.plusSpd.addEventListener("click", addSpd.bind(**this**));

**function** addSpd() {

**if** (shift == **true** && control == **true**) {

        speed += 10;

        speed \*= 1.5

    }

**else** **if** (shift == **true**) {

        speed += 10;

    }

**else** **if** (control == **true**) {

        speed \*= 1.5;

    }

**else** {

        speed += 1;

    }

}

**this**.minusSpd.addEventListener("click", remSpd.bind(**this**));

**function** remSpd() {

**if** (shift == **true** && control == **true**) {

**if** (speed > 10) {

            speed -= 10;

            speed /= 1.5

        }

    }

**else** **if** (shift == **true**) {

**if** (speed > 10) {

            speed -= 10;

        }

    }

**else** **if** (control == **true**) {

        speed /= 1.5;

    }

**else** {

        speed -= 1;

    }

}

**this**.addEventListener("tick", main.bind(**this**));

**function** main() {

**if** (h > 1) {

        h = 1;

    }

**if** (h < 0.01) {

        h = 0.01;

    }

**if** (speed > 100000) {

        speed = 100000;

    }

**if** (speed < 1) {

        speed = 1;

    }

    h = Math.round(h\*100)/100;

    speed = Math.round(speed);

**this**.h\_text.text = "h = " + h;

**this**.spd\_text.text = "Speed: " + speed;

**if** (begined) {

        frame += speed/300;

**var** iters = 0;

**if** (frame > 1 && graph < 4) {

            iters = Math.floor(frame);

            frame -= iters;

**for** (**var** i = 0 ; i < iters ; ++i) {

**if** (graph == 1) draw(EulerX, EulerY, "Euler", iter);

**if** (graph == 2) draw(RungeX, RungeY, "Runge", iter);

**if** (graph == 3) draw(AdamsX, AdamsY, "Adams", iter);

                ++iter;

            }

**if** (iter >= EulerX.length) {

                iter = 0;

                graph++;

            }

        }

    }

**if** (showExact) {

**this**.showBox1.gotoAndStop(1);

    }

**else** {

**this**.showBox1.gotoAndStop(0);

    }

**if** (showEuler) {

**this**.showBox2.gotoAndStop(1);

    }

**else** {

**this**.showBox2.gotoAndStop(0);

    }

**if** (showRunge) {

**this**.showBox3.gotoAndStop(1);

    }

**else** {

**this**.showBox3.gotoAndStop(0);

    }

**if** (showAdams) {

**this**.showBox4.gotoAndStop(1);

    }

**else** {

**this**.showBox4.gotoAndStop(0);

    }

    mousePosX = stage.mouseX / canvas.width \* 1080;

    mousePosY = stage.mouseY / canvas.height \* 720;

    closestDist = 9999999;

**if** (graph > 3) {

**for** (**var** i = 0 ; i < EulerX.length ; ++i) {

**if** (Math.abs(mousePosX - transfX(EulerX[i])) < closestDist) {

                closestDist = Math.abs(mousePosX - transfX(EulerX[i]));

                closestI = i;

            }

        }

**if** (mousePosX >= 128 && mousePosX <= 492) {

**this**.miniUI.visible = **true**;

            stage.addChild(**this**.miniUI);

**this**.miniUI.pointNum.text = "Точка #" + (closestI+1);

**this**.miniUI.pX.text = "x = " + rou(EulerX[closestI]);

**this**.miniUI.fX.text = "f(x) = " + rou(func(EulerX[closestI]));

**this**.miniUI.Eul.text = "Euler(x) = " + rou(EulerY[closestI]);

**this**.miniUI.Run.text = "Runge-K(x) = " + rou(RungeY[closestI]);

**this**.miniUI.Ada.text = "Adams(x) = " + rou(AdamsY[closestI]);

**this**.miniUI.Euld1.text = "ΔEuler = " + rou(EulerErrs2[closestI]);

**this**.miniUI.Rund1.text = "ΔRunge-K = " + rou(RungeErrs2[closestI]);

**this**.miniUI.Adad1.text = "ΔAdams = " + rou(AdamsErrs2[closestI]);

**this**.miniUI.Euld2.text = "ΔEuler = " + rou(EulerErrs1[closestI]);

**this**.miniUI.Rund2.text = "ΔRunge-K = " + rou(RungeErrs1[closestI]);

**this**.miniUI.Adad2.text = "ΔAdams = " + rou(AdamsErrs1[closestI]);

        }

**else** {

**this**.miniUI.visible = **false**;

        }

    }

}

**function** rou(num) {

**return** Math.round(num\*100000)/100000;

}

**this**.showBox1.addEventListener("click", setShow1.bind(**this**));

**function** setShow1() {

**if** (showExact) {

        showExact = **false**;

    }

**else** {

        showExact = **true**;

    }

}

**this**.showBox2.addEventListener("click", setShow2.bind(**this**));

**function** setShow2() {

**if** (showEuler) {

        showEuler = **false**;

    }

**else** {

        showEuler = **true**;

    }

}

**this**.showBox3.addEventListener("click", setShow3.bind(**this**));

**function** setShow3() {

**if** (showRunge) {

        showRunge = **false**;

    }

**else** {

        showRunge = **true**;

    }

}

**this**.showBox4.addEventListener("click", setShow4.bind(**this**));

**function** setShow4() {

**if** (showAdams) {

        showAdams = **false**;

    }

**else** {

        showAdams = **true**;

    }

}

**this**.beginBt.addEventListener("click", startGame.bind(**this**));

**function** startGame() {

    lifeTime += 1;

    frame = 0;

    iter = 0;

    graph = 1;

    begined = **true**;

    Euler(0, 1, h, y0, y\_der);

    RungeKutta(0, 1, h, y0, y\_der);

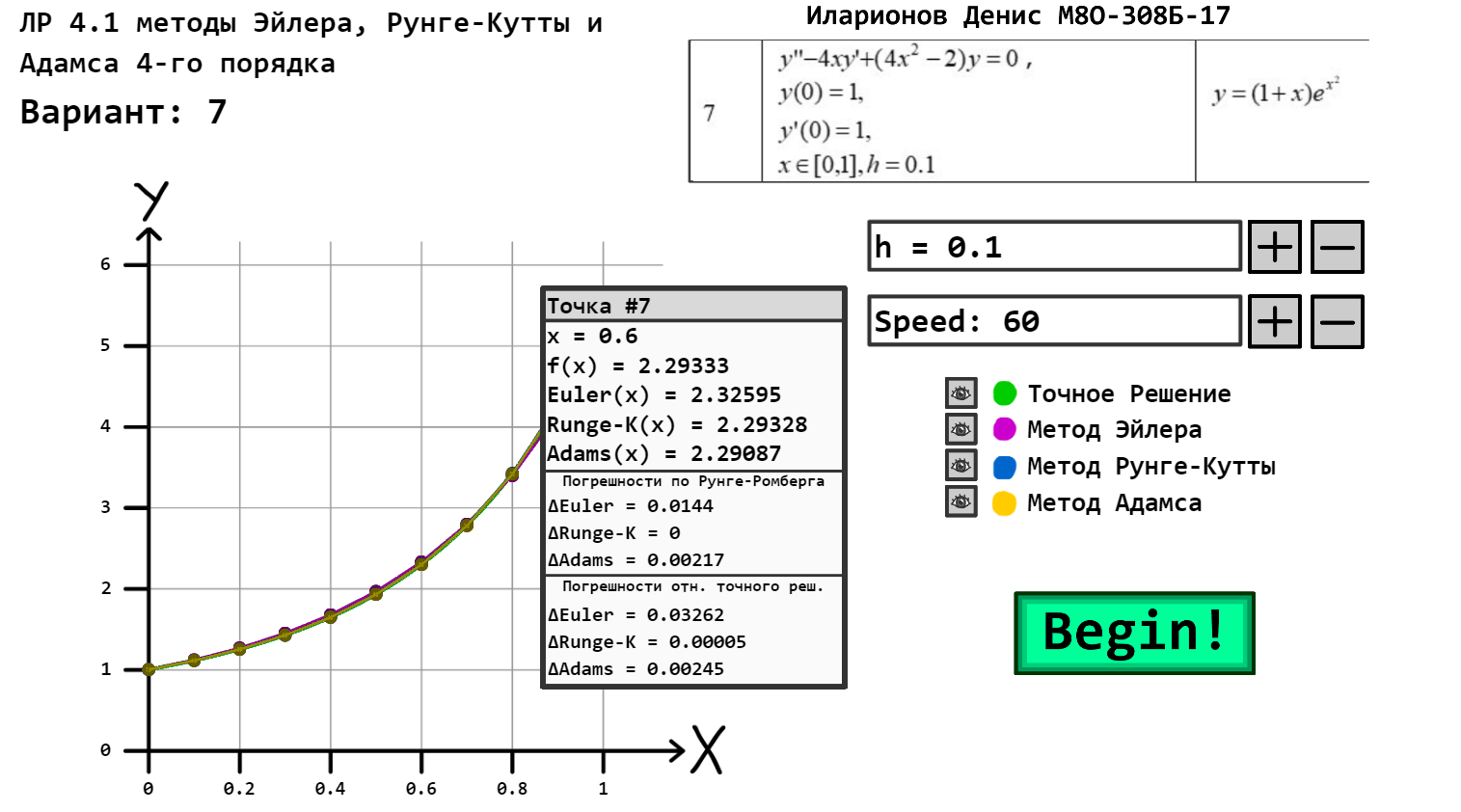
    Adams(0, 1, h, y0, y\_der);

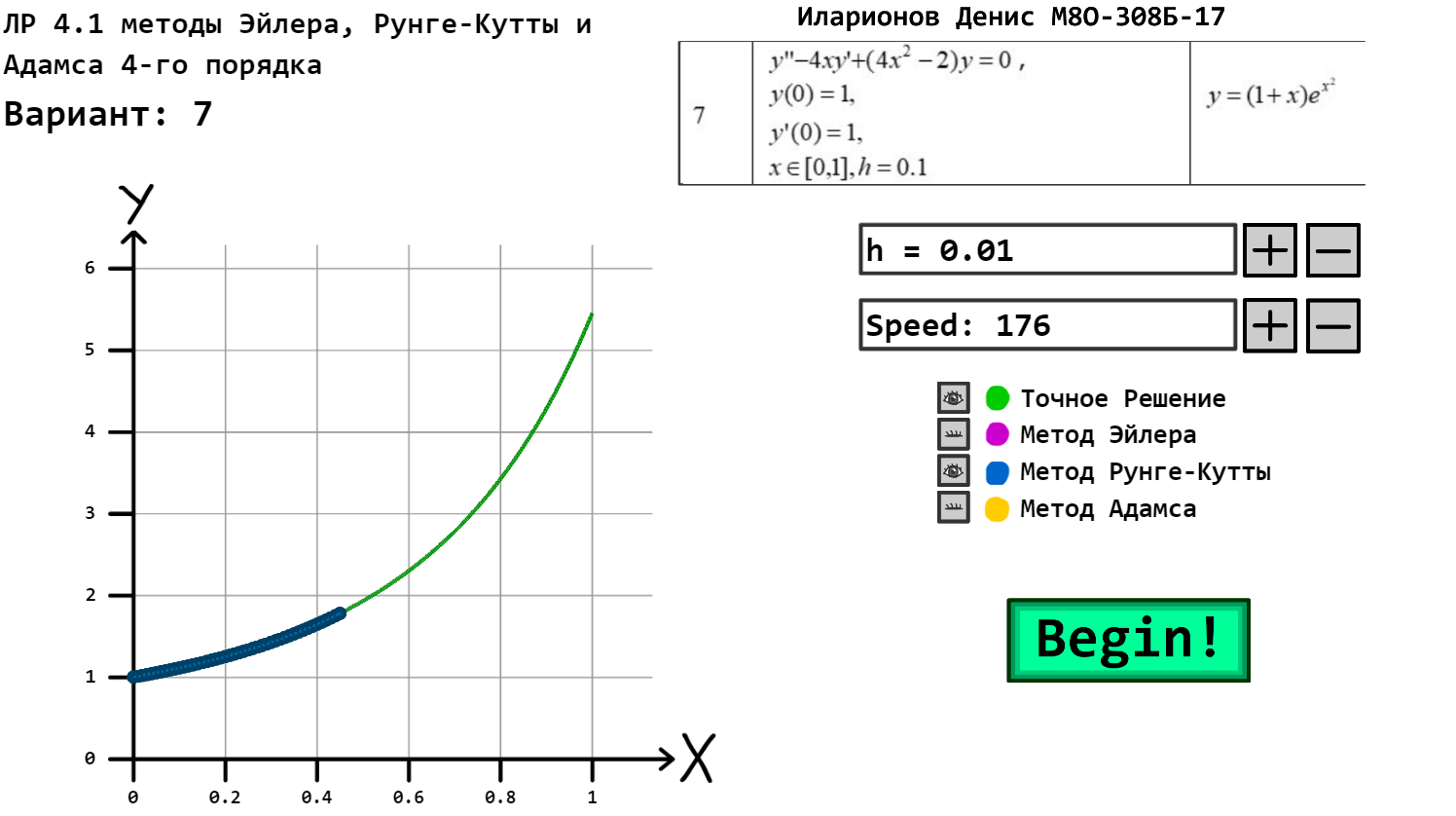
    RRombergErrors(0, 1, h);

    exErrors(0, 1, h);

}

1. Тесты (Скриншоты моей программы)





1. Данная лабораторная работа выполнена: **7 мая 2020.**

**Лабораторная работа 4.2**

1. Тема ЛР:

Реализовать метод стрельбы и конечно-разностный метод решения краевой задачи для ОДУ в виде программ. С использованием разработанного программного обеспечения решить краевую задачу для обыкновенного дифференциального уравнения 2-го порядка на указанном отрезке. Оценить погрешность численного решения с использованием метода Рунге – Ромберга и путем сравнения с точным решением.

1. Вариант : **7**

|  |  |
| --- | --- |
| (2x+1) y″+4xy′-4y=0,  y′ (0)= –1,  y′(1)+2y(1)=3 |  |

1. Алгоритм:

Метод стрельбы и конечно-разностный метод используются для решения ДУ с несколькими более сложными условиями. Метод стрельбы решает много раз метод Рунге-Кутта с условием, где y0 равно N, и каждый раз это N подбирается. Через функцию Phi, которая зависит от коэф. альфа, бета, гамма, дельта. Эти коэффициенты – коэф. в условии y’(0), y’(1), y(0) и y(1). Этот метод решается, пока погрешность не станет слишком маленькой. Конечно-разностный метод немного проще. Заполняются массивы A,B,C,D, чем-то очень похож на метод из ЛР 3.2 , когда мы находили коэф. C для построения кубических сплайнов. Отличается тем, что тут функции другие. А в целом, мы так же выполняем пошагово, и так же получается трехдиагональная матрица, которую мы уже решаем методом прогонки. Это и есть ответ, а затем мы находим погрешности.

1. Среда разработки:

Adobe Animate CC , язык - Javascript

1. Реализация

Все аналогично реализации прошлой ЛР. Также можно отключать видимость методов. Долго не получалось, ибо ответ был странным, а потом заметил, что я просто знак не тот поставил. Но тут еще границы y сами находятся, в зависимости от максимальных и минимальных точек.

1. Код (JS + Canvas)

*//x = 0 : 128,2*

*//y = 0 : 570,55*

*//x = 1 : 471 (342.8/1)*

*//y = 1 : 204.2 (-366.35)*

**var** a = 0;

**var** b = 1;

**var** h = 0.1;

**var** speed = 10; *//2 per second*

**var** alpha = 0;

**var** beta = 1;

**var** gamma = 1;

**var** delta = 2;

**var** y0 = -1;

**var** y1 = 3;

**var** eps = 0.0001;

**var** frame = 0;

**var** iter = 0;

**var** graph = 0;

**var** begined = **false**;

**var** lifeTime = 0;

**var** mousePosX = 0;

**var** mousePosY = 0;

**var** closestDist = 0;

**var** maxY = 1.25;

**var** minY = 0.75;

**var** RungeX = [];

**var** RungeY = [];

**var** shootX = [];

**var** shootY = [];

**var** KRX = [];

**var** KRY = [];

**var** shootErrs1 = [];

**var** shootErrs2 = [];

**var** KRErrs1 = [];

**var** KRErrs2 = [];

**var** lifetime = 0;

**function** transfX(x) {

**var** newX = 128.2 + ((x-a) \* 342.8/(b - a));

**return** newX;

}

**function** transfY(y) {

**var** newY = 570.55 - ((y-minY) \* 366.35/(maxY - minY));

**return** newY;

}

**var** control = **false**;

**var** shift = **false**;

**function** Koshi(x, y, y\_der) {

**return** (4\*y - 4\*x\*y\_der) / (2\*x + 1);

}

**function** func(x) {

**return** x + Math.exp(-2 \* x);

}

**function** g(x, y, z) {

**return** z;

}

**function** p(x) {

**return** 4\*x / (2\*x + 1);

}

**function** q(x) {

**return** -4 / (2\*x + 1);

}

**function** f(x) {

**return** 0;

}

**function** first\_der(x, y, x0) {

**var** i = 0;

    while (i < x.length - 1 && x[i + 1] < x0) {

        ++i;

    }

**return** (y[x.length - 1] - y[x.length - 2]) / (x[x.length - 1] - x[x.length - 2]);

}

**function** getN(prevN, N, prevAns, ans, b, delta, gamma, y1) {

**var** x = prevAns[0];

**var** y = prevAns[1];

**var** y\_der = first\_der(x, y, b);

**var** phiNprev = delta \* y[y.length - 1] + gamma \* y\_der - y1;

**var** xn = ans[0];

**var** yn = ans[1];

**var** y\_der2 = first\_der(xn, yn, b);

**var** phiN = delta \* yn[yn.length - 1] + gamma \* y\_der2 - y1;

**return** N - ((N - prevN) / (phiN - phiNprev) \* phiN);

}

**function** checkEnd(x, y, b, delta, gamma, y1, eps) {

**var** y\_der = first\_der(x, y, b);

**return** Math.abs(delta \* y[y.length - 1] + gamma \* y\_der - y1) > eps;

}

**function** shootingMethod(a, b, alpha, beta, delta, gamma, y0, y1, h, eps) {

**var** prevN = 1;

**var** N = 0.8;

**var** y\_der = (y0 - alpha \* prevN) / beta;

    RungeX = [];

    RungeY = [];

    RungeKutta(a, b, h, prevN, y\_der);

**var** ans\_prev = [[] , []];

**for** (**var** i = 0 ; i < RungeX.length ; ++i) {

        ans\_prev[0].push(RungeX[i]);

        ans\_prev[1].push(RungeY[i]);

    }

    y\_der = (y0 - alpha \* N) / beta;

    RungeX = [];

    RungeY = [];

    RungeKutta(a, b, h, N, y\_der);

**var** ans = [[] , []];

**for** (**var** i = 0 ; i < RungeX.length ; ++i) {

        ans[0].push(RungeX[i]);

        ans[1].push(RungeY[i]);

    }

*//console.log(ans\_prev);*

*//console.log(ans);*

    while (checkEnd(ans[0], ans[1], b, delta, gamma, y1, eps)) {

**var** oldN = N;

        N =  getN(prevN, N, ans\_prev, ans, b, delta, gamma, y1);

        prevN = oldN;

        ans\_prev = [[] , []];

**for** (**var** i = 0 ; i < RungeX.length ; ++i) {

            ans\_prev[0].push(ans[0][i]);

            ans\_prev[1].push(ans[1][i]);

        }

        y\_der = (y0 - alpha \* N) / beta;

        RungeX = [];

        RungeY = [];

        RungeKutta(a, b, h, N, y\_der);

        ans = [[] , []];

**for** (**var** i = 0 ; i < RungeX.length ; ++i) {

            ans[0].push(RungeX[i]);

            ans[1].push(RungeY[i]);

        }

    }

    shootX = ans[0];

    shootY = ans[1];

}

**function** KRMethod(a, b, alpha, beta, delta, gamma, y0, y1, h) {

**var** N = 0;

**var** x = [];

**for** (**var** i = a ; i <= b ; i += h) {

        i = Math.round(i\*10000)/10000

        x.push(i);

        ++N;

    }

**var** A = [0];

**for** (**var** i = 0 ; i < N - 2 ; ++i) {

        A.push(1 - p(x[i]) \* h / 2);

    }

    A.push(-gamma);

**var** B = [alpha \* h - beta];

**for** (**var** i = 0 ; i < N - 2 ; ++i) {

        B.push(q(x[i]) \* Math.pow(h, 2) - 2);

    }

    B.push(delta \* h + gamma);

**var** C = [beta];

**for** (**var** i = 0 ; i < N - 2 ; ++i) {

        C.push(1 + p(x[i]) \* h / 2);

    }

    C.push(0);

**var** D = [y0 \* h];

**for** (**var** i = 0 ; i < N - 2 ; ++i) {

        D.push(f(x[i]) \* Math.pow(h, 2));

    }

    D.push(y1 \* h);

**var** TDmatrix = [];

**for** (**var** i = 0 ; i < N + 1 ; ++i) {

**var** thisVect = [];

**for** (**var** j = 0 ; j < i-1 ; ++j) {

            thisVect.push(0);

        }

**if** (i - 1 >= 0) {

            thisVect.push(A[i]);

        }

        thisVect.push(B[i]);

**if** (i + 1 < N + 1 ) {

            thisVect.push(C[i]);

        }

**for** (**var** j = i+2 ; j < N + 1  ; ++j) {

            thisVect.push(0);

        }

        TDmatrix.push(thisVect);

    }

**var** y = progonka(TDmatrix, D);

    KRX = x;

    KRY = y;

}

**function** exact\_errors() {

    shootErrs1 = [];

    KRErrs1 = [];

**for** (**var** i = 0; i < shootX.length ; ++i) {

        shootErrs1.push(Math.abs(shootY[i] - func(shootX[i])));

        KRErrs1.push(Math.abs(KRY[i] - func(KRX[i])));

    }

}

**function** RRombergErrors(a, b, h) {

**var** N = Math.ceil((b - a) / h);

    shootErrs2 = [];

    KRErrs2 = [];

    shootingMethod(a, b, alpha, beta, delta, gamma, y0, y1, h/2, eps);

    KRMethod(a, b, alpha, beta, delta, gamma, y0, y1, h/2);

**var** newShootX1 = [];

**var** newShootY1 = [];

**var** newKRX1 = [];

**var** newKRY1 = [];

**for** (**var** i = 0 ; i < shootX.length ; ++i) {

        newShootX1.push(shootX[i]);

        newShootY1.push(shootY[i]);

        newKRX1.push(KRX[i]);

        newKRY1.push(KRY[i]);

    }

    shootingMethod(a, b, alpha, beta, delta, gamma, y0, y1, h, eps);

    KRMethod(a, b, alpha, beta, delta, gamma, y0, y1, h);

**var** newShootX = shootX;

**var** newShootY = [];

**var** newKRX = KRX;

**var** newKRY = [];

**for** (**var** i = 0, j = 0; i < N+1 && j < 2\*N+1 ; ++j) {

**if** (Math.round(shootX[i]\*100000) == Math.round(newShootX1[j]\*100000)) {

            newShootY.push(newShootY1[j]);

            newKRY.push(newKRY1[j]);

            ++i;

        }

    }

*//k = 2*

**for** (**var** i = 0 ; i < N+1 ; ++i) {

        shootErrs2.push(Math.abs(shootY[i] - newShootY[i]));

        KRErrs2.push(Math.abs(KRY[i] - newKRY[i]));

    }

}

**var** y\_der = 1;

**var** pointsX = [];

**var** pointsY = [];

**var** showExact = **true**;

**var** showShoot = **true**;

**var** showKR = **true**;

**function** draw(X, Y, type, iter) {

**var** i = iter;

**if** (i < X.length && i > 0) {

**var** point = **new** lib.bigPoint();

            stage.addChild(point);

            point.x = transfX(X[i]);

            point.y = transfY(Y[i]);

            point.type = type;

**var** join = **new** lib.line();

            stage.addChild(join);

            join.x = transfX(X[i-1]);

            join.y = transfY(Y[i-1]);

            join.endX = transfX(X[i]);

            join.endY = transfY(Y[i]);

            join.yGO = -9;

            join.decr = 0;

            join.life = lifeTime;

            point.yGO = -9;

            point.decr = 0;

            point.life = lifeTime;

            point.count = i;

            join.type = type;

**if** (type == "Shoot") {

                join.gotoAndStop(1);

                point.gotoAndStop(0);

            }

**else** {

                join.gotoAndStop(2);

                point.gotoAndStop(1);

            }

            join.len = Math.sqrt(Math.pow((join.endY - join.y), 2) + Math.pow((join.endX - join.x), 2));

            join.scaleX = join.len;

            join.rotation = Math.atan2((join.endY - join.y), (join.endX - join.x)) \* 180 / Math.PI;

            join.visible = **true**;

            join.alpha = 1;

            point.visible = **true**;

            point.alpha = 1;

            join.addEventListener('tick', liveLine);

            point.addEventListener('tick', livePoint);

        }

**else** **if** (i == 0) {

**var** point = **new** lib.bigPoint();

            stage.addChild(point);

            point.x = transfX(X[0]);

            point.y = transfY(Y[0]);

            point.type = type;

            point.yGO = -9;

            point.decr = 0;

            point.life = lifeTime;

            point.count = i;

**if** (type == "Shoot") {

                point.gotoAndStop(0);

            }

**else** {

                point.gotoAndStop(1);

            }

            point.visible = **true**;

            point.alpha = 1;

            point.addEventListener('tick', livePoint);

        }

}

**for** (**var** i = 0 ; i <= 300 ; ++i) {

**var** x = i/300;

**var** y = func(x);

    pointsX.push(x);

    pointsY.push(y);

}

*//Прорисовка ориг. функции*

**for** (**var** i = 0 ; i < 300 ; ++i) {

**var** join = **new** lib.line();

    stage.addChild(join);

    join.x = transfX(pointsX[i]);

    join.y = transfY(pointsY[i]);

    join.endX = transfX(pointsX[i+1]);

    join.endY = transfY(pointsY[i+1]);

    join.gotoAndStop(0);

    join.num = i;

    join.len = Math.sqrt(Math.pow((join.endY - join.y), 2) + Math.pow((join.endX - join.x), 2));

    join.scaleX = join.len;

    join.scaleY = 1.5;

    join.rotation = Math.atan2((join.endY - join.y), (join.endX - join.x)) \* 180 / Math.PI;

    join.visible = **true**;

    join.alpha = 1;

    join.addEventListener('tick', setPoses);

}

**function** setPoses(e) {

**var** object = e.currentTarget;

**if** (showExact) {

        object.visible = **true**;

    }

**else** {

        object.visible = **false**;

    }

    object.x = transfX(pointsX[object.num]);

    object.y = transfY(pointsY[object.num]);

    object.endX = transfX(pointsX[object.num+1]);

    object.endY = transfY(pointsY[object.num+1]);

    object.len = Math.sqrt(Math.pow((object.endY - object.y), 2) + Math.pow((object.endX - object.x), 2));

    object.scaleX = object.len;

    object.scaleY = 1.5;

    object.rotation = Math.atan2((object.endY - object.y), (object.endX - object.x)) \* 180 / Math.PI;

}

**function** liveLine(e) {

**var** object = e.currentTarget;

**var** gravity = 0.55;

**if** (object.life != lifeTime) {

        object.gotoAndStop(3);

        object.yGO += gravity;

        object.y += object.yGO;

        object.alpha \*= 0.95;

        object.scaleY += Math.pow(Math.max(0,(16 - object.scaleY)), 0.6);

        object.decr += 0.25;

        object.scaleY \*= (object.scaleY/(object.scaleY+object.decr));

    }

**if** (object.alpha <= 0.05) {

        object.alpha = 0;

        object.visible = **false**;

        object.removeEventListener('tick', liveLine);

        stage.removeChild(object);

    }

**if** (object.type == "Shoot") {

**if** (showShoot) {

            object.visible = **true**;

        }

**else** {

            object.visible = **false**;

        }

    }

**else** {

**if** (showKR) {

            object.visible = **true**;

        }

**else** {

            object.visible = **false**;

        }

    }

}

**function** livePoint(e) {

**var** object = e.currentTarget;

**var** gravity = 0.55;

**if** (object.life != lifeTime) {

        object.gotoAndStop(2);

        object.yGO += gravity;

        object.y -= object.yGO;

        object.alpha \*= 0.95;

    }

**if** (object.alpha <= 0.05) {

        object.alpha = 0;

        object.visible = **false**;

        object.removeEventListener('tick', livePoint);

        stage.removeChild(object);

    }

**if** (object.type == "Shoot") {

**if** (showShoot) {

            object.visible = **true**;

        }

**else** {

            object.visible = **false**;

        }

    }

**else** {

**if** (showKR) {

            object.visible = **true**;

        }

**else** {

            object.visible = **false**;

        }

**if** (mousePosX >= 128 && mousePosX <= 492 && graph > 2) {

**if** (object.count == closestI) {

**this**.exportRoot.miniUI.x = object.x + 160;

**this**.exportRoot.miniUI.y = mousePosY - 80;

            }

        }

    }

}

**this**.plusH.addEventListener("click", addH.bind(**this**));

**function** addH() {

    h += 0.01;

}

**this**.minusH.addEventListener("click", remH.bind(**this**));

**function** remH() {

**if** (h > 0.01) {

        h -= 0.01;

    }

}

window.addEventListener("keydown", doKeyDown.bind(**this**));

**function** doKeyDown(e) {

**if** (e.keyCode == 16) {

        shift = **true**;

    }

**if** (e.keyCode == 17) {

        control = **true**;

    }

}

window.addEventListener("keyup", doKeyUP.bind(**this**));

**function** doKeyUP(e) {

    shift = **false**;

    control = **false**;

}

**this**.plusSpd.addEventListener("click", addSpd.bind(**this**));

**function** addSpd() {

**if** (shift == **true** && control == **true**) {

        speed += 10;

        speed \*= 1.5

    }

**else** **if** (shift == **true**) {

        speed += 10;

    }

**else** **if** (control == **true**) {

        speed \*= 1.5;

    }

**else** {

        speed += 1;

    }

}

**this**.minusSpd.addEventListener("click", remSpd.bind(**this**));

**function** remSpd() {

**if** (shift == **true** && control == **true**) {

**if** (speed > 10) {

            speed -= 10;

            speed /= 1.5

        }

    }

**else** **if** (shift == **true**) {

**if** (speed > 10) {

            speed -= 10;

        }

    }

**else** **if** (control == **true**) {

        speed /= 1.5;

    }

**else** {

        speed -= 1;

    }

}

**this**.addEventListener("tick", main.bind(**this**));

**function** main() {

**if** (h > 1) {

        h = 1;

    }

**if** (h < 0.01) {

        h = 0.01;

    }

**if** (speed > 100000) {

        speed = 100000;

    }

**if** (speed < 1) {

        speed = 1;

    }

    h = Math.round(h\*100)/100;

    speed = Math.round(speed);

    a = Math.round(a\*100)/100;

    b = Math.round(b\*100)/100;

**if** (a >= b) {

        b = a + 0.01

    }

**this**.h\_text.text = "h = " + h;

**this**.spd\_text.text = "Speed: " + speed;

**if** (a < 0 && 1.2\*b >= 0) {

**this**.zeroAxis.x = transfX(0);

    }

**else** {

**this**.zeroAxis.x = 128.2

    }

**if** (begined) {

        frame += speed/300;

**var** iters = 0;

**if** (frame > 1 && graph < 3) {

            iters = Math.floor(frame);

            frame -= iters;

**for** (**var** i = 0 ; i < iters ; ++i) {

**if** (graph == 1) draw(shootX, shootY, "Shoot", iter);

**if** (graph == 2) draw(KRX, KRY, "KR", iter);

                ++iter;

            }

**if** (iter >= shootX.length) {

                iter = 0;

                graph++;

            }

        }

    }

**if** (showExact) {

**this**.showBox1.gotoAndStop(1);

    }

**else** {

**this**.showBox1.gotoAndStop(0);

    }

**if** (showShoot) {

**this**.showBox2.gotoAndStop(1);

    }

**else** {

**this**.showBox2.gotoAndStop(0);

    }

**if** (showKR) {

**this**.showBox3.gotoAndStop(1);

    }

**else** {

**this**.showBox3.gotoAndStop(0);

    }

    mousePosX = stage.mouseX / canvas.width \* 1080;

    mousePosY = stage.mouseY / canvas.height \* 720;

    closestDist = 9999999;

**if** (graph > 2) {

**for** (**var** i = 0 ; i < shootX.length ; ++i) {

**if** (Math.abs(mousePosX - transfX(shootX[i])) < closestDist) {

                closestDist = Math.abs(mousePosX - transfX(shootX[i]));

                closestI = i;

            }

        }

**if** (mousePosX >= 128 && mousePosX <= 492) {

**this**.miniUI.visible = **true**;

            stage.addChild(**this**.miniUI);

**this**.miniUI.pointNum.text = "Точка #" + (closestI+1);

**this**.miniUI.pX.text = "x = " + rou(shootX[closestI]);

**this**.miniUI.fX.text = "f(x) = " + rou(func(shootX[closestI]));

**this**.miniUI.Sh.text = "Shoot(x) = " + rou(shootY[closestI]);

**this**.miniUI.KR.text = "K-R(x) = " + rou(KRY[closestI]);

**this**.miniUI.Shd1.text = "Shoot = " + rou(shootErrs2[closestI]);

**this**.miniUI.KRd1.text = "ΔK-R = " + rou(KRErrs2[closestI]);

**this**.miniUI.Shd2.text = "Shoot = " + rou(shootErrs1[closestI]);

**this**.miniUI.KRd2.text = "ΔK-R = " + rou(KRErrs1[closestI]);

        }

**else** {

**this**.miniUI.visible = **false**;

        }

    }

*//set points*

**var** MmaxY = -99999999;

**var** MminY = 99999999;

**for** (**var** i = 0 ; i <= 300 ; ++i) {

**var** x = a + i/300\*(b-a);

**var** y = func(x);

        pointsX[i] = x;

        pointsY[i] = y;

**if** (y > MmaxY) {

            MmaxY = y;

        }

**if** (y < MminY) {

            MminY = y;

        }

    }

**for** (**var** i = 0 ; i < shootY.length ; ++i) {

**if** (shootY[i] > MmaxY) {

            MmaxY = shootY[i];

        }

**if** (shootY[i] < MminY) {

            MminY = shootY[i];

        }

    }

**for** (**var** i = 0 ; i < KRY.length ; ++i) {

**if** (KRY[i] > MmaxY) {

            MmaxY = KRY[i];

        }

**if** (KRY[i] < MminY) {

            MminY = KRY[i];

        }

    }

    maxY = Math.round(MmaxY \* 10800)/10000;

    minY = Math.round(MminY \* 9200)/10000;

**this**.y1.text = Math.round(minY\*1000)/1000 + "";

**this**.y2.text = Math.round((minY + (maxY - minY)\*(1/6))\*1000)/1000 + "";

**this**.y3.text = Math.round((minY + (maxY - minY)\*(2/6))\*1000)/1000 + "";

**this**.y4.text = Math.round((minY + (maxY - minY)\*(3/6))\*1000)/1000 + "";

**this**.y5.text = Math.round((minY + (maxY - minY)\*(4/6))\*1000)/1000 + "";

**this**.y6.text = Math.round((minY + (maxY - minY)\*(5/6))\*1000)/1000 + "";

**this**.y7.text = Math.round(maxY\*1000)/1000 + "";

**this**.x1.text = Math.round(a\*1000)/1000 + "";

**this**.x2.text = Math.round((a + (b - a)\*(1/5))\*1000)/1000 + "";

**this**.x3.text = Math.round((a + (b - a)\*(2/5))\*1000)/1000 + "";

**this**.x4.text = Math.round((a + (b - a)\*(3/5))\*1000)/1000 + "";

**this**.x5.text = Math.round((a + (b - a)\*(4/5))\*1000)/1000 + "";

**this**.x6.text = Math.round(b\*1000)/1000 + "";

}

**function** rou(num) {

**return** Math.round(num\*100000)/100000;

}

**this**.showBox1.addEventListener("click", setShow1.bind(**this**));

**function** setShow1() {

**if** (showExact) {

        showExact = **false**;

    }

**else** {

        showExact = **true**;

    }

}

**this**.showBox2.addEventListener("click", setShow2.bind(**this**));

**function** setShow2() {

**if** (showShoot) {

        showShoot = **false**;

    }

**else** {

        showShoot = **true**;

    }

}

**this**.showBox3.addEventListener("click", setShow3.bind(**this**));

**function** setShow3() {

**if** (showKR) {

        showKR = **false**;

    }

**else** {

        showKR = **true**;

    }

}

**this**.beginBt.addEventListener("click", startGame.bind(**this**));

**function** startGame() {

    lifeTime += 1;

    frame = 0;

    iter = 0;

    graph = 1;

    begined = **true**;

    shootingMethod(a, b, alpha, beta, delta, gamma, y0, y1, h, eps);

    KRMethod(a, b, alpha, beta, delta, gamma, y0, y1, h);

    exact\_errors();

    RRombergErrors(a, b, h);

}

**function** RungeKutta(a, b, h, y0, y\_der) {

**var** N = Math.floor((b - a)/h);

**var** x = [];

**var** y = [y0];

**for** (**var** i = a ; i <= b ; i += h) {

        i = Math.round(i\*10000)/10000;

        x.push(i);

    }

**var** z = [y\_der];

**for** (**var** i = 0 ; i < N; ++i) {

**var** K1 = h \* g(x[i], y[i], z[i]);

**var** L1 = h \* Koshi(x[i], y[i], z[i]);

**var** K2 = h \* g(x[i] + 0.5 \* h, y[i] + 0.5 \* K1, z[i] + 0.5 \* L1);

**var** L2 = h \* Koshi(x[i] + 0.5 \* h, y[i] + 0.5 \* K1, z[i] + 0.5 \* L1);

**var** K3 = h \* g(x[i] + 0.5 \* h, y[i] + 0.5 \* K2, z[i] + 0.5 \* L2);

**var** L3 = h \* Koshi(x[i] + 0.5 \* h, y[i] + 0.5 \* K2, z[i] + 0.5 \* L2);

**var** K4 = h \* g(x[i] + h, y[i] + K3, z[i] + L3);

**var** L4 = h \* Koshi(x[i] + h, y[i] + K3, z[i] + L3);

**var** delta\_y = (K1 + 2 \* K2 + 2 \* K3 + K4) / 6;

**var** delta\_z = (L1 + 2 \* L2 + 2 \* L3 + L4) / 6;

        y.push(y[i] + delta\_y);

        z.push(z[i] + delta\_z);

    }

*//console.log(y);*

    RungeX = x;

    RungeY = y;

}

**function** progonka(matrix, vectorB) {

**var** vectorX = [];

**var** N = vectorB.length;

**var** alphas = [];

**var** betas = [];

**for** (**var** i = 0 ; i < vectorB.length ; ++i) {

        alphas.push(0);

        betas.push(0);

    }

*//Прямой ход прогонки*

**for** (**var** i = 0; i < N; ++i) {

**var** A0, C0, B0, F0;

**if** (i - 1 < 0) {

            A0 = 0;

        }

**else** {

            A0 = matrix[i][i - 1];

        }

        C0 = -1 \* matrix[i][i];

**if** (i + 1 < N) {

            B0 = matrix[i][i + 1];

        }

**else** {

            B0 = 0;

        }

        F0 = vectorB[i];

**if** (i == 0) {

            alphas[i] = B0 / C0;

            betas[i] = -(F0 / C0);

        }

**else** **if** (i == N - 1) {

            alphas[i] = 0;

            betas[i] = (betas[i - 1] \* A0 - F0) / (C0 - alphas[i - 1] \* A0);

        }

**else** {

            alphas[i] = B0 / (C0 - alphas[i - 1] \* A0);

            betas[i] = (betas[i - 1] \* A0 - F0) / (C0 - alphas[i - 1] \* A0);

        }

    }

    vectorX[N - 1] = betas[N - 1];

**for** (**var** i = 2; i <= N; ++i) {

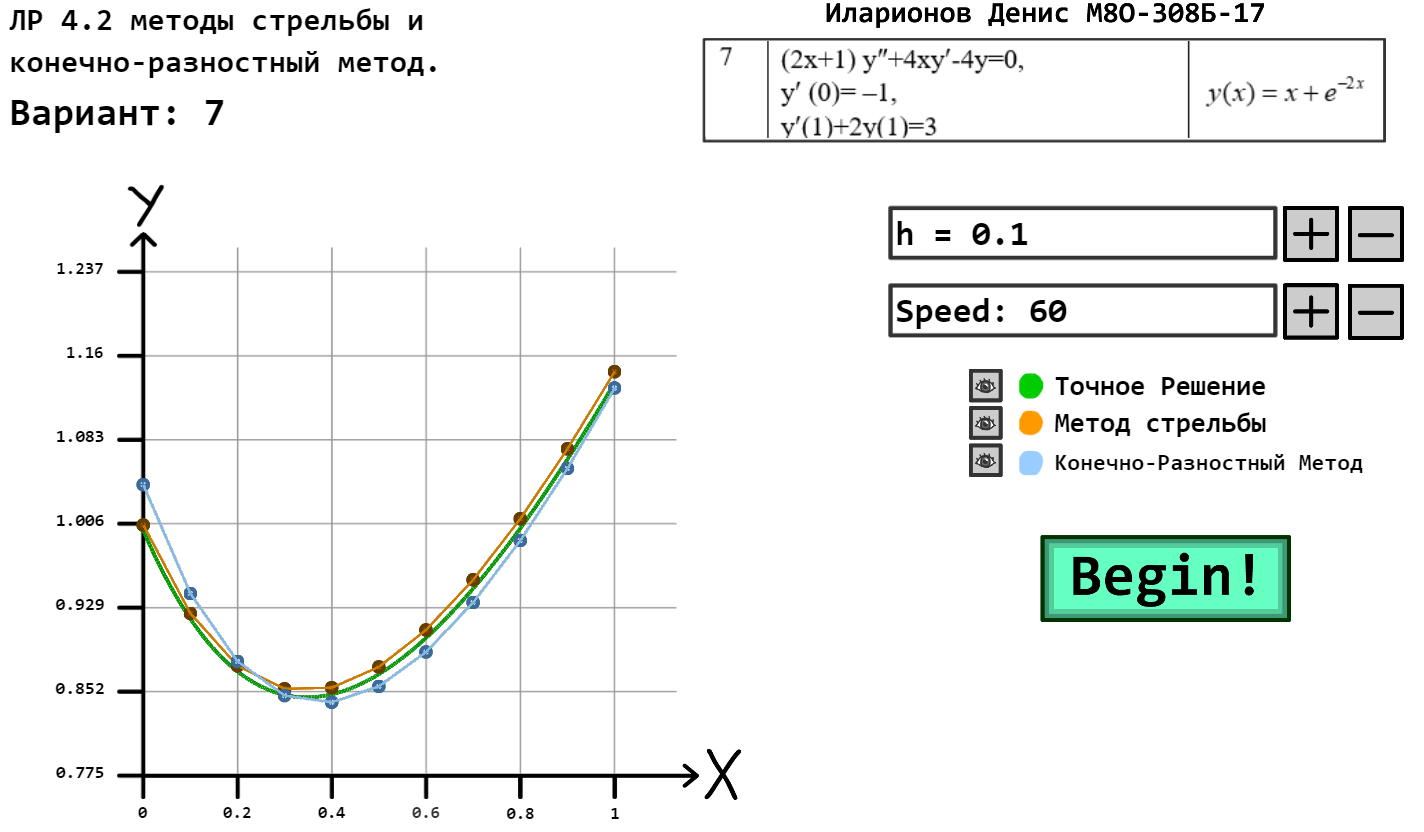
        vectorX[N - i] = alphas[N - i] \* vectorX[N - i + 1] + betas[N - i];

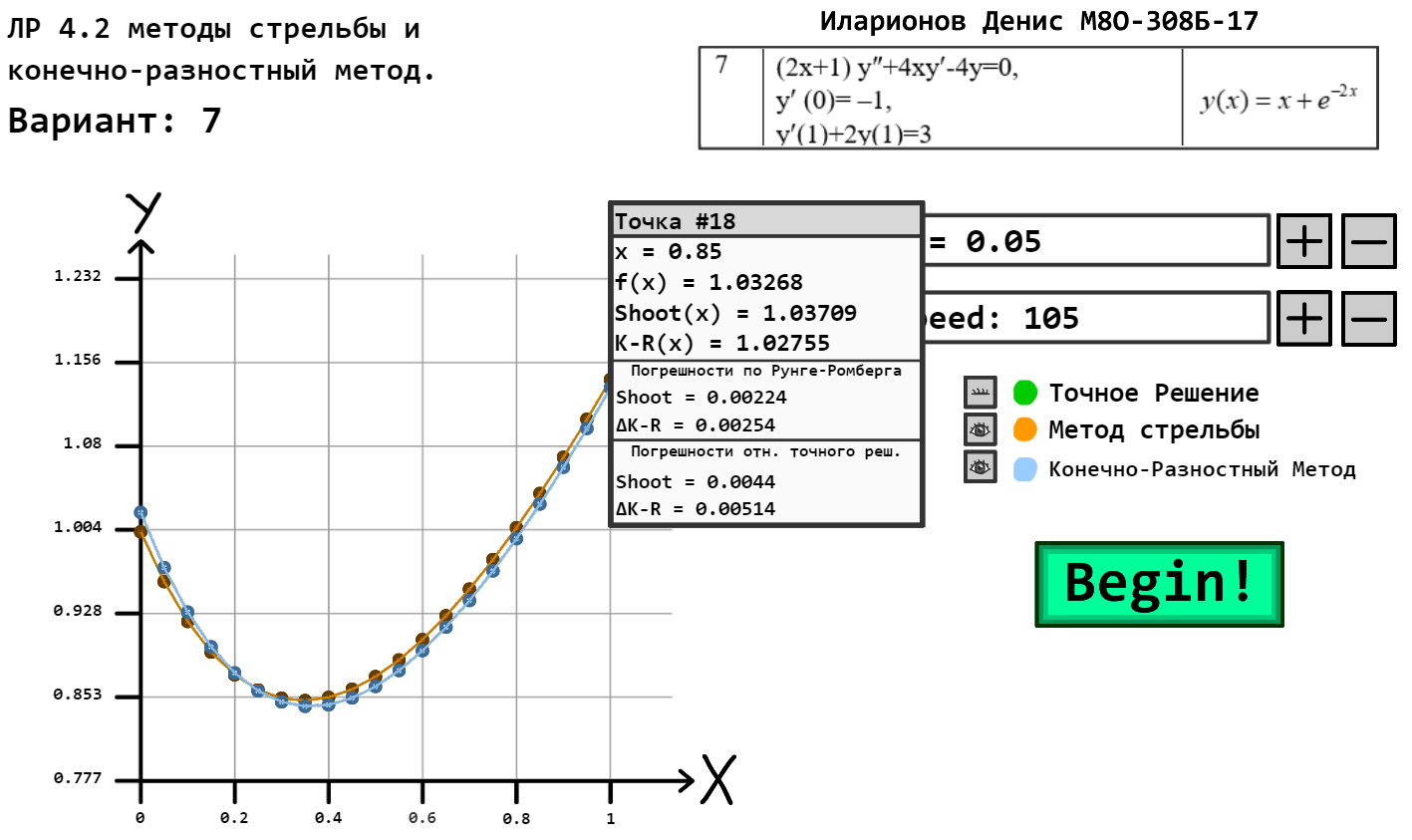
    }

**return** vectorX;

}

1. Тесты (Скриншоты моей программы)





1. Данная лабораторная работа выполнена: **8 мая 2020.**