## Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет имени Н.Э. Баумана**

**(национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)**

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

## Лабораторная работа № 2

**Тема** Многомерная интерполяция на регулярной сетке

**Студент** Батраков.Д.В

**Группа** ИУ7-42Б

## Оценка (баллы)

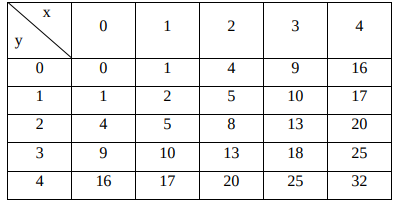
**Преподаватель** Градов В.М.

Москва. 2020 г

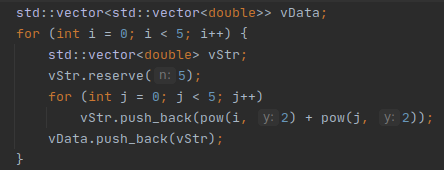
**Цель работы**. Получение навыков построения алгоритма интерполяции таблично

заданных функций полиномами Ньютона и Эрмита.

# Исходные данные



В коде можно быстрее представить так a[i][j] = i^2 + j^2



2. Степень аппроксимирующих полиномов.

3. Значение аргументов, для которых выполняется интерполяция

# Код программы

Main.cpp

#include <cmath>

#include "iomanip"

#include "TaskData.h"

#include "Newton.h"

#include "Approximation.h"

int main() {

std::vector<std::vector<double>> vData;

for (int i = 0; i < 5; i++) {

std::vector<double> vStr;

vStr.reserve(5);

for (int j = 0; j < 5; j++)

vStr.push\_back(pow(i, 2) + pow(j, 2));

vData.push\_back(vStr);

}

auto approximation = std::make\_unique<Approximation>(vData);

for (int n = 1; n < 4; n++)

std::cout << "power: " << n << " Result: " << approximation->FindResult(3.15, 2.25, n) << std::endl;

return 0;

}

Approximation.h

//

// Created by denis on 27.02.2021.

//

#ifndef LAB\_02\_APPROXIMATION\_H

#define LAB\_02\_APPROXIMATION\_H

#include "vector"

#include "TaskData.h"

class Approximation {

public:

explicit Approximation(std::vector<std::vector<double>> vData);

double FindResult(double searchX, double searchY, int power);

private:

std::vector<TaskData> GetTaskStringData(int index);

std::vector<std::vector<double>> m\_vData;

};

#endif //LAB\_02\_APPROXIMATION\_H

Approximation.cpp

//

// Created by denis on 27.02.2021.

//

#include "Approximation.h"

#include "Newton.h"

#include <utility>

#include <memory>

Approximation::Approximation(std::vector<std::vector<double>> vData): m\_vData(std::move(vData)) {

}

double Approximation::FindResult(double searchX, double searchY, int power) {

auto newton = std::make\_shared<Newton>(std::vector<TaskData>());

std::vector<double> vIndexes;

for (int i = 0; i < m\_vData.size(); i++)

{

newton->UpdateTaskData(GetTaskStringData(i));

vIndexes.push\_back(newton->GetNewtonResult(searchX, power));

}

std::vector<TaskData> finalData;

finalData.reserve(vIndexes.size());

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

finalData.emplace\_back(i, vIndexes[i], 0);

newton->UpdateTaskData(finalData);

return newton->GetNewtonResult(searchY, power);

}

std::vector<TaskData> Approximation::GetTaskStringData(int index) {

std::vector<TaskData> resV;

resV.reserve(m\_vData[0].size());

for (int i = 0; i < m\_vData[0].size(); i++)

resV.emplace\_back(i, m\_vData[index][i], 0);

return resV;

}

Newton.h

//

// Created by denis on 25.02.2021.

//

#ifndef LAB\_05\_NEWTON\_H

#define LAB\_05\_NEWTON\_H

#include <vector>

#include "TaskData.h"

class Newton {

public:

explicit Newton(std::vector<TaskData> inTaskData);

double GetNewtonResult(double x, int powerApproximation);

void UpdateTaskData(std::vector<TaskData> inTaskData);

private:

void FillChooseData(double x, int powerApproximation);

double GetResult(double x);

void FillDelta(std::vector<double> vCurrentDelta);

std::vector<TaskData> m\_vChooseData;

std::vector<TaskData> m\_vTaskData;

std::vector<std::vector<double>> m\_DeltaMatrix;

};

#endif //LAB\_05\_NEWTON\_H

Newton.cpp

//

// Created by denis on 25.02.2021.

//

#include "Newton.h"

#include "VectorUtils.h"

#include <utility>

Newton::Newton(std::vector<TaskData> inTaskData): m\_vTaskData(std::move(inTaskData)) {

}

void Newton::FillDelta(std::vector<double> vCurrentDelta) {

unsigned int deltaX = m\_vChooseData.size() - vCurrentDelta.size() + 1;

std::vector<double> vNewDelta;

vNewDelta.reserve(vCurrentDelta.size() - 1);

for (int i = 0; i < vCurrentDelta.size() - 1; i++) {

if (m\_vChooseData[i].x != m\_vChooseData[i + deltaX].x)

vNewDelta.push\_back((vCurrentDelta[i] - vCurrentDelta[i + 1]) / (m\_vChooseData[i].x - m\_vChooseData[i + deltaX].x));

else

vNewDelta.push\_back(m\_vChooseData[i].y\_p);

}

m\_DeltaMatrix.push\_back(vNewDelta);

}

double Newton::GetResult(double x) {

std::vector<double> vYDelta;

vYDelta.reserve(m\_vChooseData.size());

for (auto elem: m\_vChooseData)

vYDelta.push\_back(elem.y);

m\_DeltaMatrix.clear();

m\_DeltaMatrix.push\_back(vYDelta);

while (m\_DeltaMatrix.back().size() != 1)

FillDelta(m\_DeltaMatrix.back());

double resNewton = 0;

for (int i = 0; i < m\_DeltaMatrix.size(); i++) {

auto resTmp = m\_DeltaMatrix[i][0];

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

resTmp \*= x - m\_vChooseData[j].x;

resNewton += resTmp;

}

return resNewton;

}

double Newton::GetNewtonResult(double x, int powerApproximation) {

m\_vChooseData.clear();

FillChooseData(x, powerApproximation);

return GetResult(x);

}

void Newton::FillChooseData(double x, int powerApproximation) {

std::vector<TaskData> vDeltaXAndData;

for (auto elem: m\_vTaskData) {

vDeltaXAndData.push\_back(elem);

vDeltaXAndData.back().x = std::abs(x - elem.x);

}

auto copyData = m\_vTaskData;

while (m\_vChooseData.size() != powerApproximation + 1) {

auto findIndex = VectorUtils::GetVectorMinValue(vDeltaXAndData);

m\_vChooseData.push\_back(copyData[findIndex]);

vDeltaXAndData.erase(vDeltaXAndData.begin() + findIndex);

copyData.erase(copyData.begin() + findIndex);

}

VectorUtils::Sort(m\_vChooseData);

}

void Newton::UpdateTaskData(std::vector<TaskData> inTaskData) {

m\_vTaskData = std::move(inTaskData);

m\_vChooseData.clear();

}

TaskData.h

//

// Created by denis on 22.02.2021.

//

#ifndef LAB\_01\_TASKDATA\_H

#define LAB\_01\_TASKDATA\_H

class TaskData

{

public:

TaskData(double inX, double inY, double inY\_p);

double x;

double y;

double y\_p;

};

#endif //LAB\_01\_TASKDATA\_H

TaskData.cpp

//

// Created by denis on 25.02.2021.

//

#include "TaskData.h"

TaskData::TaskData(double inX, double inY, double inY\_p) {

x = inX;

y = inY;

y\_p = inY\_p;

}

VectorUtils.h

//

// Created by denis on 25.02.2021.

//

#ifndef LAB\_05\_VECTORUTILS\_H

#define LAB\_05\_VECTORUTILS\_H

#include <vector>

#include "TaskData.h"

#include "algorithm"

bool comp(TaskData first, TaskData second)

{

return first.x < second.x;

}

class VectorUtils

{

public:

static void Sort(std::vector<TaskData>& inVector);

static int GetVectorMinValue(std::vector<TaskData>& inVector);

};

int VectorUtils::GetVectorMinValue(std::vector<TaskData>& inVector) {

auto min = inVector[0];

int indexMin = 0;

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

if (inVector[i].x < min.x) {

min = inVector[i];

indexMin = i;

}

}

return indexMin;

}

void VectorUtils::Sort(std::vector<TaskData> &inVector) {

std::sort(inVector.begin(), inVector.end(), comp);

}

#endif //LAB\_05\_VECTORUTILS\_H

# Результаты работы

1. Результат интерполяции z(x,y) при степенях полиномов 1, 2, 3 для x=1.5, y=1.5 .



# Вопросы при защите лабораторной работы

Ответы на вопросы дать письменно в Отчете о лабораторной работе.

1. Пусть производящая функция таблицы суть z(x,y)=x 2 +y 2 . Область определения по x и y 0-5 и 0-5. Шаги по переменным равны 1. Степени nx = ny =1, x=y=1.5. Приведите по шагам те. значения функции, которые получаются в ходе последовательных интерполяций. по строкам и столбцу.

Берем два узла по столбцу и строке, а именно этот кусок таблицы:

Y/X 1 2

1 2 5

2 5 8

Результат интерполяции первой строки = 3.5

Результат интерполяции второй строки = 6.5

Результат интерполяции столбца из двух предыдущих значений = 5

1. Какова минимальная степень двумерного полинома, построенного на четырех узлах? На шести узлах?

На четырех – 0

На шести - 0

1. Предложите алгоритм двумерной интерполяции при хаотичном расположении узлов, т.е. когда таблицы функции на регулярной сетке нет, и метод последовательной интерполяции не работает. Какие имеются ограничения на расположение узлов при разных степенях полинома?

Коэффициенты полинома нужно будет находить по нескольким узлам в окрестности точки интерполяции. Также придется добавлять проверки из-за ограничений, таких как: если полином 1 степени, то коэфы не могут лежать на одной прямой, 2 степени, то не могут лежать на одной плоскости и т. д.

1. Пусть на каком-либо языке программирования написана функция, выполняющая интерполяцию по двум переменным. Опишите алгоритм использования этой функции для интерполяции по трем переменным.

Сделаем интерполяцию для двух переменных(как по плоскости) и полученные значения запомним. После прохода по каждой плоскости проведем интерполяцию уже для третьего аргумента, используя эти запомненные значения.

1. Можно ли при последовательной интерполяции по разным направлениям использовать полиномы несовпадающих степеней или даже разные методы одномерной интерполяции, например, полином Ньютона и сплайн?

Если по разным направлениям, то можно.

1. Опишите алгоритм двумерной интерполяции на треугольной конфигурации узлов.

Принцип почти такой же, только придется добавить доп проверки при каждой итериии на подсчете интерполяций по рядам.