**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

Проект

по дисциплине «Численные методы»

“Дробно-полиномиальная аппроксимация”

Студент

Группа АС 21-1 \_\_\_\_\_\_\_\_\_\_\_\_ Станиславчук С.М.

подпись, дата

Руководитель

Д.т.н, профессор кафедры АСУ \_\_\_\_\_\_\_\_\_\_\_\_ Седых И.А.

подпись, дата

Липецк 2023 г.

Содержание

2. Задание кафедры

3. Теоретическая часть

4. Постановка практической задачи

5. Программная реализация

6. Заключение

7. Программный код (в электронном виде)

2. Задание кафедры.

Нужно выбрать тему проекта (список в гугл таблице) и записаться в Google-таблице (ссылка ниже) в строке с соответствующим номером темы. Темы не должны повторяться. Каждый выбирает свою.

Темы приблизительные, можно предложить свой вариант. НО!!! Ваши предложения присылайте Ирине Александровне, их нужно будет обсудить.

Проект оформляется практически как лабораторная работа, но с некоторыми отличиями.

Выбранная тема: Дробно-полиномиальная аппроксимация

3. Теоретическая часть

Дробно-полиномиальная аппроксимация - это метод численного анализа, который используется для аппроксимации функций. Он основывается на представлении функции в виде отношения двух полиномов, что позволяет получить аппроксимацию функции с высокой точностью.

Дробно-полиномиальная аппроксимация использует формулы, которые позволяют выразить зависимость между двумя переменными, например, между переменной y (отклик) и переменной x (фактор). В частности, она представляет зависимость y от x в виде дробного полинома следующего вида:

y = b0 + b1 \* x^p1 + ... + bk \* x^pk

1 + a1 \* x^q1 + ... + am \* x^qm

Здесь b0, b1, ..., bk и a1, ..., am - это коэффициенты дробно-полиномиальной модели, которые нужно определить. p1, ..., pk и q1, ..., qm - это параметры модели, которые тоже необходимо выбрать.

Для нахождения оптимальных значений этих параметров используются статистические методы, которые минимизируют сумму квадратов ошибок аппроксимации (например, метод наименьших квадратов).

В общем виде формулы для дробно-полиномиальной аппроксимации могут выглядеть более сложно и содержать дополнительные коэффициенты и параметры, но их суть остается примерно такой же - описать зависимость между переменными в виде дробного полинома и подобрать оптимальные значения его коэффициентов и параметров на основе статистических данных.

4. Постановка практической задачи

Допустим, у нас есть набор данных, содержащий значения температуры воздуха на протяжении нескольких дней. Мы хотим построить кривую, которая будет аппроксимировать эти данные, чтобы мы могли предсказать температуру на будущие дни.

Для решения этой задачи мы можем использовать дробно-полиномиальную аппроксимацию, чтобы построить кривую, проходящую через наши данные.

5.

void fractionalPolynomial(double degree, double\* x, double\* y, double\* coef)

{

double n = degree + 1;

cout << "(int)n" << (int)n << " (double)n" << n << "\n";

double\*\* A = new double\* [(int)n];

double\* B = new double[(int)n];

double i\_d = 0;

double j\_d = 0;

for (double i = 0; i < n; i += 1 + (1.0 / n)) {

A[(int)i] = new double[(int)n];

B[(int)i] = 0;

for (double j = 0; j < n; j += 1 + (1.0 / n)) {

A[(int)i][(int)j] = 0;

for (int k = 0; k < MAX\_DEGREE; k++) {

A[(int)i][(int)j] += pow(x[k], i + j);

}

}

for (int k = 0; k < MAX\_DEGREE; k++) {

B[(int)i] += y[k] \* pow(x[k], i);

}

}

//std::cout << "A: \n";

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

{

for (int j = i + 1; j < (int)n; j++)

{

double factor = A[j][i] / A[i][i];

for (int k = i; k < (int)n; k++)

{

A[j][k] -= factor \* A[i][k];

//std::cout << A[j][k] << " ";

}

//std::cout << "\n";

B[j] -= factor \* B[i];

}

}

//std::cout << "B: \n";

for (int i = n - 1; i >= 0; i--)

{

coef[i] = B[i];

//std::cout << B[i] << "\n";

for (int j = i + 1; j < n; j++)

{

coef[i] -= A[i][j] \* coef[j];

}

coef[i] /= A[i][i];

}

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

{

delete[] A[i];

}

delete[] A;

delete[] B;

}

Этот метод реализует алгоритм дробной полиномиальной аппроксимации для подгонки полиномиальной функции к набору точек данных, заданных массивами x и y. Степень полинома задается параметром degree, который определяет количество вычисляемых коэффициентов.

Алгоритм сначала создает матрицу A размера (degree +1) x (degree +1) и вектор B размера (degree+1) для хранения системы линейных уравнений, которые необходимо решить для нахождения коэффициентов полинома. Затем он заполняет A и B, перебирая каждый член полиномиального уравнения и каждую точку данных во входных массивах.

Цикл для заполнения A вычисляет сумму степеней x[k] для каждой комбинации i и j от 0 до степени, где k - индекс текущей точки данных. Цикл для заполнения B вычисляет произведение y[k] и мощности x[k], соответствующей текущему i.

После того как A и B заполнены, алгоритм решает систему линейных уравнений, используя гауссово исключение с частичным поворотом. Это приводит к уменьшению A до верхней треугольной матрицы и соответствующему обновлению B. Полученная верхняя треугольная матрица может быть решена методом обратной подстановки для получения коэффициентов полинома.

В этой программе, коэффициенты дробно-полиномиальной модели вычисляются функцией fractionalPolynomial(). В качестве аргументов функция принимает массивы x и y с известными значениями независимой и зависимой переменных соответственно, а также степень degree дробно-полиномиальной модели. Вычисленные коэффициенты модели сохраняются в массив coef. Функция fractionalPolynomial() использует метод наименьших квадратов для решения системы линейных уравнений, чтобы найти коэффициенты модели.

Наконец, метод деаллоцирует память, используемую A и B.

Результат программы при входных данных

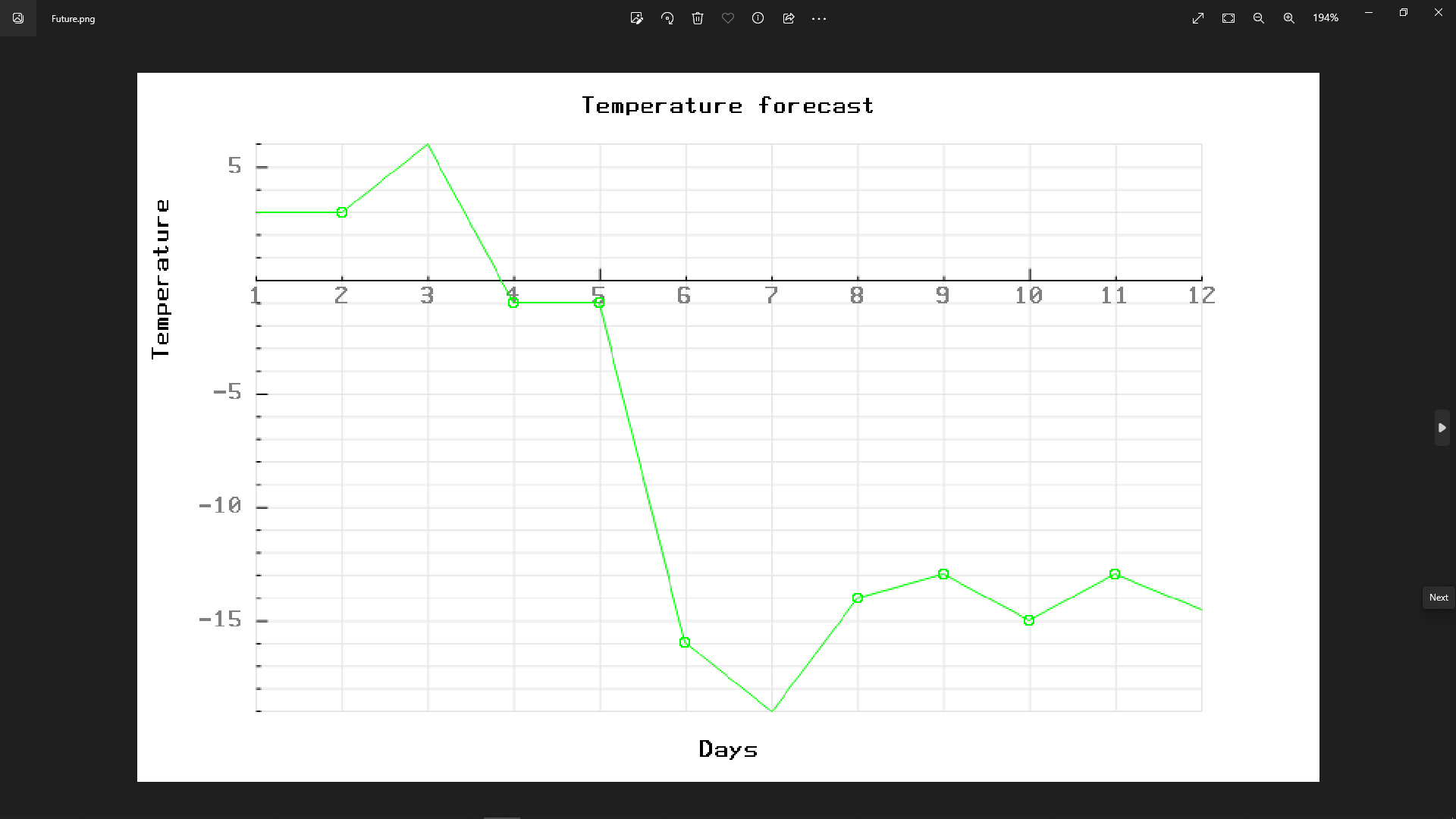
3, 3, 6, -1, -1, -16, -19, -14, -13, -15, -13; // Значения температуры

1.01.2020, 2.01.2020, 3.01.2020, 4.01.2020, 5.01.2020, 6.01.2020,

7.01.2020, 8.01.2020, 9.01.2020, 10.01.2020, 11.01.2020 // Даты

Результат:





(Дни на картинке отображаются в виде цифр, потому что библиотека, используемая для построения графика, умеет работать только с типом double (для дат я использовал собственную структуру Date, состоящей из дня, месяца и года))

6. Заключение

Таким образом, дробно-полиномиальная аппроксимация старых данных дневной температуры может помочь нам предсказать, какая температура будет завтра, и как она будет меняться в течение дня. Однако следует иметь в виду, что прогнозирование погоды - это сложная задача, и результаты могут быть неточными из-за различных факторов, таких как изменение погодных условий и т.п.

7. Полный код программы

#include <iostream>

#include <pbPlots.cpp>

#include <supportLib.cpp>

const int MAX\_DEGREE = 11; // Максимальная степень полинома

void plot(vector<double> x, vector<double> forecast, string plotname) {

RGBABitmapImageReference\* imageReference = CreateRGBABitmapImageReference();

ScatterPlotSeries\* seriesForecast = GetDefaultScatterPlotSeriesSettings();

seriesForecast->xs = &x;

seriesForecast->ys = &forecast;

seriesForecast->lineType = toVector(L"solid");

seriesForecast->pointType = toVector(L"circles");

seriesForecast->linearInterpolation = true;

seriesForecast->color = CreateRGBColor(0, 1, 0);

ScatterPlotSeries\* seriesForecastCircles = GetDefaultScatterPlotSeriesSettings();

seriesForecastCircles->xs = &x;

seriesForecastCircles->ys = &forecast;

seriesForecastCircles->lineType = toVector(L"solid");

seriesForecastCircles->pointType = toVector(L"circles");

seriesForecastCircles->linearInterpolation = false;

seriesForecastCircles->color = CreateRGBColor(0, 1, 0);

ScatterPlotSettings\* settings = GetDefaultScatterPlotSettings();

settings->width = 800;

settings->height = 480;

settings->title = toVector(L"Temperature forecast");

settings->xLabel = toVector(L"Days");

settings->yLabel = toVector(L"Temperature");

vector<ScatterPlotSeries\*> series({ seriesForecast, seriesForecastCircles });

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

settings->scatterPlotSeries->push\_back(series[i]);

}

DrawScatterPlotFromSettings(imageReference, settings, NULL);

vector<double>\* pngData = ConvertToPNG(imageReference->image);

WriteToFile(pngData, plotname);

cout << "\nPlot generated\n";

DeleteImage(imageReference->image);

}

struct Date {

int day;

int month;

int year;

};

void fractionalPolynomial(double degree, double\* x, double\* y, double\* coef)

{

double n = degree + 1;

cout << "(int)n" << (int)n << " (double)n" << n << "\n";

double\*\* A = new double\* [(int)n];

double\* B = new double[(int)n];

double i\_d = 0;

double j\_d = 0;

for (double i = 0; i < n; i += 1 + (1.0 / n)) {

A[(int)i] = new double[(int)n];

B[(int)i] = 0;

for (double j = 0; j < n; j += 1 + (1.0 / n)) {

A[(int)i][(int)j] = 0;

for (int k = 0; k < MAX\_DEGREE; k++) {

A[(int)i][(int)j] += pow(x[k], i + j);

}

}

for (int k = 0; k < MAX\_DEGREE; k++) {

B[(int)i] += y[k] \* pow(x[k], i);

}

}

//std::cout << "A: \n";

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

{

for (int j = i + 1; j < (int)n; j++)

{

double factor = A[j][i] / A[i][i];

for (int k = i; k < (int)n; k++)

{

A[j][k] -= factor \* A[i][k];

//std::cout << A[j][k] << " ";

}

//std::cout << "\n";

B[j] -= factor \* B[i];

}

}

//std::cout << "B: \n";

for (int i = n - 1; i >= 0; i--)

{

coef[i] = B[i];

//std::cout << B[i] << "\n";

for (int j = i + 1; j < n; j++)

{

coef[i] -= A[i][j] \* coef[j];

}

coef[i] /= A[i][i];

}

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

{

delete[] A[i];

}

delete[] A;

delete[] B;

}

int getTotalDays(Date date) {

int monthDays[] = { 0,31,59,90,120,151,181,212,243,273,304,334 };

int totalDays = (date.year - 2020) \* 365 + (date.year - 2020) / 4;

totalDays += monthDays[date.month - 1];

if ((date.month > 2) && (((date.year % 4 == 0) && (date.year % 100 != 0)) || (date.year % 400 == 0))) {

totalDays++;

}

totalDays += date.day;

return totalDays;

}

int main()

{

Date testiks[MAX\_DEGREE];

for (int i = 0; i < MAX\_DEGREE; i++) {

testiks[i].day = i + 1;

testiks[i].month = 1;

testiks[i].year = 2020;

}

double x[MAX\_DEGREE]; // Значения x

for (int i = 0; i < MAX\_DEGREE; i++) {

x[i] = getTotalDays(testiks[i]);

}

double y[MAX\_DEGREE] = { 19, 14, 13, 15, 13, 5, 9, 4, 1, 1, 16 }; // Значения y (значение погоды в градусах цельсия)

double coef[MAX\_DEGREE + 1]; // Коэффициенты полинома

double degree = 3.3; // Степень полинома (+1)

fractionalPolynomial(degree, x, y, coef);

std::cout << "Polynomial coefficients: ";

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

{

std::cout << coef[i] << " ";

}

std::cout << std::endl;

// Использования полинома

double future\_x = x[MAX\_DEGREE - 1] + 1;

double future\_y = 0;

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

{

future\_y += coef[i] \* pow(future\_x, i);

}

std::cout << "Given temperature: \n";

for (int i = 0; i < MAX\_DEGREE - 1; i++) {

std::cout << testiks[i].day << '.' << testiks[i].month << '.' << testiks[i].year << ": " << y[i] << "\n";

}

std::cout << "Predicted temperature for " << testiks[MAX\_DEGREE - 1].day << '.' << testiks[MAX\_DEGREE - 1].month << '.' << testiks[MAX\_DEGREE - 1].year << ": " << future\_y << "\n";

vector<double> \_x, \_y, \_futureX, \_futureY;

for (int i = 0; i < MAX\_DEGREE; i++) {

\_x.push\_back(x[i]);

\_y.push\_back(y[i]);

}

\_futureX = \_x;

\_futureY = \_y;

\_futureX.push\_back(future\_x);

\_futureY.push\_back(future\_y);

plot(\_futureX, \_futureY, "Future.png");

plot(\_x, \_y, "Past.png");

return 0;

}