|  |  |
| --- | --- |
|  | **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 Информатики и систем управления

КАФЕДРА Теоретической информатики и компьютерных технологий

**Лабораторная работа № 3**

«Аппроксимация методом наименьших квадратов. Двупараметрические модели»

по курсу «Численные методы»

Выполнил:

студент группы ИУ9-62Б

Марченко Андрей

Проверила:

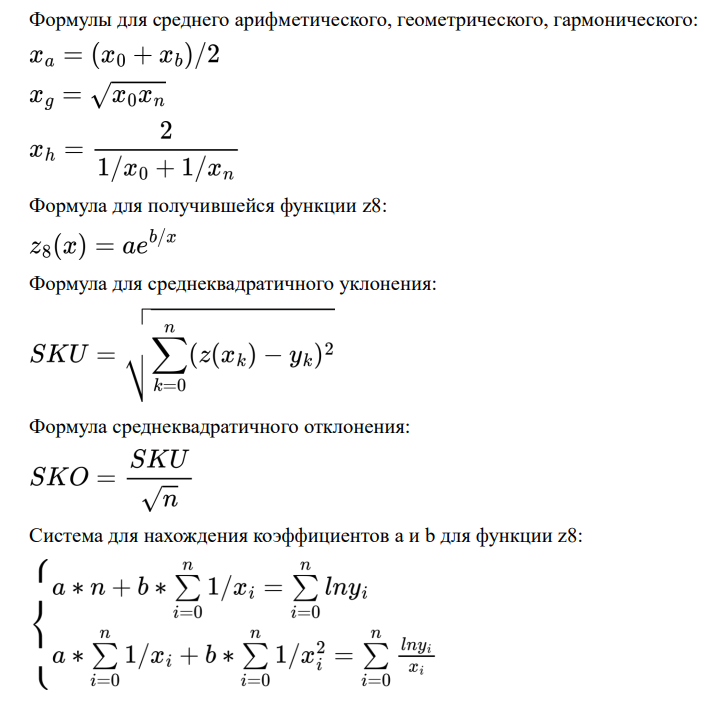
Домрачева А. Б.

Москва, 2024

1. **Цель**

Целью данной работы является решение задачи аппроксимации используя характеристические свойства заданных элементарных функций.

1. **Постановка задачи**
2. Построить графики таблично заданной функции z(x);
3. Найти значения x\_a, x\_g, x\_h, y\_a, y\_g, y\_h, z(x\_a), z(x\_g), z(x\_h). Также найти k = min(d\_i);
4. Составить систему линейных уравнений для определения коэффициентов a, b и решить её.
5. На йти среднеквадратичное отклонение
6. **Теоретические сведения**

****

1. **Индивидуальный вариант**

X = {1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0}

Y = {2.61, 1.62, 1.17, 0.75, 0.30, 0.75, 1.03, 0.81, 0.57}

1. **Реализация**

Листинг 1. Аппроксимация функции

#include <iostream>  
#include <vector>  
#include <cmath>  
#include <Eigen/Dense>  
#include <Eigen/LU>  
#include <iomanip>  
  
using namespace Eigen;  
  
std::vector <std::vector <int> > funcs = **{**{1, 3, 6}, {4, 2, 9}, {5, 8, 7}**}**;  
  
Eigen::VectorXd findLambd(const Eigen::MatrixXd& A, const Eigen::VectorXd& b) {  
 auto m = A.rows();  
 Eigen::MatrixXd U = Eigen::MatrixXd::Zero(m, m);  
 Eigen::VectorXd x = Eigen::VectorXd::Zero(m);  
 Eigen::VectorXd y = Eigen::VectorXd::Zero(m);  
  
 for (int i = 0; i < m; ++i) {  
 for (int j = 0; j < i; ++j) {  
 double sum = 0.0;  
 for (int k = 0; k < j; ++k) {  
 sum += U(i, k) \* U(j, k);  
 }  
 U(i, j) = (A(i, j) - sum) / U(j, j);  
 }  
  
 // прямой ход  
 double sum = 0.0;  
 for (int k = 0; k < i; ++k) {  
 sum += pow(U(i, k), 2);  
 }  
 U(i, i) = sqrt(A(i, i) - sum);  
  
 // обратный ход  
 double sum\_y = 0.0;  
 for (int k = 0; k < i; ++k) {  
 sum\_y += U(i, k) \* y(k);  
 }  
 y(i) = (b(i) - sum\_y) / U(i, i);  
 }  
  
 for (int i = m - 1; i >= 0; --i) {  
 double sum\_x = 0.0;  
 for (int k = i + 1; k < m; ++k) {  
 sum\_x += U(k, i) \* x(k);  
 }  
 x(i) = (y(i) - sum\_x) / U(i, i);  
 }  
 return x;  
}  
  
double avg(double x1, double x2) {  
 return (x1 + x2) / 2;  
}  
  
double geom(double x1, double x2) {  
 return sqrt(x1 \* x2);  
}  
  
double harm(double x1, double x2) {  
 return 2 / (1 / x1 + 1 / x2);  
}  
  
int findK(std::vector <double> xs, std::vector <double> ys, const std::function<double(double)>& z) {  
 double x\_first = xs[0], x\_last = xs[xs.size()-1];  
 double y\_first = ys[0], y\_last = ys[ys.size()-1];  
 std::vector <double> funcs\_x = **{**avg(x\_first, x\_last), // x\_a  
 geom(x\_first, x\_last), // x\_g  
 harm(x\_first, x\_last)**}**; // x\_h  
 std::vector <double> funcs\_y = **{**avg(y\_first, y\_last), // y\_a  
 geom(y\_first, y\_last), // y\_g  
 harm(y\_first, y\_last)**}**; // y\_h  
 double min = abs(z(funcs\_x[0]) - funcs\_y[0]);  
 int min\_x = 0, min\_y = 0;  
 for (int i = 0; i < funcs\_x.size(); i++) {  
 for (int j = 0; j < funcs\_y.size(); j++) {  
 if (std::abs(z(funcs\_x[i]) - z(funcs\_y[j])) < min) {  
 min = std::abs(z(funcs\_x[i]) - z(funcs\_y[j]));  
 min\_x = i;  
 min\_y = j;  
 }  
 }  
 }  
 return funcs[min\_x][min\_y];  
}  
  
void fillAb (MatrixXd & A, MatrixXd & b, int m, std::vector <double> xs, std::vector <double> ys, int n) {  
 for (int i = 0; i < m; ++i) {  
 for (int j = 0; j < m; ++j) {  
 A(i, j) = 0;  
 for (int k = 0; k <= n; ++k) {  
 A(i, j) += pow(xs[k], i + j);  
 }  
 }  
 b(i) = 0;  
 for (int k = 0; k <= n; ++k) {  
 b(i) += ys[k] \* pow(xs[k], i);  
 }  
 }  
}  
  
  
int main() {  
  
 int m = 4; // размерность системы  
 int n = 8; // кол-во точек разбиения  
 std::vector <double> xs = **{**1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0**}**;  
 std::vector <double> ys = **{**2.61, 1.62, 1.17, 0.75, 0.30, 0.75, 1.03, 0.81, 0.57**}**; // данные 17 варианта  
  
  
 // заполняем матрицу А и свободные коэффы b по формулам:  
 Eigen::MatrixXd A(m, m);  
 Eigen::VectorXd b(m);  
  
 for (int i = 0; i < m; ++i) {  
 for (int j = 0; j < m; ++j) {  
 A(i, j) = 0;  
 for (int k = 0; k <= n; ++k) {  
 A(i, j) += pow(xs[k], i + j);  
 }  
 }  
 b(i) = 0;  
 for (int k = 0; k <= n; ++k) {  
 b(i) += ys[k] \* pow(xs[k], i);  
 }  
 }  
  
 std::cout << "\nA:\n" << A << std::endl;  
 std::cout << "\nb:\n" << b.transpose() << std::endl;  
 Eigen::VectorXd lambd = findLambd(A, b);  
 std::cout << "\nλ:\n" << lambd.transpose() << std::endl;  
  
 // задаём функцию z(x) согласно найденным лямбдам  
 auto z = [&lambd, m](double x) {  
 double result = 0;  
 for (int i = 0; i < m; ++i) {  
 result += lambd(i) \* pow(x, i);  
 }  
 return result;  
 };  
  
 std::cout << "k: " << findK(xs, ys, z) << std::endl; // результат - 5  
  
  
  
 Eigen::MatrixXd A\_new(2, 2);  
 Eigen::VectorXd b\_new(2);  
  
 std::vector <double> xs\_rev (xs.size());  
 for (int i = 0; i < xs.size(); i++) {  
 xs\_rev[i] = 1 / xs[i];  
 }  
  
 for (int i = 0; i < 2; ++i) {  
 for (int j = 0; j < 2; ++j) {  
 A\_new(i, j) = 0;  
 for (int k = 0; k <= n; ++k) {  
 A\_new(i, j) += pow(xs\_rev[k], i + j);  
 }  
 }  
 b\_new(i) = 0;  
 for (int k = 0; k <= n; ++k) {  
 b\_new(i) += ys[k] \* pow(xs\_rev[k], i);  
 }  
 }  
  
 VectorXd new\_lambd = findLambd(A\_new, b\_new);  
  
 std::cout << "New lambd: " << std::endl << new\_lambd.transpose();  
 auto z\_new = [&new\_lambd](double x) {  
 return new\_lambd[0] / x + new\_lambd[1];  
 };  
  
 // считаем абсолютную погрешность аппроксимации  
 double D = 0;  
 for (int k = 0; k <= m; ++k) {  
 D += pow(ys[k] - z\_new(xs[k]), 2);  
 }  
  
  
 D = sqrt(D) / sqrt(n);  
 std::cout << "\nСКО: " << D << std::endl;  
  
 // считаем относительню ошибку  
 double d = 0;  
 for (int k = 0; k <= n; ++k) {  
 d += pow(ys[k], 2);  
 }  
 d = D / sqrt(d);  
 std::cout << std::fixed << std::setprecision(4);  
 std::cout << "\nотносительная погрешность: " << d << std::endl;  
  
 return 0;  
}

1. **Результаты**

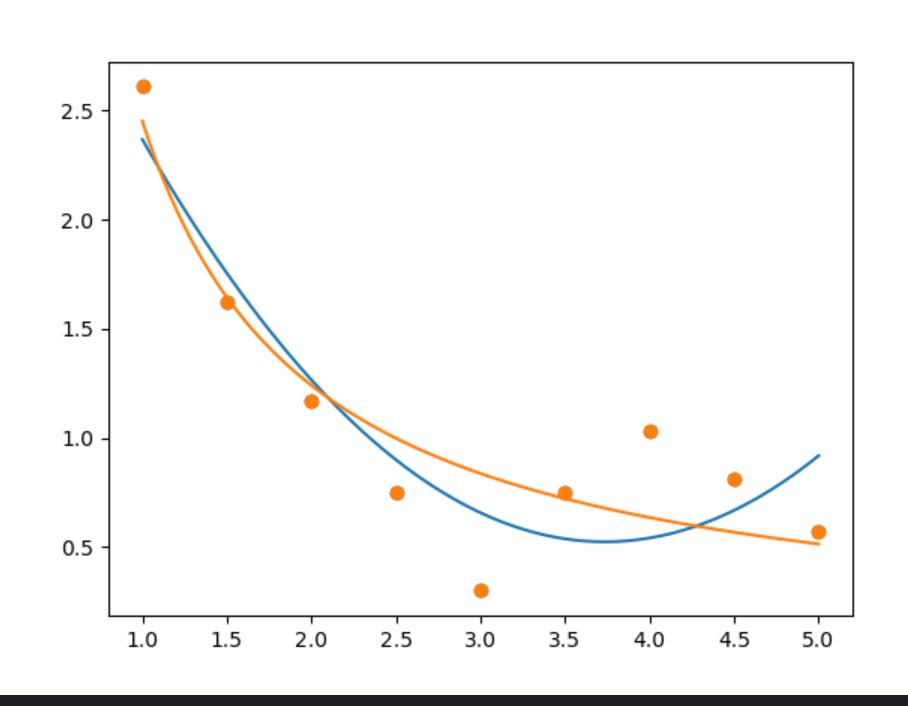
В результате отбора выбрана функция под номером 5: z\_5(x) = , где a = 0.029109, b = 2.49

СКО: 1.10086

Также была посчитана относительная погрешность: 0.2936

Также был построен график:

Рисунок 1. Графики функций



Где оранжевые точки – входные данные, синяя кривая – приближение, полученное на основе 8 лабораторной работы, и оранжевая кривая – приближение, полученное данной лабораторной работой.

1. **Выводы**

В лабораторной работе была получена аппроксимация заданных точек одной из заданных функций с подсчитанными коэффициентами. Полагаясь на СКУ и полученный график функции, можно сказать, что аппроксимация получена успешно.