Санкт-Петербургский государственный университет

Прикладная математика и информатика

Отчет по лабораторной работе №3.1 по вычислительному практикуму

**Задача обратного интерполирования**

Выполнил:

Чернов Павел Олегович

группа 221

Санкт-Петербург

2022

1. **Введение**

**Подготовительный этап:**

ВЫВЕСТИ НА ПЕЧАТЬ таблицу из (m+1) значения функции f в равноотстоящих с шагом h=(b-a)/m точках (узлах) xi=a+i·h, где i=0,1,..,m. Узлы xi‒точки деления отрезка [a; b] на m частей.

Здесь число значений в таблице m+1, a, b—параметры задачи; формула для непрерывной функции f, значе-ниями которой заполняется таблица, дана в варианте тестовой задачи.

**Решение задачи обратного интерполирования:**

Дана таблично-заданная функция (смотри таблицу, созданную на подготовительном этапе). Найти значение/значения аргумента/аргументов (задача может иметь не единственное решение!), при котором данная таблично−заданная функция принимает значение F, здесь F—параметр задачи.

1 способ решения:

ПУСТЬ таблично-заданная функция, для которой решается задача, строго монотонна (предполагается, что функция f, таблица которой дана в задаче, на рассматриваемом участке ‒это строго монотонная и непрерывная функция, то у нее существует обратная функция f-1, которая также строго монотонна и непрерывна).

ТОГДА задача обратного интерполирования может быть сведена к задаче поиска значения f-1(F) для таблично-заданной функции f-1 (при этом следует поменять местами столбцы исходной таблицы и далее трактовать значения f(xi) как аргументы для f-1).

Таким образом, ИМЕЕМ задачу алгебраического интерполирования для таблично-заданной функции f-1, где F‒точка интерполирования. Теперь, если построить интерполяционный многочлен Qn по таблице значений, то решением задачи будет значение Qn(F) ≈ f-1(F).

Степень интерполяционного многочлена n ─ параметр задачи (n≤m) ‒запросить у пользователя. При нахождении значения Qn(F) использовать программу из Задания №2 (представление в форме Лагранжа или Ньютона –неважно).

Результатом решения задачи обратного интерполирования 1 способом является значение X=Qn(F).

ПРОВЕРКА: В тестовой задаче всегда можно посчитать модуль невязки rn(X)=│f(X)−F│.

2 способ решения:

Если мы не располагаем информацией, что на рассматриваемом участке таблицы функция строго монотонна и непрерывна, и, следовательно, не полномочны «переворачивать таблицу», то возможно следующее решение. Также этот способ решения можно применять, если первый способ возможен, но не дал хороший результат (например, если обратная функция плохо приближается многочленом).

Результатом решения задачи обратного интерполирования 2 способом будет(ут) корень(ни) уравнения Pn(x)=F, где Pn(x) –интерполяционный полином функции f(x).

При построении интерполяционного многочлена Pn(x)можно использовать программу из ЛР №2. Алгебраическое уравнение решить методом секущих или методом бисекции c точностью ε (смотри ЛР №1).

ПРОВЕРКА: В тестовой задаче всегда можно посчитать модуль невязки rn(x)=│f(x)−F│ для каждого приближенного решения.

1. **Ход работы**
2. Представленная ниже программа позволяет найти приближенный аргумент, при котором заданная функция принимает значение F.
3. Для аппроксимации функции используется нахождение интерполяционного многочлена как в форме Лагранжа, так и в форме Ньютона.
4. Исходным параметром задачи является функция . В программе можно задать а) число табличных значений, по которым будет проводиться приближение, б) концы отрезка [a; b] из которого выбираются узлы аппроксимации, в) F –точка интерполирования, аргумент которой хотим найти, г) n – степень интерполяционного многочлена, который будет построен для того, чтобы найти аргумент для значения F, д) ε – допустимая погрешность вычисления. Ввод осуществляется при помощи клавиатуры.
5. Программа написана на языке С++, в расчетах используются переменные типа double.
6. **Приложение**

Программа:

#include "First\_method.h"

void sort(std::vector<std::pair<double, double>>& table, double x) { //Сортировка таблицы значений относительно расстояния до точки интерполяции

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

for (int j = i + 1; j < table.size(); ++j) {

if (abs(table[j].second - x) < abs(table[i].second - x)) {

std::pair<double, double> tmp = table[j];

table[j] = table[i];

table[i] = tmp;

}

}

}

}

double Newton\_poly\_correct(unsigned int n, double x, const std::vector <std::pair<double, double>> table) {

double res = table[0].first, mult = 1;

std::vector <std::vector<double>> A(n + 1, std::vector<double>(n + 1));

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

A[i][0] = table[i].first;

}

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

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

A[i][j] = (A[i + 1][j - 1] - A[i][j - 1]) / (table[j + i].second - table[i].second);

}

}

std::string smult;

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

mult \*= (x - table[i].second);

res += A[0][i + 1] \* mult;

}

return res;

}

#include "Second\_method.h"

double Newton\_poly\_corr(unsigned int n, double x, std::vector <std::pair<double, double>> table) {

double res = table[0].second, mult = 1;

std::vector <std::vector<double>> A(n + 1, std::vector<double>(n + 1));

/\*std::cout << "Newton polynomial:\n";

std::cout << res << "+";\*/

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

A[i][0] = table[i].second;

}

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

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

A[i][j] = (A[i + 1][j - 1] - A[i][j - 1]) / (table[j + i].first - table[i].first);

}

}

//std::string smult;

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

mult \*= (x - table[i].first);

//smult += "(x-" + std::to\_string(table[i].first) + ")";

//std::cout << A[0][i + 1] << smult << "+";

res += A[0][i + 1] \* mult;

}

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

return res;

}

double secants(double x0, double x1, double epsilon, unsigned int n, const std::vector <std::pair<double, double>> &table\_values, double F) {

double x\_approx = x1, x\_before\_last = x0, x\_last = x1 + epsilon \* 2;

while (abs(x\_approx - x\_last) > epsilon) {

x\_before\_last = x\_last;

x\_last = x\_approx;

x\_approx = x\_last - (Newton\_poly\_corr(n, x\_last, table\_values) - F)\* (x\_last - x\_before\_last) / (Newton\_poly\_corr(n, x\_last, table\_values) - Newton\_poly\_corr(n, x\_before\_last, table\_values));

}

//std::cout << "Absolute value of discrepancy : " << abs(Newton\_poly\_corr(n, x\_approx, table\_values) - F) << "\n";

return x\_approx;

}

#include "Header.h"

double f(double x) { //Интерполируемая функция

return log(1 + x) - exp(x); //вариант 13

}

#pragma once

#include <Windows.h>

#include <string>

#include "First\_method.h"

#include "Second\_method.h"

int main() {

SetConsoleTitleA("Algebraic reverse interpolation problem");

std::cout << std::fixed << std::setprecision(16);

unsigned int m; //Кол-во точек в таблице (m = 10)

std::cout << "Input number of table values: ";

std::cin >> m;

double a, b; //Концы отрезка интерполирования (a = 0, b = 1)

std::cout << "Input segment boundaries: ";

std::cin >> a >> b;

std::cout << "Table of values: \n" << std::setw(15) << "X\_i" << std::setw(17) << "f(X\_i)\n";

std::vector <std::pair<double, double>> table\_values(m);

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

table\_values[i].first = a + (b - a) \* i / m;

table\_values[i].second = f(table\_values[i].first);

std::cout << table\_values[i].first << "|" << table\_values[i].second << "\n";

}

int flag;

do {

double x; //Точка интерполирования

std::cout << "Input interpolation point: ";

std::cin >> x;

while (x > -1) {

std::cout << "Have you ever seen the initial function? You are inputting values out of its range, stupid. Fine, I'm giving you one another try: ";

std::cin >> x;

}

unsigned int n; //Степень интерполяционного М.Ч. (n = 10)

std::cout << "Input degree of interpolation polinom (Warning, degree must be less than number of table values, i.e. n < m): ";

std::cin >> n;

while (n >= m) {

std::cout << "Incorrect input, consider condition n < m, try again: ";

std::cin >> n;

}

if (n < m - 1) {

sort(table\_values, x);

std::cout << "Sorted table of values: \n" << std::setw(15) << "f(X\_i)" << std::setw(17) << "X\_i\n";

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

std::cout << table\_values[i].second << "|" << table\_values[i].first << "\n";

}

}

double first\_res = f(Newton\_poly\_correct(n, x, table\_values)), f\_res = x;

std::cout << "Value obtained by the first method: " << first\_res << "\n" <<

"Value of absolute error rate: " << abs(first\_res - f\_res) << "\n\n";

std::cout << "Input value of error rate for the second method: ";

double epsilon;

std::cin >> epsilon;

std::cout << "Value obtained by the second method: ";

double second\_res = f(secants(a, b, epsilon, n, table\_values, x));

std::cout << second\_res << "\n" <<

"Value of absolute error rate: " << abs(second\_res - f\_res) <<"\n\n";

std::cout << "If you want to try another interpolation point, enter 1, otherwise 0: ";

std::cin >> flag;

} while (flag);

return 0;

}

https://github.com/Chernovuk/Chislaki/tree/master/Third\_task/3.1