МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технологический университет «СТАНКИН»
(ФГБОУ ВО «МГТУ «СТАНКИН»)

Институт информационных систем и технологий

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ

Вычислительная математика

Отчет по лабораторной работе

«Интерполирование функции кубическим сплайном»

Вариант 17

Выполнил студент гр. ИДБ-21-06 Музафаров К.Р.

Проверил преподаватель Красикова Е.М.

Москва 2023г.

Краткие теоретические сведения

Постановка задачи:

Пусть на некотором отрезке [*a, b*] действительной оси существует некоторая непрерывная функция *y(x)*, значения которой известны лишь в *n*+1 точке данного отрезка, которые обозначим через x_i, i = $0,\bar{n}$ и x_i = a + ih, h = $\frac{b-a}{n}$, x_0 = a, x_n = b. Требуется найти для каждых двух соседних точек (узлах) x_i, x_{i+1}, i = $0,\bar{n-1}$ данного отрезка кубический полином, аппроксимирующий данную функцию в каждой точке интервала (x_i , x_{i+1}), значения которого совпадают со значениями функции на концах интервала.

Решение задачи:

Введем общее обозначение для такого полинома на каждом таком интервале (x_i, x_{i+1}), i = $0,\bar{n-1}$ через *f(x)*. Его коэффициенты определяются из условия сопряжения в узлах:

$$f_i = y_i$$

$$f'(x_i - 0) = f'(x_i + 0)$$

$$f''(x_i - 0) = f''(x_i + 0), i = 1,\bar{n-1}$$

Кроме того, на границах при $x = x_0$, $x = x_n$ ставятся условия:

$$f''(x_0) = f''(x_n) = 0 \quad (1)$$

Кубический полином ищется в виде:

$$f(x) = a_i + b_i(x - x_{i-1}) + c_i(x - x_{i-1})^2 + d_i(x - x_{i-1})^3,$$

$$X\epsilon[x_i, x_{i+1}]\,. \quad (2)$$

Из условия $f_i = y_i$ следует:

$$f(x_{i-1}) = a_i = y_{i-1}$$

$$f(x_i) = a_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = y_i,$$

$$h_i = x_i - x_{i-1}, i = 1,\bar{n-1}\,. \quad (3)$$

Вычислим производные:

$$f'(x) = b_i + 2c_i(x - x_{i-1}) + 3d_i(x - x_{i-1})^2,$$

$$f''(x) = 2c_i + 6d_i(x - X_{i-1}), x\epsilon[x_i, x_{i+1}],$$

и потребуем их непрерывности при x = x_i:

$$\begin{cases} b_{i+1} = b_i + 2c_i h_i + 3d_i h_i^2 \\ c_{i+1} = c_i + 3d_i h_i, i = 1,\bar{n} - 1 \end{cases} \quad (4)$$

Общее число неизвестных коэффициентов равно 4n, число уравнений (3) и (4) равно 4n-2. Недостающие два уравнения получаются из условий (1) при $x = x_0$ и x = x_n^2:

$$c_1 = 0, c_n + 3d_n h_n = 0.$$

Выражая из (4) $d_i = c_{i+1} - c_i/3h_i$, подставляя это выражение в (3) и исключая $a_i = y_{i-1}$, получим

$$b_i = [(y_i - y_{i-1}) - \frac{1}{3}/h_i(C_{i+1} + 2c_i), i = 1,\bar{n} - 1,$$

$$b_n = [(y_n - y_{n-1}/h_n] - \frac{2}{3}h_n c_n .$$

Подставив теперь выражения для b_i, b_{i+1}и d_i в первую формулу (4), после несложных преобразований получаем для c_i разностное уравнение второго порядка

$$h_i c_i + 2(h_i + h_{i+1})c_{i+1} + h_{i+1}c_{i+2} = 3(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i}),$$
$$i = 1,\bar{n} - 1, \quad (5)$$

с краевыми условиями

$$c_1 = c_{n+1} = 0. \quad (6)$$

Условие $cn + 1 = 0$ эквивалентно условию $c_n + 3d_n h_n = 0$ и уравнению $c_{i+1} + 3d_i h_i$. Разностное уравнение (9) с условиями (6) решается методом прогонки.

Метод прогонки

Метод прогонки – простой и эффективный алгоритм решения систем линейных алгебраических уравнений с трехдиагональными матрицами:

$$b_1x_1 + c_1x_2 = d1,$$
$$a_2x_1 + b_2x_2 + c_2x_3 = d2$$
$$a_3x_2 + b_3x_3 + c_3x_4 = d3$$
$$\dots$$
$$a_{n-1}x_{n-2} + b_{n-1}x_{n-1} + c_{n-1}x_n = d_{n-1}$$
$$a_nx_{n-1} + c_nx_n = d_n$$

Системы такого вида часто возникают при решении различных математической физики, а также при решении других вычислительных задач (например, приближение функций сплайнами).

Вывод расчетных формул

Преобразуем первое уравнение системы к виду:

$x_1=\alpha_1x_2+\beta_1$,где $\alpha_1 = -\frac{c}{b_1}, \beta_1 = d_1/b_1$.

Подставим полученноедля $x1$ выражение во второе уравнение системы:

$a_2(\alpha_1x_2 + \beta_1) + b_2x_2 + c_2x_3 = d_2$

Преобразуем это уравнение к виду:

$x_2 = \alpha_2x_3 + \beta_2$, где $\alpha_2 = -\frac{c_2}{b_2+a_2\alpha_1}, \beta_2 = \frac{d_2-a_2\beta_1}{b_2+a_2\alpha_1}$. Это выражение подставляем в третье уравнение системы и т.д.

На i-ом шаге этого процесса ($1 < i < n$) i-Е уравнение системы преобразуется

к виду $x_i = \alpha_ix_i+1 + \beta_i$, где $\alpha_i = -\frac{c_i}{b_i+a_i\alpha_{i-1}}, \beta_i = \frac{d_i-a_i\beta_{i-1}}{b_i+a_i\alpha_{i-1}}$.

На n-м шаге подстановка в последнее уравнение выражения $x_{n-1} = \alpha_{n-1}x_n + \beta_{n-1}$ дает:

$$a_n(\alpha_n - 1x_n + \beta_n - 1) + b_n x_n = d_n$$

Отсюда можно определить значения x_n:

$$x_n = \beta_n = \frac{d_n - a_n\beta_{n-1}}{b_n + a_n\alpha_{n-1}}$$

Значения остальных неизвестных x_i для $i = n - 1, n - 2, \dots, 1$ теперь легко вычисляются по формуле $x_i = \alpha_i x_{i+1} + \beta_i$.

Алгоритм прогонки

Прямой ход метода прогонки (прямая прогонка) состоит в вычислении прогоночных коэффициентов αi $(1 \le i \le n)$ и $\beta i (1 \le i \le n)$.

При i = 1 коэффициенты вычисляются по формулам:

$$\alpha_1 = -\frac{c_1}{y_1}, \beta_1 = \frac{d_1}{y_1}, y_1 = b_1,$$

а при i = 2, 3, ... , n − 1 – по рекуррентным формулам:

$$\alpha_i = -\frac{c_i}{y_i}, \beta_i = \frac{d_i - a_i\beta_{i-1}}{y_i}, y_i = b_i + a_i\alpha_{i-1}$$

При $i = n$ прямая прогонка завершается вычислением:

$$\beta_n = \frac{(d_n - a_n\beta_{n-1})}{y_n}, y_n = b_n + a_n\alpha_{n-1}$$

Обратный ход метода прогонки (обратная прогонка) дает значения неизвестных. Сначала полагают $xn = \beta n$. Затем значения остальных неизвестных вычисляют по формуле:

$x_i = \alpha_i x_i + 1 + \beta_i$, $i = n - 1, n - 2, \dots, 1$

Вычисления ведут в порядке убывания значений i от n − 1 до 1.

ВЫПОЛНЕНИЕ РАБОТЫ

1. Табличное задание функции.

2. Код программы,определяющий коэффициенты полиномов методом матричной прогонки.

```
#include <iostream>
#include <cmath>
const int n = 5; // number of cubic polynomials
const float a = -2.5;
const float b = 2.5;
const float h = (b - a) / n;
using namespace std;
void fillFunction(float*& X, float*& Y) {
	for (size_t i = 0; i < n + 1; i++) {
		X[i] = a + i * h;
		Y[i] = sin(12*X[i]/7)+cos(12*X[i]/7);
	}
}
void fillMatrix(float**& matrix, unsigned int& N, float* Y) {
	N = n - 1;
	const unsigned int _N = n - 1;
	matrix = new float* [N]; for (size_t i = 0; i < N; i++)
		matrix[i] = new float[N + 1];
	for (size_t i = 0; i < N; i++) for (size_t j = 0; j < N; j++)
		matrix[i][j] = 0;
	for (size_t i = 0; i < N; i++) {
		if (i == 0) {
			matrix[i][i] = 4;
			matrix[i][i + 1] = 1;
		}
		else if (i == N - 1) {
			matrix[i][i - 1] = 1;
			matrix[i][i] = 4;
		}
		else {
			matrix[i][i - 1] = 1; matrix[i][i] = 4; matrix[i][i + 1] = 1;
		}
	}
	for (size_t i = 0; i < N; i++) {
		if (i == 0)
			matrix[i][N] = 3.0 / (h * h) * (Y[i + 2] - 2 * Y[i + 1]);
		else if (i == N - 1)
			matrix[i][N] = 3.0 / (h * h) * (Y[i] - 2 * Y[i + 1]); else
			matrix[i][N] = 3.0 / (h * h) * (Y[i + 2] + Y[i] - 2 * Y[i + 1]);
	}
}
void print(float** matrix, unsigned int N) {
```

```
	cout << "--------------" << endl; for (size_t i = 0; i < N; i++)
	{
		for (size_t j = 0; j < N + 1; j++) cout << matrix[i][j] << ' ';
		cout << endl;
	}
	cout << "--------------" << endl;
}
float* Solve(float** matrix, unsigned int N) {
	float* P = new float[N]; float* Q = new float[N]; float* X = new float[N + 1];
float a, b, c, d;
	for (size_t i = 0; i < N; i++) {
		a = matrix[i][i - 1];
		b = matrix[i][i];
		c = matrix[i][i + 1];
		d = matrix[i][N];
		if (i == 0) {
			P[i] = -c / b;
			Q[i] = d / b;
		}
		else if (i == N - 1) {
			P[i] = 0;
			Q[i] = (d - a * Q[i - 1]) / (b + a * P[i - 1]);
		}
		else {
			P[i] = -c / (b + a * P[i - 1]);
			Q[i] = (d - a * Q[i - 1]) / (b + a * P[i - 1]);
		}
	}
	X[N - 1] = Q[N - 1]; for (int i = N - 2; i > -1; i--)
		X[i] = P[i] * X[i + 1] + Q[i];
	for (int i = N; i > 0; i--) X[i] = X[i - 1];
	X[0] = 0;
	return X;
}
void fillCoefficients(float* X, float* Y, float*& A, float*& B, float*& C, float*& D)
{
	for (size_t i = 0; i < n - 1; i++) {
		A[i] = Y[i];
		D[i] = (C[i + 1] - C[i]) / (3 * h);
		B[i] = (Y[i + 1] - Y[i]) / h - (C[i + 1] + 2 * C[i]) * h / 3;
	}
	A[n - 1] = Y[n - 1];
	B[n - 1] = (Y[n] - Y[n - 1]) / h - 2 * C[n - 1] * h / 3; D[n - 1] = -C[n - 1] / (3 *
h);
}
```

```
int main() {
	float** Matrix;
	float* X = new float[n]; float* Y = new float[n]; float* A = new float[n]; float*
B = new float[n]; float* C = new float[n]; float* D = new float[n]; unsigned int N;
	fillFunction(X, Y); fillMatrix(Matrix, N, Y); print(Matrix, N);
	C = Solve(Matrix, N); fillCoefficients(X, Y, A, B, C, D); cout << endl;
	for (size_t i = 0; i < n; i++)
		cout << "i = " << i << " A[i] = " << A[i] << " B[i] = " << B[i] << " C[i]
= "
		<< C[i] << " D[i] = " << D[i] << endl; return 0;
}
```

```
-------------
4 1 0 0 7.98537
1 4 1 0 0.69523
0 1 4 1 -9.67371
0 0 1 4 6.04398
-------------

i = 0 A[i] = 0.496501 B[i] = -2.5033 C[i] = 0 D[i] = 0.625221
i = 1 A[i] = -1.38158 B[i] = -0.131141 C[i] = 1.87566 D[i] = -0.464314
i = 2 A[i] = -0.101375 B[i] = 2.22724 C[i] = 0.48272 D[i] = -1.19801
i = 3 A[i] = 1.41058 B[i] = -0.401351 C[i] = -3.11131 D[i] = 1.80005
i = 4 A[i] = -0.302042 B[i] = -2.54803 C[i] = 2.28882 D[i] = -0.762941
```

Рис.1. Коэффициенты исходной матрицы и коэффициенты для полиномов сплайнов

3. Вычисление погрешности сплайн-интерполяции.

Для решения этой задачи воспользуемся программой desmos. Построим в ней график исходной функции и получившегося сплайна. Результат на рисунке 2. Здесь коэффициенты полиномов округлялись до третьего знака после запятой.

Из рисунка 2 видно, что наибольшая разность между графиками в середине и ближе к концам. Измерим в этих местах разность функций. Наибольшей оказалась разность в точке x = 0, она равна 0.0122.
Ссылка на график - https://www.desmos.com/calculator/vwwtbpdplj

Значение x	Значение y	Значение на полиноме	Погрешность
-2	-0,6760475728	-0,676048	-0,0000004272
-1,2	-1,3514455617	-1,3514412800	0,0000042817
-0,4	0,1407409337	0,1407365600	-0,0000043737
0,4	1,4	1,4071947600	0,00719476
1,2	0,4166472055	0,4166462400	-0,0000009655
2	-1,2421592810	-1,2421652240	-0,000005943

Среднеквадратичная погрешность	0,0000517646
Максимальная погрешность	0,00719476

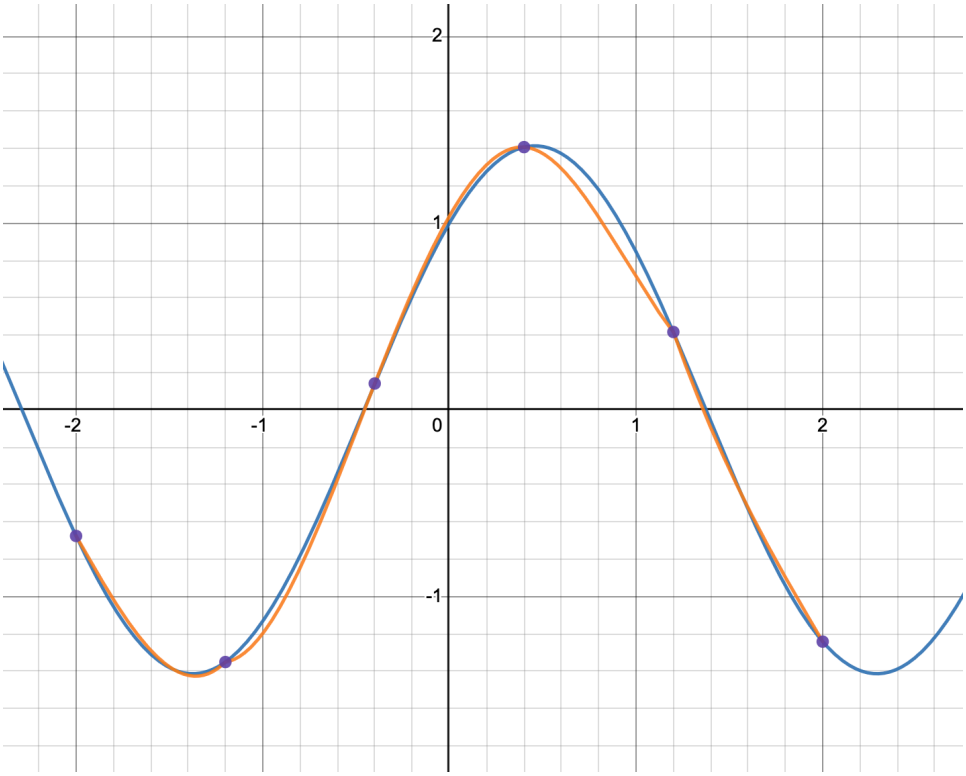
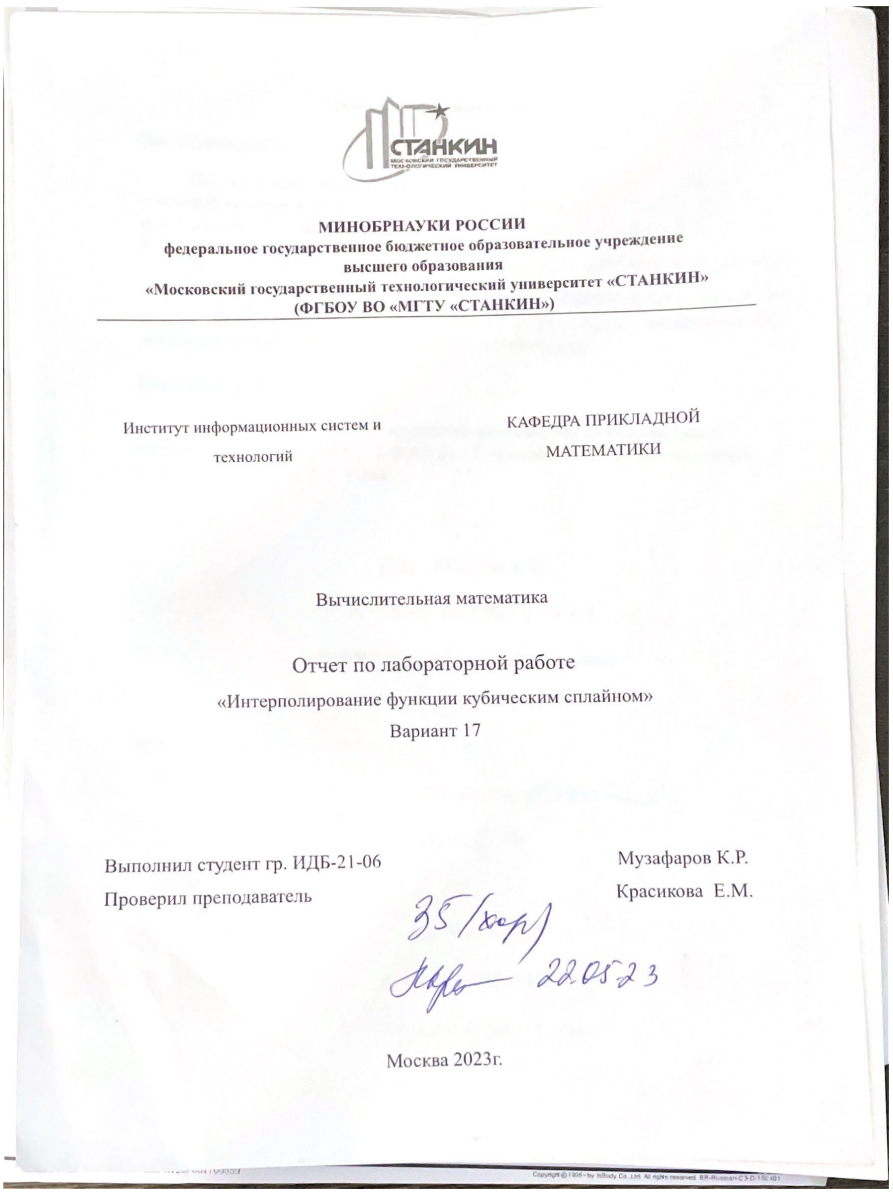


Рис.2. График функции и сплайна в desmos.

ВЫВОДЫ

В ходе данной лабораторной работы был изучен метод интерполирования кубическим сплайном, метод решения СЛАУ матричной прогонкой, написаны программы, задающие таблично исходную функцию, решающие получившуюся СЛАУ методом матричной прогонки, определяющие коэффициенты полиномов сплайна и строящие графики исходной функции и получившегося сплайна.



МИНОБРНАУКИ РОССИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технологический университет «СТАНКИН»
(ФГБОУ ВО «МГТУ «СТАНКИН»)

Институт информационных систем и технологий

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ

Вычислительная математика

Отчет по лабораторной работе
«Интерполирование функции кубическим сплайном»
Вариант 17

Выполнил студент гр. ИДБ-21-06 Музафаров К.Р.
Проверил преподаватель Красикова Е.М.

35/хор
22.05.23

Москва 2023г.