

**Московский авиационный институт  
(национальный исследовательский университет)**

**Институт №8 «Информационные технологии и прикладная  
математика»**

**Кафедра 806 «Вычислительная математика и программирование»**

**Лабораторные работы по курсу «Численные методы»**

Студент: Голошумов М.С.  
Преподаватель: Пивоваров Д.Е.  
Группа: М8О-303Б-21  
Дата:  
Оценка:  
Подпись:

**Москва, 2024**

## 3.1

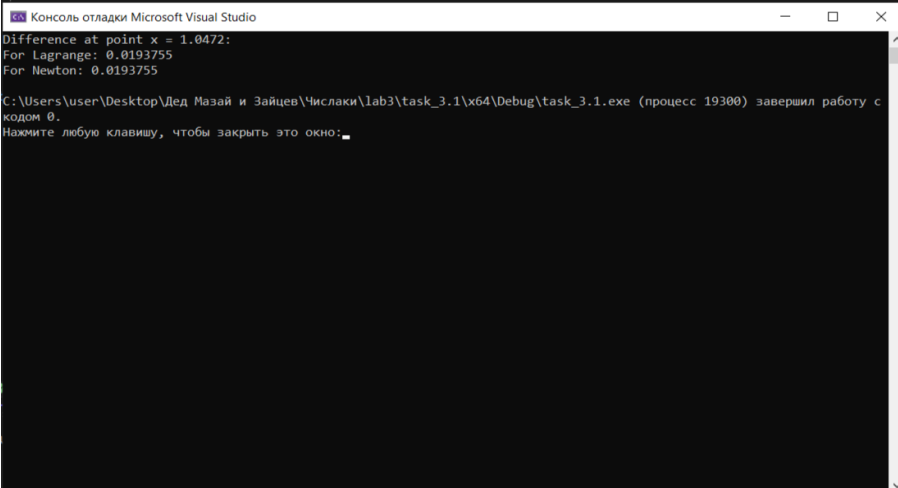
### 1 Постановка задачи

Используя таблицу значений  $Y_i$  функции  $y = f(x)$ , вычисленных в точках  $X_i, i = 0, \dots, 3$  построить интерполяционные многочлены Лагранжа и Ньютона, проходящие через точки  $\{X_i, Y_i\}$ . Вычислить значение погрешности интерполяции в точке  $X^*$ .

**Вариант: 4**

$y = \operatorname{ctg}(x), a) X_i = \pi/8, 2\pi/8, 3\pi/8, 4\pi/8; ) X_i = \pi/8, 5\pi/16, 3\pi/8, \pi/2; X^* = \pi/3$

### 2 Результаты работы



```
Консоль отладки Microsoft Visual Studio
Difference at point x = 1.0472:
For Lagrange: 0.0193755
For Newton: 0.0193755

C:\Users\user\Desktop\Дед Мазай и Зайцев\Числаки\lab3\task_3.1\x64\Debug\task_3.1.exe (процесс 19300) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно: _
```

Рис. 1: Вывод программы в консоли

### 3 Исходный код

```
1 #include <iostream>
2 #include <vector>
3 #include <cmath>
4 #include <algorithm>
5 #include <numeric>
6
7 double f(double x) {
8     return 1/tan(x);
9 }
10
11 double pi = 2 * acos(0.0);
12
13 double Lagrange(double x, std::vector<double> X_i, std::vector<double> f_i) {
14     int n = X_i.size();
15     double sum = 0;
16     for (int i = 0; i < n; i++) {
17         double composition = 1;
18         for (int j = 0; j < n; j++) {
19             if (i != j) {
20                 composition *= (x - X_i[j]) / (X_i[i] - X_i[j]);
21             }
22         }
23         sum += f_i[i] * composition;
24     }
25     return sum;
26 }
27
28 double Newton(double x, std::vector<double> X_i, std::vector<double> f_i) {
29     int n = X_i.size();
30     std::vector<std::vector<double>> differences;
31
32     auto DividedDifferences = [&]() {
33         differences = { f_i };
34         for (int i = 1; i < n; i++) {
35             std::vector<double> temp;
36             for (int k = 0; k < n - i; k++) {
37                 temp.push_back((differences[i - 1][k] - differences[i - 1][k + 1]) / (
38                     X_i[k] - X_i[k + i]));
39             }
40             differences.push_back(temp);
41         }
42     };
43     DividedDifferences();
44
45     double sum = f_i[0];
46     for (int i = 1; i < n; i++) {
```

```

47     double composition = 1;
48     for (int j = 0; j < i; j++) {
49         composition *= x - X_i[j];
50     }
51     sum += composition * differences[i][0];
52 }
53 return sum;
54 }
55
56 int main() {
57     std::vector<double> X1 = { pi/8, 2* pi / 8, 3* pi / 8, 4* pi / 8 };
58     std::vector<double> f1;
59     for (auto x : X1) {
60         f1.push_back(f(x));
61     }
62
63     std::vector<double> X2 = { pi / 8, 5*pi / 16, 3*pi / 8, pi/2 };
64     std::vector<double> f2;
65     for (auto x : X2) {
66         f2.push_back(f(x));
67     }
68
69     double X_variation = pi/3;
70     std::cout << "Difference at point x = " << X_variation << ":\n"
71         << "For Lagrange: " << std::abs(f(X_variation) - Lagrange(X_variation, X1, f1))
72         << "\n"
73         << "For Newton: " << std::abs(f(X_variation) - Newton(X_variation, X1, f1)) <<
74         "\n";
75     return 0;
76 }

```

## 3.2

### 4 Постановка задачи

Построить кубический сплайн для функции, заданной в узлах интерполяции, предполагая, что сплайн имеет нулевую кривизну при  $x = x_0$  и  $x = x_4$ . Вычислить значение функции в точке  $x = X^*$ .

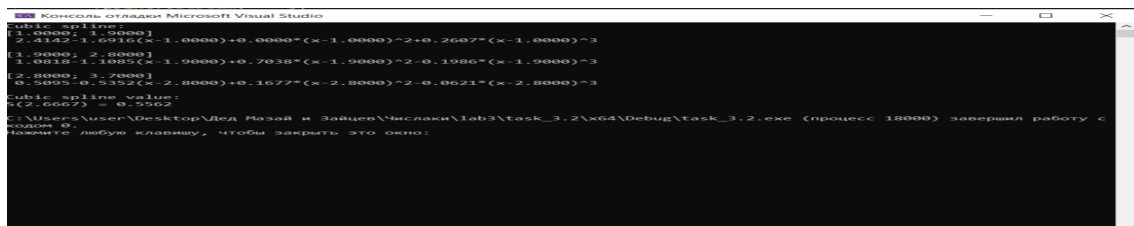
Вариант: 4

$$4. X^* = 2.66666667$$

$i$	0	1	2	3	4
$x_i$	1.0	1.9	2.8	3.7	4.6
$f_i$	2.4142	1.0818	0.50953	.11836	-0.24008

Рис. 2: Условие

### 5 Результаты работы



```
Консоль отладки Microsoft Visual Studio
[1.00000, 1.90000]
2.4142-1.0818*(x-1.0000)+0.0000*(x-1.0000)^2+0.2607*(x-1.0000)^3
[1.90000, 2.80000]
1.0818-1.1085*(x-1.9000)+0.7038*(x-1.9000)^2-0.1986*(x-1.9000)^3
[2.80000, 3.70000]
0.50953-0.5552*(x-2.8000)+0.1677*(x-2.8000)^2-0.0621*(x-2.8000)^3
Cubic spline value:
f(2.6667) = 0.5095
C:\Users\User\Desktop\Дед Мороз и зайцев\Числаки\lab3\task_3.2\x64\Debug\task_3.2.exe (процесс 18000) завершила работу с
кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рис. 3: Вывод программы в консоли

## 6 Исходный код

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include "read.h"
5
6  int DEBUG = 0;
7
8  Matrix* create_cubic_spline(Matrix* matrix) {
9      Matrix* spline, * tridiagonal_matrix, * vector, * solve;
10     int i;
11     if (!matrix || matrix->width != 2) {
12         fprintf(stderr, "Invalid size of matrix\n");
13     }
14     spline = create_matrix();
15     resize_matrix(spline, matrix->height, 5);
16
17     tridiagonal_matrix = create_matrix();
18     resize_matrix(tridiagonal_matrix, matrix->height - 2, 3);
19
20     vector = create_matrix();
21     resize_matrix(vector, matrix->height - 2, 1);
22
23     for (i = 0; i < vector->height; i++) {
24         tridiagonal_matrix->data[i][0] = (i == vector->height - 1 ? 0 : matrix->data[i
25             + 1][0] - matrix->data[i][0]);
26         tridiagonal_matrix->data[i][1] = 2 * (matrix->data[i + 2][0] - matrix->data[i
27             ] [0]);
28         tridiagonal_matrix->data[i][2] = (i == 0 ? 0 : matrix->data[i + 2][0] - matrix
29             ->data[i + 1][0]);
30         vector->data[i][0] = 3 * ((matrix->data[i + 2][1] - matrix->data[i + 1][1]) / (
31             matrix->data[i + 2][0] - matrix->data[i + 1][0]) -
32             (matrix->data[i + 1][1] - matrix->data[i][1]) / (matrix->data[i + 1][0] -
33             matrix->data[i][0]));
34     }
35     if (DEBUG) {
36         fprintf(stderr, "Tridiagonal matrix:\n");
37         print_matrix(tridiagonal_matrix, stderr);
38         fprintf(stderr, "Vector of right part:\n");
39         print_matrix(vector, stderr);
40     }
41     solve = TDMA(tridiagonal_matrix, vector);
42     if (!solve) {
43         fprintf(stderr, "Singular matrix\n");
44         return NULL;
45     }
46     remove_matrix(tridiagonal_matrix);
47     remove_matrix(vector);
```

```

43     free(tridiagonal_matrix);
44     free(vector);
45
46     spline->data[0][3] = 0;
47     for (i = 0; i < spline->height - 1; i++) {
48         spline->data[i][0] = matrix->data[i][0];
49         spline->data[i][1] = matrix->data[i][1];
50         if (i < spline->height - 2)
51             spline->data[i + 1][3] = solve->data[i][0];
52         spline->data[i][2] = (matrix->data[i + 1][1] - matrix->data[i][1]) / (matrix->
            data[i + 1][0] - matrix->data[i][0]) -
53             1.0 / 3.0 * (matrix->data[i + 1][0] - matrix->data[i][0]) * (2 * spline->
                data[i][3] + (i < spline->height - 2 ? spline->data[i + 1][3] : 0));
54         spline->data[i][4] = ((i < spline->height - 2 ? spline->data[i + 1][3] : 0) -
            spline->data[i][3]) /
55             (3 * (matrix->data[i + 1][0] - matrix->data[i][0]));
56     }
57
58     spline->data[spline->height - 1][0] = matrix->data[spline->height - 1][0];
59
60     if (DEBUG) {
61         fprintf(stderr, "Spline matrix\n");
62         print_matrix(spline, stderr);
63     }
64     remove_matrix(solve);
65     free(solve);
66     return spline;
67 }
68 double cubic_spline(Matrix* spline, double x) {
69     int i;
70     for (i = 0; i < spline->height - 1; i++)
71         if (spline->data[i][0] <= x && x <= spline->data[i + 1][0])
72             return spline->data[i][1] + spline->data[i][2] * (x - spline->data[i][0]) +
73                 spline->data[i][3] * (x - spline->data[i][0]) * (x - spline->data[i][0]) +
74                 spline->data[i][4] * (x - spline->data[i][0]) * (x - spline->data[i][0]) *
75                     (x - spline->data[i][0]);
76     if (DEBUG)
77         fprintf(stderr, "Incorrect value of argument\n");
78     return 0;
79 }
80 void print_cubic_spline(Matrix* spline, FILE* stream) {
81     int i;
82     for (i = 0; i < spline->height - 1; i++) {
83         fprintf(stream, "[% .4f; % .4f]\n", spline->data[i][0], spline->data[i + 1][0]);
84         if (spline->data[i][1] >= 0)
85             fprintf(stream, " ");
86         fprintf(stream, "% .4f", spline->data[i][1]);
87         if (spline->data[i][2] >= 0)
88             fprintf(stream, "+");

```

```

89     fprintf(stream, "%.4f(x", spline->data[i][2]);
90     if (spline->data[i][0] < 0)
91         fprintf(stream, "+");
92     fprintf(stream, "%.4f)", -spline->data[i][0]);
93     if (spline->data[i][3] >= 0)
94         fprintf(stream, "+");
95     fprintf(stream, "%.4f*(x", spline->data[i][3]);
96     if (spline->data[i][0] < 0)
97         fprintf(stream, "+");
98     fprintf(stream, "%.4f)^2", -spline->data[i][0]);
99     if (spline->data[i][4] >= 0)
100         fprintf(stream, "+");
101     fprintf(stream, "%.4f*(x", spline->data[i][4]);
102     if (spline->data[i][0] < 0)
103         fprintf(stream, "+");
104     fprintf(stream, "%.4f)^3\n\n", -spline->data[i][0]);
105 }
106 }
107
108 int main(void) {
109     float x = 2.66666667;
110     Matrix* result, * matrix;
111     FILE* fmatrix;
112
113     fmatrix = fopen("task_3.2matrix.txt", "r");
114     if (!fmatrix) {
115         fprintf(stderr, "Incorrect name of file\n");
116         return 0;
117     }
118
119     matrix = create_matrix();
120     scan_matrix(matrix, fmatrix);
121     fclose(fmatrix);
122
123     if (result = create_cubic_spline(matrix)) {
124         printf("Cubic spline:\n");
125         print_cubic_spline(result, stdout);
126         printf("Cubic spline value:\nS(%.4f) = %.4f\n", x, cubic_spline(result, x));
127         remove_matrix(result);
128         free(result);
129     }
130     remove_matrix(matrix);
131     free(matrix);
132     return 0;
133 }

1 #include "read.h"
2 #include <stdlib.h>
3
4 Matrix* create_matrix(void) {

```



```

5   Matrix* new_matrix = (Matrix*)malloc(sizeof(Matrix));
6   new_matrix->width = new_matrix->height = 0;
7   new_matrix->data = NULL;
8   return new_matrix;
9 }
10 void remove_matrix(Matrix* matrix) {
11     int i;
12     if (!matrix)
13         return;
14     for (i = 0; i < matrix->height; i++)
15         free(matrix->data[i]);
16     free(matrix->data);
17     matrix->data = NULL;
18     matrix->height = matrix->width = 0;
19 }
20 void resize_matrix(Matrix* matrix, const int height, const int width) {
21     if (height > 0) {
22         matrix->data = (double**)realloc(matrix->data, sizeof(double*) * height);
23         for (int i = matrix->height; i < height; i++) {
24             matrix->data[i] = (double*)malloc(sizeof(double) * (width <= 0 ? matrix->
                width : width));
25             for (int j = 0; j < width; j++)
26                 matrix->data[i][j] = 0;
27         }
28     }
29     if (width > 0)
30         for (int i = 0; i < matrix->height; i++) {
31             matrix->data[i] = (double*)realloc(matrix->data[i], sizeof(double) * width)
                ;
32             for (int j = matrix->width; j < width; j++)
33                 matrix->data[i][j] = 0;
34         }
35     if (width > 0)
36         matrix->width = width;
37     if (height > 0)
38         matrix->height = height;
39 }
40 void print_matrix(Matrix* matrix, FILE* stream) {
41     int i, j;
42     if (!matrix->data)
43         return;
44     for (i = 0; i < matrix->height; i++) {
45         fputc('[', stream);
46         for (j = 0; j < matrix->width; j++)
47             fprintf(stream, "%.5f ", matrix->data[i][j]);
48         fprintf(stream, "\b\b]\n");
49     }
50     fprintf(stream, "size: %d x %d\n", matrix->height, matrix->width);
51 }

```

```

52 void scan_matrix(Matrix* matrix, FILE* stream) {
53     int i, j, c = 0;
54     float a;
55     for (i = 0; c != EOF; i++) {
56         resize_matrix(matrix, i + 1, -1);
57         c = 0;
58         for (j = 0; c != '\n'; j++) {
59             if (!i)
60                 resize_matrix(matrix, -1, j + 1);
61             fscanf_s(stream, "%f", &a);
62             matrix->data[i][j] = a;
63             c = getc(stream);
64             if (c == EOF) {
65                 resize_matrix(matrix, i, -1);
66                 return;
67             }
68         }
69     }
70 }
71
72 static inline float absolute(float a) {
73     return a > 0 ? a : -a;
74 }
75
76 Matrix* tridiagonal_matrix_algorithm(Matrix* matrix, Matrix* vector) {
77     Matrix* PQ, * result;
78     int i;
79     if (!matrix || !vector || matrix->width != 3 || matrix->height != vector->height ||
80         vector->width != 1)
81         return NULL;
82     PQ = create_matrix();
83     resize_matrix(PQ, matrix->height - 1, 2);
84     result = create_matrix();
85     resize_matrix(result, matrix->height, 1);
86     PQ->data[0][0] = -matrix->data[0][2] / matrix->data[0][1];
87     PQ->data[0][1] = vector->data[0][0] / matrix->data[0][1];
88     if (absolute(matrix->data[0][1]) < absolute(matrix->data[0][2]) ||
89         absolute(matrix->data[matrix->height - 1][1]) < absolute(matrix->data[matrix->
90             height - 1][2]))
91         return NULL;
92     for (i = 1; i < PQ->height; i++) {
93         if (absolute(matrix->data[i][1]) < absolute(matrix->data[i][0]) + absolute(
94             matrix->data[i][2]))
95             return NULL;
96         double temp = matrix->data[i][0] * PQ->data[i - 1][0] + matrix->data[i][1];
97         PQ->data[i][0] = -matrix->data[i][2] / temp;
98         PQ->data[i][1] = (vector->data[i][0] - matrix->data[i][0] * PQ->data[i - 1][1])
99             / temp;
100     }

```

```

97 |     i = result->height - 1;
98 |     result->data[i][0] = (vector->data[i][0] - matrix->data[i][0] * PQ->data[i - 1][1])
    |     /
99 |     (matrix->data[i][0] * PQ->data[i - 1][0] + matrix->data[i][1]);
100 |     for (i = result->height - 2; i >= 0; i--)
101 |         result->data[i][0] = PQ->data[i][0] * result->data[i + 1][0] + PQ->data[i][1];
102 |     remove_matrix(PQ);
103 |     free(PQ);
104 |     return result;
105 | }
106 | Matrix* (* const TDMA)(Matrix*, Matrix*) = tridiagonal_matrix_algorithm;

1 | #ifndef _LAB3_
2 | #define _LAB3_
3 |
4 | #include <stdio.h>
5 |
6 | typedef struct Matrix {
7 |     double** data;
8 |     unsigned int width;
9 |     unsigned int height;
10 | } Matrix;
11 |
12 | Matrix* create_matrix(void);
13 | void remove_matrix(Matrix*);
14 | void resize_matrix(Matrix*, const int, const int);
15 | void print_matrix(Matrix*, FILE*);
16 | void scan_matrix(Matrix*, FILE*);
17 | Matrix* (* const TDMA)(Matrix*, Matrix*);
18 |
19 | #endif /* _LAB3_ */

```

Условия:

```

1 | 1.0 2.4142
2 | 1.9 1.0818
3 | 2.8 0.50953
4 | 3.7 0.11836
5 | 4.6 -0.24008

```

### 3.3

## 7 Постановка задачи

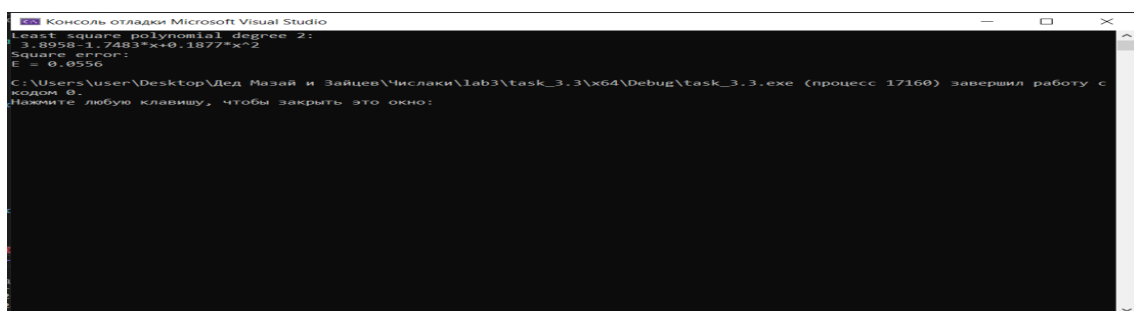
Для таблично заданной функции путем решения нормальной системы МНК найти приближающие многочлены а) 1-ой и б) 2-ой степени. Для каждого из приближающих многочленов вычислить сумму квадратов ошибок. Построить графики приближаемой функции и приближающих многочленов.

**Вариант: 4**

$i$	0	1	2	3	4	5
$x_i$	1.0	1.9	2.8	3.7	4.6	5.5
$y_i$	2.4142	1.0818	0.50953	0.11836	-0.24008	-0.66818

Рис. 4: Условия

## 8 Результаты работы



```
Консоль отладки Microsoft Visual Studio
Least square polynomial degree 2:
3.8958-1.7483*x+0.1877*x^2
Square error:
E = 0.0556
C:\Users\user\Desktop\Дед Мазай и Зайцев\Числаки\lab3\task_3.3\x64\Debug\task_3.3.exe (процесс 17160) завершил работу с
кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рис. 5: Вывод программы в консоли

## 9 Исходный код

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <math.h>
5  #include "read.h"
6
7  int FIRST = 0;
8  int SECOND = 1;
9
10 double square_error(Matrix* least_squares, Matrix* vector) {
11     double result = 0;
12     int i, j;
13     if (!least_squares || !vector || vector->width != 2 || least_squares->width != 1) {
14         fprintf(stderr, "Invalid size of matrix");
15         return 0;
16     }
17     for (i = 0; i < vector->height; i++) {
18         double temp = least_squares->data[0][0] - vector->data[i][1];
19         for (j = 1; j < least_squares->height; j++)
20             temp += least_squares->data[j][0] * pow(vector->data[i][0], j);
21         result += temp * temp;
22     }
23     return result;
24 }
25
26 Matrix* least_square_polynomial_degree_2(Matrix* vector) {
27     Matrix* result;
28     int i;
29     double a11 = vector->height, a12 = 0, a22 = 0, a23 = 0, a33 = 0, b1 = 0, b2 = 0, b3
        = 0;
30     double c11, c12, c22, d1, d2;
31     if (!vector || vector->width != 2) {
32         fprintf(stderr, "Invalid size of matrix");
33         return 0;
34     }
35     result = create_matrix();
36     resize_matrix(result, 3, 1);
37     for (i = 0; i < vector->height; i++) {
38         a12 += vector->data[i][0];
39         a22 += vector->data[i][0] * vector->data[i][0];
40         a23 += vector->data[i][0] * vector->data[i][0] * vector->data[i][0];
41         a33 += vector->data[i][0] * vector->data[i][0] * vector->data[i][0] * vector->
            data[i][0];
42         b1 += vector->data[i][1];
43         b2 += vector->data[i][1] * vector->data[i][0];
44         b3 += vector->data[i][1] * vector->data[i][0] * vector->data[i][0];
45     }
```

```

46     c11 = a22 * a11 - a12 * a12;
47     c12 = a23 * a11 - a12 * a22;
48     c22 = a11 * a33 - a22 * a22;
49     d1 = b2 * a11 - b1 * a12;
50     d2 = b3 * a11 - b1 * a22;
51     result->data[1][0] = (d1 * c22 - d2 * c12) / (c11 * c22 - c12 * c12);
52     result->data[2][0] = (d2 * c11 - d1 * c12) / (c11 * c22 - c12 * c12);
53     result->data[0][0] = (b1 - result->data[1][0] * a12 - result->data[2][0] * a22) /
        a11;
54     return result;
55 }
56
57 Matrix* least_square_polynomial_degree_1(Matrix* vector) {
58     Matrix* result;
59     int i;
60     double a11 = vector->height, a12 = 0, a22 = 0, b1 = 0, b2 = 0;
61     if (!vector || vector->width != 2) {
62         fprintf(stderr, "Invalid size of matrix");
63         return 0;
64     }
65     result = create_matrix();
66     resize_matrix(result, 2, 1);
67     for (i = 0; i < vector->height; i++) {
68         a12 += vector->data[i][0];
69         a22 += vector->data[i][0] * vector->data[i][0];
70         b1 += vector->data[i][1];
71         b2 += vector->data[i][0] * vector->data[i][1];
72     }
73     result->data[0][0] = (b1 * a22 - b2 * a12) / (a11 * a22 - a12 * a12);
74     result->data[1][0] = (b2 * a11 - b1 * a12) / (a11 * a22 - a12 * a12);
75     return result;
76 }
77
78 void print_least_square(Matrix* least_square, FILE* stream) {
79     int i;
80     if (!least_square || least_square->width != 1) {
81         fprintf(stderr, "Invalid size of matrix");
82         return;
83     }
84     if (least_square->data[0][0] >= 0)
85         fputc(' ', stream);
86     fprintf(stream, "%.4f", least_square->data[0][0]);
87     if (least_square->data[1][0] >= 0)
88         fputc('+', stream);
89     fprintf(stream, "%.4f*x", least_square->data[1][0]);
90     for (i = 2; i < least_square->height; i++) {
91         if (least_square->data[i][0] >= 0)
92             fputc('+', stream);
93         fprintf(stream, "%.4f*x^%d", least_square->data[i][0], i);

```

```

94     }
95     fputc('\n', stream);
96 }
97
98 int main(void) {
99     int i;
100     Matrix* result, * vector;
101     FILE* fvector;
102
103     fvector = fopen("task_3.3matrix.txt", "r");
104     if (!fvector) {
105         fprintf(stderr, "Incorrect name of file\n");
106         return 0;
107     }
108     vector = create_matrix();
109     scan_matrix(vector, fvector);
110     fclose(fvector);
111
112     if (FIRST && (result = least_square_polynomial_degree_1(vector))) {
113         printf("Least square polynomial degree 1:\n");
114         print_least_square(result, stdout);
115         printf("Square error:\nE = %.4f\n", square_error(result, vector));
116         remove_matrix(result);
117         free(result);
118     }
119     else if (SECOND && (result = least_square_polynomial_degree_2(vector))) {
120         printf("Least square polynomial degree 2:\n");
121         print_least_square(result, stdout);
122         printf("Square error:\nE = %.4f\n", square_error(result, vector));
123         remove_matrix(result);
124         free(result);
125     }
126     remove_matrix(vector);
127     free(vector);
128     return 0;
129 }

```

Условия:

1	1.0	2.4142
2	1.9	1.0818
3	2.8	0.50953
4	3.7	0.11836
5	4.6	-0.24008
6	5.5	-0.66818

## 3.4

### 10 Постановка задачи

Вычислить первую и вторую производную от таблично заданной функции  $y_i = f(x_i)$ ,  $i = 0, 1, 2, 3, 4$  в точке  $x = X_i$ .

Вариант: 4

$$X^* = 0.2$$

$\bar{x}$	0	1	2	3	4
$x_i$	0.0	0.1	0.2	0.3	0.4
$y_i$	1.0	1.1052	1.2214	1.3499	1.4918

Рис. 6: Условия

### 11 Результаты работы

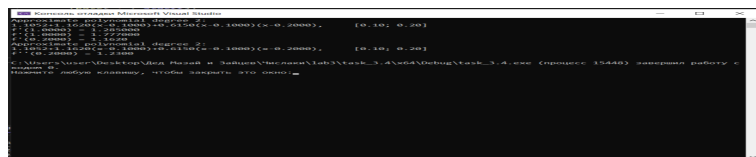


Рис. 7: Вывод программы в консоли



## 12 Исходный код

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include "read.h"
5
6 int DEBUG = 1;
7
8 double first_derivative(Matrix* vector, double x) {
9     int i;
10    double b, c, b_1;
11    if (!vector || vector->width != 2) {
12        fprintf(stderr, "Invalid size of matrix\n");
13        return 0;
14    }
15    for (i = 1; i < vector->height; i++)
16        if (vector->data[i - 1][0] <= x && x <= vector->data[i][0])
17            break;
18    if (i > vector->height - 2) {
19        fprintf(stderr, "Too few information about the function\n");
20        return 0;
21    }
22
23    b = (vector->data[i][1] - vector->data[i - 1][1]) / (vector->data[i][0] - vector->
24        data[i - 1][0]);
25    c = ((vector->data[i + 1][1] - vector->data[i][1]) / (vector->data[i + 1][0] -
26        vector->data[i][0]) - b) /
27        (vector->data[i + 1][0] - vector->data[i - 1][0]);
28    if (DEBUG) {
29        fprintf(stderr, "Approximate polynomial degree 2:\n%.4f", vector->data[i -
30            1][1]);
31        if (b >= 0)
32            fputc('+', stderr);
33        fprintf(stderr, "%.4f(x", b);
34        if (vector->data[i - 1][0] < 0)
35            fputc('+', stderr);
36        fprintf(stderr, "%.4f)", -vector->data[i - 1][0]);
37        if (c >= 0)
38            fputc('+', stderr);
39        fprintf(stderr, "%.4f(x", c);
40        if (vector->data[i - 1][0] < 0)
41            fputc('+', stderr);
42        fprintf(stderr, "%.4f)", -vector->data[i - 1][0]);
43        if (vector->data[i][0] < 0)
44            fputc('+', stderr);
45        fprintf(stderr, "%.4f)\t\t[%.2f; %.2f]\n",
46            -vector->data[i][0], vector->data[i - 1][0], vector->data[i][0]);
47    }
```

```

45     b_1 = (vector->data[i + 1][1] - vector->data[i][1]) / (vector->data[i + 1][0] -
vector->data[i][0]);
46     printf("f'(1.0000) = %f\n", b_1);
47     printf("f'(1.0000) = %f\n", c + b);
48     return b;
49 }
50 double second_derivative(Matrix* vector, double x) {
51     int i;
52     double b, c;
53     if (!vector || vector->width != 2) {
54         fprintf(stderr, "Invalid size of matrix\n");
55         return 0;
56     }
57     for (i = 1; i < vector->height; i++)
58         if (vector->data[i - 1][0] <= x && x <= vector->data[i][0])
59             break;
60     if (i > vector->height - 2) {
61         fprintf(stderr, "Too few information about the function\n");
62         return 0;
63     }
64     b = (vector->data[i][1] - vector->data[i - 1][1]) / (vector->data[i][0] - vector->
data[i - 1][0]);
65     c = ((vector->data[i + 1][1] - vector->data[i][1]) / (vector->data[i + 1][0] -
vector->data[i][0]) - b) /
        (vector->data[i + 1][0] - vector->data[i - 1][0]);
67     if (DEBUG) {
68         fprintf(stderr, "Approximate polynomial degree 2:\n%.4f", vector->data[i -
1][1]);
69         if (b >= 0)
70             fputc('+', stderr);
71         fprintf(stderr, "%.4f(x", b);
72         if (vector->data[i - 1][0] < 0)
73             fputc('+', stderr);
74         fprintf(stderr, "%.4f)", -vector->data[i - 1][0]);
75         if (c >= 0)
76             fputc('+', stderr);
77         fprintf(stderr, "%.4f(x", c);
78         if (vector->data[i - 1][0] < 0)
79             fputc('+', stderr);
80         fprintf(stderr, "%.4f)(x", -vector->data[i - 1][0]);
81         if (vector->data[i][0] < 0)
82             fputc('+', stderr);
83         fprintf(stderr, "%.4f),\t[%.2f; %.2f]\n",
84             -vector->data[i][0], vector->data[i - 1][0], vector->data[i][0]);
85     }
86     return 2 * c;
87 }
88
89 int main(void) {

```

```

90 | int i;
91 | Matrix* vector;
92 | float x = 0.2;
93 | FILE* fvector;
94 |
95 | fvector = fopen("task_3.4matrix.txt", "r");
96 | if (!fvector) {
97 |     fprintf(stderr, "Incorrect name of file\n");
98 |     return 0;
99 | }
100 | vector = create_matrix();
101 | scan_matrix(vector, fvector);
102 | fclose(fvector);
103 |
104 | printf("f' (%.4f) = %.4f\n", x, first_derivative(vector, x));
105 | printf("f'' (%.4f) = %.4f\n", x, second_derivative(vector, x));
106 | remove_matrix(vector);
107 | free(vector);
108 | return 0;
109 | }

```

Условия:

```

1 | 0.0 1.0
2 | 0.1 1.1052
3 | 0.2 1.2214
4 | 0.3 1.3499
5 | 0.4 1.4918

```

## 3.5

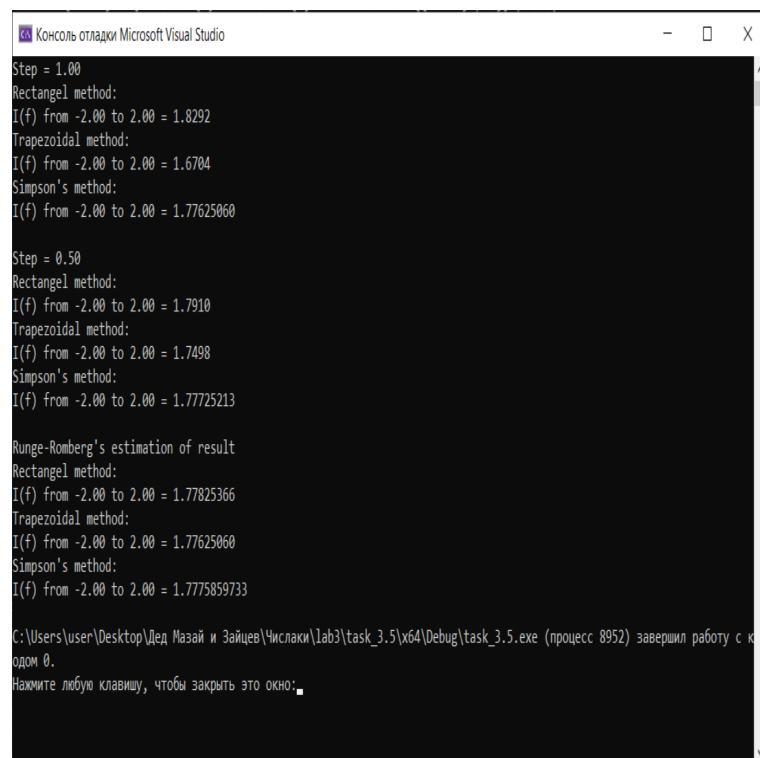
### 13 Постановка задачи

Вычислить определенный интеграл  $\int_{X_0}^{X_1} y dx$ , методами прямоугольников, трапеций, Симпсона с шагами  $h_1, h_2$ . Оценить погрешность вычислений, используя Метод Рунге-Ромберга:

**Вариант: 4**

$$y = \frac{3x+4}{2x+7}$$
$$X_0 = -2, X_k = 2, h_1 = 1.0, h_2 = 0.5$$

### 14 Результаты работы



```
Консоль отладки Microsoft Visual Studio
Step = 1.00
Rectangel method:
I(f) from -2.00 to 2.00 = 1.8292
Trapezoidal method:
I(f) from -2.00 to 2.00 = 1.6704
Simpson's method:
I(f) from -2.00 to 2.00 = 1.77625060

Step = 0.50
Rectangel method:
I(f) from -2.00 to 2.00 = 1.7910
Trapezoidal method:
I(f) from -2.00 to 2.00 = 1.7498
Simpson's method:
I(f) from -2.00 to 2.00 = 1.77725213

Runge-Romberg's estimation of result
Rectangel method:
I(f) from -2.00 to 2.00 = 1.77825366
Trapezoidal method:
I(f) from -2.00 to 2.00 = 1.77625060
Simpson's method:
I(f) from -2.00 to 2.00 = 1.7775859733

C:\Users\user\Desktop\Дед Мазай и Зайцев\Числаки\lab3\task_3.5\Debug\task_3.5.exe (процесс 8952) завершил работу с кодом 0.
Нажмите любую клавишу, чтобы закрыть это окно:
```

Рис. 8: Вывод программы в консоли

## 15 Исходный код

```
1  #include <stdio.h>
2  #include <string.h>
3
4  double Function(double x) {
5      return (3*x + 4) / (2 * x + 7);
6  }
7
8  double rectangle_method(double (*Function)(double), double a, double b, double step) {
9      double x, sum = 0;
10     if (a >= b) {
11         fprintf(stderr, "Incorrect limits of interval\n");
12         return 0;
13     }
14     for (x = a + step; x < b; x += step)
15         sum += step * Function(x - step / 2);
16     x -= step;
17     sum += (b - x) * Function((x + b) / 2);
18     return sum;
19 }
20
21 double trapezoidal_method(double (*Function)(double), double a, double b, double step)
22 {
23     double x, sum = 0;
24     if (a >= b) {
25         fprintf(stderr, "Incorrect limits of interval\n");
26         return 0;
27     }
28     for (x = a + step; x < b; x += step)
29         sum += 0.5 * step * (Function(x) + Function(x - step));
30     x -= step;
31     sum += 0.5 * (b - x) * (Function(x) + Function(b));
32     return sum;
33 }
34
35 double simpson_method(double (*Function)(double), double a, double b, double step) {
36     double x, sum = 0;
37     if (a >= b) {
38         fprintf(stderr, "Incorrect limits of interval\n");
39         return 0;
40     }
41     for (x = a + step; x < b; x += step)
42         sum += step * (Function(x) + Function(x - step) + 4 * Function(x - step / 2)) /
43         6.0;
44     x -= step;
45     sum += (b - x) * (Function(x) + Function(b) + 4 * Function((x + b) / 2)) / 6.0;
46     return sum;
47 }
```

```

46
47 double runge_romberg_method(double first_estimate, double first_step, double
    second_estimate, double second_step) {
48     double k = second_step / first_step;
49     if (first_step == second_step) {
50         fprintf(stderr, "Equal step of estimates\n");
51         return 0;
52     }
53     return first_estimate + (first_estimate - second_estimate) / (k * k - 1);
54 }
55
56 int main(int argc, char* argv[]) {
57     int i;
58     float left_limit = -2, right_limit = 2, step1 = 1.0, step2 = 0.5;
59     double rectangle1, rectangle2, trapezoidal1, trapezoidal2, simpsons1, simpsons2;
60
61     printf("Step = %.2f\nRectangel method:\n", step1);
62     printf("I(f) from %.2f to %.2f = %.4f\n",
63         left_limit, right_limit, rectangle1 = rectangle_method(Function, left_limit,
64             right_limit, step1));
65     printf("Trapezoidal method:\n");
66     printf("I(f) from %.2f to %.2f = %.4f\n",
67         left_limit, right_limit, trapezoidal1 = trapezoidal_method(Function, left_limit
68             , right_limit, step1));
69     printf("Simpson's method:\n");
70     printf("I(f) from %.2f to %.2f = %.8f\n",
71         left_limit, right_limit, simpsons1 = simpson_method(Function, left_limit,
72             right_limit, step1));
73
74     printf("\nStep = %.2f\nRectangel method:\n", step2);
75     printf("I(f) from %.2f to %.2f = %.4f\n",
76         left_limit, right_limit, rectangle2 = rectangle_method(Function, left_limit,
77             right_limit, step2));
78     printf("Trapezoidal method:\n");
79     printf("I(f) from %.2f to %.2f = %.4f\n",
80         left_limit, right_limit, trapezoidal2 = trapezoidal_method(Function, left_limit
81             , right_limit, step2));
82     printf("Simpson's method:\n");
83     printf("I(f) from %.2f to %.2f = %.8f\n",
84         left_limit, right_limit, simpsons2 = simpson_method(Function, left_limit,
85             right_limit, step2));
86
87     printf("\nRunge-Romberg's estimation of result\nRectangel method:\n");
88     printf("I(f) from %.2f to %.2f = %.8f\n",
89         left_limit, right_limit, runge_romberg_method(rectangle1, step1, rectangle2,
90             step2));
91     printf("Trapezoidal method:\n");
92     printf("I(f) from %.2f to %.2f = %.8f\n",

```

```

86 |         left_limit, right_limit, runge_romberg_method(trapezoidal1, step1, trapezoidal2
87 |             , step2));
87 |     printf("Simpson's method:\n");
88 |     printf("I(f) from %.2f to %.2f = %.10f\n",
89 |         left_limit, right_limit, runge_romberg_method(simpsons1, step1, simpsons2,
90 |             step2));
90 |     return 0;
91 | }

```