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 «Информатика, искусственный интеллект и системы управления»

КАФЕДРА «Теоретическая информатика и компьютерные технологии»

ОТЧЕТ

по лабораторной работе № 1

по курсу «Численные методы»

на тему: «Сплайн-интерполяция»

Вариант № 4

Студент ИУ9-61Б (Группа) _____ (Подпись, дата) Афанасьев И. (И. О. Фамилия)

Преподаватель _____ (Подпись, дата) Домрачева А. Б. (И. О. Фамилия)

2024 г.

1 Постановка задачи

Протабулировать функцию $f(x)$ на отрезке $[a, b]$ с шагом $h = \frac{b-a}{32}$ и распечатать таблицу (x_i, y_i), $i = 0, ..., n$. Для полученных узлов (x_i, y_i), $i = 0, ..., n$ построить кубический сплайн (распечатать массивы a, b, c, d). Вычислить значения $f(x)$ в точках $x_i^* = a + (i - \frac{1}{2})h$, $i = 1, ..., n$. Вычислить значения оригинальной функции в указанных точках и сравнить полученные результаты.

Индивидуальный вариант: функция $f(x) = 2x\cos(\frac{x}{2})$ на отрезке $[0, \pi]$.

2 Теоретический раздел

Пусть значения функции $y = f(x)$ известны в точках (x_i, y_i), $i = 0, ..., n$. *Интерполяционной* называется функция $y = \varphi(x)$, проходящая через заданные точки, называемые *узлами интерполяции*: $\varphi(x_i) = y_i = f(x_i)$. *Сплайном k-го порядка с дефектом* def называется функция, проходящая через все узлы (x_i, y_i), $i = 0, ..., n$, являющаяся многочленом k-й степени на каждом отрезке разбиения $[x_i, x_{i+1}]$ в отдельности и имеющая $(k - \text{def})$ непрерывных производных на отрезке $[x_0, x_n]$.

Для каждого отрезка разбиения отыскивается кубический сплайн в виде

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3, \quad x \in [x_i, x_{i+1}], \quad i = 0, ..., n-1.$$

На частные многочлены накладываются условия

$$S_i(x_i) = y_i, \quad i = 0, ..., n-1, \quad S_{n-1}(x_n) = y_n$$

(сплайн проходит через все узлы);

$$S_{i-1}(x_i) = S_i(x_i); \quad S'_{i-1}(x_i) = S'_i(x_i); \quad S''_{i-1}(x_i) = S''_i(x_i); \quad i = 0, ..., n-1$$

(непрерывность сплайна и его первых двух производных в промежуточных узлах);

$$S''_0(x_0) = 0; \quad S''_{n-1}(x_n) = 0$$

(краевые условия).

Эти условия приходят к трёхдиагональной СЛАУ относительно коэффициентов c_i:

$$c_{i-1} + 4c_i + c_{i+1} = \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}, \quad i = 1, ..., n-1, \quad c_0 = c'_n = 0,$$

где $h = x_{i+1} - x_i$, $i = 0, ..., n-1$ — постоянный шаг интерполирования. Система решается методом прогонки. Остальные коэффициенты выражаются через c_i по следующим формулам:

$$a_i = y_i, \quad i = 0, ..., n-1;$$

$$b_i = \frac{y_{i+1} - y_i}{h}\frac{h}{3}(c_{i+1} + 2c_i), \quad i = 0, ..., n-2; \quad b_{n-1} = \frac{y_n - y_{n-1}}{h}\frac{2}{3}hc_{n-1};$$

$$d_i = \frac{c_{i+1} - c_i}{3h}; \quad i = 0, ..., n-2; \quad d_{n-1} = -\frac{c_n}{3h}.$$

3 Практический раздел

В листинге 3.1 представлен исходный код программы на языке C++.

Листинг 3.1 – Исходный код программы на языке C++

```
#include <cassert>
#include <cmath>
#include <functional>
#include <iomanip>
#include <iostream>
#include <numbers>
#include <vector>

static constexpr std::size_t kWidth = 12;

namespace {

void PrintResult(const std::vector<double>& xs, const
    std::vector<double>& ys1,
                 const std::vector<double>& ys2,
                 const std::vector<double>& errors) {
  const auto n = xs.size();

  assert(n == ys1.size() && n == ys2.size() && n ==
      errors.size());

  std::cout << "Comparsion\n"
            << std::setw(kWidth) << "i" << std::setw(kWidth) <<
               "x_i"
            << std::setw(kWidth) << "actual y_i" <<
               std::setw(kWidth)
            << "spline y_i" << std::setw(kWidth) << "error" <<
               '\n';

  for (std::size_t i = 0; i < n; ++i) {
    std::cout << std::setw(kWidth) << i << std::setw(kWidth) <<
       xs[i]
              << std::setw(kWidth) << ys1[i] <<
                 std::setw(kWidth) << ys2[i]
              << std::setw(kWidth) << errors[i] << '\n';
  }
}
```

```
31
32 void PrintCoeffs(const std::vector<double>& as, const
      std::vector<double>& bs,
33                   const std::vector<double>& cs, const
                        std::vector<double>& ds) {
34   const auto n = as.size();
35
36   assert(n == bs.size() && n == cs.size() && n == ds.size());
37
38   std::cout << "Coefficient arrays a, b, c, d\n"
39             << std::setw(kWidth) << "i" << std::setw(kWidth) <<
                   "a_i"
40             << std::setw(kWidth) << "b_i" << std::setw(kWidth)
                   << "c_i"
41             << std::setw(kWidth) << "d_i" << '\n';
42
43   for (std::size_t i = 0; i < n; ++i) {
44     std::cout << std::setw(kWidth) << i << std::setw(kWidth) <<
          as[i]
45               << std::setw(kWidth) << bs[i] << std::setw(kWidth)
                     << cs[i]
46               << std::setw(kWidth) << ds[i] << '\n';
47   }
48 }
49
50 std::pair<std::vector<double>, std::vector<double>> Tabulate(
51     const std::function<double(const double)>& f, const
         std::size_t n,
52     const double a, const double h) {
53   std::vector<double> xs, ys;
54   xs.reserve(n + 1);
55   ys.reserve(n + 1);
56
57   xs.push_back(a);
58   ys.push_back(f(a));
59
60   for (std::size_t i = 1; i <= n; ++i) {
61     xs.push_back(xs[i - 1] + h);
62     ys.push_back(f(xs[i]));
63   }
64
```

```
65    return {std::move(xs), std::move(ys)};
66  }
67
68  std::vector<std::vector<double>> GetMatrix(const std::size_t n) {
69    assert(n >= 3);
70
71    std::vector<std::vector<double>> m(n - 1,
          std::vector<double>(n - 1));
72
73    m[0][0] = 4;
74    m[0][1] = 1;
75
76    for (std::size_t i = 1, end = n - 2; i < end; ++i) {
77      m[i][i - 1] = 1;
78      m[i][i] = 4;
79      m[i][i + 1] = 1;
80    }
81
82    m[n - 2][n - 3] = 1;
83    m[n - 2][n - 2] = 4;
84
85    return m;
86  }
87
88  std::vector<double> GetFreeCoeffs(const std::vector<double>& ys,
89                                    const std::size_t n, const
                                        std::size_t h) {
90    std::vector<double> fc;
91    fc.reserve(n - 1);
92
93    for (std::size_t i = 1; i < n; ++i) {
94      fc.push_back((ys[i + 1] - 2 * ys[i] + ys[i - 1]) / h / h);
95    }
96
97    return fc;
98  }
99
100 bool CheckDiagonalPredominanceConditions(
101     const std::vector<std::vector<double>>& matrix) {
102   const auto n = matrix.size();
103
```

```
104   if (std::abs(matrix[0][1] / matrix[0][0]) > 1) {
105     return false;
106   }
107
108   if (std::abs(matrix[n - 1][n - 2] / matrix[n - 1][n - 1]) > 1)
          {
109     return false;
110   }
111
112   for (auto i = 1; i < n; i++) {
113     if (std::abs(matrix[i][i]) <
114         std::abs(matrix[i][i - 1]) + std::abs(matrix[i][i + 1]))
              {
115       return false;
116     }
117   }
118
119   return true;
120 }
121
122 std::vector<double> CalculateSolution(
123     const std::vector<std::vector<double>>& matrix,
124     const std::vector<double>& free_coeffs) {
125   assert(CheckDiagonalPredominanceConditions(matrix));
126
127   const auto n = free_coeffs.size();
128
129   std::vector<double> alphas(n - 1);
130   std::vector<double> betas(n - 1);
131
132   assert(matrix[0][0] != 0);
133
134   alphas[0] = -matrix[0][1] / matrix[0][0];
135   betas[0] = free_coeffs[0] / matrix[0][0];
136
137   for (auto i = 1; i < n - 1; i++) {
138     alphas[i] =
139         -matrix[i][i + 1] / (matrix[i][i - 1] * alphas[i - 1] +
              matrix[i][i]);
140     betas[i] = (free_coeffs[i] - matrix[i][i - 1] * betas[i -
        1]) /
```

```
141                 (matrix[i][i - 1] * alphas[i - 1] + matrix[i][i]);
142   }
143
144   std::vector<double> result(n);
145   result[n - 1] = (free_coeffs[n - 1] - matrix[n - 1][n - 2] *
        betas[n - 2]) /
146                   (matrix[n - 1][n - 2] * alphas[n - 2] +
                        matrix[n - 1][n - 1]);
147
148   for (int i = n - 2; i >= 0; i--) {
149     result[i] = alphas[i] * result[i + 1] + betas[i];
150   }
151
152   return result;
153 }
154
155 }  // namespace
156
157 int main() {
158   constexpr std::size_t n = 32;
159
160   constexpr auto a = 0.0;
161   constexpr auto b = std::numbers::pi;
162   constexpr auto h = (b - a) / n;
163   const auto f = [](const double x) { return 2 * x * std::cos(x
        / 2); };
164
165   const auto [xs, ys] = Tabulate(f, n, a, h);
166
167   const auto m = GetMatrix(n);
168   const auto fc = GetFreeCoeffs(ys, n, h);
169   const auto s = CalculateSolution(m, fc);  // cs[1..n-1]
170
171   std::vector<double> as, bs, cs, ds;
172   as.reserve(n);
173   bs.reserve(n);
174   cs.reserve(n);
175   ds.reserve(n);
176
177   for (std::size_t i = 0; i < n; ++i) {
178     as.push_back(ys[i]);
```

```
179   }
180
181   cs.push_back(0);
182   for (std::size_t i = 0, end = n - 1; i < end; ++i) {
183     cs.push_back(ys[i]);
184   }
185
186   for (std::size_t i = 0, end = n - 1; i < end; ++i) {
187     bs.push_back((ys[i + 1] - ys[i]) / 3 * (cs[i + 1] + 2 *
          cs[i]));
188   }
189   bs.push_back((ys[n] - ys[n - 1]) * 2 * cs[n - 1] / 3);
190
191   for (std::size_t i = 0, end = n - 1; i < end; ++i) {
192     ds.push_back((cs[i + 1] - cs[i]) / 3 / h);
193   }
194   ds.push_back(-cs[n] / 3 / h);
195
196   PrintCoeffs(as, bs, cs, ds);
197
198   std::vector<double> test_xs, f_ys, spline_ys, errors;
199
200   test_xs.reserve(n);
201   f_ys.reserve(n);
202   spline_ys.reserve(n);
203   errors.reserve(n);
204
205   for (std::size_t i = 0; i < n; i++) {
206     test_xs.push_back(a + (i + 0.5) * h);
207     f_ys.push_back(f(test_xs[i]));
208
209     const auto x = test_xs[i] - xs[i];
210     spline_ys.push_back(as[i] + bs[i] * x + cs[i] * std::pow(x,
          2) +
211                         ds[i] * std::pow(x, 3));
212
213     errors.push_back(std::abs(f_ys[i] - spline_ys[i]));
214   }
215
216   PrintResult(test_xs, f_ys, spline_ys, errors);
217 }
```

4 Тестирование

В листинге 4.1 представлены результаты работы программы.

Листинг 4.1 – Результаты работы программы

```
Coefficient arrays a, b, c, d
           i          a_i          b_i          c_i          d_i
           0            0            0            0            0
           1     0.196113    0.0127274            0     0.665864
           2     0.390808    0.0500789     0.196113     0.661049
           3     0.582673     0.085329     0.390808      0.65144
           4     0.770307     0.117442     0.582673     0.637074
           5     0.952326     0.145457     0.770307      0.61801
           6      1.12737     0.168513     0.952326     0.594324
           7       1.2941      0.18587      1.12737     0.566111
           8      1.45123     0.196928       1.2941     0.533485
           9      1.59748     0.201248      1.45123     0.496578
          10      1.73165     0.198563      1.59748     0.455539
          11      1.85256      0.18879      1.73165     0.410535
          12       1.9591     0.172039      1.85256     0.361747
          13      2.05022     0.148616       1.9591     0.309375
          14      2.12492     0.119024      2.05022     0.253632
          15      2.18228    0.0839613      2.12492     0.194747
          16      2.22144    0.0443095      2.18228     0.132961
          17      2.24162  0.00112739      2.22144    0.0685292
          18      2.24213   -0.0443663      2.24162  0.00171792
          19      2.22234   -0.0908101      2.24213   -0.0671949
          20      2.18172    -0.136722      2.22234    -0.137921
          21      2.11982    -0.180522      2.18172    -0.210165
          22      2.03629    -0.220561      2.11982     -0.28362
          23      1.93086    -0.255151      2.03629    -0.357974
          24      1.80335     -0.28259      1.93086     -0.43291
          25       1.6537    -0.301203      1.80335    -0.508104
          26      1.48193    -0.309367       1.6537     -0.58323
          27      1.28814    -0.305548      1.48193    -0.657958
          28      1.07257    -0.288334      1.28814    -0.731957
          29     0.835503    -0.256468      1.07257    -0.804897
          30     0.577369    -0.208875     0.835503    -0.876446
          31     0.298667    -0.114961     0.577369           -0
Comparsion
           i          x_i   actual y_i   spline y_i        error
           0    0.0490874    0.0981452            0    0.0981452
```

1	0.147262	0.293726	0.196817	0.0969097
2	0.245437	0.487182	0.393817	0.0933651
3	0.343612	0.677106	0.58788	0.0892255
4	0.441786	0.862104	0.777551	0.0845528
5	0.539961	1.0408	0.961395	0.0794082
6	0.638136	1.21186	1.13801	0.0738506
7	0.736311	1.37395	1.30601	0.0679358
8	0.834486	1.52579	1.46407	0.0617151
9	0.93266	1.66615	1.61092	0.0552345
10	1.03084	1.79383	1.7453	0.048534
11	1.12901	1.9077	1.86605	0.0416469
12	1.22718	2.00666	1.97206	0.0345993
13	1.32536	2.08968	2.06227	0.0274098
14	1.42353	2.15583	2.13574	0.0200891
15	1.52171	2.20419	2.19155	0.0126407
16	1.61988	2.23395	2.22889	0.0050604
17	1.71806	2.24438	2.24704	0.00266285
18	1.81623	2.23481	2.24535	0.0105465
19	1.91441	2.20466	2.22328	0.0186138
20	2.01258	2.15345	2.18035	0.0268925
21	2.11076	2.08078	2.11619	0.0354142
22	2.20893	1.98632	2.03054	0.0442132
23	2.30711	1.86987	1.9232	0.053325
24	2.40528	1.7313	1.79408	0.0627849
25	2.50346	1.57058	1.6432	0.0726269
26	2.60163	1.38778	1.47066	0.082882
27	2.69981	1.18306	1.27664	0.0935769
28	2.79798	0.956696	1.06143	0.104733
29	2.89616	0.709041	0.825403	0.116362
30	2.99433	0.440553	0.569025	0.128472
31	3.09251	0.151788	0.294415	0.142627