МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»**

Кафедра математической физики и вычислительной математики

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ ПОДГОТОВКЕ

по дисциплине «Методы вычислений»

студента 3 курса 351 группы

направления 09.03.04 Программная инженерия

факультета компьютерных наук и информационных технологий

Устюшина Богдана Антоновича

Проверил: _____

СОДЕРЖАНИЕ

1 Глава 1

1.1 Задание 1

Вычислить сумму функционального ряда
$f(x) = f_0(x) + f_1(x) + \ldots + f_k(x) + \ldots$
$f(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots$
Решение:

```
#include <iostream>
#include <vector>
#include <iomanip>
int main()
{
        double eps,
                sum,
                arg,
                term;
        unsigned int fact = 1,
                stepNumber = 0;
        setlocale(LC_ALL, "Russian");
        std::cout << std::fixed << std::setprecision(8);
        std::cout << "Введите x" << std::endl;
        std::cin >> arg;
        std::cout << "Введите eps" << std::endl;
        std::cin >> eps;
        term = arg;
        sum = arg;
        while (abs(term) > eps)
        {
                fact += 2;
                term *= -arg * arg / (fact * (fact - 1));
                sum += term;
                stepNumber++;
        }
        fact = 1;
```

```
28          std::cout << "Аргумент\t" << "Значение\t" << "Количество
    ↪  итераций\n";
29          std::cout << arg << "\t" << sum << '\t' << stepNumber <<
    ↪  '\n';
30          return 0;
31  }
```

Запуск программы:

```
Введите x
3
Введите eps
0.00001
Аргумент        Значение        Количество итераций
3.00000000      0.14112002      8
```

1.2 Задание 2

Вычислить интерполяционный многочлен

x	-2	-1	0	2
$f(x)$	-8	-1	0	8

$P(x) = a_3x^3 + a_2x^2 + a_1x^1 + a_0$

Решение:

```
1   #include <iostream>
2   #include <vector>
3   #include <utility>
4   using namespace std;
5   const int nodeNumber = 4;
6   int main()
7   {
8           setlocale(LC_ALL, "ru");
9           cout << "Матрица коэффициентов многочлена P(x):\n";
10          vector<pair<double, double>> functionNodes = {
11                  make_pair(-2, -8), make_pair(-1, -1), make_pair(0,
    ↪  0), make_pair(2, 8)
12          };
13          for (pair<double, double> val : functionNodes)
```

```
14          {
15                  for (int i = nodeNumber - 1; i >= 0; i--)
16                          cout << pow(val.first, i) << '\t';
17                  cout << val.second << '\n';
18          }
19          return 0;
20  }
```

Запуск программы:

```
Матрица коэффициентов многочлена P(x):
-8      4       -2      1       -8
-1      1       -1      1       -1
0       0       0       1       0
8       4       2       1       8
```

1.3 Задание 3

Построить интерполяционный многочлен в форме Лагранжа по данным из задания 2

$L_n(x) = \sum_{k=0}^{n} f_k = \frac{(x-x_0)(x-x_1)...(x-x_{k-1})(x-x_{k+1})...(x-x_n)}{(x_k-x_0)(x_k-x_1)...(x_k-x_{k-1})(x_k-x_{k+1})...(x_k-x_n)}$

Вывести значения интерполяционной формулы и значения интерполянты в старых и новых точках.

Решение:

```
1   #include <iostream>
2   #include <vector>
3   using namespace std;
4   const uint32_t nodeNumber = 4;
5   double lagrangeSeries(double x, const vector<pair<double,
    ↪  double>>& functionNodes)
6   {
7           double sum = 0;
8           double item = 1;
9           for (pair<double, double> excl : functionNodes)
10          {
```

```
11                  item *= excl.second;
12                  for (pair<double, double> val : functionNodes)
13                  {
14                          if (val.first != excl.first)
15                                  item *= (x - val.first) /
    ↪  (excl.first - val.first);
16                  }
17                  sum += item;
18                  item = 1;
19          }
20          return sum;
21  }
22  int main()
23  {
24          setlocale(LC_ALL, "russian");
25          vector<pair<double, double>> functionNodes =
26          {
27                  make_pair(-2, -8),
28                  make_pair(-1, -1),
29                  make_pair(0, 0),
30                  make_pair(2, 8)
31          };
32          cout << "Вывод матрицы коэффициентов из предыдущего
    ↪  задания: \n";
33          for (pair<double, double> val : functionNodes)
34          {
35                  for (int i = nodeNumber - 1; i >= 0; i--)
36                  {
37                          cout << pow(val.first, i) << '\t';
38                  }
39                  cout << val.second << '\n';
40          }
41          cout << "\nПодсчёт значений в узловых точках по методу
    ↪  Лагранжа:" << '\n' << "x: \t";
```

```
42          for (int i = 0; i < nodeNumber - 1; i++)
43          {
44                  cout << functionNodes[i].first << '\t'
45                          << (functionNodes[i].first +
    ↪ functionNodes[i + 1].first) / 2 << '\t';
46          }
47          cout << functionNodes[nodeNumber - 1].first << '\n' <<
    ↪ "f(x): \t";
48          for (int i = 0; i < nodeNumber - 1; i++)
49          {
50                  cout << lagrangeSeries(functionNodes[i].first,
    ↪ functionNodes) << '\t'
51                          << lagrangeSeries((functionNodes[i].first
    ↪ + functionNodes[i + 1].first) / 2, functionNodes) << '\t';
52          }
53          cout << lagrangeSeries(functionNodes[nodeNumber -
    ↪ 1].first, functionNodes) << '\n';
54          return 0;
55  }
```

Запуск программы:

```
Вывод матрицы коэффициентов из предыдущего задания:
-8      4       -2      1       -8
-1      1       -1      1       -1
0       0       0       1       0
8       4       2       1       8

Подсчёт значений в узловых точках по методу Лагранжа:
x:      -2      -1.5    -1      -0.5    0       1       2
f(x):   -8      -3.375  -1      -0.125  0       1       8
```

1.4 Задание 4

Построить интерполяционный многочлен в форме Ньютона по данным из задания 2

$N_n(x) = f(x_0) + f(x_0; x_1)(x - x_0) + \ldots + f(x_0; x_1; \ldots; x_n)(x - x_0)(x - x_1)\ldots(x - x_{n-1})$

Вывести значения интерполяционной формулы и значения интерполянты в старых и новых точках, а также полученную таблицу разделённых разностей для метода Ньютона.

Решение:

```
#include <iostream>
#include <vector>
#include <iomanip>
using namespace std;
const uint32_t nodeNumber = 4;
double newtoneSeries(double x, vector<vector<double>>& divDiff,
    const vector<pair<double, double>>& functionNodes)
{
        double ans = divDiff[0][0];
        double currentProd = 1;
        for (int j = 1; j < 4; j++)
        {
                for (int i = 0; i < 4 - j; i++)
                        divDiff[i][j] = (divDiff[i + 1][j - 1] -
    divDiff[i][j - 1]) / (functionNodes[i + j].first -
    functionNodes[i].first);
                currentProd *= (x - functionNodes[j - 1].first);
                ans += divDiff[0][j] * currentProd;
        }
        return ans;
}
int main()
{
        setlocale(LC_ALL, "ru");
        vector<pair<double, double>> functionNodes =
        {
                make_pair(-2, -8),
                make_pair(-1, -1),
                make_pair(0, 0),
                make_pair(2, 8)
```

```
28          };
29          cout << "Исходные значения многочлена: \n";
30          cout << "x: \t";
31          for (pair<double, double> val : functionNodes)
32                  cout << val.first << '\t';
33          cout << '\n';
34          cout << "f(x): \t";
35          for (pair<double, double> val : functionNodes)
36                  cout << val.second << '\t';
37          vector<vector<double>> divDiff;
38          divDiff.resize(4);
39          for (int i = 0; i < 4; i++)
40          {
41                  divDiff[i].resize(4 - i);
42                  divDiff[i][0] = functionNodes[i].second;
43          }
44          cout << "\n\nТаблица разделённых разностей: \n";
45          for (int j = 1; j < 4; j++)
46                  for (int i = 0; i < 4 - j; i++)
47                          divDiff[i][j] = (divDiff[i + 1][j - 1] -
    ↪ divDiff[i][j - 1]) / (functionNodes[i + j].first -
    ↪ functionNodes[i].first);
48
49          for (auto el : divDiff)
50          {
51                  for (auto el2 : el)
52                          cout << setprecision(4) << el2 << '\t';
53                  cout << '\n';
54          }
55          cout << "\nРезультаты вычислений по методу Ньютона: \n";
56          cout << "x: \t";
57          for (int i = 0; i < nodeNumber - 1; i++)
58                  cout << functionNodes[i].first << '\t'
```

```
59                          << (functionNodes[i].first +
    ↪ functionNodes[i + 1].first) / 2 << '\t';
60          cout << functionNodes[nodeNumber - 1].first << '\n';
61          cout << "f(x): \t";
62          for (int i = 0; i < nodeNumber - 1; i++)
63                  cout << newtoneSeries(functionNodes[i].first,
    ↪ divDiff, functionNodes) << '\t'
64                          << newtoneSeries((functionNodes[i].first +
    ↪ functionNodes[i + 1].first) / 2, divDiff, functionNodes) <<
    ↪ '\t';
65          cout << newtoneSeries(functionNodes[nodeNumber - 1].first,
    ↪ divDiff, functionNodes) << '\n';
66  }
```

Запуск программы:

```
Исходные значения многочлена:
x:      -2      -1      0       2
f(x):   -8      -1      0       8

Таблица разделённых разностей:
-8      7       -3      1
-1      1       1
0       4
8

Результаты вычислений по методу Ньютона:
x:      -2      -1.5    -1      -0.5    0       1       2
f(x):   -8      -3.375  -1      -0.125  0       1       8
```

1.5 Задание 5

Построить кусочно-непрерывную «склейку» кубических сплайнов для следующих данных интерполяции:

Решение:

```
1  #include <iostream>
2  #include <vector>
3  using namespace std;
4  const uint32_t nodeNumber = 4;
5  int main()
```

```
6  {
7          setlocale(LC_ALL, "ru");
8          vector<pair<double, double>> functionNodes =
9          {
10                 make_pair(-2, -8),
11                 make_pair(-1, -1),
12                 make_pair(0, 0),
13                 make_pair(2, 8)
14         };
15         cout << "Значения многочлена: \n";
16         cout << "x: \t";
17         for (pair<double, double> val : functionNodes)
18                 cout << val.first << '\t';
19         cout << '\n';
20         cout << "f: \t";
21         for (pair<double, double> val : functionNodes)
22                 cout << val.second << '\t';
23         cout << '\n' << '\n';
24         int splineNum = nodeNumber - 1;
25         vector<vector<double>> matrix;
26         matrix.resize(4 * splineNum);
27         for (int i = 0; i < 4 * splineNum; i++)
28                 matrix[i].resize(4 * splineNum + 1);
29         cout << "Таблица коэффициентов для метода кубических
   ↪ сплайнов: \n";
30         for (int j = 'a'; j <= 'd'; j++)
31                 for (int i = 0; i < splineNum; i++)
32                         cout << char(j) << '_' << i + 1 << '\t';
33         cout << 'f' << '\n';
34         double h = functionNodes[1].first -
   ↪ functionNodes[0].first;
35         for (int i = 0; i < splineNum; i++)
36         {
37                 for (int j = 0; j < 4 * splineNum; j++)
```

```
38                {
39                        if (j == i)
40                                matrix[i][j] = 1;
41                        else
42                                matrix[i][j] = 0;
43                }
44                matrix[i][4 * splineNum] =
    ↪ functionNodes[i].second;
45        }
46        for (int i = splineNum; i < 2 * splineNum; i++)
47        {
48                for (int j = 0; j < 4 * splineNum; j++)
49                {
50                        if (j % splineNum == i % splineNum)
51                                matrix[i][j] = pow(h, (j /
    ↪ splineNum));
52                        else
53                                matrix[i][j] = 0;
54                }
55                matrix[i][4 * splineNum] = functionNodes[i %
    ↪ splineNum + 1].second;
56        }
57        for (int i = 2 * splineNum; i < 3 * splineNum - 1; i++)
58                for (int j = 0; j < 4 * splineNum; j++)
59                {
60                        if (j / splineNum == 1 && j % splineNum -
    ↪ 1 == i % splineNum)
61                                matrix[i][j] = -1;
62                        else if (j % splineNum == i % splineNum)
63                                matrix[i][j] = (j / splineNum) *
    ↪ pow(h, (j / splineNum));
64                        else
65                                matrix[i][j] = 0;
66                }
```

```
67          matrix[8][6] = 2;
68          matrix[9][7] = 2;
69          matrix[8][7] = -2;
70          matrix[9][8] = -2;
71          matrix[8][9] = 6 * h;
72          matrix[9][10] = 6 * h;
73          matrix[8][10] = -2;
74          matrix[10][6] = 2;
75          matrix[11][8] = 2;
76          matrix[11][11] = 6 * h;
77          for (auto str : matrix)
78          {
79                  for (auto it : str)
80                          cout << it << '\t';
81                  cout << '\n';
82          }
83  }
```

Запуск программы:

```
Значения многочлена:
x:      -2      -1      0       2
f:      -8      -1      0       8

Таблица коэффициентов для метода кубических сплайнов:
a_1     a_2     a_3     b_1     b_2     b_3     c_1     c_2     c_3     d_1     d_2     d_3     f
1       0       0       0       0       0       0       0       0       0       0       0       -8
0       1       0       0       0       0       0       0       0       0       0       0       -1
0       0       1       0       0       0       0       0       0       0       0       0       0
1       0       0       1       0       0       1       0       0       1       0       0       -1
0       1       0       0       1       0       0       1       0       0       1       0       0
0       0       1       0       0       1       0       0       1       0       0       1       8
0       0       0       1       -1      0       2       0       0       3       0       0       0
0       0       0       0       1       -1      0       2       0       0       3       0       0
0       0       0       0       0       0       2       -2      0       6       -2      0       0
0       0       0       0       0       0       0       2       -2      0       6       0       0
0       0       0       0       0       0       2       0       0       0       0       0       0
0       0       0       0       0       0       0       0       2       0       0       6       0
```

2 Глава 2

2.1 Задание 1

Решить следующую СЛАУ методом Гаусса.

Решение:

```
1  #include <iostream>
2  #include <vector>
3  #include <iomanip>
4  using namespace std;
5  const int N = 5;
6  const double V = 3;
7  void PrintMatrix(vector<vector<double>> A, vector<double> B)
8  {
9          cout << "Текущая матрица СЛАУ: \n";
10         for (int i = 0; i < N; i++)
11         {
12                 for (int j = 0; j < N; j++)
13                         cout << setprecision(3) << A[i][j] <<
   ↪  "\t\t";
14                 cout << setprecision(4) << "| \t" << B[i] <<
   ↪  '\n';
15         }
16 }
17 int main()
18 {
19         vector<vector<double>> A;
20         vector<double> B;
21         vector<double> x;
22         A.resize(N);
23         for (int i = 0; i < N; i++)
24                 A[i].resize(N);
25         B.resize(N);
26         x.resize(N);
27         setlocale(LC_ALL, "russian");
28         for (int i = 0; i < N; i++)
```

```
29          {
30                  for (int j = 0; j < N; j++)
31                  {
32                          if (i == j)
33                                  A[i][j] = V + (i * 2);
34                          else
35                                  A[i][j] = (V + (i * 2)) / 100;
36                  }
37                  x[i] = V + (i * 2);
38          }
39          double sum;
40          for (int i = 0; i < N; i++)
41          {
42                  sum = 0;
43                  for (int j = 0; j < N; j++)
44                          sum += A[i][j] * x[j];
45                  B[i] = sum;
46                  sum = 0;
47          }
48          PrintMatrix(A, B);
49          cout << "\nПРЯМОЙ ХОД:\n";
50          double currentDiag;
51          double multiplier;
52          for (int i = 0; i < N; i++)
53          {
54                  currentDiag = A[i][i];
55                  for (int j = i; j < N; j++)
56                          A[i][j] /= currentDiag;
57                  B[i] /= currentDiag;
58                  for (int k = i + 1; k < N; k++)
59                  {
60                          multiplier = A[k][i];
61                          for (int j = i; j < N; j++)
62                                  A[k][j] -= multiplier * A[i][j];
```

```
63                          B[k] -= multiplier * B[i];
64                  }
65                  PrintMatrix(A, B);
66          }
67          for (int i = N - 1; i >= 0; i--)
68          {
69                  x[i] = B[i];
70                  for (int j = i + 1; j < N; j++)
71                          x[i] -= A[i][j] * x[j];
72          }
73          cout << "\nРЕЗУЛЬТАТ ОБРАТНОГО ХОДА:\nМатрица X:\n";
74          for (auto row : x)
75                  cout << setprecision(6) << row << '\n';
76          return 0;
77  }
```

Запуск программы:

```
Текущая матрица СЛАУ:
3               0.03            0.03            0.03            0.03            |       9.96
0.05            5               0.05            0.05            0.05            |       26.5
0.07            0.07            7               0.07            0.07            |       50.96
0.09            0.09            0.09            9               0.09            |       83.34
0.11            0.11            0.11            0.11            11              |       123.6

ПРЯМОЙ ХОД:
Текущая матрица СЛАУ:
1               0.01            0.01            0.01            0.01            |       3.32
0               5               0.0495          0.0495          0.0495          |       26.33
0               0.0693          7               0.0693          0.0693          |       50.73
0               0.0891          0.0891          9               0.0891          |       83.04
0               0.109           0.109           0.109           11              |       123.3
Текущая матрица СЛАУ:
1               0.01            0.01            0.01            0.01            |       3.32
0               1               0.0099          0.0099          0.0099          |       5.267
0               0               7               0.0686          0.0686          |       50.36
0               0               0.0882          9               0.0882          |       82.57
0               0               0.108           0.108           11              |       122.7
Текущая матрица СЛАУ:
1               0.01            0.01            0.01            0.01            |       3.32
0               1               0.0099          0.0099          0.0099          |       5.267
0               0               1               0.0098          0.0098          |       7.196
0               0               0               9               0.0874          |       81.94
0               0               0               0.107           11              |       121.9
Текущая матрица СЛАУ:
1               0.01            0.01            0.01            0.01            |       3.32
0               1               0.0099          0.0099          0.0099          |       5.267
0               0               1               0.0098          0.0098          |       7.196
0               0               0               1               0.00971         |       9.107
0               0               0               0               11              |       121
Текущая матрица СЛАУ:
1               0.01            0.01            0.01            0.01            |       3.32
0               1               0.0099          0.0099          0.0099          |       5.267
0               0               1               0.0098          0.0098          |       7.196
0               0               0               1               0.00971         |       9.107
0               0               0               0               1               |       11

РЕЗУЛЬТАТ ОБРАТНОГО ХОДА:
Матрица X:
3
5
7
9
11
```

2.2 Задание 2

Вычислить определитель и обратную матрицу данной.

Решение:

```
1 #include <iostream>
2 #include <vector>
3 #include <iomanip>
4 using namespace std;
5 const int N = 3;
6 const double V = 3;
```

```
7  int main()
8  {
9          vector<vector<double>> A;
10         vector<vector<double>> A_copy;
11         vector<double> B;
12         vector<double> x;
13         A.resize(N);
14         for (int i = 0; i < N; i++)
15                 A[i].resize(N);
16         B.resize(N);
17         x.resize(N);
18         setlocale(LC_ALL, "russian");
19         A[0][0] = 3, A[0][1] = 3, A[0][2] = -1;
20         A[1][0] = 4, A[1][1] = 1, A[1][2] = 3;
21         A[2][0] = 1, A[2][1] = -2,A[2][2] = -2;
22         B[0] = 4, B[1] = 8, B[2] = 11;
23         cout << "Рассматриваемая матрица: \n";
24         for (int i = 0; i < N; i++)
25         {
26                 for (int j = 0; j < N; j++)
27                         cout << setprecision(3) << A[i][j] <<
   ↪  "\t\t";
28                 cout << setprecision(4) << "| \t" << B[i] <<
   ↪  '\n';
29         }
30         cout << fixed;
31         double determinator = 1;
32         double currentDiag;
33         double multiplier;
34         A_copy = A;
35         for (int i = 0; i < N; i++)
36         {
37                 currentDiag = A_copy[i][i];
38                 for (int j = i; j < N; j++)
```

```
39                      A_copy[i][j] /= currentDiag;
40              for (int k = i + 1; k < N; k++)
41              {
42                      multiplier = A_copy[k][i];
43                      for (int j = i; j < N; j++)
44                              A_copy[k][j] -= multiplier *
    ↪ A_copy[i][j];
45              }
46              determinator *= currentDiag;
47      }
48      cout << "\nОпределитель: " << determinator << "\n\n";
49      vector<vector<double>> A_reversed;
50      A_reversed.resize(N);
51      for (int k = 0; k < N; k++)
52      {
53              A_reversed[k].resize(N);
54              A_copy = A;
55              for (int i = 0; i < N; i++)
56              {
57                      if (i == k)
58                              B[i] = 1;
59                      else
60                              B[i] = 0;
61              }
62              for (int i = 0; i < N; i++)
63              {
64                      currentDiag = A_copy[i][i];
65                      for (int j = i; j < N; j++)
66                              A_copy[i][j] /= currentDiag;
67                      B[i] /= currentDiag;
68
69                      for (int k = i + 1; k < N; k++)
70                      {
71                              multiplier = A_copy[k][i];
```

```
72                                  for (int j = i; j < N; j++)
73                                          A_copy[k][j] -= multiplier
    ↪ * A_copy[i][j];
74                                  B[k] -= multiplier * B[i];
75
76                          }
77                  }
78                  for (int i = N - 1; i >= 0; i--)
79                  {
80                          A_reversed[k][i] = B[i];
81                          for (int j = i + 1; j < N; j++)
82                                  A_reversed[k][i] -= A_copy[i][j] *
    ↪ A_reversed[k][j];
83                  }
84          }
85          cout << "Обратная матрица: \n";
86          for (int i = 0; i < N; i++)
87          {
88                  for (int j = 0; j < N; j++)
89                          cout << A_reversed[j][i] << "\t\t";
90                  cout << '\n';
91          }
92  }
```

Запуск программы:

```
Рассматриваемая матрица:
3               3               -1              |       4
4               1               3               |       8
1               -2              -2              |       11

Определитель: 54.0000

Обратная матрица:
0.0741          0.1481          0.1852
0.2037          -0.0926         -0.2407
-0.1667         0.1667          -0.1667
```

2.3 Задание 3

Решить данную СЛАУ методом прогонки.

Решение:

```
1  #include <iostream>
2  #include <iomanip>
3  #include <vector>
4  using namespace std;
5  const int N = 5;
6  const double V = 3;
7  void PrintMatrix(vector<vector<double>> A, vector<double> B)
8  {
9          cout << "Текущая матрица СЛАУ: \n";
10         for (int i = 0; i < N; i++)
11         {
12                 for (int j = 0; j < N; j++)
13                         cout << setprecision(3) << A[i][j] <<
   ↪  "\t\t";
14                 cout << setprecision(4) << "| \t" << B[i] <<
   ↪  '\n';
15         }
16 }
17 int main()
18 {
19         setlocale(LC_ALL, "russian");
20         vector<vector<double>> M;
21         vector<double> D;
22         vector<double> x;
23         vector<double> triag_P;
24         vector<double> triag_Q;
25         vector<vector<double>> tripples;
26         M.resize(N);
27         tripples.resize(N);
28         for (int i = 0; i < N; i++)
29         {
```

```
30                 M[i].resize(N);
31                 tripples[i].resize(3);
32         }
33         D.resize(N);
34         x.resize(N);
35         triag_P.resize(N);
36         triag_Q.resize(N);
37         for (int i = 0; i < N; i++)
38         {
39                 for (int j = 0; j < N; j++)
40                 {
41                         if (i == j)
42                                 M[i][j] = V + (i * 2);
43                         else
44                                 M[i][j] = (V + (i * 2)) / 100;
45                 }
46                 x[i] = V + (i * 2);
47         }
48         tripples[0][0] = 0;
49         tripples[0][1] = M[0][0];
50         tripples[0][2] = M[0][1];
51         tripples[N - 1][0] = M[N - 1][N - 2];
52         tripples[N - 1][1] = M[N - 1][N - 1];
53         tripples[N - 1][2] = 0;
54         for (int i = 1; i < N - 1; i++)
55                 for (int j = 0; j < 3; j++)
56                         tripples[i][j] = M[i][j + i - 1]; //j == 1
    ↪  ? -M[i][j + i - 1] :
57         D[0] = tripples[0][1] * x[0] + tripples[0][2] * x[1];
58         D[N - 1] = tripples[N - 1][0] * x[N - 2] + tripples[N -
    ↪  1][1] * x[N - 1];
59         double sum;
60         for (int i = 1; i < N - 1; i++)
61         {
```

```
62                  sum = 0;
63                  for (int j = -1; j < 2; j++)
64                  {
65                          sum += tripples[i][j + 1] * x[i + j];
66                  }
67                  D[i] = sum;
68                  sum = 0;
69          }
70          PrintMatrix(M, D);
71          triag_P[0] = tripples[0][2] / -tripples[0][1];
72          triag_Q[0] = -D[0] / -tripples[0][1];
73          for (int i = 1; i < N - 1; i++)
74          {
75                  triag_P[i] = tripples[i][2] / (-tripples[i][1] -
    ↪ tripples[i][0] * triag_P[i - 1]);
76                  triag_Q[i] = (tripples[i][0] * triag_Q[i - 1] -
    ↪ D[i]) / (-tripples[i][1] - tripples[i][0] * triag_P[i - 1]);
77          }
78          triag_P[N - 1] = 0;
79          triag_Q[N - 1] = (tripples[N - 1][0] * triag_Q[N - 2] -
    ↪ D[N - 1]) / (-tripples[N - 1][1] - tripples[N - 1][0] *
    ↪ triag_P[N - 2]);
80          cout << "Прогоночные коэффициенты P: \n";
81          for (auto row : triag_P)
82                  cout << row << '\n';
83          cout << "\nПрогоночные коэффициенты Q: \n";
84          for (auto row : triag_Q)
85                  cout << row << '\n';
86          cout << '\n';
87          x[N - 1] = triag_Q[N - 1];
88          for (int i = N - 2; i >= 0; i--)
89                  x[i] = triag_P[i] * x[i + 1] + triag_Q[i];
90          cout << "Вектор X: \n";
91          for (auto it : x)
```

```
92                  cout << it << '\n';
93  }
```

Запуск программы:

```
Текущая матрица СЛАУ:
3               0.03            0.03            0.03            0.03            |       9.15
0.05            5               0.05            0.05            0.05            |       25.5
0.07            0.07            7               0.07            0.07            |       49.98
0.09            0.09            0.09            9               0.09            |       82.62
0.11            0.11            0.11            0.11            11              |       122
Прогоночные коэффициенты P:
-0.01
-0.01
-0.01
-0.01
0

Прогоночные коэффициенты Q:
3.05
5.07
7.09
9.11
11

Вектор X:
3
5
7
9
11
```

2.4 Задание 4

Решить данную СЛАУ методом простой итерации.

Решение:

```
1  #include <iostream>
2  #include <iomanip>
3  #include <vector>
4  using namespace std;
5  const int N = 5;
6  const double V = 3;
7  const double eps = 0.0001;
8  void PrintMatrix(vector<vector<double>> A, vector<double> B)
9  {
10          cout << "Текущие матрицы A и B: \n";
11          for (int i = 0; i < N; i++)
```

```
12          {
13                  for (int j = 0; j < N; j++)
14                          cout << setprecision(3) << A[i][j] <<
    ↪   "\t\t";
15                  cout << setprecision(4) << "|\t" << B[i] <<
    ↪   '\n';
16          }
17  }
18  int main()
19  {
20          setlocale(LC_ALL, "ru");
21          vector<vector<double>> A;
22          vector<double> B;
23          vector<double> x;
24          vector<vector<double>> alpha;
25          vector<double> beta;
26          A.resize(N);
27          B.resize(N);
28          x.resize(N);
29          alpha.resize(N);
30          beta.resize(N);
31          for (int i = 0; i < N; i++)
32          {
33                  A[i].resize(N);
34                  alpha[i].resize(N);
35          }
36          for (int i = 0; i < N; i++)
37          {
38                  for (int j = 0; j < N; j++)
39                  {
40                          if (i == j)
41                                  A[i][j] = V + (i * 2);
42                          else
43                                  A[i][j] = (V + (i * 2)) / 100;
```

```
44                  }
45                  x[i] = V + (i * 2);
46          }
47          double sum;
48          for (int i = 0; i < N; i++)
49          {
50                  sum = 0;
51                  for (int j = 0; j < N; j++)
52                  {
53                          sum += A[i][j] * x[j];
54                  }
55                  B[i] = sum;
56                  sum = 0;
57          }
58          for (int i = 0; i < N; i++)
59          {
60                  for (int j = 0; j < N; j++)
61                  {
62                          alpha[i][j] = i == j ? 0 : -(A[i][j] /
    ↪  A[i][i]);
63                  }
64                  beta[i] = B[i] / A[i][i];
65                  x[i] = 0;
66          }
67          PrintMatrix(A, B);
68          vector<double> new_x;
69          new_x.resize(N);
70          bool finished = false;
71          while (!finished)
72          {
73                  finished = true;
74                  double sum;
75                  for (int i = 0; i < N; i++)
76                  {
```

```
77                          sum = 0;
78                          for (int j = 0; j < N; j++)
79                          {
80                                  sum += alpha[i][j] * x[j];
81                          }
82                          new_x[i] = sum + beta[i];
83                          if (abs(new_x[i] - x[i]) >= eps)
84                                  finished = false;
85                          sum = 0;
86                  }
87                  x = new_x;
88
89                  cout << "Промежуточный вектор x: \n";
90                  for (auto row : x)
91                          cout << row << '\n';
92          }
93          cout << "\nОТВЕТ: \nКонечный вектор x: \n";
94          for (auto row : x)
95                  cout << row << '\n';
96  }
```

Запуск программы:

```
Текущие матрицы A и B:
3               0.03            0.03            0.03            0.03            |       9.96
0.05            5               0.05            0.05            0.05            |       26.5
0.07            0.07            7               0.07            0.07            |       50.96
0.09            0.09            0.09            9               0.09            |       83.34
0.11            0.11            0.11            0.11            11              |       123.6
Промежуточный вектор x:
3.32
5.3
7.28
9.26
11.24
Промежуточный вектор x:
2.989
4.989
6.989
8.989
10.99
Промежуточный вектор x:
3
5
7
9
11
Промежуточный вектор x:
3
5
7
9
11
Промежуточный вектор x:
3
5
7
9
11

ОТВЕТ:
Конечный вектор x:
3
5
7
9
11
```

3 Глава 3

3.1 Задание 1

Решить дифференциальное уравнение:

- Методом Эйлера
- Методом (усовершенствованным) Эйлера
- Методом предварительного и корректирующего счёта

Решение:

```
#include <iostream>
#include <iomanip>
#include <vector>
using namespace std;
const int N = 5;
const double V = 3;
const double h = 0.01;
double func(double x, double y_x)
{
        return 2 * V * x + V * x * x - y_x;
}

double theor_res(double x)
{
        return V * x * x;
}
int main()
{
        setlocale(LC_ALL, "ru");
        vector<double> nodes;
        nodes.resize(N);
        vector<double> pract_value_m1;
        pract_value_m1.resize(N);
        vector<double> theor_value;
        theor_value.resize(N);
        vector<double> error_m1;
        error_m1.resize(N);
```

```
28          double x = 1;
29          for (int i = 0; i < N; i++)
30                  nodes[i] = x;
31                  theor_value[i] = theor_res(x);
32                  x += h;
33          pract_value_m1[0] = V;
34          for (int i = 1; i < N; i++)
35                  pract_value_m1[i] = pract_value_m1[i - 1] + h *
    ↪ func(nodes[i - 1], pract_value_m1[i - 1]);
36          for (int i = 0; i < N; i++)
37                  error_m1[i] = abs(theor_value[i] -
    ↪ pract_value_m1[i]);
38          cout << "МЕТОД ЭЙЛЕРА: \n";
39          cout << fixed << setprecision(8);
40          cout << "Значения узлов: \n";
41          for (auto it : nodes)
42                  cout << it << '\t';
43          cout << '\n' << "Значения метода: \n";
44          for (auto it : pract_value_m1)
45                  cout << it << '\t';
46          cout << '\n';
47          cout << "Теоретическое значение: \n";
48          for (auto it : theor_value)
49                  cout << it << '\t';
50          cout << '\n';
51          cout << "Погрешность: \n";
52          for (auto it : error_m1)
53                  cout << it << '\t';
54          cout << "\n\n";
55          vector<double> pract_value_m2;
56          pract_value_m2.resize(N);
57          vector<double> error_m2;
58          error_m2.resize(N);
59          pract_value_m2[0] = V;
```

```
60          double func_half_h;
61          for (int i = 1; i < N; i++)
62                  func_half_h = pract_value_m2[i - 1] + h / 2 *
    ↪ func(nodes[i - 1], pract_value_m2[i - 1]);
63                  pract_value_m2[i] = pract_value_m2[i - 1] + h *
    ↪ func(nodes[i - 1] + h / 2, func_half_h);
64          for (int i = 0; i < N; i++)
65                  error_m2[i] = theor_value[i] - pract_value_m2[i];
66          cout << "УСОВЕРШЕСТВОВАННЫЙ МЕТОД ЭЙЛЕРА: \n";
67          cout << "Значения узлов: \n";
68          for (auto it : nodes)
69                  cout << it << '\t';
70          cout << '\n' << "Значения метода: \n";
71          for (auto it : pract_value_m2)
72                  cout << it << '\t';
73          cout << '\n';
74          cout << "Теоретическое значение: \n";
75          for (auto it : theor_value)
76                  cout << it << '\t';
77          cout << '\n';
78          cout << "Погрешность: \n";
79          for (auto it : error_m2)
80                  cout << it << '\t';
81          cout << "\n\n";
82          vector<double> pract_value_m3;
83          pract_value_m3.resize(N);
84          vector<double> error_m3;
85          error_m3.resize(N);
86          pract_value_m3[0] = V;
87          double func_h;
88          for (int i = 1; i < N; i++)
89                  func_h = pract_value_m3[i - 1] + h * func(nodes[i
    ↪ - 1], pract_value_m3[i - 1]);
```

```
90                  pract_value_m3[i] = pract_value_m3[i - 1] + h / 2
    ↪ * (func(nodes[i - 1], pract_value_m3[i - 1]) + func(nodes[i],
    ↪ func_h));
91          for (int i = 0; i < N; i++)
92                  error_m3[i] = abs(theor_value[i] -
    ↪ pract_value_m3[i]);
93          cout << "МЕТОД ПРЕДВАРИТЕЛЬНОГО И КОРРЕКТИРУЮЩЕГО
    ↪ СЧЁТА: \n";
94          cout << "Значения узлов: \n";
95          for (auto it : nodes)
96                  cout << it << '\t';
97          cout << '\n' << "Значения метода: \n";
98          for (auto it : pract_value_m3)
99                  cout << it << '\t';
100         cout << '\n';
101         cout << "Теоретическое значение: \n";
102         for (auto it : theor_value)
103                 cout << it << '\t';
104         cout << '\n';
105         cout << "Погрешность: \n";
106         for (auto it : error_m3)
107                 cout << it << '\t';
108         cout << " \n\n";
109         return 0;
110 }
```

Запуск программы:

```
МЕТОД ЭЙЛЕРА:
Значения узлов:
1.00000000      1.01000000      1.02000000      1.03000000      1.04000000
Значения метода:
3.00000000      3.06000000      3.12060300      3.18180897      3.24361788
Теоретическое значение:
3.00000000      3.06030000      3.12120000      3.18270000      3.24480000
Погрешность:
0.00000000      0.00030000      0.00059700      0.00089103      0.00118212

УСОВЕРШЕСТВОВАННЫЙ МЕТОД ЭЙЛЕРА:
Значения узлов:
1.00000000      1.01000000      1.02000000      1.03000000      1.04000000
Значения метода:
3.00000000      3.06030075      3.12120149      3.18270223      3.24480296
Теоретическое значение:
3.00000000      3.06030000      3.12120000      3.18270000      3.24480000
Погрешность:
0.00000000      -0.00000075     -0.00000149     -0.00000223     -0.00000296

МЕТОД ПРЕДВАРИТЕЛЬНОГО И КОРРЕКТИРУЮЩЕГО СЧЁТА:
Значения узлов:
1.00000000      1.01000000      1.02000000      1.03000000      1.04000000
Значения метода:
3.00000000      3.06030150      3.12120299      3.18270446      3.24480591
Теоретическое значение:
3.00000000      3.06030000      3.12120000      3.18270000      3.24480000
Погрешность:
0.00000000      0.00000150      0.00000299      0.00000446      0.00000591
```

3.2 Задание 2

Решить следующую краевую задачу разностным методом.

Решение:

```
1  #include <iostream>
2  #include <iomanip>
3  #include <vector>
4  using namespace std;
5  const int N = 25;
6  const double V = 8;
7  double func(double x)
8  {
9          double T = V;
10         return 4 * V * x * x * x * x
11                 - 3 * V * T * x * x * x
```

```
12                  + 6 * V * x
13                  - 2 * V * T;
14  }
15  double theor_res(double x)
16  {
17          double T = V;
18          return V * x * x * (x - T);
19  }
20  void triag_method(vector<vector<double>> M, vector<double> D,
    ↪  vector<double>& x)
21  {
22          vector<double> triag_P;
23          vector<double> triag_Q;
24          vector<vector<double>> tripples;
25          triag_P.resize(N);
26          triag_Q.resize(N);
27          tripples.resize(N);
28          for (int i = 0; i < N; i++)
29                  tripples[i].resize(3);
30          tripples[0][0] = 0;
31          tripples[0][1] = M[0][0];
32          tripples[0][2] = M[0][1];
33          tripples[N - 1][0] = M[N - 1][N - 2];
34          tripples[N - 1][1] = M[N - 1][N - 1];
35          tripples[N - 1][2] = 0;
36          for (int i = 1; i < N - 1; i++)
37                  for (int j = 0; j < 3; j++)
38                          tripples[i][j] = M[i][j + i - 1];
39          triag_P[0] = tripples[0][2] / -tripples[0][1];
40          triag_Q[0] = -D[0] / -tripples[0][1];
41          for (int i = 1; i < N - 1; i++)
42          {
43                  triag_P[i] = tripples[i][2] / (-tripples[i][1] -
    ↪  tripples[i][0] * triag_P[i - 1]);
```

```
44                 triag_Q[i] = (tripples[i][0] * triag_Q[i - 1] -
    ↪ D[i]) / (-tripples[i][1] - tripples[i][0] * triag_P[i - 1]);
45         }
46         triag_P[N - 1] = 0;
47         triag_Q[N - 1] = (tripples[N - 1][0] * triag_Q[N - 2] -
    ↪ D[N - 1]) / (-tripples[N - 1][1] - tripples[N - 1][0] *
    ↪ triag_P[N - 2]);
48         x[N - 1] = triag_Q[N - 1];
49         for (int i = N - 2; i >= 0; i--)
50                 x[i] = triag_P[i] * x[i + 1] + triag_Q[i];
51 }
52 int main()
53 {
54         setlocale(LC_ALL, "ru");
55         vector<double> nodes;
56         nodes.resize(N);
57         vector<double> pract_value;
58         pract_value.resize(N);
59         vector<double> theor_value;
60         theor_value.resize(N);
61         vector<double> error;
62         error.resize(N);
63         vector<vector<double>> M;
64         vector<double> b;
65         M.resize(N);
66         for (int i = 0; i < N; i++)
67                 M[i].resize(N);
68         b.resize(N);
69         double x = 0;
70         double h = (V - x) / (N - 1);
71         nodes[0] = x;
72         theor_value[0] = theor_res(x);
73         x += h;
74         M[0][0] = 1;
```

```
75          M[N - 1][N - 1] = 1;
76          for (int i = 1; i < N - 1; i++)
77          {
78                  nodes[i] = x;
79                  theor_value[i] = theor_res(x);
80
81                  M[i][i - 1] = 1 / (h * h) - (x * x) / (2 * h);
82                  M[i][i] = -2 / (h * h) + x;
83                  M[i][i + 1] = 1 / (h * h) + (x * x) / (2 * h);
84
85                  b[i] = func(x);
86                  x += h;
87          }
88          nodes[N - 1] = x;
89          theor_value[N - 1] = theor_res(x);
90          b[0] = b[N - 1] = 0;
91          cout << fixed << setprecision(8);
92          triag_method(M, b, pract_value);
93          double max_error = 0;
94          int k = 0;
95          for (int i = 0; i < N; i++)
96          {
97                  error[i] = abs(theor_value[i] - pract_value[i]);
98                  if (error[i] > max_error)
99                  {
100                         max_error = error[i];
101                         k = i;
102                 }
103         }
104         cout << "Значение узлов \t\t Практическое
    ↪  значение \t Теоретическое значение \t Погрешность \n";
105         for (int i = 0; i < N; i++)
106                 cout << nodes[i] << " \t\t " << pract_value[i] <<
    ↪  " \t\t " << theor_value[i] << " \t\t " << error[i] << '\n';
```

```
107         cout << "Максимальная погрешность: " << max_error <<
    ↪  "\nНомер узла максимальной погрешности: " << k;
108         return 0;
109 }
```

Запуск программы:

```
Значение узлов          Практическое значение   Теоретическое значение  Погрешность
0.00000000              0.00000000              -0.00000000             0.00000000
0.33333333              0.02874441              -6.81481481             6.84355923
0.66666667              -12.89544343            -26.07407407            13.17863065
1.00000000              -38.30984042            -56.00000000            17.69015958
1.33333333              -75.67156776            -94.81481481            19.14324706
1.66666667              -123.13192417           -140.74074074           17.60881657
2.00000000              -177.37094880           -192.00000000           14.62905120
2.33333333              -234.96929389           -246.81481481           11.84552093
2.66666667              -293.58899309           -303.40740741           9.81841431
3.00000000              -351.66251356           -360.00000000           8.33748644
3.33333333              -407.64836321           -414.81481481           7.16645160
3.66666667              -459.87334123           -466.07407407           6.20073284
4.00000000              -506.61879098           -512.00000000           5.38120902
4.33333333              -546.14433916           -550.81481481           4.67047566
4.66666667              -576.69737799           -580.74074074           4.04336275
5.00000000              -596.51788634           -600.00000000           3.48211366
5.33333333              -603.84102785           -606.81481481           2.97378696
5.66666667              -596.89872655           -599.40740741           2.50868085
6.00000000              -573.92063248           -576.00000000           2.07936752
6.33333333              -533.13476642           -534.81481481           1.68004839
6.66666667              -472.76794566           -474.07407407           1.30612841
7.00000000              -391.04608714           -392.00000000           0.95391286
7.33333333              -286.19441762           -286.81481481           0.62039719
7.66666667              -156.43763082           -156.74074074           0.30310992
8.00000000              0.00000000              -0.00000000             0.00000000
Максимальная погрешность: 19.14324706
Номер узла максимальной погрешности: 4
```

3.3 Задание 3

Решить краевую задачу из предыдущего задания методом неопределённых коэффициентов.

Решение:

```
1 #include <iostream>
2 #include <vector>
3 #include <iomanip>
4 using namespace std;
5 const int N = 25;
6 const double V = 4;
7 const double T = V;
```

```
 8 double p(double x)
 9 {
10         return x * x;
11 }
12 double q(double x)
13 {
14         return x;
15 }
16 double func(double x)
17 {
18         return 4 * V * x * x * x * x
19                 - 3 * V * T * x * x * x
20                 + 6 * V * x
21                 - 2 * V * T;
22 }
23 double phi(int k, double x, double T)
24 {
25         return pow(x, k + 1) * (x - T);
26 }
27 double phi_1(int k, double x, double T)
28 {
29         return (k + 2) * pow(x, k + 1) - (k + 1) * T * pow(x, k);
30 }
31 double phi_2(int k, double x, double T)
32 {
33         return (k + 1) * (k + 2) * pow(x, k) - k * (k + 1) * T *
    ↪  pow(x, k - 1);
34 }
35 double coefs(int k, double x, double T)
36 {
37         return phi_2(k, x, T) + p(x) * phi_1(k, x, T) + q(x) *
    ↪  phi(k, x, T);
38 }
39 double theor_res(double x)
```

```
40 {
41         return V * x * x * (x - T);
42 }
43 int main()
44 {
45         setlocale(LC_ALL, "ru");
46         vector<double> nodes(N);
47         vector<double> A(N);
48         vector<double> pract_value(N);
49         vector<double> theor_value(N);
50         vector<double> error(N);
51         vector<vector<double>> M;
52         vector<double> B(N);
53         M.resize(N);
54         for (int i = 0; i < N; i++)
55                 M[i].resize(N);
56         B.resize(N);
57         double x = 0;
58         double h = (V - x) / N;
59         for (int i = 0; i < N; i++)
60         {
61                 x += h;
62                 for (int j = 0; j < N; j++)
63                         M[i][j] = coefs(j + 1, x, T);
64                 B[i] = func(x);
65                 nodes[i] = x;
66                 theor_value[i] = theor_res(x);
67         }
68         int l = 0;
69         double currentDiag;
70         double multiplier;
71         for (int i = 0; i < N; i++)
72         {
73                 currentDiag = M[i][i];
```

```
74                  for (int j = i; j < N; j++)
75                          M[i][j] /= currentDiag;
76                  B[i] /= currentDiag;
77                  for (int k = i + 1; k < N; k++)
78                  {
79                          multiplier = M[k][i];
80                          for (int j = i; j < N; j++)
81                                  M[k][j] -= multiplier * M[i][j];
82                          B[k] -= multiplier * B[i];
83                  }
84          }
85          for (int i = N - 1; i >= 0; i--)
86          {
87                  A[i] = B[i];
88                  for (int j = i + 1; j < N; j++)
89                          A[i] -= M[i][j] * A[j];
90          }
91          for (int i = 0; i < N; i++)
92          {
93                  double s = 0;
94                  for (int j = 0; j < N; j++)
95                          s += A[j] * phi(j + 1, nodes[i], T);
96                  pract_value[i] = s;
97                  error[i] = abs(pract_value[i] - theor_value[i]);
98          }
99          cout << "Значение узлов \t Практическое
    ↪ значение \t Теоретическое значение \t Погрешность \n";
100         for (int i = 0; i < N; i++)
101                 cout << fixed << setprecision(6) << nodes[i] <<
    ↪ "\t" << pract_value[i] << "\t\t" << theor_value[i] <<
    ↪ "\t\t" << scientific << error[i] << '\n';
102 }
```

Запуск программы:

Значение узлов	Практическое значение	Теоретическое значение	Погрешность
0.160000	-0.393216	-0.393216	9.627910e-12
0.320000	-1.507328	-1.507328	1.914824e-11
0.480000	-3.244032	-3.244032	2.858203e-11
0.640000	-5.505024	-5.505024	3.729017e-11
0.800000	-8.192000	-8.192000	4.485479e-11
0.960000	-11.206656	-11.206656	5.080025e-11
1.120000	-14.450688	-14.450688	5.478995e-11
1.280000	-17.825792	-17.825792	5.663381e-11
1.440000	-21.233664	-21.233664	5.643130e-11
1.600000	-24.576000	-24.576000	5.454837e-11
1.760000	-27.754496	-27.754496	5.147527e-11
1.920000	-30.670848	-30.670848	4.780176e-11
2.080000	-33.226752	-33.226752	4.400391e-11
2.240000	-35.323904	-35.323904	4.043699e-11
2.400000	-36.864000	-36.864000	3.726086e-11
2.560000	-37.748736	-37.748736	3.451106e-11
2.720000	-37.879808	-37.879808	3.215916e-11
2.880000	-37.158912	-37.158912	3.014833e-11
3.040000	-35.487744	-35.487744	2.842171e-11
3.200000	-32.768000	-32.768000	2.687273e-11
3.360000	-28.901376	-28.901376	2.555467e-11
3.520000	-23.789568	-23.789568	2.428990e-11
3.680000	-17.334272	-17.334272	2.276934e-11
3.840000	-9.437184	-9.437184	2.403233e-11
4.000000	0.000000	0.000000	3.077189e-25

4 Глава 4

Решить данное интегральное уравнение.

Решение:

4.1 Задание 1

```
1  #include <iostream>
2  #include <vector>
3  #include <iomanip>
4  using namespace std;
5  const int N = 3;
6  const int N_NODES = 25;
7  const double a = 0;
8  const double b = 1;
9  const double V = 4;
10 const double T = V;
11 double a1(double x)
12 {
13         return x;
14 }
15 double a2(double x)
16 {
17         return x * x;
18 }
19 double a3(double x)
20 {
21         return x * x * x;
22 }
23 double b1(double t)
24 {
25         return t;
26 }
27 double b2(double t)
28 {
29         return t * t;
```

```
30 }
31 double b3(double t)
32 {
33         return t * t * t;
34 }
35 double theor_res(double x)
36 {
37         return V * x;
38 }
39 double f(double x)
40 {
41         return V * ((4.0 / 3) * x + (1.0 / 4) * x * x + (1.0 / 5)
   ↪  * x * x * x);
42 }
43 int main()
44 {
45         setlocale(LC_ALL, "ru");
46         vector<double> nodes(N_NODES);
47         vector<double> q_value(N);
48         vector<double> pract_value(N_NODES);
49         vector<double> theor_value(N_NODES);
50         vector<double> error(N_NODES);
51         vector<vector<double>> M;
52         vector<double> B(N);
53         M.resize(N);
54         for (int i = 0; i < N; i++)
55                 M[i].resize(N);
56         B.resize(N);
57         double x = 0;
58         double h = (b - a) / N_NODES;
59         for (int i = 0; i < N_NODES; i++)
60         {
61                 nodes[i] = x;
62                 theor_value[i] = theor_res(x);
```

```
63                  x += h;
64          }
65          //Вычисленные alpha
66          for (int i = 0; i < N; i++)
67                  for (int j = 0; j < N; j++)
68                          M[i][j] = i == j ? 1 + 1.0 / (i + j + 3) :
    ↪  1.0 / (i + j + 3);;
69          //Вычисленные phi
70          B[0] = 1969.0 / 900;
71          B[1] = 5.0 / 3;
72          B[2] = 283.0 / 210;
73          double currentDiag;
74          double multiplier;
75          for (int i = 0; i < N; i++)
76          {
77                  currentDiag = M[i][i];
78                  for (int j = i; j < N; j++)
79                          M[i][j] /= currentDiag;
80                  B[i] /= currentDiag;
81
82                  for (int k = i + 1; k < N; k++)
83                  {
84                          multiplier = M[k][i];
85                          for (int j = i; j < N; j++)
86                          {
87                                  M[k][j] -= multiplier * M[i][j];
88                          }
89                          B[k] -= multiplier * B[i];
90                  }
91          }
92          for (int i = N - 1; i >= 0; i--)
93          {
94                  q_value[i] = B[i];
95                  for (int j = i + 1; j < N; j++)
```

```
96                          q_value[i] -= M[i][j] * q_value[j];
97          }
98          double s = 0;
99          for (int i = 0; i < N_NODES; i++)
100         {
101                 s += a1(nodes[i]) * q_value[0];
102                 s += a2(nodes[i]) * q_value[1];
103                 s += a3(nodes[i]) * q_value[2];
104                 pract_value[i] = f(nodes[i]) - s;
105                 error[i] = abs(pract_value[i] - theor_value[i]);
106                 s = 0;
107         }
108         cout << "Значение узлов \t Практическое
    ↪ значение \t Теоретическое значение \t Погрешность \n";
109         for (int i = 0; i < N_NODES; i++)
110                 cout << fixed << setprecision(6) << nodes[i] <<
    ↪ "\t" << pract_value[i] << "\t\t" << theor_value[i] <<
    ↪ "\t\t" << scientific << error[i] << '\n';
111 }
```

Запуск программы:

Значение узлов	Практическое значение	Теоретическое значение	Погрешность
0.000000	0.000000	0.000000	0.000000e+00
0.040000	0.160000	0.160000	2.775558e-17
0.080000	0.320000	0.320000	0.000000e+00
0.120000	0.480000	0.480000	1.110223e-16
0.160000	0.640000	0.640000	1.110223e-16
0.200000	0.800000	0.800000	1.110223e-16
0.240000	0.960000	0.960000	1.110223e-16
0.280000	1.120000	1.120000	0.000000e+00
0.320000	1.280000	1.280000	2.220446e-16
0.360000	1.440000	1.440000	2.220446e-16
0.400000	1.600000	1.600000	0.000000e+00
0.440000	1.760000	1.760000	2.220446e-16
0.480000	1.920000	1.920000	0.000000e+00
0.520000	2.080000	2.080000	4.440892e-16
0.560000	2.240000	2.240000	0.000000e+00
0.600000	2.400000	2.400000	4.440892e-16
0.640000	2.560000	2.560000	4.440892e-16
0.680000	2.720000	2.720000	4.440892e-16
0.720000	2.880000	2.880000	4.440892e-16
0.760000	3.040000	3.040000	4.440892e-16
0.800000	3.200000	3.200000	0.000000e+00
0.840000	3.360000	3.360000	4.440892e-16
0.880000	3.520000	3.520000	0.000000e+00
0.920000	3.680000	3.680000	4.440892e-16
0.960000	3.840000	3.840000	8.881784e-16