ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧЕРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Санкт-Петербургский политехнический университет Петра Великого»

Институт компьютерных наук и технологий
Высшая школа программной инженерии

Отчет по лабораторной работе по дисциплине «Вычислительная математика»

Выполнила студентка гр. 3530904/80001 Прохорова А. И.

Руководитель Устинов С. М.

Санкт-Петербург
2019

1 Задание

Для таблично заданной функции f(x)

x	-1.0	-0.9	-0.8	-0.7	-0.6	-0.5
f(x)	0.5440	-0.4121	-0.9894	-0.6570	0.2794	0.9589

построить сплайн-функцию и использовать её для нахождения корня уравнения $f(x) = 1.8 \cdot x^2$ на промежутке $[-1, -0.5]$ методом бисекции.

2 Ход работы

2.1 Алгоритм решения

Расчёт выражения производится с помощью алгоритма сортировочной станции - способа разбора математических выражений, записанных в инфиксной нотации. Код программы приведён в приложении.

3 Выводы

В ходе работы был изучен функциональный подход к программированию, который значительно отличается от стандартного императивного подхода. Изучены некоторые основные алгоритмы, используемые в функциональном программировании и произведена работа с ними.

4 Приложение

4.1 Код lab1.hpp

```
#ifndef COMPUTATIONAL_MATH_LAB1
#define COMPUTATIONAL_MATH_LAB1

#include <iosfwd>
#include <vector>

#define CUSTOM_EPSILON 1e-17

struct point_t
{
  double x;
  double y;
};

struct function_t
{
  std::vector<double> x;
  std::vector<double> y;

  function_t();
  function_t(size_t n);

  size_t size() const;
  double* X();
  double* Y();
};
std::istream& operator>>(std::istream& in, function_t& function);
std::ostream& operator<<(std::ostream& out, const function_t& function);

```

```
30 struct spline_t
31 {
32   std::vector<double> b;
33   std::vector<double> c;
34   std::vector<double> d;
35
36   spline_t(size_t n);
37   double* B();
38   double* C();
39   double* D();
40   size_t size() const;
41 };
42 std::ostream& operator<<(std::ostream& out, const spline_t& factors);
43
44
45
46 #endif // COMPUTATIONAL_MATH_LAB1
```

4.2 Код lab1.cpp

```
1  #include <iostream>
2  #include <iomanip>
3  #include <algorithm>
4  #include <iterator>
5
6  #include "../cmath.h"
7  #include "lab1.hpp"
8
9  double RightPart(double x)
10 {
11   return 1.8 * x * x;
12 }
13
14 double bisection(function_t& function, spline_t& factors,
15     double leftX, double rightX, double (*rightPart)(double))
16 {
17   double middleX = -1;
18   double middleY = -4;
19   int last = 5;
20
21   while (abs(middleY) > CUSTOM_EPSILON)
22   {
23     middleX = (leftX + rightX) / 2;
24     middleY = seval(function.size(), middleX,
25         function.X(), function.Y(),
26         factors.B(), factors.C(), factors.D(), &last) -
27         rightPart(middleX);
28
29     double rightY = seval(function.size(), rightX,
30         function.X(), function.Y(),
31         factors.B(), factors.C(), factors.D(), &last) -
32         rightPart(rightX);
33     double leftY = seval(function.size(), leftX,
34         function.X(), function.Y(),
35         factors.B(), factors.C(), factors.D(), &last) -
36         rightPart(leftX);
37
38     if (leftY * rightY > 0)
39     {
40       std::cout << leftY << " " << rightY << "\n";
41       throw std::runtime_error("LeftY and RightY must be of different signs.");
42     }
```

```
43      if (leftY * middleY < 0)
44      {
45        rightX = middleX;
46      }
47      else if (middleY * rightY < 0)
48      {
49        leftX = middleX;
50      }
51      else
52      {
53        throw std::runtime_error("middleY must have different sign with one of the
      border.");
54      }
55    }
56    return middleX;
57  }
58
59
60
61  int main()
62  {
63    function_t function;
64    std::cin >> function;
65    std::cout << "Original function f(x):\n" << function << "\n";
66
67    spline_t factors(function.size());
68
69    int flag = 0;
70    spline(function.size(), 0, 0, 0, 0, function.X(), function.Y(), factors.B(),
        factors.C(), factors.D(), &flag);
71    if (flag != 0)
72    {
73      std::cout << "Error occured in spline function.\n";
74      return 1;
75    }
76    std::cout << "Spline coefficients:\n" << factors << "\n";
77
78    function_t newFunction(function.size());
79    std::copy(function.x.begin(), function.x.end(), newFunction.x.begin());
80    int last = 0;
81    std::transform(newFunction.x.begin(), newFunction.x.end(), newFunction.y.begin(),
82        [&](double x){
83            return seval(function.size(), x,
84              function.X(), function.Y(),
85              factors.B(), factors.C(), factors.D(), &last) -
86              RightPart(x);
87          });
88
89    std::cout << "Function given by equation f(x) = rightPart(x)\n" << newFunction << "
        \n";
90
91    std::vector<double> ans;
92    for (int i = 1; i < newFunction.size(); ++i)
93    {
94      if (newFunction.y[i] * newFunction.y[i - 1] < 0)
95      {
96        ans.push_back(bisection(function, factors,
97            newFunction.x[i - 1], newFunction.x[i], RightPart));
98      }
99    }
100
101   std::cout << "Number of roots: " << ans.size() << "\n";
102   std::copy(ans.begin(), ans.end(), std::ostream_iterator<double>(std::cout, " "));
```

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

  return 0;
}
```

4.3 Код lab1_impl.cpp

```
#include "lab1.hpp"

#include <iostream>
#include <iterator>
#include <algorithm>
#include <functional>

std::istream& operator>>(std::istream& in, point_t& point)
{
  in >> point.x >> point.y;
  return in;
}
std::ostream& operator<<(std::ostream& out, const point_t& point)
{
  out << point.x << " ; " << point.y << "\n";
  return out;
}
function_t::function_t()
{}
function_t::function_t(size_t n):
  x(n),
  y(n)
{}
size_t function_t::size() const
{
  return x.size();
}
double* function_t::X()
{
  return x.data();
}
double* function_t::Y()
{
  return y.data();
}

std::istream& operator>>(std::istream& in, function_t& function)
{
  std::vector<point_t> points(std::istream_iterator<point_t>(std::ref(in)),
      std::istream_iterator<point_t>());
  std::transform(points.begin(), points.end(), std::back_inserter(function.x), [](
    const point_t& point){return point.x;});
  std::transform(points.begin(), points.end(), std::back_inserter(function.y), [](
    const point_t& point){return point.y;});
  return in;
}
std::ostream& operator<<(std::ostream& out, const function_t& function)
{
  std::transform(function.x.begin(), function.x.end(), function.y.begin(),
      std::ostream_iterator<point_t>(out), [](double x, double y){return point_t{x, y
    };});
  return out;
}
spline_t::spline_t(size_t n):
  b(n),
```

```
53    c(n),
54    d(n)
55  {
56  }
57  double* spline_t::B()
58  {
59    return b.data();
60  }
61  double* spline_t::C()
62  {
63    return c.data();
64  }
65  double* spline_t::D()
66  {
67    return d.data();
68  }
69  size_t spline_t::size() const
70  {
71    return b.size();
72  }
73  std::ostream& operator<<(std::ostream& out, const spline_t& factors)
74  {
75    for (int i = 0; i < factors.size(); ++i)
76    {
77      out << "spline " << i << ": " << factors.b[i] << "; " << factors.c[i] << "; " <<
      factors.d[i] << "\n";
78    }
79    return out;
80  }
```