Labaratory №2

Beernadsky Gregory, Komarik Zakhar, Shnaider Ksenia

1 Tasks

1.1

Разработать подпрограмму генерации регулярных и адаптивных сеточных разбиений произвольного отрезка $[a, b]$ в зависимости от числа сегментов разбиения и величины коэффициента разрядки r

1.2

Кусочно-полиномиальная интерполяция. Разработать класс, реализующий интерфейс кубического интерполяционного сплайна с непрерывными первой и второй производными и удовлетворяющего краевым условиям нулевой кривизны $S''(a) = S''(b) = 0$.

1.3

Для набора аналитических функций $f(x) = x, x^2, x^3, x^4, sin(x)$ провести исследования на вложенных сетках. Для этого задайте шаг h и постройте равномерное сеточное разбиение отрезка $[a, b]$. Получите таблицу значений сплайна и его двух первых производных в точках, которые НЕ совпадают с узловыми (не менее 10). Повторить данные исследования на сетках с шагом $h/2$ и $h/4$. Полученные результаты сопоставьте с аналитической оценкой точности сплайн-аппроксимации: если $f(x) \in C^{k+1}[a, b]$, $0 \le k \le 3$, то для интерполяционного сплайна $S(x)$ выполнено $max_{x\in[a,b]}|f(^{(m)}(x) - S^{(m)}(x)| \le Ch^{k+1-m}max_{x\in[a,b]}|f^{(m)}(x)|, C - const$ т.е. погрешность аппроксимации ограничена сверху величиной $O(h^{k+1-m})$ при ограниченной m-ой производной аппроксимируемой функции. В отчёте привести величину шага h и соответствующую норму погрешности аппроксимации.

1.4

Выяснить как влияет на вторую производную сгущение сетки к концам отрезка $[a, b]$.

1.5

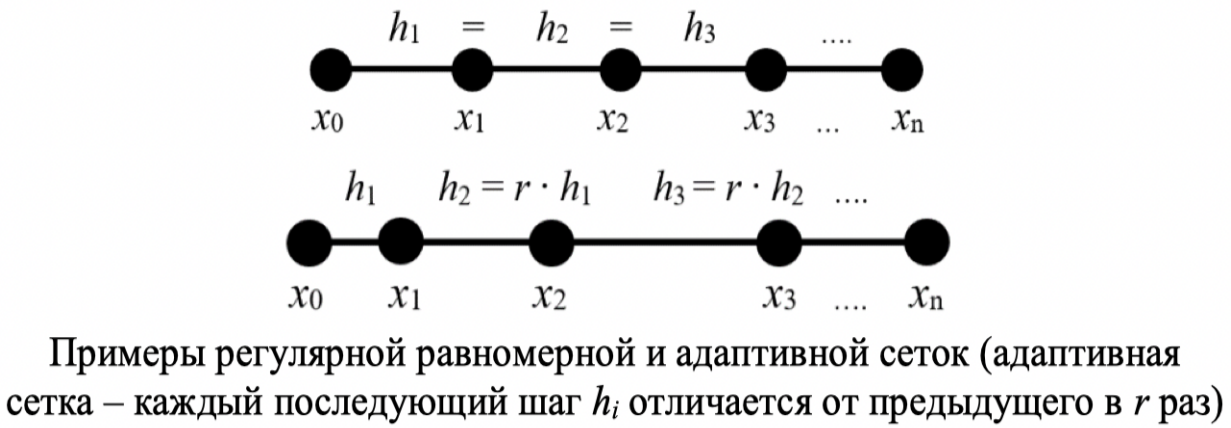
Сглаживание и аппроксимация МНК. Разработать класс, реализующий интерфейс сглаживающего сплайна. На каждом сегменте разбиения использовать базисную систему финитных функций первого порядка. Сглаживающий сплайн $g(x)$ строить как решение задачи о минимизации функционала в линейном подпространстве $\Omega \subset C[a, b] \quad \Phi = (1 - p)||f(x) - g(x)||_2^2 + p||g'(x)||_2^2$, где p – параметр сглаживания.

1.6

Для сильно осциллирующей функции $f(x) = x|sin(10000x)|$, x – радианы, на одной диаграмме изобразить интерполяционный и сглаживающий сплайны. Параметр сглаживания p варьировать от 0 до 1. Использовать равномерную сетку с шагом h и $h/2$.

1.7

Выяснить, на что влияет варьирование весовых коэффициентов в дискретном скалярном произведении при построении сглаживающего сплайна.



Примеры регулярной равномерной и адаптивной сеток (адаптивная сетка – каждый последующий шаг h_i отличается от предыдущего в r раз)

Рис. 1: Взято из учебного пособия

2 Solutions

2.1

1. Spliting.hpp

```
1  template <typename T = double>
2  std::vector<Point<T>> spliting(T first,
3                                 T second,
4                                 size_t amount,
5                                 T r = 1.,
6                                 bool dir = true)
7  {
8      auto left{std::min(first, second)}, right{std::max(first, second)};
9      std::vector<Point<T>> grid;
10     grid.recerve(amount);
11     for (auto current = (dir ? left : right);
12         (dir ? right : left) - current > eps<T>;
13         current += (step *= r))
14             grid.emplace_back({current});
15     grid.emplace_back((dir ? right : left));
16     return grid;
17 }
```

2.2

1. Splain.hpp

```
1  #pragma once
2
3  #include <array>
4  #include <vector>
5  #include <stdexcept>
6  #include <utility>
7
8  namespace af
9  {
10 template <typename T = double> using Point = std::array<T, 3>;
```

```cpp
template <typename T = double> using SplainValue = Point<T>;

template <typename T = double> class Splain
{
  public:
    Splain()
    {
      if (!std::is_floating_point<T>::value)
        throw std::logic_error("For normal work use floating types : float, double, long double\n");
    }
    /**
     * first is anchor points
     * second is values of tables function
     */
    virtual void update(std::vector<Point<T>> const &, std::vector<T> const &) = 0;
    /**
     * first is point
     * second is tabel function values
     */
    virtual SplainValue<T> getValue(Point<T> const &) const = 0;
};
} // namespace af
```

2. CubicInterpolationSplain.hpp

```cpp
#pragma once

#include "../Splain.hpp"
#include "../../constants.hpp"
#include <algorithm>
#include <cmath>
#include <stdexcept>

namespace af
{
template <typename T = double> class CubicInterpolationSplain : public af::Splain<T>
{
  public:
    void update(std::vector<Point<T>> const &, std::vector<T> const &) override;
    SplainValue<T> getValue(Point<T> const &) const override;

  private:
    /**
     * @brief grid of splain points
     */
    std::vector<Point<T>> grid;
    /**
     * @brief coefficient [a, b, c, d] of cubic interpolation splain
     */
    std::vector<std::array<T, 4>> coefficient;
};
} // namespace af

#include "CubicInterpolationSplain.inl"
```

3. CubicInterpolationSplain.inl

```cpp
#include "CubicInterpolationSplain.hpp"

template <typename T>
void af::CubicInterpolationSplain<T>::update(std::vector<af::Point<T>> const &
    points, std::vector<T> const &fValues)
{
    size_t amountSegment = points.size();

    if (amountSegment-- <= 1)
        throw std::invalid_argument("A few amount of point!");

    this->grid.resize(amountSegment + 1);
    std::copy(points.begin(), points.end(), grid.begin());

    coefficient.resize(amountSegment);

    T current{}, next{}; // h (step)
    std::vector<T> fi(amountSegment - 1);

    // get coefficient [a, b, d] and fi
    for (size_t i = 0; i < amountSegment - 1; ++i)
    {
        current = grid[i + 1][0] - grid[i][0];
        next = grid[i + 2][0] - grid[i + 1][0];

        //  b[i] = 2 * (current + next)
        coefficient[i][1] = 2 * (current + next);

        // a[i + 1] = current
        coefficient[i + 1][0] = current;

        // d[i] = next
        coefficient[i][3] = next;

        fi[i] = 3. * ((fValues[i + 2] - fValues[i + 1]) / next - (fValues[i + 1] -
            fValues[i]) / current);
    }

    // to forward
    for (size_t i = 1; i < amountSegment - 1; ++i)
    {
        // a[i] / b[i - 1]
        auto buf = coefficient[i][0] / coefficient[i - 1][1];

        // b[i] = a[i] / b[i - 1] * d[i - 1]
        coefficient[i][1] -= buf * coefficient[i - 1][3];

        fi[i] -= buf * fi[i - 1];
    }

    // to back
    // c[amountSegment - 1] = fi[amountSegment - 2] / b[amountSegment - 2]
    coefficient[amountSegment - 1][2] =
        fi[amountSegment - 2] / coefficient[amountSegment - 2][1];
    for (size_t i = amountSegment - 3; i < amountSegment; --i)
```

```
54          // c[i + 1] = (fi[i] - c[i + 2] * d[i]) / b[i]
55          coefficient[i + 1][2] =
56              (fi[i] - coefficient[i + 2][2] * coefficient[i][3]) / coefficient[i
                ][1];
57      // c[0] = 0.0
58      coefficient[0][2] = 0.;
59
60      // coefficient of splain
61      for (size_t i = 0; i < amountSegment - 1; ++i)
62      {
63          current = grid[i + 1][0] - grid[i][0];
64          // a[1] = fValues[i]
65          coefficient[i][0] = fValues[i];
66          // b[i] = (fValues[i + 1] / fValues[i]) / current - (c[i + 1] + 2 * c[i])
                * current / 3
67          coefficient[i][1] =
68              (fValues[i + 1] - fValues[i]) / current - (coefficient[i + 1][2] + 2.
                    * coefficient[i][2]) * current / 3.;
69          // d[i] = (c[i + 1] - c[i]) / (current * 3)
70          coefficient[i][3] =
71              (coefficient[i + 1][2] - coefficient[i][2]) / (current * 3.);
72      }
73
74      // last coefficient
75      current = grid[amountSegment][0] - grid[amountSegment - 1][0];
76      // a[1] = fValues[i]
77      coefficient[amountSegment - 1][0] = fValues[amountSegment - 1];
78      // b[i] = (fValues[i + 1] / fValues[i]) / current - (c[i + 1] + 2 * c[i]) *
            current / 3
79      coefficient[amountSegment - 1][1] = (fValues[amountSegment] - fValues[
            amountSegment - 1]) / current -
80                                          2. * coefficient[amountSegment - 1][2] *
                                                current / 3.;
81      // d[i] = (c[i + 1] - c[i]) / (current * 3)
82      coefficient[amountSegment - 1][3] = -coefficient[amountSegment - 1][2] / (
            current * 3.);
83  }
84
85  template <typename T> af::SplainValue<T> af::CubicInterpolationSplain<T>::getValue
        (af::Point<T> const &point) const
86  {
87      size_t amountSegment = grid.size();
88      if (amountSegment-- <= 1)
89          throw std::invalid_argument("Grid is so small!");
90
91      for (size_t i = 0; i < amountSegment; ++i)
92      {
93          if (point[0] > grid[i][0] && point[0] < grid[i + 1][0] ||
94              fabs(point[0] - grid[i][0]) < eps<T> ||
95              fabs(point[0] - grid[i + 1][0]) < eps<T>)
96          {
97              T distance = fabs(point[0] - grid[i][0]);
98              return  {
99                          coefficient[i][0] + coefficient[i][1] * distance +
                                coefficient[i][2] * distance * distance + coefficient[
                                i][3] * distance * distance * distance,
```

```
100
101                         coefficient[i][1] + 2. * coefficient[i][2] * distance + 3.
                                * coefficient[i][3] * distance * distance,
102
103                         2. * coefficient[i][2] + 6. * coefficient[i][3] * distance
104                     };
105             }
106         }
107         throw std::domain_error("Point out of the domain");
108     }
```