|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования «Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра параллельных вычислительных технологий | | |
| Лабораторная работа № 2 | | |
| по дисциплине «Численные методы» | | |
| **Интерполяционные и сглаживающие сплайны** | | |
|  | | |
|  |  |  |
| Группа ПМИ-91 | Жарков федор |
|  |  |
|  |  |
|  |  |
|  |  |
| Преподаватели | МАРКОВ СЕГРЕЙ ИГОРЕВИЧ |
|  | Иткина Наталья Борисовна |
| Новосибирск, 2021 | | |

1. **Цель работы**

Изучить и реализовать алгоритмы построения полиномиальных интерполяционных и сглаживающих сплайнов.

1. **Задание**
2. Разработать подпрограмму генерации регулярных и адаптивных сеточных разбиений произвольного отрезка [a,b] в зависимости от числа сегментов разбиения и величины коэффициента разрядки r
3. Кусочно-полиномиальная интерполяция. Разработать класс, реализующий интерфейс кубического интерполяционного сплайна с непрерывными первой и второй производными и удовлетворяющего краевым условиям нулевой кривизны .
4. Для набора аналитических функций провести исследования на вложенных сетках. Для этого задайте шаг h и постройте равномерное сеточное разбиение отрезка К содержанию [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 , то для интерполяционного сплайна S(x) выполнено

т.е. погрешность аппроксимации ограничена сверху величиной при ограниченной m-ой производной аппроксимируемой функции. В отчёте привести величину шага h и соответствующую норму погрешности аппроксимации.

1. Выяснить как влияет на вторую производную сгущение сетки к концам отрезка [a,b].
2. 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

где p – параметр сглаживания.

1. Для сильно осциллирующей функции 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.
2. Выяснить, на что влияет варьирование весовых коэффициентов в дискретном скалярном произведении при построении сглаживающего сплайна.
3. **Программа**

Ссылка на репозиторий: <https://github.com/SoBark/Interpolation-smoothing_splines>

1. **Ход работы**
2. Подпрограмма генерации регулярного сеточного разбиения:

void RegularSplit(std::vector<Com\_Methods::Point>& Points, double a, double b, int n)

{

//if (a > b) return;

double h = (b - a) / n; //определение равномерного шага

//создание равномерного сеточного разбиения

for (int i = 0; i <= n - 1; i++)

Points.push\_back(Point{ a + h \* i,0,0 });

Points.push\_back(Point{ b,0,0 });

}

Подпрограмма генерации адаптивного сеточного разбиения:

void RegularSplit(std::vector<Com\_Methods::Point>& Points, const std::string PATH)

{

std::ifstream input(PATH);

double a, b;

int n;

if (!input.is\_open())

throw std::exception("Input file has not found...");

input >> a >> b >> n;

double h = (b - a) / n; //определение равномерного шага

//создание равномерного сеточного разбиения

for (int i = 0; i < n; i++)

Points.push\_back(Point{ a + h \* i,0,0 });

Points.push\_back(Point{ b,0,0 });

}

1. Объявление класса кубического сплайна, реализующего интерфейс сплайна:

//кубический интерполяционный сплайн

class Cubic\_Interpolation\_Spline\_1D : public Spline

{

private:

//точки сетки

std::vector<Point> Points;

//коэффициенты сплайна

std::vector<double> a, b, c, d;

public:

//обновить сплайн

void Update\_Spline(const std::vector<Point> &Points, const std::vector<double> &F\_Value) override;

//вычислить значение сплайна в точке P

void Get\_Value(const Point &P, double \* Res)const override;

};

Определение методов класса кубического сплайна:

//обновить сплайн

void Cubic\_Interpolation\_Spline\_1D::Update\_Spline(const std::vector<Point> &Points,

const std::vector<double> &F\_Value)

{

//обновление списка точек сплайна

this->Points.clear();

for (auto & elem : Points) this->Points.push\_back(elem);

//число отрезков разбиения

int Num\_Segment = Points.size() - 1;

//длина текущего отрезка и следующего

double h\_current, h\_next;

//изменение размеров векторов коэффициентов

a.resize(Num\_Segment);

b.resize(Num\_Segment);

c.resize(Num\_Segment);

d.resize(Num\_Segment);

//вектор правой части СЛАУ

std::vector<double> f(Num\_Segment - 1);

//вычисление коэффициентов

for (int i = 0; i < Num\_Segment - 1; i++)

{

//длина текущего и следующего отрезков

h\_current = Points[i + 1].x() - Points[i].x();

h\_next = Points[i + 2].x() - Points[i + 1].x();

//формируем диагональ

b[i] = 2 \* (h\_current + h\_next);

//формируем нижнюю диагональ

a[i + 1] = h\_current;

//формируем верхнюю диагональ

d[i] = h\_next;

//правая часть

f[i] = 3.0 \* ((F\_Value[i + 2] - F\_Value[i + 1]) / h\_next - (F\_Value[i + 1] - F\_Value[i]) / h\_current);

}

//метод прогонки: прямой ход

for (int j = 1; j < Num\_Segment - 1; j++)

{

b[j] -= a[j] / b[j - 1] \* d[j - 1]; //диагональ

f[j] -= a[j] / b[j - 1] \* f[j - 1]; //правая часть

}

//метод прогонки: обратный ход

c[Num\_Segment - 1] = f[Num\_Segment - 2] / b[Num\_Segment - 2];

for (int j = Num\_Segment - 2; j > 0; j--)

{

c[j] = (f[j - 1] - c[j + 1] \* d[j - 1]) / b[j - 1];

}

//добавляем краевые условия нулевой кривизны

c[0] = 0.0;

for (int i = 0; i < Num\_Segment - 1; i++)

{

h\_current = Points[i + 1].x() - Points[i].x();

a[i] = F\_Value[i];

b[i] = (F\_Value[i + 1] - F\_Value[i]) / h\_current - (c[i + 1] + 2.0 \* c[i]) \* h\_current / 3.0;

d[i] = (c[i + 1] - c[i]) / h\_current / 3.0;

}

//на последнем сегменте

h\_current = Points[Num\_Segment].x() - Points[Num\_Segment - 1].x();

a[Num\_Segment - 1] = F\_Value[Num\_Segment - 1];

b[Num\_Segment - 1] = (F\_Value[Num\_Segment] - F\_Value[Num\_Segment - 1]) / h\_current - 2.0 \* c[Num\_Segment - 1] \* h\_current / 3.0;

d[Num\_Segment - 1] = -c[Num\_Segment - 1] / h\_current / 3.0;

}

//вычислить значение сплайна в точке P

void Cubic\_Interpolation\_Spline\_1D::Get\_Value(const Point &P, double \* Res)const

{

//машинный ноль

double eps = Com\_Methods::\_\_EPS\_\_;

//число отрезков

int Num\_Segment = Points.size() - 1;

//поиск отрезка, которому принадлежит точка

for (int i = 0; i < Num\_Segment; i++)

{

if (P.x() > Points[i].x() && P.x() < Points[i + 1].x() ||

fabs(P.x() - Points[i].x()) < eps ||

fabs(P.x() - Points[i + 1].x()) < eps)

{

double diff = (P.x() - Points[i].x());

Res[0] = a[i] + b[i] \* diff + c[i] \* pow(diff, 2) + d[i] \* pow(diff, 3);

Res[1] = b[i] + 2.0 \* c[i] \* diff + 3.0 \* d[i] \* pow(diff, 2);

Res[2] = 2.0 \* c[i] + 6.0 \* d[i] \* diff;

return;

}

}

throw std::exception("The point is not found in the segments...");

}

1. Были проведенные исследования на вложенных сетках для функций . Для построения сплайнов был выбран промежуток [0,1] с шагами 0.1, 0.05 и 0.025. В результате получились следующие таблицы норм погрешности:

* для

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| h | max|f(x)-g(x)| | max|f'(x)-g'(x)| | max|f''(x)-g''(x)| | pow(h,2) |
| 0.1 | 0 | 0 | 0 | 0.01 |
| 0.05 | 0 | 0 | 0 | 0.0025 |
| 0.025 | 0 | 0 | 0 | 0.000625 |

* для

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| h | max|f(x)-g(x)| | max|f'(x)-g'(x)| | max|f''(x)-g''(x)| | pow(h,2) |  |  |  |
| 0.1 | 0.00098 | 0.057735 | 2 | 0.01 | 4 | 1.999965 | 1 |
| 0.05 | 0.000245 | 0.028868 | 2 | 0.0025 | 4.016393 | 2 | 1 |
| 0.025 | 0.000061 | 0.014434 | 2 | 0.000625 |  |  |  |

* для

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| h | max|f(x)-g(x)| | max|f'(x)-g'(x)| | max|f''(x)-g''(x)| | pow(h,2) |  |  |  |
| 0.1 | 0.00294 | 0.173205 | 6 | 0.01 | 4 | 1.999988 | 1 |
| 0.05 | 0.000735 | 0.086603 | 6 | 0.0025 | 3.994565 | 2.000023 | 1 |
| 0.025 | 0.000184 | 0.043301 | 6 | 0.000625 |  |  |  |

* для

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| h | max|f(x)-g(x)| | max|f'(x)-g'(x)| | max|f''(x)-g''(x)| | pow(h,2) |  |  |  |
| 0.1 | 0.00586344 | 0.345833 | 12 | 0.01 | 3.991885 | 1.997499 | 1 |
| 0.05 | 0.00146884 | 0.173133 | 12 | 0.0025 | 4.002398 | 1.999365 | 1 |
| 0.025 | 0.00036699 | 0.086594 | 12 | 0.000625 |  |  |  |

Также были проведены исследования на вложенной сетке для . Для исследования был выбран промежуток [0,7] c шагами 0.5, 0.25 и 0.125. Результаты представлены в таблице ниже:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| h | max|f(x)-g(x)| | max|f'(x)-g'(x)| | max|f''(x)-g''(x)| | pow(h,2) |  |  |  |
| 0.5 | 0.008294476 | 0.096549746 | 0.656986599 | 0.25 | 4.090453 | 2.026451 | 1 |
| 0.25 | 0.002027764 | 0.047644746 | 0.656986599 | 0.0625 | 4.022319 | 2.007215 | 1 |
| 0.125 | 0.000504128 | 0.023736746 | 0.656986599 | 0.015625 |  |  |  |

Из проведенных исследований видно, что с уменьшением шага сетки абсолютная погрешность аппроксимации функции и её производной, также уменьшается, причем погрешность при переходе с более грубой сетки на менее грубую уменьшается в одну и ту же величину.

1. Для построения сетки сгущающейся к концам отрезка [a,b] соединялись две адаптивных сетки построенных на промежутках [a,b/2] и [b/2, b] с коэффициентами разрядки r>1 и 1/r соответственно. Следующие результаты были получены на отрезке [0,1], с коэффициентом разрядки r=1.5, число промежутков n=20

* для ничего необычного замечено не было
* для

* для
* для
* для

Полученные результаты объясняются требованием нулевой кривизны для кубического сплайна.

Также были построены сплайны на том же промежутке, но с коэффициентом разрядки r=3.5. Таким образом точки сетки сместились ближе к концам отрезка.

* для ничего необычного замечено не было
* для
* для
* для
* для

Вывод: чем чаще и ближе к концу неудовлетворяющему условию нулевой кривизны расположены точки разбиения сетки, тем отклонение приближенного значения второй производной больше. В случае невыполнения нулевой кривизны на концах интерполируемой функции, невозможно говорить о нахождении приближенного значения второй производной интерполируемой функции методом кубического сплайна

1. Объявление класса, реализующего интерфейс сглаживающего сплайна

//сглаживающий сплайн

class Smoothing\_Spline\_1D : public Spline

{

private:

//параметр сглаживания

double SMOOTH;

//точки сетки

std::vector<Point> Points;

//коэффициенты разложения по базису

std::vector<double> alpha;

//переход на мастер элемент [-1, 1]:

//Seg\_Num - номер сегмента, Х - абсцисса, Ksi - координата на мастер-элементе

void Transition\_To\_Master\_Element(int Seg\_Num, const double &X, double &Ksi) const;

//базисные функции на [-1, 1]:

//Number - номер функции, Ksi - координата на мастер-элементе

double Basis\_Function(int Number, const double &Ksi) const;

//производные базисных функций на [-1, 1]:

//Number - номер функции, Ksi - координата на мастер-элементе

double Der\_Basis\_Function(int Number, const double & Ksi) const;

public:

//конструктор: SMOOTH - параметр сглаживания

Smoothing\_Spline\_1D(const double &SMOOTH);

//обновить сплайн

void Update\_Spline(const std::vector<Point> &Points, const std::vector<double> &F\_Value) override;

//вычислить значение сплайна в точке P

void Get\_Value(const Point &P, double \* Res)const override;

};

Определение класса:

//конструктор: SMOOTH - параметр сглаживания

Smoothing\_Spline\_1D::Smoothing\_Spline\_1D(const double &SMOOTH)

{

this->SMOOTH = SMOOTH;

}

//переход на мастер элемент [-1, 1]:

//Seg\_Num - номер сегмента, Х - абсцисса, Ksi - координата на мастер-элементе

void Smoothing\_Spline\_1D::Transition\_To\_Master\_Element(int Seg\_Num, const double &X, double &Ksi) const

{

Ksi = 2.0 \* (X - Points[Seg\_Num].x()) / (Points[Seg\_Num + 1].x() - Points[Seg\_Num].x()) - 1.0;

}

//базисные функции на [-1, 1]:

//Number - номер функции, Ksi - координата на мастер-элементе

double Smoothing\_Spline\_1D::Basis\_Function(int Number, const double &Ksi) const

{

switch (Number)

{

case 1: {return 0.5\*(1 - Ksi); break; }

case 2: {return 0.5\*(1 + Ksi); break; }

default: {throw std::exception("Error in the basis function number..."); break; }

}

}

//производные базисных функций на [-1, 1]:

//Number - номер функции, Ksi - координата на мастер-элементе

double Smoothing\_Spline\_1D::Der\_Basis\_Function(int Number, const double &Ksi) const

{

switch (Number)

{

case 1: {return -0.5; break; }

case 2: {return 0.5; break; }

default: {throw std::exception("Error in the basis function derivative number..."); break; }

}

}

//обновить сплайн

void Smoothing\_Spline\_1D::Update\_Spline(const std::vector<Point> & Points,

const std::vector<double> & F\_Value)

{

//обновление списка точек сплайна

this->Points.clear();

for (auto & elem : Points) this->Points.push\_back(elem);

//число отрезков разбиения

int Num\_Segments = Points.size() - 1;

//коэффициенты разложения по базису

alpha.resize(Num\_Segments + 1);

//диагонали матрицы

std::vector <double> a, b, c;

a.resize(Num\_Segments + 1); b.resize(Num\_Segments + 1); c.resize(Num\_Segments + 1);

//процедура для ассемблирования СЛАУ:

//Num\_Segment - номер отрезка, P - точка, F\_Val - значение табличной функции в точке, w - вес

std::function<void(int Num\_Segment, const Point &P, const double &F\_Val, const double &w)>

Assembling = [&](int i, const Point &P, const double &F\_Val, const double &w)

{

double X = P.x(), Ksi;

//переход на мастер-элемент

Transition\_To\_Master\_Element(i, X, Ksi);

//вычисление значений базисных функций на мастер-элементе

double f1 = Basis\_Function(1, Ksi);

double f2 = Basis\_Function(2, Ksi);

//внесение вкладов в матрицу СЛАУ

b[i] += (1.0 - SMOOTH) \* w \* f1 \* f1;

b[i + 1] += (1.0 - SMOOTH) \* w \* f2 \* f2;

a[i + 1] += (1.0 - SMOOTH) \* w \* f1 \* f2;

c[i] += (1.0 - SMOOTH) \* w \* f2 \* f1;

//внесение вкладов в вектор правой части СЛАУ

alpha[i] += (1.0 - SMOOTH) \* w \* f1 \* F\_Val;

alpha[i + 1] += (1.0 - SMOOTH) \* w \* f2 \* F\_Val;

};

//сборка СЛАУ по сетке: сумма вкладов от каждого сегмента разбиения

for (int i = 0; i < Num\_Segments; i++)

{

//добавление узла сетки в СЛАУ

double W = 1.0;

Assembling(i, this->Points[i], F\_Value[i], W);

Assembling(i, this->Points[i + 1], F\_Value[i + 1], W);

//вклад от сглаживания по первой производной

double h = Points[i + 1].x() - Points[i].x();

b[i] += 1.0 / h \* SMOOTH;

b[i + 1] += 1.0 / h \* SMOOTH;

a[i + 1] -= 1.0 / h \* SMOOTH;

c[i] -= 1.0 / h \* SMOOTH;

}

//метод прогонки: прямой ход

for (int j = 1; j < Num\_Segments + 1; j++)

{

//диагональ

b[j] -= a[j] / b[j - 1] \* c[j - 1];

//правая часть

alpha[j] -= a[j] / b[j - 1] \* alpha[j - 1];

}

//метод прогонки: обратный ход

alpha[Num\_Segments] /= b[Num\_Segments];

for (int j = Num\_Segments - 1; j >= 0; j--)

alpha[j] = (alpha[j] - alpha[j + 1] \* c[j]) / b[j];

}

//вычислить значение сплайна в точке P

void Smoothing\_Spline\_1D::Get\_Value(const Point &P, double \* Res)const

{

//машинный ноль

double eps = 1e-7;

//число отрезков

int Num\_Segments = Points.size() - 1;

//координата точки

double X = P.x();

//поиск отрезка, которому принадлежит точка

for (int i = 0; i < Num\_Segments; i++)

{

if (X > Points[i].x() && X < Points[i + 1].x() ||

fabs(X - Points[i].x()) < eps ||

fabs(X - Points[i + 1].x()) < eps)

{

//длина отрезка

double h = Points[i + 1].x() - Points[i].x();

//переход на местер-элемент, Ksi - координата на мастер-элементе

double Ksi;

Transition\_To\_Master\_Element(i, X, Ksi);

//вычисляем значение сплайна и производной по базисным функциям

Res[0] = alpha[i] \* Basis\_Function(1, Ksi) +

alpha[i + 1] \* Basis\_Function(2, Ksi);

Res[1] = (alpha[i] \* Der\_Basis\_Function(1, Ksi) +

alpha[i + 1] \* Der\_Basis\_Function(2, Ksi)) \* 2.0 / h;

Res[2] = 0.0;

return;

}

}

throw std::exception("The point is not found in the segments...");

1. На равномерной сетке, построенной на отрезке [-1,1] с шагами h1=0.05 и h2=0.025, были построены сплайны функции с разными параметрами сглаживания p и весовыми коэффициентами w=1.

Графики сплайнов при h1=0.05:

Графики сплайнов при h2=0.025:

1. На равномерной сетке, построенной на отрезке [-1,1] с шагом h=0.05, были построены сплайны функции с разными параметрами сглаживания p и весовыми коэффициентами w1=1 и w2=0.5.

При p=0 весовые коэффициенты не вносят изменений

При p=0.1:

При p=0.5:

При p=0.99:

1. **Выводы:**

* Из проведенных исследований видно, что с уменьшением шага сетки абсолютная погрешность аппроксимации функции и её производной, также уменьшается, причем погрешность при переходе с более грубой сетки на менее грубую уменьшается в одну и ту же величину.
* чем чаще и ближе к концу неудовлетворяющему условию нулевой кривизны расположены точки разбиения сетки, тем отклонение приближенного значения второй производной больше. В случае невыполнения нулевой кривизны на концах интерполируемой функции, невозможно говорить о нахождении приближенного значения второй производной интерполируемой функции методом кубического сплайна
* При увеличении коэффициента сглаживания график сплайна становиться более плавным, график “сжимается”, максимумы уменьшаются, минимумы увеличиваются
* При уменьшении весовых коэффициентов график сплайна “прижимается” к числовой оси, становиться более пологим.