摘要

实现了 BSpline 和 ppForm-Spline 的程序包，并调用这两个样条拟合函数和曲线。

1 设计思路

我设计了如下几个类：

- `class Function` 函数抽象基类
- `class Polynomial` 多项式类，继承自 `class Function`
- `class ppForm` ppFormSpline 插值类
- `class linear_ppForm` 用 ppForm 实现 S_2^1 样条，继承自 `class ppForm`
- `class cubic_ppForm` 用 ppForm 实现 S_3^2，继承自 `class ppForm`
- `class B_base<degree>` 模板类，用来实现任意阶任意节点上的 B-Spline 的基函数
- `class class BSpline<degree>` 模板类，实现任意阶的 BSpline S_n^{n-1}
- `plane_curve_fit` 平面上曲线的拟合，支持均匀节点和累计弧长的节点。派生了两个类，分别支持 Bspline 和 pFormSpline 拟合平面上的曲线，
- `class spherefit` 实现用 ppSpline 拟合球面上的曲线，并保证拟合后的曲线仍然落在球面上。
- `class Table` 用来解决 F 题，生成 $(t-x)_+^n$ 的 $n+2$ 阶差商表。

边界条件用枚举的形式定义

- `enum class boundaryType` 记录五种边界条件，如果不是 S_3^2 就把边界条件记作 `non`

最后我还调用了第二章作业的 `class Hermiteinterpolation` 用来求 ppForm 的分段多项式。

1.1 class Function

在抽象基类 `Function` 中，实现函数的求值和用两点法求数值一阶导数，实现数值二阶导数。

```
    class Function {
        public:
        virtual double operator() (double x) const = 0;
        virtual double derivative(double x) const {}
        double doubleDerivative(double x) const{}
    };

```

1.2 class Polynomial

在多项式类中，重载实现了多项式加、减、乘法运算，另外又实现了多项式在指定点求值，多项式求导数。

```
class Polynomial{
pubilc:
    vector<double> getcoefficents() const{}
    int Degree() const {}}
    bool notequal(const Polynomial &p1) const{}

```

```
        //实现多项式的加法
        Polynomial operator+(const Polynomial &p1 ) const{}

        //实现多项式减法
        Polynomial operator-()const{}
        Polynomial operator-(const Polynomial &p1){}

        //实现多项式数乘
        Polynomial operator*(const double &a) const{}

        //实现多项式乘法
        Polynomial operator*(const Polynomial &p1)const{}

        //多项式求值
        double operator()(double(x))const{}

        //多项式求导
        Polynomial derivative()const{}

        //三阶导数
        Polynomial thirdDerivative() const{}

        double thirdDerivative(const double &x) const{}

        //指定点求导数
        double derivative(const double &x){}

        void printToJson(const string& filename) const {}
    };
    #endif
```

1.3 ppForm Spline 的实现

1. 首先定义基类 `class ppForm`，记录边界条件、节点、在每个区间上的多项式。可以实现给定函数和节点、给定节点和节点上的函数值构造 pp-Spline。最后会输出在每一个区间上的多项式系数。

```
        class ppForm{
        protected:
            int n;                //记录节点个数
            boundaryType btype=boundaryType::non;  //记录边界条件
            vector<double> knots; //记录给定的节点
            vector<double> vals;  //记录节点上的函数值
            vector<Polynomial> pols; //每一个多项式代表了在对应编号区间上得到的插值多项式
        public:
            ppForm(const vector<double> &_knots, const Function &F):knots{_knots}
            ppForm(const vector<double> &_knots, const vector<double> &_vals):knots{_knots}, vals{_vals}

```

   成员变量中的 `vals` 会在得到多项式之后释放掉。

2. 实现 S_2^1: 只要知道节点和节点上的函数值，在用直线将点连接起来，即完成了拟合。具体的，把用直线“连起来”的想法用成员函数 `fit()` 实现。

```
        class linear_ppForm:public ppForm{
        private:
        //计算每个区间上的多项式
            void fit(){}
            public:
```

```
            linear_ppForm(){}
            linear_ppForm(const vector<double> &_knots, const Function &F):ppForm(_knots, F){}
            linear_ppForm(const vector<double> &_knots, const vector<double> &_vals):ppForm(_knots,_vals){}
        };

```

3. 实现 S_3^2 样条，在 `class cubic_ppForm` 中，用一个二维的 `vector` 向量储存要求解的系数矩阵（虽然这不是一个很好的存储稀疏矩阵的方式）。根据 lemma 3.3，可以确定 $N-2$ 个方程，再根据边界条件确定两个方程，下面推导五种边界条件对应的方程：

   - natural:
   - specific:
   - not-a-knot:
   - complete:
   - periodic:

1.4 BSpline 的实现

用 `class B_base<degree>` 实现 BSpline 的基函数。

```
        template<int degree>
        class B_base{
        protected:
            int n; //记录节点 个数
            vector<double> knots; //记录节点
            vector<Polynomial> pols;  //记录分段多项式
        public:
            B_base(){};
            B_base(const vector<double> &_knots)

            //给定节点的指标i,构造support在knots[i-1]到knots[i+d]上的d阶B样条
            vector<double> setknots(const int &index, const int & d) const{}

            vector<Polynomial> getPolynomial() const

            //构造分段多项式
            void getBase()

            //在给定节点上求值
            double operator()(const int &index) const

            //在给定节点求一阶导数,如果只求导, 不要对B_base<degree>执行getbase()
            double derivative(const int &index) const

            //3rd-derivative
            double left_thirdDerivatiev(const double &x) const

            double right_thirdDerivative(const double &x) const

```

通过模板函数 `class B_base<degree>` 实现，其中 `degree` 表示 BSpline 的阶。由于 BSpline 需要确定基函数与系数，因此如果给定 N 个节点 x_1, x_2, ..., x_N 我们需要在 x_1 左边和 x_N 右边多指定 *degree* 个节点。

```
        template<int degree>
        class BSpline{
```

```
        private:
        vector<double> knots; //记录节点 要多记录2*degree个节点
        vector<vector<double>> A; //系数矩阵
        vector<double> b; //记录节点上的函数值，最终会将基函数的系数储存在b中
        vector<vector<Polynomial>> bases; //记录基函数
        vector<Polynomial> pols;  //记录多项式
        int n;  //the number of bases
        boundaryType btype=boundaryType::non;
    };

```

最终在得到 `pols` 之后，会释放 `A`，`b` 所占用的内存。Bspline 设计了如下几个接口：

```
    BSpline(){}
    BSpline(const vector<double> &_knots, const vector<double> &vals,const boundaryType &_btype=
boundaryType::non,const double &a1=0.0, const double &a2=0.0)

    BSpline(const vector<double> &_knots, const vector<vector<Polynomial>> &base,const vector<double> &coef
)

    double calculateValue(const double &x)

    void print(const string& filename)

```

分别实现了给定节点和函数，给定节点和在节点上的函数值以及边界信息 `a_1`，`a_2`，给定任意阶 Bspline 在指点点求值。最后将结果输出每个区间段的多项式系数到文件。这里 Bspline 的五种边界条件推导和 ppForm 基本类似，就不再重复写出。

1.5 平面曲线的拟合

首先定义了基类 `class plane_curve_fit`，成员变量如下和主要的成员函数：

```
protected:
    vector<double> knots; //knots
    boundaryType btype;
    knotsType ktype;
    vector<double> x_vals;
    vector<double> y_vals;
    vector<Polynomial> polsX;
    vector<Polynomial> polsY;
public:
    //实现将对t的参数转化为累计弧长参数
    void setknots(const int &N, const double &a, const double &b, const Function &f1, const Function &f2)

    //直接取均匀参数
    void uniknots(const int &N, const double &a, const double &b, const Function &f1, const Function &f2)


```

最后会将得到的二维的点以 `json` 格式输出到文件。然后派生出两个类 `class cubic_bspline_fit:public plane_curve_fi` 和 `class cubic_ppform_fit:public plane_curve_fit` 分别实现用 BForm 和 ppForm 的 S_3^2 拟合曲线。

1.6 球面样条的拟合

对于单位球面上的简单闭曲线 γ，在一般情况下，总是能找到一个点 $p \in S^, p \notin$。在我实现的球面样条拟合中，我总是假设曲线是在单位球面上（如果不是只要做伸缩变换就可以了，这是容易在函数实例化的时候实现的），并

且不会经过南极点 p_0。因此 $S \setminus p_0$ 上有一个整体的坐标卡 $\phi:\ S \setminus p_0 \to \mathbb{R}^2$，$\phi(x,y,z) = \left(\frac{x}{1+z}, \frac{y}{1+z}\right)$ 就是球极投影。只要在 $\mathbb{R}^2$ 上拟合曲线之后，再拉回到球面上即可保证拟合的曲线仍然落在球面上。球面样条的拟合在 `class spherefit:public plane_curve_fit` 里实现。由于 B 样条拟合需要自己设置额外的节点，因此使用 ppForm-spline 在平面上拟合。

```
class spherefit:public plane_curve_fit{
private:
    vector<vector<double>> points;
    void clear(){
        points.clear();
        points.shrink_to_fit();
    }
public:
    spherefit(const int &n, const double &a, const double &b, const Function &f1, const Function &f2, const
 knotsType &_ktype)

    void cubic_ppFit(const boundaryType &_btype)
    void print(const string &filename)

```

这里构造函数给出的函数应该是经过 ϕ 作用之后的函数。比如要拟合曲线 $\gamma(t) = (a(t),\ b(t),\ c(t))$，应该实例化的函数是 $\frac{a(t)}{1+c(t)}$ 和 $\frac{b(t)}{1+c(t)}$，但是最后输出到文件的点是三维的点。