Programming Task #3: Design Documents

Aoyang Yu

December 6, 2021

In the following descriptions, mathematical constants and the corresponding programming constants will be used in a mixed style, picking the appropriate one.

For simplicity of presentation, namespace qualifications will be omitted.

Numerical Types

- `using Real = double;`.
  This is for flexibility on floating-point types. User codes *should* use `Real` instead of `double`.

class Polynomial

- Models the set $\mathbb{P}_n[T]$, which contains all polynomials with degree less than or equal to n and coefficients of type T.
- Template: `template<int Order, typename CoefType>`. The template parameters corresponds to n and T respectively.
- Public Member Functions
  1. Arithmetic operators (addition, subtraction, multiplication,)
  2. Comparison operators (equal)
  3. `template<typename T> CoefType eval(const T &x) const;`
     Type Requirement: CoefType $*$ T $\to$ CoefType
     Input: A point x of type T.
     Output: The polynomial's value at x.
  4. `Polynomial derivative() const;`
     Input: None.
     Output: The derivative of the current polynomial.

5. `friend ostream& operator<<(ostream &, const Polynomial &);`
   Print the polynomial to output stream.

6. `CoefType& operator[](size_t i);`
   `const CoefType& operator[](size_t i) const;`
   Input: An index i.
   Precondition: $0 \le i \le n$.
   Output: The coefficient of x^i of this polynomial.

class Spline

- Models the set $\mathbb{S}_{n,d}^{n-1}$, which contains splines of degree n and smoothness class $n-1$ mapping $\mathbb{R}$ to $\mathbb{R}^d$. Two forms, namely piecewise-polynomial splines and B-splines, are possible.
- Template: `template<int Dim, int Order, SplineType st>`.
  The template parameters corresponds to d, n and the form of splines respectively, where `SplineType=`{`ppForm, cardinalB`}.
- Template Specialization: `Dim = 1` and `Dim > 1` are different; `ppForm` and `cardinalB` are different. Therefore, we have four specializations. They along with the corresponding member variables are described below.

  1. `Dim=1,st=SplineType::ppForm`.
     - `int N`: the number of knots.
     - `vector<Real> t`: the knots in strict ascending order. It is 0-indexed, which means the first knot is `t[0]` and the Nth knot is `t[N-1]`.
     - `vector<Polynomial<Order, Real>> piece`: the $N-1$ pieces of polynomials, 0-indexed.
  2. `st=SplineType::ppForm`.
     - `int N`: the number of knots.
     - `vector<Real> t`: the knots in strict ascending order, 0-indexed.
     - `vector<Polynomial<Order, Vec<Dim, Real>>> piece`: the $N-1$ pieces of polynomials, with vector coefficients. 0-indexed.
  3. `Dim=1,st=SplineType::cardinalB`.
     - `vector<Polynomial<Order, Real>> Base`: $B^n_{0,\mathbb{Z}}(x)$ where $n =$ Order.
     - `int N`: the number of knots.
     - `int p`: the knots base, which means the N knots are $p+1, \dots, p+N$.
     - `vector<Real> _a`: the coefficients on B-spline bases. We have `_a[j]` corresponds to $a_{p+2-n+j}$.

Mathematically: This specialization models $S(x) = \sum_{j=p+2-n}^{p+N} a_j B_{j,\mathbb{Z}}^n(x)$.

4. `st=SplineType::cardinalB`.

   – `vector<Polynomial<Order, Real>> Base`: the B-spline base, as above.
   – `int N`: the number of knots, as above.
   – `int p`: the knots base, as above.
   – `vector<Vec<Dim, Real>> _a`: the coefficients on B-spline bases. We have `_a[j]` corresponds to $\boldsymbol{a}_{p+2-n+j}$.

   Mathematically: This specialization models $S(x) = \sum_{j=p+2-n}^{p+N} \boldsymbol{a}_j B_{j,\mathbb{Z}}^n(x)$.

- Public Member Functions
  – When `Dim = 1`
    1. `Real eval(Real x) const;`
       Get the value of the spline at x.
       Precondition: $t_1 \le x \le t_N$.

  – When `Dim > 1`
    1. `Vec<Dim, Real> eval(Real x) const;`
       Get the value of the spline at x.
       Precondition: $t_1 \le x \le t_N$.

class SplineCondition

- Template: `template<ValType>`. We define `default` to be the value produced by the default constructor of `ValType`. For numerical types like `float,double,int`, we have `default=0`.
  This template parameter enables higher-dimensional conditions to be passed.
- This is the data structure for representing a set of conditions for building a spline. It contains
  1. A number $N \in \mathbb{N}$ denoting the number of knots.
  2. A strictly ascending list $t_1, \ldots, t_N \in \mathbb{R}$ of the knots.
  3. A table of condition $c_{i,j}$:`ValType`, $i, j \ge 0$.

  The specific configuration of these data is related to the boundary conditions, and will be defined exactly in the following section.
  I don't use the `InterpCond` because spline conditions have their own characteristics and I want to start from scratch.
- Public Member Functions

- `void clear();`
  Clear the table, conceptually set $N \leftarrow 0$, $t \leftarrow$ `empty` and $c_{i,j} \leftarrow$ `default`, $\forall i, j$.

- `void setN(size_t N);`
  Set N and let $t_1, \dots, t_N \leftarrow 0$.

- `size_t getN() const;`
  Get N.

- `void setT(size_t i, Real u);`
  Set $t_i \leftarrow u$.
  Precondition: $0 \le i \le N$.

- `size_t getT(size_t i) const;`
  Get t_i.
  Precondition: $0 \le i \le N$.

- `void setC(size_t i, size_t j, const ValType &v);`
  Set $c_{i,j} \leftarrow v$.

- `const ValType& getC(size_t i, size_t j) const;`
  Get $c_{i,j}$. Asking for any unspecified entry will get `default`.

enum class BCType

- This enum class give names to several **b**oundary **c**onditions. The following is a listing of all the boundary conditions supported, along with descriptions of the corresponding `SplineCondition`. Denote the spline type by `st` and the spline order by `Order`.

  - `BCType::complete`: Complete cubic spline.
    Constraint: `Ord=3, st=ppForm,cardinalB`.
    SplineCondition:

    1. $N \leftarrow$ the number of knots.
    2. For `ppForm`, $t_i \leftarrow \text{knot}_i$ for $i = 1, \dots, N$.
       For `cardinalB`, $t_0 \leftarrow a$, indicating that the knots are $\{a + 1, \dots, a + N\}$.
    3. For $i = 1, \dots, N$, $c_{i,0} \leftarrow f(t_i)$.
    4. $c_{1,1} \leftarrow f'(t_1), c_{N,1} \leftarrow f'(t_N)$.

  - `BCType::notAknot`: Not-a-knot cubic spline.
    Constraint: `Ord=3, st=ppForm`.
    SplineCondition:

    1. $N \leftarrow$ the number of knots.
    2. For $i = 1, \dots, N$, $t_i \leftarrow \text{knot}_i, c_{i,0} \leftarrow f(t_i)$.

- `BCType::periodic`: Periodic cubic spline.
  Mathematically: $s(f;a) = f(a)$ and $s^{(j)}(f;b) = s^{(j)}(f;a)$ for $j = 0, 1, 2$.
  Constraint: `Ord=3, st=ppForm`.
  SplineCondition:

  1. $N \leftarrow$ the number of knots.
  2. For $i = 1, \ldots, N$, $t_i \leftarrow \text{knot}_i$. For $i = 1, \ldots, N-1$, $c_{i,0} \leftarrow f(t_i)$.

- `BCType::middleP`: Interpolation sites are the endpoints of the big interval and middle points of the sub-intervals.
  Mathematically: For $a \in \mathbb{Z}$, find $s \in \mathbb{S}_2^1$ with knots $\{a+1, a+2, \ldots, a+N\}$ that interpolates f at sites $\{a+1, a+1/2, a+3/2, \ldots, a+N-1/2, a+N\}$.
  Constraint: `Ord=2, st=cardinalB`.
  SplineCondition:

  1. $N \leftarrow$ the number of knots.
  2. If the knots are $a+1, \ldots, a+N$, then set $t_0 \leftarrow a$.
  3. $c_{0,0} \leftarrow f(a+1)$; $\forall i = 1, \ldots, N-1, c_{i,0} \leftarrow f(a+i+1/2)$; $c_{N,0} \leftarrow f(a+N)$.

- `BCType::linear`: Find $s \in \mathbb{S}_1^0$. In this case, boundary conditions are not needed.
  Constraint: `Ord=1, st=ppForm`.
  SplineCondition:

  1. $N \leftarrow$ the number of knots.
  2. For $i = 1, \ldots, N$, $t_i \leftarrow \text{knot}_i, c_{i,0} \leftarrow f(t_i)$.

namespace SplineBuilder

- This namespace contains the spline building routines.
- **Different from the advice given in the homework statement**, I only provide one class template named `template<int Dim, int Order, SplineType st, BCType bc> class Interpolate;`. This class supports multi-dimensional (include one-dimensional) spline interpolation. It has a specialization for each interpolation scheme defined above and is indicated by `SplineType` and `BCType`. Each of the specializations has only one member function – a static member function `create` which takes in a `SplineCondition<Vec<Dim, Real>>` and gives out a corresponding `Spline`.
  The following is a list of the specializations and their `create`'s return type.
- Specializations

  1. `class Interpolate<Dim, 3, SplineType::ppForm, BCType::complete>`
     `create` returns: `Spline<Dim, 3, SplineType::ppForm>`

2. `class Interpolate<Dim, 3, SplineType::ppForm, BCType::notAknot>`
   `create` returns: `Spline<Dim, 3, SplineType::ppForm>`
3. `class Interpolate<Dim, 3, SplineType::ppForm, BCType::periodic>`
   `create` returns: `Spline<Dim, 3, SplineType::ppForm>`
4. `class Interpolate<Dim, 1, SplineType::ppForm, BCType::linear>`
   `create` returns: `Spline<Dim, 1, SplineType::ppForm>`
5. `class Interpolate<Dim, 3, SplineType::cardinalB, BCType::complete>`
   `create` returns: `Spline<Dim, 3, SplineType::cardinalB>`
6. `class Interpolate<Dim, 2, SplineType::cardinalB, BCType::middleP>`
   `create` returns: `Spline<Dim, 2, SplineType::cardinalB>`

- What about the curve fittings? Well, since multi-dimensional interpolation is provided, we ask the user to construct a `SplineCondition` for his/her specific need of curve fittings and call the interpolation routines.
- Please note that we do not provide scalar interpolation and viewing scalars as 1-dimensional vectors. Therefore, to do scalar interpolation, please construct a `Vec<1, Real>`.