SLEEFE

1.0

Generated by Doxygen 1.9.1

Chapter 1

A library to build Subdividable Linear Equi-spaced Efficient Function Enclosures (SLEEFE)

Chapter 2

Introduction

The SLEEFE library consists of a set of C++ classes for computing and representing sleefes of univariate polynomial functions in Bernstein-Bézier form.

Sleefe provides a tightly-sandwiching, piecewise-linear upper and lower bound for a piecewise polynomial on a uniformly-spaced partition of the function domain.

The bounds are sharp for quadratic polynomials and decrease by a factor of 4 under uniform refinement.

A detailed description of sleefes can be found in the following:

- David Lutterkort. Doctoral Dissertation, Purdue University, 2000.
- David Lutterkort and Jörg Peters. Linear Envelopes for Uniform B-spline Curves. In Curves and Surfaces, St Malo, France, p. 239–246, July 1-7, 2000.
- David Lutterkort and Jörg Peters. Tight Linear Envelopes for Splines, *Numerische Mathematik*, 89(4), p. 735-748, Oct. 2001.
- Jörg Peters and Xiaobin Wu. On the Optimality of Piecewise Linear Max-Norm Enclosures Based on Slefes, International Conference on Curves and Surfaces, Saint-Malo, France, 2002.

Chapter 3

Library API

To build a sleefe for a univariate polynomial function in Bernstein-Bézier form, you need to create an instance of the class `UniSleefeBuilder` using the default constructor. For instance,

```
UniSleefeBuilder builder;
```

creates an instance named `builder` of the class class `UniSleefeBuilder`.

The next step is to call the method `build()` to compute the lower and upper components of sleefe:

```
UniSleefe build(int numberOfSegments, const std::vector<double> &coeffs) const;
```

This method takes the *number of line segments* of each sleefe component and the *Bernstein-Bézier coefficients* of a univariate polynomial function as input.

Its output is an instance of the class `UniSleefe`, which represents the lower and upper components of the sleefe computed from the given input.

The API of the class `UniSleefe` is very simple. It offers public methods to obtain the coordinates of the breakpoints (i.e., vertices) of the lower and upper components of the sleefe:

```
const std::vector<double> &lowerValues() const;
const std::vector<double> &upperValues() const;
```

There are also two methods for computing points on a piecewise-linear parametrization of the lower and upper sleefe components.

```
double lowerValueAt(double t) const;
double upperValueAt(double t) const;
```

For a given value $t \in [0, 1]$ (i.e., the univariate polynomial function domain), the two methods above provide lower and upper bounds for the value of the function at t, respectively.

Chapter 4

Example

The following program illustrates the usage of the classes `UniSleefeBuilder` and `UniSleefe`.

```
#include <Sleefe.hpp> // this is the only header file needed to use the library
#include <cstdlib>
#include <iomanip>
#include <iostream>
#include <vector>
using namespace sleefe;
int main() {
  // For this example, let us define coefficients of univariate Bezier
  // functions of degree 2 through 9. For each set of coefficients, we
  // will build  sleefes with  2,...,d segments for  the corresponding
  // function.
  std::vector<std::vector<double» bezierCoeffs = {
      {0.0, 1.0, 0.8},                                  // degree 2
      {0.0, 1.0, 0.8, -0.2},                            // degree 3
      {0.0, 1.0, 0.8, -0.2, 2.5},                       // degree 4
      {0.0, 1.0, 0.8, -0.2, 2.5, 3.5},                  // degree 5
      {0.0, 1.0, 0.8, -0.2, 2.5, 3.5, 2.0},             // degree 6
      {0.0, 1.0, 0.8, -0.2, 2.5, 3.5, 2.0, 5.2},        // degree 7
      {0.0, 1.0, 0.8, -0.2, 2.5, 3.5, 2.0, 5.2, 4.0},   // degree 8
      {0.0, 1.0, 0.8, -0.2, 2.5, 3.5, 2.0, 5.2, 4.0, 0.5} // degree 9
  };
  // To build one or more sleefes, we start by defining a builder.
  UniSleefeBuilder builder;
  std::cout « std::fixed « std::setprecision(10) « std::endl;
  for (const auto &coeffs : bezierCoeffs) {
    // Compute the degree of the Bezier function.
    const auto degree = static_cast<int>(coeffs.size()) - 1;
    // Create sleefes with 1,...,9 segments for the current function.
    for (auto nSegs = 1; nSegs <= UniSleefeBuilder::MaximumNumberOfSegments;
         ++nSegs) {
      std::cout « "Degree " « degree « ", " « nSegs
                « " segments(s): " « std::endl;
      // A sleefe is created by calling method 'build()'.
      const auto sleefe = builder.build(nSegs, coeffs);
      // Display the values of the sleefe upper component breakpoints.
      for (const auto &pt : sleefe.upperValues()) {
        std::cout « pt « "  ";
      }
      std::cout « std::endl;
      // Display the values of the sleefe lower component breakpoints.
      for (const auto &pt : sleefe.lowerValues()) {
        std::cout « pt « "  ";
      }
      std::cout « std::endl;
    }
    std::cout « std::endl;
  }
  return EXIT_SUCCESS;
}
```

To access the classes, we need to include a single header file:

```
#include <Sleefe.hpp>
```

Note that the library source code belongs to the namespace `sleefe`:

```
using namespace sleefe;
```

In function `main()`, we define Bernstein-Bézier coefficients for polynomial functions of degree 2 through 9:

```
std::vector<std::vector<double» bezierCoeffs = {
    {0.0, 1.0, 0.8},                                  // degree 2
    {0.0, 1.0, 0.8, -0.2},                            // degree 3
    {0.0, 1.0, 0.8, -0.2, 2.5},                       // degree 4
    {0.0, 1.0, 0.8, -0.2, 2.5, 3.5},                  // degree 5
    {0.0, 1.0, 0.8, -0.2, 2.5, 3.5, 2.0},             // degree 6
    {0.0, 1.0, 0.8, -0.2, 2.5, 3.5, 2.0, 5.2},        // degree 7
    {0.0, 1.0, 0.8, -0.2, 2.5, 3.5, 2.0, 5.2, 4.0},   // degree 8
    {0.0, 1.0, 0.8, -0.2, 2.5, 3.5, 2.0, 5.2, 4.0, 0.5} // degree 9
};
```

Each set of coefficients corresponds to a unique polynomial function of fixed degree. The code computes sleefes with varying number of line segments for each function. More specifically, each iteration of the outer loop

```
for (const auto &coeffs : bezierCoeffs) {
  ...
}
```

creates `UniSleefeBuilder::MaximumNumberOfSegments` sleefes for the function defined by the current set of Bernstein-Bézier coefficients in `coeffs`. Each iteration of the inner loop

```
for (auto nSegs = 1; nSegs <= UniSleefeBuilder::MaximumNumberOfSegments; ++nSegs) {
  ...
}
```

computes one sleefe whose components have exactly `nSegs` line segments each. Currently, the value of the constant `UniSleefeBuilder::MaximumNumberOfSegments` is limited to 9.

Observe that the instance of the class `UniSleefeBuilder` is created before the outer loop is reached:

```
UniSleefeBuilder builder;
```

This is because we only need one builder. All sleefes are created with this single builder instance inside the inner loop:

```
for (auto nSegs = 1; nSegs <= UniSleefeBuilder::MaximumNumberOfSegments; ++nSegs) {
  ...
  const auto sleefe = builder.build(nSegs, coeffs);
  ...
 }
```

After building the sleefe, the bounds associated with its lower and upper components are obtained by calling methods from the `UniSleefe` class:

```
sleefe.upperValues()
```

and

```
sleefe.lowerValues()
```

The code simply writes the collected bounds to the standard output:

```
for (const auto &pt : sleefe.upperValues()) {
  std::cout « pt « "  ";
}
...
for (const auto &pt : sleefe.lowerValues()) {
  std::cout « pt « "  ";
}
```

The example in this section is part of the library source code distribution. You can also take a look at the unit tests that are also part of the same distribution to see more examples of the classes API usage.

Chapter 5

License

MIT License

Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 7

Class Documentation

7.1 sleefe::UniSleefe Class Reference

A class representing sleefes of univariate Bezier polynomial functions.

`#include <UniSleefe.hpp>`

Public Member Functions

- UniSleefe (const std::vector< double > &lower, const std::vector< double > &upper)

  Creates a sleefe for a univariate function.
- double lowerValueAt (double t) const

  Computes the value of the lower component of the sleefe at a given parameter value.
- double upperValueAt (double t) const

  Computes the value of the upper component of the sleefe at a given parameter value.
- const std::vector< double > & lowerValues () const

  Returns the values of the lower component of the sleefe at the breakpoints.
- const std::vector< double > & upperValues () const

  Returns the values of the upper component of the sleefe at the breakpoints.
- int numberOfSegments () const

  Returns the number of segments of each sleefe component.

Private Attributes

- int _numberOfSegments
- std::vector< double > _lower
- std::vector< double > _upper

7.1.1 Detailed Description

A class representing sleefes of univariate Bezier polynomial functions.

Definition at line 11 of file UniSleefe.hpp.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 UniSleefe()

```
sleefe::UniSleefe::UniSleefe (
            const std::vector< double > & lower,
            const std::vector< double > & upper )
```

Creates a sleefe for a univariate function.

Parameters

lower	The values of the upper component of the sleefe at the breakpoints.
upper	The values of the lower component of the sleefe at the breakpoints.

Definition at line 47 of file UniSleefe.cpp.

```
49     : _lower(lower), _upper(upper) {
50   const auto numberOfValues = _lower.size();
51
52   if (numberOfValues != _upper.size()) {
53     throw std::invalid_argument("The number of values of the lower and upper "
54                                 "envelopes must be the same");
55   }
56
57   if (numberOfValues < 2) {
58     throw std::invalid_argument("The number of values of the lower and upper "
59                                 "envelopes must be greater than 1");
60   }
61
62   _numberOfSegments = static_cast<int>(numberOfValues) - 1;
63 }
```

References _lower, _numberOfSegments, and _upper.

7.1.3 Member Function Documentation

7.1.3.1 lowerValueAt()

```
sleefe::UniSleefe::lowerValueAt (
            double t ) const
```

Computes the value of the lower component of the sleefe at a given parameter value.

Parameters

t	A parameter value in the interval [0,1].

Returns

The value of the lower component of the sleefe at `t`.

Definition at line 65 of file UniSleefe.cpp.

```
65                                                {
66   if (t < 0.0 || t > 1.0) {
67     throw std::invalid_argument(
68         "The parameter value must be a number in the real line interval [0,1]");
69   }
70
71   return interpolate(_lower, _numberOfSegments, t);
72 }
```

References _lower, and _numberOfSegments.

7.1.3.2 lowerValues()

```
sleefe::UniSleefe::lowerValues ( ) const
```

Returns the values of the lower component of the sleefe at the breakpoints.

Returns

The values of the lower component of the sleefe at the breakpoints.

Definition at line 83 of file UniSleefe.cpp.

```
83 { return _lower; }
```

References _lower.

7.1.3.3 numberOfSegments()

```
sleefe::UniSleefe::numberOfSegments ( ) const
```

Returns the number of segments of each sleefe component.

Returns

The number of segments of each sleefe component.

Definition at line 87 of file UniSleefe.cpp.

```
87 { return _numberOfSegments; }
```

References _numberOfSegments.

7.1.3.4 upperValueAt()

```
sleefe::UniSleefe::upperValueAt (
            double t ) const
```

Computes the value of the upper component of the sleefe at a given parameter value.

Parameters

t	A parameter value in the interval [0,1].

Returns

The value of the upper component of the sleefe at `t`.

Definition at line 74 of file UniSleefe.cpp.

```
74                                          {
75   if (t < 0.0 || t > 1.0) {
76     throw std::invalid_argument(
77         "The parameter value must be a number in the real line interval [0,1]");
78   }
79
80   return interpolate(_upper, _numberOfSegments, t);
81 }
```

References _numberOfSegments, and _upper.

7.1.3.5 upperValues()

`sleefe::UniSleefe::upperValues ( ) const`

Returns the values of the upper component of the sleefe at the breakpoints.

Returns

The values of the upper component of the sleefe at the breakpoints.

Definition at line 85 of file UniSleefe.cpp.

```
85 { return _upper; }
```

References _upper.

7.1.4 Member Data Documentation

7.1.4.1 _lower

`std::vector<double> sleefe::UniSleefe::_lower  [private]`

The values of the lower component of the sleefe.

Definition at line 71 of file UniSleefe.hpp.

Referenced by lowerValueAt(), lowerValues(), and UniSleefe().

7.1.4.2 _numberOfSegments

```
int sleefe::UniSleefe::_numberOfSegments  [private]
```

The number of segments of each sleefe component.

Definition at line 68 of file UniSleefe.hpp.

Referenced by lowerValueAt(), numberOfSegments(), UniSleefe(), and upperValueAt().

7.1.4.3 _upper

```
std::vector<double> sleefe::UniSleefe::_upper  [private]
```

The values of the upper component of the sleefe.

Definition at line 74 of file UniSleefe.hpp.

Referenced by UniSleefe(), upperValueAt(), and upperValues().

The documentation for this class was generated from the following files:

- UniSleefe.hpp
- UniSleefe.cpp

7.2 sleefe::UniSleefeBuilder Class Reference

A class to build sleefes from Bezier coefficients of univariate polynomial functions.

```
#include <UniSleefeBuilder.hpp>
```

Public Types

- using UniSleefeTableType = std::vector< std::vector< std::vector< double > >>

Public Member Functions

- UniSleefeBuilder ()=default
- UniSleefe build (int numberOfSegments, const std::vector< double > &coeffs) const

  Builds a sleefe for a univariate Bezier function of a given degree.

Static Public Attributes

- constexpr static int MaximumDegree = 9
- constexpr static int MaximumNumberOfSegments = 9
- static const UniSleefeTableType UpperBounds
- static const UniSleefeTableType LowerBounds

Private Member Functions

- double aerp (double t, double a, double b) const

7.2.1 Detailed Description

A class to build sleefes from Bezier coefficients of univariate polynomial functions.

Definition at line 13 of file UniSleefeBuilder.hpp.

7.2.2 Member Typedef Documentation

7.2.2.1 UniSleefeTableType

```
using sleefe::UniSleefeBuilder::UniSleefeTableType = std::vector<std::vector<std::vector<double>
>>
```

Data type for the pre-tabulated upper and lower bounds.

Definition at line 46 of file UniSleefeBuilder.hpp.

7.2.3 Constructor & Destructor Documentation

7.2.3.1 UniSleefeBuilder()

```
sleefe::UniSleefeBuilder::UniSleefeBuilder ( )  [default]
```

Default constructor.

7.2.4 Member Function Documentation

7.2.4.1 aerp()

```
double sleefe::UniSleefeBuilder::aerp (
            double t,
            double a,
            double b ) const  [private]
```

Computes the affine combination of two values.

Parameters

t	The ratio of the affine combination.
a	A real value.
b	A real value.

Returns

The value $(1-t)a+tb$

Definition at line 85 of file UniSleefeBuilder.cpp.

```
85                                                                    {
86   if (t < 0.0 || t > 1.0) {
87     throw std::invalid_argument(
88         "The weight must be a number in the real line interval [0,1]");
89   }
90
91   return std::clamp((1.0 - t) * a + t * b, std::min(a, b), std::max(a, b));
92 }
```

Referenced by build().

7.2.4.2 build()

```
sleefe::UniSleefeBuilder::build (
            int numberOfSegments,
            const std::vector< double > & coeffs ) const
```

Builds a sleefe for a univariate Bezier function of a given degree.

Parameters

numberOfSegments	Number of linear pieces in the sleefe.
coeffs	Array with the Bezier coefficients of the function.

Returns

A sleefe for the given univariate function.

Definition at line 8 of file UniSleefeBuilder.cpp.

```
9                                                                                {
10   const auto degree = static_cast<int>(coeffs.size()) - 1;
11
12   if (degree < 2 || degree > MaximumDegree) {
13     throw std::invalid_argument("The degree of the univariate function "
14                                 "must be at least 2 and no more than " +
15                                 std::to_string(MaximumDegree));
16   }
17
18   if (numberOfSegments < 1 || numberOfSegments > MaximumNumberOfSegments) {
19     throw std::invalid_argument("The number of segments of the sleefe must "
```

```
20                                "be greater than 0 and no more than " +
21                                std::to_string(MaximumNumberOfSegments));
22   }
23
24   if (coeffs.size() != size_t(degree + 1)) {
25     throw std::runtime_error("The number of coefficients and the degree "
26                              "are inconsistent with each other");
27   }
28
29   // Compute the 2nd differences of the coefficients of the input function.
30   const auto numberOfBasisFunctions = degree - 1;
31   std::vector<double> secondDiffs(numberOfBasisFunctions);
32   for (auto idx = 0; idx < numberOfBasisFunctions; ++idx) {
33     secondDiffs[idx] = coeffs[idx] - 2.0 * coeffs[idx + 1] + coeffs[idx + 2];
34   }
35
36   // Save the left and right most coefficients.
37   const auto lCoeff = coeffs[0];
38   const auto rCoeff = coeffs[degree];
39
40   // Compute the values of the lower and upper sleefe components at the
41   // breakpoints.
42   std::vector<double> lowerValues(numberOfSegments + 1);
43   std::vector<double> upperValues(numberOfSegments + 1);
44
45   const auto numberOfCoefficients = numberOfSegments + 1;
46   const auto degIdx = degree - 2;
47
48   // Select pre-tabulated lower and upper bounds for the given degree and
49   // number of segments.
50   const auto &lowerBounds = LowerBounds[degIdx][numberOfSegments - 1];
51   const auto &upperBounds = UpperBounds[degIdx][numberOfSegments - 1];
52
53   // Compute a lower and upper value for each breakpoint (there are
54   // numberOfSegments+1).
55   for (auto segIdx = 0; segIdx <= numberOfSegments; ++segIdx) {
56     const auto tValue = static_cast<double>(segIdx) / numberOfSegments;
57
58     // Compute the linear contribution to the component values of the sleefe.
59     lowerValues[segIdx] = aerp(tValue, lCoeff, rCoeff);
60     upperValues[segIdx] = aerp(tValue, lCoeff, rCoeff);
61
62     // Add contribution from every coefficient of the basis functions.
63     for (auto bsfIdx = 0; bsfIdx < numberOfBasisFunctions; ++bsfIdx) {
64       // Computes the index of the first coefficient of the current basis
65       // function.
66       const auto offset = bsfIdx * numberOfCoefficients;
67
68       if (secondDiffs[bsfIdx] > 0) {
69         lowerValues[segIdx] +=
70             lowerBounds[offset + segIdx] * secondDiffs[bsfIdx];
71         upperValues[segIdx] +=
72             upperBounds[offset + segIdx] * secondDiffs[bsfIdx];
73       } else {
74         lowerValues[segIdx] +=
75             upperBounds[offset + segIdx] * secondDiffs[bsfIdx];
76         upperValues[segIdx] +=
77             lowerBounds[offset + segIdx] * secondDiffs[bsfIdx];
78       }
79     }
80   }
81
82   return UniSleefe(lowerValues, upperValues);
83 }
```

References aerp(), LowerBounds, MaximumDegree, MaximumNumberOfSegments, and UpperBounds.

7.2.5 Member Data Documentation

7.2.5.1 LowerBounds

```
const UniSleefeBuilder::UniSleefeTableType sleefe::UniSleefeBuilder::LowerBounds  [static]
```

Table of lower bounds of sleefes for univariate functions.

Definition at line 52 of file UniSleefeBuilder.hpp.

Referenced by build().

7.2.5.2 MaximumDegree

```
constexpr static int sleefe::UniSleefeBuilder::MaximumDegree = 9  [static], [constexpr]
```

Maximum degree of a polynomial basis function.

Definition at line 40 of file UniSleefeBuilder.hpp.

Referenced by build().

7.2.5.3 MaximumNumberOfSegments

```
constexpr static int sleefe::UniSleefeBuilder::MaximumNumberOfSegments = 9  [static], [constexpr]
```

Maximum number of linear segments of a sleefe.

Definition at line 43 of file UniSleefeBuilder.hpp.

Referenced by build().

7.2.5.4 UpperBounds

```
const UniSleefeBuilder::UniSleefeTableType sleefe::UniSleefeBuilder::UpperBounds  [static]
```

Table of upper bounds of sleefes for univariate functions.

Pre-tabulated upper bounds: indexed by degree and number of line segments.

Definition at line 49 of file UniSleefeBuilder.hpp.

Referenced by build().

The documentation for this class was generated from the following files:

- UniSleefeBuilder.hpp
- UniSleefeBuilder.cpp
- UniSleefeTables.cpp

Index