- Home
- **Tools**
  - Option Pricer
  - Share Sim
  - Browser Check
- OpenSource
  - Spline
  - Black Scholes
  - Poisson Solver
- Documents
  - Notes
  - PhD Thesis
  - Diploma Thesis
- Gallery
  - Astronomy

Cubic Spline interpolation in C++

Aims

- simple to use and requiring no dependencies
- simple implementation for easy extention/modification
- efficient: O(N) to generate spline, O(log(N)) to evaluate spline at a single point, where N is the number of input data points
- current implementation: natural boundary conditions (2nd derivatives are zero) and linear extrapolation
- TODO: sort input vector, implement an efficient algorithm to evaluate spline(X) where X is a vector, ...

Download Source Code

It is implemented as a single header file:

- spline.h (released under the GPLv2 or above)
- latest developer version is available at github ttk592/spline

Usage

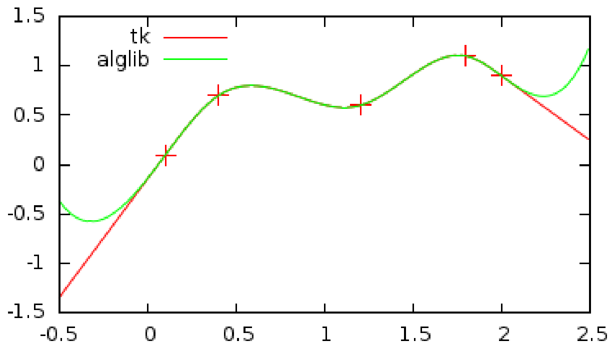
```
#include <cstdio>
#include <cstdlib>
#include <vector>
#include "spline.h"

int main(int argc, char** argv) {

   std::vector<double> X(5), Y(5);
   X[0]=0.1; X[1]=0.4; X[2]=1.2; X[3]=1.8; X[4]=2.0;
   Y[0]=0.1; Y[1]=0.7; Y[2]=0.6; Y[3]=1.1; Y[4]=0.9;

   tk::spline s;
   s.set_points(X,Y);    // currently it is required that X is already sorted

   double x=1.5;

   printf("spline at %f is %f\n", x, s(x));

   return EXIT_SUCCESS;
}
```

Compile:

```
$ g++ -Wall demo.cpp -o demo
$ ./demo
spline at 1.500000 is 0.915345
```

Comparison

The result of this library is compared against alglib. Interpolation results are identical but extrapolation differs, as this library is designed to extrapolate linearly.



Source:

- example.cpp
- example_alglib.cpp

Benchmark

A simple benchmark shows that performance is comparable with alglib. The following operations are measured:

- Spline creation: calculate the coefficients of a spline given input vectors X and Y,
- Random access: evaluate spline(x), i.e. y-value of the spline at a random point x,
- Grid transform: given vectors X1,Y1, and a new vector X2, then we want to calculate the corresponding Y2 vector, i.e. Y2[i]=spline(X2[i]). Alglib outperforms here, because this operation can be implemented more efficiently (TODO list).

The output "cycl" is the number of cpu cycles required for a single operation.

```
$ ./bench
usage: ./bench <cpu mhz> <spline size> <loops in thousand>

$ ./bench 2666 10 10000     # splines based on 10 grid points, 10million loops
                                tk                      alglib
random access:   loops=1e+07,   0.198s (  53 cycl)      0.400s ( 107 cycl)
spline creation: loops=1e+06,   2.279s (6075 cycl)      3.566s (9507 cycl)
grid transform:  loops=1e+06,   2.685s (7157 cycl)      3.356s (8948 cycl)

accuracy: max difference = 5.55e-16, l2-norm difference = 2.46e-20

$ ./bench 2666 10000 10000  # splines based on 10000 grid points, 10million loops
                                tk                      alglib
random access:   loops=1e+07,   1.370s ( 365 cycl)      1.538s ( 410 cycl)
spline creation: loops=1e+03,   1.726s (4.6e+06 cycl)   1.691s (4.5e+06 cycl)
grid transform:  loops=1e+03,   2.258s (6.0e+06 cycl)   1.414s (3.8e+06 cycl)

accuracy: max difference = 4.41e-13, l2-norm difference = 7.85e-19
```

Source:

- bench.cpp

Compile as follows where the alglib source and compiled lib needs to be in the appropriate directory:

```
$ g++ -Wall -O2 -I../alglib/src -c bench.cpp -o bench.o
$ g++ bench.o -o bench -L../alglib/ -lalglib
```

Maths

Given a set of inputs $\{(x_1, y_1), \dots, (x_n, y_n)\}$ with $x_1 < x_2 < \dots < x_n$. Define piecewise cubic polynomials $f_1, \dots, f_{n-1}$ with $f_i : [x_i, x_{i+1}] \to \mathbb{R}$ by

$$f_i(x) := a_i(x-x_i)^3 + b_i(x-x_i)^2 + c_i(x-x_i) + y_i, \qquad x \in [x_i, x_{i+1}], \tag{1}$$

with derivatives

$$\begin{aligned} f_i'(x) &= 3a_i(x-x_i)^2 + 2b_i(x-x_i) + c_i, \\ f_i''(x) &= 6a_i(x-x_i) + 2b_i. \end{aligned}$$

In order for the overall function to be twice continuously differentiable we require

$$
\begin{aligned}
f_i(x_{i+1}) = y_{i+1}: \quad & a_i h_i^3 + b_i h_i^2 + c_i h_i = y_{i+1} - y_i, \\
f'_{i-1}(x_i) = f'_i(x_i): \quad & 3a_{i-1}h_{i-1}^2 + 2b_{i-1}h_{i-1} + c_{i-1} = c_i, \\
f''_{i-1}(x_i) = f''_i(x_i): \quad & 6a_{i-1}h_{i-1} + 2b_{i-1} = 2b_i,
\end{aligned}
$$

with $h_i := x_{i+1} - x_i$. This gives $3(n-1)$ equations which can be simplified by first expressing a and c in terms of b and then solving the remaining equation system for b: from the continuity of the second derivative it follows

$$
f''_i(x_{i+1}) = f''_{i+1}(x_{i+1}) \quad \Rightarrow \quad a_i = \frac{b_{i+1} - b_i}{3h_i}, \tag{2}
$$

$$
f_i(x_{i+1}) = y_{i+1} \quad \Rightarrow \quad c_i = \frac{y_{i+1} - y_i}{h_i} - \frac{1}{3}(2b_i + b_{i+1})h_i, \tag{3}
$$

$$
f'_{i-1}(x_i) = f'_i(x_i) \quad \Rightarrow \quad \frac{1}{3}h_{i-1}b_{i-1} + \frac{2}{3}(h_{i-1} + h_i)b_i + \frac{1}{3}h_i b_{i+1} = \frac{y_{i+1} - y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}. \tag{4}
$$

First solve the tridiagonal equation system (4) then using (2) and (3) to get the remaining coefficients. For extrapolation we use second order polynomials (as we have one condition less). From the continuity conditions it follows:

$$
\begin{aligned}
f_0(x) &:= b_1(x - x_1)^2 + c_1(x - x_1) + y_1, \quad x \le x_1, \\
f_n(x) &:= b_n(x - x_n)^2 + c_n(x - x_n) + y_n, \quad x \ge x_n.
\end{aligned}
$$

By defining f_n, the right boundary condition simplifies. For example for zero curvature at x_1 and x_n we require $b_1 = 0$ and $b_n = 0$.

Math symbols are rendered by MathJax which requires JavaScript.

xhtml