三次样条曲线插值的基本原理及其 C#实现

作　者:　simxpert

2018.5.14

作者简介：

本人长期从事结构强度仿真计算方面的学习和工作，可以为您解决各种结构强度仿真分析以及专业计算软件定制方面的工作。

扫描二维码可添加本人微信：

目　　录

三次样条曲线插值的数学原理及其 C#实现

作者: simxpert

1. 简介

在工程实践中经常会用到插值，最简单的插值是在相邻的两个样本点之间线性插值，有时候希望精度高一点，一般都会用三次样条曲线插值。

本文先从理论上讲解三次样条插值的数学原理，然后讲解使用 C#实现三次样条插值算法的基本过程，最后以实例对算法进行验证。

2. 三次样条插值的基本数学原理

本文并不打算详细讲解完整的三次样条曲线的推导过程，只是梳理一下三次样条曲线差值的大概的脉络，详细的推导过程可以参考随意一本数值计算教程。

2.1. 插值的问题的提出

在给定的区间[a,b]上有 N 个点，已知其横坐标为：$a=x_0<x_1<\ldots\ldots<x_{n-1}<x_n=b$，对应纵坐标为：$y_0,y_1,\ldots\ldots y_{n-1},y_n$。

这 N 个点的每两个相邻点 x_i,x_{i+1} 构成一个子区间$[x_i,x_{i+1}]$，[a,b]之间共计有 n 个子区间：$[x_0,x_1],\ldots,[x_{n-1},x_n]$ 。(N=n+1)

问题：根据上述已知条件，任意给定一个在 x_t，$x_t \in [a,b]$，如何近似求得一个合理的 y_t 值？

一个最简单易行的办法就是线性插值：在每个子区间$[x_i,x_{i+1}]$中，把两个端点用直线连接起来，可以由这段直线的方程 $S_i(x)$来求这个子区间上任意一点的纵坐标值。我们只需要确定给定的点 x_t 落在哪个子区间就可以了。

线性插值的缺点很明显：首先是精度低。其次，整个曲线不光滑。因为在整个区间的插值函数 S(x)是由多段的直线 $S_i(x)$连接而成，在每个拐点处连接是不平滑，从数学上看也就是在拐点处的一阶导数不连续，更别提二阶段导数的连续性了。

2.2. 插值函数的待定系数变量和约束方程

为了得到精度比较高而且过渡比较平滑的曲线，最常见的就是使用三次样条插值。如果我们把前面讲的子区间的线性函数提升为 3 次多项式：

$$S_i(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \tag{1}$$

为了保证曲线的连续性和光滑性，必须要对每个区间的插值函数提出一些要求，这些要求包括：

1).连续性。除了两个边界节点外，任意一个节点 x_i 同时属于$[x_{i-1},x_i]$和$[x_i,x_{i+1}]$这两个相邻的子区间。我们必须要保证在用这两个区间上的插值函数计算出来的 x_i 的纵坐标值相等，而且要等于已知的 yi 值。即：

$$S_{i-1}(x_i)=S_i(x_i)=y_i,i=1,2,\ldots,n-1。 \tag{2}$$

式(2)包含了 2n-2 个方程，再加两个边界点上 $S_0(x_0)=y_0,S_{n-1}(x_n)=y_n$，共计得到 2n 个方程。

2).一阶导数和二阶导数连续。这个约束条件是为了保证插值函数的曲线足够的光滑。除了两头的节点外，任意一个中间节点的一阶导数和二阶导数必须连续。即：

$$S'_{i-1}(x_i) = S'_i(x_i),i=1,2,\ldots,n-1 \tag{3}$$

$$S''_{i-1}(x_i) = S''_i(x_i),i=1,2,\ldots,n-1 \tag{4}$$

式(3),(4)可以得到 2n-2 个方程。

由于上述约束条件总计可以得到 4n-2 个方程。从(1)式可以看出，每个插值函数有 4 个未知的待定系数，[a,b]区间内有 n 个子区间，故一共有 4n 个未知待定系数。未知的变量的数目比约束方程的数目多 2，也就是说我们还缺少两个约束条件。

缺少的这两个约束条件可以由边界条件来指定。也就是在两个边界点上各施加一个边界条件。边界条件是是人为指定的，常见的边界条件有 4 种，这个我们

在后面会提到。

2.3. 插值函数的最终表达式

根据前面的论述，我们只要求出每个子区间的插值函数的 4 个系数 a_0, a_1, a_2,a_3 就能完全确定这 n 个插值函数。但是实际中，为了数学推导的方便，我们并不会直接求出这 4 个变量，而是采取了间接的表达方式来描述每个插值函数。

不同的推导方法，得到的表达形式也各不相同，常见的有三转角法，三弯矩法，B 样条基函数法等方法。本文并不打算进行详细的推导。因为没必要，任何一本数值计算的书上都可以找到非常详细的。在这里我直接给出推导结果:

$$S_i(x) = y_i\varphi_0(\frac{x - x_i}{h_i}) + y_{i+1}\varphi_0(\frac{x_{i+1} - x}{h_i}) + m_i h_i\varphi_1(\frac{x - x_i}{h_i}) - m_{i+1}h_i\varphi_1(\frac{x_{i+1} - x}{h_i}) \quad (5)$$

其中 $x \in [x_i, x_{i+1}]$, $h_i = x_{i+1} - x_i$, $i = 0,1,\ldots,n-1$。

$$\varphi(x) = (2x + 1)(x - 1)^2, \varphi_1(x) = x(x - 1)^2 \quad (6)$$

式(5)就是插值函数的最终表达式。式(5)看起来比(1)复杂多了，其实仔细看，如果全部展开，仍然是一个三次多项式。在这个公式中，只有 m_i 是未知的，其他都可以根据已知条件直接算出来的。剩下的任务就是如何求出 m_i。

我们把(5)所代表的插值函数代入到之前说的约束条件中(2),(3),(4)中，经过整理，可以得到式(7)：

$$\lambda_i m_{i-1} + 2m_i + \mu_i m_{i+1} = g_i, \quad i=1,2,\ldots n-1. \quad (7)$$

其中 $\lambda_i = \frac{h_i}{h_i + h_{i-1}}$, $\mu_i = 1 - \lambda_i$, $g_i = 3(\lambda_i \frac{y_i - y_{i-1}}{h_{i-1}} + \mu_i \frac{y_{i+1} - y_i}{h_i})$。

2.4. 添加边界条件求解 m_i

前面讲过，我们还需要添加两个边界条件才能求解出位置系数。常见的边界

条件大概有三、四种。本文不打算涉及所有的边界条件，只选用比较常用的第二类边界条件(又叫"自然边界条件")：两个边界点的二阶导数为给定的值，其中最常见给定值为 0，即在两个边界点上有：

$$S''_0(x_0) = 0, \quad S''_{n-1}(x_n) = 0 \tag{8}$$

根据式(7)，结合边界条件(8)，可以推导出求解系数 m_i 的矩阵：

$$\begin{bmatrix} 2 & \mu_1 & & & & & \\ \lambda_2 & 2 & \mu_2 & & & & \\ & \lambda_3 & 2 & \mu_3 & & & \\ & & \lambda_4 & 2 & \ddots & & \\ & & & \ddots & \ddots & \ddots & \\ & & & & \lambda_{n-2} & 2 & \mu_{n-1} \\ & & & & & \lambda_{n-1} & 2 \end{bmatrix} \begin{bmatrix} m_1 \\ m_2 \\ m_3 \\ \vdots \\ \vdots \\ m_{n-2} \\ m_{n-1} \end{bmatrix} = \begin{bmatrix} g_1 - \lambda_1 m_0 \\ g_2 \\ g_3 \\ \vdots \\ \vdots \\ g_{n-2} \\ g_{n-1} \end{bmatrix} \tag{9}$$

从上面的公式中 λ，μ，g 都是可以根据已知条件算出来的，只有 m_i 是未知量。式(9)中的矩阵是一个典型的严格对角占优的对角方程组，可以用追赶法很快得到解。

2.5. 根据插值函数进行插值计算

在根据 2.4 章中的(9)式求得 m_i 之后，代入到 2.3 章中的式(5)，即得到所有的子区间上的插值函数 $S_i(x)$。

对于一个给定的待插值点 x_t，我们先要确定 x_t 落入到哪个子区间，然后调用该子区间的插值函数 $S_i(x)$即可求出 x_t 对应的插值 y_t。

3. 三次样条插值函数的 C#实现

在 C#中实现了三次样条插值函数，实现过程封装在 SPLine 类中。SPLine 类的成员变量为：

```
class SPLine
{
double[] Xi;//存储已知点的x值
double[] Yi;//存储已知点的y值；
double[] A;//存储追赶法矩阵中的A
```

```
double[] B; //存储追赶法矩阵中的B
double[] C; //存储追赶法矩阵中的C

double[] H;//存储前面公式中的hi
double[] Lamda;//存储前面公式中的 λi
double[] Mu;//存储 μi
double[] G;//存储gi
double[] M;//存储待求的未知量mi
int N;//已知点的总数
int n;//n=N-1。 n为区间数，N为点数。
}
```

成员函数介绍：

private void GetH()

功能是求取前文中的h。$h_i=x_i-x_{i-1}$。

private void GetLamda_Mu_G()

功能是根据已知条件求出 λ， μ，g。计算公式见第3页中间的公式。

private void GetABC()

功能是求出追赶法中的矩阵里面的A,B,C，至于追赶法里面的A,B,C是怎么回事，这里简单给一个截图：

$$\begin{bmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ & \ddots & \ddots & \ddots & & \\ & & a_i & b_i & c_i & \\ & & & \ddots & \ddots & \ddots \\ & & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & & & a_n & b_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_i \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_i \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

数组A，B，C分别保存的是上图中的a,b,c。

上图中的待求变量x对应本文中的待求变量m。上图中的d对应本文中的数组G。

private double fai0(double x)

private double fai1(double x)

这两个函数对应2.3章中的式(6)中的两个三次多项式。

private int GetSection(double x)

判定待求值的 x 属于哪个子区间，并返回区间编号。

有一种特殊情况：如果给定的插值点 x 不在区间[a,b]范围内，是不能应用插值函数的。基于实际项目的需要，本文中的程序对这种情况的处理方式：取距离 x 最近的边界节点的值作为其插值结果。当然也可以根据需要改成抛出"超出边界"的异常。

当 x 比最小节点 x_0 还要小，则 GetSection 返回的区间编号为-1，如果 x 比最大节点 x_n 还要大，则 GetSection 返回的区间编号为-999。相应地，如果返回-1，则 $S(x_0)=f(x_0)=y_0$,如果返回-999 则 $S(x_0)=f(x_{n-1})=y_{n-1}$

public double Interpolate(double x)

该函数为计算 x 的插值的插值函数。计算公式依据的是第 2.3 章中的式(6)。

public bool Init(double[] Xi, double[] Yi)

该函数为初始化函数。把[a,b]区间的(xi,yi)作为输入数据，初始化 SPLine 类的成员变量，并调用追赶法 Chase.Solve()求出待定系数 m_i。

在 SPLine 类中引用了追赶法求解代数方程组的 Chase 类，Chase 类也是自己编写的专门求解形如式(9)的方程组的。

SPLine 类和 Chase 类的代码在附录中给出。

4. SPLine 类的使用方法

首先当然是把 SPLine.cs 和 Chase.cs 这两个个代码源文件加入到 solution 中，然后在要使用该类的地方添加对命名空间的引用 using SPLINE; using MyAlgbra;

下面是一段验证代码，通过这段代码也同时说明了该如何使用 SPLine 类。

```
private void button2_Click(object sender, EventArgs e)
        {
          double[] X = {1,2,4,5};
          double[] Y = {1,3,4,2};
          double s;
          SPLine sp = new SPLine();
```

```
            sp.Init(X, Y);
            s = sp.Interpolate(0.5);
        }
```

5. 附录 1-SPLine 类的源代码

完整的 SPLine.cs 的代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections;
using MyAlgbra;
namespace SPLINE
{
    class SPLine
    {
        double[] Xi;
        double[] Yi;
        double[] A;
        double[] B;
        double[] C;
        double[] H;
        double[] Lamda;
        double[] Mu;
        double[] G;
        double[] M;
        int N;//
        int n;//
       public SPLine()
        {
            N = 0;
            n = 0;
        }
       public bool Init(double[] Xi, double[] Yi)
       {
           if (Xi.Length != Yi.Length)
               return false;
           if (Xi.Length == 0)
               return false;

//根据给定的Xi,Yi元素的数目来确定N的大小。
           this.N = Xi.Length;
           n = N - 1;
```

```
//根据实际大小给各个成员变量分配内存
            A = new double[N-1];
            B = new double[N];
            C = new double[N-1];

            this.Xi = new double[N];
            this.Yi = new double[N];

            H = new double[N-1];
            Lamda = new double[N-1];
            Mu = new double[N-1];

            G = new double[N];
            M = new double[N];
//把输入数据点的数值赋给成员变量。
          for (int i = 0; i <= n; i++)
            {
                this.Xi[i] = Xi[i];
                this.Yi[i] = Yi[i];
            }

/************ 求出hi,Lamda(i),Mu(i),gi ******************/
            GetH();
            GetLamda_Mu_G();
            GetABC();
/***************** 调用追赶法求出系数矩阵M ********************/
            Chasing chase = new Chasing();
            chase.Init(A, B, C, G);
            chase.Solve(out M);
            return true;
        }
        private void GetH()
         {
             //Get H first;
             for (int i = 0; i <=n-1; i++)
             {
                 H[i] = Xi[i + 1] - Xi[i];
             }
         }
        private void GetLamda_Mu_G()
         {
             double t1, t2;
             for (int i = 1; i <= n - 1; i++)
```

```
        {
            Lamda[i] = H[i] / (H[i] + H[i - 1]);
            Mu[i] = 1 - Lamda[i];

            t1 = (Yi[i] - Yi[i - 1]) / H[i - 1];
            t2 = (Yi[i + 1] - Yi[i]) / H[i];
            G[i]=3*(Lamda[i]*t1+Mu[i]*t2);
        }
        G[0] = 3*(Yi[1] - Yi[0])/H[0];
        G[n] = 3*(Yi[n] - Yi[n-1])/H[n-1];
        Mu[0] = 1;
        Lamda[0] = 0;
    }
    private void GetABC()
    {
        for (int i = 1; i <= n - 1; i++)
        {
            A[i-1] = Lamda[i];
            C[i] = Mu[i];
        }
        A[n-1] = 1; C[0] = 1;

        for (int i = 0; i <= n; i++)
        {
            B[i] = 2;
        }
    }
    private double fai0(double x)
    {
        double t1, t2;
        double s;
        t1 = 2 * x + 1;
        t2 = (x - 1) * (x - 1);
        s = t1 * t2;

        return s;
    }
    private double fai1(double x)
    {
        double s;
        s=x*(x - 1) * (x - 1);
        return s;
    }
    public double Interpolate(double x)
```

```
        {
            double s = 0;
            double P1, P2;
            double t = x;
            int iNum;

            iNum=GetSection(x);

            if (iNum == -1) //
            {
                iNum = 0;
                t = Xi[iNum];
                return Yi[0];
            }
            if (iNum == -999)//
            {
                iNum = n - 1;
                t = Xi[iNum+1];
                return Yi[n];

            }
            P1 = (t - Xi[iNum]) / H[iNum];
            P2 = (Xi[iNum + 1] - t) / H[iNum];

            s = Yi[iNum] * fai0(P1) + Yi[iNum + 1] * fai0(P2) +
M[iNum] * H[iNum] * fai1(P1) - M[iNum+1] * H[iNum] * fai1(P2);
            return s;
        }

        private int GetSection(double x)
        {
            int iNum = -1;
           // double EPS = 1.0e-6;
            if (x < Xi[0])
            {
                return -1;
            }
            if (x > Xi[N - 1])
            {
                return -999;
            }

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

```
                if (x >= Xi[i] && x <= Xi[i + 1])
                {
                    iNum = i;
                    break;
                }
            }
            return iNum;
        }
    }
}
```

6. 附录 2-Chase 类的源代码

追赶法求线性代数方程组的 Chase 类完整代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace MyAlgbra
{
    class Chasing
    {
        protected int N;//Dimension of Martrix Ax=d;
        protected double[] d;//Ax=d;
        protected double[] Aa;//a in A;
        protected double[] Ab; //b in A:
        protected double[] Ac;// c in A;
        protected double[] L;//LU-->L;
        protected double[] U;//LU-->U;
        public double[] S;//store the result;
        //constructor without parameters;
        public Chasing()
        {

        }
       public bool Init(double[] a ,double[] b,double[] c,double[] d)
        {
           //check validation of dimentions;
            int na = a.Length;
            int nb = b.Length;
            int nc = c.Length;
            int nd = d.Length;
```

```
            if(nb<3)
                return false;

            N = nb;

            if (na != N - 1 || nc != N - 1 ||nd!=N)
                return false;
            S = new double[N];
            L = new double[N - 1];
            U = new double[N];

            Aa = new double[N - 1];
            Ab = new double[N];
            Ac = new double[N - 1];
            this.d = new double[N];

            //init Aa,Ab,Ac,Ad;
            for (int i = 0; i <= N - 2; i++)
            {
                Ab[i] = b[i];
                this.d[i] = d[i];

                Aa[i] = a[i];
                Ac[i] = c[i];
            }

            Ab[N - 1] = b[N - 1];
            this.d[N - 1] = d[N - 1];
            return true;
        }
        public bool Solve(out double[] R)
        {
            R = new double[Ab.Length];
/**********************A=LU************************************/
            U[0] = Ab[0];
            for (int i = 2; i <= N; i++)
            {
               // L[i] = Aa[i] / U[i - 1];
                L[i - 2] = Aa[i - 2] / U[i - 2];
                //U[i]=Ab[i]-Ac[i-1]*L[i];
                U[i-1] = Ab[i-1] - Ac[i - 2] * L[i-2];
            }
/************************END of A=LU **********************/
```

```
/*****************        Ly=d    ******************************/
            double[] Y = new double[d.Length];
            Y[0] = d[0];

            for (int i = 2; i <= N; i++)
            {
                //Y[k]=d[k]-L[k]*Y[k-1];
                Y[i - 1] = d[i - 1]-(L[i-2]) * (Y[i - 2]);

            }
 /*****************  End of Ly=d    ****************************/


/****************    Ux=Y    *********************************/
            //X[n]=Y[n]/U[n];
            R[N - 1] = Y[N - 1] / U[N - 1];
            //X[k]=(Y[k]-C[k]*X[k+1])/U[k];(n-1,,.....1)
            for (int i =N-1; i >= 1; i--)
            {
                R[i - 1] = (Y[i - 1] - Ac[i - 1] * R[i]) / U[i - 1];
            }
/****************    End of Ux=Y    ************************/
            for (int i = 0; i < R.Length; i++)
            {
                if (double.IsInfinity(R[i]) || double.IsNaN(R[i]))
                    return false;
            }
            return true;
        }
    }
}
```