MAP 5611 Intro to Computational Finance HW3

Student: Sen Zhang

I. Executive Summary

In this report, the algorithm of cubic spline works correctly through testing different conditions of function. The curves of yield rate and forward rate are shown. The yield rate, discount factor and forward rate for seven years are computed.

II. Statement of Problem

Bonds are discounted from their value at maturity by a discount factor,

$$D(t) = e^{-t \cdot y(t)}$$

where *y* is the yield. This discount rate can be written in terms of forward rate, *f(t).*

$$D(t) = e^{-t \cdot y(t)} = e^{-\int_0^t f(s)dt}$$

Given the following yield data:

Maturity	0.5	1.0	2.0	4.0	5.0	10.0	15.0	20.0
Rate	0.0552	0.0600	0.0682	0.0801	0.0843	0.0931	0.0912	0.0857

1. Implement the cubic spline algorithm and test the algorithm work correctly.
2. Use a natural cubic spline to plot the yield curve for twenty years.
3. Use the natural cubic spline to compute and plot the forward rate curve over twenty years.
4. What are the yield, discount factor and forward rate for seven years?

III. Description of The Mathematics

x:	x_0	x_1	x_2	...	x_n
y:	y_0	y_1	y_2	...	y_n

A cubic spline S is to interpolate the table above. The interpolation function S_i is different for each interval $[x_i, x_{i+1}]$. We need *S* is cubic, and *S, S', S''* are continuous at x_i's.

That *S* is cubic means *S''* is a linear function.

Define: $M_i = S(x_i)$

Then

$$S_i''(x) = \frac{M_i(x - x_{i+1})}{x_i - x_{i+1}} + \frac{M_{i+1}(x - x_i)}{x_{i+1} - x_i}$$

Set $\Delta x_i = x_{i+1} - x_i$

Then

$$S_i''(x) = \frac{M_i(x - x_{i+1})}{-\Delta x_i} + \frac{M_{i+1}(x - x_i)}{\Delta x_i}$$

Integrate twice to get the spline function:

$$S_i(x) = \frac{M_i(x - x_{i+1})^3}{-6\Delta x_i} + \frac{M_{i+1}(x - x_i)^3}{6\Delta x_i} + C(x - x_i) + D(x_{i+1} - x)$$

But $S_i(x_i) = y_i$

Then we have

$$C = \frac{1}{\Delta x_i}(y_{i+1} - \frac{M_{i+1}}{6}\Delta x_i^2)$$

$$D = \frac{1}{\Delta x_i}(y_i - \frac{M_i}{6}\Delta x_i^2)$$

So

$$S_i(x) = \frac{M_i(x - x_{i+1})^3}{-6\Delta x_i} + \frac{M_{i+1}(x - x_i)^3}{6\Delta x_i} + \frac{1}{\Delta x_i}(y_{i+1} - \frac{M_{i+1}}{6}\Delta x_i^2)(x - x_i) + \frac{1}{\Delta x_i}(y_i - \frac{M_i}{6}\Delta x_i^2)(x_{i+1} - x)$$

To gain M_i

$$S_i^{'}(x_i) = -\frac{\Delta x_i M_i}{3} - \frac{\Delta x_i M_{i+1}}{6} + y[x_i,\ x_{i+1}]$$
$$S_{i-1}^{'}(x_i) = \frac{\Delta x_i M_i}{3} + \frac{M_{i-1}\Delta x_i}{6} + y[x_{i-1},\ x_i]$$

Set $S_{i-1}^{'}(x_i) = S_i^{'}(x_i)$, then

$$\Delta x_{i-1} M_{i-1} + 2(\Delta x_i + \Delta x_{i-1}) M_i + \Delta x_i M_{i+1}$$
$$= 6\{y[x_i,\ x_{i+1}] - y[x_{i-1},\ x_i]\}$$

Let

$$A_i = 2(\Delta x_i + \Delta x_{i-1})$$
$$R_i = 6\{y[x_i,\ x_{i+1}] - y[x_{i-1},\ x_i]\}$$

Then

$$\Delta x_{i-1} M_{i-1} + A_i M_i + \Delta x_i M_{i+1} = R_i,\ for\ i = 1, 2, \ldots, n-1$$

Thus there are n-1 equations for n+1 unknown M_i.

For unknown M_0 & M_n

1. If $y''(x_0)$ and $y''(x_n)$ are known
   Set $M_0 = y''(x_0)$
   $M_n = y''(x_n)$
2. If we don't know y''
   Approximate M_0 & M_n using $y[x_0,\ x_1,\ x_2]$ and $y[x_{n-2},\ x_{n-1},\ x_n]$
3. Natural cubic spline: $M_0 = 0 = M_n$

For other M_i *for i= 1, …,n-1*

$$\begin{bmatrix} A_1 & \Delta x_1 & & & & \\ \Delta x_1 & . & . & & & \\ & . & . & . & & \\ & & . & . & \Delta x_{n-2} & \\ & & & \Delta x_{n-2} & A_{n-1} \end{bmatrix} \begin{bmatrix} M_1 \\ . \\ . \\ . \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} R_1 \\ . \\ . \\ . \\ R_{n-1} \end{bmatrix}$$

This linear system of equations is symmetric, tri-diagonal, diagonally dominant. It can be solved by Thomas Algorithm.

After solving this linear system, the M_i *for i=0, 1, …,n* are gained.

Thus the cubic spline is:

$$S_i(x) = \frac{M_i(x - x_{i+1})^3}{-6\Delta x_i} + \frac{M_{i+1}(x - x_i)^3}{6\Delta x_i} + \frac{1}{\Delta x_i}(y_{i+1} - \frac{M_{i+1}}{6}\Delta x_i^{\,2})(x - x_i) + \frac{1}{\Delta x_i}(y_i - \frac{M_i}{6}\Delta x_i^{\,2})(x_{i+1} - x)$$

$$\text{for } x \in [x_i,\ x_{i+1}],\ i = 0,\ 1,\ldots,\ n-1$$

IV. Description of The Algorithm

The algorithm for solving the linear system of equations:

Input n, x_i, y_i

(To store vectors)

for $i = 0$ *to* $n-1$

$\Delta x_i = x_{i+1} - x_i$

$$b_i = \frac{6(y_{i+1} - y_i)}{\Delta x_i}$$

end

(Forward elimination)

$A_1 = 2(\Delta x_0 + \Delta x_1)$

$R_1 = b_1 - b_0$

for $i = 2$ *to* $n-1$

$$A_i = 2(\Delta x_{i-1} + \Delta x_i) - \frac{\Delta x_{i-1}{}^2}{A_{i-1}}$$

$$R_i = b_i - b_{i-1} - \frac{\Delta x_{i-1} R_{i-1}}{A_{i-1}}$$

end

(Back substitution)

$M_n = 0$

$M_0 = 0$

for $i = n-1$ *to* 1

$$M_i = \frac{R_i - \Delta x_i M_{i+1}}{A_i}$$

end

Since S_i is different for each interval $[x_i, x_{i+1}]$, it is needed to estimate the interval that x is in.

Algorithm for searching the range of x:

(Binary search)

Input x

$L = 0$ $Q = n$

for $i = 0$ *to* n

if $Q - L = 1$, *exit*

else if $x \le x_j$

then $Q = j$

else

$L = j$

end if

end for

Then the approximation result is $S_i(x)$ at x.

Algorithm for testing

```
for (int i=0; i<=39; i++)
{
        x_in = 0.5 + i*0.5;
    spline sp;
    FM *pointS = sp.Spline(x, y, x_in, n);
    FM a1 = pointS[0];
        FM a2 = pointS[1];

    S[i] = a1;
        Sx[i] = a2;
        Y[i] = pow(x_in, 1);
        Yx[i] = 1;//*pow(x_in, 0);
}

FM maxv = abs(Y[0] - S[0]);
FM maxvx = abs(Yx[0] - Sx[0]);
for (int i=1; i<=39; i++)
{
        maxv = max(maxv, abs(Y[i] - S[i]));
        maxvx = max(maxvx, abs(Yx[i] - Sx[i]));

}

cout<<"max value of original function is "<<maxv<<endl;
cout<<"max value of 1st derivative is "<<maxvx<<endl;
```

Algorithm for plotting yield rate curve:

```
ofstream myFile;
myFile.open( "Yield.txt");// x_in is maturity and a1 is yield rate
```

```
for (int i=0; i<=100;i++)
{
        x_in = 0.5 + i*0.195;
    spline sp;
    FM *pointS = sp.Spline(x, y, x_in, n);
    FM a1 = pointS[0];
        myFile << x_in << "\t" << a1 << endl;
}
myFile.close();
```

Algorithm for plotting forward rate curve:

```
ofstream myFile;
myFile.open( "Forward rate.txt");// x_in is maturity, a1 is yield rate and a2 is the
first derivative of yield rate
for (int i=0; i<=100;i++)
{
        x_in = 0.5 + i*0.195;
    spline sp;
    FM *pointS = sp.Spline(x, y, x_in, n);
        FM a1 = pointS[0];
    FM a2 = pointS[1];
        myFile << x_in << "\t" << a2*x_in+a1 << endl;
}
myFile.close();
```

V. Results

1.

To test the cubic spline algorithm is working correctly.

Theorem1: Let $y \in C^4[a, b]$ and partition [a, b] into intervals Δx_i's. There exists $\Delta x = \max_i \Delta x_i \quad \beta = \frac{\max_i \Delta x_i}{\min_i \Delta x_i}$. Let S be cubic spline with

$$S''(x_0) = y''(x_0)$$
$$S''(x_n) = y''(x_n)$$

Then

$$\| y^{(\gamma)} - S^{(\gamma)} \|_\infty \le C_\gamma \Delta x^{4-\gamma} \| y^{(4)} \|_\infty \qquad (*)$$

$$C_0 = \frac{5}{384}, C_1 = \frac{1}{24}, C_2 = \frac{3}{8}, C_3 = \frac{1}{2}(\beta + \frac{1}{\beta})$$

To test the algorithm, two conditions $\gamma = 0 \ \& \ 1$ are necessary to check if formula (*) is satisfied since S and S' are used in task 3 and 4.

Considering the right hand side of formula (*), $y^{(4)}$ can be 0, constant or a function. So different functions are needed to test this algorithm.

Given the data of x:

x	0.5	1.0	2.0	4.0	5.0	10.0	15.0	20.0

1) Linear function. $y^{(4)}$ is 0 , for example $y = x$

   $M_0 = 0 = M_n$

   $\| y^{(0)} - S^{(0)} \|_\infty = 0$ and $\| y^{(1)} - S^{(1)} \|_\infty = 0$

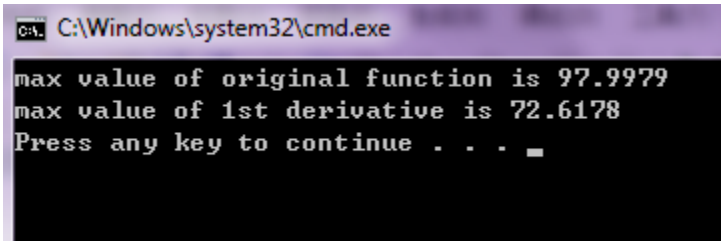
   So this algorithm is correct for approximation of linear function.

2) Function of degree 4. $y^{(4)}$ is constant, for example $y = x^4$

   $M_0 = 3, M_n = 4800, \ \Delta x = 5, y^{(4)} = 24$

	$\| y^{(\gamma)} - S^{(\gamma)} \|_\infty$	$C_\gamma \Delta x^{4-\gamma} \| y^{(4)} \|_\infty$
$\gamma = 0$	97.9979	195.3125
$\gamma = 1$	72.6178	125

   Since $\| y^{(\gamma)} - S^{(\gamma)} \|_\infty \le C_\gamma \Delta x^{4-\gamma} \| y^{(4)} \|_\infty$, the algorithm works correctly for function of degree 4.

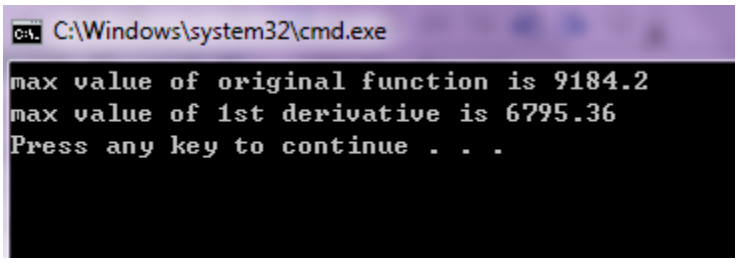


3) Function of degree 5. $y^{(4)}$ is a function, for example $y = x^5$

   $M_0 = 2.5, M_n = 160000, \ \Delta x = 5, \ \| y^{(5)} \|_\infty = 2400$

	$\| y^{(\gamma)} - S^{(\gamma)} \|_\infty$	$C_\gamma \Delta x^{4-\gamma} \| y^{(4)} \|_\infty$
$\gamma = 0$	9184.2	19531.25
$\gamma = 1$	6795.36	12500

Since $\| y^{(\gamma)} - S^{(\gamma)} \|_\infty \leq C_\gamma \Delta x^{4-\gamma} \| y^{(4)} \|_\infty$, the algorithm works correctly for function of degree 5.



In general, after testing the condition of $y^{(4)}$ is 0, constant and a function, we can conclude that this algorithm works correctly.

2.

Given the following yield data:

Table 1, Yield data

Maturity	0.5	1.0	2.0	4.0	5.0	10.0	15.0	20.0
Rate	0.0552	0.0600	0.0682	0.0801	0.0843	0.0931	0.0912	0.0857

A cubic spline is used to plot the yield curve for twenty years

Natural cubic spline: $M_0 = 0 = M_n$

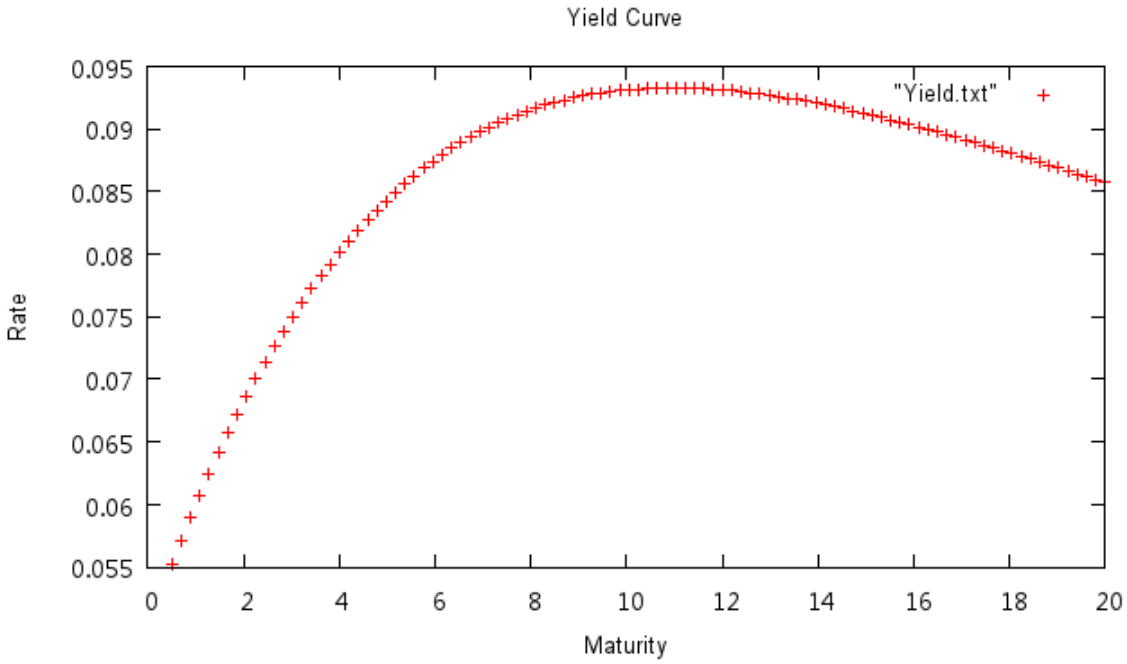
The M_i for the table is as following:

Table 2, M_i

i	M_i
0	0
1	-0.00229829
2	-0.00150512
3	-0.00108551
4	-0.00097673
5	-0.00036674
6	-0.00012432
7	0

Let maturity t=[0.5 : 0.195 : 20.0], which means there are 101 points (t, y), where t is maturity and y is yield rate. Then plot these points using ***gnuplot.***

Graph 1, Yield Rate Curve for twenty years



3.

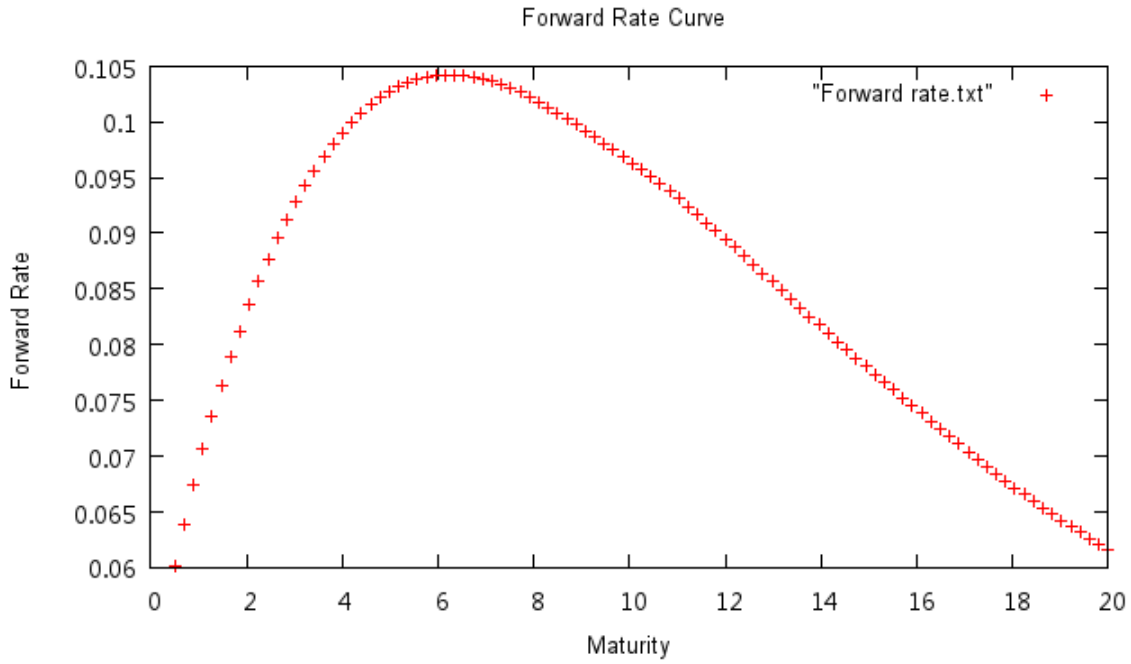
Natural cubic spline: $M_0 = 0 = M_n$

For forward rate based on table 1

$$D(t) = e^{-t \cdot y(t)} = e^{-\int_0^t f(s)dt}$$
$$\Rightarrow f(t) = y + ty'$$

Let maturity t=[0.5 : 0.195 : 20.0], which means there are 101 points (t, f(t)), where t is maturity and f(t) is forward rate. Then plot these points using ***gnuplot.***

Graph 2, Forward Rate Curve for twenty years



4.

The discount factor is $D(t) = e^{-t \cdot y(t)}$ and the forward rate is $f(t) = y + ty'$.

So the yield rate, discount factor and forward rate for seven years are as following:

Table 3, Yield Rate, Discount Factor and Forward Rate for seven years

Maturity(years)	Yield Rate	Discount Factor	Forward Rate
0.5	0.0552	0.972777	0.060096
1	0.06	0.941765	0.069217
1.5	0.064338	0.908004	0.076588
2	0.0682	0.872494	0.08283
2.5	0.071674	0.835952	0.088146
3	0.074798	0.799001	0.092543
3.5	0.077598	0.762165	0.096125
4	0.0801	0.725859	0.098999
4.5	0.082329	0.690403	0.101209
5	0.0843	0.656062	0.102768
5.5	0.086027	0.623037	0.103739
6	0.087525	0.591465	0.104192
6.5	0.08881	0.561432	0.104187
7	0.089896	0.532979	0.103785

VI. Conclusions

It can be found that the algorithm of cubic spline works correctly through testing different conditions of functions. And the curves show that the cubic spline is smooth. The yield rate, discount factor and forward rate at the seventh year are

Maturity(years)	Yield Rate	Discount Factor	Forward Rate
7	0.089896	0.532979	0.103785

VII. Program Listing

spline.h

```
#define FM double
#pragma once

class spline
{public:
	FM  *Spline(FM x[], FM y[], FM x_in, int n);
};
```

spline.cpp

```
#include <iostream>
#include <fstream>
#include "math.h"
#include "stddef.h"
#include "spline.h"
#include <iomanip>
using namespace std;

FM* spline::Spline(FM x[], FM y[], FM x_in, int n)
{
//	To store vectors

	const size_t array_size = 10; // the arry size may be wrong!!!!!!!!!!

	FM dx[array_size-1], b[array_size-1], A[array_size-1], R[array_size-1], M[array_size];

	for (int i=0; i <= n-1; i++)
	{
		dx[i] = x[i+1] - x[i];
		b[i] = 6 * (y[i+1] - y[i]) / dx[i];
	}

//  Forward elimination

	A[1] = 2 * (dx[0] + dx[1]);
```

```
        R[1] = b[1] - b[0];

        for (int i=2; i<= n-1; i++)
        {
                A[i] = 2 * (dx[i] + dx[i-1]) - dx[i-1]*dx[i-1]/A[i-1];
                R[i] = b[i] - b[i-1] - dx[i-1] * R[i-1] / A[i-1];
        }

//  Back substitution

        M[n] = 0;     //R[n] / A[n];
        //M[n-1] = R[n-1] / A[n-1];
    M[0] = 0;

        for (int i=n-1; i>= 1; i--)
        {
                M[i] = (R[i] - dx[i] * M[i+1])/A[i];

        }


//  To find the range of x_in

        int L = 0;
        int Q = n;

        for (int k=0; k<=n; k++)
        {
                int j = int ((L + Q)/2);

                if (Q - L ==1)
                {
                        break;
                }
                else
                {
                        if (x_in <= x[j])
                        {
                                Q = j;
                        }
                    else
                    {
                            L = j;
                    }
                }

        }

        FM S[2];

        S[0] = M[L]/(6*dx[L])*pow((x[L+1]-x_in), 3)
                + M[L+1]/(6*dx[L])*pow((x_in-x[L]), 3)
                + 1/dx[L]*(y[L+1]-M[L+1]/6*dx[L]*dx[L])*(x_in-x[L])
                + 1/dx[L]*(y[L]-M[L]/6*dx[L]*dx[L])*(x[L+1]-x_in);
```

```
        S[1] = -M[L]/(2*dx[L])*pow((x[L+1]-x_in), 2)
                + M[L+1]/(2*dx[L])*pow((x_in-x[L]), 2)
                + 1/dx[L]*(y[L+1]-M[L+1]/6*dx[L]*dx[L])
                - 1/dx[L]*(y[L]-M[L]/6*dx[L]*dx[L]);

        FM *pointS;
        pointS = S;

        return pointS;
}

int main()
{
        FM S[40],Sx[40], x_in, Y[40], Yx[40];
    int n = 7;

        FM x[] = { 0.5, 1.0, 2.0, 4.0, 5.0, 10.0, 15.0, 20.0 };
        FM y[8];
        for (int i=0; i<=n; i++)
        {
                y[i] = pow(x[i], 1);

        }


        for (int i=0; i<=39; i++)
        {
                x_in = 0.5 + i*0.5;
            spline sp;
            FM *pointS = sp.Spline(x, y, x_in, n);
            FM a1 = pointS[0];
                FM a2 = pointS[1];

            S[i] = a1;
                Sx[i] = a2;
                Y[i] = pow(x_in, 1);
                Yx[i] = 1;//*pow(x_in, 0);
        }

        FM maxv = abs(Y[0] - S[0]);
    FM maxvx = abs(Yx[0] - Sx[0]);
        for (int i=1; i<=39; i++)
        {
                maxv = max(maxv, abs(Y[i] - S[i]));
                maxvx = max(maxvx, abs(Yx[i] - Sx[i]));

        }


        cout<<"max value of original function is "<<maxv<<endl;
        cout<<"max value of 1st derivative is "<<maxvx<<endl;

        /*FM x[] = { 0.5, 1.0, 2.0, 4.0, 5.0, 10.0, 15.0, 20.0 };
        FM y[] = { 0.0552, 0.060, 0.0682, 0.0801, 0.0843, 0.0931, 0.0912, 0.0857 };
```

```
spline sp;
FM *pointS = sp.Spline(x, y, x_in, n);

FM a1 = pointS[0];
FM a2 = pointS[1];

cout << "The yield at "<<x_in<<" is: " << a1<<endl;




FM D = exp(-a1*x_in);
cout << "The discount factor at "<<x_in<<" is: " << D<<endl;

FM forward = a2*x_in+a1;
cout << "The forward rate at "<<x_in<<" is: " << forward<<endl;

ofstream myFile;
myFile.open( "prob.txt");
for (int i=0; i<=13;i++)
{
        x_in = 0.5 + i*0.5;
    spline sp;
    FM *pointS = sp.Spline(x, y, x_in, n);
    FM a1 = pointS[0];
        FM a2 = pointS[1];
    FM D = exp(-a1*x_in);
    FM forward = a2*x_in+a1;

        myFile  << x_in<< setprecision(7)<<"\t"<< a1<< "\t"<<D<< "\t"<<forward<<endl;

}
myFile.close();

ofstream myFile;
myFile.open( "Yield.txt");
for (int i=0; i<=100;i++)
{
        x_in = 0.5 + i*0.195;
    spline sp;
    FM *pointS = sp.Spline(x, y, x_in, n);
    FM a1 = pointS[0];
        myFile << x_in << "\t" << a1 << endl; // x_in is maturity and a1 is yield rate
}
myFile.close();

ofstream myFile;
myFile.open( "Forward rate.txt");
for (int i=0; i<=100;i++)
{
        x_in = 0.5 + i*0.195;
    spline sp;
    FM *pointS = sp.Spline(x, y, x_in, n);
        FM a1 = pointS[0];
    FM a2 = pointS[1];
```

```
                myFile << x_in << "\t" << a2*x_in+a1 << endl; // x_in is maturity, a1 is yield
rate and a2 is the first derivative of yield rate
        }
        myFile.close();*/
        return 0;
}
```