Natural and Clamped Cubic Spline Interpolation

Purpose

Cubic splines are used for function interpolation and approximation.
For a function f(x) defined on the interval [a,b], either in functional or tabular form, cubic spline interpolation is the process of constructing (generally different) piecewise continuous cubic polynomials on subintervals [ti,ti+1] of the function domain [a,b]. Cubic splines are preferred to polynomial interpolants because cubic splines are locally only cubics, and are hence simple to evaluate. They exhibit less severe oscillatory behavior than interpolating polynomials. In fact, the natural cubic spline is the smoothest possible function of all square integrable functions. Also, interpolating polynomials, since they are defined globally rather than locally as are splines, are adversely affected by bad data: error in one data point will affect the entire interpolating polynomial. Conversely, with a spline it is possible to confine the ill-effects of an erroneous data point.
Cubic splines have the following properties: (i) they interpolate the given data;
(ii) they have continuity of the zeroth, first and second derivatives at interior
points; (iii) they satisfy certain boundary conditions. *The nat*ural *or* free boundary
condition is the most common. Alternatively one may use *the cl*amp*ed or* fixed
boundary condition.

The natural cubic spline is constructed by imposing the following conditions:
(a) If f(x) is given in functional form, the given domain [a,b] is subdivided into subintervals [ti,ti+1] , i = 0, ... , n. and the function evaluated at each ti.
The subintervals need not be of equal size, but must be ordered such that
ti < ti+1. For f(x) is given in tabular form the subintervals are defined by the given data.
(b) A cubic is set up over each subinterval and defined by the relation
S[i,x] = ai + bi (x-t[i]) + ci (x-t[i])^2 + di (x-t[i])^3, xÎ [ti,ti+1] , i = 0, ... , n-1
(c) The **requirement of interpolation of** the given data imposes the condition that
S[i,x] = f[t[i]] = ai for each point i = 0, ... , n.
(d) The **requirement of continui**ty of the spline imposes the condition that
S[i,t[i]] = S[i,t[i+1]] for each interior point i = 1, ... , n-1.
(e) **The requirement of continuity of the first deriva**tive of the spline imposes the condition that S'[i,t[i]] = S'[i,t[i+1]] for each interior point i = 1, ... , n-1.
(f) **The requirement of continuity of the second deriv**ative of the spline imposes the condition that S''[i,t[i]] = S''[i,t[i+1]] for each interior point i = 1, ... , n-1.
(g) For the natural cubic spline, boundary conditions are S''[0,t[0]] = S''[n-1,t[n]] = 0.
For the clamped cubic spline, the boundary conditions are S'[0,t[0]] = f'[t[0]];
S'[n-1,t[n]] = f'[n].

Since the ai's are known automatically by condition (a), there will result a linear
system of 3n equations for the 3n unknowns bi, ci and di, i = 0, ... , n-1.
These equations are usually solved by matrix methods such as inversion or
LU-Factorization. However, they are ha***ndled very*** conveniently **by M**athematica's built-in Solve function.

Usage

This program requires input of n+1 data points {t[i],f[t[i]]}, i = 0, ... , n.
Both natural and clamped boundary cubic splines will be constructed and
plotted against the given data for comparative purposes.
The integral of the spline is also computed and printed.
Your data need not be evenly spaced, but must be ordered
(a = t[0] < t[1] < ... < t[n] = b)
Note that n will be one less than the actual number of data p**oints.**

Execution Time on the Macintosh II

The **Solve** function took approximately 68 seconds to solve a 9 x 9 system of equations (ten input data points) on a Macintosh II.
If a larger system is involved, matrix inversion should be used.
The steps involved are similar to those carried out by th**e Sol**ve method. In the matrix inversion method, the system [LHS]{SplineCoefficients} = {RHS} must be solved for the unknown spline coefficients b[i], c[i] and d[i]. Generation of the
array elements of both [LHS] and {RHS} is facilitated via t**he Coeffici**ent
function ***of Mathema***tica.

Spline Procedures

```
Clear[a, b, c, d, f, t, S, x, CubicSpline, plot]
S[i_Integer, x_] := a[i] + b[i] (x - t[i]) + c[i] (x - t[i])^2 + d[i] (x - t[i])^3
ConstructCubicSpline :=
 Block[{i}, EquationList = {}; EquationList = Append[EquationList, S[n - 1, t[n]] == a[n]];
  Do[EquationList = Append[EquationList, S[i, t[i + 1]] == S[i + 1, t[i + 1]]], {i, 0, n - 2}];
  Do[EquationList = Append[EquationList,
     Expand[∂x S[i, x] /. x → t[i + 1]] == Expand[∂x S[i + 1, x] /. x → t[i + 1]]], {i, 0, n - 2}];
  Do[EquationList = Append[EquationList, Expand[∂{x,2} S[i, x] /. x → t[i + 1]] ==
      Expand[∂{x,2} S[i + 1, x] /. x → t[i + 1]]], {i, 0, n - 2}]]
ShowUnknowns := Block[{i}, Unknowns = {}; Do[Unknowns = Append[Unknowns, b[i]],
   {i, 0, n - 1}]; Do[Unknowns = Append[Unknowns, c[i]], {i, 0, n - 1}];
  Do[Unknowns = Append[Unknowns, d[i]], {i, 0, n - 1}]; Unknowns]
SolveandPrintSpline := Block[{i},
  Print["\nThe solution of the system of spline equations "];
  Print[EquationList]; Print["\nwith unknowns "]; Print[ShowUnknowns];
  Result = N[Solve[EquationList, Unknowns]]; Do[b[i - 1] = Result[[1, i, 2]];
   c[i - 1] = Result[[1, i + n, 2]]; d[i - 1] = Result[[1, i + 2 n, 2]], {i, 1, 3 n/3}];
  Print["\nThe cubic spline is given as:"]; Do[Print["\nS[", i, ",x] = ",
    S[i, x], "  for ", t[i], " < x < ", t[i + 1]], {i, 0, n - 1}]]
NaturalCubicSpline := Block[{i}, ConstructCubicSpline;
  EquationList = Append[EquationList, Expand[∂{x,2} S[0, x] /. x → t[0]] == 0];
  EquationList = Append[EquationList, Expand[∂{x,2} S[n - 1, x] /. x → t[n]] == 0];
  title = "FreeCubicSpline"; SolveandPrintSpline; PiecewiseCubicPlot]
ClampedCubicSpline := Block[{i}, ConstructCubicSpline;
  EquationList = Append[EquationList, Expand[∂x S[0, x] /. x → t[0]] == f'[t[0]]];
  EquationList = Append[EquationList, Expand[∂x S[n - 1, x] /. x → t[n]] == f'[t[n]]];
  title = "FixedCubicSpline"; SolveandPrintSpline; PiecewiseCubicPlot]
PiecewiseCubicPlot := Block[{i},
  Print["The cubic spline approximation of the function is plotted below."];
  Do[plot[i] = Table[N[{x, S[i, x]}], {x, t[i], t[i + 1], 1/10 (t[i + 1] - t[i])}],
   {i, 0, n - 1}]; SplinePlot = ListPlot[Flatten[Table[plot[i], {i, 0, n - 1}], 1],
    PlotJoined → True, PlotLabel → title];
  TrueDataPlot = ListPlot[InputData, PlotJoined → True, PlotLabel → "Actual Data"];
  Show[SplinePlot, TrueDataPlot];]
```

Construction of Natural Cubic Spline by the Solve Function

■ Data Input

```
NumberofPoints = 7
n = NumberofPoints - 1
Do[t[i] = 0.5 i, {i, 0, 6}]
f[t[0]] = 70; f[t[1]] = 70; f[t[2]] = 66
f[t[3]] = 52; f[t[4]] = 18; f[t[5]] = 11
f[t[6]] = 10
InputData = Table[{t[i], f[t[i]]}, {i, 0, n}]
LeftLimit = t[0]; RightLimit = t[n]
spacing = (RightLimit - LeftLimit)/n
Print["The input data are: "]
Do[a[i] = f[t[i]]; Print["x[", i, "] = ", N[t[i]], "\ty[", i, "] = ", N[a[i]]], {i, 0, n}]
```

```
7
```

```
6
```

```
66
```

```
11
```

```
10
```

```
{{0, 70}, {0.5, 70}, {1., 66}, {1.5, 52}, {2., 18}, {2.5, 11}, {3., 10}}
```

```
3.
```

```
0.5
```

```
The input data are:
x[0] = 0.    y[0] = 70.
x[1] = 0.5    y[1] = 70.
x[2] = 1.    y[2] = 66.
x[3] = 1.5    y[3] = 52.
x[4] = 2.    y[4] = 18.
x[5] = 2.5    y[5] = 11.
x[6] = 3.    y[6] = 10.
```

```
NaturalCubicSpline
```

The solution of the system of spline equations

$$\{11 + 0.5\,b[5] + 0.25\,c[5] + 0.125\,d[5] == 10,$$
$$70 + 0.5\,b[0] + 0.25\,c[0] + 0.125\,d[0] == 70 + 0.\,b[1] + 0.\,c[1] + 0.\,d[1],$$
$$70 + 0.5\,b[1] + 0.25\,c[1] + 0.125\,d[1] == 66 + 0.\,b[2] + 0.\,c[2] + 0.\,d[2],$$
$$66 + 0.5\,b[2] + 0.25\,c[2] + 0.125\,d[2] == 52 + 0.\,b[3] + 0.\,c[3] + 0.\,d[3],$$
$$52 + 0.5\,b[3] + 0.25\,c[3] + 0.125\,d[3] == 18 + 0.\,b[4] + 0.\,c[4] + 0.\,d[4],$$
$$18 + 0.5\,b[4] + 0.25\,c[4] + 0.125\,d[4] == 11 + 0.\,b[5] + 0.\,c[5] + 0.\,d[5],$$
$$b[0] + 1.\,c[0] + 0.75\,d[0] == b[1] + 0.\,c[1] + 0.\,d[1],\ b[1] + 1.\,c[1] + 0.75\,d[1] == b[2] + 0.\,c[2] + 0.\,d[2],$$
$$b[2] + 1.\,c[2] + 0.75\,d[2] == b[3] + 0.\,c[3] + 0.\,d[3],\ b[3] + 1.\,c[3] + 0.75\,d[3] == b[4] + 0.\,c[4] + 0.\,d[4],$$
$$b[4] + 1.\,c[4] + 0.75\,d[4] == b[5] + 0.\,c[5] + 0.\,d[5],\ 2\,c[0] + 3.\,d[0] == 2\,c[1] + 0.\,d[1],$$
$$2\,c[1] + 3.\,d[1] == 2\,c[2] + 0.\,d[2],\ 2\,c[2] + 3.\,d[2] == 2\,c[3] + 0.\,d[3],$$
$$2\,c[3] + 3.\,d[3] == 2\,c[4] + 0.\,d[4],\ 2\,c[4] + 3.\,d[4] == 2\,c[5] + 0.\,d[5],\ 2\,c[0] == 0,\ 2\,c[5] + 3.\,d[5] == 0\}$$

with unknowns

$$\{b[0], b[1], b[2], b[3], b[4], b[5], c[0], c[1],$$
$$c[2], c[3], c[4], c[5], d[0], d[1], d[2], d[3], d[4], d[5]\}$$

The cubic spline is given as:

$$S[0,x] = 70 + 1.73846\,x + 1.77636\times10^{-14}\,x^2 - 6.95385\,x^3 \quad \text{for } 0 < x < 0.5$$

$$S[1,x] = 70 - 3.47692\,(-0.5 + x) - 10.4308\,(-0.5 + x)^2 + 2.76923\,(-0.5 + x)^3 \quad \text{for } 0.5 < x < 1.$$

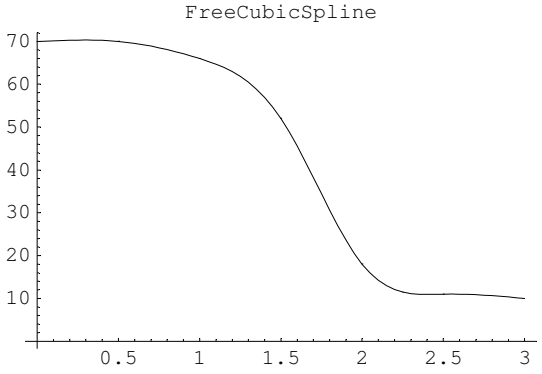
$$S[2,x] = 66 - 11.8308\,(-1. + x) - 6.27692\,(-1. + x)^2 - 52.1231\,(-1. + x)^3 \quad \text{for } 1. < x < 1.5$$

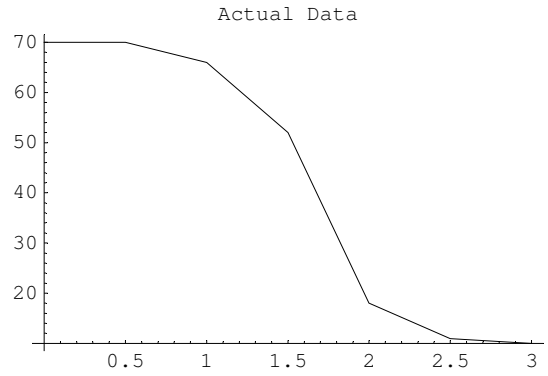
$$S[3,x] = 52 - 57.2\,(-1.5 + x) - 84.4615\,(-1.5 + x)^2 + 125.723\,(-1.5 + x)^3 \quad \text{for } 1.5 < x < 2.$$

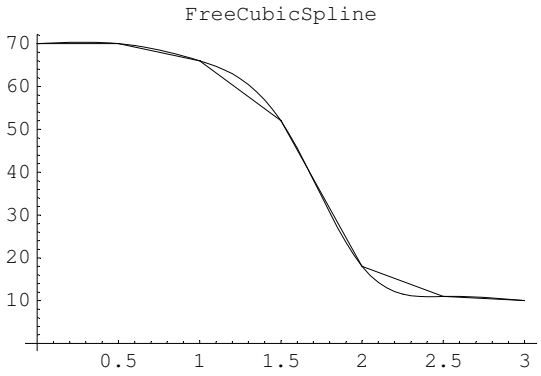
$$S[4,x] = 18 - 47.3692\,(-2. + x) + 104.123\,(-2. + x)^2 - 74.7692\,(-2. + x)^3 \quad \text{for } 2. < x < 2.5$$

$$S[5,x] = 11 + 0.676923\,(-2.5 + x) - 8.03077\,(-2.5 + x)^2 + 5.35385\,(-2.5 + x)^3 \quad \text{for } 2.5 < x < 3.$$

The cubic spline approximation of the function is plotted below.




Actual Data
70
60
50
40
30
20
0.5
1
1.5
2
2.5
3




```
FreeSplinePlot = SplinePlot;
```

Examples of Integration and Differentiation

Cubic Spline Integration

```
SplineIntegral = Sum[Integrate[S[i, t], Release[{t, t[i], t[i + 1]}]], {i, 0, n-1}];
Print["The integral of the cubic spline is ", SplineIntegral, "."]
```

```
The integral of the cubic spline is 128.606.
```

Differentiation: Location of Inflection Point

```
Solve[∂{x,2} S[3, x] == 0, x]
```

```
{{x → 1.72394}}
```

Construction of Clamped Cubic Spline by the Solve Function

The clamped, or fixed boundary, cubic spline uses the boundary conditions S'[0,x[0]] = f'[x[0]]; S'[n-1,x[n]] = f'[x[n]].
*The clamped cubic spline should be used whenever these boundary conditions are known, as it will better represent the data compared to the natural cubic spline. Construction of the clamped boundary spline is identical to the natural cubic spline except for the boundary condit*ions.

■ Data Input

For comparison, let us use the same data as before but estimate f'[x[0]] and f'[x[n]] by finite difference formulas. We use a three point forward difference approximation for f'[x[0]] and a three point backward difference approximation for f'[x[n]] = f'[x[6]] to approximate the derivative boundary conditions.

```
Clear[a, b, c, d, f, t]
NumberofPoints = 7
n = NumberofPoints - 1
Do[t[i] = 0.5 i, {i, 0, 6}]
f[t[0]] = 70; f[t[1]] = 70; f[t[2]] = 66
f[t[3]] = 52; f[t[4]] = 18; f[t[5]] = 11
f[t[6]] = 10
h = 0.5;
f'[t[0]] = (-3 f[t[0]] + 4 f[t[1]] - f[t[2]])/(2 h)
f'[t[n]] = (f[t[n - 2]] - 4 f[t[n - 1]] + 3 f[t[n]])/(2 h)
InputData = Table[{t[i], f[t[i]]}, {i, 0, n}]
LeftLimit = t[0]; RightLimit = t[n]
Print["The input data are: "]
Do[a[i] = f[t[i]];
  Print["x[", i, "] = ", N[t[i]], "\ty[", i, "] = ", N[a[i]]], {i, 0, n}];
Print["with boundary conditions f[", t[0], "] = ",
 f'[t[0]], ", f[", t[n], "] = ", f'[t[n]]]
```

```
7

6

66

11

10

4.

4.

{{0, 70}, {0.5, 70}, {1., 66}, {1.5, 52}, {2., 18}, {2.5, 11}, {3., 10}}

3.

The input data are:

x[0] = 0.    y[0] = 70.

x[1] = 0.5    y[1] = 70.

x[2] = 1.    y[2] = 66.
```

x[3] = 1.5 y[3] = 52.

x[4] = 2. y[4] = 18.

x[5] = 2.5 y[5] = 11.

x[6] = 3. y[6] = 10.

with boundary conditions f[0] = 4., f[3.] = 4.

ClampedCubicSpline

The solution of the system of spline equations

$$\{11 + 0.5\,b[5] + 0.25\,c[5] + 0.125\,d[5] == 10,\ 70 + 0.5\,b[0] + 0.25\,c[0] + 0.125\,d[0] == 70 + 0.\,b[1] + 0.\,c[1] + 0.\,d[1],\ 70 + 0.5\,b[1] + 0.25\,c[1] + 0.125\,d[1] == 66 + 0.\,b[2] + 0.\,c[2] + 0.\,d[2],\ 66 + 0.5\,b[2] + 0.25\,c[2] + 0.125\,d[2] == 52 + 0.\,b[3] + 0.\,c[3] + 0.\,d[3],\ 52 + 0.5\,b[3] + 0.25\,c[3] + 0.125\,d[3] == 18 + 0.\,b[4] + 0.\,c[4] + 0.\,d[4],\ 18 + 0.5\,b[4] + 0.25\,c[4] + 0.125\,d[4] == 11 + 0.\,b[5] + 0.\,c[5] + 0.\,d[5],\ b[0] + 1.\,c[0] + 0.75\,d[0] == b[1] + 0.\,c[1] + 0.\,d[1],\ b[1] + 1.\,c[1] + 0.75\,d[1] == b[2] + 0.\,c[2] + 0.\,d[2],\ b[2] + 1.\,c[2] + 0.75\,d[2] == b[3] + 0.\,c[3] + 0.\,d[3],\ b[3] + 1.\,c[3] + 0.75\,d[3] == b[4] + 0.\,c[4] + 0.\,d[4],\ b[4] + 1.\,c[4] + 0.75\,d[4] == b[5] + 0.\,c[5] + 0.\,d[5],\ 2\,c[0] + 3.\,d[0] == 2\,c[1] + 0.\,d[1],\ 2\,c[1] + 3.\,d[1] == 2\,c[2] + 0.\,d[2],\ 2\,c[2] + 3.\,d[2] == 2\,c[3] + 0.\,d[3],\ 2\,c[3] + 3.\,d[3] == 2\,c[4] + 0.\,d[4],\ 2\,c[4] + 3.\,d[4] == 2\,c[5] + 0.\,d[5],\ b[0] == 4.,\ b[5] + 1.\,c[5] + 0.75\,d[5] == 4.\}$$

with unknowns

$$\{b[0], b[1], b[2], b[3], b[4], b[5], c[0], c[1], c[2], c[3], c[4], c[5], d[0], d[1], d[2], d[3], d[4], d[5]\}$$

The cubic spline is given as:

$$S[0,x] = 70 + 4.\,x - 7.81538\,x^2 - 0.369231\,x^3 \quad \text{for } 0 < x < 0.5$$

$$S[1,x] = 70 - 4.09231\,(-0.5 + x) - 8.36923\,(-0.5 + x)^2 + 1.10769\,(-0.5 + x)^3 \quad \text{for } 0.5 < x < 1.$$

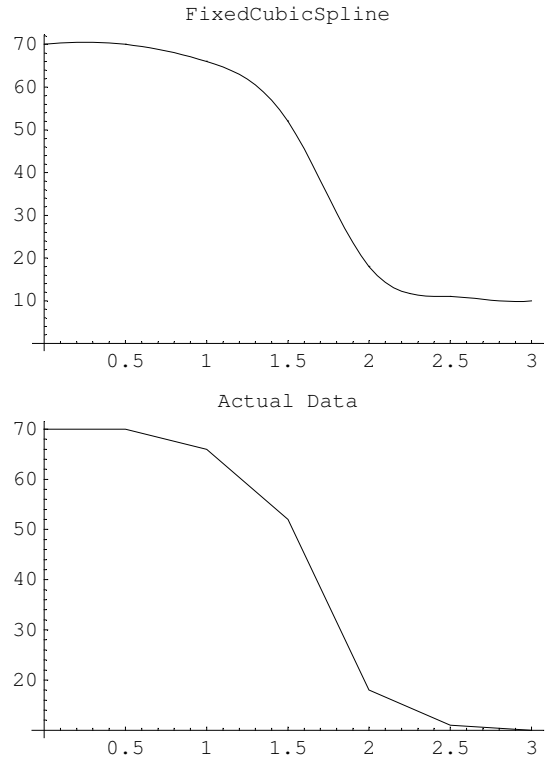
$$S[2,x] = 66 - 11.6308\,(-1. + x) - 6.70769\,(-1. + x)^2 - 52.0615\,(-1. + x)^3 \quad \text{for } 1. < x < 1.5$$

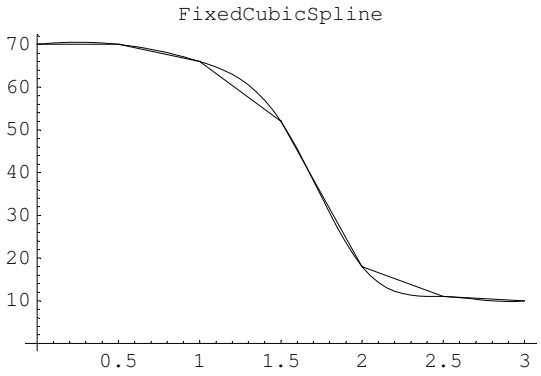
$$S[3,x] = 52 - 57.3846\,(-1.5 + x) - 84.8\,(-1.5 + x)^2 + 127.138\,(-1.5 + x)^3 \quad \text{for } 1.5 < x < 2.$$

$$S[4,x] = 18 - 46.8308\,(-2. + x) + 105.908\,(-2. + x)^2 - 80.4923\,(-2. + x)^3 \quad \text{for } 2. < x < 2.5$$

$$S[5,x] = 11 - 1.29231\,(-2.5 + x) - 14.8308\,(-2.5 + x)^2 + 26.8308\,(-2.5 + x)^3 \quad \text{for } 2.5 < x < 3.$$

The cubic spline approximation of the function is plotted below.

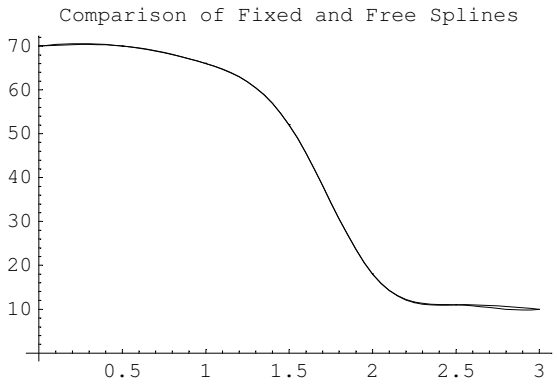

FixedCubicSpline
70
60
50
40
30
20
10
0.5
1
1.5
2
2.5
3
Actual Data
70
60
50
40
30
20
0.5
1
1.5
2
2.5
3




```
FixedSplinePlot = SplinePlot;
```

Comparison of Free and Fixed Cubic Spline Plots

```
Show[FreeSplinePlot, FixedSplinePlot,
  PlotLabel → "Comparison of Fixed and Free Splines"];
```



For this example, the only difference occurs near the end points. For the above, FreeSplinePlot and FixedSplinePlot were created using the procedur**e** **PiecewiseCubicPlo**t and a scrap sheet.

Solution of Spline Equations by Matrix Inversion

If n >= 10 or so, the time requirements of the **Solve** function becomes prohibitive on computers such as the MacPlus or Mac SE. For this reason, the matrix inversion capabilities of *Mathematic*a are used to solve the system [EquationList]{unknowns} = {RHS} defined by the cubic spline properties. First, the system of linear equations requiring solution are generated in the identical manner as above and stored in EquationList. Then *the Mathema*tica function CoefficientList is used to store the coefficients in a matrix array. The vector of unknowns is generated as above. The solution is obtained as Inverse[Eqns].Unknowns and the solution proceeds as previously. For illustrative purposes we consider the same data used in the Natural Cubic Spline example preceding.

```
Clear[myrow, Equation]
Do[Equation[i] = (S[i, t[i + 1]] - a[i]) - (S[i + 1, t[i + 1]] - a[i + 1]);
 constant[i] = -a[i] + a[i + 1], {i, 0, n - 2}]
Do[Equation[i + n - 1] = Expand[∂x S[i, x] /. x → t[i + 1]] - Expand[∂x S[i + 1, x] /. x → t[i + 1]];
 constant[i + n - 1] = 0, {i, 0, n - 2}]
Do[Equation[i + 2 n - 2] = Expand[∂{x,2} S[i, x] /. x → t[i + 1]] -
    Expand[∂{x,2} S[i + 1, x] /. x → t[i + 1]]; constant[i + 2 n - 2] = 0, {i, 0, n - 2}]
Equation[3 n - 3] = S[n - 1, t[n]] - a[n - 1]
constant[3 n - 3] = a[n] - a[n - 1]
Equation[3 n - 2] = Expand[∂{x,2} S[0, x] /. x → t[0]]
constant[3 n - 2] = 0
Equation[3 n - 1] = Expand[∂{x,2} S[n - 1, x] /. x → t[n]]
constant[3 n - 1] = 0;
```

```
-1.
```

```
-1
```

```
-15.6308
```

```
0
```

```
50.8308
```

```
Do[myrow[j] = {};
 myrow[j] = Append[myrow[j], Table[Coefficient[Equation[j], b[i]], {i, 0, n - 1}]];
 myrow[j] = Append[myrow[j], Table[Coefficient[Equation[j], c[i]], {i, 0, n - 1}]];
 myrow[j] = Append[myrow[j], Table[Coefficient[Equation[j], d[i]], {i, 0, n - 1}]];
 myrow[j] = Flatten[myrow[j]], {j, 0, 3 n - 1}]
```

```
LHS = Table[myrow[i], {i, 0, 3 n - 1}]
```

```
{{0.5, 0, 0, 0, 0, 0, 0.25, 0, 0, 0, 0, 0, 0.125, 0, 0, 0, 0, 0},
 {0, 0.5, 0, 0, 0, 0, 0, 0.25, 0, 0, 0, 0, 0, 0.125, 0, 0, 0, 0},
 {0, 0, 0.5, 0, 0, 0, 0, 0, 0.25, 0, 0, 0, 0, 0, 0.125, 0, 0, 0},
 {0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0.25, 0, 0, 0, 0, 0, 0.125, 0, 0},
 {0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0.25, 0, 0, 0, 0, 0, 0.125, 0},
 {1, -1, 0, 0, 0, 0, 1., 0, 0, 0, 0, 0, 0.75, 0, 0, 0, 0, 0},
 {0, 1, -1, 0, 0, 0, 0, 1., 0, 0, 0, 0, 0, 0.75, 0, 0, 0, 0},
 {0, 0, 1, -1, 0, 0, 0, 0, 1., 0, 0, 0, 0, 0, 0.75, 0, 0, 0},
 {0, 0, 0, 1, -1, 0, 0, 0, 0, 1., 0, 0, 0, 0, 0, 0.75, 0, 0},
 {0, 0, 0, 0, 1, -1, 0, 0, 0, 0, 1., 0, 0, 0, 0, 0, 0.75, 0},
 {0, 0, 0, 0, 0, 0, 2, -2, 0, 0, 0, 0, 3., 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 2, -2, 0, 0, 0, 0, 3., 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 2, -2, 0, 0, 0, 0, 3., 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 2, -2, 0, 0, 0, 0, 3., 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, -2, 0, 0, 0, 0, 3., 0},
 {0, 0, 0, 0, 0, 0.5, 0, 0, 0, 0, 0, 0.25, 0, 0, 0, 0, 0, 0.125},
 {0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
 {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 3.}}
```

```
RHS = Table[constant[i], {i, 0, 3 n - 1}]
```

```
{0, -4, -14, -34, -7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 0, 0}
```

```
Inverse[LHS].RHS
```

$$\left\{1.738461538461538421,\ -3.476923076923076941,\ -11.83076923076923076,\right.$$
$$-57.19999999999999998,\ -47.36923076923076923,\ 0.6769230769230769078,$$
$$\frac{1.812121111532168788}{10^{19}},\ -10.4307692307692307,\ -6.276923076923076917,$$
$$-84.46153846153846156,\ 104.123076923076923,\ -8.030769230769230722,$$
$$\left.-6.953846153846153808,\ 2.769230769230769193,\ -52.12307692307692311,\right.$$
$$\left.125.7230769230769231,\ -74.76923076923076919,\ 5.353846153846153825\right\}$$

To continue, use the following step to assign the above values to the matrix coefficients and print out the splines as previously shown.

```
Do[(* assign coefficients *)
     b[i-1] = %[[i]];
     c[i-1] = %[[i+n]];
     d[i-1] = %[[i+2*n]],{i,1,n}
    ];
```

These results are identical to those given by the **Solve** function.

Whereas the **Solv**e function took 3.3667 seconds to set up and 23.1167 seconds to solve the 6 x 6 system, the matrix inversion method function took 6.85 seconds

to set up and 15.9667 seconds to solve the same system on a Macintosh II computer.

In this case, the matrix method took approximately 14% less time overall than t**he So**lve function (set up and solution).

For a 9 x 9 system using the function f(x) = x^2 on [0,1**],** **S**olve took

6.0833 seconds to set up and 68.7833 seconds to solve. Although matrix inversion took 13.1 seconds to set up the same system, it took only 39.033 seconds to solve.

The matrix inversion technique here yielded an approximately 30% reduction in

overall time.

For a 10 x 10 s**ystem**, Solve took 7.01607 and 76.3 seconds compared to

matrix inversion times of 14.8 and 56.5 seconds for set up and solution respectively. Hence the matrix solution yielded only a 14% improvement for this case.

For a 15 x 15 sy**stem,** the Solve function took 10.233 and 272.083 seconds with

matrix inversion taking 30.7167 and 212.35 seconds for set up and solution respectively. In this case, the matrix solution yielded a 16% improvement which means a reduction of 45 seconds in run time.