Cubic Spline Interpolation

by Jennifer Voitle and Edward Lumsdaine

Reference

■ Authors

Jennifer Voitle, Edward Lumsdaine

■ Summary

Constructs and plots natural, clamped, periodic and B cubic splines interpolating functions.

■ Context

NumericalMath`SplineInterpolation`

■ Package Version

1.0

■ History

Procedure PeriodicSpline added 20 March 1991 by J. Voitle

■ Keywords

splines, cubic splines, interpolation, approximation

■ Source

John R. Rice, Numerical Methods, Software and Analysis,
IMSL Reference Edition, McGraw-Hill Book Company, 1983.

■ *Mathematica* Version

1.2-4.0

■ Limitation

In Procedure BSplineInterpolation, the points {x[1],f[1]} and {x[n-1],f[n-1]} are not interpolated. The procedure requires at least four data points.

Discussion

The package SplineInterpolation.m contains procedures for construction, integration and plotting of natural, clamped, periodic and B-cubic interpolating splines.
The procedures require a set of n data points {{x[0],f[0]},{x[1],f[1]}, ... , {x[n],f[n]}}. For the clamped and B-Cubic spline, the endpoint derivatives f'[0], f'[n] are also required.
Usage:

```
NaturalCubicSpline[{{x[0],f[0]},{x[1],f[1]},...,
{x[n],f[n]}}]
ClampedCubicSpline[{{x[0],f[0]},{x[1],f[1]},...,
{x[n],f[n]},{f'[0],f'[n]}}]
PeriodicSpline{{x[0],f[0]},{x[1],f[1]},...,
{x[n],f[n]}}]
BCubicSpline[{{x[0],f[0]},{x[1],f[1]},...,
{x[n],f[n]},{f'[0],f'[n]}}]
```

The natural, clamped and B Spline procedures return a list of spline equations which may be plotted with package procedure PiecewiseCubicPlot. The periodic spline procedure returns only a plot. If the individual splines are desired, the remark brackets (* *) can be removed from the print statement in the package. Spline integration may also be performed (for all except the periodic spline) by IntegrateSpline.

Examples

NaturalCubicSpline constructs the cubic spline with free boundary conditions (S''[0,x[0]] = S''[n-1,x[n]] = 0) on a given set of data. (Note that the data must describe a function (the x's must be unique) and should be ordered such that x[0] < x[1] ... < x[n]. However this last is checked for and unordered data are rearranged first.)
For f[x] = Sin[x] with four points on [0,Pi/2], we first create a data set. Note that uniform spacing is not required, but it is used for convenience here.

```
data = Table[{i, Sin[i]}, {i, 0, π/2, π/(2 3)}]
```

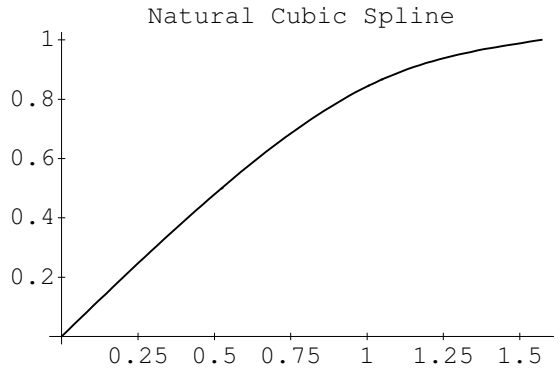
$$\left\{\{0, 0\}, \left\{\frac{\pi}{6}, \frac{1}{2}\right\}, \left\{\frac{\pi}{3}, \frac{\sqrt{3}}{2}\right\}, \left\{\frac{\pi}{2}, 1\right\}\right\}$$

```
Needs["NumericalMath`SplineInterpolation`"]
NaturalCubicSpline[data]
```

```
The spline is constructed as follows:

S[0,x] = 0. + 0.993617 x + 0. x^2 - 0.141114 x^3  for 0. < x < 0.523599

S[1,x] = 0.0124356 + 0.922366 x + 0.136079 x^2 - 0.227744 x^3  for 0.523599 < x < 1.0472

S[2,x] = -0.67269 + 2.88511 x - 1.7382 x^2 + 0.368857 x^3  for 1.0472 < x < 1.5708
```

To plot the spline, call PiecewiseCubicPlot.

```
FreePlot = PiecewiseCubicPlot[PlotLabel → "Natural Cubic Spline"]
```




```
-Graphics-
```

Use IntegrateSpline to compute the integral of the spline just constructed:

```
IntegrateSpline
```

```
The integral of the cubic spline is 0.996214.
```

This compares reasonably well to the true integral,
which is -(Cos[Pi/2] - Cos[0]) = 1.000000.

If the endpoint derivatives are known, a clamped cubic spline may give a better approximation. We can use **Append** to add these points to the data set:

```
data = Append[data, {Sin'[0], Sin'[π/2]}]
```

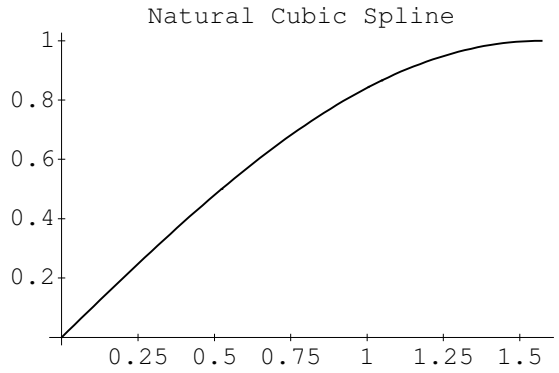
$$\left\{\{0, 0\}, \left\{\frac{\pi}{6}, \frac{1}{2}\right\}, \left\{\frac{\pi}{3}, \frac{\sqrt{3}}{2}\right\}, \left\{\frac{\pi}{2}, 1\right\}, \{1, 0\}\right\}$$

Now call **ClampedCubicSpline** with this argument. We will store the plot as ClampedPlot for later comparison to the plot of the natural cubic spline.

ClampedCubicSpline[data]

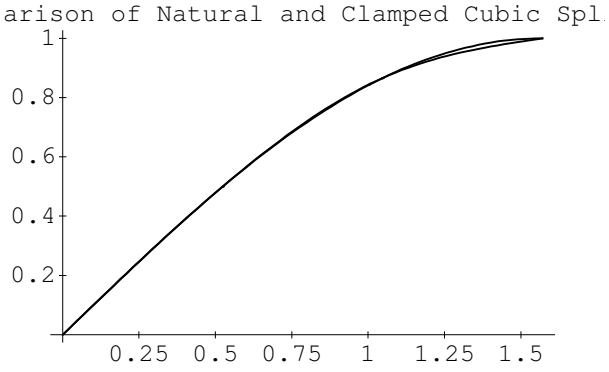
```
The spline is constructed as follows:
                                        2             3
S[0,x] = 0. + 1. x - 0.00142815 x  - 0.161669 x

    for 0. < x < 0.523599
                                                    2
S[1,x] = -0.00605192 + 1.03467 x - 0.0676524 x  -

            3
   0.11951 x   for 0.523599 < x < 1.0472
                                                 2
S[2,x] = -0.093313 + 1.28466 x - 0.30637 x  -

              3
   0.0435235 x   for 1.0472 < x < 1.5708
```

ClampedPlot = PiecewiseCubicPlot[PlotLabel → "Natural Cubic Spline"]



-Graphics-

Show[ClampedPlot, FreePlot,
PlotLabel → "Comparison of Natural and Clamped Cubic Splines"]



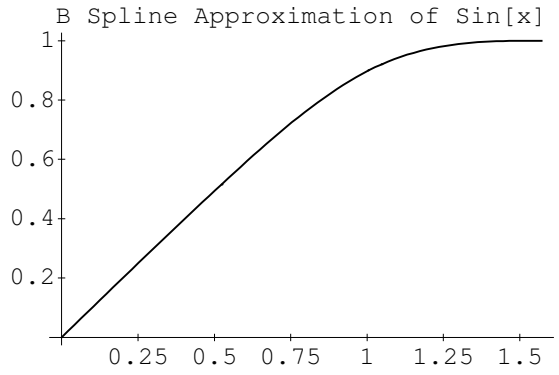
-Graphics-

For this case, the plots differ slightly. To construct a B-cubic spline, use the same data format as for the clamped cubic spline. A B-cubic spline is a combination of natural and clamped splines as it takes on both sets of boundary conditions S"[x[0]] = S"[x[n]] = 0, S'[x[0]] = f'[x[0]], S'[x[n]] = f'[x[n]]. The two extra boundary conditions means that two requirements must be relaxed elsewhere; hence we forego interpolation of the points {{x[1],f[x[1]]}, {x[n-1],f[x[n-1]]}}.

BSplineInterpolation[data]

```
The B spline is constructed as follows:
                                    -19  2             3
S[0,x] = 0. + 1. x + 2.57498 10    x  - 0.054799 x

    for 0. < x < 0.523599
                                                    2
S[1,x] = 0.0636677 + 0.635211 x + 0.696696 x  -

             3
   0.498329 x   for 0.523599 < x < 1.0472
                                                  2
S[2,x] = -1.14381 + 4.09437 x - 2.60655 x  +

             3
   0.553128 x   for 1.0472 < x < 1.5708
```

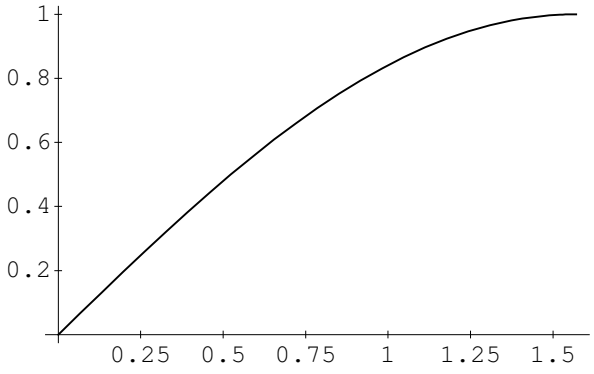
BPlot = PiecewiseCubicPlot[PlotLabel → "B Spline Approximation of Sin[x]"]



-Graphics-

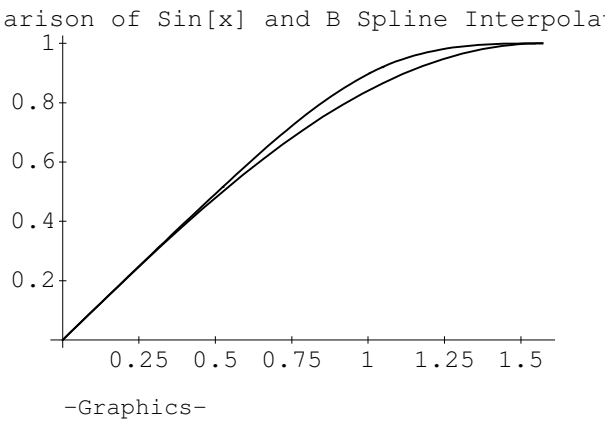
For comparative purposes, let us plot this against the exact function with Show:

yExact = Plot[Sin[x], {x, 0, $\frac{\pi}{2}$}]



-Graphics-

```
Show[yExact, BPlot, PlotLabel → "Comparison of Sin[x] and B Spline Interpolation"]
```



-Graphics-

Note that the B-spline fails to interpolate the points
{{Pi/6,1/2},{Pi/3, Sqrt[3]/2}} as expected.

To graph a smooth closed curve in the plane one may use PeriodicSpline. This procedure constructs parametrized splines x = S1[t], y = S2[t] which satisfy the periodic boundary conditions S[x[0]] = S[x[n]], S'[x[0]] = S'[x[n]], S''[x[0]] = S''[x[n]].
To generate some interesting data which will result in a pretty curve, we use Rice's data below:

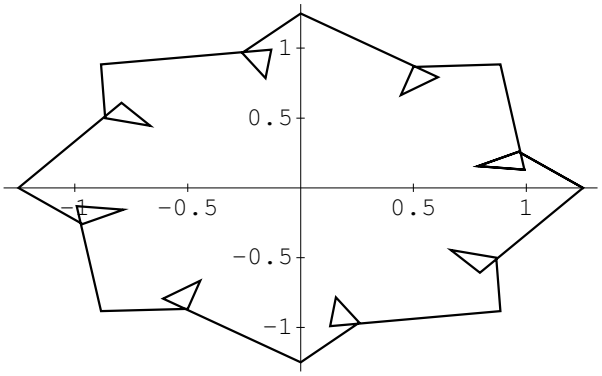
```
Clear[xb, xt, xmplus, ymplus, xmminus, ymminus, yb, yt, theta]
theta := π/n
xt[i_] := rt Cos[2 i theta]
xb[i_] := rb Cos[2 i theta + theta/2]
xmplus[i_] := Cos[2 i theta + theta/2 + theta/6]
xmminus[i_] := Cos[2 i theta + theta/2 - theta/6]
yt[i_] := rt Sin[2 i theta]
yb[i_] := rb Sin[2 i theta + theta/2]
ymplus[i_] := Sin[2 i theta + theta/2 + theta/6]
ymminus[i_] := Sin[2 i theta + theta/2 - theta/6]
rt = 1.25; rb = 0.8;
```

```
n = 8
RiceData = Flatten[N[Table[{{xt[i], yt[i]}, {xmplus[i], ymplus[i]},
     {xb[i], yb[i]}, {xmminus[i], ymminus[i]}}, {i, 0, n}]], 1]
```

```
{{1.25, 0.}, {0.9659258262890682868, 0.2588190451025207623},
 {0.7846282243225843593, 0.1560722576129026143},
 {0.9914448613738104112, 0.1305261922200515915},
 {0.8838834764831844055, 0.8838834764831844055},
 {0.5, 0.8660254037844386468}, {0.4444561864156817799, 0.6651756898420361896},
 {0.6087614290087206396, 0.7933533402912351645},
 {0., 1.25}, {-0.2588190451025207623, 0.9659258262890682869},
 {-0.1560722576129026141, 0.7846282243225843593},
 {-0.1305261922200515914, 0.9914448613738104112},
 {-0.8838834764831844055, 0.8838834764831844055},
 {-0.8660254037844386468, 0.5}, {-0.6651756898420361895, 0.4444561864156817801},
 {-0.7933533402912351644, 0.6087614290087206396},
 {-1.25, 0.}, {-0.9659258262890682869, -0.2588190451025207623},
 {-0.7846282243225843594, -0.1560722576129026139},
 {-0.9914448613738104112, -0.1305261922200515911},
 {-0.8838834764831844055, -0.8838834764831844055},
 {-0.5, -0.8660254037844386468}, {-0.4444561864156817803, -0.6651756898420361893},
 {-0.6087614290087206398, -0.7933533402912351643},
 {0., -1.25}, {0.2588190451025207619, -0.9659258262890682869},
 {0.1560722576129026138, -0.7846282243225843594},
 {0.1305261922200515908, -0.9914448613738104113},
 {0.8838834764831844055, -0.8838834764831844055},
 {0.8660254037844386468, -0.5}, {0.6651756898420361893, -0.4444561864156817804},
 {0.7933533402912351639, -0.6087614290087206403},
 {1.25, 0.}, {0.965925826289068287, 0.2588190451025207614},
 {0.7846282243225843594, 0.1560722576129026137},
 {0.9914448613738104112, 0.1305261922200515911}}
```

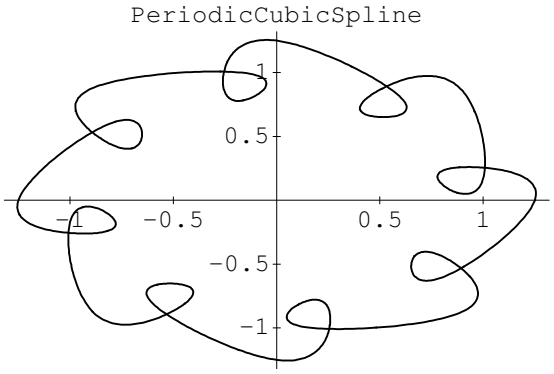
Let us make a ListPlot of the above data to get an idea of the results to expect:

```
ListPlot[RiceData, PlotJoined → True]
```



Note that we do get a closed curve, but it is certainly not smooth. Call **PeriodicSpline** with the data, dropping the last quartet (it is a repeat of the first quartet, and we don't want extra lines on our graph).

```
RiceData = Drop[RiceData, -4]
Timing[PeriodicSpline[RiceData, PlotPoints → 40]]
```



Implementation

■ Preparation

■ Create the context for this notebook:

```
BeginPackage["NumericalMath`SplineInterpolation`"]
```

■ **Usage messages:**

```
SplineInterpolation::"usage" =
 "The package SplineInterpolation contains code for  construction of
   natural, clamped, periodic and B cubic splines. A plotting utility,
   PiecewiseCubicPlot, and integration procedure, IntegrateSpline,
   are also provided.  For information on usage of these procedures,
   type ?NaturalCubicSpline, ?ClampedCubicSpline, ?PeriodicSpline,
   ?BSplineInterpolation, ?PiecewiseCubicPlot or ?IntegrateSpline. "
NaturalCubicSpline::"usage" = "NaturalCubicSpline[{{x0,y0},{x1,y1},...{xn,yn}}]
   constructs and prints a natural cubic spline interpolating the given   data
   points. The natural spline boundary conditions   S''[x0] = S''[xn]=0 are used."
ClampedCubicSpline::"usage" = "ClampedCubicSpline[{{x0,y0},{x1,y1},...
   {xn,yn},{y'0,y'n}}]   constructs and prints a clamped cubic
   spline interpolating the given   data points. The clamped spline
   boundary conditions   S'[x0] = f'[x0],S'[xn] = f'[xn] are used."
BSplineInterpolation::"usage" = "BSplineInterpolation[{{x0,y0},{x1,y1},...{xn,
   yn},{y'0,y'n}}]   constructs and prints a B-spline interpolating the given
   data points. Both natural and clamped spline boundary    conditions S'[
   x0] = f'[x0]; S'[xn] = f'[xn];   S''[x0] = S''[xn] = 0 are used.  The B-
   Spline does   not interpolate the given data at the points   {{x1,f[x1]},
   {xn-1,f[xn-1]}}.      NOTE:  n must be >2 for BSplineInterpolation."
PeriodicSpline::"usage" = "PeriodicSpline[{{x0,y0},{x1,y1},...{xn,yn}}]
   constructs and plots a periodic spline.  This procedure is used for
   plotting smooth, closed curves.  The periodic boundary conditions
   S[x0] = S[xn], S'[x0] = S'[xn] and S''[x0] = S''[xn] are used."
IntegrateSpline::"usage" = "IntegrateSpline computes the integral
   of the spline     over the interval [x[0],x[n]].
   It is not intended for use     with PeriodicSpline."
CubicSpline::"smallnerr" = "Number of data points, n =`1` is  insufficient
   for spline interpolation: you must provide at least `2` data points."
CubicSpline::"nonfunctionerr" = "The x coordinates provided `1`
   do not describe a function.  The x's must be unique."
PiecewiseCubicPlot::"usage" = "PiecewiseCubicPlot[]    constructs a plot of
   the constructed spline.  The option    PlotPoints->num may be included.
   Otherwise, the default    value of PlotPoints is used.  Note that setting num=
   1 yields    a piecewise linear plot, identical to the result    obtained via
   ListPlot, with PlotJoined->True.       NOTE:  Either NaturalCubicSpline or
   ClampedCubicSpline    must be executed prior to calling PiecewiseCubicPlot."
```

Definitions

```
Begin["`Private`"]
Unprotect[ClampedCubicSpline, NaturalCubicSpline, IntegrateSpline,
 PiecewiseCubicPlot, BSplineInterpolation, S, PeriodicSpline]
```

The cubic spline is constructed as piecewise cubics **S[i,x]** on subintervals {[x[i], x[i+1]} for i = 0 to n. **S[i,x]** is defined in the shifted power form to reduce the number of unknowns.
Also, t[i] is used for x[i] since x cannot serve both as a subscripted and unsubscripted variable at once.

```
S[i_, x_] := a[i] + b[i] (x - t[i]) + c[i] (x - t[i])^2 + d[i] (x - t[i])^3
```

The procedure **WriteEquations** is common to many of the routines. In **WriteEquations**, a list of the spline conditions is built up and stored as **EquationList**. These conditions include interpolation at all points and continuity of the 0th, 1st and 2nd derivatives at the interior points x[i], i = 1,..., n - 1. Thus **EquationList** stores all equations save for the individual boundary conditions, which are applied later. **EquationList** contains the system of linear equations which is to be solved for the unknown spline coefficients.

```
WriteEquations :=
 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]]];
   EquationList = Append[EquationList,
     Expand[∂x S[i, x] /. x → t[i + 1]] == Expand[∂x S[i + 1, x] /. x → t[i + 1]]];
   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}]]
```

Procedure **MakeUnknowns** builds up a list of all unknown coefficients b[i], c[i] and d[i]. Note that the a[i] are known since the shifted power form is used for S[i,x]. The list is stored as
Unknowns.

```
MakeUnknowns :=
 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}]]
```

Procedure **SolvetheSystem** is used only for natural and clamped cubic splines. **SolvetheSystem** stores the solution of **Solve[EquationList,Unknowns]** in the variable **Result**, and then assigns the unknowns b[i], c[i] and d[i] to their rules from **Solve**. The spline is then printed out in terms of the subsplines S[i,x] on [ti,ti+1]. The variable **Global`x** is used as the argument in S[i,x] as we are in the **SplineInterpolation`Private`** context and x would be represented as **SplineInterpolation`Private`x** were we to just use x as the argument.

```
SolvetheSystem :=
 Block[{}, 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["The spline is constructed as follows: "]; Do[Print["S[", i, ",x] = ",
    Expand[S[i, Global`x]], "  for ", t[i], " < x < ", t[i + 1]], {i, 0, n - 1}]]
```

SolvetheSystem must be slightly modified when called by **BSplineInterpolation**, since slightly different conditions were applied. The first two elements in **Result** will be the two unknown a[i]'s rather than b[0] and b[1] as they were in **SolvetheSystem**. This is basically the only difference in the two procedures. The following procedure, **SolvetheBSplineSystem**, is the B-spline analog of **SolvetheSystem** .

```
SolvetheBSplineSystem := Block[{}, Result = N[Solve[EquationList, Unknowns]];
  a[1] = Result[[1, 1, 2]]; a[n - 1] = Result[[1, 2, 2]]; Do[b[i - 3] = Result[[1, i, 2]];
   c[i - 3] = Result[[1, i + n, 2]]; d[i - 3] = Result[[1, i + 2 n, 2]], {i, 3, 3 + n - 1}];
  Print["The B spline is constructed as follows: "]; Do[Print["S[", i, ",x] = ",
    Expand[S[i, Global`x]], "  for ", t[i], " < x < ", t[i + 1]], {i, 0, n - 1}]]
```

The Periodic Spline also requires slight modification of **SolvetheSystem** . Since there are two splines x=S1[t], y = S2[t] constructed, the spline output is suppressed.

```
SolvePeriodicSystem := Block[{}, LHS = Table[Row[i], {i, 0, 3 n - 1}];
  RHS = Table[constant[i], {i, 0, 3 n - 1}]; Result = LinearSolve[LHS, RHS];
  Do[b[i - 1] = Result[[i]]; c[i - 1] = Result[[i + n]]; d[i - 1] = Result[[i + 2 n]], {i, 1, n}];]
```

The procedure sets up the periodic spline equations and calls upon the periodic spline solver for solution.

```
WriteandSolvePeriodicEquations :=
 Block[{i}, Clear[Row, Equation, b, c, d]; 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];
   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; 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] - S[0, t[0]]; constant[3 n - 3] = a[0] - a[n - 1];
  Equation[3 n - 2] = Expand[∂x S[0, x] /. x → t[0]] - Expand[∂x S[n - 1, x] /. x → t[n]];
  constant[3 n - 2] = 0;
  Equation[3 n - 1] = Expand[∂{x,2} S[0, x] /. x → t[0]] - Expand[∂{x,2} S[n - 1, x] /. x → t[n]];
  constant[3 n - 1] = 0; Do[Row[j] = {};
   Row[j] = Append[Row[j], Table[Coefficient[Equation[j], b[i]], {i, 0, n - 1}]];
   Row[j] = Append[Row[j], Table[Coefficient[Equation[j], c[i]], {i, 0, n - 1}]];
   Row[j] = Append[Row[j], Table[Coefficient[Equation[j], d[i]], {i, 0, n - 1}]];
   Row[j] = Flatten[Row[j]], {j, 0, 3 n - 1}]; MakeUnknowns; SolvePeriodicSystem]
```

The following procedure, NaturalCubicSpline, performs construction of a natural cubic spline. The user calls it with a list of {x,y} data pairs describing his data. The syntax of the call is

```
NaturalCubicSpline[{{x[0],y[0]},{x[1],y[1]},
                          ... , {x[n],y[n]}}]
```

These data need not be evenly spaced, but must define a function: that is, each x[i] must be unique. A check is provided for uniqueness of the x[i]'s through the built**t-in** Union function. The input data list (slist) is renamed as SplineData in case alterations must be performed. A list of the x[i]'s is constructed as local variable xlist. The com**mand** **xlist != Union[**xlist] evaluates to True if any of the x[i]'s are duplicated; it is False otherwise. If the test passes, a message (CubicSpline::nonfunctionerr) is printed and a Break is executed.
Additionally, the data must be provided in an ordered form such that x[0]<x[1] < ... < x[n]. Should the user provide data which do not satisfy this condition, no error message is printed. The in***trinsic Math***ematica pr**oced**ure Sort operates on the data and renders them ordered. Once the data have been deemed acceptable, the spline construction is continued. First, the {x,y} data are stored as {t[i],a[i]} pai**rs, and WriteE**quations is called. The free boundary conditions are appe**nded to Equa**tionList, which completes the list of spline conditions. The solution of the linear system is perfo**rmed in Solvet**heSystem.

```
NaturalCubicSpline[slist_List] :=
 Block[{i}, Clear[a, b, c, d]; SplineData = N[slist]; MinNumberofPoints = 3;
  If[Length[SplineData] < MinNumberofPoints, Message[CubicSpline::"smallnerr",
    Length[SplineData], MinNumberofPoints]; Return[Hold[NaturalCubicSpline[data]]]];
  If[SplineData ≠ Sort[SplineData], SplineData = Sort[SplineData]];
  xlist = Table[SplineData[[i, 1]], {i, Length[SplineData]}]; If[xlist ≠ Union[xlist],
   Message[CubicSpline::"nonfunctionerr", xlist]; Return[]]; n = Length[SplineData] - 1;
  Do[t[i] = SplineData[[i + 1, 1]]; a[i] = SplineData[[i + 1, 2]], {i, 0, n}]; WriteEquations;
  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];
  MakeUnknowns; SolvetheSystem]
```

The **ClampedCubicSpline** function detailed below is very similar to the **NaturalCubicSpline** procedure, except that in addition to the {x,y} data, the user must also provide derivative information f'[x[0]] and f'[x[n]]. This information is added to the end of the data list. The syntax of the call is

```
ClampedCubicSpline[{{x[0],y[0]},{x[1],y[1]},
    ... , {x[n],y[n]},{f'[x[0]],f'[x[n]]}}]
```

Since we will drop this derivative information from the list once read, it is necessary to rename the data list. This is the purpose of the line SplineData = slist.
The derivative values f'[x[0]] and f'[x[n]] are stored in the local variables LeftDeriv and RightDeriv respectively. Then they are dropped from the data list, leaving just the {x,y} point information. These values are stored as {t[i],a[i]} pairs in exactly the same way as do**ne in NaturalCubic**Spl**ine. WriteEqu**ations is called, which accumulates all equatio**ns in Equati**onList save for the clamped boundary conditions, which are append**ed to Equati**onList. The linear system is then solved by proc**edure Solvethe**System previously described.

```
ClampedCubicSpline[slist_List] :=
 Block[{i}, Clear[a, b, c, d]; SplineData = N[slist]; MinNumberofPoints = 4;
  If[Length[SplineData] < MinNumberofPoints, Message[CubicSpline::"smallnerr",
    Length[SplineData], MinNumberofPoints]; Return[Hold[ClampedCubicSpline[data]]]];
  LeftDeriv = SplineData[[Length[SplineData], 1]];
  RightDeriv = SplineData[[Length[SplineData], 2]]; SplineData = Drop[SplineData, -1];
  If[SplineData ≠ Sort[SplineData], SplineData = Sort[SplineData]];
  xlist = Table[SplineData[[i, 1]], {i, Length[SplineData]}];
  If[xlist ≠ Union[xlist], Message[CubicSpline::"nonfunctionerr", xlist]; Return[]];
  n = Length[SplineData] - 1;
  Do[t[i] = SplineData[[i + 1, 1]]; a[i] = SplineData[[i + 1, 2]], {i, 0, n}]; WriteEquations;
  EquationList = Append[EquationList, Expand[∂x S[0, x] /. x → t[0]] == LeftDeriv];
  EquationList = Append[EquationList, Expand[∂x S[n - 1, x] /. x → t[n]] == RightDeriv];
  MakeUnknowns; SolvetheSystem]
```

Interpolation by Cubic B-Splines is performed in the following routine, BSplineInterpolation. The arguments to this procedure are identical to the arguments in ClampedCubicSpline; hence the syntax of the call is

```
BSplineInterpolation[{{x[0],y[0]},{x[1],y[1]},
    ... , {x[n],y[n]},{f'[x[0]],f'[x[n]]}}]
```

The B-Spline is very similar to the preceding splines, except both the natural and fixed boundary conditions are applied. We then have two less degrees of freedom, so must drop two other requirements. It is common to relax requirement on interpolation at two of the input points. Here we have arbitrarily decided to leave the points {t[1],a[1]} and {t[n-1],a[n-1] }uninterpolated. This could be changed by the reader if desired. The proce**dure BSplineInterpol**ation will not work on a set of n<=3 data points without modification (if n = 3, we cannot drop the points t[1] and t[n-1] since they are the same point, hence we would also need to drop one of the end points.) It is anticipated that most users will have more than 3 data points. An error check is performed to ensure that 4 or more points are used. Inclusion of the derivatives makes the minimal length of the input list equal to five. Should a smaller list be provided, a message (BSpline::smallnerr) is printed, a Break is executed and the output Hold[BSplineInterpolation[data]] is returned.
The procedure is very similar to the other spline procedures previously described, the sole differences being the boundary conditions and the two undetermined a[i]'s. Hence this UnknownList will also include a[1] and a[n-1], which are prepended to the UnknownList written by procedure MakeUnknowns. The solution is executed by procedure SolveBSplineUnknowns.

```
BSplineInterpolation[slist_List] :=
 Block[{i}, Clear[a, b, c, d]; SplineData = N[slist]; MinNumberofPoints = 5;
  If[Length[SplineData] < MinNumberofPoints, Message[CubicSpline::"smallnerr",
    Length[SplineData], MinNumberofPoints]; Return[Hold[BSplineInterpolation[data]]]];
  LeftDeriv = SplineData[[Length[SplineData], 1]];
  RightDeriv = SplineData[[Length[SplineData], 2]]; SplineData = Drop[SplineData, -1];
  If[SplineData ≠ Sort[SplineData], SplineData = Sort[SplineData]];
  xlist = Table[SplineData[[i, 1]], {i, Length[SplineData]}];
  If[xlist ≠ Union[xlist], Message[CubicSpline::"nonfunctionerr", xlist]; Return[]];
  n = Length[SplineData] - 1; Do[t[i] = SplineData[[i + 1, 1]], {i, 0, n}];
  a[0] = SplineData[[1, 2]]; a[n] = SplineData[[n + 1, 2]];
  Do[a[i] = SplineData[[i + 1, 2]], {i, 2, n - 2}]; WriteEquations;
  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];
  EquationList = Append[EquationList, Expand[∂x S[0, x] /. x → t[0]] == LeftDeriv];
  EquationList = Append[EquationList, Expand[∂x S[n - 1, x] /. x → t[n]] == RightDeriv];
  MakeUnknowns; Unknowns = Prepend[Unknowns, a[n - 1]];
  Unknowns = Prepend[Unknowns, a[1]]; SolvetheBSplineSystem]
```

The procedure **PeriodicSpline** is used for creation of smooth closed plane curves. The data satisfy the periodic boundary condtions S[0,x[0]] = S[n-1,x[n]], S'[0,x[0]] = S'[n-1,x[n]], and S"[0,x[0]] = S"[n-1,x[n]]. The input {x[i],y[i]} data are used to create two splines x = S1[t], y = S2[t]. The parametrizations are then plotted. Since **PeriodicSpline** must construct two splines, it takes longer than the other spline routines. The splines are not printed out here; the only output is the plot. If splines are desired, the user can remove the remark statements around the Print statements for the splines. The S1[t] values are stored in the list **AbscissaPoints** while the S2[t] values are stored in **Ordinate-Points**. These lists are then merged in **SplineData** which is plotted.

```
PeriodicSpline[slist_List, opts___Rule] := Block[{i, plotpoints, X, Y},
  Clear[a, b, c, d]; plotpoints = PlotPoints /. {opts} /. Options[Plot];
  SplineData = slist; MinNumberofPoints = 3; If[Length[SplineData] < MinNumberofPoints,
   Message[CubicSpline::"smallnerr", Length[SplineData], MinNumberofPoints];
   Return[Hold[NaturalCubicSpline[data]]]]; n = Length[SplineData];
  Do[t[i] = i + 1; a[i] = SplineData[[i + 1, 1]]; aa[i] = SplineData[[i + 1, 2]], {i, 0, n - 1}];
  t[n] = n + 1; a[n] = a[0]; WriteandSolvePeriodicEquations; title = "PeriodicCubicSpline";
  Do[X[i] = Drop[Table[N[S[i, x]], {x, t[i], t[i + 1], (t[i + 1] - t[i])/plotpoints}], -1], {i, 0, n - 2}];
  X[n - 1] = Table[N[S[n - 1, x]], {x, t[n - 1], t[n], (t[n] - t[n - 1])/plotpoints}];
  AbscissaPoints = Flatten[Table[X[i], {i, 0, n - 1}], 1]; Clear[a, b, c, d];
  Do[a[i] = aa[i], {i, 0, n - 1}]; a[n] = a[0]; WriteandSolvePeriodicEquations;
  Do[Y[i] = Drop[Table[N[S[i, x]], {x, t[i], t[i + 1], (t[i + 1] - t[i])/plotpoints}], -1], {i, 0, n - 2}];
  Y[n - 1] = Table[N[S[n - 1, x]], {x, t[n - 1], t[n], (t[n] - t[n - 1])/plotpoints}];
  OrdinatePoints = Flatten[Table[Y[i], {i, 0, n - 1}], 1]; SplineData =
   Table[{AbscissaPoints[[i]], OrdinatePoints[[i]]}, {i, Length[OrdinatePoints]}];
  ListPlot[SplineData, PlotJoined → True, PlotLabel → title]]
```

Spline plots are generated by the included procedure **PiecewiseCubicPlot[]**. To plot the piecewise cubics which comprise the splines, the idea is to define S[i,x] only on its subinterval [t[i],t[i+1]]. The inherent ***Mathematica*** function **Which** is unsatisfactory for this purpose since the number of splines (n) is arbitrary. It is not possible (as of v. 1.2) to pass a list of conditions to **Which**. However, **ListPlot** can handle lists, which is why it is used here. The procedure **PiecewiseCubicPlot** has the option **PlotPoints**, which is by default equal to the value of the **PlotPoints** option in **Plot**. (Note that **PlotPoints->1** yields a piecewise linear function which is identical to the output from **ListPlot** with **PlotJoined->True**.)

The local variable plot[i] holds a table of points {x,S[i,x]} for x running from t[i] to t[i+1] in increments of plotpoints. Hence we represent the continous cubic S[i,x] by a list of points, which is the same way **Plot** works. The list of these lists is Flattened and plotted by **ListPlot**.

```
PiecewiseCubicPlot[opts___Rule] :=
 Block[{i, plotpoints}, plotpoints = PlotPoints /. {opts} /. Options[Plot];
  title = PlotLabel /. {opts} /. Options[Plot];
  Do[plot[i] = Table[N[{x, S[i, x]}], {x, t[i], t[i + 1], (t[i + 1] - t[i])/plotpoints}], {i, 0, n - 1}];
  SplinePlot = ListPlot[Flatten[Table[plot[i], {i, 0, n - 1}], 1],
    PlotJoined → True, PlotLabel → title]]
```

The procedure **IntegrateSpline** provides integration of the spline. Since the analytic result is known, it is used to avoid the time-consuming process of loading IntegralTables. The result is stored and printed as **SplineIntegral**.

```
IntegrateSpline :=
 Block[{i}, SplineIntegral = Sum[a[i] (t[i + 1] - t[i]) + 1/2 b[i] (t[i + 1] - t[i])^2 +
      1/3 c[i] (t[i + 1] - t[i])^3 + 1/4 d[i] (t[i + 1] - t[i])^4, {i, 0, n - 1}];
  Print["The integral of the cubic spline is ", SplineIntegral, "."]]

End[]
Protect[NaturalCubicSpline, ClampedCubicSpline,
 BSplineInterpolation, PiecewiseCubicPlot, PeriodicSpline, IntegrateSpline]
```

■ Finish

```
EndPackage[]
```