

Splines

We now do piecewise interpolation using **cubic splines**. This results in a smooth interpolating curve which does not have the degree of oscillation that many polynomial interpolants have.

```
> restart;
```

To gain access to the [Spline](#) command, we load the [CurveFitting](#) package.

```
> with(CurveFitting);
```

```
[ArrayInterpolation, BSpline, BSplineCurve, Interactive, LeastSquares, PolynomialInterpolation,  
RationalInterpolation, Spline, ThieleInterpolation]
```

We are going to attempt to draw the back of the duck on page 97 of the text. We first enter the data from Table 3.13 on the same page. We can access the data two different ways. The first is to create separate lists for the x and $f(x)$ (or y) values.

```
> X:= [.9, 1.3, 1.9, 2.1, 2.6, 3.0, 3.9, 4.4, 4.7, 5.0, 6.0, 7.0, 8.0, 9.2, 10.5,  
11.3, 11.6, 12.0, 12.6, 13.0, 13.3];
```

```
X := [0.9, 1.3, 1.9, 2.1, 2.6, 3.0, 3.9, 4.4, 4.7, 5.0, 6.0, 7.0, 8.0, 9.2, 10.5, 11.3, 11.6, 12.0, 12.6, 13.0,  
13.3]
```

```
> Y:= [1.3, 1.5, 1.85, 2.1, 2.6, 2.7, 2.4, 2.15, 2.05, 2.1, 2.25, 2.3, 2.25, 1.95,  
1.4, .9, .7, .6, .5, .4, .25];
```

```
Y := [1.3, 1.5, 1.85, 2.1, 2.6, 2.7, 2.4, 2.15, 2.05, 2.1, 2.25, 2.3, 2.25, 1.95, 1.4, 0.9, 0.7, 0.6, 0.5, 0.4,  
0.25]
```

At this point we digress to see what kind of result Lagrange interpolation gives using the [PolynomialInterpolation](#) command from the **CurveFitting** package (as distinct from the **NumericalAnalysis** package). We need to increase [Digits](#) here.

```
> Digits:=25;
```

```
Digits := 25
```

```
> L:=PolynomialInterpolation(X,Y,x);
```

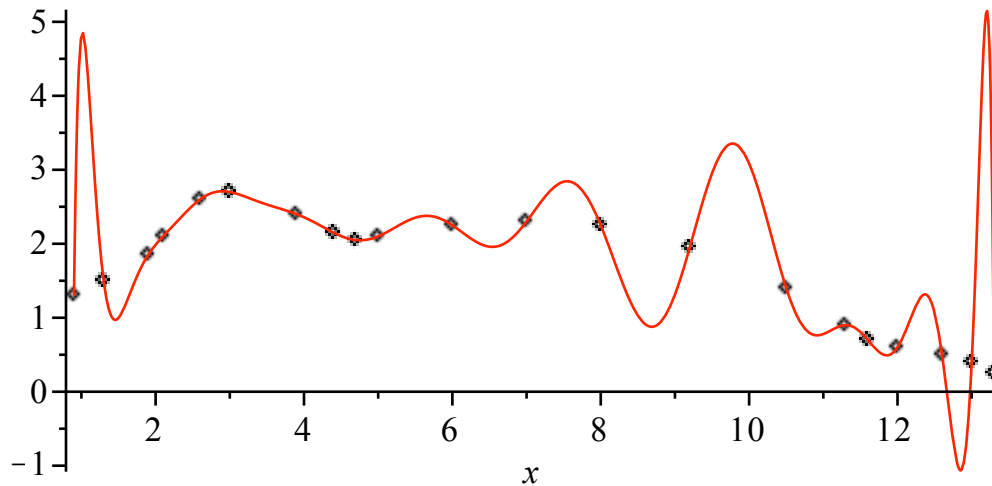
```
L := 52462.58032870225450359861 x - 1.286834057790684836071605 105 x2  
+ 4.279489271446121762378730 10-9 x19 + 0.00001110758954236296207707053 x17  
- 2.772830850465977395651647 10-7 x18 + 1.899949932380636671311183 105 x3  
- 1.898509728616989751143148 105 x4 + 1.367779002266297072085843 105 x5  
- 73866.56567846122183344343 x6 + 30677.63497406521236486221 x7  
- 9968.978857330867177561784 x8 + 2564.201308195982818178016 x9  
- 525.8127535066468982152202 x10 + 86.25139799248564473512300 x11  
- 11.31605155586641405206707 x12 + 1.182840316499033710862577 x13  
- 0.09769038007304036390115707 x14 + 0.006285899667516608381788952 x15  
- 0.0003081591893084367616781800 x16 - 3.074530780107976382929712 10-11 x20  
- 9652.785079125204480303104
```

We plot this Lagrange polynomial along with the points using [pointplot](#) from the [plots](#) package.

```

> with(plots):
> pp:=pointplot({seq([X[i],Y[i]],i=1..21)}):
> Lp:=plot(L,x=.9..13.3,scaling=constrained):
> display(pp,Lp);

```



* Points 1
— Curve 1

Not too much like the duck's back. For working with splines, we can reset **Digits** back to its default of 10.

```
> Digits:=10;
```

Digits := 10

The arguments for the first form of the **natural spline** are the lists of x and y values, followed by the variable used to write the polynomial (here a polynomial in x).

```
> S:=Spline(X,Y,x);
```

$$\begin{aligned}
& 0.814338535669392 + 0.539623849256231 x - 0.247649057851443 (x - 0.9)^3 \\
& 0.953022008066200 + 0.420752301487538 x - 0.297178869421732 (x - 1.3)^2 + 0.946912093052318 \\
& -0.214925165488130 + 1.08680271867796 x + 1.40726289807244 (x - 1.9)^2 - 2.95638245731128 \\
& -0.619378164362130 + 1.29494198302959 x - 0.366566576314327 (x - 2.1)^2 - 0.446634779489688 \\
& 1.05716176254522 + 0.593399322097992 x - 1.03651874554886 (x - 2.6)^2 + 0.445051100759697 \\
& 2.76657343792932 - 0.0221911459764410 x - 0.502457424637224 (x - 3.0)^2 + 0.174159870143962 \\
& 4.36328350090711 - 0.503406025873617 x - 0.0322257752485276 (x - 3.9)^2 + 0.078075653991522 \\
& 4.24913026676541 - 0.477075060628502 x + 0.0848877057387567 (x - 4.4)^2 + 1.31417128415047 \\
& 2.38518609518372 - 0.0713161904646223 x + 1.26764186147418 (x - 4.7)^2 - 1.58121890345515 \\
& 0.788300887565041 + 0.262339822486992 x - 0.155455151635463 (x - 5.0)^2 + 0.043115329148470 \\
& 1.76534696003113 + 0.0807755066614790 x - 0.0261091641900502 (x - 6.0)^2 - 0.004666342471428 \\
& 2.19809294393035 + 0.0145581508670923 x - 0.0401081916043365 (x - 7.0)^2 - 0.024449959262755 \\
& 3.36206488103879 - 0.139008110129848 x - 0.113458069392604 (x - 8.0)^2 + 0.0174706898617866 \\
& 5.03967368751645 - 0.335834096469179 x - 0.0505635858901720 (x - 9.2)^2 - 0.012727908254745 \\
& 6.98421410366947 - 0.531829914635187 x - 0.100202428083680 (x - 10.5)^2 - 0.0203252232779193 \\
& 9.16231397936830 - 0.731178228262681 x - 0.148982963950687 (x - 11.3)^2 + 1.21340500868023 \\
& 6.41820438975741 - 0.492948654289432 x + 0.943081543861517 (x - 11.6)^2 - 0.839274770344844 \\
& 2.29602370758892 - 0.141335308965743 x - 0.0640481805522953 (x - 12.0)^2 + 0.036382085084593 \\
& 2.75414596908792 - 0.178900473737137 x + 0.00143957259997275 (x - 12.6)^2 - 0.44797097064282 \\
& 5.50607346035431 - 0.392774881565716 x - 0.536125592171421 (x - 13.0)^2 + 0.595695102412690
\end{aligned}$$

$S :=$

We expand this result.

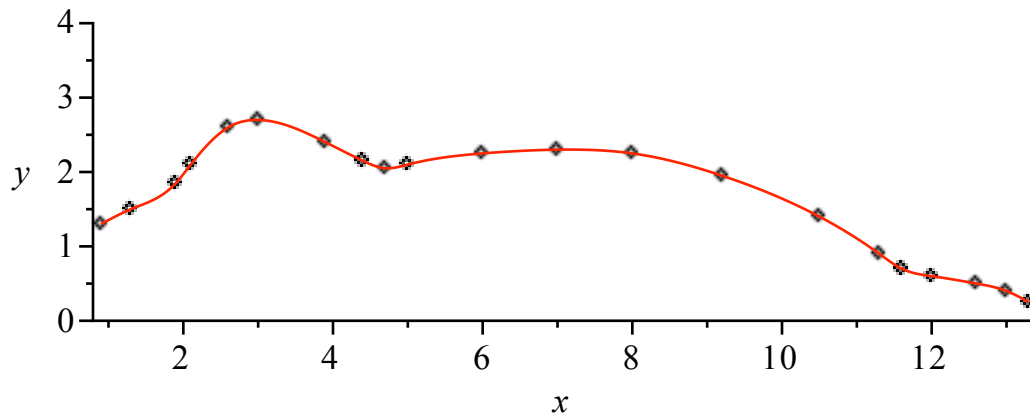
> S:=expand(S);

S :=

$$\begin{aligned} & 0.994874698843094 - 0.0621633613227766 x - 0.247649057851443 x^3 + 0.668652456198897 x^2 \\ & - 1.62957614969247 + 5.99426167375929 x - 3.99013603232577 x^2 + 0.946912093052318 x^3 \\ & 25.1431211712514 - 36.2784183066785 x + 18.2586429047467 x^2 - 2.95638245731128 x^3 \\ & 1.90034792694569 - 3.07445652909881 x + 2.44723253447071 x^2 - 0.446634779489688 x^3 \\ & - 13.7719231043175 + 15.0089331223587 x - 4.50791733147449 x^2 + 0.445051100759697 x^3 \\ & - 6.45785987769265 + 7.69486989573386 x - 2.06989625593288 x^2 + 0.174159870143962 x^3 \\ & - 0.758240259746141 + 3.31054711269808 x - 0.945710926949345 x^2 + 0.0780756539915228 x^3 \\ & - 106.053810419206 + 75.1029813123295 x - 17.2621732450474 x^2 + 1.31417128415047 x^3 \\ & 194.554285028573 - 116.774526420295 x + 23.5628284001919 x^2 - 1.58121890345515 x^3 \\ & - 8.48749404688038 + 5.05054102497693 x - 0.802185088862525 x^2 + 0.0431153291484708 x^3 \\ & 1.83334702301793 - 0.109879509972225 x + 0.0578850002956675 x^2 - 0.00466634247142876 x^3 \\ & 8.61912758244311 - 3.01807117829730 x + 0.473340952913535 x^2 - 0.0244499592627558 x^3 \\ & - 12.8442447693226 + 5.03069345361485 x - 0.532754626075483 x^2 + 0.0174706898617866 x^3 \\ & 10.6710412008438 - 2.63733458013506 x + 0.300726681940811 x^2 - 0.0127279082547458 x^3 \\ & 19.4658830045450 - 5.15014652404970 x + 0.540042105170777 x^2 - 0.0203252232779193 x^3 \\ & - 1760.67976749717 + 467.454893432157 x - 41.2834127582104 x^2 + 1.21340500868023 x^3 \\ & 1443.33988886395 - 361.170879764683 x + 30.1498435518621 x^2 - 0.839274770344844 x^3 \\ & - 69.7951573181189 + 17.1128817808337 x - 1.37380324359766 x^2 + 0.0363820850845934 x^3 \\ & 899.093070885677 - 213.574791601023 x + 16.9347422628989 x^2 - 0.447970970642828 x^3 \\ & - 1393.84129161730 + 315.563907438125 x - 23.7682345862663 x^2 + 0.595695102412690 x^3 \end{aligned}$$

Let's plot this along with the points.

```
> Sp:=plot(S,x=.9..13.3,y=0..4,scaling=constrained):  
> display(pp,Sp);
```



* Points 1 — Curve 1

Now this looks pretty good. We evaluate this function at $x = 6.4$.

```
> eval(S,x=6.4);
                2.27783409047601
```

We can also enter our data as a list of ordered pairs of the form $[x, y]$.

```
> points:=seq([X[i],Y[i]],i=1..21);
points := [[0.9, 1.3], [1.3, 1.5], [1.9, 1.85], [2.1, 2.1], [2.6, 2.6], [3.0, 2.7], [3.9, 2.4], [4.4, 2.15],
           [4.7, 2.05], [5.0, 2.1], [6.0, 2.25], [7.0, 2.3], [8.0, 2.25], [9.2, 1.95], [10.5, 1.4], [11.3, 0.9],
           [11.6, 0.7], [12.0, 0.6], [12.6, 0.5], [13.0, 0.4], [13.3, 0.25]]
```

The alternate form of the arguments replaces the x and y lists with the list of points.

```
> SS:=expand(Spline(points,x));
```

$$\begin{aligned}
SS := & \left\{ \begin{aligned}
& 0.994874698843094 - 0.0621633613227766 x - 0.247649057851443 x^3 + 0.668652456198897 x^2 \\
& - 1.62957614969247 + 5.99426167375929 x - 3.99013603232577 x^2 + 0.946912093052318 x^3 \\
& 25.1431211712514 - 36.2784183066785 x + 18.2586429047467 x^2 - 2.95638245731128 x^3 \\
& 1.90034792694569 - 3.07445652909881 x + 2.44723253447071 x^2 - 0.446634779489688 x^3 \\
& - 13.7719231043175 + 15.0089331223587 x - 4.50791733147449 x^2 + 0.445051100759697 x^3 \\
& - 6.45785987769265 + 7.69486989573386 x - 2.06989625593288 x^2 + 0.174159870143962 x^3 \\
& - 0.758240259746141 + 3.31054711269808 x - 0.945710926949345 x^2 + 0.0780756539915228 x^3 \\
& - 106.053810419206 + 75.1029813123295 x - 17.2621732450474 x^2 + 1.31417128415047 x^3 \\
& 194.554285028573 - 116.774526420295 x + 23.5628284001919 x^2 - 1.58121890345515 x^3 \\
& - 8.48749404688038 + 5.05054102497693 x - 0.802185088862525 x^2 + 0.0431153291484708 x^3 \\
& 1.83334702301793 - 0.109879509972225 x + 0.0578850002956675 x^2 - 0.00466634247142876 x^3 \\
& 8.61912758244311 - 3.01807117829730 x + 0.473340952913535 x^2 - 0.0244499592627558 x^3 \\
& - 12.8442447693226 + 5.03069345361485 x - 0.532754626075483 x^2 + 0.0174706898617866 x^3 \\
& 10.6710412008438 - 2.63733458013506 x + 0.300726681940811 x^2 - 0.0127279082547458 x^3 \\
& 19.4658830045450 - 5.15014652404970 x + 0.540042105170777 x^2 - 0.0203252232779193 x^3 \\
& - 1760.67976749717 + 467.454893432157 x - 41.2834127582104 x^2 + 1.21340500868023 x^3 \\
& 1443.33988886395 - 361.170879764683 x + 30.1498435518621 x^2 - 0.839274770344844 x^3 \\
& - 69.7951573181189 + 17.1128817808337 x - 1.37380324359766 x^2 + 0.0363820850845934 x^3 \\
& 899.093070885677 - 213.574791601023 x + 16.9347422628989 x^2 - 0.447970970642828 x^3 \\
& - 1393.84129161730 + 315.563907438125 x - 23.7682345862663 x^2 + 0.595695102412690 x^3
\end{aligned} \right.
\end{aligned}$$

Subtraction is used to show that both results are identical.

```
> S-SS;
```

0

We can find splines of any positive degree by adding a fourth argument to [Spline](#), which give the degree of the polynomial we wish to use for each piece. The default is **degree=3** for cubic splines. Let's try quadratic splines.

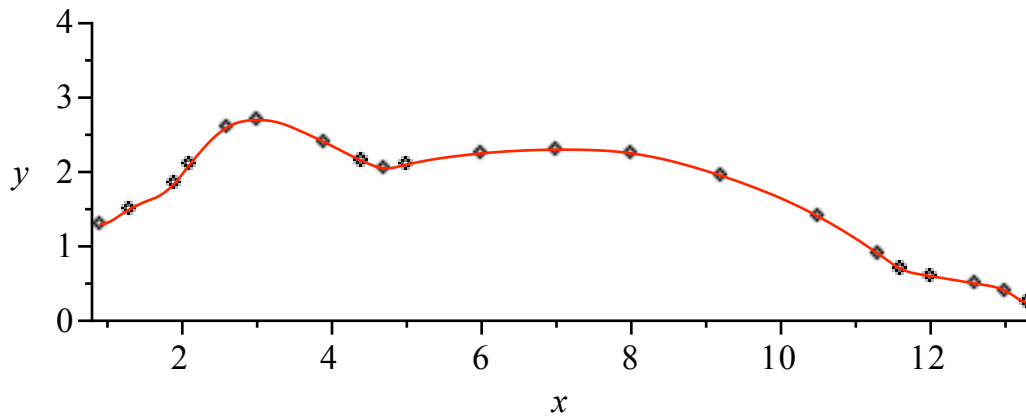
```
> QS:=Spline(X,Y,x,degree=2);
```

$$\begin{aligned}
& 1.30000000000000 + 6.93889390390723 \cdot 10^{-17} x + 1.79158819972477 (x - 0.9)^2 & x < 1.1000000 \\
& 0.763251727713764 + 0.566729440220182 x - 0.374764599174318 (x - 1.3)^2 & x < 1.6000000 \\
& -0.207438810195743 + 1.08286253168197 x + 1.23498641827730 (x - 1.9)^2 & x < 2.1000000 \\
& -0.540577459050656 + 1.25741783764317 x - 0.362209888471304 (x - 2.1)^2 & x < 2.3500000 \\
& 1.06611342359133 + 0.589956375541796 x - 0.972713035731443 (x - 2.6)^2 & x < 2.8000000 \\
& 2.67509609412070 + 0.00830130195976668 x - 0.481424648223630 (x - 3.0)^2 & x < 3.4500000 \\
& 4.31752420239939 - 0.491672872410099 x - 0.0741022121873322 (x - 3.9)^2 & x < 4.1500000 \\
& 4.13386835056243 - 0.450879170582370 x + 0.155689615842791 (x - 4.4)^2 & x < 4.5500000 \\
& 2.39831507813192 - 0.0741095910918983 x + 1.10020898245878 (x - 4.7)^2 & x < 4.8500000 \\
& 0.955649747664536 + 0.228870050467093 x - 0.0902768439288111 (x - 5.0)^2 & x < 5.5000000 \\
QS := & \left\{ \begin{aligned} & 1.68633878126194 + 0.0939435364563437 x - 0.0446496700819380 (x - 6.0)^2 & x < 6.5000000 \\ & 2.24771888443609 + 0.00746873079484493 x - 0.0418251355795608 (x - 7.0)^2 & x < 7.5000000 \\ & 3.36004736980331 - 0.138755921225413 x - 0.104399516440697 (x - 8.0)^2 & x < 8.6000000 \\ & 5.01519525116799 - 0.333173396866086 x - 0.0576150465931966 (x - 9.2)^2 & x < 9.8500000 \\ & 7.10137799595479 - 0.542988380567123 x - 0.103781094715293 (x - 10.5)^2 & x < 10.9000000 \\ & 8.86633170632204 - 0.704985106754163 x - 0.0987148130185069 (x - 11.3)^2 & x < 11.4500000 \\ & 6.41279651967967 - 0.492482458593075 x + 0.807056973555466 (x - 11.6)^2 & x < 11.8000000 \\ & 2.61837843678178 - 0.168198203065148 x + 0.00365366526435099 (x - 12.0)^2 & x < 12.3000000 \\ & 2.59735134293439 - 0.166456455788444 x - 0.000750753136510407 (x - 12.6)^2 & x < 12.8000000 \\ & 6.90039039163099 - 0.500030030125460 x - 0.833183182706031 (x - 13.0)^2 & x < 13.1500000 \\ & 0.250000000000000 + 2.49994994979090 (x - 13.3)^2 & \text{otherwise} \end{aligned} \right.
\end{aligned}$$

```

> QSp:=plot(QS,x=.9..13.3,y=0..4,scaling=constrained):
> display(pp,QSp);

```



* Points 1 — Curve 1

Not quite as good as the cubics. Higher degrees do not necessarily yield better results. Now let's look at problem 3d on p. 99 of the text. We enter the x and y lists.

```
> XX:=[.1,.2,.3,.4];
```

```
XX := [0.1, 0.2, 0.3, 0.4]
```

```
> YY:[-.62049958,-.28398668,.00660095,.24842440];
```

```
YY := [-0.62049958, -0.28398668, 0.00660095, 0.24842440]
```

Since we do not have derivative information at the endpoints, we compute the **natural cubic spline**.

```
> S:=expand(Spline(XX,YY,x));
```

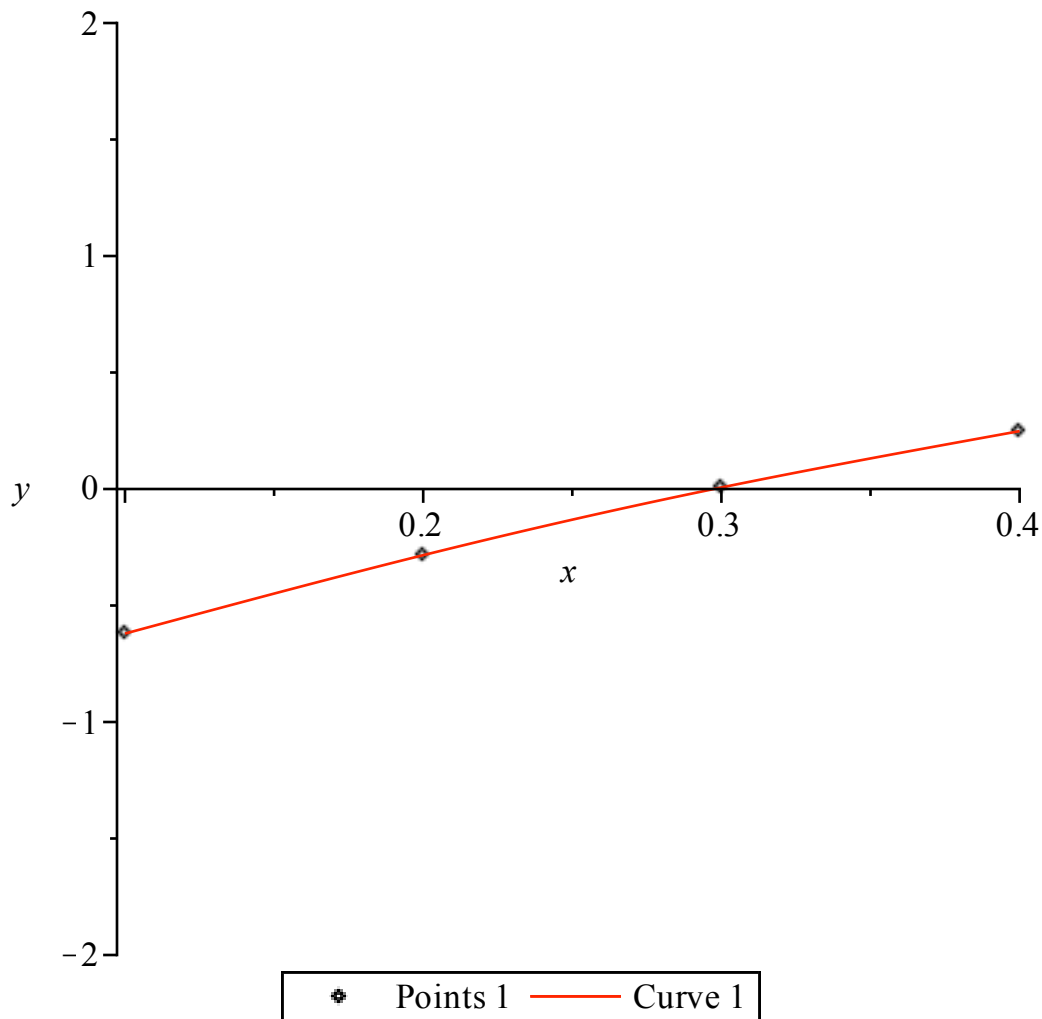
$$S := \begin{cases} -0.957012480000000 + 3.185213133333333 x - 8.995793333333334 x^3 + 2.698738000000000 x^2 & x < 0.1 \\ -1.021408400000000 + 4.151151933333333 x - 2.130956000000001 x^2 - 0.9463033333333326 x^3 & 0.1 < x < 0.2 \\ -1.315395200000000 + 7.091019933333333 x - 11.930516000000000 x^2 + 9.942096666666667 x^3 & \text{otherwise} \end{cases}$$

We plot the points along with the **natural cubic spline**.

```
> pp:=pointplot({seq([XX[i],YY[i]],i=1..4)}):
```

```
Sp:=plot(S,x=0.1..0.4,y=-2..2):
```

```
> display(pp,Sp);
```



Now we wish to find the **clamped cubic spline**. To do this, we make a list of the derivatives at each end point: $[f'(x_0), f'(x_n)]$. We will call this list **clamp**. This is now problem 5d on p. 100.

```
> clamp:=[3.585502082,2.16529366];
      clamp := [3.585502082, 2.16529366]
```

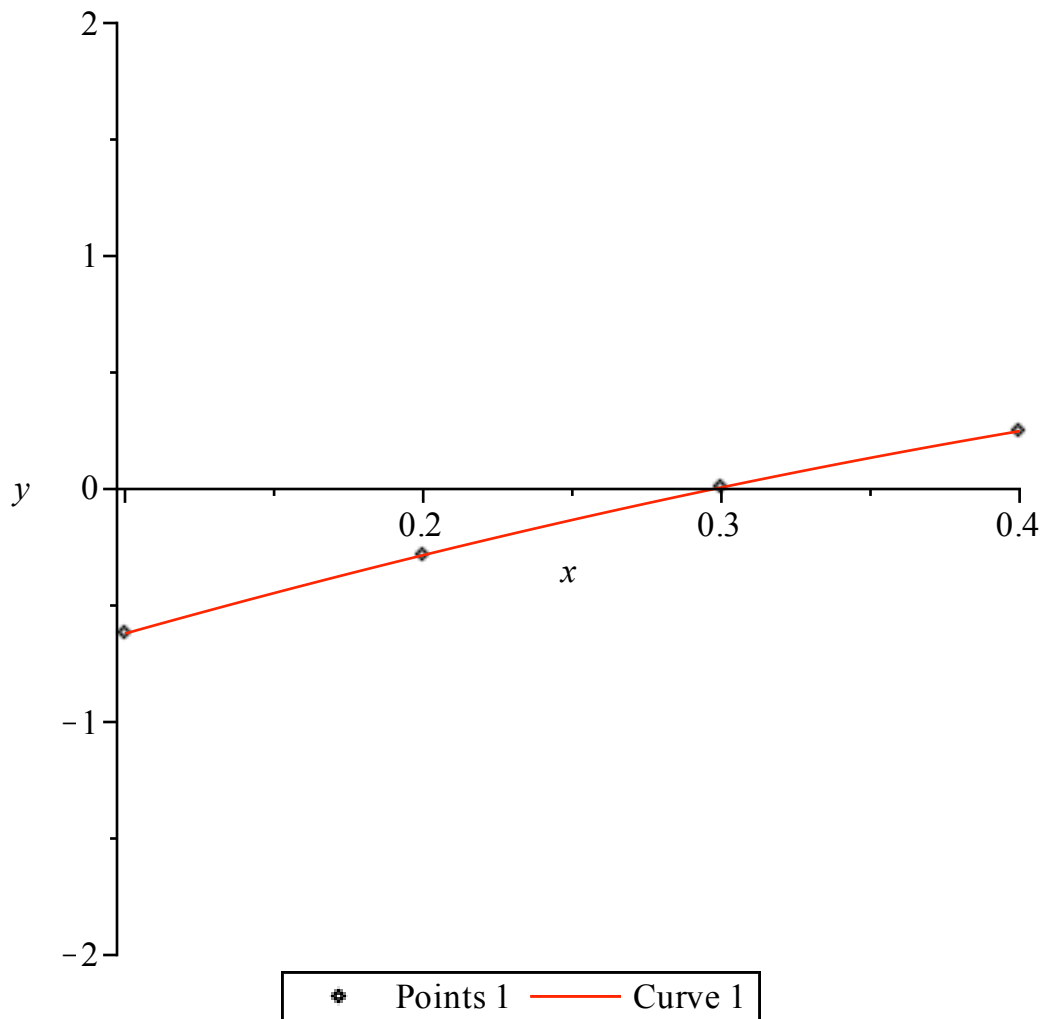
We use this as an extra **endpoints** argument as below.

```
> CS:=expand(Spline(XX,YY,x,endpoints=clamp));
```

$$CS := \begin{cases} -1.00017613322667 + 4.00347416666667x - 2.02153818533333x^2 - 0.455481586666678x^3 & x \\ -0.999946315200000 + 4.00002689626667x - 2.00430183333334x^2 - 0.484208839999998x^3 & x \\ -1.00096194144000 + 4.01018315866667x - 2.03815604133336x^2 - 0.446593053333305x^3 & \text{oti} \end{cases}$$

We plot the points along with the **clamped cubic spline**.

```
> CSp:=plot(CS,x=0.1..0.4,y=-2..2);
> display(pp,CSp);
```



nalib

For small data sets (there are sometimes memory problems with larger sets), we have two library procedures that implement the algorithms in the text. We load the class library.

```
> libname:="c:/nalib",libname;
libname := "/nalib", "c:/nalib", "/Library/Frameworks/Maple.framework/Versions/15/lib",
"/Library/Frameworks/Maple.framework/Versions/15/toolbox/NAG/lib"
> with(numanal);
[SOR, SOR_dir, adaptq, adaptq_dir, bezier, bezier_dir, bisection, bisection_dir, chop, chop_dir,
clamped_spline, clamped_spline_dir, divided_diff, divided_diff_dir, extrap, extrap_dir,
falseposition, falseposition_dir, fixedpoint, fixedpoint_dir, gaussseidel, gaussseidel_dir, hermite,
hermite_dd, hermite_dd_dir, hermite_dir, horner, horner_dir, jacobi, jacobi_dir, muller,
muller_dir, natural_spline, natural_spline_dir, newton, newton_dir, romberg, romberg_dir,
secant, secant_dir, steffensen, steffensen_dir]
```

We look at the directions for **natural_spline**.

```
> natural_spline_dir();
natural_spline returns the piecewise cubic spline.
```

The output also includes a table of values

for the cubic polynomials

$$S[i](x) = a[i] + b[i](x - x[i]) + c[i](x - x[i])^2 + d[i](x - x[i])^3$$

for $i = 0..n-1$ making up the spline.

The arguments for `natural_spline` are:

- (1) the list of x-values
- (2) the list of $f(x)$ or y-values
- (3) the variable for returning the piecewise cubic spline

If assigning the result to a variable, have the variable and the 3rd argument the same.

If S is the variable for returning the spline and has already been given a value, the procedure should be preceded by the statement:

`S := 'S'`

```
> NS := natural_spline(XX, YY, NS);
```

The coefficients of the natural cubic spline $S[i]$ on the subintervals are:

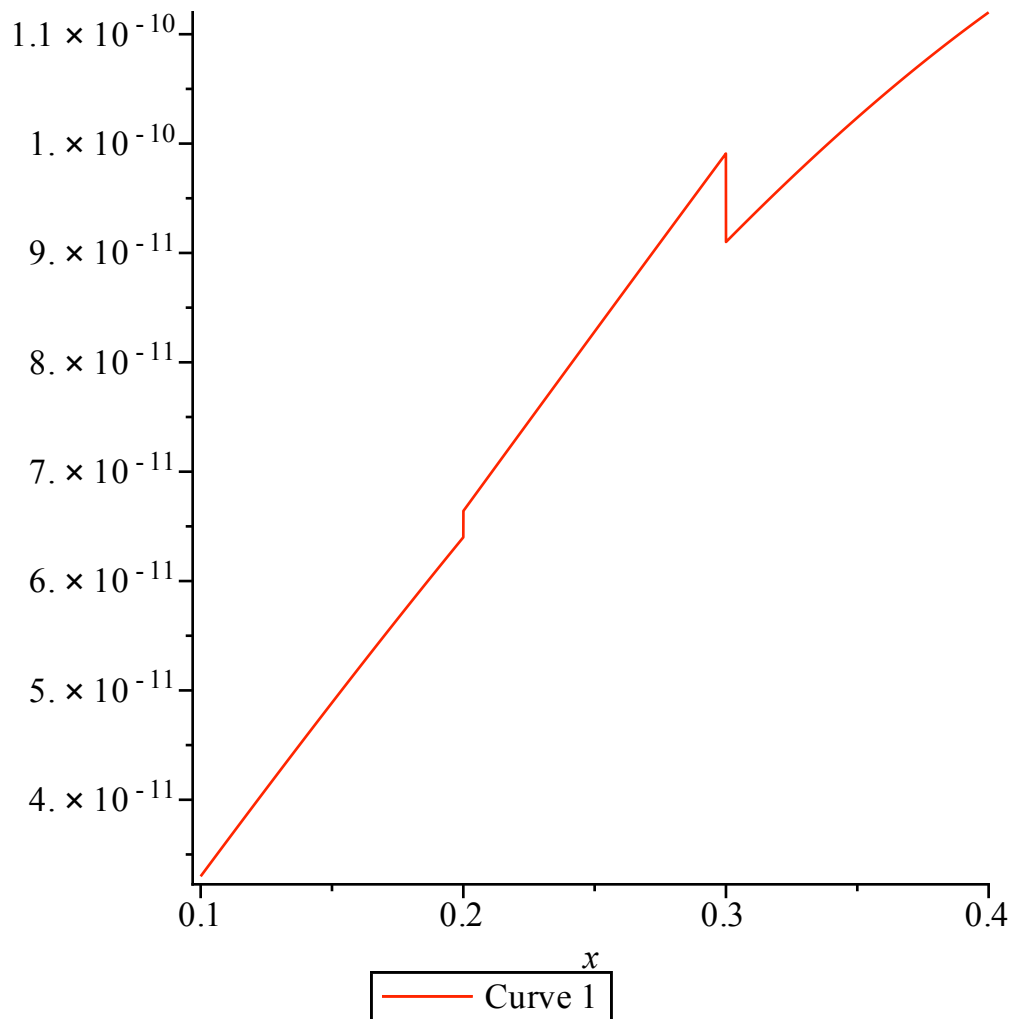
$a(i)$	$b(i)$	$c(i)$	$d(i)$
-0.62049958	3.45508693	0.00000000	-8.99579333
-0.28398668	3.18521313	-2.69873800	-0.94630333
0.00660095	2.61707643	-2.98262900	9.94209667

$NS :=$

$$\left\{ \begin{array}{ll} -0.9570124800 + 3.185213133x - 8.995793333x^3 + 2.698738000x^2 & x < 0.2000000000 \\ -1.021408400 + 4.151151933x - 2.130956000x^2 - 0.9463033333x^3 & x < 0.3000000000 \\ -1.315395200 + 7.091019933x - 11.93051600x^2 + 9.942096667x^3 & 0.3000000000 \leq x \end{array} \right.$$

We plot the difference between the Maple and class library procedure.

```
> plot(S - NS, x = 0.1..0.4);
```



Any difference is clearly round-off error. We now look at the directions for **clamped_spline**.

```
> clamped_spline_dir();
clamped_spline returns the piecewise cubic spline.
```

The output also includes a table of values for the cubic polynomials

$$S[i](x) = a[i] + b[i](x - x[i]) + c[i](x - x[i])^2 + d[i](x - x[i])^3$$

for $i = 0..n-1$ making up the spline.

The arguments for **clamped_spline** are:

- (1) the list of x-values
- (2) the list of f(x) or y-values
- (3) $f'(x[0])$
- (4) $f'(x[n])$
- (5) the variable for returning the piecewise cubic spline

If assigning the result to a variable, have the variable and the 5th argument the same.

If S is the variable for returning the spline and has already been given a value, the procedure should be preceded by the statement:
`S:='S'`

```
> CLS:=clamped_spline(XX,YY,clamp[1],clamp[2],CLS);
```

The coefficients of the clamped cubic spline $S[i]$ on the subintervals are:

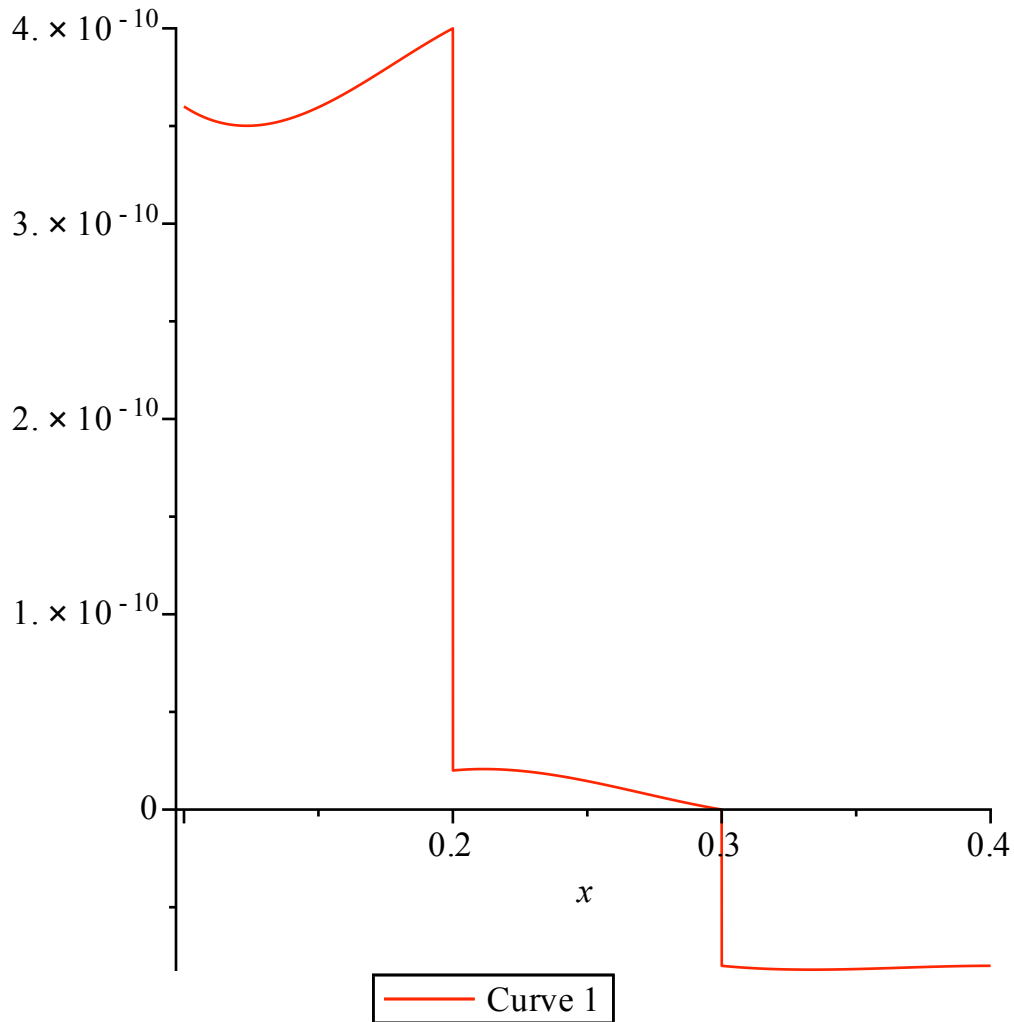
$a(i)$	$b(i)$	$c(i)$	$d(i)$
-0.62049958	3.58550208	-2.15818268	-0.45548149
-0.28398668	3.14020110	-2.29482713	-0.48420887
0.00660095	2.66670941	-2.44008979	-0.44659304

$CLS :=$

$$\begin{cases} -1.000176134 + 4.003474175x - 2.021538237x^2 - 0.4554814900x^3 & x < 0.2000000000 \\ -0.9999463147 + 4.000026890x - 2.004301809x^2 - 0.4842088700x^3 & x < 0.3000000000 \\ -1.000961942 + 4.010183164x - 2.038156056x^2 - 0.4465930400x^3 & 0.3000000000 \leq x \end{cases}$$

We again plot the difference between the Maple and class library procedure.

```
> plot(CS-CLS,x=.1..(.4));
```



Again, any difference is clearly due to round-off error.

NumericalAnalysis

We now turn to the [CubicSpline](#) command in the [NumericalAnalysis](#) package. We go back to the data for the duck's back.

```
> with(Student[NumericalAnalysis]):  
> duck1:=CubicSpline(points,independentvar=x);  
duck1 := POLYINTERP([[0.9, 1.3], [1.3, 1.5], [1.9, 1.85], [2.1, 2.1], [2.6, 2.6], [3.0, 2.7], [3.9,  
2.4], [4.4, 2.15], [4.7, 2.05], [5.0, 2.1], [6.0, 2.25], [7.0, 2.3], [8.0, 2.25], [9.2, 1.95], [10.5,  
1.4], [11.3, 0.9], [11.6, 0.7], [12.0, 0.6], [12.6, 0.5], [13.0, 0.4], [13.3, 0.25]], independentvar  
= x, INFO)
```

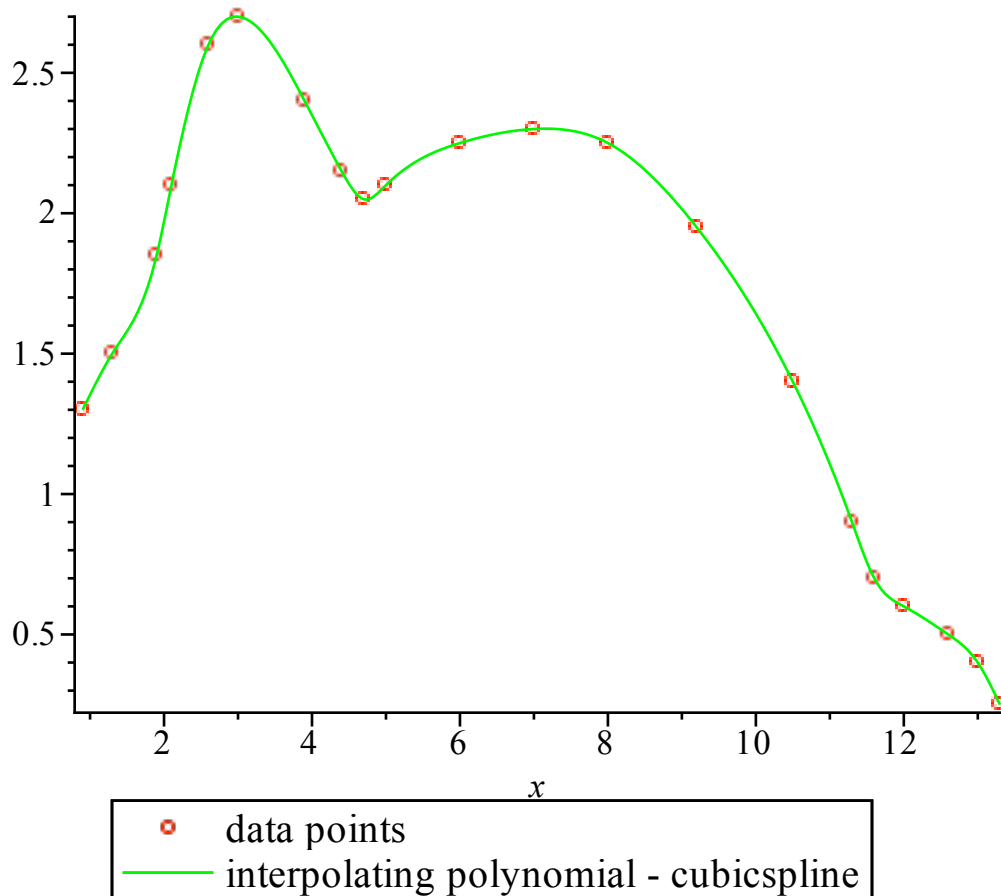
Again, we are given a **POLYINTERP** structure. We list the interpolating polynomial.

```
> duck:=expand(Interpolant(duck1));  
duck := {  
0.994874698843094 - 0.0621633613227766 x - 0.247649057851443 x3 + 0.668652456198897 x2  
- 1.62957614969247 + 5.99426167375929 x - 3.99013603232577 x2 + 0.946912093052318 x3  
25.1431211712514 - 36.2784183066785 x + 18.2586429047467 x2 - 2.95638245731128 x3  
1.90034792694569 - 3.07445652909881 x + 2.44723253447071 x2 - 0.446634779489688 x3  
- 13.7719231043175 + 15.0089331223587 x - 4.50791733147449 x2 + 0.445051100759697 x3  
- 6.45785987769265 + 7.69486989573386 x - 2.06989625593288 x2 + 0.174159870143962 x3  
- 0.758240259746141 + 3.31054711269808 x - 0.945710926949345 x2 + 0.0780756539915228 x3  
- 106.053810419206 + 75.1029813123295 x - 17.2621732450474 x2 + 1.31417128415047 x3  
194.554285028573 - 116.774526420295 x + 23.5628284001919 x2 - 1.58121890345515 x3  
- 8.48749404688038 + 5.05054102497693 x - 0.802185088862525 x2 + 0.0431153291484708 x3  
1.83334702301793 - 0.109879509972225 x + 0.0578850002956675 x2 - 0.00466634247142876 x3  
8.61912758244311 - 3.01807117829730 x + 0.473340952913535 x2 - 0.0244499592627558 x3  
- 12.8442447693226 + 5.03069345361485 x - 0.532754626075483 x2 + 0.0174706898617866 x3  
10.6710412008438 - 2.63733458013506 x + 0.300726681940811 x2 - 0.0127279082547458 x3  
19.4658830045450 - 5.15014652404970 x + 0.540042105170777 x2 - 0.0203252232779193 x3  
- 1760.67976749717 + 467.454893432157 x - 41.2834127582104 x2 + 1.21340500868023 x3  
1443.33988886395 - 361.170879764683 x + 30.1498435518621 x2 - 0.839274770344844 x3  
- 69.7951573181189 + 17.1128817808337 x - 1.37380324359766 x2 + 0.0363820850845934 x3  
899.093070885677 - 213.574791601023 x + 16.9347422628989 x2 - 0.447970970642828 x3  
- 1393.84129161730 + 315.563907438125 x - 23.7682345862663 x2 + 0.595695102412690 x3}
```

We plot the graph.

```
> Draw(duck1);
```

Cubic spline interpolation
natural boundary conditions



This doesn't look like a duck since we cannot get the **scaling=constrained** option here. We evaluate the spline at $x = 6.4$ and $x = 9.73$.

```
> ApproximateValue(duck1, [6.4, 9.73]);
[[6.4, 2.27783409047601], [9.73, 1.75590972479754]]
```

We now go to the second set of data.

```
> points1 := [seq([XX[i], YY[i]], i=1..4)];
points1 := [[0.1, -0.62049958], [0.2, -0.28398668], [0.3, 0.00660095], [0.4, 0.24842440]]
```

We apply **CubicSpline** to find the **natural or free cubic spline**.

```
> NS1 := CubicSpline(points1, independentvar=x);
NS1 := POLYINTERP([[0.1, -0.62049958], [0.2, -0.28398668], [0.3, 0.00660095], [0.4,
0.24842440]], independentvar = x, INFO)
```

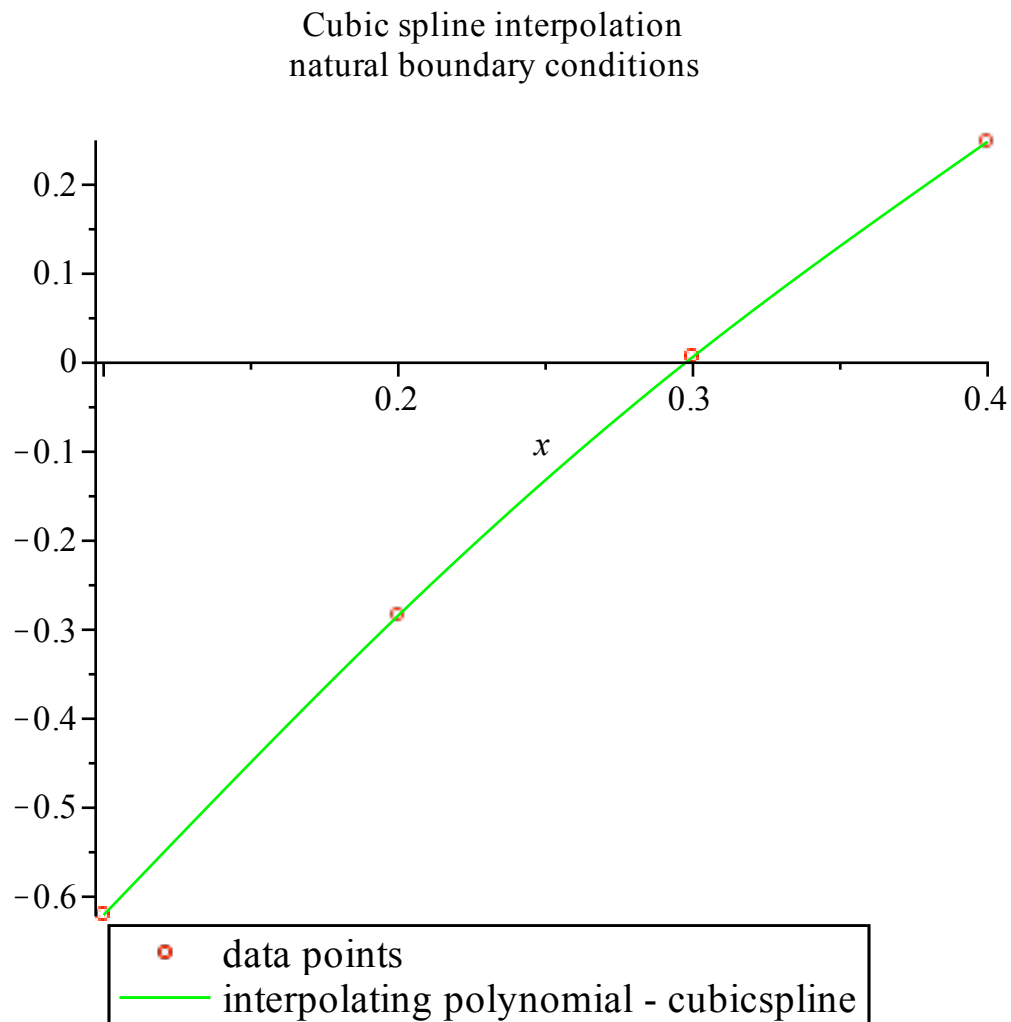
We find the interpolating polynomial.

```
> NS := expand(Interpolant(NS1));
```

$$NS := \begin{cases} -0.957012480000000 + 3.18521313333333 x - 8.99579333333334 x^3 + 2.69873800000000 x^2 & x < 0 \\ -1.02140840000000 + 4.15115193333333 x - 2.13095600000001 x^2 - 0.946303333333326 x^3 & x < 0 \\ -1.31539520000000 + 7.09101993333333 x - 11.9305160000000 x^2 + 9.94209666666667 x^3 & \text{other} \end{cases}$$

We graph the points and the cubic spline.

```
> Draw(NS1);
```



We find the value at 0.25.

```
> ApproximateValue(NS1, .25);  
[[0.25, -0.131591156250000]]
```

We now apply **CubicSpline** to find the **clamped cubic spline** with $\frac{d}{dx} f(0) = 3.585502082$ and

$\frac{d}{dx} f(0.4) = 2.16529366$. We need to set **boundary conditions=clamped** here and list the derivatives at the endpoints. The default is **boundaryconditions=natural**.

```
> NS2:=CubicSpline(points1,independentvar=x,boundaryconditions=  
clamped(clamp[1],clamp[2]));  
NS2 := POLYINTERP([[0.1, -0.62049958], [0.2, -0.28398668], [0.3, 0.00660095], [0.4,  
0.24842440]], independentvar = x, boundaryconditions = clamped(3.585502082, 2.16529366),  
INFO)
```

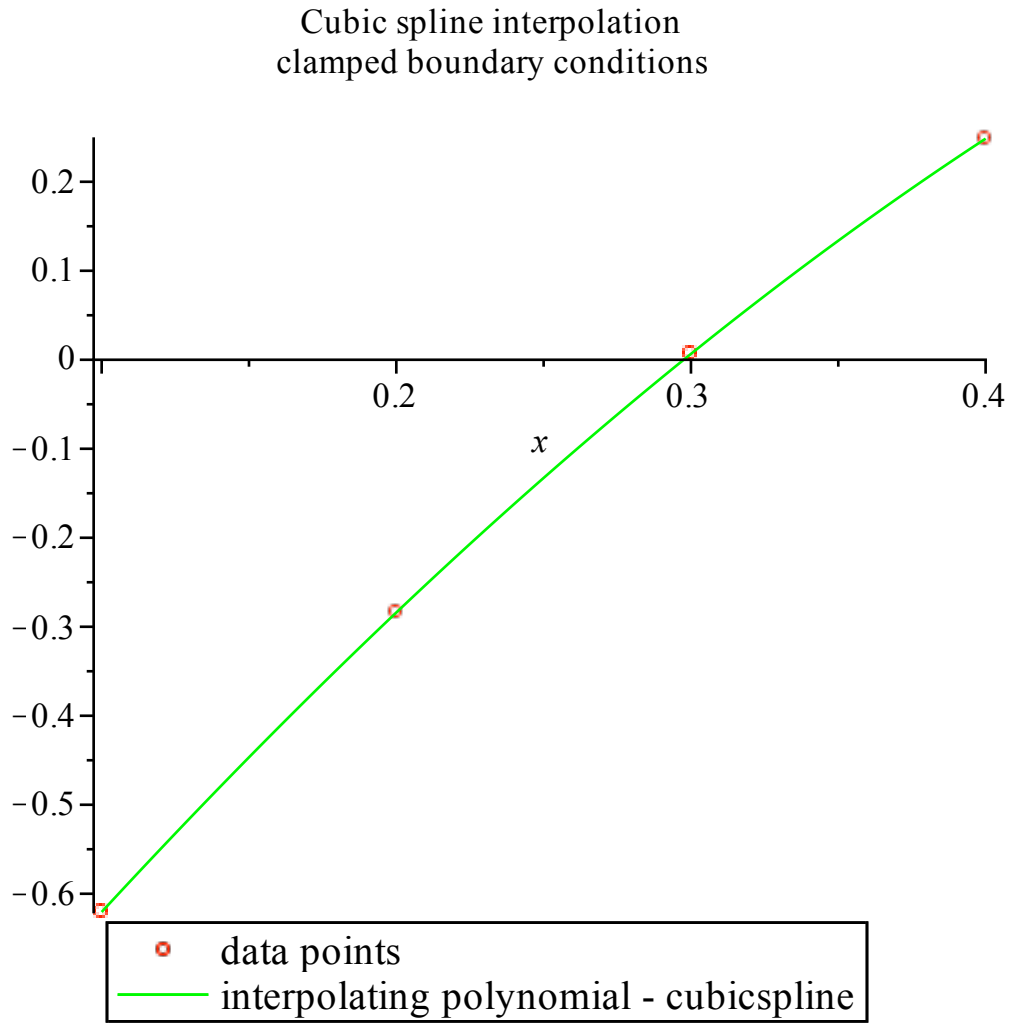
We find the interpolating polynomial.

```
> NS:=expand(Interpolant(NS2));
```

$$NS := \begin{cases} -1.00017613322667 + 4.00347416666667 x - 2.02153818533333 x^2 - 0.455481586666678 x^3 & x \\ -0.999946315200000 + 4.00002689626667 x - 2.00430183333334 x^2 - 0.484208839999998 x^3 & x \\ -1.00096194144000 + 4.01018315866667 x - 2.03815604133336 x^2 - 0.446593053333305 x^3 & \text{other } x \end{cases}$$

We graph the points and the cubic spline.

```
> Draw(NS2);
```



We find the value at 0.25.

```
> ApproximateValue(NS2, .25);
[[0.25, -0.132774218841667]]
```