

# Lagrange Interpolation

## Finding the Lagrange Polynomial by Maple

```
> restart;
```

Given a set of points  $(x_0, f(x_0)), (x_1, f(x_1)), \dots, (x_n, f(x_n))$  on the graph of a function  $f$ , there is a unique polynomial of degree  $n$  or less that passes through each of the  $n + 1$  points. For an example, suppose we are given the three points  $(0, f(0)), (.3, f(.3)),$  and  $(.6, f(.6))$  on the graph of the function  $f(x) = e^{(2x)} \cos(3x)$ . Then there is a polynomial of degree two or less that goes through these three points. We first enter the function.

```
> f:=x->exp(2*x)*cos(3*x);
```

$$f := x \rightarrow e^{2x} \cos(3x)$$

Maple's command for finding this interpolating polynomial is [PolynomialInterpolation](#) from the [CurveFitting](#) package. The three arguments of **PolynomialInterpolation** are an ordered [list](#) of the  $x$ -values, a corresponding ordered list of the  $y$ -values, and the variable used in the polynomial.

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

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

```
> p:=PolynomialInterpolation([0,.3,.6],[f(0),f(.3),f(.6)],x);
```

$$p := -11.22017745 x^2 + 3.808210602 x + 1$$

We could also do this by using a list variable for the data pairs.

```
> XY:=[[0,f(0)],[.3,f(.3)],[.6,f(.6)]];
```

$$XY := [[0, 1], [0.3, 1.132647210], [0.6, -0.7543375196]]$$

```
> p:=PolynomialInterpolation(XY,x);
```

$$p := -11.22017745 x^2 + 3.808210602 x + 1$$

We can also use separate list variables for the  $X$  and  $Y$  lists.

```
> X:=[0,.3,.6];
```

$$X := [0, 0.3, 0.6]$$

```
> Y:=[f(0),f(.3),f(.6)];
```

$$Y := [1, 1.132647210, -0.7543375196]$$

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

$$p := -11.22017745 x^2 + 3.808210602 x + 1$$

## The Interpolating Error

The error  $f(x) - p(x)$  in approximating  $f(x)$  by  $p(x)$  over the interval  $[x_0, x_n]$  is given by the formula

$$\frac{1}{(n+1)!} f^{(n+1)}(\xi(x)) (x-x_0)(x-x_1)\dots(x-x_n)$$

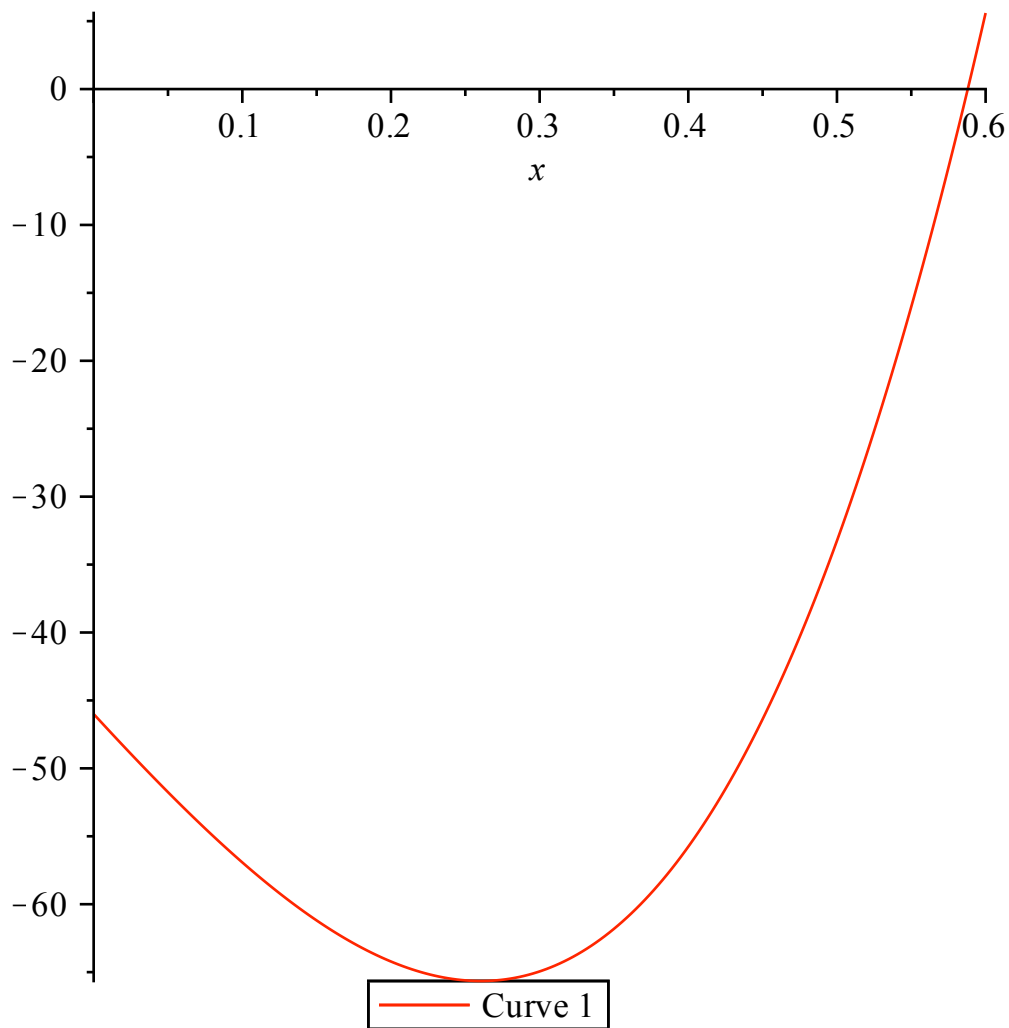
provided the function has enough derivatives. Since  $n = 2$  in our current example, we first compute the third derivative of  $f(x)$ .

```
> f3p:=(D@@3)(f);
```

$$f3p := x \rightarrow -46 e^{2x} \cos(3x) - 9 e^{2x} \sin(3x)$$

We look at the graph of this derivative over the interval [0, .6].

```
> plot(f3p(x), x=0..(.6));
```



We see that the values of the third derivative over this interval are between -68 and 8. Since we are interested in an absolute error bound, we look for the minimum value here by using the [minimize](#) command.

```
> minimize(f3p(x), x=0..0.6);
```

-65.65219517

We set an upper bound for the maximum absolute value of the 3rd derivative as 66.

```
> max3deriv:=66;
```

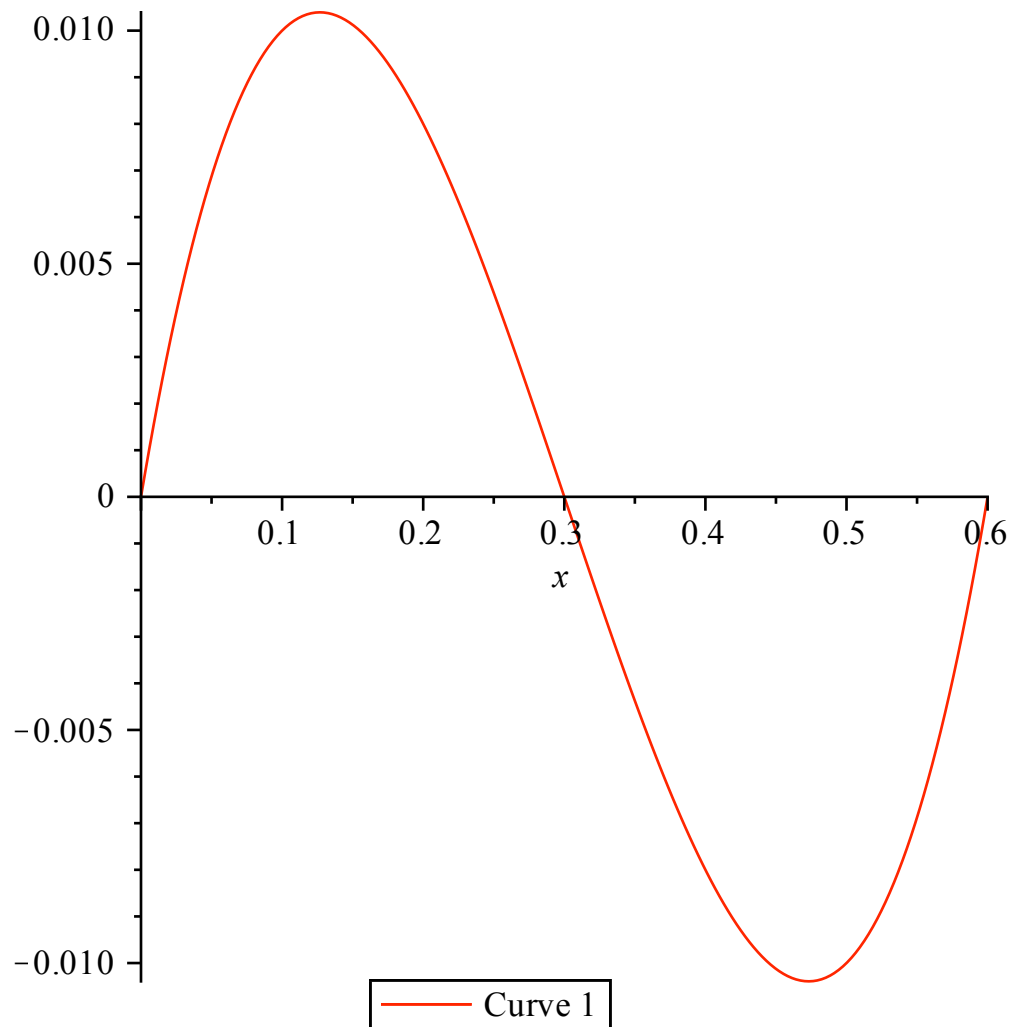
max3deriv := 66

Next, we look for the largest absolute value for  $(x - 0)(x - 0.3)(x - 0.6)$  over our interval.

```
> g:=x*(x-.3)*(x-.6);
```

$g := x(x - 0.3)(x - 0.6)$

```
> plot(g, x=0..(.6));
```



These values are between  $-0.011$  and  $0.011$ . Thus the expression is absolutely bounded by  $0.011$ . Note that we could again get a finer upper bound by using **minimize**, but what we have should be fine enough.

```
> maxinterval:=.011;
```

```
maxinterval := 0.011
```

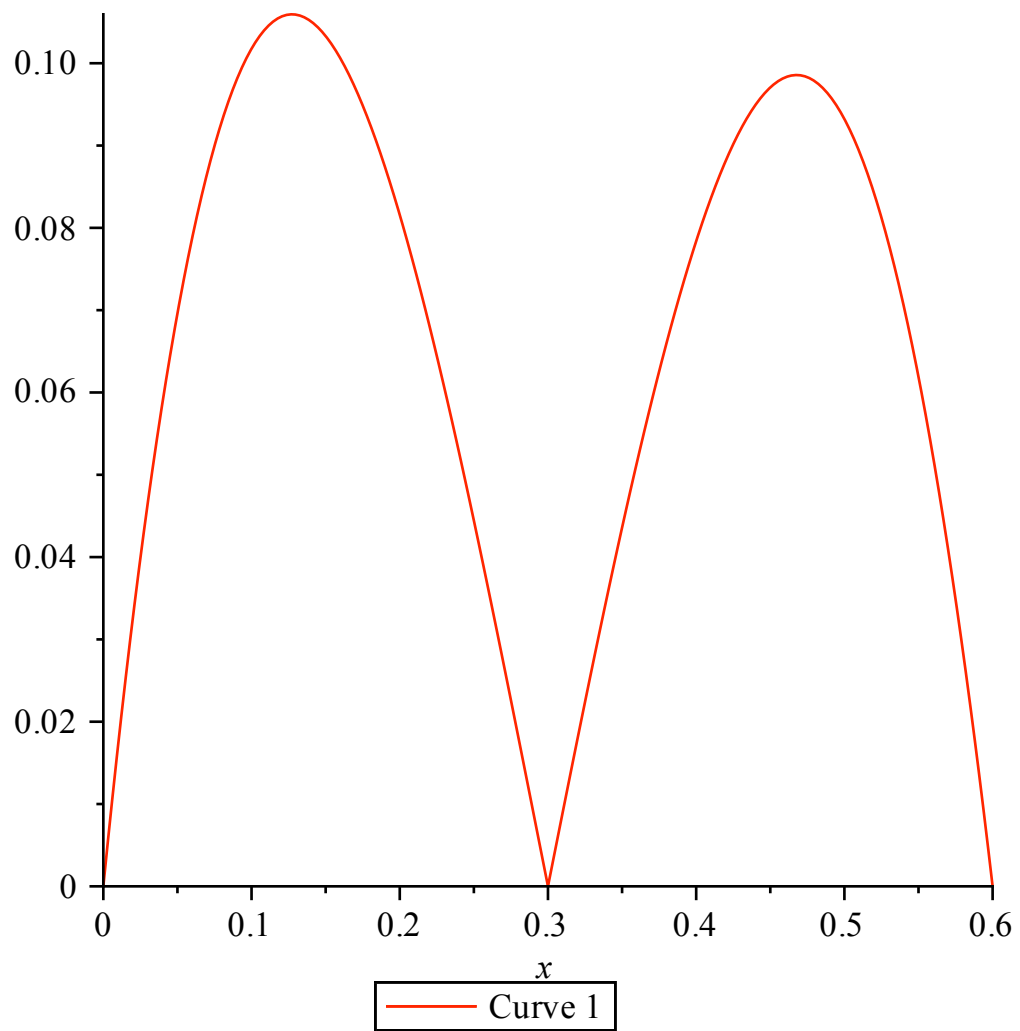
We can now find an upper bound for the **maximum absolute error for any point in the interval**.

```
> error_bound:=max3deriv/3!*maxinterval;
```

```
error_bound := 0.121
```

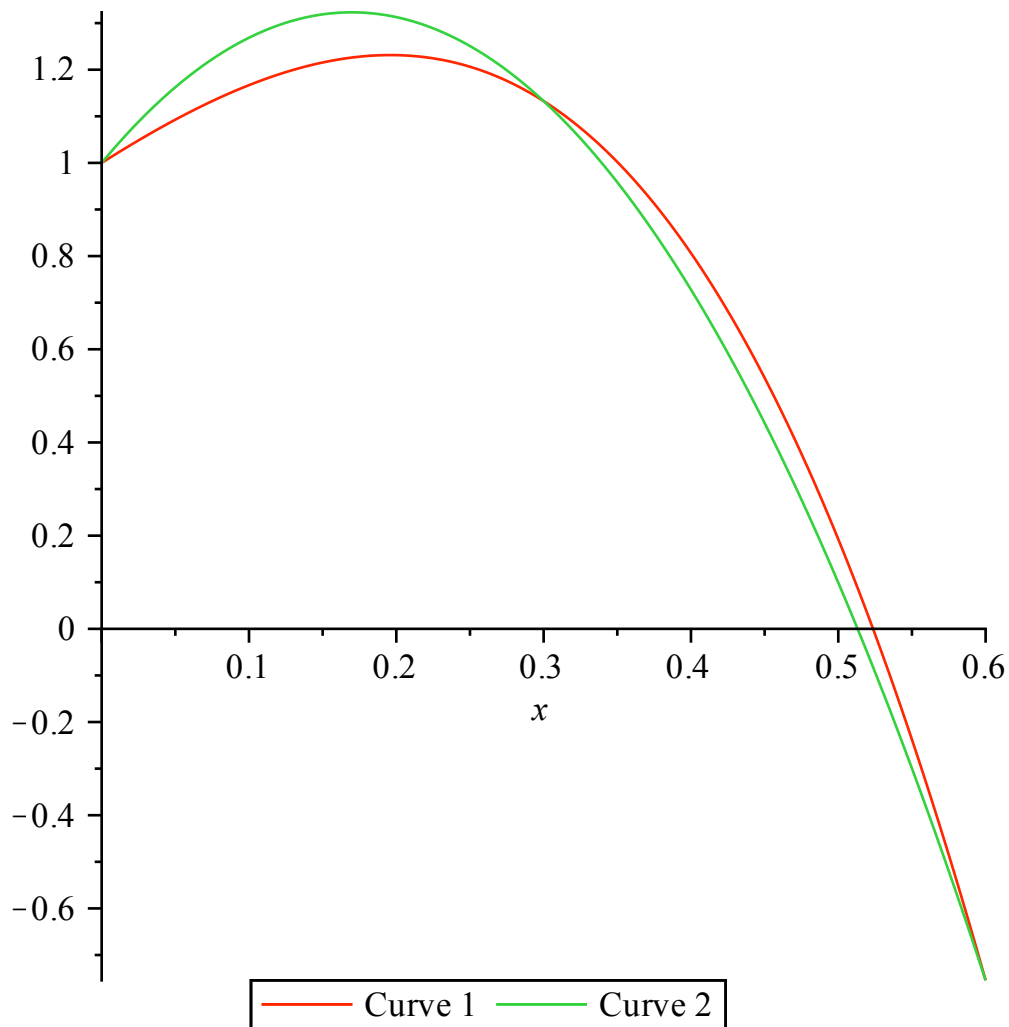
Let's look at the graph of the absolute error.

```
> plot(abs(f(x)-p),x=0..(.6));
```



It is certainly less than .12. Finally, let's look at the graphs of both  $f$  and  $p$ .

```
> plot({f(x),p},x=0..(.6));
```



Suppose we now want to find an **upper bound for the absolute error at the point 0.4**. To do this, we simply need to replace **maxinterval** by  $g(0.4)$ .

```
> gg:=eval(g,x=.4);
                                gg := -0.008
> point_error_bound:=abs(max3deriv/3!*gg);
                                point_error_bound := 0.088
```

### *NumericalAnalysis*

The command [PolynomialInterpolation](#) is also in the [NumericalAnalysis](#) subpackage. This command now supersedes the same command in the **CurveFitting** package.

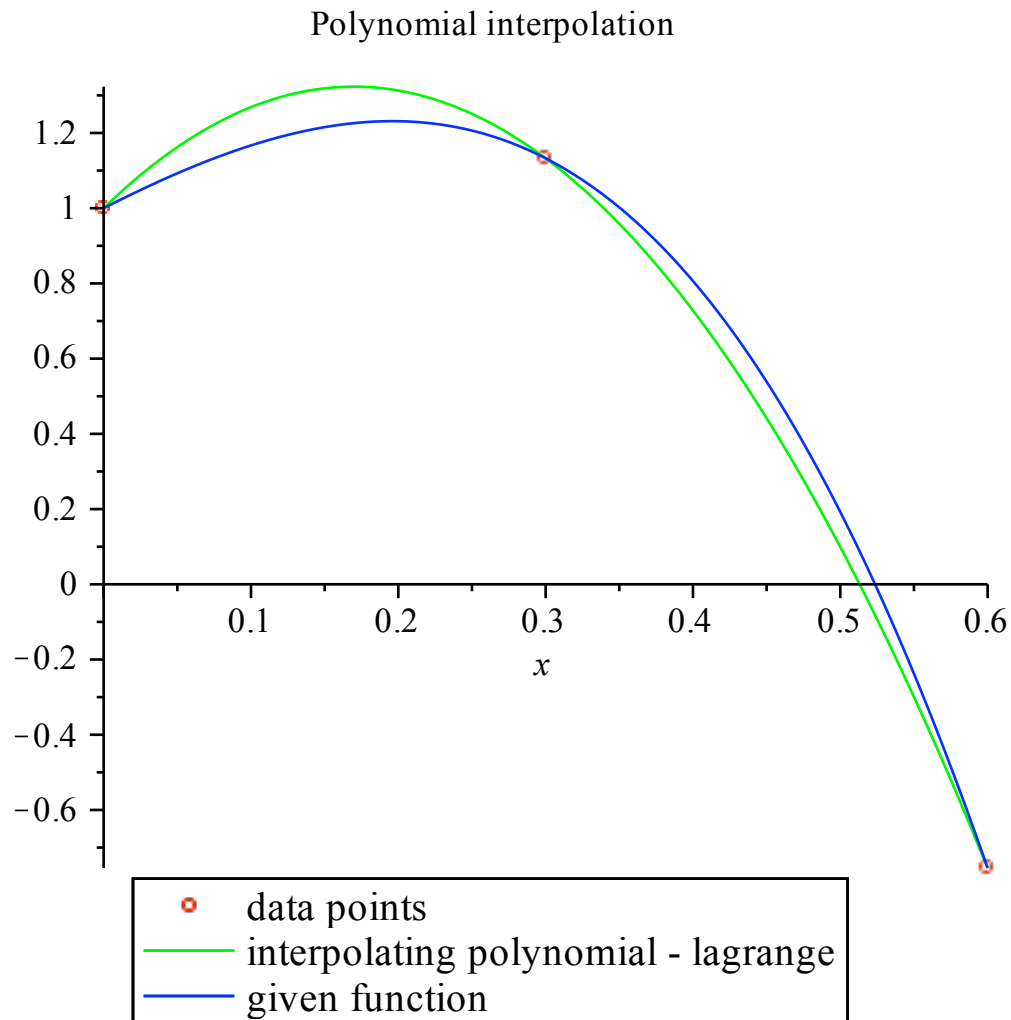
```
> with(Student[NumericalAnalysis]):
We need to provide the data as a list of XY data pairs.
> XY:=[[0,f(0)],[.3,f(.3)],[.6,f(.6)]];
                                XY := [[0, 1], [0.3, 1.132647210], [0.6, -0.7543375196]]
> p1:=PolynomialInterpolation(XY,method=lagrange,function=f(x),
                                independentvar=x,errorboundvar=xi);
p1 := POLYINTERP([[0, 1], [0.3, 1.132647210], [0.6, -0.7543375196]], method = lagrange,
function = e2x cos(3x), independentvar = x, errorboundvar = xi, INFO)
```

The procedure **POLYINTERP** is where all the computed information is stored.

```
> p:=Interpolant(p1);  
p := 5.555555556 (x - 0.3) (x - 0.6) - 12.58496900 x (x - 0.6) - 4.190763998 x (x - 0.3)  
> p:=expand(p);  
p := -11.22017744 x2 + 3.808210599 x + 1.000000000
```

The small differences from what we got above are due to different rounding schemes. We use the [Draw](#) command to plot the function and polynomial interpolant.

```
> Draw(p1);
```



Next we find the Remainder Term using the [RemainderTerm](#) command.

```
> r:=RemainderTerm(p1);  
r :=  $\left( \frac{1}{6} (-46 e^{2\xi} \cos(3\xi) - 9 e^{2\xi} \sin(3\xi)) x (x - 0.3) (x - 0.6) \right)$  &where {0. ≤ ξ and ξ  
≤ 0.6}
```

The [UpperBoundOfRemainderTerm](#) command gives us an absolute upper bound for the error at 0.4, which could be replaced by a list if more points were desired.

```
> ub1:=UpperBoundOfRemainderTerm(p1,0.4);  
ub1 := [[0.4, 0.08753626023]]
```

For more information, we use the [ApproximateExactUpperBound](#) command, which for each point yields

a list containing the input value, the value from the approximating polynomial, the exact value from the function, and an upperbound for the error.

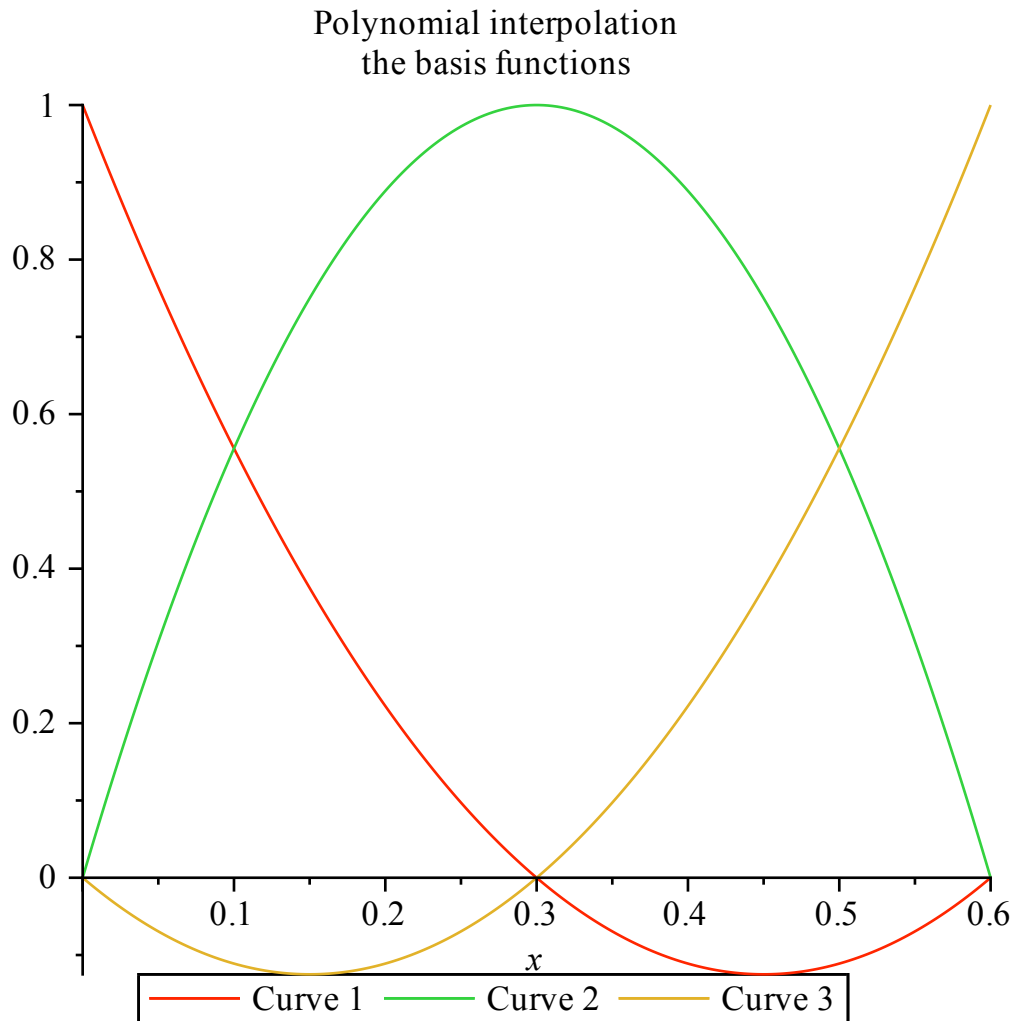
```
> ub2:=ApproximateExactUpperBound(p1, [.4, .6]);  
ub2 := [[0.4, 0.7280558490, 0.8064420132, 0.08753626023], [0.6, -0.7543375196, -0.7543375196,  
0.]]
```

We can also extract the 3 basis functions here.

```
> b:=BasisFunctions(p1);  
b := [5.555555556 (x - 0.3) (x - 0.6), -11.11111111 x (x - 0.6), 5.555555556 x (x - 0.3)]
```

We draw the basis functions.

```
> Draw(p1, objects=[BasisFunctions]);
```



### Finding a Lagrange Polynomial from Data Points Step-by-Step

Please do not use this part of the worksheet for home work.

```
> restart;
```

We do Problem 29a on Page 117, just using Sample 1. Choose n to be one less than the number of data points since the Lagrange quotients are indexed from 0 through n.

```
> n:=6;
```

$n := 6$

The list of x-values follows. Maple indexes them as X[1], X[2], ..., X[n], X[n+1].

```
> X:=[0,6,10,13,17,20,28];  
X:= [0, 6, 10, 13, 17, 20, 28]
```

The list of y-values follows. Maple indexes them as Y[1], Y[2], ..., Y[n], Y[n+1].

```
> Y:=[6.67,17.33,42.67,37.33,30.10,29.31,28.74];  
Y:= [6.67, 17.33, 42.67, 37.33, 30.10, 29.31, 28.74]
```

Set up the array L[n,k] where k = 0, 1, ..., n.

```
> L:=array(n..n,0..n);  
L := array(6..6, 0..6, [ ])
```

We initialize the array, setting each element equal to 1 for multiplication by the composing factors.

```
> for k from 0 to n do L[n,k]:=1:od:
```

For each k, we multiply L[n,k] by the n appropriate factors  $\frac{x - x_i}{x_k - x_i}$ . Because of the list indexing in

Maple, however, this is implemented in the loop structure by  $\frac{x - X_{i+1}}{X_{k+1} - X_{i+1}}$ . Each L[n,k] is also made

into a function.

```
> for k from 0 to n do  
  for i from 0 to n do  
    if i<>k then L[n,k]:=L[n,k]*(x-X[i+1])/(X[k+1]-X[i+1]) fi;  
  od;  
  L[n,k]:=unapply(L[n,k],x);  
od;
```

$$L_{6,0} := x \rightarrow -\frac{1}{1237600} \left( -\frac{1}{6}x + 1 \right) (x - 10) (x - 13) (x - 17) (x - 20) (x - 28)$$

$$L_{6,1} := x \rightarrow -\frac{1}{569184} x (x - 10) (x - 13) (x - 17) (x - 20) (x - 28)$$

$$L_{6,2} := x \rightarrow \frac{1}{151200} x (x - 6) (x - 13) (x - 17) (x - 20) (x - 28)$$

$$L_{6,3} := x \rightarrow -\frac{1}{114660} x (x - 6) (x - 10) (x - 17) (x - 20) (x - 28)$$

$$L_{6,4} := x \rightarrow \frac{1}{172788} x (x - 6) (x - 10) (x - 13) (x - 20) (x - 28)$$

$$L_{6,5} := x \rightarrow -\frac{1}{470400} x (x - 6) (x - 10) (x - 13) (x - 17) (x - 28)$$

$$L_{6,6} := x \rightarrow \frac{1}{14636160} x (x - 6) (x - 10) (x - 13) (x - 17) (x - 20)$$

We now initialize the Lagrange polynomial for the addition of terms.

```
> P:=0:
```

We build the Lagrange polynomial term by term.

```
> for k from 0 to n do  
  P:=P+Y[k+1]*L[n,k](x):  
od:
```

We multiply out the polynomial to put it in its usual form.

```
> P:=collect(P,x);
```

$$P := 6.670000000 + 0.00004094575680 x^6 - 0.003671679405 x^5 + 0.1269023635 x^4 - 2.094639082 x^3 + 16.14272437 x^2 - 42.6434810 x$$

We make a function of the polynomial.

```
> P:=unapply(P,x);
```

$$P := x \rightarrow 6.670000000 + 0.00004094575680 x^6 - 0.003671679405 x^5 + 0.1269023635 x^4 - 2.094639082 x^3 + 16.14272437 x^2 - 42.6434810 x$$

Of course, using Maple, we could just do the following after reloading the **CurveFitting** package.

```
> with(CurveFitting):
```

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

$$Q := 0.00004094575671 x^6 - 0.003671679393 x^5 + 0.1269023632 x^4 - 2.094639075 x^3 + 16.14272434 x^2 - 42.64348083 x + 6.67$$

The small differences in the two methods would be due to rounding error.