

## Least Squares Regression

```
> restart;
```

We load the packages we need.

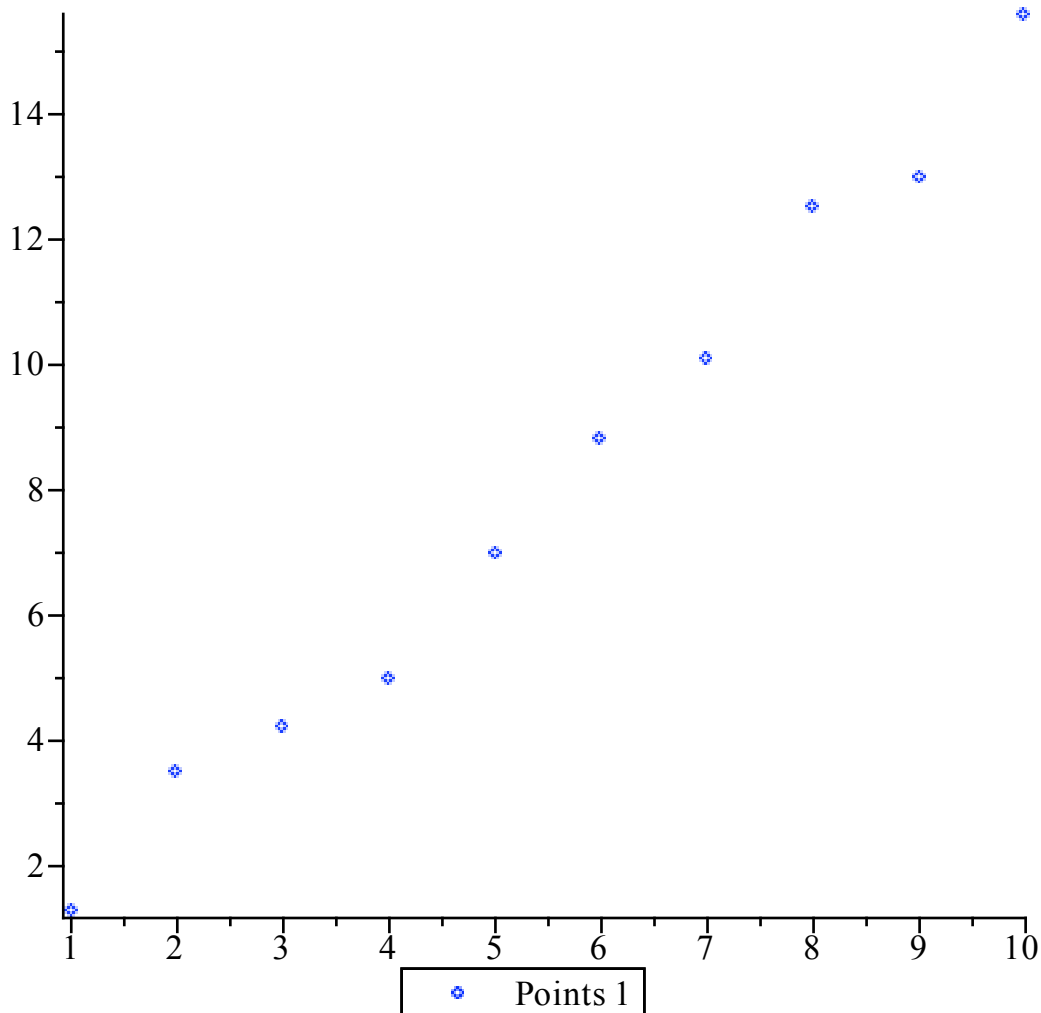
```
> with(plots):with(Student[LinearAlgebra]):with(Student[NumericalAnalysis]):
```

We look at two different sets of data and their scatterplots. The first set of data:

```
> X1:=[[1,1.3],[2,3.5],[3,4.2],[4,5.0],[5,7.0],[6,8.8],[7,10.1],[8,12.5],[9,13.0],[10,15.6]];
X1:= [[1, 1.3], [2, 3.5], [3, 4.2], [4, 5.0], [5, 7.0], [6, 8.8], [7, 10.1], [8, 12.5], [9, 13.0], [10, 15.6]]
```

We use the [pointplot](#) command to plot the data.

```
> g1:=pointplot(X1,color=blue):
> display(g1);
```



The second set of data and its scatterplot:

```
> X2:=[[.2,.050446],[.3,-.098426],[.6,.33277],[.9,.72660],[1.1,
```

```

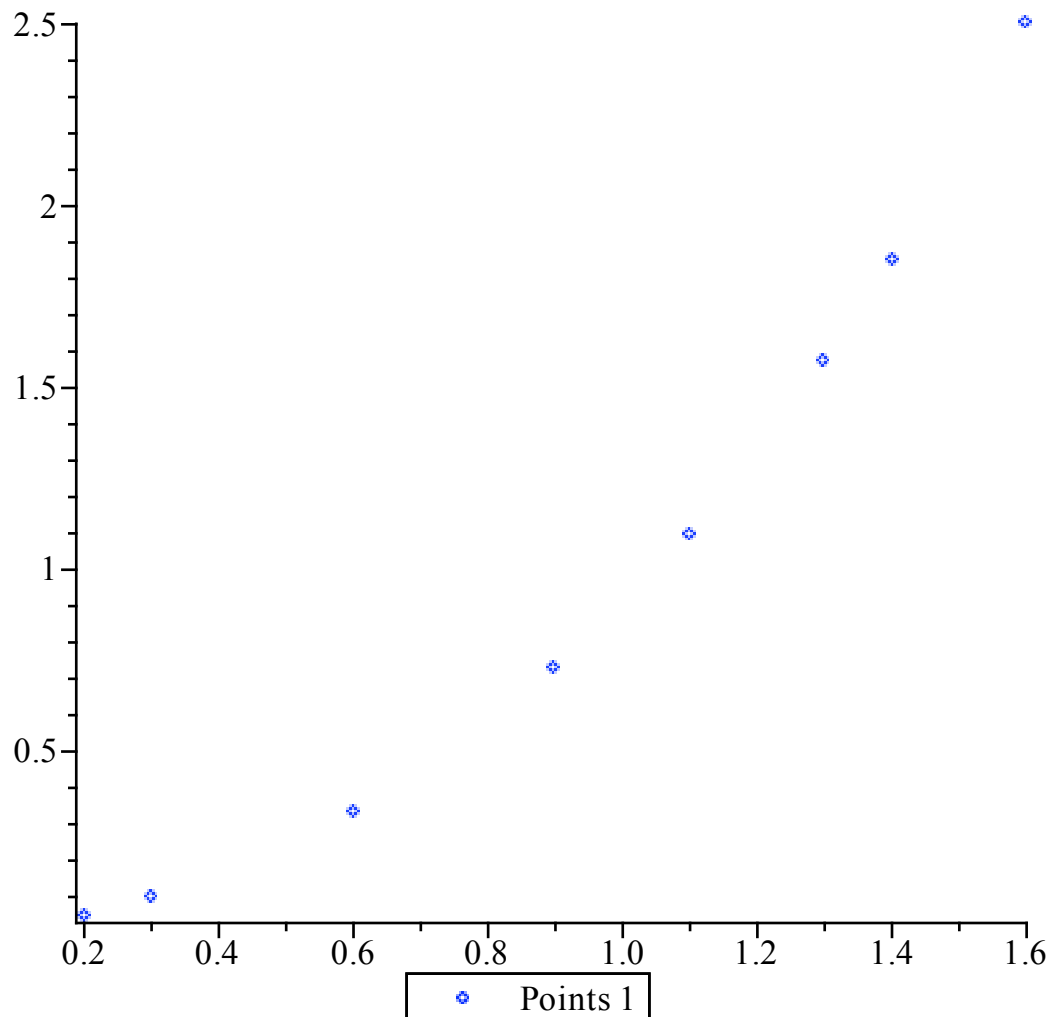
1.0972],[1.3,1.5697],[1.4,1.8487],[1.6,2.5015]];
X2 := [[0.2, 0.050446], [0.3, 0.098426], [0.6, 0.33277], [0.9, 0.72660], [1.1, 1.0972], [1.3, 1.5697],
[1.4, 1.8487], [1.6, 2.5015]]

```

```

> g2:=pointplot(X2,color=blue):
> display(g2);

```



Let's use Lagrange interpolation to interpolate the data sets. The first data set.

```

> p1:=PolynomialInterpolation(X1,independentvar=x,mrthod=lagrange);
p1 := POLYINTERP([[1, 1.3], [2, 3.5], [3, 4.2], [4, 5.0], [5, 7.0], [6, 8.8], [7, 10.1], [8, 12.5], [9,
13.0], [10, 15.6]], independentvar = x, mrthod = lagrange, INFO)
> poly1:=expand(Interpolant(p1));
poly1 := 31.31186446 x - 37.40852124 x2 + 27.4028439 x3 - 12.26852467 x4 + 3.37803228 x5
- 0.56982638 x6 + 0.057185845 x7 - 0.00312748014 x8 + 0.000071649030 x9 - 10.60000000

```

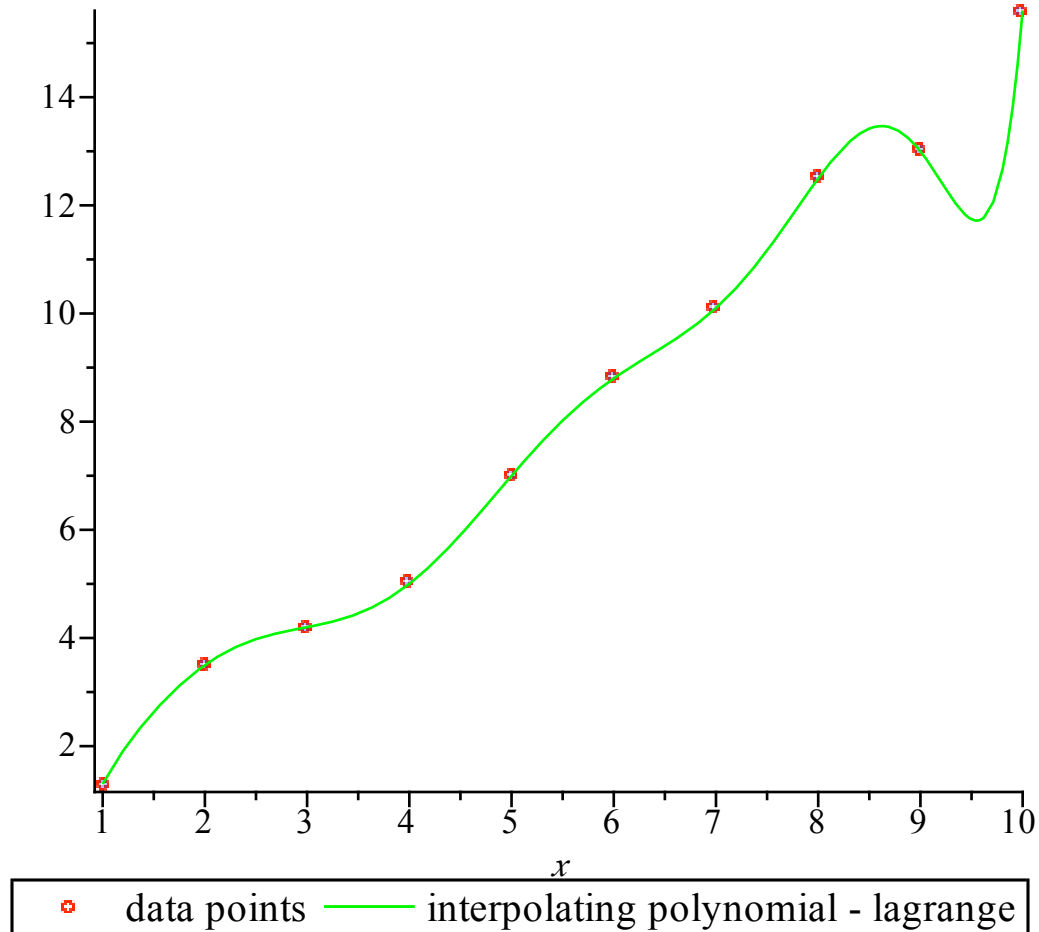
We plot the interpolating polynomial along with the data points.

```

> g11:=Draw(p1,x=1..10):
> display({g1,g11});

```

## Polynomial interpolation



Whereas the data points are more or less in a line, the interpolating polynomial is not, and as such is not likely a good tool for finding intermediate values. We now use Lagrange interpolation on the second set of values and plot this polynomial along with the data.

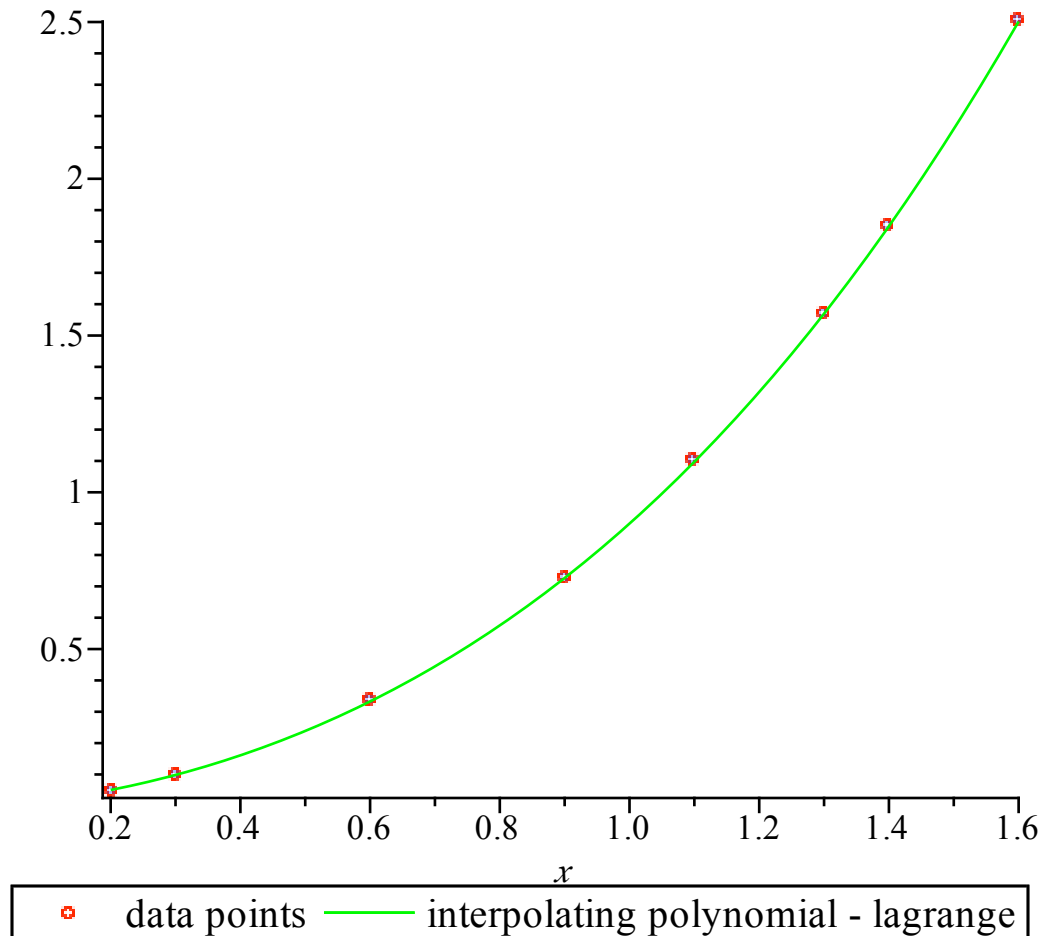
```
> p2:=PolynomialInterpolation(X2,independentvar=x,mrthod=lagrange);  
p2:=POLYINTERP([[0.2, 0.050446], [0.3, 0.098426], [0.6, 0.33277], [0.9, 0.72660], [1.1,  
1.0972], [1.3, 1.5697], [1.4, 1.8487], [1.6, 2.5015]], independentvar = x, mrthod = lagrange,  
INFO)
```

```
> poly2:=expand(Interpolant(p2));
```

```
poly2:= 0.10473301 x + 0.845868 x2 - 0.456660 x3 + 0.735559 x4 - 0.474410 x5 + 0.173062 x6  
- 0.0264051 x7 - 0.001717897
```

```
> g22:=Draw(p2,x=.2..1.6);  
> display({g2,g22});
```

## Polynomial interpolation



Here the second degree polynomial seems to give a good fit.

We can also look for a best curve of a certain type that fits the data in the sense of least squares, not concerning ourselves with hitting the exact data points.

We consider only linear regression for the first data set. We use the [LeastSquares](#) command [CurveFitting](#) package to get the regression equation. We waited until now to load this package since it changes the **PolynomialInterpolation** command.

```
> with(CurveFitting);  
[ArrayInterpolation, BSpline, BSplineCurve, Interactive, LeastSquares, PolynomialInterpolation,  
RationalInterpolation, Spline, ThieleInterpolation]
```

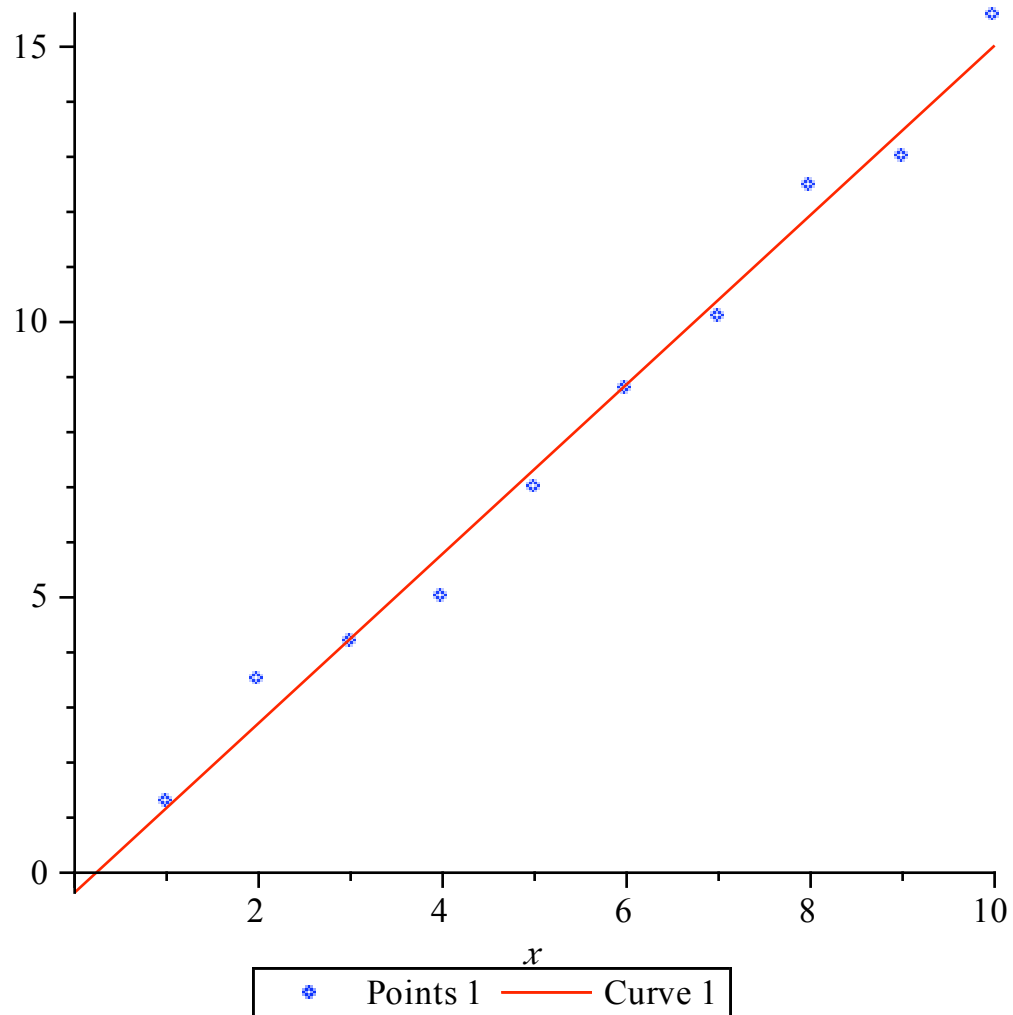
```
> f:=LeastSquares(X1,x);  
f:= -0.3600000000 + 1.538181818 x
```

We change the expression to a function.

```
> f:=unapply(f,x);  
f:= x→ -0.3600000000 + 1.538181818 x
```

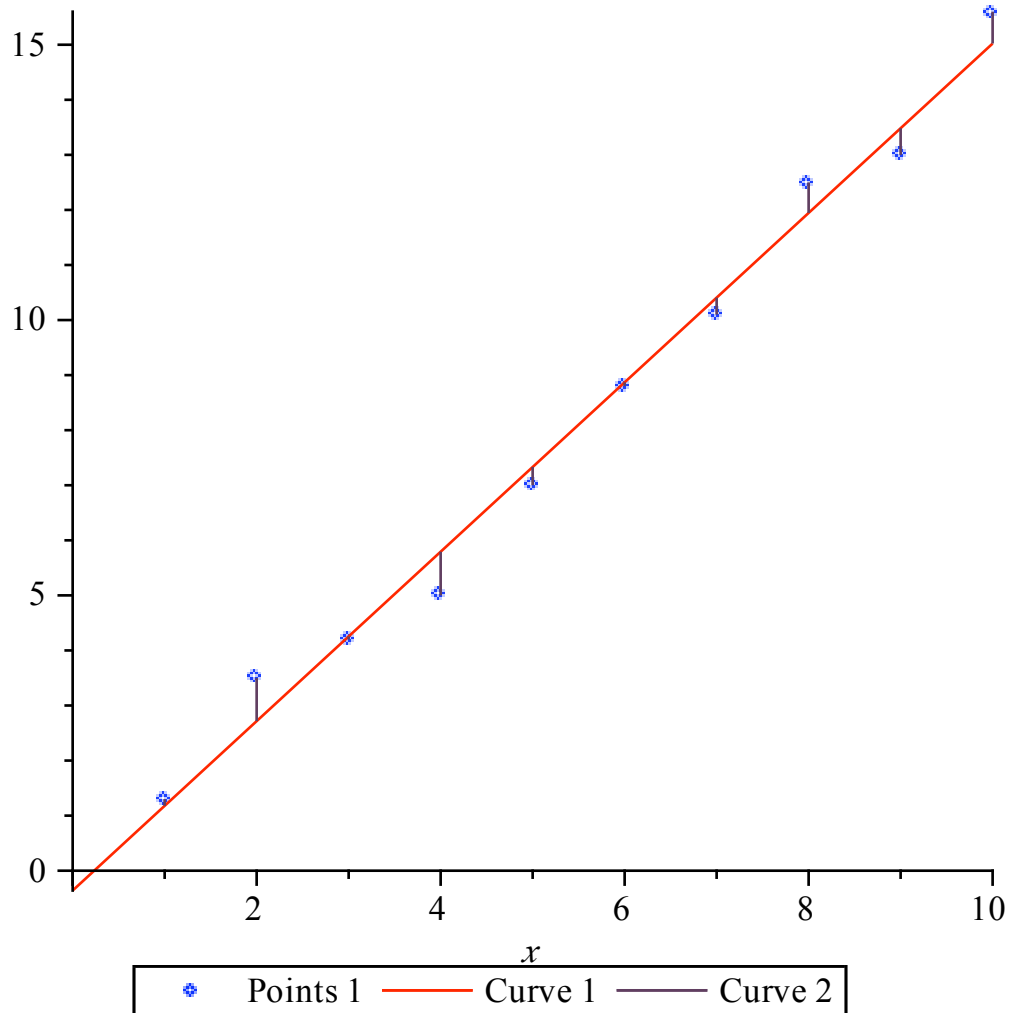
We plot the regression line along with the points.

```
> g111:=plot(f(x),x=0..10):  
> display({g1,g111});
```



We graphically show the absolute differences using the [pointplot](#) command.

```
> for i from 1 to 10 do
  p[i]:=pointplot([X1[i],[X1[i,1],f(i)]],style=line,color=violet):
od:
display(g1,g111,{seq(p[i],i=1..10)});
```

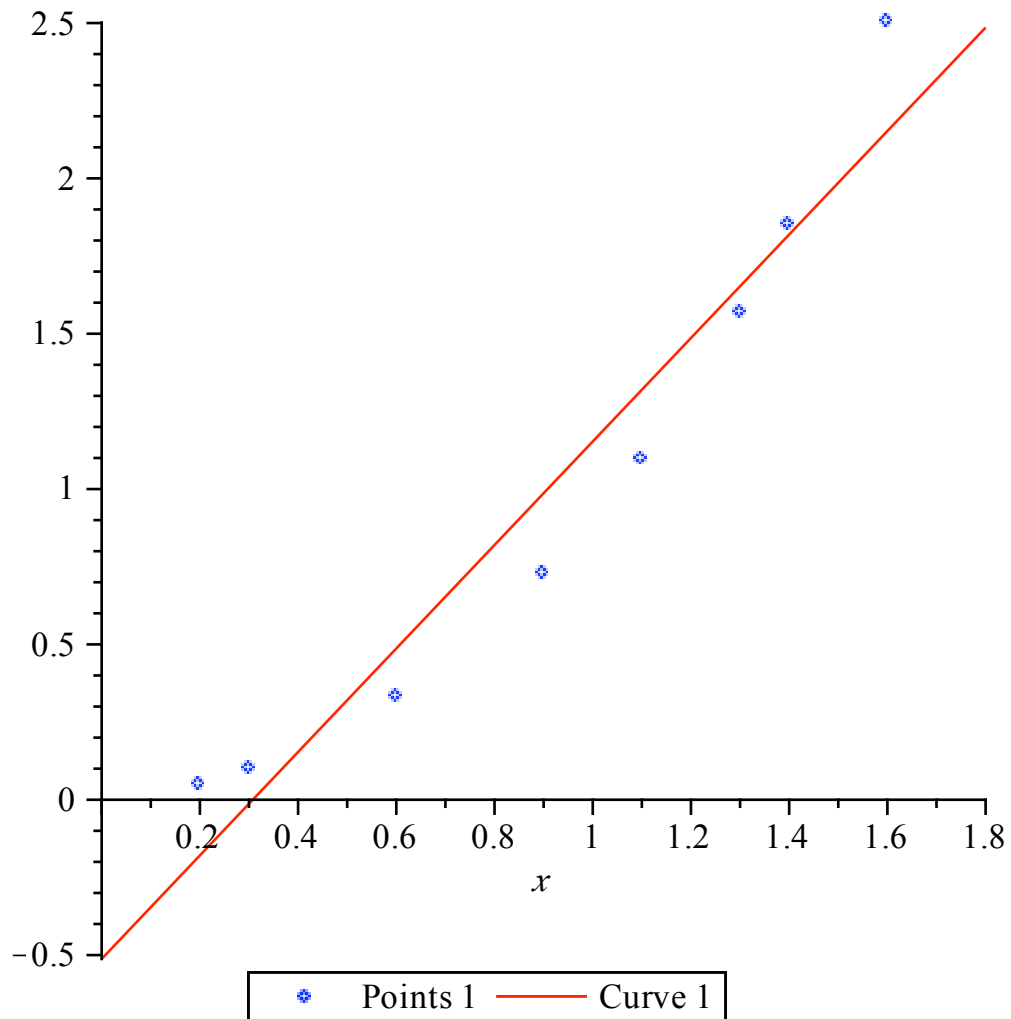


We compute the sum of the squares of the absolute differences, the number least squares regression minimizes.

```
> ls:=sum('(X1[i,2]-f(i))^2','i'=1..10);
          ls := 2.344727279
```

We consider several types of regression for the second set of data and compute the least squares data. We start with linear regression.

```
> f:=LeastSquares(X2,x);
          f := -0.5124568240 + 1.665540080 x
> f:=unapply(f,x);
          f := x → -0.5124568240 + 1.665540080 x
> g:=plot(f(x),x=0..1.8):
> display({g2,g});
```



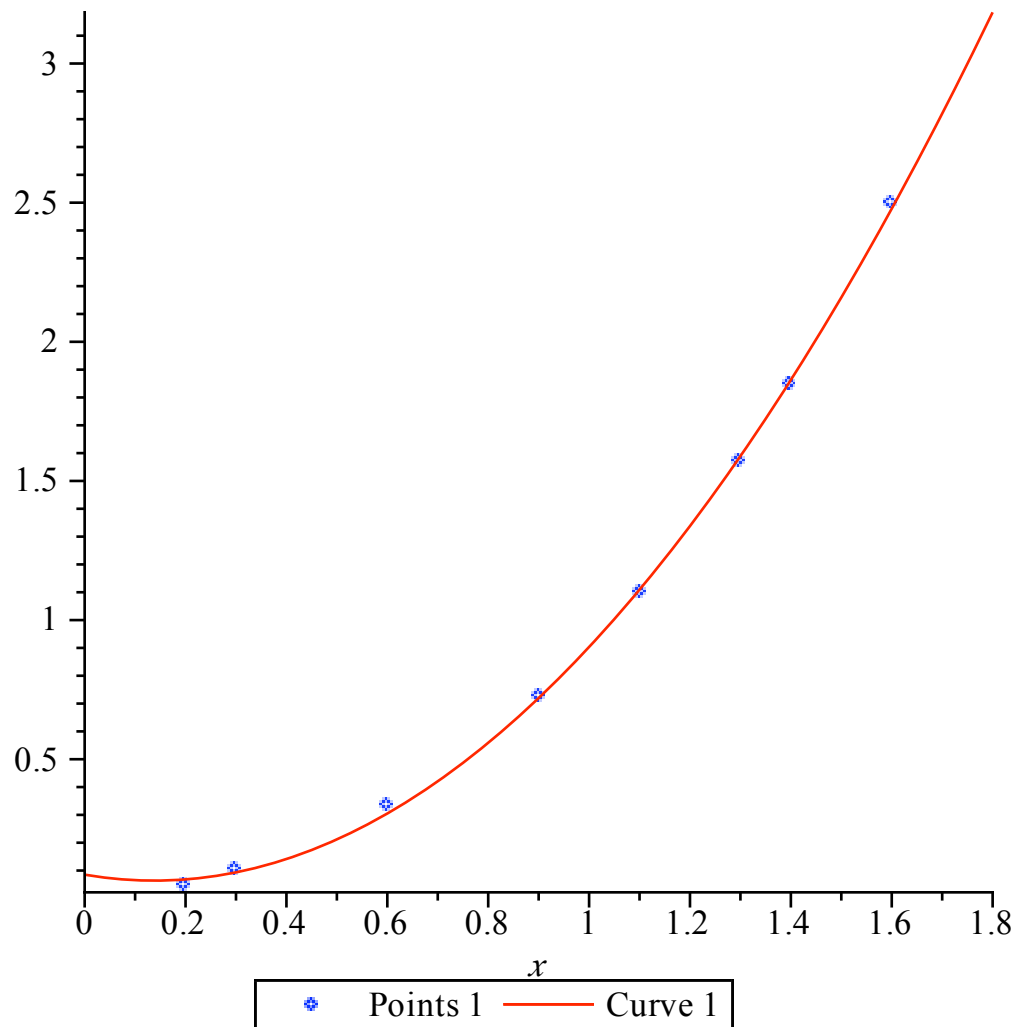
```
> ls:=sum(' (X2[i,2]-f(X2[i,1]))^2', 'i'=1..8);
ls := 0.3355898557
```

Even visually, this doesn't look like a good fit. We next look at polynomial regression of degree two. Notice the extra parameter.

```
> f:=LeastSquares(X2, x, curve=a*x^2+b*x+c);
f := 0.08514393252 - 0.3114034568 x + 1.129423867 x^2
```

```
> f:=unapply(f, x);
f := x → 0.08514393252 - 0.3114034568 x + 1.129423867 x^2
```

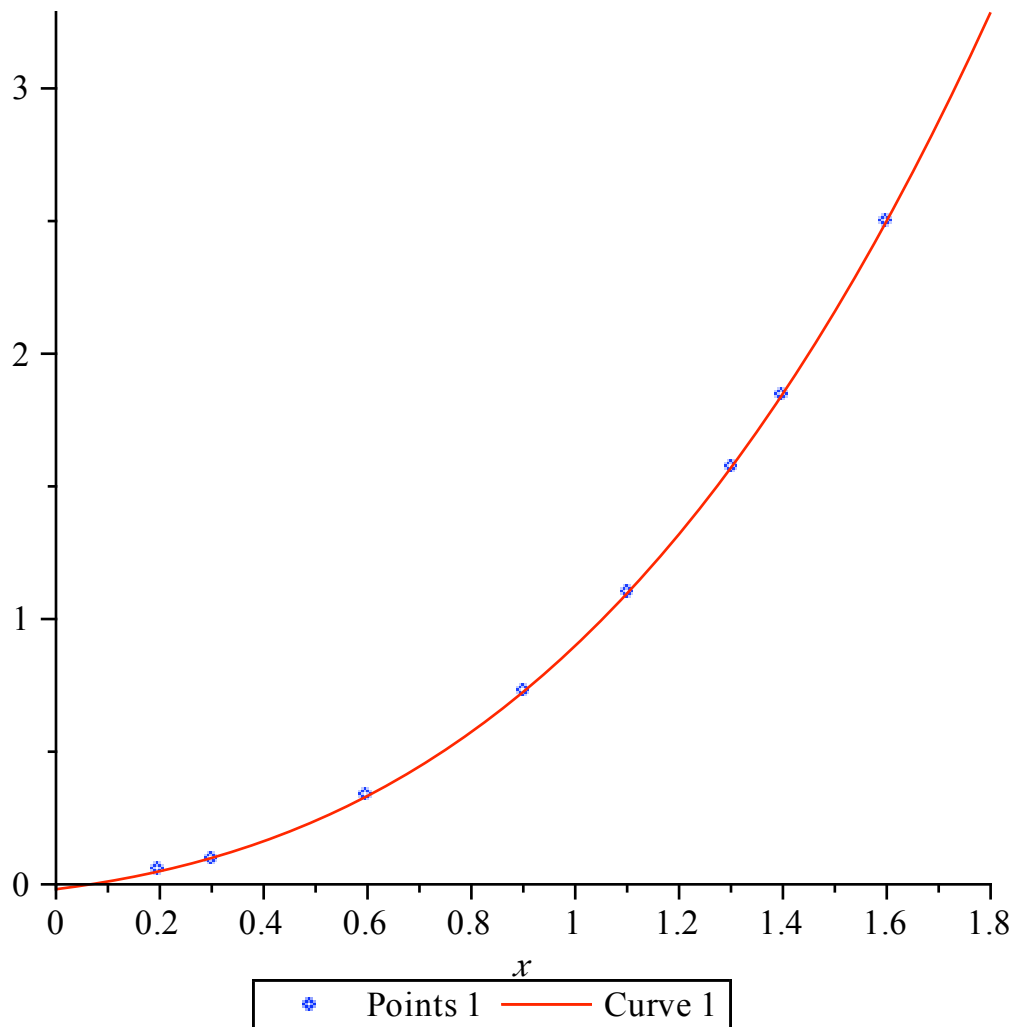
```
> g:=plot(f(x), x=0..1.8);
> display({g2, g});
```



```
> ls:=sum('(X2[i,2]-f(X2[i,1]))^2','i'=1..8);
ls:=0.002419914886
```

Both from visual inspection and computation of `ls`, this is a clear improvement. We next look at polynomial regression of degree three.

```
> f:=LeastSquares(X2,x,curve=a*x^3+b*x^2+c*x+d);
f:=-0.01840139930+0.2483857842x+0.4029322131x^2+0.2662080977x^3
> f:=unapply(f,x);
f:=x→-0.01840139930+0.2483857842x+0.4029322131x^2+0.2662080977x^3
> g:=plot(f(x),x=0..1.8);
> display({g2,g});
```



```
> ls:=sum('(X2[i,2]-f(X2[i,1]))^2','i'=1..8);
ls := 0.000005074671415
```

This cubic regression is even better. We now look at exponential regression of the form  $y = b e^{ax}$ . We implement this by using linear regression on the logarithm of the approximating equation:  $\ln y = \ln b + ax$ . We start by taking  $\ln$  of the list of y values.

```
> lny2:=[seq(ln(X2[i,2]),i=1..8)];
lny2 := [-2.986851822, -2.318450282, -1.100303718, -0.3193791592, 0.09276148008,
0.4508845183, 0.6144826894, 0.9168905519]
```

Now we do linear regression on the lists  $[\text{seq}(X2[i,1],i=1..8)]$  and  $\text{lny2}$  and change the result to a function.

```
> f:=LeastSquares([seq(X2[i,1],i=1..8)],lny2,x);
f := -3.085493303 + 2.707294687 x
> ln_y:=unapply(f,x);
ln_y := x → -3.085493303 + 2.707294687 x
```

We now find  $a$  and  $b$  by matching terms of the regression function with the logarithmic equation.

```
> b:=solve(ln_y(0)=ln(b),b);
b := 0.04570748070
> a:=ln_y(1)-ln_y(0);
```

$a := 2.707294687$

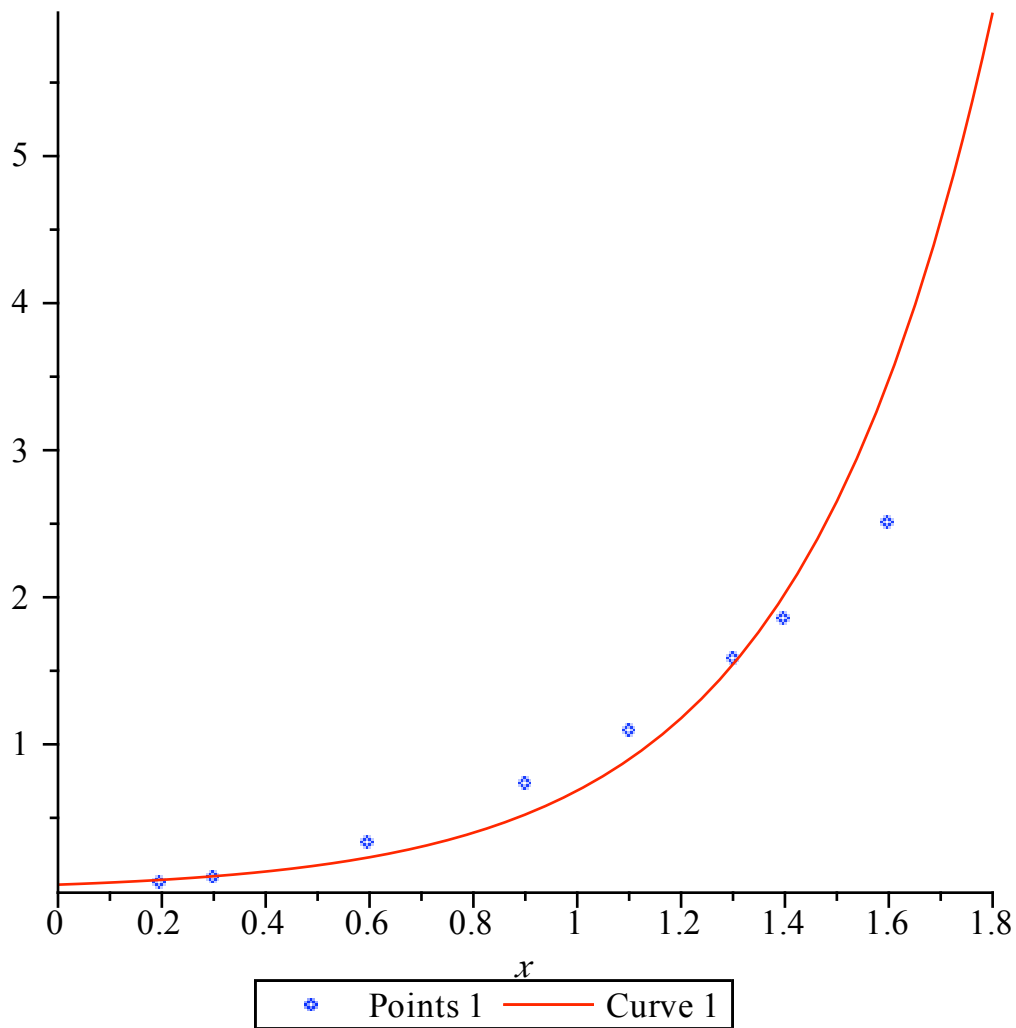
We form the exponential regression equation as a function.

```
> ff:=x->b*exp(a*x);
```

$ff := x \rightarrow b e^{ax}$

```
> g:=plot(ff(x),x=0..1.8):
```

```
> display({g2,g});
```



```
> ls:=sum('(X2[i,2]-ff(X2[i,1]))^2','i'=1..8);
```

$ls := 1.075048504$

This is clearly not as good as the cubic regression.