

Hermite Interpolation

Finding a Hermite Polynomial from Data Points Step-by-Step

```
> restart;
```

We take our data from Problem 6 on page 90. I set the digits to 15 since the data is given to almost 10 decimal places, the default value of digits.

```
> Digits:=15;
```

```
Digits := 15
```

I enter the actual function that the data values are taken from as a Maple function.

```
> f:=x->exp(0.1*x^2);
```

```
f := x → e0.1 x2
```

We find the derivative of the function.

```
> fprime:=D(f);
```

```
fprime := x → 0.2 x e0.1 x2
```

Choose n to be one less than the number of data points. We are given 4 data points.

```
> n:=3;
```

```
n := 3
```

The list of x-values follows. They are indexed as X[1], X[2], ..., X[n], X[n+1].

```
> X:=[1,1.5,2,3];
```

```
X := [1, 1.5, 2, 3]
```

The list of f(x)-values follows. They are indexed as Y[1], Y[2], ..., Y[n], Y[n+1].

```
> Y:=[1.105170918,1.252322716,1.491824698,2.459603111];
```

```
Y := [1.105170918, 1.252322716, 1.491824698, 2.459603111]
```

The list of f'(x)-values follows. They are indexed as Z[1], Z[2], ..., Z[n], Z[n+1].

```
> Z:=[.2210341836,.3756968148,.5967298792,1.475761867];
```

```
Z := [0.2210341836, 0.3756968148, 0.5967298792, 1.475761867]
```

Find the Lagrange coefficient polynomials $L[n,j]$ where $j = 0, 1, \dots, n$.

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

```
L3,0 := x → 9.000000000000000 - 1.000000000000000 x3 + 6.500000000000000 x2
      - 13.500000000000000 x
```

```
L3,1 := x → -16.000000000000000 + 2.666666666666666 x3 - 16.000000000000000 x2
      + 29.333333333333333 x
```

```
L3,2 := x → 9.000000000000000 - 2.000000000000000 x3 + 11.000000000000000 x2
```

$$- 18.000000000000000 x$$

$$L_{3,3} := x \rightarrow -1.000000000000000 + 0.3333333333333334 x^3 - 1.500000000000000 x^2 + 2.166666666666667 x$$

Find the derivatives of the Lagrange coefficient polynomials $L[n,j]$ where $j = 0, 1, \dots, n$.

```
> Lprime:=array(n..n,0..n):
> for j from 0 to n do
  Lprime[n,j]:=D(L[n,j]);
od;
```

$$Lprime_{3,0} := x \rightarrow -3.000000000000000 x^2 + 13.000000000000000 x - 13.500000000000000$$

$$Lprime_{3,1} := x \rightarrow 7.999999999999998 x^2 - 32.000000000000000 x + 29.333333333333333$$

$$Lprime_{3,2} := x \rightarrow -6.000000000000000 x^2 + 22.000000000000000 x - 18.000000000000000$$

$$Lprime_{3,3} := x \rightarrow 1.000000000000000 x^2 - 3.000000000000000 x + 2.166666666666667$$

Find the polynomials $H[n,j]$ where $j = 0, 1, \dots, n$.

```
> H:=array(n..n,0..n):
> for j from 0 to n do
  H[n,j]:=unapply(collect((1-2*(x-X[j+1])*Lprime[n,j](X[j+1]))*(L[n,j](x))^2,x),x);
od;
```

$$H_{3,0} := x \rightarrow -486.0000000000000 + 7.000000000000000 x^7 - 97.00000000000000 x^6 + 562.7500000000000 x^5 - 1770.0000000000000 x^4 + 3255.7500000000000 x^3 - 3496.5000000000000 x^2 + 2025.0000000000000 x$$

$$H_{3,1} := x \rightarrow -256.0000000000026 + 9.48148148148191 x^7 - 120.8888888888895 x^6 + 635.259259259295 x^5 - 1777.77777777788 x^4 + 2853.92592592611 x^3 - 2624.000000000020 x^2 + 1280.000000000011 x$$

$$H_{3,2} := x \rightarrow 729.0000000000000 - 16.0000000000000 x^7 + 212.0000000000000 x^6 - 1168.0000000000000 x^5 + 3465.0000000000000 x^4 - 5976.0000000000000 x^3 + 5994.0000000000000 x^2 - 3240.0000000000000 x$$

$$H_{3,3} := x \rightarrow 14.000000000000000 - 0.481481481481486 x^7 + 5.888888888888891 x^6 - 30.0092592592593 x^5 + 82.7777777777778 x^4 - 133.675925925926 x^3 + 126.5000000000000 x^2 - 65.00000000000001 x$$

Find the polynomials $Hhat[n,j]$ where $j = 0, 1, \dots, n$.

```
> Hhat:=array(n..n,0..n):
> for j from 0 to n do
  Hhat[n,j]:=unapply(collect((x-X[j+1])*L[n,j](x))^2,x),x);
od;
```

$$Hhat_{3,0} := x \rightarrow -81.00000000000000 + 1.000000000000000 x^7 - 14.000000000000000 x^6 + 82.25000000000000 x^5 - 262.7500000000000 x^4 + 492.7500000000000 x^3$$

$$- 542.2500000000000 x^2 + 324.0000000000000 x$$

$$Hhat_{3,1} := x \rightarrow -384.0000000000000 + 7.111111111111108 x^7 - 95.9999999999998 x^6$$

$$+ 540.4444444444444 x^5 - 1642.666666666667 x^4 + 2908.444444444444 x^3$$

$$- 2997.333333333333 x^2 + 1664.000000000000 x$$

$$Hhat_{3,2} := x \rightarrow -162.0000000000000 + 4.0000000000000 x^7 - 52.0000000000000 x^6$$

$$+ 281.0000000000000 x^5 - 818.0000000000000 x^4 + 1386.0000000000000 x^3$$

$$- 1368.0000000000000 x^2 + 729.0000000000000 x$$

$$Hhat_{3,3} := x \rightarrow -3.000000000000000 + 0.111111111111112 x^7 - 1.333333333333334 x^6$$

$$+ 6.694444444444444 x^5 - 18.250000000000000 x^4 + 29.19444444444445 x^3$$

$$- 27.41666666666667 x^2 + 14.000000000000000 x$$

Find the Hermite polynomial $H_{(2n+1)}(x)$.

```
> HH:=0:
  for j from 0 to n do
    HH:=HH+Y[j+1]*H[n,j](x)+Z[j+1]*Hhat[n,j](x):
  od:
```

```
> HH:=unapply(collect(HH,x),x);
```

$$HH := x \rightarrow 0.99809516577000 + 0.00017158416343 x^7 - 0.0013243050518 x^6$$

$$+ 0.0062401696614 x^5 - 0.010023101401 x^4 + 0.021932755878 x^3 + 0.080793565433 x^2$$

$$+ 0.009285083540 x$$

We evaluate the Hermite polynomial at 1.25.

```
> HH(1.25);
```

$$1.16911825830786$$

We compare this to the value of f at 1.25.

```
> f(1.25);
```

$$1.16911844616950$$

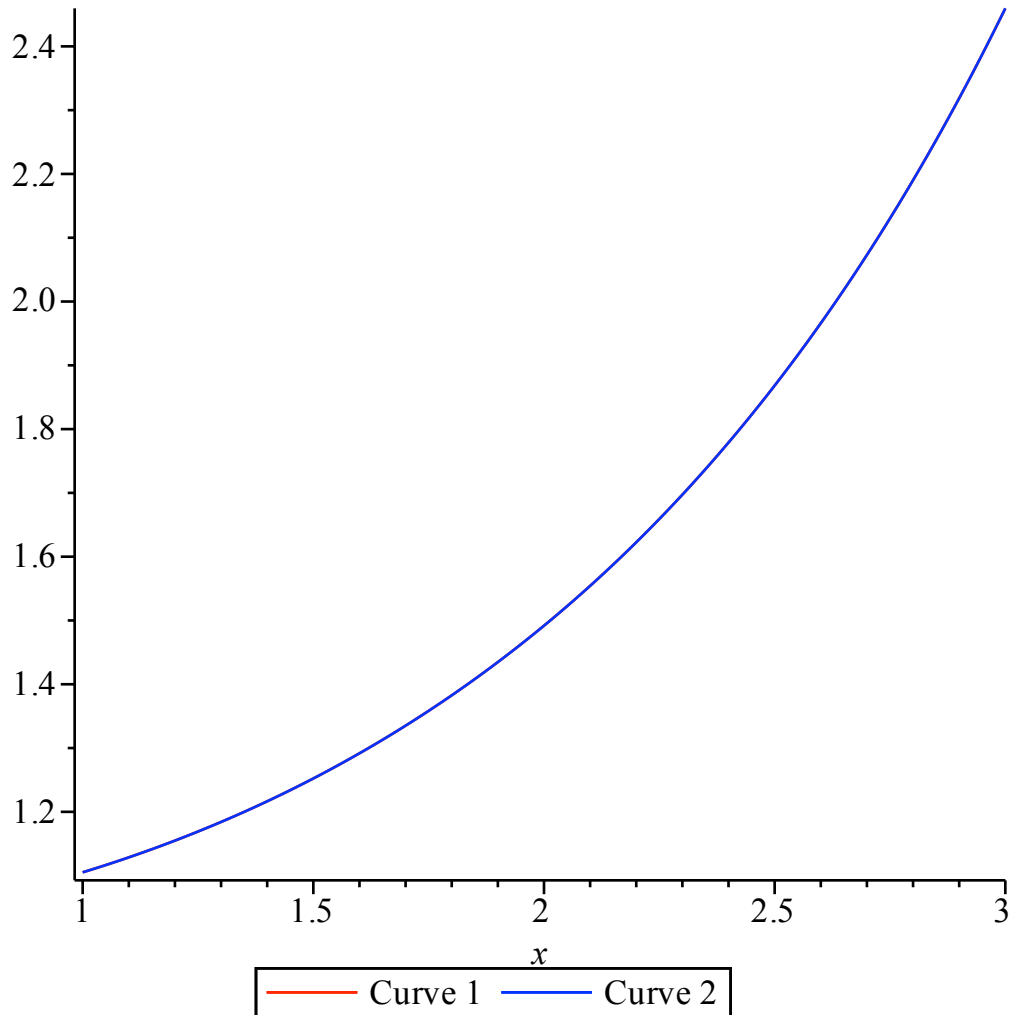
The difference at 1.25.

```
> f(1.25)-HH(1.25);
```

$$1.8786164 \cdot 10^{-7}$$

Pretty good. After loading the plots package, we graph both the function (red) and the Hermite polynomial (blue).

```
> with(plots):
> p1:=plot({f(x)},x=1..3,color=red):
  p2:=plot({HH(x)},x=1..3,color=blue):
  display(p1,p2);
```



Almost identical!

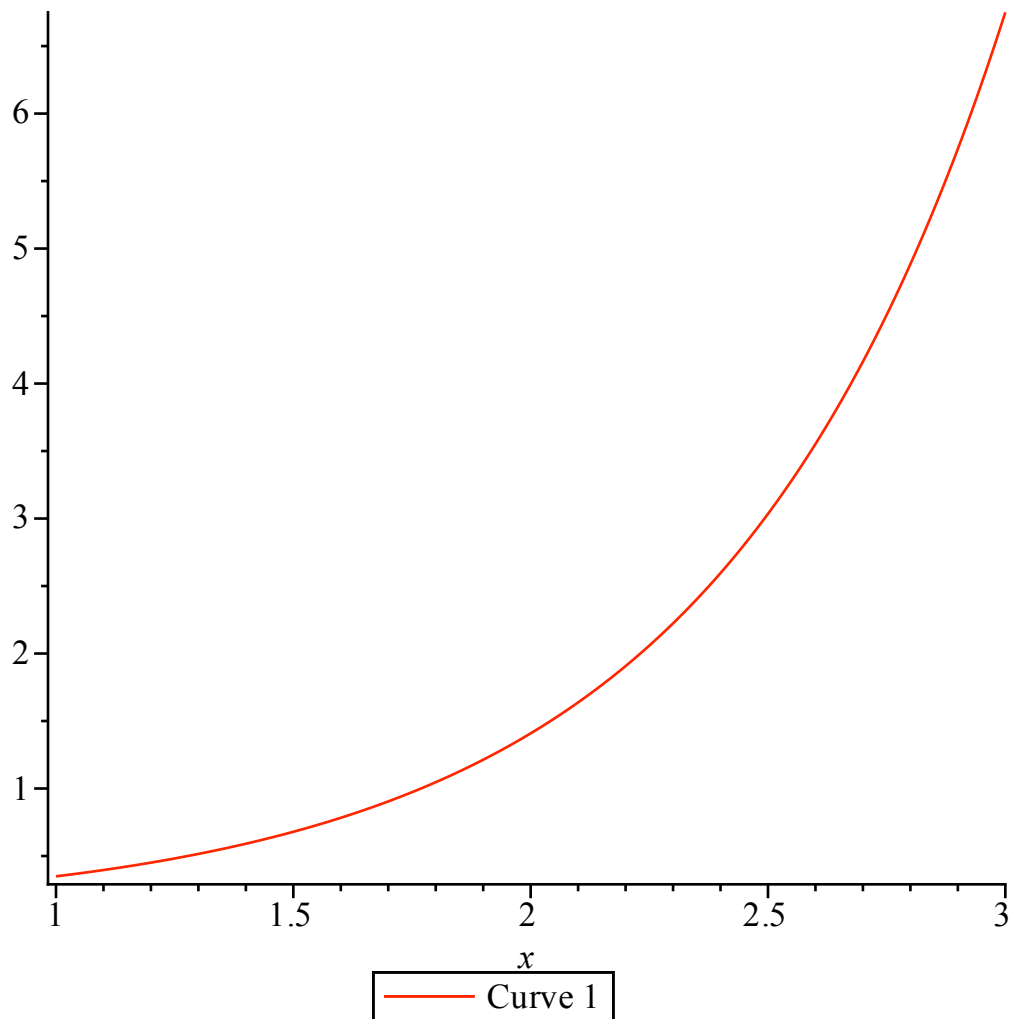
Hermite error at a point and over an interval.

The error $f(x) - H_{2n+1}(x)$ in approximating $f(x)$ by $H_{2n+1}(x)$ is

$$\frac{1}{(2n+2)!} f^{(2n+2)}(\xi(x)) (x-x_0)^2 (x-x_1)^2 \dots (x-x_n)^2.$$

Let's find the maximum possible error at 1.25. We need the 8th derivative, which we find and plot..

```
> f8 := (D@@8)(f);
f8 := x -> 0.1680 e^{0.1 x^2} + 0.13440 x^2 e^{0.1 x^2} + 0.013440 x^4 e^{0.1 x^2} + 0.0003584 x^6 e^{0.1 x^2}
      + 0.00000256 x^8 e^{0.1 x^2}
> plot(f8(x), x=1..3);
```



We find the maximum absolute value of this derivative.

```
> maxderiv:=maximize(f8(x),x=0..3);
maxderiv := 6.74991282367438
```

Looking at the graph, we set `maxderiv` equal to 7. We compute the **maximum absolute error at the point 1.25**.

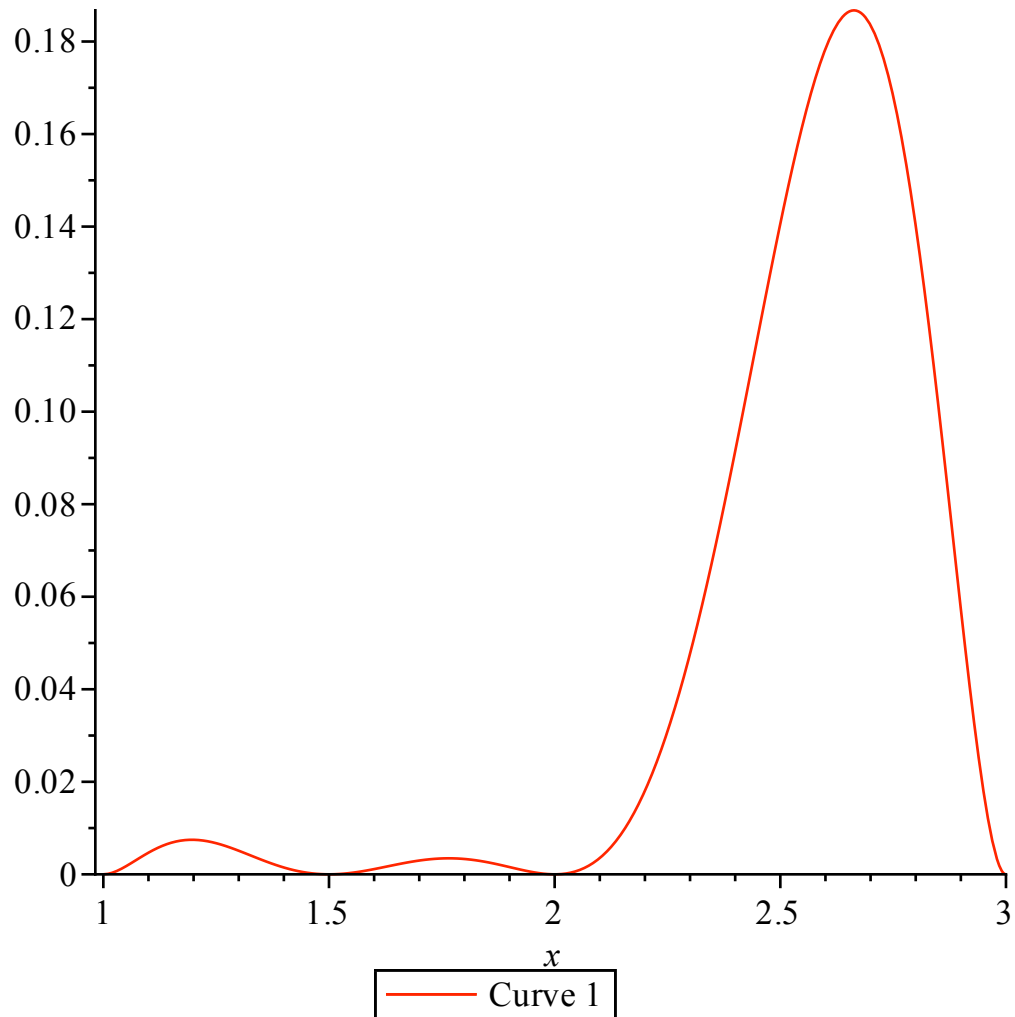
```
> maxerror:=(maxderiv/8!)*(1.25-1)^2*(1.25-1.5)^2*(1.25-2)^2*(1.25-3)^2;
maxerror := 0.00000112651323713590
```

Our error above was certainly within that bound. What if we wanted an **error bound over the entire interval** $[1, 3]$? Instead of using 1.25 for x in $(x-1)^2(x-1.5)^2(x-2)^2(x-3)^2$, we first set $g(x) = (x-1)^2(x-1.5)^2(x-2)^2(x-3)^2$ and find a bound for this function on $[1, 3]$.

```
> g:=(x-1)^2*(x-1.5)^2*(x-2)^2*(x-3)^2;
g := (x - 1)^2 (x - 1.5)^2 (x - 2)^2 (x - 3)^2
```

We plot this function over $[1,3]$.

```
> plot(g,x=1..3);
```



We find the maximum absolute value of this function.

```
> maxfunc:=maximize(g,x=1..3);
maxfunc := 0.186737243767296
```

We see we can use 0.187 as an upper bound on the absolute value.

```
> maxdistance:=0.187;
maxdistance := 0.187
```

We now compute the upper bound on the error over the interval.

```
> maxerrorinterval:=(7/8!)*maxdistance;
maxerrorinterval := 0.0000324652777777778
```

Thus we have a very good approximation.

nalib

Two Hermite procedures

```
> restart;
> libname:="c:/nalib",libname;
libname := "/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 return to Runge's function,

$$f(x) = \frac{1}{1 + 25x^2},$$

evaluated to four decimal places on the interval [-1, 1]. We use Hermite interpolation with 9 equally spaced points. We begin by entering the function, the x-list, the f(x)-list, and the f'(x)-list.

```
> f:=1/(1+25*x^2);
```

$$f := \frac{1}{1 + 25x^2}$$

```
> X:=[-1,-.75,-.5,-.25,0,.25,.5,.75,1];
```

```
X := [-1, -0.75, -0.5, -0.25, 0, 0.25, 0.5, 0.75, 1]
```

```
> Y:=[.0385,.0664,.1379,.3902,1.0000,.3902,.1379,.0664,.0385];
```

```
Y := [0.0385, 0.0664, 0.1379, 0.3902, 1.0000, 0.3902, 0.1379, 0.0664, 0.0385]
```

```
> Z := [.0740,.1653,.4756,1.9036,0,-1.9036,-.4756,-.1653,-.0740];
```

```
Z := [0.0740, 0.1653, 0.4756, 1.9036, 0, -1.9036, -0.4756, -0.1653, -0.0740]
```

Our first procedure for Hermite interpolation is **hermite**, which compresses the above into a Maple procedure. After checking the directions for **hermite**, we use it to find the Hermite interpolating polynomial of degree at most 17 ($2N + 1$ here) that interpolates Runge's function at the chosen points.

```
> hermite_dir();
```

hermite returns the interpolating polynomial.

The arguments for hermite are:

- (1) the list of x-values
- (2) the list of f(x) or y-values
- (3) the list of f'(x)-values
- (4) the variable for returning the polynomial

If assigning the result to a variable, have the variable and the 4th argument the same.

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

```
p:='p'
```

```
> p:=hermite(X,Y,Z,p);
```

The interpolating polynomial is $-19.43532037x^2 + 6027.771893x^8 - 0.0000375973x^7 - 1567.114239x^6 - 0.375973e-4x^7 - 1567.114239x^6 + 0.5861119e-5x^5 + 232.2699421x^4 - 0.2346588e-6x^3 + 1.485189e-3x^{17} + 2882.961048x^{16} - 0.659077e-3x^{15} - 10926.47355x^{14} + 0.886016e-3x^{13} + 16721.44643x^{12} - 0.6965352e-3x^{11} - 13352.38772x^{10} + 0.207761e-3x^9 + 0.9999999986$

```
p := -19.43532037 x^2 + 6027.771893 x^8 - 0.0000375973 x^7 - 1567.114239 x^6
```

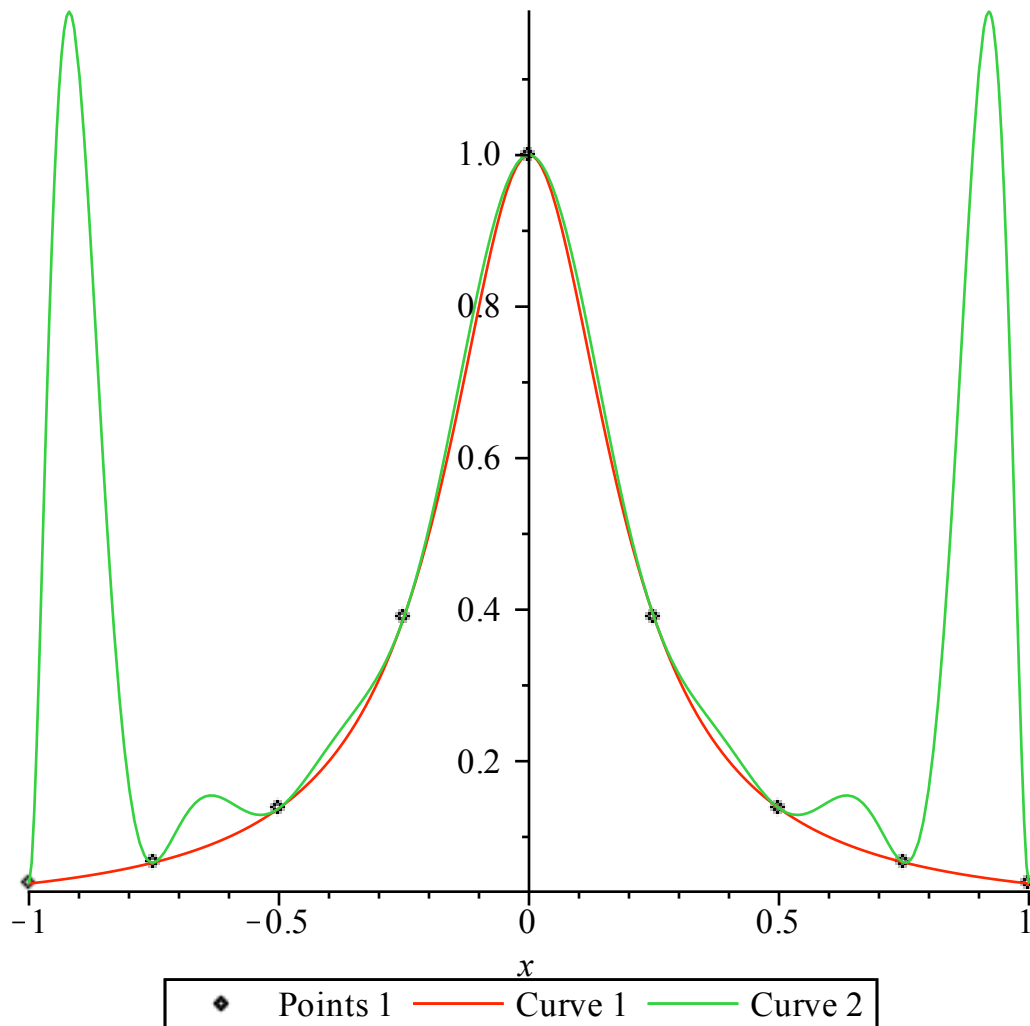
$$\begin{aligned}
& + 0.000005861119 x^5 + 232.2699421 x^4 - 2.346588 \cdot 10^{-7} x^3 + 0.0001485189 x^{17} \\
& + 2882.961048 x^{16} - 0.000659077 x^{15} - 10926.47355 x^{14} + 0.000886016 x^{13} \\
& + 16721.44643 x^{12} - 0.0006965352 x^{11} - 13352.38772 x^{10} + 0.000207761 x^9 + 0.9999999986
\end{aligned}$$

Let's see how well this approximates Runge's function.

```

> with(plots):
> plot1:=pointplot({seq( [X[i],Y[i]], i=1..9)}):
> plot2:=plot({f,p},x=-1..1):
> display(plot1,plot2);

```



I guess this is just a difficult function for any interpolating polynomial. But do notice that the function and interpolating polynomial both have the same derivative at the specified points. Our second procedure for Hermite interpolation uses the divided-differences approach and is called `hermite_dd`. Let's use it to look at Problem 7 on Page 90 of the text. We run `hermite_dd` on the data given, realizing that the speed row gives the derivatives. One caution about `hermite_dd` is that it rounds all values output to the table to 7 decimal places, so that may sometimes have an effect on accuracy.

```

> hermite_dd_dir();
hermite_dd builds a divided-difference table and
returns the interpolating polynomial
with all coefficients rounded to 7 decimal places.

```


For Part (a), we simply find $p(10)$ and $p'(10)$.

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

```
p := x → 75.00000000 x + 7.157109000 x2 - 10.08864880 x3 + 5.504311400 x4 - 1.537134200 x5  
+ 0.2428377000 x6 - 0.02185520000 x7 + 0.001039500000 x8 - 0.00002020000000 x9
```

```
> p(10);
```

742.45610

This seems reasonable.

```
> pp:=D(p);
```

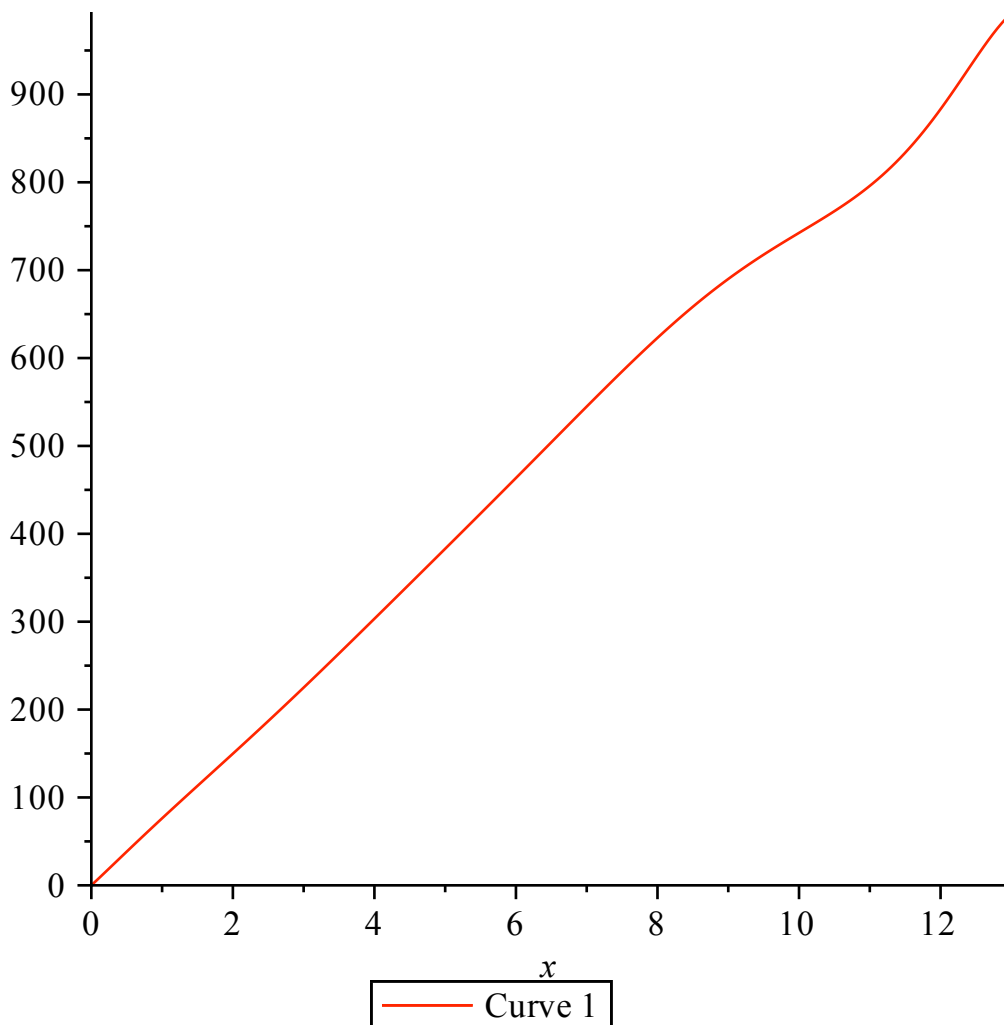
```
pp := x → 75.00000000 + 14.31421800 x - 30.26594640 x2 + 22.01724560 x3 - 7.685671000 x4  
+ 1.457026200 x5 - 0.1529864000 x6 + 0.008316000000 x7 - 0.0001818000000 x8
```

```
> pp(10);
```

48.30314

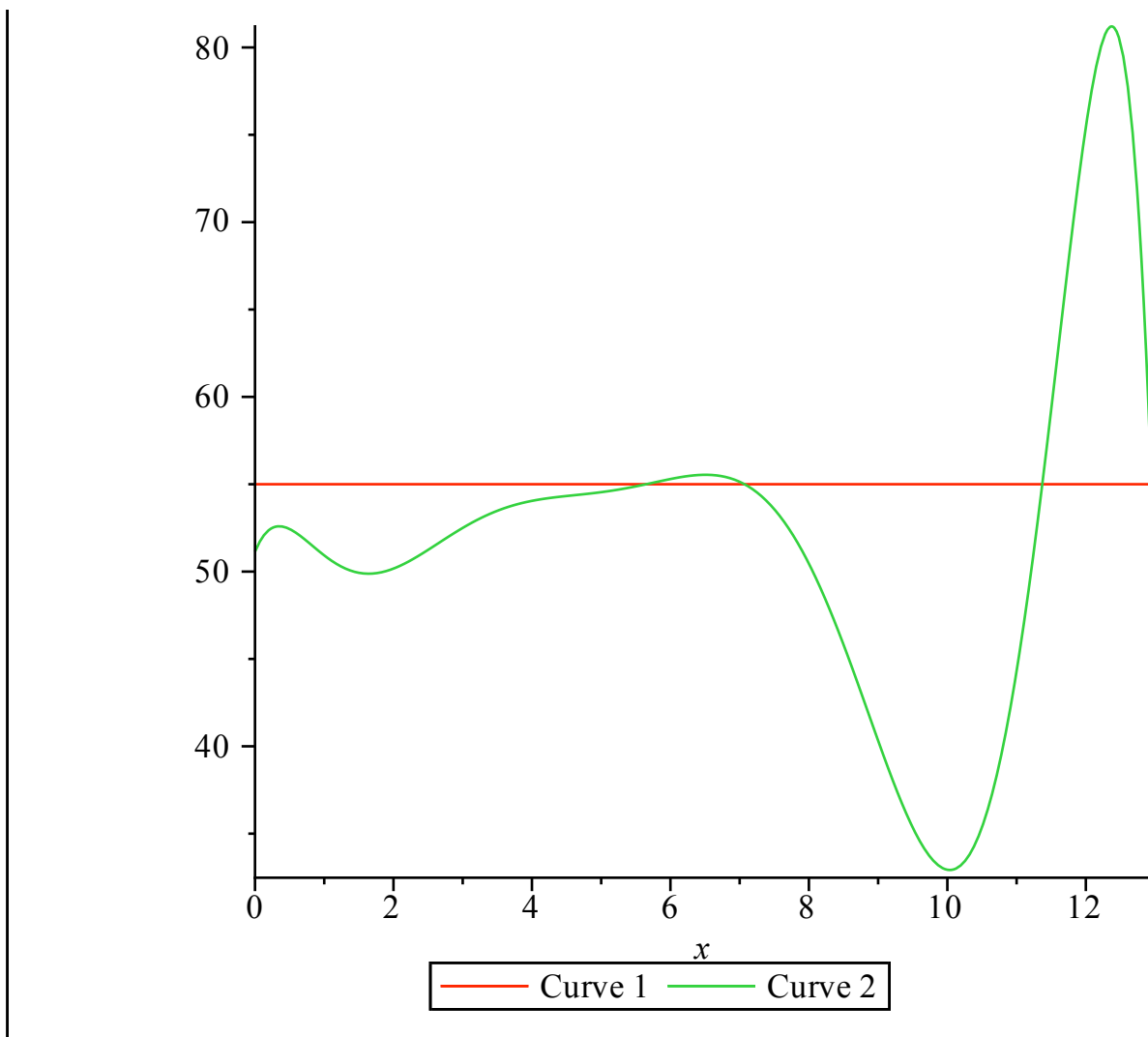
What do you think of this answer? Let's look at the plot of our polynomial.

```
> plot(p(x),x=0..13);
```



What do you see? Now let's plot its derivative (i.e., the speed) in mph.

```
> plot({3600/5280*pp(x),55},x=0..13);
```



Now what do you see? How well does it fit the data?

NumericalAnalysis

The command [PolynomialInterpolation](#) in the [NumericalAnalysis](#) subpackage can also be used for Hermite polynomials.

```
> restart;with(Student[NumericalAnalysis]):
```

I enter the actual function that the data values are taken from as a Maple function. The data is from Problem 6 on page 90.

```
> f:=x->exp(0.1*x^2);
```

$$f := x \rightarrow e^{0.1 x^2}$$

We find the derivative of the function.

```
> fprime:=D(f);
```

$$fprime := x \rightarrow 0.2 x e^{0.1 x^2}$$

We need to provide the data as a list of *XYD* data triples. We start with 5 points.

```
> XYD:=[[1, f(1), fprime(1)],[1.5, f(1.5), fprime(1.5)],[2, f(2), fprime(2)
],[3, f(3), fprime(3)]];
XYD := [[1, 1.105170918, 0.2210341836], [1.5, 1.252322716, 0.3756968148], [2, 1.491824698,
0.5967298792], [3, 2.459603111, 1.475761867]]
```

For Newton's divided-difference method, we use the argument **method=hermite**.

```
> p1:=PolynomialInterpolation(XYD,method=hermite,function=f(x),
independentvar=x);
p1 := POLYINTERP([ [1, 1.105170918, 0.2210341836], [1.5, 1.252322716, 0.3756968148], [2,
1.491824698, 0.5967298792], [3, 2.459603111, 1.475761867]], method = hermite, function
= e0.1 x2, independentvar = x, INFO)
```

The procedure **POLYERP** is where all the computed information is stored. We find the divided-difference table.

```
> DividedDifferenceTable(p1);
[[1.105170918, 0, 0, 0, 0, 0, 0, 0],
 [1.105170918, 0.2210341836, 0, 0, 0, 0, 0, 0],
 [1.252322716, 0.2943035960, 0.1465388248, 0, 0, 0, 0, 0],
 [1.252322716, 0.3756968148, 0.1627864376, 0.03249522560, 0, 0, 0, 0],
 [1.491824698, 0.4790039640, 0.2066142984, 0.04382786080, 0.01133263520, 0, 0, 0],
 [1.491824698, 0.5967298792, 0.2354518304, 0.05767506400, 0.01384720320, 0.002514568000,
0, 0],
 [2.459603111, 0.9677784130, 0.3710485338, 0.09039780227, 0.02181515885, 0.003983977825,
0.0007347049125, 0],
 [2.459603111, 1.475761867, 0.5079834540, 0.1369349202, 0.03102474529, 0.006139724293,
0.001077873234, 0.0001715841608]]
```

Remember that the **i** and **x_i** columns are not included here. We find the interpolating polynomial.

```
> p:=Interpolant(p1);
p := 0.8841367344 + 0.2210341836 x + 0.1465388248 (x - 1.)2 + 0.03249522560 (x - 1.)2 (x
- 1.5) + 0.01133263520 (x - 1.)2 (x - 1.5)2 + 0.002514568000 (x - 1.)2 (x - 1.5)2 (x
- 2.) + 0.0007347049125 (x - 1.)2 (x - 1.5)2 (x - 2.)2 + 0.0001715841608 (x
- 1.)2 (x - 1.5)2 (x - 2.)2 (x - 3.)
```

Notice the **basis functions** it is given in terms of.

```
> b:=BasisFunctions(p1);
b := [1, x - 1., (x - 1.)2, (x - 1.)2 (x - 1.5), (x - 1.)2 (x - 1.5)2, (x - 1.)2 (x - 1.5)2 (x
- 2.), (x - 1.)2 (x - 1.5)2 (x - 2.)2, (x - 1.)2 (x - 1.5)2 (x - 2.)2 (x - 3.)]
```

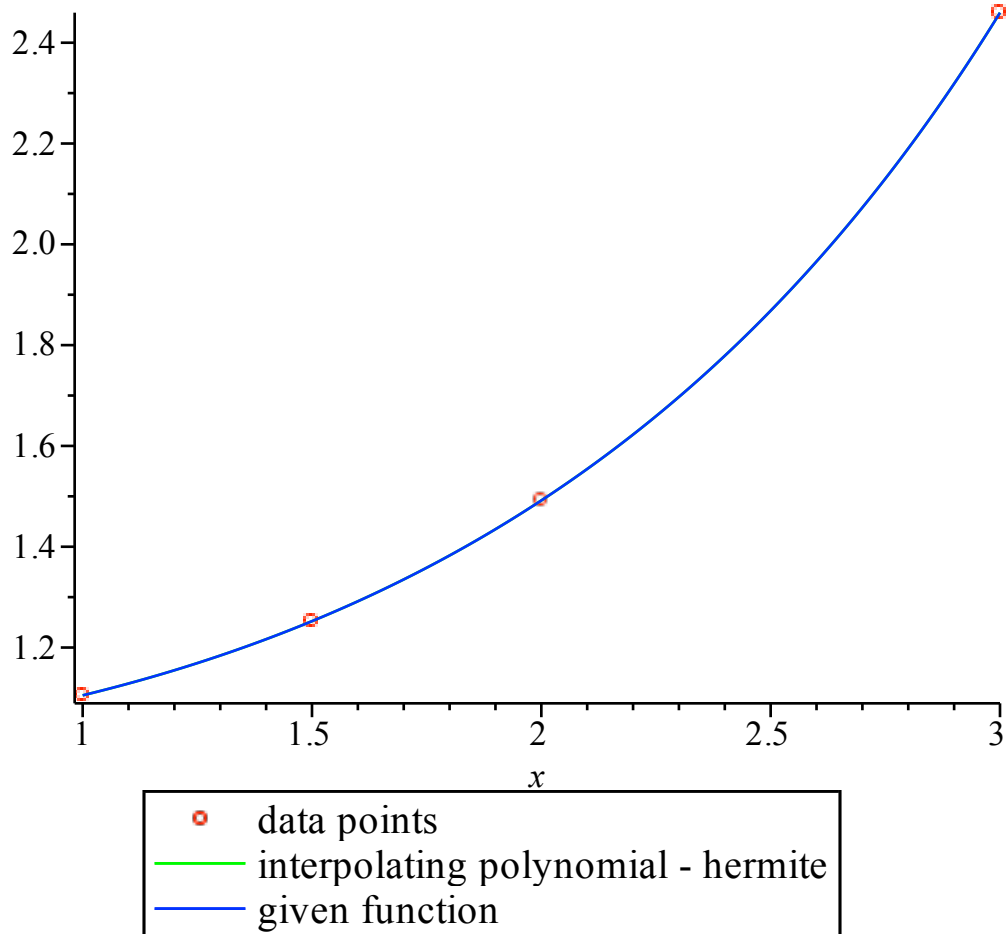
We simplify our polynomial.

```
> p:=expand(p);
p := 0.00928508307 x + 0.9980951657 + 0.08079356632 x2 + 0.02193275499 x3
- 0.01002310087 x4 + 0.006240169478 x5 + 0.0001715841608 x7 - 0.001324305018 x6
```

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);
```

Polynomial interpolation



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

```
> r:=RemainderTerm(p1);
```

$$r := \left(\frac{1}{40320} \left(0.1680 e^{0.1 xi_var^2} + 0.13440 xi_var^2 e^{0.1 xi_var^2} + 0.013440 xi_var^4 e^{0.1 xi_var^2} + 0.0003584 xi_var^6 e^{0.1 xi_var^2} + 0.00000256 xi_var^8 e^{0.1 xi_var^2} \right) (x - 1.)^2 (x - 1.5)^2 (x - 2.)^2 (x - 3.)^2 \right) \&where \{1. \leq xi_var \text{ and } xi_var \leq 3.\}$$

The [UpperBoundOfRemainderTerm](#) command gives us an absolute upper bound for the error at 1.25, which could be replaced by a list if more points were desired.

```
> ub1:=UpperBoundOfRemainderTerm(p1,1.25);
      ub1 := [[1.25, 0.000001126513237]]
```

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(p1,1.25);
      ub2 := [[1.25, 1.169118258, 1.169118446, 0.000001126513237]]
```

We view the graph of the function and the Hermite polynomial together.

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

Polynomial interpolation

