The Interpolation Problem

Problem: We know the value of a function, $x(t)$, at points $t_0, t_1, \ldots, t_n$. For some new point, t, what is the value of $x(t)$?

Examples:

- We have measured the elevation at points along a certain highway. We need to know the elevation at some intermediate points.

- A computer program that takes a long time to run has calculated the switching speeds of transistors of various sizes. We want to know the switching speed for other, intermediate, sizes.

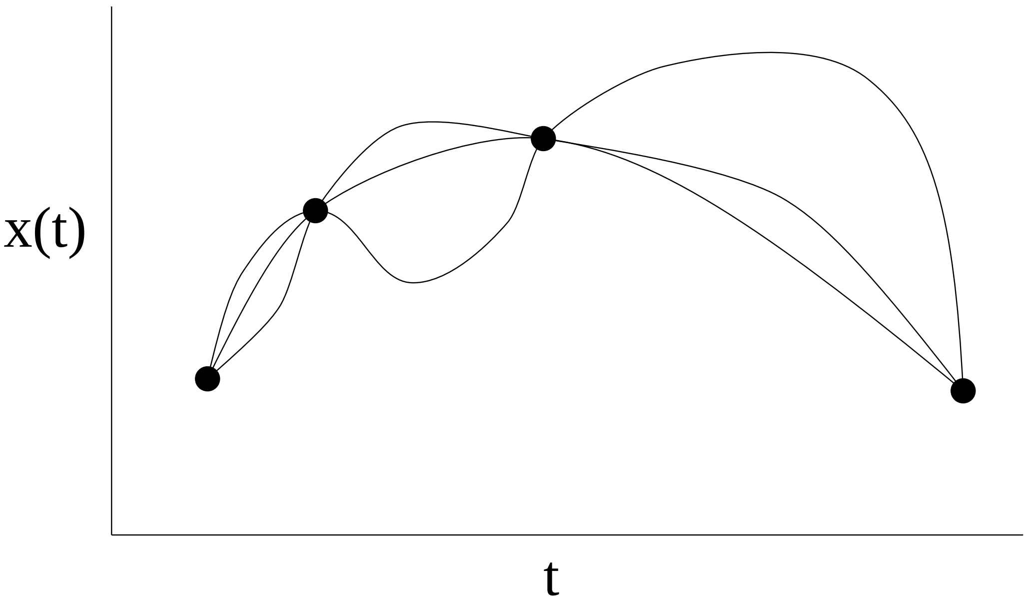
We will assume that t is a real number, but situations where t is a vector of real numbers also arise in practice.

We will often look at several functions — eg, $x(t)$, $y(t)$, $z(t)$, defining a parametric curve.

Interpolation is Not Well Defined

Unfortunately, interpolation is not a well-defined problem. Many possible functions will pass through the known points.

For example:



Things are even worse when we look beyond the range of the known points — when we try to *extrapolate*, rather than interpolate.

So What Should We Do?

We can't solve the interpolation problem unless we are willing to provide some more information about the function.

Usually, people assume that the true function is likely to be *smooth*, in some sense.

A statistical approach: Assume some probability distribution over functions, then pick the most likely function that matches the known values.

A more common appraoch: Assume the function is one from a restricted class, of which only one matches the known values.

Usually, we don't really believe the true function is from this class, but we hope that won't matter too much.

Polynomial Interpolation

If we know the value of a function at $n+1$ distinct points, we can interpolate by finding the polynomial of degree n or less that passes through these points.

There is exactly one such polynomial, so this problem is well defined.

Is this solution reasonable? Two contrasting facts:

- Any continuous function can be approximated to any accuracy (over a finite range) by a polynomial of some degree.
- The polynomial of degree n or less that interpolates $n+1$ values of a function may be much different from one that approximates the function well.

The Lagrange Interpolation Formula

Suppose that we do want to do polynomial interpolation. We need to figure out the polynomial of degree n (or less) that passes through the points

$$(t_0, x_0),\ (t_1, x_1),\ \ldots,\ (t_n, x_n)$$

We could solve a system of linear equations:

$$\begin{aligned}
a_n\, t_0^n \ + \ a_{n-1}\, t_0^{n-1} \ + \cdots + \ a_0 \ &= \ x_0 \\
a_n\, t_1^n \ + \ a_{n-1}\, t_1^{n-1} \ + \cdots + \ a_0 \ &= \ x_1 \\
&\vdots \\
a_n\, t_n^n \ + \ a_{n-1}\, t_n^{n-1} \ + \cdots + \ a_0 \ &= \ x_n
\end{aligned}$$

But it may be easier to use the Lagrange interpolation formula:

$$x(t) \ = \ \sum_{i=0}^{n} \ x_i \ \prod_{\substack{j=0 \\ j \neq i}}^{n} \ \frac{t - t_j}{t_i - t_j}$$

Does the Lagrange Formula Work?

Here's the Lagrange interpolation formula again:

$$x(t) \;=\; \sum_{i=0}^{n} \; x_i \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{t - t_j}{t_i - t_j}$$

Note first that this *is* a polynomial in t of degree n or less. Remember, the x_i and t_i are constants!

But does it pass through the given points?

For $n = 4$, consider the value at $t = t_3$. The terms with $i = 0, 1, 2, 4$ will be zero, since they include a factor of $(t_3 - t_3) = 0$. The term with $i = 3$ will be

$$x_3 \; \frac{t_3 - t_0}{t_3 - t_0} \, \frac{t_3 - t_1}{t_3 - t_1} \, \frac{t_3 - t_2}{t_3 - t_2} \, \frac{t_3 - t_4}{t_3 - t_4} \;=\; x_3$$

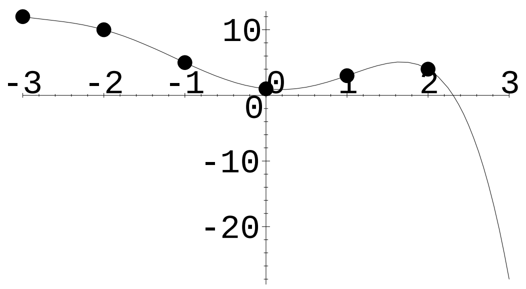
The value at $t = t_3$ will be x_3, which is right!

An Example in Maple

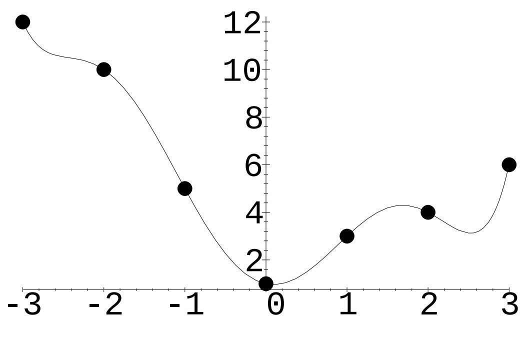
```
> ti:=[-3,-2,-1,0,1,2];
> xi:=[12,10,5,1,3,4];
> interp(ti,xi,t);
```

$$-\frac{13}{120}t^5 - 1/2\,t^4 + 3/8\,t^3 + 7/2\,t^2 - \frac{19}{15}t + 1$$

```
> with(plots):
> pts:=plot([seq([ti[i],xi[i]],i=1..6)],style=point):
> inter:=plot(interp(ti,xi,t),t=-3..3):
> display([pts,inter]);
```




```
> ti:=[ti[],3];
> xi:=[xi[],6];
> pts2:=plot([seq([ti[i],xi[i]],i=1..7)],style=point):
> inter2:=plot(interp(ti,xi,t),t=-3..3):
> display([pts2,inter2]);
```



Behaviour of Polynomial Interpolation

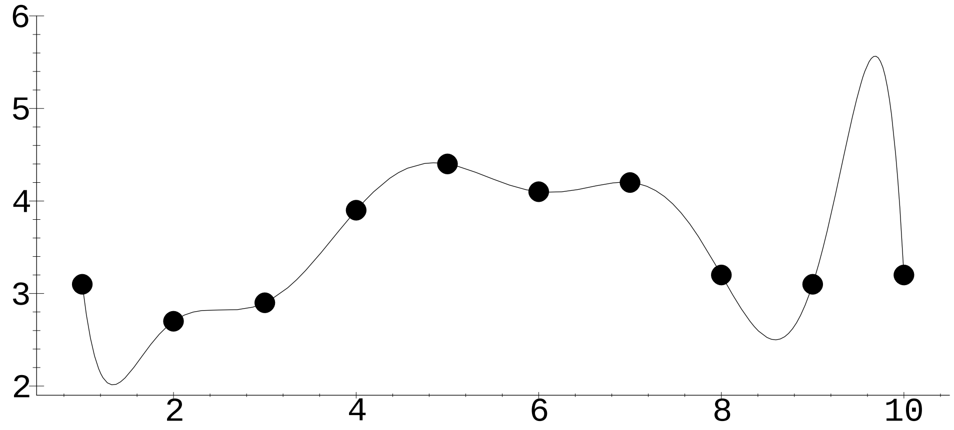
Polynomial interpolation has some properties that may often be undesirable:

- Adding or changing a data point can have effect on the whole interpolant — even far from the new point.
- Small changes in data points can have big effects on the interpolant.
- Trying to extrapolate with a polynomial can give very bad results.

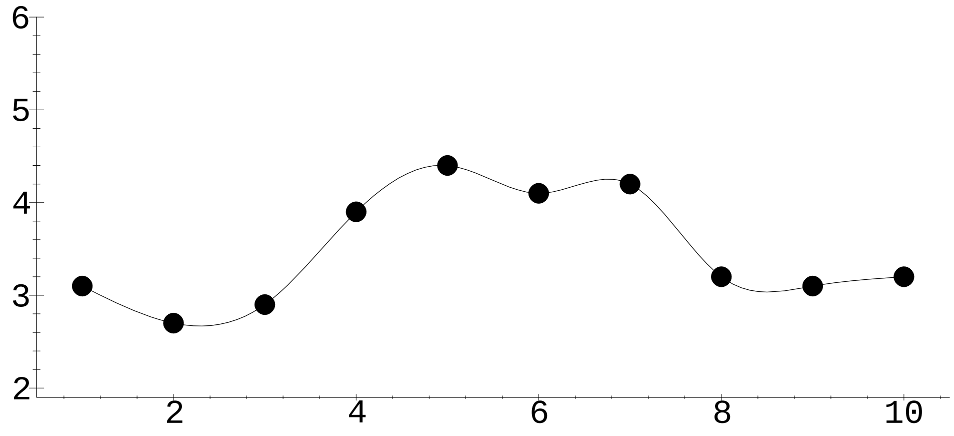
Some of these problems can be fixed by using *piecewise polynomial* interpolation.

Piecewise Polynomial Interpolation

A degree-9 polynomial interpolating 10 points:



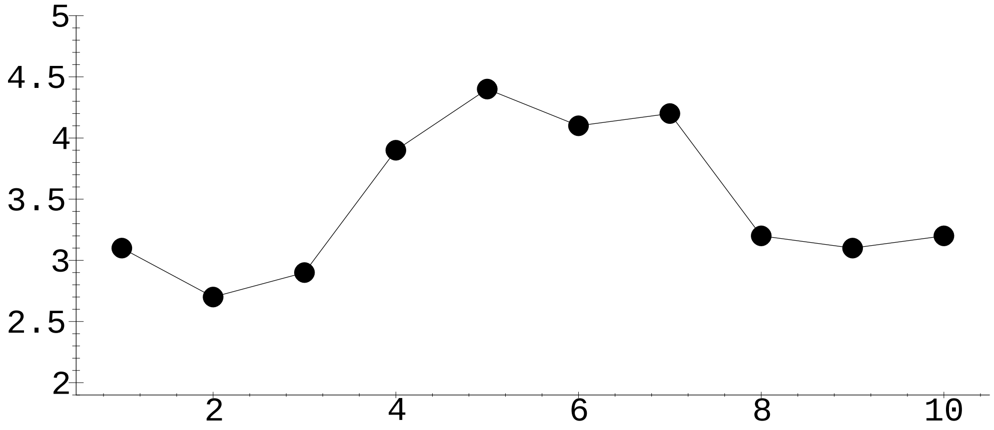
This isn't ideal. We can instead use several polynomials. Below is the piecewise cubic *natural spline* interpolant for the data above:



The points at which we switch from one polynomial to another are called *knots*. There is a knot at every data point above.

Linear Interpolation

The simplest interpolation scheme is *piecewise linear* interpolation — we just joint the data points by line segments:



This method has two problems:

- It's probably not very accurate.
- The interpolant has sharp corners.

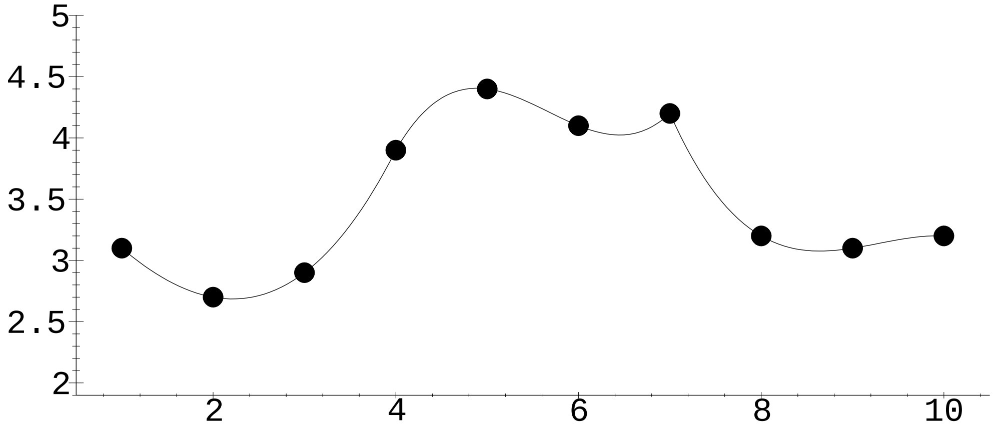
One the other hand, it has some advantages:

- Changing a data point changes only the two adjacent line segments.
- The interpolant never wanders far from the data points.

Piecewise Lagrange Interpolation

We might try to get a more accurate piecewise interpolant by using Lagrange interpolation with more than two points.

Here is a piecewise cubic interpolant, fit to groups of four points at a time:



Here, there are knots at the 4th and 7th data points.

We see a problem: The pieces don't always join up smoothly.

Making the Pieces Join Smoothly

To get pieces that join smoothly, we can stick with cubic polynomials, but fit each of them to just *two* data points. Each piece then has freedom to join smoothly with its neighbors.

How much freedom?

Each cubic polynomial has four parameters. Two are taken up by the requirement that it interpolate its two data points. That leaves two parameters for each piece that we can use to make the whole interpolant smooth.

We'll look at two such schemes for piecewise cubic interpolation: The *natural cubic splines* and the *Catmull-Rom* interpolant.

The Natural Cubic Splines

Suppose we interpolate $n+1$ data points using n cubic polynomials. Here's one way to use the $4n$ parameters available:

- Require each piece to interpolate its data points. That gives $2n$ constraints.
- Require adjacent pieces to have matching first derivatives. That's another $n-1$ constraints.
- Require adjacent pieces to have matching second derivatives as well, for another $n-1$ constraints.
- That leaves two more free parameters. We can require the second derivatives at the ends to be zero.

This scheme gives the *natural cubic splines*.

How Can We Find a Natural Spline?

How can we find the natural cubic spline that interpolates the data (t_0, x_0), (t_1, x_1), (t_2, x_2)?

The spline will consist of two pieces, say:

$$\begin{array}{ll} a_3t^3 + a_2t^2 + a_1t + a_0 & (t_0 \le t \le t_1) \\ b_3t^3 + b_2t^2 + b_1t + b_0 & (t_1 \le t \le t_2) \end{array}$$

We can find the a's and b's by solving the following system of 8 linear equations:

$$\begin{aligned} a_3t_0^3 + a_2t_0^2 + a_1t_0 + a_0 &= x_0 \\ a_3t_1^3 + a_2t_1^2 + a_1t_1 + a_0 &= x_1 \\ b_3t_1^3 + b_2t_1^2 + b_1t_1 + b_0 &= x_1 \\ b_3t_2^3 + b_2t_2^2 + b_1t_2 + b_0 &= x_2 \end{aligned}$$

$$3a_3t_1^2 + 2a_2t_1 + a_1 = 3b_3t_1^2 + 2b_2t_1 + b_1$$

$$6a_3t_1 + 2a_2 = 6b_3t_1 + 2b_2$$

$$\begin{aligned} 6a_3t_0 + 2a_2 &= 0 \\ 6c_3t_3 + 2c_2 &= 0 \end{aligned}$$

Properties of Natural Cubic Splines

- Finding the natural spline for a set of data points is possible in a reasonable amount of time, but it is a bit inconvenient.

- All the data points influence all the pieces of a natural spline, though nearby data points have more effect.

- The natural spline corresponds to the most probable function that passes through the data, according to a fairly reasonable probability distribution over functions.

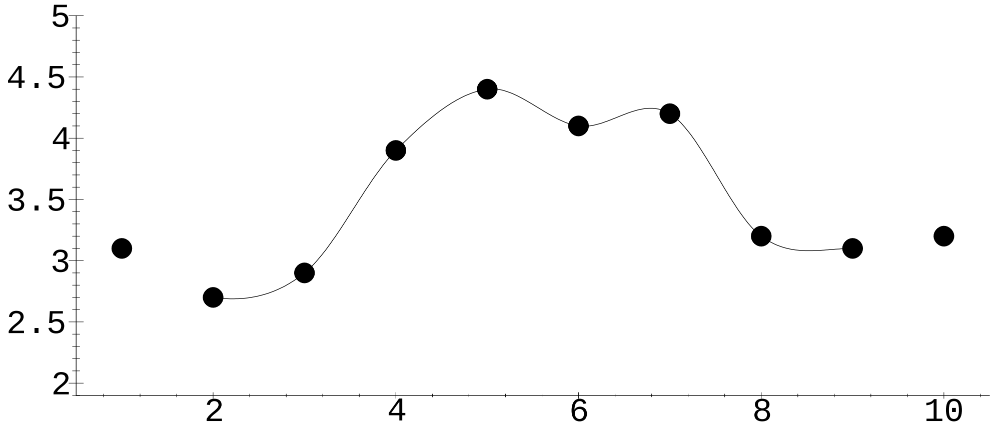
The Catmull-Rom Interpolant

We may prefer a piecewise cubic interpolant that is easier to compute than a natural spline. We may also want changes to a data point to change only the nearby pieces.

The *Catmull-Rom* interpolant has both these properties. In this scheme is specified by

- Requiring the ends of each piece to match the data points.
- Requiring the slope at the ends of each piece to match the slope determined by the two adjacent data points.

An example:



Polynomial Interpolation is Linear

Suppose that the polynomial

$$p(t) = a_n t^n + a_{n-1} t^{n-1} + \cdots + a_0$$

interpolates the data points

$$(t_0, x_0),\ (t_1, x_1),\ \ldots,\ (t_n, x_n)$$

and that the polynomial

$$q(t) = b_n t^n + b_{n-1} t^{n-1} + \cdots + b_0$$

interpolates the data points

$$(t_0, y_0),\ (t_1, y_1),\ \ldots,\ (t_n, y_n)$$

Then it is easy to see that the polynomial

$$p(t) + q(t) = (a_n + b_n) t^n + \cdots + (a_0 + b_0)$$

interpolates the data points

$$(t_0, x_0 + y_0),\ (t_1, x_1 + y_1),\ \ldots,\ (t_n, x_n + y_n)$$

Similarly, $cp(t)$ interpolates $(t_0, cx_0), \ldots, (t_n, cx_n)$. Polynomial interpolation is a *linear* operation.

Note: This use of the word "linear" is different from "linear interpolation" as interpolation by line segments.

Linearity of Piecewise Interpolation

Piecewise polynomial interpolation will also be linear, provided that it is based on linear functions of the data points and interpolants.

For example:

- Piecewise Lagrange interpolation is linear, since each piece is just a polynomial interpolant.

- Natural cubic splines are linear, since the derivatives they are based on are linear, ie:

  $$\frac{d}{dx}(f(x) + g(x)) = \frac{d}{dx}f(x) + \frac{d}{dx}g(x)$$

  Hence, if derivatives match for two splines, they will match for their sum.

- The Catmull-Rom interpolant is linear, since it is based on the data points and on the slopes defined by pairs of points, which are linear with respect to the function values.

Basis Functions for Linear Interpolants

If we are using a linear scheme, we can build an interpolating function from *basis functions* that interpolate just one non-zero point.

For a set of locations $t_0, t_1, \ldots, t_n$, we can find the following basis functions:

$$
\begin{aligned}
p_0(t) &\text{ interpolates } (t_0, 1),\ (t_1, 0),\ \ldots,\ (t_n, 0).\\
p_1(t) &\text{ interpolates } (t_0, 0),\ (t_1, 1),\ \ldots,\ (t_n, 0).\\
&\qquad \cdots\\
p_n(t) &\text{ interpolates } (t_0, 0),\ (t_1, 0),\ \ldots,\ (t_n, 1).
\end{aligned}
$$

Now, to interpolate the data points

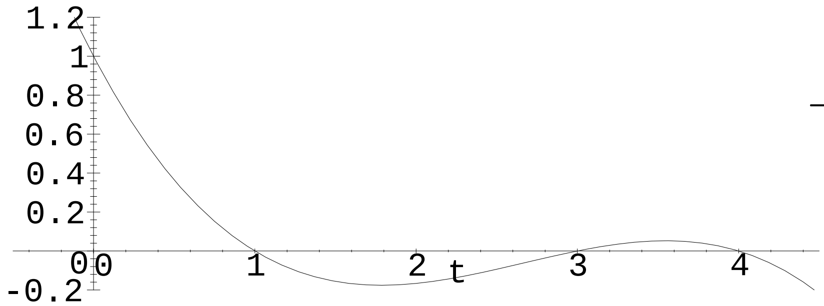
$$(t_0, x_0),\ (t_1, x_1),\ \ldots,\ (t_n, x_n)$$

we just compute

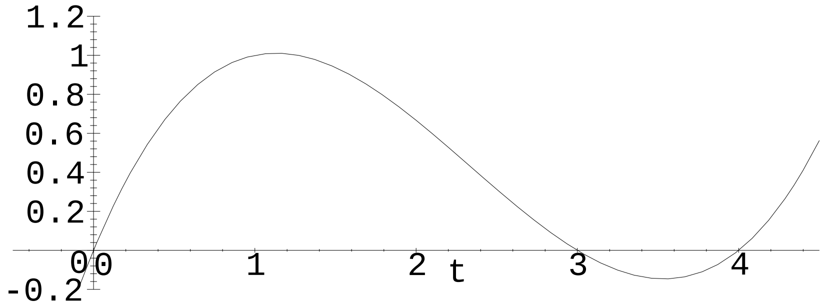
$$p(t) = x_0\, p_0(t) \ +\ x_1\, p_1(t) \ +\ \cdots\ +\ x_n\, p_n(t)$$

Lagrange Interpolation Basis Functions

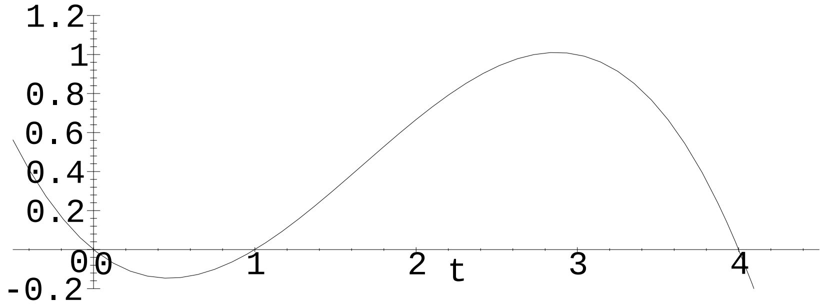
Here are the basis functions for Lagrange interpolation at $t_0 = 0,\ t_1 = 1,\ t_2 = 3,\ t_3 = 4$:



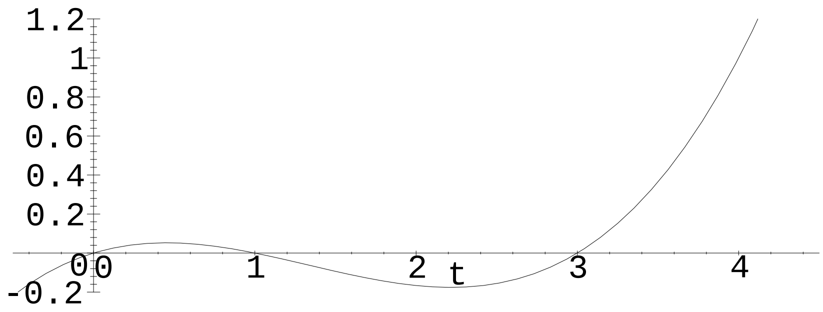
$$-\frac{1}{12}t^3+\frac{2}{3}t^2-\frac{19}{12}t+1$$



$$\frac{1}{6}t^3-\frac{7}{6}t^2+2t$$



$$-\frac{1}{6}t^3+\frac{5}{6}t^2-\frac{2}{3}t$$



$$\frac{1}{12}t^3-\frac{1}{3}t^2+\frac{1}{4}t$$