Theorem: Let $p(t)$ be the interpolating polynomial of degree $n-1$ such that $p(x_i) = f(x_i)$ for $i = 1, \ldots, n$. Assume the x_i's are different, also that f has n continuous derivatives. Then

$$f(t) = \underbrace{p(t)}_{\text{interpolating polynomial}} + \frac{q(t)}{n!} f^{(n)}(\xi) \leftarrow E(t)$$

where $q(t)$ is another polynomial

$$q(t) = (t - x_1)(t - x_2) \cdots (t - x_n)$$

and ξ is between $\min(t, x_1, x_2, \ldots, x_n)$ and $\max(t, x_1, x_2, \ldots, x_n)$.

Let $E(t) = f(t) - p(t)$

Define, $F(t) = E(t) - \alpha q(t)$

Want to choose α so that $F(t)$ has one more zero at $t = t_*$. Here t_* is any point of interest other than $x_1, x_2, \ldots$ or x_n.

Want $F(t_*) = 0$ means $E(t_*) - \alpha q(t_*) = 0$

or that $\alpha = \dfrac{E(t_*)}{q(t_*)}$

With this choice of α the function $F(t)$ has zeros at $x_1, x_2, \ldots, x_n$ and also t_*.

By Rolle's theorem, since

$F(t)$	has	$n+1$	zeros
$F'(t)$	has	n	zeros
$F''(t)$	has	$n-1$	zeros
$\vdots$			
$F^{(n)}(t)$	has	1	zero

not the zeros of $F'(t)$ are between $\min(x_1,\dots,x_n,t_*)$ and $\max(x_1,\dots,x_n,t_*)$.

In particular this is a point ξ such that

$F^{(n)}(\xi)=0$, where ξ is between $\min(x_1,\dots,x_n,t_*)$ and $\max(x_1,\dots,x_n,t_*)$.

What does this mean?

$$F(t)=E(t)-\alpha q(t)$$

$$F^{(n)}(t)=E^{(n)}(t)-\alpha q^{(n)}(t)$$

$\alpha=\dfrac{E(t_*)}{q(t_*)}$

Since

$$q(t)=(t-x_1)(t-x_2)\cdots(t-x_n)$$

then

$$q^{(n)}(t)=n!$$

Also

$$E(t)=f(t)-p(t)$$

Therefor

$$E^{(n)}(t)=f^{(n)}(t)-p^{(n)}(t)=f^{(n)}(t)$$

since $p(t)$ is a polynomial of degree $n-1$

Substituting gives

$$F^{(n)}(t) = f^{(n)}(t) - \frac{E(t_*)}{q(t_*)}\, n!$$

Setting $t = \xi$ obtains

solve for $E(t_*)$

$$F^{(n)}(\xi) = f^{(n)}(\xi) - \frac{E(t_*)}{q(t_*)}\, n! = 0$$

Therefore

$$E(t_*) = \frac{q(t_*)}{n!}\, f^{(n)}(\xi)$$

at the point of interest...

In the theorem I was trying to show

$$E(t) = \frac{q(t)}{n!}\, f^{(n)}(\xi)$$

a function of

This is exactly the same thing except there is a different "dummy" variable used in the equation...

Reflect: Two ways of approximating a function $f(x)$ using a polynomial.

1. Taylor polynomial
2. Interpolating polynomial.

Could combine the ideas of both together...

Theorem: Let $p(t)$ be the interpolating polynomial of degree $n-1$ such that $p(x_i) = f(x_i)$ for $i = 1, \ldots, n$. Assume the x_i's are different, also that f has n continuous derivatives. Then

$$f(t) = \underbrace{p(t)}_{\text{interpolating polynomial}} + \frac{q(t)}{n!} f^{(n)}(\xi)$$

where $q(t)$ is another polynomial

$$q(t) = (t - x_1)(t - x_2) \cdots (t - x_n)$$

and ξ is between $\min(t, x_1, x_2, \ldots, x_n)$ and $\max(t, x_1, x_2, \ldots, x_n)$.

Corollary: The interpolating polynomial of degree $n-1$ passing through the points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ is unique...

Why? Suppose you have two interpolating polynomials $p_1(x)$ and $p_2(x)$...

To use the theorem let $f(x) = p_1(x)$ and $p(x) = p_2(x)$ Then

$$f(t) = \underbrace{p(t)}_{\text{interpolating polynomial}} + \frac{q(t)}{n!} f^{(n)}(\xi)$$

where another po

Plugging in p_1 and p_2 gives

$$p_1(t) = p_2(t) + \frac{q(t)}{n!} \underbrace{p_1^{(n)}}_{\to 0}(\xi)$$

since p_1 is a polynomial of degree $n-1$

It follow that

$$p_1(t) = p_2(t)$$

which means the interpolating polynomial is unique...

Only thing left is to show interpolating polynomials really exist...

Lagrange basis functions

Consider n points

$$(x_1, y_1), (x_2, y_2) \ldots, (x_n, y_n)$$

Define

$$l_k(t) = \prod_{j \neq k} \frac{t - x_j}{x_k - x_j}$$

Example

$n = 3 \quad k = 2$

$$l_2(t) = \prod_{j \neq 2} \frac{t - x_j}{x_2 - x_j} = \left(\frac{t - x_1}{x_2 - x_1}\right)\left(\frac{t - x_3}{x_2 - x_3}\right)$$

Then the interpolating polynomial of degree $n-1$ passing through the points is

$$p(t) = \sum_{k=1}^{n} y_k l_k(t)$$

Let's check that $p(t)$ is really the interpolating polynomial of degree $n-1$ passing through the points

$$(x_1, y_1),\ (x_2, y_2),\ \ldots,\ (x_n, y_n)$$

numerically using Julia.

First note that

$$l_k(t) = \prod_{j \neq k} \frac{t - x_j}{x_k - x_j}$$

for each k one of the terms in the product are left out. Thus, $l_k(t)$ is a polynomial of degree $n-1$ for each k and therefore the sum $p(t)$ also has degree $n-1$

Let's code $l_k(t)$ in Julia using a loop to make the product. It might also be possible to create the same product using built-in vector functions, but since Julia is just-in-time compiled, then loops run fast and there is no need to be clever with vector functions unless one wants to be.

```
julia> function l(k,t)
           r=1
           for j=1:n
               if j!=k
                   r=r*(t-x[j])/(x[k]-x[j])
               end
           end
           return r
       end
l (generic function with 1 method)
```

This if statement leaves out the term when $j=k$ from the product

x is a global array we need to define...

random values for y_i

```
julia> x=[1,2,3,5,6,7]
6-element Vector{Int64}:
 1
 2
 3
 5
 6
 7
```

we chose 6 points so $n=6$

```
julia> y=rand(6)
6-element Vector{Float64}:
 0.8413339735417158
 0.5657361231467164
 0.40460908183600086
 0.6315139489670885
 0.9947259371102335
 0.46531504264051704
```

These will be different for everyone else

Now define the sum for

$$p(t) = \sum_{k=1}^{n} y_k l_k(t)$$

Again use a loop, just because it is easy and runs fast using the just-in-time compiler in Julia.

```
julia> function p(t)
           s=0
           for k=1:n
               s=s+y[k]*l(k,t)
           end
           return s
       end
p (generic function with 1 method)
```

this could be written as `s+=y[k]*l(k,t)` as in C or C++.

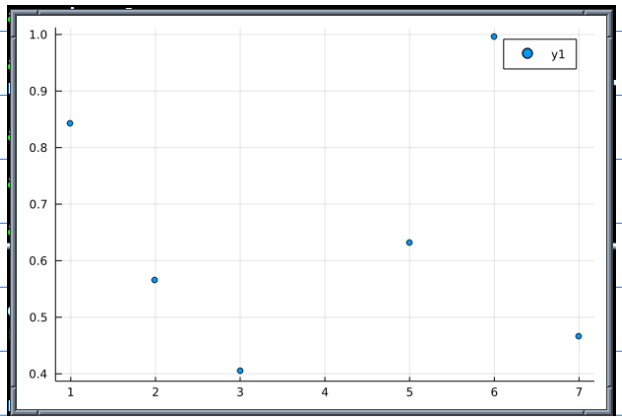
Finally, let's plot our polynomial to see if it passes through the points...

```
julia> using Plots
julia> scatter(x,y)
```

load plot library

plot the points

The plot should look something like this, except the random y-coordinates will be in different places



The first semicolon expands the iterator into a vector

The second semicolon suppresses the out put so that huge vector doesn't get printed to the screen.

```
julia> xs=[1:0.1:7;];
julia> n=6
6
```

← almost forgot to set n.

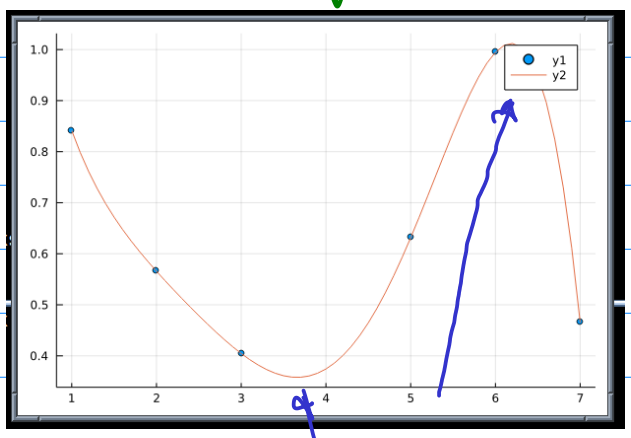
note that n could also be set with `n= size(x)[1]`

The ! mark tells Julia to overlay the previous graph with the new one.

```
julia> plot!(xs,p.(xs))
```

The dot means apply p pointwise to each of the elements in xs.

The graph should look like



If the polynomial doesn't pass through the points, then something went wrong and should be fixed...

polynomial goes lower than any of the points as well as higher...

Note that high-degree polynomial interpolants tend to wiggle more than one would expect from the data... that's because the q(t) part of the error

$$f(t) = \underbrace{p(t)}_{\text{interpolating polynomial}} + \frac{q(t)}{n!} f^{(n)}(\xi) \quad \leftarrow E(t)$$

where another polynomial

given by

$$q(t) = (t-x_1)(t-x_2)\cdots(t-x_n)$$

Can also be quite wiggley. We'll discuss ways to minimize this problem later.

Follows are some more examples of graphs of interpolating polynomials through random points for comparison ...