# Nonlinear Regression

Recal <u>linear</u> regression:

fit $y(x) = a_1 f_1(x) + a_2 f_2(x) + \dots + a_n f_n(x)$

to m data points.

$$\begin{bmatrix} f_1(x_1) & f_2(x_1) & \cdots & f_n(x_1) \\ f_2(x_1) & & & \\ \vdots & & & \\ f_n(x_1) & - & - & f_n(x_m) \end{bmatrix} \begin{bmatrix} a_1 \\ \vdots \\ a_n \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix}$$

Now look at using a non-linear function, such as

$$y(x) = \frac{a_1 x}{a_2 + x} \rightarrow \underline{a} = \begin{bmatrix} a_1 \\ a_2 \end{bmatrix}$$

Not possible to make a linear system to solve.

Use the Gauss-Newton Algorithm.

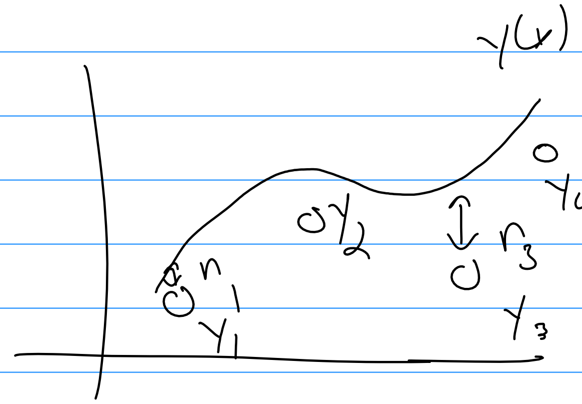
Let a set of m data points $(x_i, y_i)$ be given and we want to fit a function $y(x, \underline{a})$, where $\underline{a}$ is the vector of unknown coefficients.

Define the residual at the m points as

$$r_i = y_i - y(x_i, \underline{a})$$

We want to minimize the objective function

$$S(\underline{a}) = \sum_{i=1}^{m} r_i^2$$




If we used a Newton method then:

$$\underline{a}_{k+1} = \underline{a}_k - \underline{H}_k^{-1} \underline{g}_k$$

$\underline{g}$ = gradient of S w.r.t $\underline{a}$,

$$g_j = \frac{\partial S}{\partial a_j}$$

$\underline{H}$ = Hessian of S, $H_{jk} = \dfrac{\partial g_j}{\partial a_k}$

Since $S = \sum_{i=1}^{m} r_i^2$

$$\Rightarrow g_j = 2 \sum_{i=1}^{m} r_i \frac{\partial r_i}{\partial a_j}$$

and

$$H_{jk} = 2\sum_{i=1}^{m} \left( \frac{\partial r_i}{\partial a_j} \frac{\partial r_i}{\partial a_k} + r_i \frac{\partial^2 r_i}{\partial a_j \partial a_k} \right)$$

(Gauss-Newton ignores the $r_i \frac{\partial^2 r_i}{\partial a_j \partial a_k}$ part.

Reason is that $2^{nd}$ derivatives are ~~noisy~~.

$$H_{jk} \simeq 2\sum_{i=1}^{m} \frac{\partial r_i}{\partial a_j} \frac{\partial r_i}{\partial a_k} = 2\sum_{i=1} J_{ij} J_{ik}$$

$J_{ij} = \frac{\partial r_i}{\partial a_j}$ is the Jacobian of $\underline{r}$ w.r.t. $\underline{a}$.

Now, notice that

$$\underline{J}^T \underline{r} = \sum_{i=1}^{m} \frac{\partial r_i}{\partial a_j} r_i$$

$$\underline{J}^T \underline{J} = \sum_{i=1}^{m} J_{ij} J_{jk}$$

$$\Rightarrow \underline{g} = 2\underline{J}^T \underline{r} \quad \& \quad \underline{H} \simeq 2\underline{J}^T \underline{J}$$

$$\Rightarrow \underline{a}_{k+1} = \underline{a}_k - \underline{H}_k^{-1} \underline{g}_k$$

$$= \underline{a}_k - \frac{1}{2}\left(\underline{J}_k^T \underline{J}_k\right)^{-1} \left(2\underline{J}_k^T \underline{r}_k\right)$$

$$\Rightarrow \underline{a}_{k+1} = \underline{a}_k - (\underline{J}_k^T \underline{J}_k)^{-1} \underline{J}_k^T \underline{r}_k$$

Typically this is combined w/ a line search!

$$\underline{\Delta}_k = -(\underline{J}_k^T \underline{J}_k)^{-1} \underline{J}_k^T \underline{r}_k$$

then $\underline{a}_{k+1} = \underline{a}_k + \alpha_k \underline{\Delta}_k$

for an $\alpha_k$ such that $S(\underline{a}_{k+1}) < S(\underline{a}_k)$

Advantage of this method: Does not need $2^{nd}$ - derivatives of $\underline{r}$

Disadvantage: It might not converge

For convergence you need the approximate $\underline{H}$ to be close to the true one.

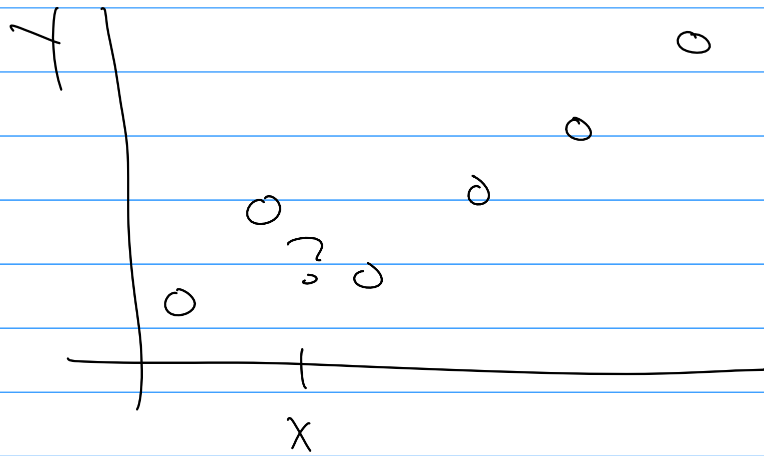
$$\Rightarrow \left| r_i \frac{\partial^2 r_i}{\partial a_j \partial a_k} \right| \ll \left| \frac{\partial r_i}{\partial a_j} \frac{\partial r_i}{\partial a_k} \right|$$

This typically happens if $r_i$ is already small or if $y(x, \underline{a})$ is only $\underbrace{\text{mildly non-linear}}$,

Derivative higher than first are small.

# Interpolation

Obtaining values between known data points.




Methods:

① Polynomial (Global) interpolation

② Spline (Piecewise) interpolation

---

## ① Polynomial Interpolation

Consider $n$- data points:

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

Construct an $n-1$ order polynomial:

$$f(x) = \underbrace{a_0 + a_1 x + \ldots + a_{n-1}}_{\text{unknown}} x^{n-1}$$

If you restrict $f(x)$ to
$f(x_1)=y_1,\ f(x_2)=y_2,\ \ldots$ then
you get:

$$\underbrace{\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^{n-1} \\ 1 & x_2 & \cdots & & x_2^{n-1} \\ \vdots & & \ddots & & \\ \vdots & & & \ddots & \\ 1 & - & - & - & x_n^{n-1} \end{bmatrix}}_{\text{Called the Vandermonde Matrix}} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$$

The Vandermonde matrix is <u>very</u> ill-conditioned.

ex) Consider $(300, 1)$, $(400, 1.1)$, $(500, 1.3)$

Create $y(x) = a_0 + a_1 x + a_2 x^2$

$$\underbrace{\begin{bmatrix} 1 & 300 & 90000 \\ 1 & 400 & 160000 \\ 1 & 500 & 250000 \end{bmatrix}}_{\text{Condition \# of } \sim 5.9\times10^6 \,!} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1.1 \\ 1.3 \end{bmatrix}$$

But, there is only <u>one</u> interpolating polynomial, but there are many methods to obtain that function.

- Lagrange form of Polynomial Interpolation

Instead of $f(x) = a_0 + a_1 x + \cdots + a_{n-1} x^{n-1}$

Try $f(x) = y_1 L_1(x) + y_2 L_2(x) + \cdots + y_n L_n(x)$

$y_1, y_2, \ldots, y_n$ are known

$L_i(x)$ are the Lagrange Polynomials

Start w/ 2 points: $(x_1, y_1)$ & $(x_2, y_2)$

$\Rightarrow$ A linear line

$\Rightarrow f(x) = y_1 L_1(x) + y_2 L_2(x)$

We need $f(x_1) = y_1 \Rightarrow L_1(x_1) = 1, \; L_2(x_1) = 0$

$f(x_2) = y_2 \Rightarrow L_1(x_2) = 0, \; L_2(x_2) = 1$

$\Rightarrow$ Use $L_1(x) = \dfrac{x - x_2}{x_1 - x_2}$ $\qquad L_2(x) = \dfrac{x - x_1}{x_2 - x_1}$

Notice that $L_1(x) + L_2(x) = 1$

w/ 3- points

$$f(x) = y_1 L_1(x) + y_2 L_2(x) + y_3 L_3(x)$$

$$L_1(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)}$$

$$L_2(x) = \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)}$$

$$L_3(x) = \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)}$$

In general, n- points will hav

$$y(x) = \sum_{i=1}^{n} y_i L_i(x)$$

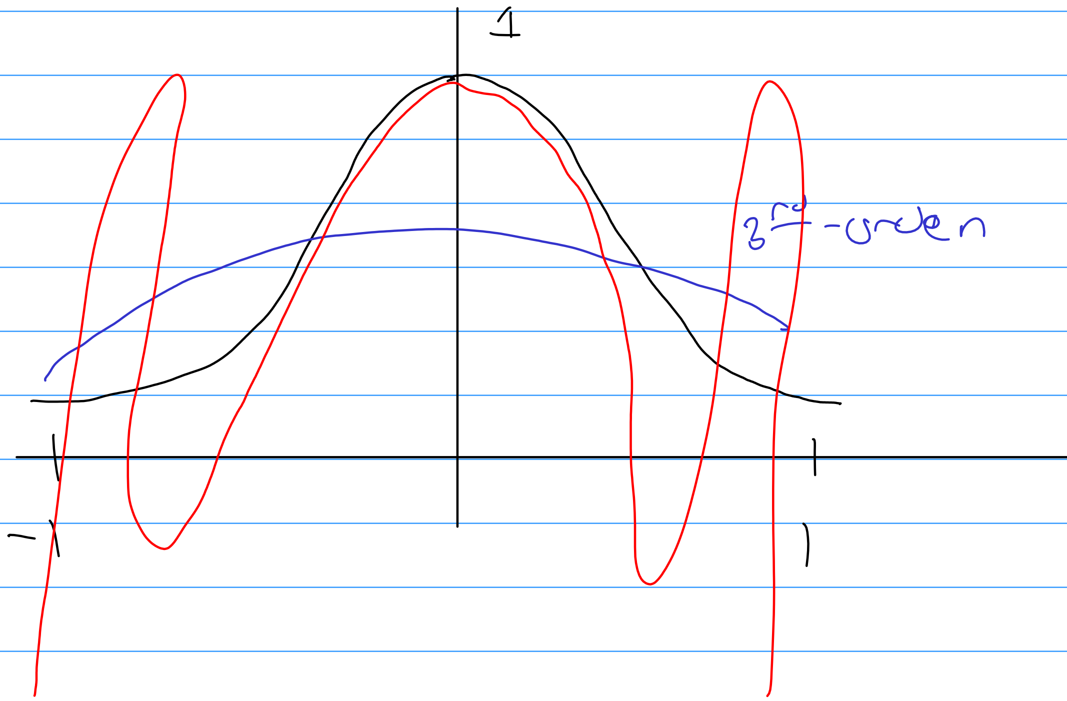
with

$$L_i(x) = \prod_{\substack{j=1 \\ i \neq j}}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

Issue with polynomial interpolation in general: Runge's Phenomenon

Consider interpolating $\frac{1}{1+25x^2}$ over $x \in [-1,1]$

Using uniform grid spacing.



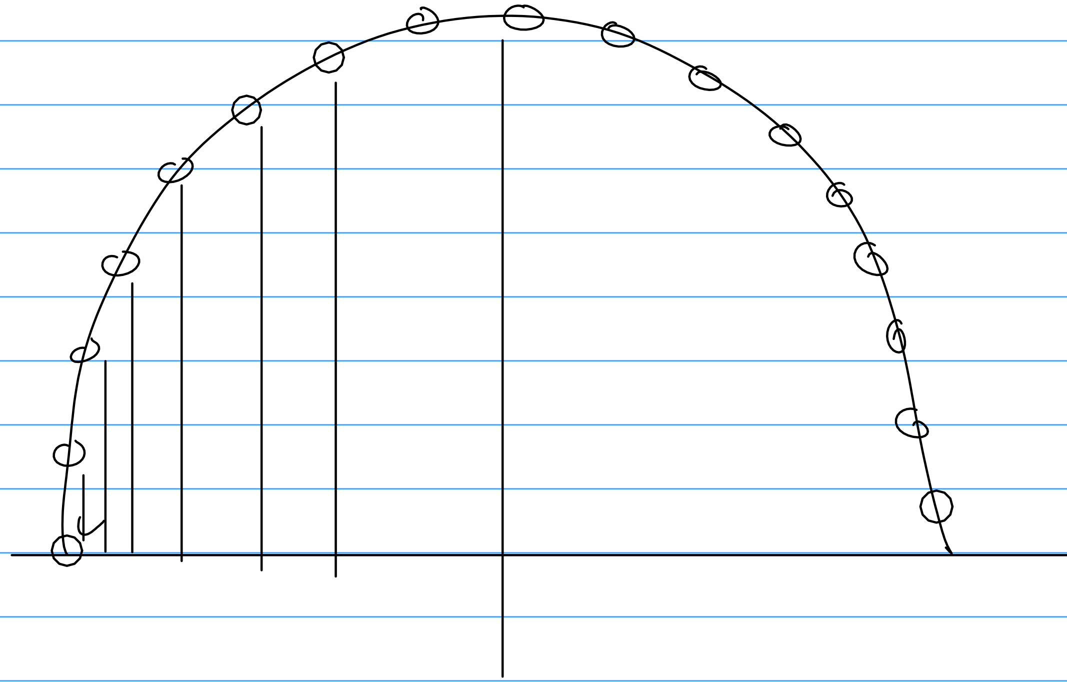

A solution: Don't use uniform points.

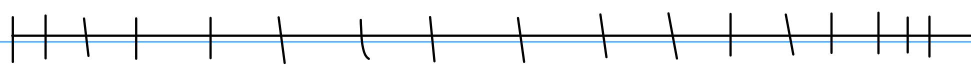
Use non-uniform points.

e.g. Chebyshev Nodes

for $x \in [-1,1]$, $x_i = \cos\left(\frac{2i-1}{2n}\pi\right)$

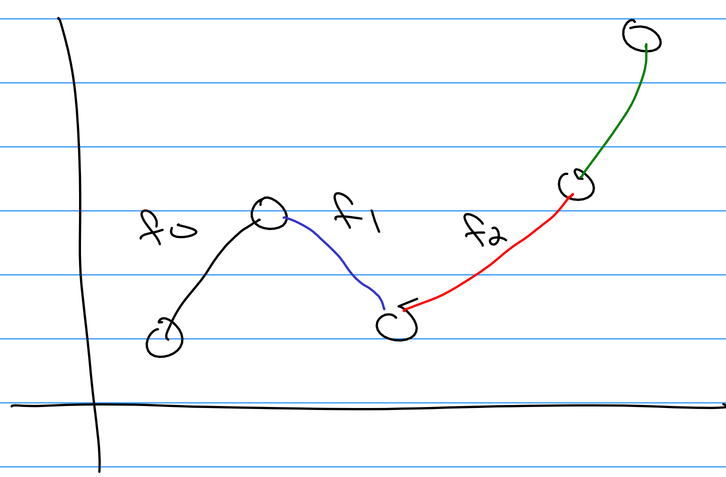
$i = 1, \dots, n$





## ② Spline (Piecewise interpolation)

Instead of trying to interpolate the entire data set w/ a single function, interpolate it with many, piecewise functions.



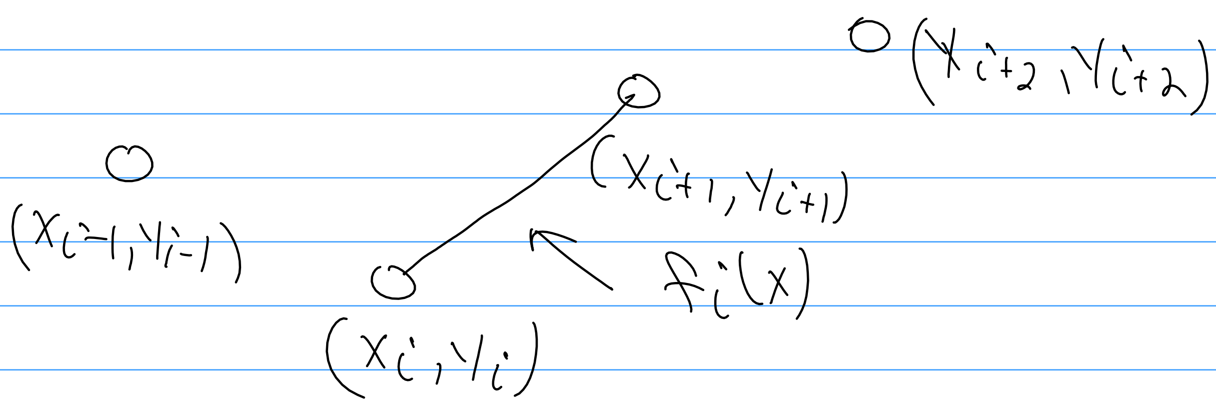
Let the data be $(x_1, y_1), \ldots, (x_n, y_n)$

In the interval $[x_i, x_{i+1}]$ for $i = 1, \ldots, n-1$

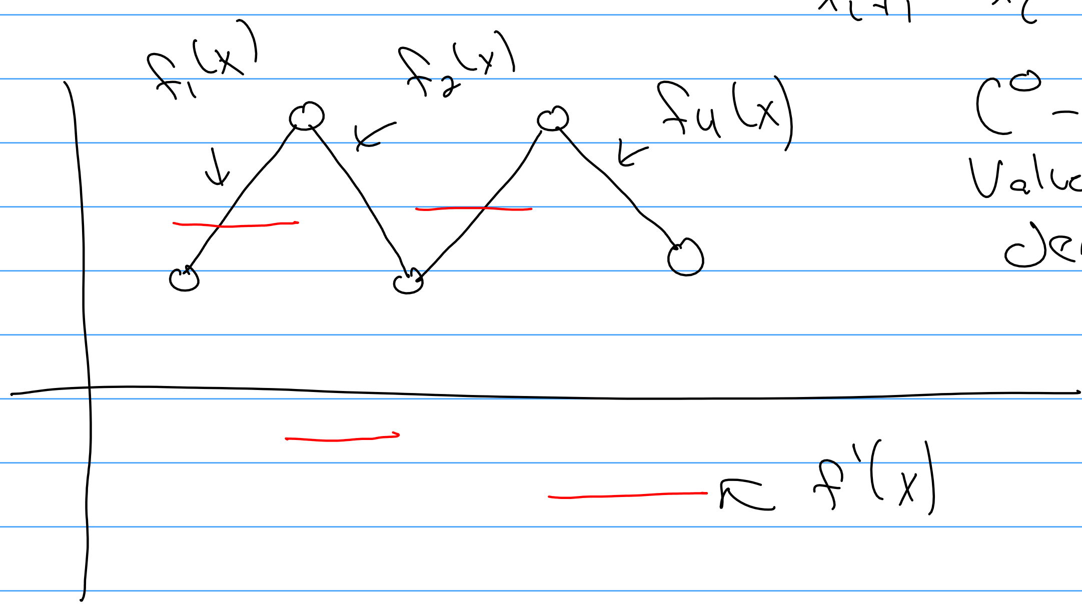
Let $f_i(x) = a_i + b_i(x - x_i)$

A linear spline

We need $f_i(x_i) = y_i$ & $f_i(x_{i+1}) = y_{i+1}$




$\Rightarrow a_i = y_i$ & $b_i = \dfrac{y_{i+1} - y_i}{x_{i+1} - x_i}$




$C^0$ - Continuous

Values match, derivatives do not.

The points at which adjacent splines meet are called "knots"

If 2 adjacent splines have the same function value at a knot $\Rightarrow$ $C^0$ continuous.

To take derivatives, you need:

$C^1$ - Continuous (1st derivatives & values match)

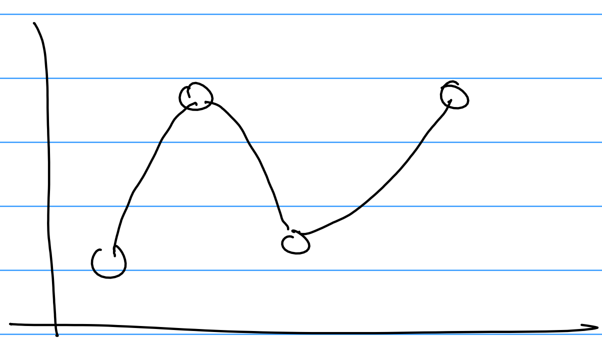
$C^2$ - Cont (value, 1st & 2nd derivatives)

Most common spline: Cubic Splines

$\rightarrow$ Let $f_i(x) = a_i + b_i(x-x_i) + c_i(x-x_i)^2 + d_i(x-x_i)^3$

valid in $x \in [x_i, x_{i+1}]$

Also, define $h_i = x_{i+1} - x_i$



4 points $\rightarrow$ 3 splines
4 unknowns per spline

$\Rightarrow$ 12 unknowns in total

In general, for $n$- data points there are

$4(n-1)$ unknowns for cubic splines

$\Rightarrow$ Need $4(n-1)$ conditions

#1 we need $f_i(x_i) = y_i$ for $i = 1, \dots, n-1$

$\Rightarrow a_i = y_i$

#2 Need $C^0$ Continuous

$$f_i(x_{i+1}) = f_{i+1}(x_{i+1}) \qquad i = 1, \dots, n-1$$

$$a_i + b_i h_i + c_i h_i^2 + d_i h_i^3 = a_{i+1} = y_{i+1}$$

$$f_i(x_{i+1}) = a_i + b_i \overbrace{(x_{i+1} - x_i)}^{h_i} + c_i \overbrace{(x_{i+1} - x_i)^2}^{h_i^2} + d_i \overbrace{(x_{i+1} - x_i)^3}^{h_i^3}$$

$$f_{i+1}(x_{i+1}) = a_{i+1} + b_{i+1}(x_{i+1} - x_{i+1}) + c_{i+1}(x_{i+1} - x_{i+1})^2 + d_{i+1}(x_{i+1} - x_{i+1})^3$$

#3 $C^1$ - Continuous First Derivative

$$f_i'(x_{i+1}) = f_{i+1}'(x_{i+1}) \quad i=1, \ldots, n-2$$

$$f_i'(x) = b_i + 2c_i(x-x_i) + 3d_i(x-x_i)^2$$

$$\Rightarrow b_i + 2c_ih_i + 3d_ih_i^2 = b_{i+1}$$

#4 $C^2$- Continuous

$$f_i''(x_{i+1}) = f_{i+1}''(x_{i+1}) \quad i=1, \ldots, n-2$$

$$f_i''(x) = 2c_i + 6d_i(x-x_i)$$

$$\Rightarrow c_i + 3d_ih_i = c_{i+1}$$

Thus we have $2(n-1) + 2(n-2)$ equations

Still 2 short

Common Options

1) Natural $f_1''(x_1) = 0$ & $f_{n-1}''(x_n) = 0$

2) Clamped $f_1'(x_1) = A_1$ & $f_{n-1}'(x_n) = A_2$

$A_1$ & $A_2$ are known

3) "not-a-knot" $f_1'''(x_2) = f_2'''(x_2)$

$f_{n-2}'''(x_{n-1}) = f_{n-1}'''(x_{n-1})$

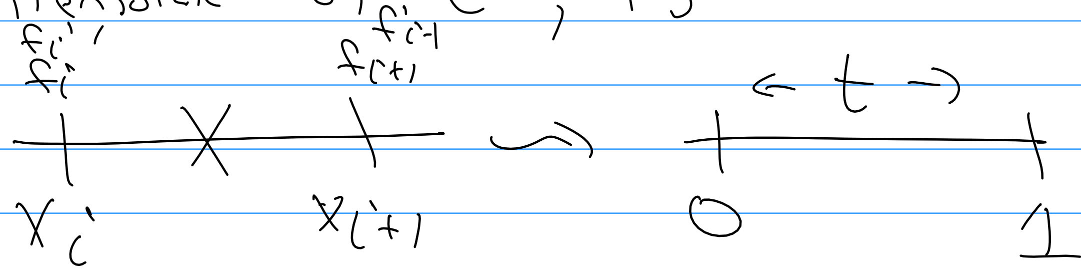
Note: This does result in a $4(n-1)$ by $4(n-1)$ linear system to solve.

---

Aside: A closely related topic is piecewise interpolation using "weight" functions.

ex1) Cubic Hermite Spline Interpolation

Interpolate on $[0, 1]$




$$t = \frac{x - x_i}{x_{i+1} - x_i} = \frac{x - x_i}{h_i}$$

In Hermite interpolation match values & first derivatives.

$\Rightarrow$ let $f_i$, $f'_i$, $f_{i+1}$ & $f'_{i+1}$ be known

$$\Rightarrow p(t) = (2t^3 - 3t^2 + 1) f_i + (t^3 - 2t^2 + t) h_i f'_i + \underbrace{(-2t^3 + 3t^2)}_{\text{The "weights"}} f_{i+1} + (t^3 - t^2) h_i f'_{i+1}$$

Value at $x \in (x_i, x_{i+1})$ is

$$p\left(\frac{x - x_i}{h_i}\right)$$

Derivative at $x \in (x_i, x_{i+1}]$ is

$$\frac{1}{h_i} p'\left(\frac{x - x_i}{h_i}\right)$$