Math 471 Numerical Methods

Chapter 5. Interpolation

Overlap §5.1, 5.3, 5.5, 5.6

Basic question for interpolation: given data set $(x_0, y_0), (x_1, y_1), ..., (x_n, y_n)$, how to find a interpolation $p(x)$, which is of a "understandable" form for computer, such that

$$p(x_i) = y_i, \quad \text{for } i = 0, 1, ..., n$$

Alternatively, we may ask: how to approximate a given function $f(x)$ by a computationally feasible function $p(x)$ such that

$$p(x_i) = f(x_i), \quad \text{for } i = 0, 1, ..., n$$

Polynomials will be use in this chapter for interpolation since it is very easy for a computer to store and evaluate. More generally, one may use other common forms of functions to interpolate/approximate data. For instance, trigonometric interpolation.

• Example. Find the quadratic polynomial that interpolates data set $(-1, 0), (1, -2), (3, 4)$.

Solution. Let $p(x) = ax^2 + bx + c$ and plug in the data set

$$\begin{aligned} (-1, 0) &\implies 0 = p(-1) = a - b + c \\ (1, -2) &\implies -2 = p(1) = a + b + c \\ (3, 4) &\implies 4 = p(3) = 9a + 3b + c = 4 \end{aligned}$$

Solve this linear system for a, b, c,

$$a = 1, \quad b = -1, \quad c = 2$$

So the answer is

$$p(x) = x^2 - x - 2.$$

The following theorem confirms the existence of such polynomial.

Theorem 1 *Given $n+1$ pairs of data $\{(x_i, y_i)\}_{i=0}^n$ with x_i mutually distinct, there exists a unique polynomial of degree n or less*

$$p(x) = a_0 + a_1 x + a_2 x^2 + ... + a_n x^n$$

such that

$$p(x_i) = y_i, \quad \textit{for } i = 0, 1, ..., n$$

Note. The degree of $p(x)$ equals the number of data pairs minus 1. But it is possible that $a_n = 0$ so that $p(x)$ is effectively of degree less than n. In such case, $p(x)$ is still a *degenerated* n-th degree polynomial.

Proof. Among many proofs for this theorem, we use one based on Linear Algebra.

Plug the data set (x_i, y_i) into the definition of $p(x)$,

$$y_i = f(x_i) = a_0 + a_1 x_i + a_2 x_i^2 + ... + a_n x_i^n, \quad \text{for } i = 0, 1, ...,$$

Considering $a_0, a_1, ..., a_n$ as the unknowns, the above $n+1$ equations can be written as an $(n+1)$-by-$(n+1)$ linear system

$$\begin{cases} a_0 + x_0 \cdot a_1 + x_0^2 \cdot a_2 + x_0^3 \cdot a_3 + ... + x_0^n \cdot a_n = y_0 \\ a_0 + x_1 \cdot a_1 + x_1^2 \cdot a_2 + x_1^3 \cdot a_3 + ... + x_1^n \cdot a_n = y_1 \\ a_0 + x_2 \cdot a_1 + x_2^2 \cdot a_2 + x_2^3 \cdot a_3 + ... + x_2^n \cdot a_n = y_2 \\ \qquad\qquad\qquad\qquad ... \quad ... \quad ... \quad ..... \\ a_0 + x_n \cdot a_1 + x_n^2 \cdot a_2 + x_n^3 \cdot a_3 + ... + x_n^n \cdot a_n = y_n \end{cases}$$

i.e.

$$X\vec{a} = \vec{y} \tag{1}$$

where

$$X = \begin{pmatrix} 1 & x_0 & x_0^2 & x_0^3 & ... & x_0^n \\ 1 & x_1 & x_1^2 & x_1^3 & ... & x_1^n \\ 1 & x_2 & x_2^2 & x_2^3 & ... & x_2^n \\ .. & ... & ... & ... & ... & ... \\ 1 & x_n & x_n^2 & x_n^3 & ... & x_n^n \end{pmatrix} \quad \text{and } \vec{a} = \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix}, \quad \vec{y} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_0 \end{pmatrix}.$$

By Linear Algebra, the above system admits a unique solution if and only if $\det(X) \neq 0$. And it is a known fact that

$$\det(X) = (x_0 - x_1)(x_0 - x_2)...(x_0 - x_n) \cdot (x_1 - x_2)(x_1 - x_3)...(x_1 - x_n)... = \prod_{i<j}(x_i - x_j).$$

Apparently $\det(X) \neq 0$ since all x_i's are mutually distinct.

DONE

§ 5.1 Lagrange form.

Theorem 1 is of theoretical interest but is rarely used in practice to find the interpolation polynomial. The reason is that directly solving system (1) requires $O(n^3)$ operations since A is a dense matrix. Algorithms of $O(n^2)$ operation count have been therefore introduced. Two common methods are called the Lagrange form and the Newton form.

The Lagrange form is based on the so called Lagrange polynomial defined as

$$l_k(x) = \frac{(x-x_0)(x-x_1)...(x-x_{k-1})(x-x_{k+1})...(x-x_n)}{(x_k-x_0)(x_k-x_1)...(x_k-x_{k-1})(x_k-x_{k+1})...(x_k-x_n)} = \frac{\prod_{i\neq k}(x-x_i)}{\prod_{i\neq k}(x_k-x_i)}$$

For fixed n, there are $n+1$ Lagrange polynomials satisfying

$$l_k(x_i) = \begin{cases} 0 & \text{if } i \neq k \\ 1 & \text{if } i = k \end{cases} = \delta_{ik}$$

So each $l_k(x)$ "stands out" at $x = x_k$ and vanishes at all other x_i's. Then, by taking the linear combination of them

$$p(x) = y_0 l_0(x) + y_1 l_1(x) + y_2 l_2(x) + ... + y_n l_n(x) = \sum_{i=0}^{n} y_i l_i(x).$$

we find the desired interpolation polynomial.

The verification is easy: at $x = x_k$,

$$\begin{aligned} p(x_k) =& y_0 l_0(x_k) + y_1 l_1(x_k) + y_2 l_2(x) + ...y_k l_k(x_k) + ... + y_n l_n(x_k) \\ =& y_0 \cdot 0 + y_1 \cdot 0 + ... + y_k \cdot 1 + ... + y_n \cdot 0 \\ =& y_k. \end{aligned}$$

Uniqueness. Is this THE polynomial we are looking for? The answer is yes, thanks to Theorem 1. Notice $p(x)$ we found above is of degree n or less. Thus, by Theorem 1, this is the only interpolation polynomial we could find.

Note. The Lagrange polynomials are solely determined by x_i's, independent of y_i's.

• Example. Given the data set from last problem, $(-1, 0), (1, -2), (3, 4)$, use the Lagrange form to find the interpolation polynomial.

Solution. We have $x_0 = -1, x_1 = 1, x_2 = 3$. Construct the Lagrange polynomials

$$l_0(x) = \frac{(x-1)(x-3)}{(-1-1)(-1-3)} = \frac{1}{8}(x^2 - 4x + 3)$$

$$l_1(x) = \frac{(x+1)(x-3)}{(1+1)(1-3)} = -\frac{1}{4}(x^2 - 2x - 3)$$

$$l_0(x) = \frac{(x+1)(x-1)}{(3+1)(3-1)} = \frac{1}{8}(x^2-1)$$

Then, apply $y_0 = 0, y_1 = -2, y_2 = 4$ to obtain

$$p(x) = 0l_0(x) - 2l_1(x) + 4l_2(x) = x^2 - x - 2$$

The same answer as before!

DONE.

Operation count: there are two types of calculations: the construction of $p(x)$ stored in a way that a computer can understand, and the evaluation of $p(\hat{x})$ at any point $\hat{x}$. The construction is done almost without cost since the coef of each Lagrange polynomial $l_k(x)$ is simply y_k. On the other hand, it is expensive to evaluate $p(\hat{x})$ for $\hat{x} \neq$ any x_i. It involves n subtractions and $(n-1)$ multiplications for the numerator for the least (the denominator can be computed only once) in each $l_k(x)$ for $0 \leq k \leq n$. So, totally it requires $O(n^2)$ operations to evaluate the entire polynomial.

The remedy: Newton form.

§ 5.3 Newton form.

Idea: building the interpolation incrementally with an iterative procedure. (iteration welcomed!)

Step 0. Start with a zero-th order polynomial that interpolates one point (x_0, y_0). It must be constant, $p_0(x) = y_0$.

Step 1. Construct a 1st order polynomial $p_1(x)$ based upon $p_0(x)$

$$p_1(x) = p_0(x) + r(x) \tag{2}$$

Since both p_1, p_0 interpolates (x_0, y_0), we must have

$$y_0 = p_1(x_0) = p_0(x_0) + r(x_0) \implies r(x_0) = 0 \implies r(x) = a_1(x - x_0)$$

, a consequence of having x_0 as a root of $r(x)$ which is of degree 1. To dertermine a_1, we plug the second point (x_1, y_1) into (2),

$$y_1 = p_1(x_1) = p_0(x_1) + a_1(x_1 - x_0)$$

$$\implies a_1 = \frac{y_1 - p_0(x_1)}{x_1 - x_0} \qquad \text{found } p_1(x) = p_0(x) + a_1(x - x_0)$$

.....

Step k-1. ... found $p_{k-1}(x)$

Step k. Based on $p_{k-1}(x)$, we construct

$$p_k = p_{k-1} + r(x)$$

Like before, we know that both $p_k(x)$ and $p_{k-1}(x)$ interpolate (x_0, y_0), (x_1, y_1), ... (x_{k-1}, y_{k-1}) and therefore $r(x) = p_k(x) - p_{k-1}(x)$ has roots at $x = x_0, x_1, ..., x_{k-1}$. Since $r(x)$ is at most of degree k, we have

$$r(x) = a_k(x - x_0)(x - x_1)...(x - x_{k-1})$$

with a_k to be dertermined. So

$$p_k(x) = p_{k-1}(x) + a_k(x - x_0)(x - x_1)...(x - x_{k-1}) \tag{3}$$

Plug the latest pair of data (x_k, y_k) into the above equation and solve for a_k

$$a_k = \frac{y_k - p_{k-1}(x_k)}{(x_k - x_0)(x_k - x_1)...(x_k - x_{k-1})}$$

$$\Longrightarrow \quad \text{found } p_k(x) \text{ as in 3}$$

and so on and so forth until the entire data set is used.

For the final answer, we can expand $p_n(x)$ iteratively

$$\begin{aligned} p_n(x) =& p_{n-1}(x) + a_n(x - x_0)(x - x_1)...(x - x_{n-1}) \\ =& p_{n-2}(x) + a_{n-1}(x - x_0)(x - x_1)...(x - x_{n-2}) + a_n(x - x_0)(x - x_1)...(x - x_{n-1}) \\ =& p_{n-3}(x) + ... \\ & ..... \\ =& a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + ... + a_n(x - x_0)...(x - x_{n-1}) \end{aligned} \tag{4}$$

Cost of adding more data. Adding one more data pair (x_{n+1}, y_{n+1}) won't affect any of the above calculation. We just need to do one more step after finding $p_n(x)$, that is, construction of

$$p_{n+1}(x) = p_n(x) + r(x) = p_n(x) + a_{n+1}(x - x_0)...(x - x_n)$$

or more specfically, solving for a_{n+1} in the above formula – $O(n)$ ops.

Nested form. To make evaluation of $p(x)$ faster, we introduce the so-called nested form, which is a post-interpolation procedure to make the final polynomial easy to evaluate at arbitrary point $x \neq x_i$. It is a reformulation of (4) after finding all the a_i's

$$p_n(x) = a_0 + (x - x_0)(a_1 + (x - x_1)(a_2 + (x - x_2)(...a_n(x - x_{n-1})))...) \tag{5}$$

It is easy to argue that to the operation count for evaluating $p(x)$ at any x is $O(n)$ with the nested form while $O(n^2)$ for the original Newton form as in (4).

• *Divided difference* and tree diagram – an alternative approch to compute a_n in the Newton form.

Introduce the divided difference $f[x_a, x_{a+1}, x_{a+2}, ..., x_b]$ for data set $\{(x_i, y_i)\}_{i=0}^n$. It's defined initially as one variable "functions"

$$f[x_0] = y_0,\ f[x_1] = y_1, ...,\ f[x_n] = y_n$$

as then recursively as multivariable "functions"

$$f[x_a, x_{a+1}, x_{a+2}, ..., x_b] = \frac{f[x_a, x_{a+1}, ..., f_{b-1}] - f[x_{a+1}, x_{a+2}, ..., x_b]}{x_a - x_b}$$

The coef a_i are written off from the process

$$a_0 = f[x_0],\ a_1 = f[x_0, x_1],\ a_2 = f[x_0, x_1, x_2],\ ...\ a_n = f[x_0, x_1, ..., x_n]$$

which are plugged into (4) and (5) to get the desired interpolation polynomial.

A tree diagram is often used to demonstrate this procedure. Let's have 4 pairs of data $(x_0, y_0), ...(x_3, y_3)$. First, construct the top level leaves directly from these data

$y_0 = f[x_0]$

$y_1 = f[x_1]$

$y_2 = f[x_2]$

$y_3 = f[x_3]$

Then, merge neighboring nodes to get to the second level

$$
\begin{array}{lll}
f[x_0] & & \\
 & \rightarrow \quad f[x_0, x_1] = \dfrac{f[x_0] - f[x_1]}{x_0 - x_1} & \\
f[x_1] & & \\
 & \rightarrow \quad f[x_1, x_2] = \dfrac{f[x_1] - f[x_2]}{x_1 - x_2} & \\
f[x_2] & & \\
 & \rightarrow \quad f[x_2, x_3] = \dfrac{f[x_2] - f[x_3]}{x_2 - x_3} & \\
f[x_3] & &
\end{array}
$$

Then, to the third level

$$
\begin{array}{llll}
f[x_0] & & & \\
 & \rightarrow \quad f[x_0, x_1] & & \\
f[x_1] & & \rightarrow \quad f[x_0, x_1, x_2] = \dfrac{f[x_0, x_1] - f[x_1, x_2]}{x_0 - x_2} & \\
 & \rightarrow \quad f[x_1, x_2] & & \\
f[x_2] & & \rightarrow \quad f[x_1, x_2, x_3] = \dfrac{f[x_1, x_2] - f[x_2, x_3]}{x_1 - x_3} & \\
 & \rightarrow \quad f[x_2, x_3] & & \\
f[x_3] & & &
\end{array}
$$

And finally

$$
\begin{array}{llll}
f[x_0] & & & \\
 & \rightarrow \quad f[x_0, x_1] & & \\
f[x_1] & & \rightarrow \quad f[x_0, x_1, x_2] & \\
 & \rightarrow \quad f[x_1, x_2] & & \rightarrow \quad f[x_0, ..., x_3] = \dfrac{f[x_0, x_1, x_2] - f[x_1, x_2, x_3]}{x_0 - x_3} \\
f[x_2] & & \rightarrow \quad f[x_1, x_2, x_3] & \\
 & \rightarrow \quad f[x_2, x_3] & & \\
f[x_3] & & &
\end{array}
$$

Note. The coef a_i are located on the top edge of this diagram.

Final words about the Newton form: by Theorem 1, regardless of the appraoch or form we use, the final results are always identical – uniqueness of polynomial interpolation.

Error estimates. Both Lagrange and Newton forms give the same polynomial and thus same error

$$e(x) = f(x) - p(x). \tag{6}$$

One regards this as a smooth function, then it vanishes at $x = x_i,\ i = 0, 1, 2, ...n$. To

find a bound for $e(x)$, we use the following auxiliary funtion

$$g(t) = e(t) - e(x)\prod_{i=0}^{n}\frac{t-x_i}{x-x_i}. \tag{7}$$

Note that this is a function of t for any fixed x. Now set the independent variable $t = x_i$ to have

$$g(x_i) = e(x_i) - e(x)\cdot 0 = e(x_i).$$

Due to interpolation and (6), $e(x_i) = f(x_i) - p(x_i) = 0$ and thus

$$g(x_i) = 0 \text{ for } i = 0, 1, 2, ..., n$$

Moreover, setting $t = x$ in (7), we have

$$g(x) = e(x) - e(x)\cdot 1 = 0.$$

Therefore, $g(t)$ is a smooth function that vanishes at $n+2$ points. By the mean value theorem $g'(t)$ has at least one zero inside (away from the interfaces of) each one of the $n+1$ sub-intervals separated by these $n+2$ points. So, $g'(t)$ vanishes at $n+1$ points. By the same token, $g''(t)$ vanishes at n points ..... $g^{(n+1)}(t)$ vanishes at some point $t = \xi$. Now, differentiate the RHS of (7) with respect to t for $n+1$ times.

$$e^{(n+1)}(t) = f^{(n+1)}(t) - p^{(n+1)}(t) = f^{(n+1)}(t)$$

since $p(x)$ is a polynomial of degree no greater than n. Also,for the product term in (7),

$$\frac{d^{n+1}}{dt^{n+1}}\prod_{i=0}^{n}\frac{t-x_i}{x-x_i} = (n+1)!\prod_{i=0}^{n}\frac{1}{x-x_i} = \frac{(n+1)!}{\prod_{i=0}^{n}(x-x_i)}$$

Therefore,

$$g^{(n+1)}(t) = f^{(n+1)}(t) - \frac{e(x)(n+1)!}{\prod_{i=0}^{n}(x-x_i)}$$

and apply $g^{(n+1)}(\xi) = 0$ in above and rearrange,

$$e(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\prod_{i=0}^{n}(x-x_i) \tag{8}$$

for some ξ. This is the error formula.

• § **5.6 Cubic spline interpolation.**

Another philosophy of interpolation: instead of using a single polynomial with high degrees, we use a set of polynomials with lower degrees on each sub-intervals (x_i, x_{i+1}). This way, one can avoid certain oscillations caused by polynomials with too high degrees.

The easiest one is *piecewise linear interpolation*: use a straight line (i.e. polynomial of 1st degree, i.e. linear functions) to connect two neighboring points (x_i, y_i), (x_{i+1}, y_{i+1}). More specifically, for $x \in [x_i, x_{i+1}]$, the interpolation polynomial is

$$p_i(x) = y_i + \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i).$$

What about error bound for piecewise linear interpolation? First, assume

$$a = x_0 < x_1 < x_2 < ... < x_n = b.$$

We then apply the error formula (8) with $n = 1$ on each and every $p_i(x)$ for $x \in [x_i, x_{i+1}]$,

$$f(x) - p_i(x) = \frac{f''(\xi_i)}{2!}(x - x_i)(x - x_{i+1})$$

for some $\xi_i \in [x_i, x_{i+1}]$. In order to obtain a uniform bound on $e(x)$ that is independent of x and i, we need to find bounds for the f'' and $(x - x_i)(x - x_{i+1})$ terms above

$$|f''(\xi)| \leq M_2 \stackrel{def}{=} \max_{x \in [a,b]} |f''(x)|,$$

$$|(x - x_i)(x - x_{i+1})| \leq \left(\frac{(x - x_i) + (x_{i+1} - x)}{2} \right)^2 \leq \frac{h_{max}^2}{4}$$

where

$$h_{max} \stackrel{def}{=} \max_{0 \leq i \leq n-1} \{|x_{i+1} - x_i|\}$$

is maximal length of the sub-intervals.

In short, the error is bounded by

$$|e(x)| \leq \frac{M_2 h_{max}^2}{4}.$$

The two factors that decide this bound is the magnitude of f'' and the resolution h_{max}. We say: piecewise linear interpolation is of $O(h^2)$ accuracy.

However, this interpolation has a big disadvantge: the fitting curve has sharp corners at (x_i, y_i) as the slope the curve changes abruptly. Therefore, we should also take into account the smoothness of the interpolation.

Cubic spline interpolation can satisfy these requirements. It uses a set of cubic functions, i.e. polynomials of degree 3 for interpolation, yielding a fitting curve with C^2 smoothness, that is, with continuous 1st and 2nd derivatives.

Defintion. Denote the cubic polynomials as $s_0(x), s_1(x), s_2(x), ..., s_n(x)$. Each $s_i(x)$ is defined for $x \in [x_i, x_{i+1}]$, satisfying

1. Interpolation condition: for $i = 0, 1, ..., n-1$

$$s_i(x_i) = y_i$$

$$s_i(x_{i+1}) = y_{i+1}$$

2. Smoothness condition: for $i = 1, 2, ..., n-1$

$$s'_{i-1}(x_i) = s'_i(x_i)$$

$$s''_{i-1}(x_i) = s''_i(x_i)$$

How to find $s_i(x)$?

A straightforward way is to use undertermined coefficients. Let

$$s_i(x) = a_i x^3 + b_i x^2 + c_i x + d_i$$

where $\{a_i, b_i, c_i, d_i\}_{i=0}^n$ are $4n$ coef to be determined using the above conditions. Notice there are totally $4n-2$ conditions: $2n$ for interpolation, $n-1$ for 1st derivatives and $n-1$ for second derivatives. So we can use 2 extra conditions so that the coef can be determined *uniquely*. For example, a natural cubic spline uses zero boundary conditions

$$s_0(x_0) = 0, \quad s_{n-1}(x_n) = 0.$$

A much more efficient way is to use the following formulation (Let $h_i = x_{i+1} - x_i$ for $i = 0, 1, ..., n-1$).

$$\begin{aligned} s_i(x) =& \frac{a_i}{6h_i}(x_{i+1} - x)^3 + \frac{a_{i+1}}{6h_i}(x - x_i)^3 + \\ & \left(y_i - \frac{a_i h_i^2}{6}\right)\frac{x_{i+1} - x}{h_i} + \left(y_{i+1} - \frac{a_{i+1}h_i^2}{6}\right)\frac{x - x_i}{h_i} \end{aligned} \tag{9}$$

for $x \in [x_i, x_{i+1}]$ and $0 \le i \le n-1$

Here, the undetermined coef are a_0, a_1, ... a_n.

It is easy to verify that (9) already satifies the interpolation conditions

$$s_i(x_i) = y_i, \qquad s_i(x_{i+1}) = y_{i+1} \qquad \text{for } i = 0, 1, ...n-1$$

and the 2nd derivative condition

$$s''_{i-1}(x_i) = s''_i(x_i), \qquad \text{for } i = 1, 2, ..., n-1$$

Now, we use the 1st derivative condition

$$s'_{i-1}(x_i) = s'_i(x_i), \qquad \text{for } i = 1, 2, ..., n-1$$

to find a_i.

Differentiate (9) once to get

$$s'_i(x) = -\frac{a_i(x_{i+1}-x)^2}{2h_i} + \frac{a_{i+1}(x-x_i)^2}{2h_i} - \left(y_i - \frac{a_i h_i^2}{6}\right)\frac{1}{h_i} + \left(y_{i+1} - \frac{a_{i+1}h_i^2}{6}\right)\frac{1}{h_i}$$

Plug in $x = x_i$ to get

$$s'_i(x_i) = -\frac{a_i h_i}{3} - \frac{a_{i+1}h_i}{6} - \frac{y_i}{h_i} + \frac{y_{i+1}}{h_i} \tag{10}$$

Plug in $x = x_{i+1}$ to get

$$s'_i(x_{i+1}) = \frac{a_{i+1}h_i}{3} + \frac{a_i h_i}{6} - \frac{y_i}{h_i} + \frac{y_{i+1}}{h_i} \tag{11}$$

Since the 1st derivative condition is $s'_i(x_i) = s'_{i-1}(x_i)$, we shift equation (11) from x_{i+1} to x_i by replacing every occurence of i with $i-1$

$$s'_{i-1}(x_i) = \frac{a_i h_{i-1}}{3} + \frac{a_{i-1}h_{i-1}}{6} - \frac{y_{i-1}}{h_{i-1}} + \frac{y_i}{h_{i-1}}$$

Now, equate the above equation with (10)

$$\begin{aligned} &-\frac{a_i h_i}{3} - \frac{a_{i+1}h_i}{6} - \frac{y_i}{h_i} + \frac{y_{i+1}}{h_i} \\ =&\frac{a_i h_{i-1}}{3} + \frac{a_{i-1}h_{i-1}}{6} - \frac{y_{i-1}}{h_{i-1}} + \frac{y_i}{h_{i-1}} \end{aligned}$$

and simplify

$$\frac{h_{i-1}}{6}a_{i-1} + \frac{h_{i-1}+h_i}{3}a_i + \frac{h_i}{6}a_{i+1} = \frac{y_{i+1}-y_i}{h_i} - \frac{y_i - y_{i-1}}{h_{i-1}}$$

for $i = 1, 2, ..., n-1$. Notice there are only $n-1$ equations with $n+1$ a_i's. We then impose the boundary conditions $s_0''(x_0) = 0$, $s_{n-1}''(x_n) = 0$ so that $a_0 = 0$, $a_n = 0$ (in which case it is called "natural cubic spline"). Now it only remains to solve the linear system,

$$\begin{pmatrix} \frac{h_0+h_1}{3} & \frac{h_1}{6} & & & & & \\ \frac{h_1}{6} & \frac{h_1+h_2}{3} & \frac{h_2}{6} & & & & \\ & \ddots & \ddots & \ddots & & & \\ & & \frac{h_{i-1}}{6} & \frac{h_{i-1}+h_i}{3} & \frac{h_i}{6} & & \\ & & \ddots & \ddots & & \ddots & \\ & & & & \frac{h_{n-2}}{6} & \frac{h_{n-2}+h_{n-1}}{3} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_i \\ \vdots \\ a_{n-1} \end{pmatrix} = \begin{pmatrix} \frac{y_2-y_1}{h_1} - \frac{y_1-y_0}{h_0} \\ \frac{y_3-y_2}{h_2} - \frac{y_2-y_1}{h_1} \\ \vdots \\ \frac{y_{i+1}-y_i}{h_i} - \frac{y_i-y_{i-1}}{h_{i-1}} \\ \vdots \\ \frac{y_n-y_{n-1}}{h_{n-1}} - \frac{y_{n-1}-y_{n-2}}{h_{n-2}} \end{pmatrix}$$

This result is the best linear system one can expect to solve: *tridiagonal, symmetric and strictly diagonally dominant!* — solved in $O(n)$ ops with direct methods.