We have previously considered the methodology of Cubic Hermite Spline Interpolation

$$S_K(x_K) = y_K$$
$$S_K(x_{K+1}) = y_{K+1}$$
$$S_K'(x_K) = y_K'$$
$$S_K'(x_{K+1}) = y_{K+1}'$$

One of the methods for computing the cubic $s_K(x)$ was to fit a cubic polynomial to the 2 "real" x-values x_K & x_{K+1} and the 2 additional "fictitious" x-values x_K^*, x_{K+1}^*:

We used Newton interpolation & divided differences, and ultimately obtained the Hermite spline by taking the limit behavior:

$$x_K^* \longrightarrow x_K$$

$$x_{K+1}^* \longrightarrow x_{K+1}$$

Interestingly, by introducing these 2 fictitious x-values we can also get an estimate for the error.

Remember, for a 4th order interpolant with points $x_k^*, x_k, x_{k+1}, x_{k+1}^*$:

$$f(x) - S_k(x) = \frac{f^{(4)}(\theta_k)}{4!}(x-x_k^*)(x-x_k)(x-x_{k+1})(x-x_{k+1}^*)$$

$$(x \in [x_k, x_{k+1}])$$

In the limit $x_k^* \longrightarrow x_k$

$x_{k+1}^* \longrightarrow x_{k+1}$:

$$f(x) - S_k(x) = \frac{f^{(4)}(\theta_k)}{24}(x-x_k)^2(x-x_{k+1})^2$$

$$\Rightarrow |f(x) - S_k(x)| \leq \frac{1}{24}\|f^{(4)}\|_\infty \underbrace{\left[\max |(x-x_k)(x-x_{k+1})|\right]^2}_{= \frac{h_k^2}{4}, \text{ from previous lecture}}$$

$$\Rightarrow |f(x) - S_k(x)| \leq \frac{1}{24}\|f^{(4)}\|_\infty \frac{h_k^4}{16} = \frac{1}{384}\|f^{(4)}\|_\infty h_k^4$$

(compare with $|f(x) - S_k(x)| \leq \frac{5}{384}\|f^{(4)}\|_\infty h^4$, for cubic spline method).

Let us summarize some differences of these 2 methods

Cubic Hermite Spline	Cubic Spline
Requires the derivatives $\{y'_k\}$	Requires only function values $\{y_k\}$
Error $\leq \frac{1}{384} \|f^{(4)}\|_\infty h^4$	Error $\leq \frac{5}{384} \|f^{(4)}\|_\infty h^4$
Continuous 1st derivatives (C_1)	Continuous 2nd derivatives (C_2)
Each S_k can be computed independently, and in parallel	The coefficients of the S_k's must be computed jointly for all k.

Matlab:
`s = pchip(x,y)`
`z = ppval(s,w)`
	Matlab
	`s = spline(x,y)`
	`z = ppval(s,w)`

Note Many times, we don't have derivative values $\{y'_k\}$ at our disposal, but we would still like to use a Cubic Hermite Spline. In this case, we can try to synthesize approximate values of y'_k's from the $\{y_k\}$

Possible approaches:

- $y_n' \approx \frac{1}{2}\left[\frac{y_{n+1}-y_n}{x_{n+1}-x_n} + \frac{y_n - y_{n-1}}{x_n - x_{n-1}}\right]$
- Catmull-Rom: $y_n' \approx \frac{y_{n+1}-y_{n-1}}{x_{n+1}-x_{n-1}}$

} Interpolation error is now bounded by $K \cdot h^2$, due to error in derivative approximation

- More accurate formula for y_n', using more points (e.g. x_{n-2} through x_{n+2})

Adjusted derivatives to force monotonous interpolation ⇒

⇒ if y_i's are in increasing/decreasing magnitude, $s(x)$ has same monotonicity

} ⇒ This is what MATLAB's pchip does.

Linear algebra

We shall turn our attention to solving linear systems of equations $Ax = b$

$$A \in \mathbb{R}^{m \times n}$$
$$x \in \mathbb{R}^{n}$$
$$b \in \mathbb{R}^{m}.$$

We already saw examples of methods that required the solution of a linear system as part of the overall algorithm, e.g. the Vandermonde system for interpolation, which was a square system ($m = n$).

Another category of methods that leads to rectangular systems with $m > n$ is least square methods. They answer questions of the form:

→ What is the best n-order polynomial we can use to approximate (not interpolate) $(m+1)$ data points (where $m > n$).

→ More generally, find the solution that most closely satisfies m equations, in the presence of n ($n < m$) unknowns.

All these algorithms need to be conscious about <u>error</u>, and there are at least 3 sources for it.

→ Some algorithms are "imperfect" in the sense that they require several iterations to generate a good quality approximation. Thus, intermediate results are subject to <u>error</u>

→ Sometimes, it is not possible to find an "ideal" solution, e.g. because we have more equations than unknowns. In this case, not all equations will be satisfied exactly, and we need a notion of the "error" incurred in not satisfying certain equations fully.

→ Inputs to an algorithm are often corrupted by noise, roundoff error, etc. For example, instead of solving an "intended" system $Ax = b$ we may be solving $A^* x = b^*$ where the entries in A^* & b^* have been subject to noise and inaccuracy. It is important to know how those translate to errors in determining x.

(Vector and Matrix) Norms :

3/10/11

Norms are valuable tools in arguing about the extent and magnitude of error. We will introduce some concepts that we will use broadly later on.

Definition A vector norm is a function from $\mathbb{R}^n$ to $\mathbb{R}$, with a certain number of properties. If $\underline{x} \in \mathbb{R}^n$, we symbolize its norm by $\|\underline{x}\|$. The defining properties of a norm are:

(i) $\|\underline{x}\| \geq 0$ for all $x \in \mathbb{R}^n$

also $\|\underline{x}\| = 0$ iff $\underline{x} = \underline{0}$

(ii) $\|\alpha \underline{x}\| = |\alpha| \cdot \|\underline{x}\|$ for all $\alpha \in \mathbb{R}$, $\underline{x} \in \mathbb{R}^n$

(iii) $\|\underline{x}+\underline{y}\| \leq \|x\| + \|y\|$ for all $x, y \in \mathbb{R}^n$

Note that the properties above do not determine a unique for of a "norm" function, in fact many different valid norms exist. Typically, we will use subscripts ($\|\cdot\|_a, \|\cdot\|_b$) to denote different types of norms.