Lecture 17 Numerical Analysis April 3, 2018

Last time:

Jacobi's Method:

For symmetric real matrix A, compute $R_1 \dots R_M$ such that

$$(R_1 \cdot R_2 \cdots R_M)^T A (R_1 \cdots R_M) \approx D = \begin{pmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{pmatrix}$$

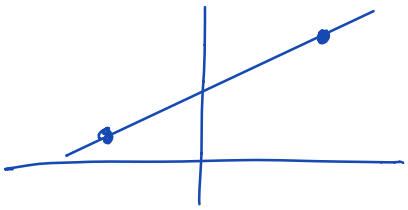
Each R_j is a rotation matrix which zeros out two off-diagonal elements.

The scheme can be shown to converge, and real-life convergence is often much faster than indicated in the proof.

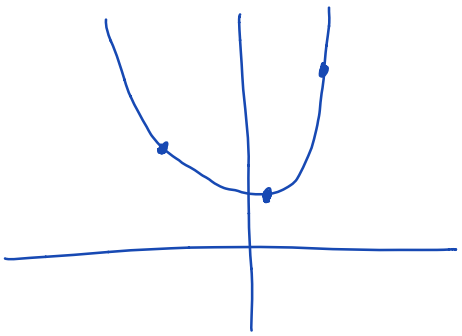
All $\lambda_j, \underline{v}_j$ are computed simultaneously.

Next up: Polynomial Interpolation

Ex: • 2 points in xy-plane define a line



• 3 points define a parabola (deg 2 pol.)



In general, n+1 unique points define a polynomial of degree n:

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \dots + a_1 x + a_0$$

$$\Rightarrow \begin{cases} p(x_0) = y_0 \\ \vdots \\ p(x_n) = y_0 \end{cases}$$ n+1 equations for the n+1 unknowns a_j.

The point of interpolation

Most functions (i.e. solutions to ODEs, PDEs, etc.) are *not* polynomials, and do *not* have closed form solutions. *However*, most of the time they can be locally approximated via polynomials (think Taylor series). Polynomial interpolation is at the *core* of Numerical Analysis.

With this in mind, given (x_j, y_j), $j=0,\dots,n$ on $[a,b]$, how do we compute the *interpolant*.

Option 1 Solve for the coefficients in

$$p_n(x) = a_n x^n + \dots + a_1 x + a_0$$

$$\Rightarrow \underbrace{\begin{pmatrix} x_0^n & x_0^{n-1} & \dots & x_0 & 1 \\ x_1^n & x_1^{n-1} & \dots & x_1 & 1 \\ \vdots & & & & \\ x_n^n & x_n^{n-1} & \dots & x_n & 1 \end{pmatrix}}_{\text{Vandermonde Matrix} = A} \begin{pmatrix} a_n \\ a_{n-1} \\ \vdots \\ a_0 \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{pmatrix}$$

Badly chosen points, exponential κ

If the x_j are distinct, A is formally invertible, but horribly ill-conditioned.
Never try to invert it!

Option 2 The coefficients a_j usually don't matter — the goal is usually to evaluate p_n at some $x \neq x_j$.

While the polynomial p_n is unique, there are many ways to construct/evaluate it, the most common of which is the Lagrange Interpolation Polynomial.

Idea: Construct a sequence of polynomials $L_k \in P_n$ such that

$$L_k(x_j) = \begin{cases} 1 & j=k \\ 0 & j \neq k \end{cases}.$$

P_n: vector space of deg $\leq n$ polynomials.

Then $p_n(x) = \sum_{k=0}^{n} L_k(x)\, y_k$ is the interpolating polynomial for the data (x_j, y_j), $j = 0, \ldots, n$.

The construction of such polynomials L_k is straight forward:

$$L_k = \prod_{\substack{j=0 \\ j \neq k}}^{n} \frac{x - x_j}{x_k - x_j}$$

$$= \left(\prod_{j=0}^{n} \frac{1}{(x_k - x_j)} \right) \left(\prod_{j=0}^{n} (x - x_j) \right)$$

Theorem Given data (x_j, y_j), $j = 0, \ldots, n$, there exists a unique polynomial $p_n \in P_n$ such that

$$p_n(x_j) = y_j.$$

Proof: Existence by Lagrange formula, uniqueness see text.

The form of the interpolating polynomial:

$$p_n(x) = \sum_{k=0}^{n} L_k(x) \, y_k$$

is referred to as the "Lagrange interpolation formula of degree n".

There are a few questions that can be asked of p_n at this point:

(1) If (x_j, y_j) come from a smooth function, what is the interpolation error. I.e.

(2) What is the cost of evaluating p_n? If a new data point is added, (x_{n+1}, y_{n+1}), what is the cost of updating p_n?

(3) In floating point arithmetic, is the evaluation of p_n stable?

Question 1 Note, if $y_j = f(x_j)$, then $p_n(x_j) = f(x_j)$ by construction. If $x \neq x_j$, then

Thm: Let $f \in C^{n+1}[a,b]$. For $x \in [a,b]$, there exists $\xi = \xi(x) \in (a,b)$ such that

$$f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} \prod_{j=0}^{n} (x - x_j)$$

depends on the choice of points.

Mention L_∞ vs. L_2 approx.

Interp. vs. Approx.

Moreover,

$$|f(x) - p_n(x)| \leq \frac{M_{n+1}}{(n+1)!} |\pi_{n+1}(x)|$$

with $M_{n+1} = \max_{t \in [a,b]} |f^{(n+1)}(t)|$ and

$$\pi_{n+1}(x) = \prod_{j=0}^{n} (x - x_j).$$

See text for proof (Taylor + Rolle's).

Merely note that this theorem is only useful if M_{n+1} can be calculated, and that the interpolation error depends on where the nodes are located ← this will be important later on.

Question 2 The cost of evaluating p_n.

$$p_n(x) = \sum_{k=0}^{n} L_k(x) y_k \quad , \quad L_k(x) = \prod_{\substack{j=0 \\ j \neq k}}^{n} \frac{x - x_j}{x_k - x_j}$$

Computing $L_k(x)$ requires $2n$ subtractions, $2(n-1)$ multiplies, and one divide $\Rightarrow$ $O(n)$ flops.

There are $n+1$ L_k's, so $\Rightarrow$ $O(n^2)$ flops to evaluate p_n.

Compare this with Horner's Method:

If the coefficients $a_0, \ldots, a_n$ are known in

$p_n(x) = a_0 + a_1 x + \ldots + a_n x^n$, then we can rewrite p_n as:

$$p_n(x) = a_0 + x\left(a_1 + x\left(a_2 + x\left(a_3 \cdots + x\left(a_{n-1} + a_n x\right)\right)\right)\right)$$

b_{n-1}

b_{n-2}

$\vdots$

b_1

b_0

$b_{n-1} = a_{n-1} + a_n x$ (1 mult, 1 add)

$b_{n-2} = a_{n-2} + b_{n-1} x$ (1 mult, 1 add)

$\vdots$

$b_0 = a_0 + b_1 x$ (1 mult, 1 add)

$= p_n(x)$ $\Rightarrow$ $2n$ flops.

This means that the Lagrange form is very inefficient.

If new data is added, (x_{n+1}, y_{n+1}),

$$L_k \to L_k \frac{(x - x_{n+1})}{(x_k - x_{n+1})}, \quad L_{n+1} = \text{as before.}$$

$\Rightarrow O(n)$ update

$\Rightarrow O(n^2)$ eval.