Lecture 18 (11/08/2019)

How to find polynomial interpolation (in Lagrange form) by Matlab?

$x = [x_1\ x_2\ \cdots\ x_n]$
$y = [y_1\ y_2\ \cdots\ y_n]$
} arrays of x- and y-coordinates (given)

We know that $L_i(t) = \dfrac{(t-x_1)\cdots(t-x_{i-1})(t-x_{i+1})\cdots(t-x_n)}{(x_i-x_1)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)}$

We will program an array $L = [L_1\ L_2\ \cdots\ L_n]$. This is a row vector (a $1\times n$ matrix) whose entries are polynomials, not numbers. We will need a symbolic variable t. It can be declared on Matlab as

```
syms t
```

Then we initialize array L by

```
L = zeros(1,n,'sym')
```

(1,n: $1\times n$ matrix; 'sym': each entry is matrix of 'symbolic' format)

How to find L_i?

We relabel the array x as follows:

$x_1\ x_2\ \cdots\ x_{i-1}\ x_{i+1}\ \cdots\ x_n$
$z_1\ z_2\ \cdots\ z_{i-1}\ z_i\ \cdots\ z_{n-1}$

In Matlab, one can write

```
z = x
z(i) = []
```

← removing the i'th entry of array z.

Then L_i can be written simply as

$$L_i(t) = \frac{(t-z_1)\cdots(t-z_{n-1})}{(x_i-z_1)\cdots(x_i-z_{n-1})}$$

One can rewrite L_i as

$$L_i(t) = \frac{t-z_1}{x_i-z_1}\cdots\frac{t-z_{n-1}}{x_i-z_{n-1}}$$

(product of n-1 fractions)

One will need two 'for' loops: one to run over the array L, one to compute the product above. Once the array L is found, the interpolating polynomial P is given by

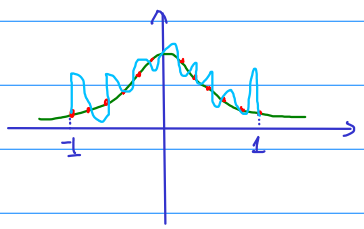
$$P(t) = y_1 L_1(t) + \cdots + y_n L_n(t) = [L_1(t) \cdots L_n(t)] \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = L * \text{transpose}(y)$$

See the code posted on the course website for detail.

Suppose we are given a curve (not a polynomial curve), say

$$f(x) = \frac{1}{1+x^2}$$

We sample many points on this curve, with $x \in [-1, 1]$.



A natural question is: can the polynomial curve approximate the given curve as n goes to infinity?
This question will be studied in the next homework set.

Newton formula

Recall that there is only one polynomial of degree $\leq n-1$ passing through n given points $(x_1, y_1), \ldots, (x_n, y_n)$. However, there are several quick ways to find this polynomial (called P).

Lagrange's formula:

$$P(x) = y_1 L_1(x) + \cdots + y_n L_n(x).$$

Newton's formula:

$$P(x) = c_0 + c_1 N_1(x) + \cdots + c_{n-1} N_{n-1}(x)$$

where

$$N_1(x) = x - x_1$$
$$N_2(x) = (x-x_1)(x-x_2)$$
$$\cdots\cdots$$
$$N_{n-1}(x) = (x-x_1)\cdots(x-x_{n-1})$$

$1, N_1, N_2, \ldots, N_{n-1}$ are called Newton basis polynomials.

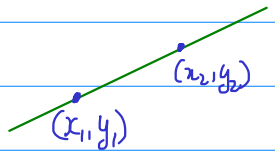
In practice, when we have one more data point, say (x_{n+1}, y_{n+1}), we would need one more basis polynomial

$$N_n(x) = (x-x_1)\cdots(x-x_{n-1})(x-x_n).$$

The coefficients $c_0, c_1, \ldots, c_{n-1}$ stay the same. Only c_n has to be computed.

In Lagrange's formula, the situation is somewhat reversed: one doesn't have to recompute the coefficients, but every basis polynomials. How to compute $c_0, c_1, \ldots, c_{n-1}$?

Let's consider the case $n = 2$:



$$P(x) = c_0 + c_1(x-x_1)$$

equation of a line

For this line to pass through (x_1, y_1) and (x_2, y_2), one needs

$$c_0 = f(x_1)$$
$$c_1 = \frac{f(x_2) - f(x_1)}{x_2 - x_1}$$

this is called difference quotient.

This quotient is denoted by $f[x_1, x_2]$.

Suppose we now have one more point (x_3, y_3). The polynomial curve has to be a parabola because one cannot expect a line to pass through three points (at general positions).

Newton's idea is to keep the polynomials 1 and $x-x_1$, and add a new polynomial $(x-x_1)(x-x_2)$.

$$P(x) = c_0 + c_1(x-x_1) + c_2(x-x_1)(x-x_2).$$

Substituting $x = x_3$, one can find c_2 as follows:

$$c_2 = \frac{f(x_3) - (f(x_1) + f[x_1, x_2])(x-x_1)}{(x_3-x_1)(x_3-x_2)}$$

After some algebraic computations, one sees that

$$c_2 = \frac{f[x_3, x_2] - f[x_2, x_1]}{x_3 - x_1}$$ } called divided difference

This quotient denoted by $f[x_1, x_2, x_3]$.

In general,

$$P(x) = c_0 + c_1(x-x_1) + c_2(x-x_1)(x-x_2) + \cdots + c_n(x-x_1)\cdots(x-x_{n-1})$$

where c_k, denoted by $f[x_1, \ldots, x_{k+1}]$, is computed recursively as

$$f[x_1, \ldots, x_{k+1}] = \frac{f[x_2, \ldots, x_{k+1}] - f[x_1, \ldots, x_k]}{x_{k+1} - x_1}$$