How to put a polynomial through points

Ed Bueler

MATH 426 Numerical Analysis

purpose

The topic of these slides is covered in Chapter 8 of the text.[1] *When we get there we will be more thorough.*

The emphasis here, in these slides, is on how *to put a polynomial through points. The polynomial interpolation error theorem (Chapter 8), addresses the "how good is the result" question. While it is a good idea to look at Chapter 8, it is not needed for understanding these slides or for doing Assignment #3.*

[1] Greenbaum & Chartier, *Numerical Methods: Design, Analysis, and Computer Implementation of Algorithms*, Princeton University Press 2012).

an example of the problem

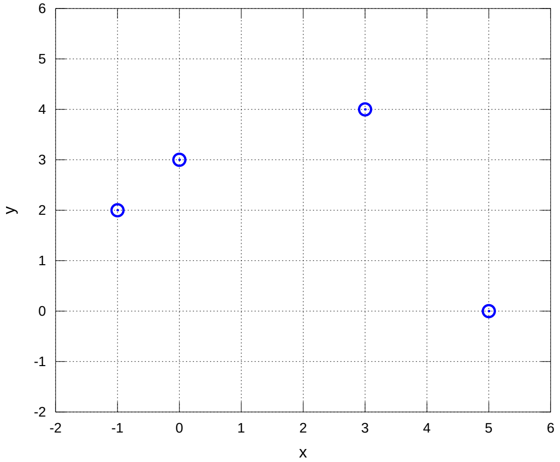
- suppose you have a function $y = f(x)$ which goes through these points:

$$(-1, 2), \quad (0, 3), \quad (3, 4), \quad (5, 0)$$

- the x-coordinates of these points are not equally-spaced!
  - in these notes I will *never* assume the x-coordinates are equally-spaced
- name the points (x_i, y_i), for $i = 1, 2, 3, 4$
- there is a polynomial $P(x)$ of degree 3 which goes through these points
- we will build it concretely
- we will show later that there is only one such polynomial

a picture of the problem

- figure below shows the points from the previous slide
- they may be values of a function $f(x)$ ... but we don't know or see that function



how to find $P(x)$

- suppose $P(x)$ is the degree 3 polynomial through the 4 points
- a standard way to write it is:

$$P(x) = c_0 + c_1 x + c_2 x^2 + c_3 x^3$$

- *note*: there are 4 unknown coefficients and 4 points
  - degree $n-1$ polynomials have the right length for n points
- the facts "$P(x_i) = y_i$" for the given points gives 4 linear equations in 4 unknowns:

$$\begin{aligned}
c_0 + c_1(-1) + c_2(-1)^2 + c_3(-1)^3 &= 2 \\
c_0 + c_1(0) + c_2(0)^2 + c_3(0)^3 &= 3 \\
c_0 + c_1(3) + c_2(3)^2 + c_3(3)^3 &= 4 \\
c_0 + c_1(5) + c_2(5)^2 + c_3(5)^3 &= 0
\end{aligned}$$

- MAKE SURE that you are clear on how I got these equations, and that you can do the same thing in an example with different points or different polynomial degree

a linear system

- you can solve the equations by hand ... that would be tedious
- you are allowed a wand ... I mean a MATLAB
- the system has a matrix form "$A\mathbf{v} = \mathbf{b}$" with $\mathbf{v}$ unknown:

$$\begin{bmatrix} 1 & -1 & (-1)^2 & (-1)^3 \\ 1 & 0 & 0^2 & 0^3 \\ 1 & 3 & 3^2 & 3^3 \\ 1 & 5 & 5^2 & 5^3 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 4 \\ 0 \end{bmatrix}$$

- I am not simplifying the numbers in the matrix ... because:
  - a machine can do that, and
  - the pattern is what matters[2]
- MAKE SURE you can convert from the original "fit a polynomial through these points" question into the matrix form "$A\mathbf{v} = \mathbf{b}$"

[2] often you should keep things unsimplified if they are going to become code

how to *easily* find $P(x)$

- MATLAB is designed to solve linear systems ... easily!
- enter the matrix and the known vector into MATLAB:

```
>> A = [1 -1 (-1)^2 (-1)^3; 1 0 0^2 0^3; 1 3 3^2 3^3; 1 5 5^2 5^3]
A =
     1    -1     1    -1
     1     0     0     0
     1     3     9    27
     1     5    25   125
>> b = [2; 3; 4; 0]
b =
   2
   3
   4
   0
```

- solve the linear system to get $\mathbf{v} = [c_0\ c_1\ c_2\ c_3]$:

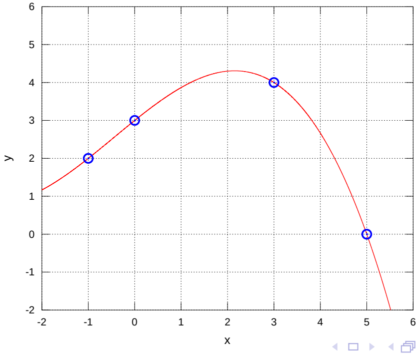
```
>> v = A \ b
v =
   3.000000
   0.983333
  -0.066667
  -0.050000
```

- so the polynomial is $P(x) = 3 + 0.983333x - 0.066667x^2 - 0.05x^3$

did we solve the problem?

- the polynomial we found does go through the points:

```
>> 3.000000 + 0.983333*(-1) - 0.066667*(-1)^2 -0.050000*(-1)^3
ans =  2
>> 3.000000 + 0.983333*(0) - 0.066667*(0)^2 -0.050000*(0)^3
ans =  3
>> 3.000000 + 0.983333*(3) - 0.066667*(3)^2 -0.050000*(3)^3
ans =  4.0000
>> 3.000000 + 0.983333*(5) - 0.066667*(5)^2 -0.050000*(5)^3
ans = -1.0000e-05
```



summary: linear systems in MATLAB

- before we move on, here is a summary of some basics
- you enter matrices like A by rows
  - spaces separate entries
  - semicolons separate rows
- column vectors like $\mathbf{b}$ are just matrices with one column
  - to quickly enter column vectors use the transpose operation:

```
>> b = [2 3 4 0]'
b =
   2
   3
   4
   0
```

- to solve the system $A\mathbf{v} = \mathbf{b}$ we "divide by" the matrix: $\mathbf{v} = A^{-1}\mathbf{b}$ ... but this is *left* division; MATLAB has a single-character *backslash* operation:

```
>> v = A \ b
```

- the forward slash does not work because of the sizes of the matrix and the vector are not right:

```
>> v = b / A   % NOT CORRECT for our A and b; wrong sizes
```

the general case

- suppose we have n points (x_i, y_i) with distinct x-coordinates
  - for example, if $n = 4$ we have points (x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4)
- then the polynomial has degree one less: the polynomial $P(x)$ which goes through the n points has degree $n - 1$
- the polynomial has this form:

$$P(x) = c_0 + c_1 x + c_2 x^2 + \cdots + c_{n-1} x^{n-1}$$

- the equations which determine $P(x)$ say that *the polynomial goes through the points*:

$$P(x_i) = y_i \qquad \text{for} \quad i = 1, 2, \dots, n$$

- this is a system of n equations of this form:

$$c_0 + c_1 x_i + c_2 x_i^2 + \cdots + c_{n-1} x_i^{n-1} = y_i \qquad \text{for} \quad i = 1, 2, \dots, n$$

- the n coefficients c_i are unknown, while the x_i and y_i are known

the pattern in the matrix

- as a matrix:

$$A = \begin{bmatrix} 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{n-1} \\ & \vdots & & \ddots & \\ 1 & x_n & x_n^2 & \dots & x_n^{n-1} \end{bmatrix}$$

- and **b** is a column vector with entries y_i: $\mathbf{b} = [y_1 \quad y_2 \quad \dots\ y_n]'$
- as before, this gives a system of n equations: $A\mathbf{v} = \mathbf{b}$
- the matrix A is called a *Vandermonde matrix*[3]

[3] Alexandre-Théophile Vandermonde, in papers on determinants in 1772

the Vandermonde matrix is a built-in

- actually, Vandermonde matrices are already built-in to MATLAB
- for example, the Vandermonde matrix A for our original four points $(-1, 2), (0, 3), (3, 4), (5, 0)$ is

```
>> vander([-1 0 3 5])
ans =
    -1      1     -1      1
     0      0      0      1
    27      9      3      1
   125     25      5      1
```

- two comments:
  - oops! the columns are in reversed order, compared to our choice
  - note that *only* the x-coordinates are needed to build A, and not the y-coordinates
- fix the column order to agree with our earlier choice using "`fliplr`" (flip left-to-right):

```
>> A = fliplr(vander([-1 0 3 5]))
A =
     1     -1      1     -1
     1      0      0      0
     1      3      9     27
     1      5     25    125
```

example of the Vandermonde matrix method

- a complete code to solve our 4-point problem:

```
A = fliplr(vander([-1 0 3 5]));
b = [2 3 4 0]';
v = A \ b
```

- after the coefficents `v` are computed, they form $P(x)$ this way:

$$P(x) = \mathtt{v(1)} + \mathtt{v(2)}\, x + \mathtt{v(3)}\, x^2 + \cdots + \mathtt{v(n)}\, x^{n-1}$$

- thus we can plot the 4 points and the polynomial this way:

```
plot([-1 0 3 5],[2 3 4 0],'o','markersize',12)
x = -2:0.01:6;
P = v(1) + v(2)*x + v(3)*x.^2 + v(4)*x.^3;
hold on,  plot(x,P,'r'),  hold off,  xlabel x,  ylabel y
```

- this was the graph shown a few slides back

example of `polyfit`

- actually we don't even need to call `vander` ourselves, because polynomial interpolation is built-in to MATLAB:

```
v = polyfit([-1 0 3 5], [2 3 4 0], 3)
```

- the only difference is that the coefficient order is reversed:

$$P(x) = \texttt{v(n)} + \texttt{v(n-1)}\, x + \dots + \texttt{v(1)}\, x^{n-1}$$

- plot using the "evaluate a polynomial" built-in called `polyval()`:

```
plot([-1 0 3 5],[2 3 4 0],'o','markersize',12)
x = -2:0.01:6;
P = polyval(v,x);
hold on,  plot(x,P,'r'),  hold off,  xlabel x,  ylabel y
```

Lagrange's idea (1795): no systems at all!

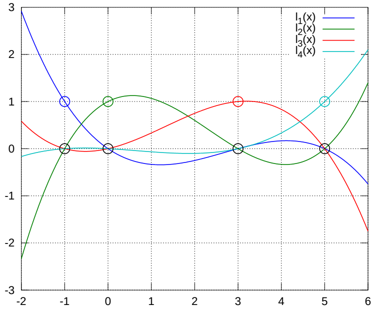
- a new idea, illustrated with the same points $(-1,2),(0,3),(3,4),(5,0)$
- Lagrange (and others) could directly write down four polynomials corresponding to the x-coordinates $x_1,\dots,x_4$:

$$\ell_1(x) = \frac{(x-x_2)(x-x_3)(x-x_4)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)} = \frac{x(x-3)(x-5)}{(-1)(-4)(-6)}$$
$$\ell_2(x) = \frac{(x-x_1)(x-x_3)(x-x_4)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)} = \frac{(x+1)(x-3)(x-5)}{(1)(-3)(-5)}$$
$$\ell_3(x) = \frac{(x-x_1)(x-x_2)(x-x_4)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)} = \frac{(x+1)(x)(x-5)}{(4)(3)(-2)}$$
$$\ell_4(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)} = \frac{(x+1)(x)(x-3)}{(6)(5)(2)}$$

- these are called the *Lagrange polynomials*
- the *pattern*:
  1. numerator and denominator have similar structure
  2. ...but the denominators are constant
  3. $\ell_i(x)$ has no "$(x-x_i)$" factor in the numerator
  4. $\ell_i(x)$ has no "(x_i-x_i)" factor in the denominator
  5. as long as the $\{x_i\}$ are distinct, we never divide by zero

Lagrange's idea: polynomials which "hit one point"

- consider a plot of $\ell_1(x)$, $\ell_2(x)$, $\ell_3(x)$, $\ell_4(x)$:



- a crucial pattern emerges:

  the polynomial $\ell_i(x)$ has value 0 at all of the x-values of the points, except that it is 1 at x_i

- MAKE SURE make sure you can find the Lagrange polynomials if I give you the x-values of n points
- but why is this helpful?

Lagrange's idea, cont.

- the picture on the last page illustrates what is generally true of the Lagrange polynomials:

$$\ell_i(x_j) = \begin{cases} 1, & j = i, \\ 0, & \text{otherwise.} \end{cases}$$

- so why does this help find $P(x)$?
- recall that we have values y_i which we want the polynomial $P(x)$ to "hit"
- that is, we *want* this to be true for each i:

$$P(x_i) = y_i$$

- *thus the answer is*:

$$P(x) = y_1\ell_1(x) + y_2\ell_2(x) + y_3\ell_3(x) + y_4\ell_4(x)$$

Lagrange's idea, cont.[2]

- *wait*, why is this the answer?:

$$P(x) \stackrel{*}{=} y_1\ell_1(x) + y_2\ell_2(x) + y_3\ell_3(x) + y_4\ell_4(x)$$

- because $P(x)$ *is* of degree three[4], and because:

$$\begin{aligned} P(x_1) &= y_1\ell_1(x_1) + y_2\ell_2(x_1) + y_3\ell_3(x_1) + y_4\ell_4(x_1) \\ &= y_1 \cdot 1 + y_2 \cdot 0 + y_3 \cdot 0 + y_4 \cdot 0 \\ &= y_1, \end{aligned}$$

  and

$$\begin{aligned} P(x_2) &= y_1\ell_1(x_2) + y_2\ell_2(x_2) + y_3\ell_3(x_2) + y_4\ell_4(x_2) \\ &= y_1 \cdot 0 + y_2 \cdot 1 + y_3 \cdot 0 + y_4 \cdot 0 \\ &= y_2, \end{aligned}$$

  and so on

---

[4]it is a linear combination of degree 3 polynomials

Lagrange's idea, cont.[3]

- on the last slide we saw that $P(x_i) = y_i$ because the polynomials $\ell_i(x)$ help "pick out" the point x_i in the general expression $*$ on the last slide
- we can say this more clearly using summation notation:
  - the polynomial is a sum of the Lagrange polynomials with coefficients y_i:

$$P(x) = \sum_{i=1}^{4} y_i \ell_i(x)$$

  - when we plug in one of the x-coordinates of the points, we get only one "surviving" term in the sum:

$$P(x_j) = \sum_{i=1}^{4} y_i \ell_i(x_j) = y_j \cdot 1 + \sum_{i \neq j} y_i \cdot 0 = y_j$$

returning to our 4-point example

- for our 4 concrete points $(-1,2),(0,3),(3,4),(5,0)$, we can slightly-simplify the Lagrange polynomials we have computed already:

$$\begin{aligned}\ell_1(x) &= -\frac{1}{24}x(x-3)(x-5)\\ \ell_2(x) &= +\frac{1}{15}(x+1)(x-3)(x-5)\\ \ell_3(x) &= -\frac{1}{24}(x+1)(x)(x-5)\\ \ell_4(x) &= +\frac{1}{60}(x+1)(x)(x-3)\end{aligned}$$

- so the polynomial which goes through our points is

$$\begin{aligned}P(x) = -(2)\frac{1}{24}x(x-3)(x-5) &+ (3)\frac{1}{15}(x+1)(x-3)(x-5)\\ -(4)\frac{1}{24}(x+1)(x)(x-5) &+ (0)\frac{1}{60}(x+1)(x)(x-3)\end{aligned}$$

- a tedious calculation simplifies this to

$$P(x) = 3 + \frac{59}{60}x - \frac{1}{15}x^2 - \frac{1}{20}x^3,$$

which is exactly what we found earlier

so, is the Lagrange scheme a good idea?

- for n points $\{(x_i, y_i)\}$ we have the following nice formulas which "completely answer" the polynomial interpolation problem:

$$\ell_i(x) = \prod_{j \neq i} \frac{x - x_j}{x_i - x_j}$$

$$P(x) = \sum_{i=1}^{n} y_i \ell_i(x)$$

- note "$\prod$" is a symbol for a product, just like "$\sum$" is a symbol for sum
- we solve no linear systems and we just write down the answer!
- is this scheme a good idea in practice?
  NOT REALLY!

so, is the Lagrange scheme a good idea?

- we have seen that using the Lagrange formulas to find $P(x)$ is ...awkward?
- the problem with the Lagrange form is that even when we write down the correct linear combination of Lagrange polynomials $\ell_i(x)$ to give $P(x)$, we do *not* have quick ways of getting:
  - the coefficients a_i in the standard (monomial) form:

$$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$

  - the values of the polynomial $P(x)$ at locations $\bar{x}$ in between the x_i:

$$P(\bar{x}) = \bar{y}$$

- generally-speaking, the output values of a polynomial are the desired numbers; this is the purpose of polynomial *interpolation*
- **moral**: sometimes a *formula* for the answer is less useful than an algorithm that leads to the numbers you actually want

conclusion: how to do polynomial interpolation

problem: find the degree $n-1$ polynomial $P(x)$ which goes through n given points (x_i, y_i)

- we have two methods:
    - the Vandermonde matrix method $\leftarrow$ built-in as `polyfit`
    - Lagrange's direct formula for the polynomial
- the Vandermonde linear system is easily solved in MATLAB
- Lagrange's direct formula requires us to simplify algebraically
- bigger issue addressed in Chapter 8:
    - *question*: how accurate is polynomial interpolation?
    - *answer*: the polynomial interpolation error theorem