How to put a polynomial through points

Ed Bueler

MATH 310 Numerical Analysis

September 2012

purpose

These notes are an online replacement for the 14 September class of Math 310 (Fall 2012), while Bueler is away.

*The topics here are also covered in Chapter 8 of the text (Greenbaum & Chartier). The emphasis here is on **how** to put a polynomial through points. When we get to Chapter 8 we will address the "how good" question.*

an example of the problem

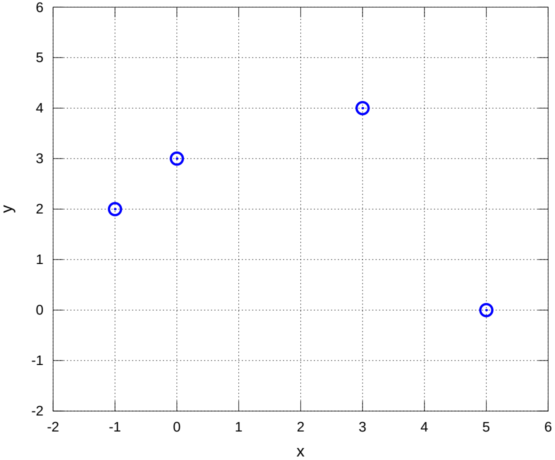
- suppose you have a function $y = f(x)$ which goes through these points:

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

- the x-coordinates of these points are not equally-spaced!
  - in these notes I will *never* assume the x-coordinates are equally-spaced
- let us name these 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
- as stated, we suppose that they are values of a function $f(x)$
- but we don't *see* that function



how to find $P(x)$

- $P(x)$ 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 "$P(x) = y$" for the given points gives 4 equations:

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

- 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
- we want to automate the process
- we have a great matrix-vector tool, namely MATLAB
- I recognize the system has a matrix form "$A\mathbf{v} = \mathbf{b}$":

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

- (a known square matrix A) $\times$ (an unknown vector $\mathbf{v}$) = (a known vector $\mathbf{b}$)
- I am not simplifying the numbers in the matrix ... because:
    - a machine can do that, and
    - the pattern in the matrix entries is clear if they are unsimplified
- MAKE SURE you can convert from the original "fit a polynomial through these points" question into the matrix form "$A\mathbf{v} = \mathbf{b}$"

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 $P(x) = 3 + 0.983333x - 0.066667x^2 - 0.05x^3$

notes on matrices and vectors in MATLAB

- 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, so MATLAB makes it into a single-character operation, the *backslash* operation:

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

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

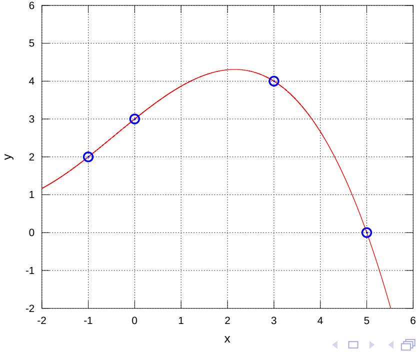
```
>> v = b / A   % NOT CORRECT for our A and b; wrong sizes
```

did we solve the problem?

- the polynomial we found had better 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
```

- a graph is convincing, too:



- 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*:

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

- written out there are 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, for the general case

- 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 $\mathbf{b}$ 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*, from about 1772

Vandermonde matrix, 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
- we easily fix the column order to agree with our earlier ordering using "`fliplr`", which stands for "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
```

Vandermonde matrix method for polynomial interpolation

- thus a complete code to solve our 4 point problem earlier is:

```
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:

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

- 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 "convincing" graph, shown earlier

on the cost of solving Vandermonde matrix problems

- now, often we want to do a polynomial fit problem like this *many times for different data*
- so, is it quick? here are some facts to know about solving these systems:
  - if there are n points then the matrix A has n rows and n columns
  - *internally in* MATLAB, the linear system $A\mathbf{v} = \mathbf{b}$ is solved by Gaussian elimination
  - Gaussian elimination does about $\frac{2}{3}n^3$ arithmetic operations (i.e. additions, subtractions, multiplications, divisions) to solve such a linear system
- so finding the coefficients of the polynomial $P(x)$ through n points takes about n^3 operations
- but then you need more operations to *evaluate* that polynomial, which is what you usually do with it

“new” idea: Newton’s form

- before Vandermonde there was already a good, practical idea
  - an old idea of Newton, perhaps about 1690
- the idea is to write the polynomial through the data $P(x)$ *not* using the “monomials” $1, x, x^2, x^3, \dots, x^{n-1}$,
- ...but instead to use a form of the polynomial which includes the x-coordinates of the data points:

$$\begin{aligned} P(x) = c_0 + c_1(x - x_1) + c_2(x - x_1)(x - x_2) + \\ \dots + c_{n-1}(x - x_1)(x - x_2)\dots(x - x_{n-1}) \end{aligned}$$

- do you see why this helps?

Newton's form example: 4 points

- with the $n = 4$ points $(-1, 2), (0, 3), (3, 4), (5, 0)$ we can write

$$P(x) = c_0 + c_1(x+1) + c_2(x+1)(x) + c_3(x+1)(x)(x-3)$$

- this polynomial must go through the four points, so:

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

- note that lots of terms are zero!
- the system of equations has the form

$$M\mathbf{w} = \mathbf{b}$$

where M is a triangular matrix, $\mathbf{b}$ is the same as in the Vandermonde form, and $\mathbf{w}$ has the unknown coefficients:

$$\mathbf{w} = [c_0\ c_1\ c_2\ c_3]'$$

Newton's form example, cont.

- can you solve this by hand?
- *yes*: find c_0 from first equation, then c_1 from second equation, etc.
- I get $c_0 = 2$, $c_1 = 1$, $c_2 = -1/6$, $c_3 = -1/20$, so

$$P(x) = 2 + (x+1) - \frac{1}{6}(x+1)(x) - \frac{1}{20}(x+1)(x)(x-3)$$

- MAKE SURE you can do this yourself, on a similar example

Newton's form example, cont.[2]

- so we have a concrete polynomial, but not in standard form:

$$P(x) = 2 + (x+1) - \frac{1}{6}(x+1)(x) - \frac{1}{20}(x+1)(x)(x-3)$$

- an uninteresting calculation puts it in standard form:

$$\begin{aligned} P(x) &= 3 + \frac{59}{60}x - \frac{1}{15}x^2 - \frac{1}{20}x^3 \\ &= 3 + 0.983333x - 0.066667x^2 - 0.05x^3 \end{aligned}$$

- which is exactly the same polynomial we found earlier

Newton's form for polynomial interpolation: example code

- the advantage of the Newton form is that a *triangular* matrix M is created
  - which makes it easier to solve the system by hand
  - only $O(n^2)$ operations are needed to solve the system
  - the polynomial comes out in a non-standard form but it is just as easy to evaluate at a point
- for now here is a short code to solve the 4 point problem:

newt4.m

```
% NEWT4  Compute P(x) using the Newton form, for 4 points.

n = 4;  x = [-1 0 3 5]';  y = [2 3 4 0]';  % the points

M = zeros(n,n);  % makes M the right size
% form M by columns
M(:,1) = ones(n,1);
for j=2:n
  M(j:n,j) = M(j:n,j-1) .* (x(j:n) - x(j-1));
end
b = y;

w = M \ b  % w has the coefficients of the polynomial:
% P(x) = w1 + w2 (x-x1) + w3 (x-x1) (x-x2) + w4 (x-x1) (x-x2) (x-x3)
```

Newton's form shows there is a unique interpolating polynomial

- for both Vandermonde and Newton matrix approximations we build an invertible matrix, so in each case there is exactly one solution
- this is easiest to see from the general Newton form matrix:

$$M = \begin{bmatrix} 1 & & & & \\ 1 & (x_2 - x_1) & & & \\ 1 & (x_3 - x_1) & (x_3 - x_1)(x_3 - x_2) & & \\ \vdots & \vdots & \vdots & \ddots & \\ 1 & (x_n - x_1) & (x_n - x_1)(x_n - x_2) & \dots & (x_n - x_1)(x_n - x_2)\dots(x_n - x_{n-1}) \end{bmatrix}$$

- the diagonal entries are all nonzero as long as the x-coordinates are distinct
- we can calculate the determinant of this Newton form matrix M
- because the matrix is triangular, the determinant is the product of the diagonal:

$$\det M = \prod_{i>j} (x_i - x_j) \neq 0$$

- so the polynomial $P(x)$ always exists and is unique

Lagrange's idea: no systems at all!

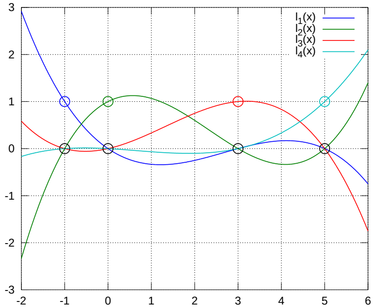
- another new idea
- given the same $n = 4$ points $(-1, 2), (0, 3), (3, 4), (5, 0)$
- Lagrange and others, by about 1800, knew how to write down four polynomials, now called the *Lagrange polynomials*, corresponding to the x-coordinates $x_1, \dots, x_4$:

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

- the *pattern* needs attention: **I.** the numerator and denominator have the same pattern, but the denominator is a constant with no variable "x"; **II.** $\ell_i(x)$ has no "$(x - x_i)$" factor in the numerator, nor "$(x_i - x_i)$" factor in the denominator; **III.** as long as the x_i are distinct, we never divide by zero

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

- why is this helpful?
- 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

Lagrange's idea, cont.

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

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

- also, the Lagrange polynomials for the 4 points are each of degree 3
- 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?:

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

- because $P(x)$ *is* of degree three, as a linear combination of degree 3 polynomials, 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

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:

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

- 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! *yeah*!
- is this scheme a good idea in practice?
  NOT REALLY!

so, is the Lagrange scheme a good idea? cont.

- we have seen that actually using the formulas to find a familiar form for $P(x)$ 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:
  - either the coefficients a_i in the standard form,
  $$P(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{n-1} x^{n-1}$$
  - or 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
- ... and we'll get back to that!

conclusion: how to do polynomial interpolation

- the problem is to find the degree $n-1$ polynomial $P(x)$ which goes through n given points (x_i, y_i)
- we have three methods, all of which do the job:
  - the Vandermonde matrix method,
  - the Newton polynomial form, and its triangular matrix method,
  - and Lagrange's direct formula for the polynomial
- the first two require solving linear systems, while the last does not
  - Lagrange's direct formula requires us to simplify like crazy
  - Newton gives easier linear systems (triangular) than does Vandermonde
  - MATLAB makes solving linear systems easy anyway
- later in the course:
  - we will address how accurate polynomial interpolation is
  - we will get to one more algorithm, Neville's algorithm, which gets the polynomial values but skips finding any coefficients