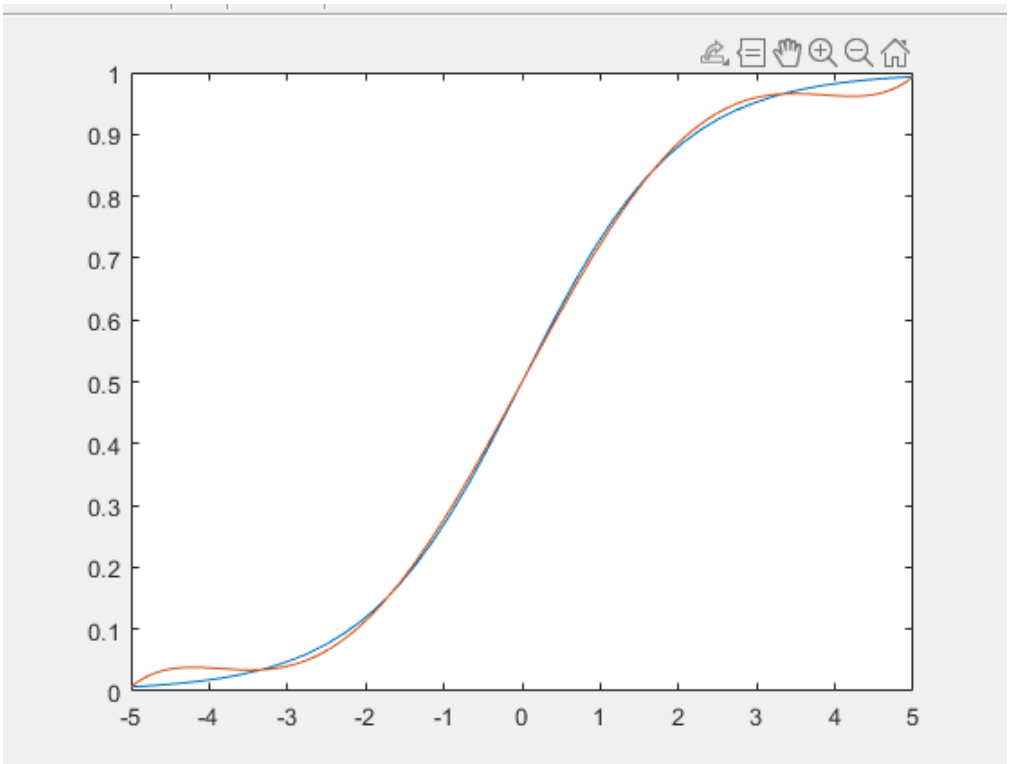
NumComp - Fall 2022
Project #6

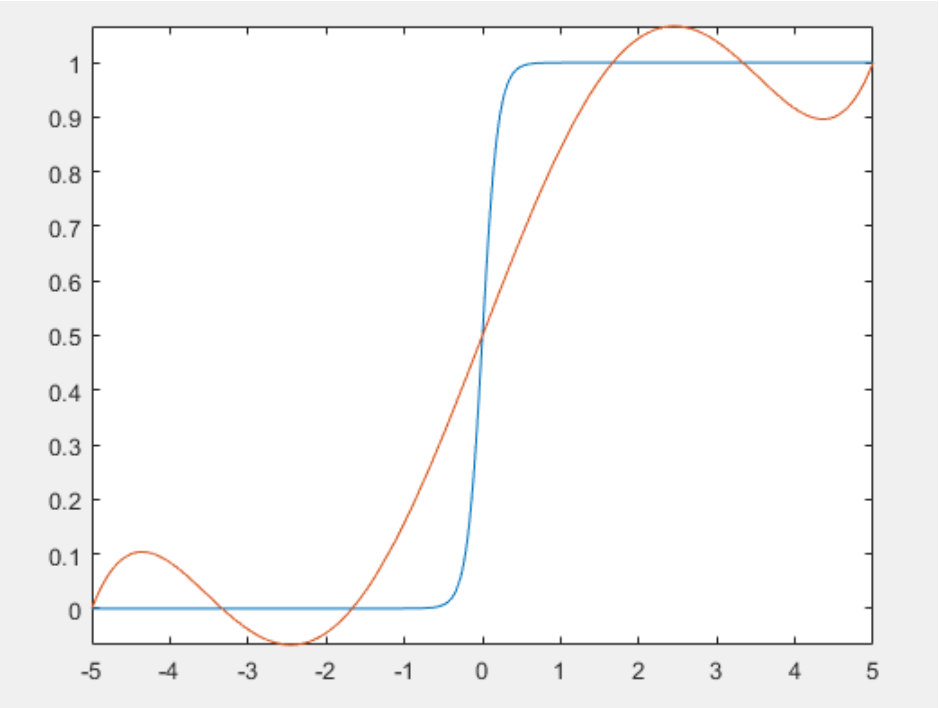
Due –
Isaiah Thomas

Interpolant plots

$\theta = 1$



$\theta = 10$



- For when $\theta = 1$, the interpolated approximation feels emotionally solid for the range we are working in. Area between curves is minimal, that is until the truth function curve begins to flatten. Then the approximate begins to show larger error.
- For when $\theta = 10$, just by sight of the plot, it is atrocious. Area between curves is largely visible and growing with x.

Truth and interpolant data

	A	B	C	D	E	F	G	H	I	J	K
1			truth x,y			Coeffients of interpolant				Largest error	
2											
3	theta 1		-5.0000	0.0067		0.5				2.2898	
4			-3.3333	0.0344		0.2331					
5			-1.6667	0.1589		0					
6			0	0.5000		-0.0108					
7			1.6667	0.8411		0					
8			3.3333	0.9656		0.0002					
9			5.0000	0.9933		0					
10											
11	theta 10		-5.0000	0.0000		0.5				6.31E+19	
12			-3.3333	0.0000		0.37					
13			-1.6667	0.0000		0					
14			0	0.5000		-0.027					
15			1.6667	1.0000		0					
16			3.3333	1.0000		0.0006					
17			5.0000	1.0000		0					
18											

Error analysis

- Differentiabiity seems to play a huge part on how accurate this method turns out. This makes sense as increasing polynomial degrees also increases possible differential values, which would increase accuracy on method results. Smooth function is interpolated well, aggressive function is not.
- As also mentioned in lecture, initial data affects method results intensely as well. For when theta is low and the rate of change is low, equidistant values of x will suffice. However for the when theta is large and the function has sudden large rates of change, equidistant points will likely fail to capture this change. Having the initial x values cluster where the differential is large will likely result in less error.

Code!

```
function pairs = gendata(npoints,range, theta)
%generates x y pairs based on truth function
    k= 1;
    step = (range(2)-range(1))/(npoints-1);
    pairs = zeros(npoints, 2);
    for x = range(1):step:range(2)
        y = thetaexp(theta,x);
        %disp(k);
        %disp(x);
        %disp(y);
        pairs(k,1) = x;
        pairs(k,2) = y;
        k = k + 1;
    end
end

```

```
function C = davanmon(pairs)
%generates coeffients of polynomial
    dim = size(pairs);
    x = transpose(pairs(:,1));
    y = pairs(:,2);
    vM = zeros(dim(1));
    disp(x);
    disp(y);
    for i = 1:1:dim(1)
        for j = 1:1:dim(1)

            vM(i,j) = x(i)^(j-1);

        end

    end

    C = vM\y;


end


```

```
function polyapprox = newpoly(coe)
%adds symbolic respective monomial x
%to previously calculated coeffients
    syms x;
    polyapprox = 0;
    for i = 1:1:length(coe)

        polyapprox = polyapprox + (coe(i) * x^(i-1));

    end
end

```

verror.m

```
function eout = verror(npoints,range, theta, poly)
% does the function listed on last page of the hw
    syms x;
    k= 1;
    step = (range(2)-range(1))/(npoints-1);

    curerror = zeros(npoints, 1);
    for i = range(1):step:range(2)
        y1 = thetaexp(theta,i);
        y2 = subs(poly, x, i);

        curerror(k) = (abs(y1 - y2)) / (abs(y1));
        k = k + 1;
    end
    disp(curerror);
    eout = max(curerror);
end

```