Homework #8

Tanner J. Evans

CS/Math 375-003

November 2, 2020

1. Approximate the integral $\int_0^2 f(x)dx$, given that $f(x) = \frac{1}{1+x^2}$

2. (a) The Vandermonde interpolant of the data will take the form of a polynomial of $n-1$ degree. The coefficients c_k are determined by n equations in a linear system, such that the input x_j yields the output y_j. In the system $Ac = b$, $b = y$, A is the set of linear equations whose individual values are determined by plugging in the associated value of x into the successive powers, and c is the set of coefficients to be solved for:

$$\begin{bmatrix} x_1^0 & x_1^1 & x_1^2 & \dots & x_1^{n-1} \\ x_2^0 & x_2^1 & x_2^2 & & \\ x_3^0 & x_3^1 & x_3^2 & & \\ \vdots & & & \ddots & \\ x_n^0 & & & & x_n^{n-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

(b) interpvan.m

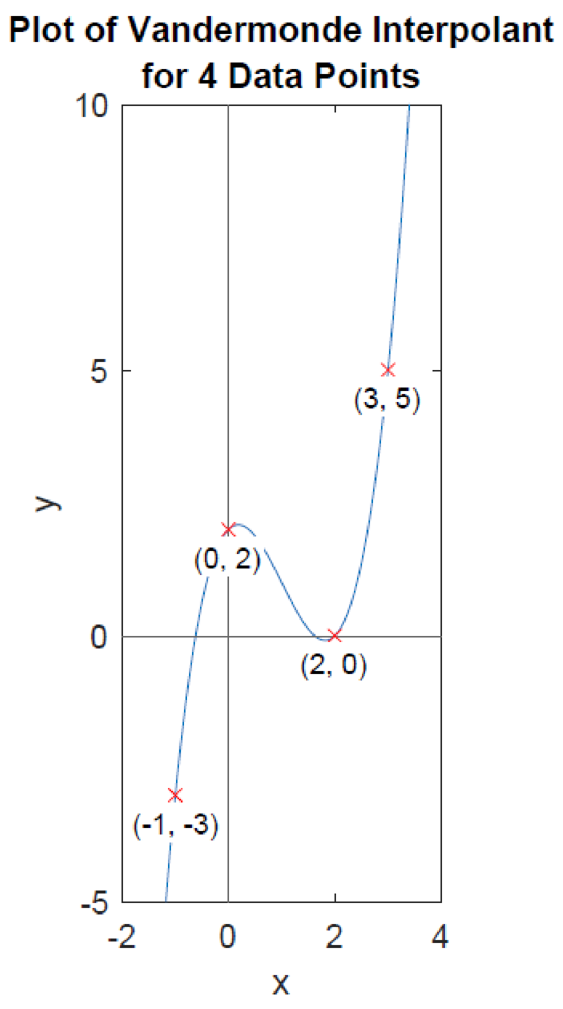
```
% function c = interpvan(x, y)
% hw8q1b
% Returns the column vector of coefficients c that results from inter-
% polation of input vectors x and y via the Vandermonde approach.
function c = interpvan(x, y)
    % Error checking.
    xDim = size(x);
    yDim = size(y);
    n = length(x);
    if (xDim(1) ~= 1 && xDim(2) ~= 1 || yDim(1) ~= 1 && yDim(2) ~= 1)
        error('Input vectors must be one-dimensional.')
    end
    if (yDim(1) == 1)
        y = y';
    end
    if (n ~= length(y))
        error('Input vectors must be of the same length.')
    end

    % Setting up result and working matrices.
    v = vandermonde_hw7q5b(x);
    [P, L, U] = LUpartialpivot(v);
    b = P*y;
    cL = ltrisol(L, b);
    x = utrisol(U, cL);
    c = x;
end
```

(c) hornervan.m

```
% function y = hornervan(c, z)
% Uses Horner's algorithm to evaluate a polynomial c at point z, assuming
% a polynomial derived via the Vandermonde interpolation approach.

function y = hornervan(c, z)
    y = c(length(c));
    for i = (length(c)-1) : -1 : 1
        y = y*z + c(i);
    end
end
```

(d) hw8q1d.m

```
x = [ -1 0 2 3 ];
y = [ -3 2 0 5 ];
c = interpvan(x, y);
t = linspace(-2,4);
z = hornervan(c, t);
plot(t, z);
axis equal
xlim([-2, 4])
ylim([-5, 10])
hold on
title({'Plot of Vandermonde Interpolant','for 4 Data Points'})
xlabel('x')
ylabel('y')
xL = xlim;
yL = ylim;
line([0 0], yL, 'Color', [.4, .4, .4]);
line(xL, [0 0], 'Color', [.4, .4, .4]);
for i = 1 : length(x)
    plot(x(i), y(i), 'Marker', 'x', 'MarkerEdgeColor', 'red')
    label = "(" + sprintf("%d", x(i)) + ", " + sprintf("%d", y(i)) + ")";
    text(x(i), y(i) - .5, label, 'FontSize', 9, 'HorizontalAlignment',
       ...
        'center', 'BackgroundColor', 'white', 'Margin', 0.01)
end
```



The polynomial for the Vandermonde interpolant for this data is $2 + x - 3x^2 + x^3$.

3. (a) The Newton approach to finding an interpolant involves filling out a lower-triangular matrix. The linear system will take the form:

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & (x_2 - x_1) & 0 & & \\ 1 & (x_3 - x_1) & (x_3 - x_1) * (x_3 - x_2) & & \\ \vdots & & & \ddots & \\ 1 & & & & \prod_{i=1}^{n-1}(x_n - x_i) \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

(b) `interpnew.m`

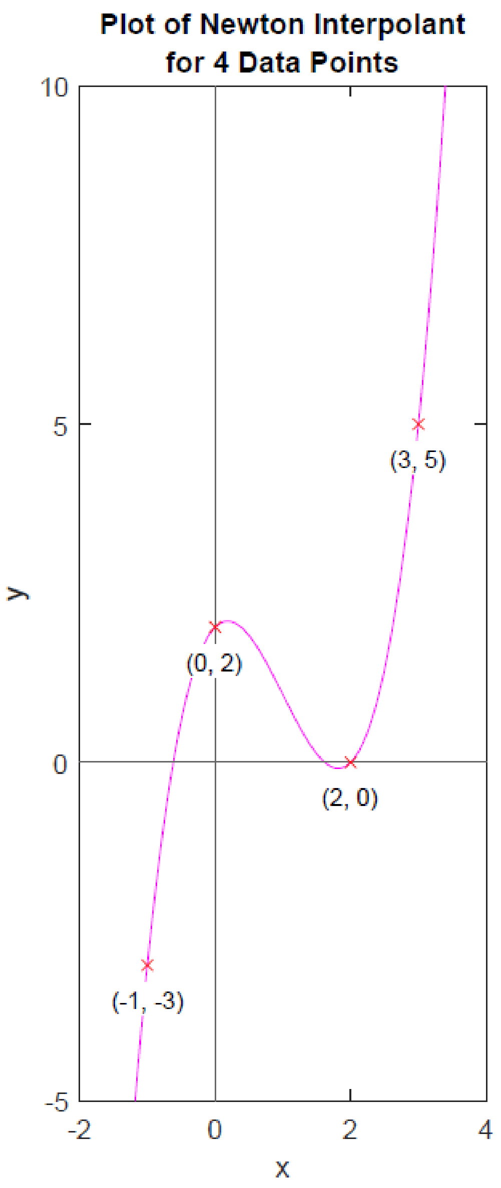
```
% function c = interpnew(x, y)
% hw8q2
% Returns the column vector of coefficients c that results from inter-
% polation of input vectors x and y via Newton's approach.
function c = interpnew(x, y)
    % Error checking.
    xDim = size(x);
    yDim = size(y);
    n = length(x);
    if (xDim(1) ~= 1 && xDim(2) ~= 1 || yDim(1) ~= 1 && yDim(2) ~= 1)
        error('Input vectors must be one-dimensional.')
    end
    if (yDim(1) == 1)
        y = y';
    end
    if (n ~= length(y))
        error('Input vectors must be of the same length.')
    end

    % Finding Newton interpolation matrix.
    A = zeros(n, n);

    for i = 1 : n
        A(i, 1) = 1;
        for j = 2 : i
            A(i, j) = (x(i) - x(j-1))*A(i, j-1);
        end
    end

    c = ltrisol(A, y);
end
```

(c) `hornernew.m`

```
% function y = hornernew(c, x, z)
% Uses Horner's algorithm to evaluate a polynomial c at point z, given a
% set of x's, assuming a polynomial derived via the Newton interpolation
% approach.

function y = hornernew(c, x, z)
    n = length(c);
    y = c(n);
    for i = (n-1) : -1 : 1
        y = y*(z-x(i)) + c(i);
    end
end
```

(d) `hw8q2d.m`

```
x = [ -1 0 2 3 ];
y = [ -3 2 0 5 ];
c = interpnew(x, y);
t = linspace(-2,4);
z = hornernew(c, x, t);
plot(t, z, 'Color', 'magenta');
axis equal
xlim([-2, 4])
ylim([-5, 10])
hold on
title({'Plot of Newton Interpolant','for 4 Data Points'})
xlabel('x')
ylabel('y')
xL = xlim;
yL = ylim;
line([0 0], yL, 'Color', [.4, .4, .4]);
line(xL, [0 0], 'Color', [.4, .4, .4]);
for i = 1 : length(x)
    plot(x(i), y(i), 'Marker', 'x', 'MarkerEdgeColor', 'red')
    label = "(" + sprintf("%d", x(i)) + ", " + sprintf("%d", y(i)) + ")";
    text(x(i), y(i) - .5, label, 'FontSize', 9, 'HorizontalAlignment',
        ...
        'center', 'BackgroundColor', 'white', 'Margin', 0.01)
end
```



The coefficients returned by this approach do not match the polynomial as it is drawn. The represent the singular solution to the Newton interpolation matrix. When data is plugged in, they yield algebraically equivalent values to the Vandermonde approach $(2 + x - 3x^2 + x^3)$, but the data is held as: $-3 + 5(x + 1) - 2(x + 1)(x) + 1(x + 1)(x)(x - 2)$. It is notable that in the evaluation of this equation at an x it is designed around, after a certain point the remaining terms yield 0.

4. (a) The Lagrange interpolant for the set of data used for the last two problems is a function can be represented

thus:

$$\begin{aligned}
p(x) &= c_1L_1(x)+c_2L_2(x)+c_3L_3(x)+c_4L_4(x)\\
&= c_1\frac{(x-x_2)(x-x_3)(x-x_4)}{(x_1-x_2)(x_1-x_3)(x_1-x_4)}+c_2\frac{(x-x_1)(x-x_3)(x-x_4)}{(x_2-x_1)(x_2-x_3)(x_2-x_4)}\\
&\quad +c_3\frac{(x-x_1)(x-x_2)(x-x_4)}{(x_3-x_1)(x_3-x_2)(x_3-x_4)}+c_4\frac{(x-x_1)(x-x_2)(x-x_3)}{(x_4-x_1)(x_4-x_2)(x_4-x_3)}\\
&= c_1\frac{(x)(x-2)(x-3)}{(-1)(-3)(-4)}+c_2\frac{(x+1)(x-2)(x-3)}{(1)(-2)(-3)}\\
&\quad +c_3\frac{(x+1)(x)(x-3)}{(3)(2)(-1)}+c_4\frac{(x+1)(x)(x-2)}{(4)(3)(1)}\\
&= -3\frac{x^3-5x^2+6x}{-12}+2\frac{x^3-4x^2+x+6}{6}+0\frac{x^3-2x^2-3x}{-6}+5\frac{x^3-x^2-2x}{12}
\end{aligned}$$

Just like the Newton approach, this polynomial perfectly matches the one produced by the Vandermonde approach, represented differently. It does not require solving a system of linear equations.

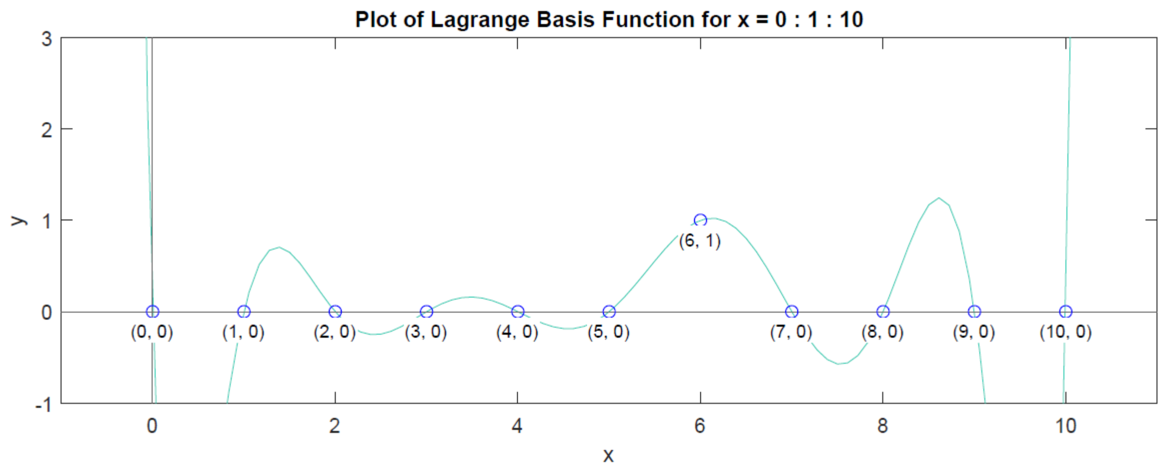
(b) We can show that these interpolants are algebraically equivalent:

$$\begin{aligned}
\text{Vandermonde Interpolant} &= 2+x-3x^2+x^3\\
\text{Newton Interpolant} &= -3+5(x+1)-2(x+1)(x)+1(x+1)(x)(x-2)\\
&= -3+5x+5-2x^2-2x+x^3-2x^2+x^2-2x\\
&= 2+x-3x^2+x^3\\
&= \text{Vandermonde Interpolant}\\
\text{Lagrange Interpolant} &= -3\frac{x^3-5x^2+6x}{-12}+2\frac{x^3-4x^2+x+6}{6}+0\frac{x^3-2x^2-3x}{-6}+5\frac{x^3-x^2-2x}{12}\\
&= \frac{1}{4}x^3-\frac{5}{4}x^2+\frac{6}{4}x+\frac{1}{3}x^3-\frac{4}{3}x^2+\frac{1}{3}x+\frac{6}{3}+\frac{5}{12}x^3-\frac{5}{12}x^2+\frac{10}{12}x\\
&= \frac{3}{12}x^3-\frac{15}{12}x^2+\frac{18}{12}x+\frac{4}{12}x^3-\frac{16}{12}x^2+\frac{4}{12}x+\frac{24}{12}+\frac{5}{12}x^3-\frac{5}{12}x^2+\frac{10}{12}x\\
&= \frac{12}{12}x^3-\frac{36}{12}x^2+\frac{12}{12}x+\frac{24}{12}\\
&= 2+x-3x^2+x^3\\
&= \text{Vandermonde Interpolant} = \text{Newton Interpolant}
\end{aligned}$$

5. For this question, we can use our Vandermonde or Newton approach to represent the Lagrange basis at a series of points:
`hw8q4.m`

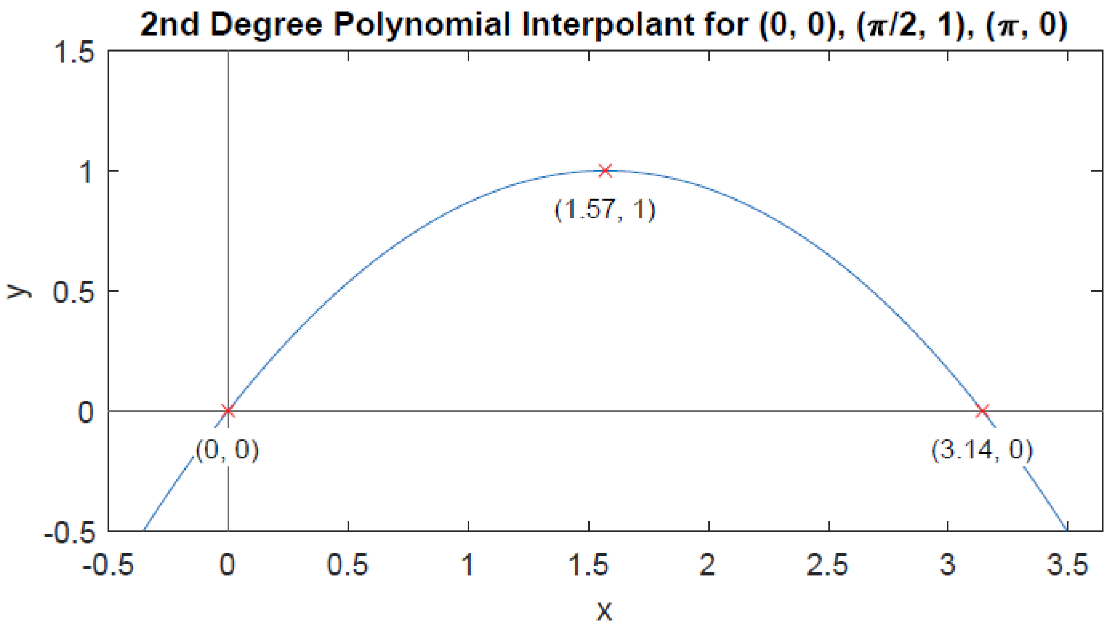
```
x = 0 : 1 : 10;
y = zeros(1, 11);
y(7) = 1;

c = interpvan(x, y);
t = linspace(-.5, 10.5);
z = hornervan(c, t);
plot(t, z, 'Color', [.2 .8 .7]);
hold on
title('Plot of Lagrange Basis Function for x = 0 : 1 : 10')
axis equal
xlim([-1, 11])
ylim([-1, 3])
xlabel('x')
ylabel('y')
xL = xlim;
```

```
yL = ylim;
line([0 0], yL, 'Color', [.4, .4, .4]);
line(xL, [0 0], 'Color', [.4, .4, .4]);
for i = 1 : length(x)
    plot(x(i), y(i), 'Marker', 'o', 'MarkerEdgeColor', 'blue')
    label = "(" + sprintf("%d", x(i)) + ", " + sprintf("%d", y(i)) + ")";
    text(x(i), y(i) - .2, label, 'FontSize', 9, 'HorizontalAlignment', ...
        'center', 'BackgroundColor', 'white', 'Margin', 0.01)
end
```



6. (a) hw8q5a.m

```
x = [ 0 pi/2 pi ];
y = [ 0 1 0 ];
c = interpvan(x, y);
t = linspace(-.5, pi+.5);
z = hornervan(c, t);
plot(t, z);
axis equal
xlim([-.5, pi+.5])
ylim([-.5, 1.5])
hold on
title('2nd Degree Polynomial Interpolant for (0, 0), (\pi/2, 1), (\pi, 0)
    ')
xlabel('x')
ylabel('y')
xL = xlim;
yL = ylim;
line([0 0], yL, 'Color', [.4, .4, .4]);
line(xL, [0 0], 'Color', [.4, .4, .4]);
for i = 1 : length(x)
    plot(x(i), y(i), 'Marker', 'x', 'MarkerEdgeColor', 'red')
    label = "(" + sprintf("%.3g", x(i)) + ", " + sprintf("%g", y(i)) + ")
        ";
    text(x(i), y(i) - .15, label, 'FontSize', 9, 'HorizontalAlignment',
        ...
         'center', 'BackgroundColor', 'white', 'Margin', 0.01)
end
```



The second degree interpolating polynomial for the provided points is $p_2 = \frac{4}{\pi}x - \frac{4}{\pi^2}x^2$.

(b)

$$\begin{aligned} p_2\left(\frac{\pi}{4}\right) &= \left(\frac{4}{\pi}\right)\left(\frac{\pi}{4}\right) - \left(\frac{4}{\pi^2}\right)\left(\frac{\pi^2}{4^2}\right) \\ &= 1 - \frac{1}{4} \\ &= \frac{3}{4} \end{aligned}$$

(c) Since this function approximates $\sin(x)$, we know that the derivative's absolute value will never be greater than 1. We can therefore approximate the maximum error of the interpolant via:

$$\begin{aligned} \left|f^{(x)} - p_n(x)\right| &\leq 1 * \left|\frac{(x-x_1)(x-x_2)(x-x_3)}{n!}\right| \\ &\leq \left|\frac{(\frac{\pi}{4})(-\frac{\pi}{4})(-\frac{3\pi}{4})}{6}\right| \\ &\leq \frac{\pi^3}{128} \\ &\leq 0.24224 \end{aligned}$$

(d) We can find our actual error, since we know the function we are approximating:

$$\begin{aligned} \text{error} = |f(x) - p_n(x)| &= \left|sin(x) - \left(\frac{4}{\pi}x - \frac{4}{\pi^2}x^2\right)\right| \\ &= \left|sin(\frac{\pi}{4}) - \left(\frac{4}{\pi} * \frac{\pi}{4} - \frac{4}{\pi^2} * \frac{\pi^2}{16}\right)\right| \\ &= \left|sin(\frac{\pi}{4}) - \left(1 - \frac{1}{4}\right)\right| \\ &= |-0.04289| \\ &= 0.04289 \end{aligned}$$

This is less than the maximum error we calculated, which is expected since the polynomial fits the sine curve very well.

7. For each of these problems, we can find the Chebyshev nodes via the following formula:

$$Ch_k = \frac{(x_n + x_1)}{2} + \frac{(x_n - x_1)}{2}\cos\left(\frac{\pi(2k-1)}{2n}\right)$$

I built a short matlab script to perform the calculations:
`cheb.m`

```
function points = cheb(low, high, n)
    points = zeros(1, n);
    for k = 1 : n
        points(k) = (high + low)/2 + ((high-low)/2)*cos(pi*(2*k-1)/(2*n));
    end
end
```

(a)

```
>> cheb(-1, 1, 6)
ans =
      0.96593      0.70711      0.25882     -0.25882     -0.70711
         -0.96593
```

(b)

```
>> cheb(-2, 2, 4)
ans =
       1.8478      0.76537     -0.76537      -1.8478
```

(c)

```
>> cheb(4, 12, 6)
ans =
       11.864       10.828       9.0353       6.9647       5.1716
          4.1363
```

(d)

```
>> cheb(-0.3, 0.7, 5)
ans =
      0.67553      0.49389          0.2    -0.093893     -0.27553
```

8. Using Chebyshev points for polynomial interpolation eliminates the Runge phenomenon, wherein the polynomial interpolant moves further and further away from the values between the sample points near the ends of the range. When choosing points via the Chebyshev process, the points bunch up near the ends of the range, minimizing errors between points.