1. (Polynomial interpolation, 10pts) Consider the following three samples of some function $f(x)$:

<table>
<thead>
<tr>
<th>$x$</th>
<th>$f(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1/2</td>
<td>-1/4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>-1/4</td>
</tr>
</tbody>
</table>

(a) Determine the unique second degree polynomial interpolant $p_2(x)$ to the above data using Lagrange’s interpolation formula. You do not need to simplify the result. You may either work this out by hand or simply write an anonymous function in $f$ or the polynomial.

(b) Repeat part (a) but use the Vandermonde matrix approach. In this case you will be solving for the coefficients $a_0$, $a_1$, and $a_2$ in the polynomial $p_2(x) = a_0 + a_1x + a_2x^2$ that interpolates the data. You can simply set up the linear system and solve the linear system in MATLAB. Be sure to print out the coefficients.

(c) Show that (a) and (b) give the same result. You can do this by numerically comparing the polynomials at several (i.e. 100) points over the interval $[-1, 1]$.

2. (Barycentric interpolation 15pts) As discussed in class, Lagrange’s interpolation formula can be rearranged into the barycentric formula (or more specifically the second (true) form of the barycentric formula):

$$p_n(x) = \frac{\sum_{j=0}^{n} w_j f_j}{\sum_{j=0}^{n} w_j},$$

where

$$w_j = \frac{1}{\prod_{i=0}^{n} (x_j - x_i)}, \quad j = 0, 1, \ldots, n.$$

Here $x_j$ are the given nodes and $f_j$ the corresponding function values. In this problem you will demonstrate the superiority of this method for high degree polynomial interpolation over the `polyfit` and `polyval` functions in MATLAB.

(a) Consider the function $f(x) = \sin(2\pi|x - 0.13|)$. Generate samples of this function at the following points: $x_j = -\cos(j\pi/n)$, $j = 0, 1, \ldots, n$, where $n = 55$. This gives a total of 56 values $(x_j, f(x_j))$. Download that `baryinterp` function from the course webpage, which implements the barycentric polynomial interpolation formula (1). Use this function to construct the 55 degree polynomial interpolant $p_{55}(x)$ to the data $(x_j, f(x_j))$, $j = 0, 1, \ldots, 55$, and evaluate it at the point $u_i = -\cos(i\pi/m)$, $i = 0, 1, \ldots, m$, where $m = 1000$. Produce a plot containing the exact function $f(x)$ and the polynomial interpolant evaluated at $u_i$. Set the limits on the y-axis of this plot to $[-1, 1]$.

(b) Repeat part (a), but use the `polyfit` and `polyval` functions to generate the polynomial interpolant. Remember to set the limits on the y-axis to $[-1, 1]$. 


Mathematically speaking the results of part (a) and part (b) should be identical. However, because we are using floating point arithmetic they are not. The results from part (a) are much better because the barycentric formula for computing the interpolating polynomial is numerically stable, whereas the algorithm behind `polyfit` (which solves a linear system using Vandermonde matrices) is numerically ill-conditioned.

3. (Approximation theory, 25pts) A common practice in numerical computing is to replace complicated functions (ones that involve expensive function evaluations, for example) with simple ones that approximate the function to some specified tolerance (usually machine precision). The simplest functions, both in representation and efficient computational manipulations, are polynomials. Can we approximate a given function with a polynomial to a specified precision? This is one of the central questions of a fundamental area of mathematics called approximation theory.

In Calculus you learned that a function $f(x)$ can be approximated as a Taylor series, assuming it has enough derivatives. The Taylor series of a function about a point $c$ is

$$f(x) = f(c) + (x - c)f'(c) + \frac{(x - c)^2}{2!}f''(c) + \frac{(x - c)^3}{3!}f'''(c) + \cdots.$$  

Truncating this infinite series expansion after $n + 1$ terms gives a polynomial approximant of degree $n$. One disadvantage of this approach is that it may be difficult (or impossible) to generate the needed derivatives of $f$. Another (as you will see below) is that the approximation may only be good in a limited neighborhood around $x = c$.

An alternative approach to generating a polynomial approximants of $f$ is to use polynomial interpolation at a suitably chosen set of points over the interval one wants the approximation to be valid.

In this problem you will experiment with these approaches. In particular, for the polynomial interpolation approach you will experiment with the following three node sets, which go by the names equispaced, Chebyshev, and Legendre points, respectively:

(i) \[ x_j = -1 + \frac{2j}{14}, \quad j = 0, 1, \ldots, 14 \]

(ii) \[ x_j = -\cos\left(\frac{j\pi}{14}\right), \quad j = 0, 1, \ldots, 14 \]

(iii) -0.987992518020485 -0.394151347077563 0.570972172608539  
-0.937273392400706 -0.201194093997435 0.724417731360170  
-0.848206583410427 0 0.848206583410427  
-0.724417731360170 0.201194093997435 0.937273392400706  
-0.570972172608539 0.394151347077563 0.987992518020485.

There are 15 points in each set, so you will be generating polynomial interpolants of degree 14. Note that all of these interpolants must be computed using the `baryinterp` function from the course webpage.

(a) Consider the function $f(x) = e^{3.2x}$. Generate the Taylor series polynomial approximation to $f$ of degree 14 about $c = 0$. Evaluate the approximation at 201 equally spaced points between $[-1, 1]$ (generated using `linspace(-1, 1, 201)`) and compute the error in the approximation at these points. Plot the error vs. these 201 equally spaced points. Report the 1-norm, 2-norm, and $\infty$-norm of the error in a nice table. (Hint: You may want to use the function `polyval` to evaluate the Taylor polynomial once you determine the coefficients.)

(b) Repeat part (a), but using polynomial interpolation at each of the three point sets (i), (ii), and (iii) given above. You should produce three different plots and three tables with results for the 1-norm, 2-norm, and $\infty$-norm associated with each point set.
(c) Which technique produced the best results from (a) and (b)? Explain the criteria you used to determine what “best” means.

(d) **Extra credit (5 points)** Chebfun is a software system for computing with functions in MATLAB (see http://www.chebfun.org). The key technology employed by Chebfun is interpolation at Chebyshev points. Download Chebfun and install it in . Use it to construct a function approximation to \( f(x) = e^{3x} \). From this construction determine the degree of the polynomial that can be used to approximate \( f \) to machine precision. Use the Chebfun function `minimax` to generate the “best” approximation to \( f \) of degree 14. Plot the error in this approximation together with the error in the 14th degree polynomial interpolant of \( f \) at the Chebyshev points from part (b). The best approximation in this case is the one that minimizes the maximum absolute difference between the function and the polynomial (called the max-norm). These are optimal and are used in many libraries for computing special functions.

4. **(Barycentric trigonometric interpolation, 30pts)** The problem of interpolating a periodic function from samples at equally spaced points over its period arises in many disciplines and dates back to at least Gauss’s work in the early 1800s on analyzing the orbits of certain asteroids. The standard approach for constructing an interpolant is to use the discrete Fourier Transform (DFT), or the fast algorithm for computing it known as the FFT (which Gauss also discovered). In this problem you will explore a different method for constructing periodic interpolants based on **barycentric interpolation**.

Let \( t_k = 2\pi k/N, \quad k = 0, 1, \ldots, N - 1 \) be equally spaced locations over \( 0 \leq t < 2\pi \) and \( f_k, \quad k = 0, 1, \ldots, N - 1 \), denote samples of some 2\(\pi\)-periodic function \( f(t) \) taken at \( t_k \). In this problem we will only consider odd values of \( N \), so we set \( N = 2n + 1 \), for some integer \( n \geq 0 \). For odd \( N \), the barycentric trigonometric interpolant to the data \( (t_k, f_k), \quad k = 0, \ldots, N - 1 \), is given as

\[
p_n(t) = \sum_{k=0}^{N-1} (-1)^k f_k \csc \left( \frac{t - t_k}{2} \right) \sum_{k=0}^{N-1} (-1)^k \csc \left( \frac{t - t_k}{2} \right).
\]  

(2)

If \( t \) happens to be exactly equal to a grid point \( t_j \), then we simply set \( p_n(t_j) = f_j \). This formula (2) equivalent to the trigonometric expansion of the form

\[
p_n(t) = \sum_{k=0}^{n} a_k \cos(kt) + \sum_{k=1}^{n} b_k \sin(kt)
\]

that interpolates the given data, i.e. \( p_n(t_j) = f_j, \quad j = 0, \ldots, N - 1 \). The parameter \( n \) is known as the **degree** of the trigonometric interpolant.

(a) Write a MATLAB function `trigbaryinterp` that implements the barycentric trigonometric interpolation formula (2). Your function should take as input a vector containing the function samples \( f_k, \quad k = 0, \ldots, N - 1 \), and a vector containing \( m \) locations where the interpolant is to be evaluated. Below is a possible skeleton for the function:

```
function p = trigbaryinterp(f,t)
    % Determine the number of sample points
    N = numel(f);
    % Only odd numbers of samples allowed:
    if mod(N,2) == 0
        error('TRIGBARYINTERP:oddPoints','Number of samples must be odd.');
    end
```
% Generate the sample points
k = (0:N-1)';
tk = 2*pi*k/N;

% Get the number of entries in t
m = numel(t);

% Initialize values to store the trig polynomial
p = zeros(size(t));

% Loop over the sample point values
for i=1:m
   %
   % Your code goes here to implement the barycentric formula.
   %
end
end

You may want to look at the baryinterp function for guidance in implementing this function.

(b) Verify that your code works by applying it to the function \( f(t) = \sin(2t)\cos(3t) \) for \( N = 13 \) (i.e. \( n = 6 \)). You can do this by first generating \( t_k = 2\pi k/13 \), and \( f_k = f(t_k) \), \( k = 0, \ldots, 12 \). Then use your function to evaluate the trigonometric interpolant \( p_6 \) to this data at \( m = 121 \) equally spaced points over \([0, 2\pi]\). Finally, make a plot of the error \( p_6(t) - f(t) \) vs. \( t \) at these 111 points. If everything is working correctly, the maximum absolute value of the error should be less than \( 4 \times 10^{-15} \).

(c) When the underlying function \( f \) that is being sampled is smooth and \( 2\pi \)-periodic, the trigonometric interpolant provides excellent accuracy, even better than interpolation at the Chebyshev points. As an example, consider the function \( f(t) = \cos(20e^{\sin(t-1/2)}) \). Generate \( N = 101 \) equally spaced samples of this function over \([0, 2\pi]\) and interpolate these samples to \( m = 1001 \) equally spaced points over \([0, 2\pi]\) (similar to part (a)). Make a plot of the error in the interpolant at these points vs. \( t \). Repeat this computation, but using \( N = 101 \) Chebyshev points and the baryinterp function you used in problems 2 and 3 above to generate an error plot (use the same \( m = 1001 \) equally spaced evaluation points). Compare the two approximations. small

Note: You need to generate the Chebyshev points over the interval \([0, 2\pi]\). This can be done as

\[ x_k = \pi \left( 1 + \cos \left( \frac{k}{N-1} \pi \right) \right), \quad k = 0, \ldots, N - 1 \]

(d) Consider now the non-smooth \( 2\pi \)-periodic square wave:

\[
 f(t) = \begin{cases} 
 1 & \text{if } 0 \leq t < \pi, \\
 -1 & \text{if } \pi \leq t < 2\pi, \\
 f(t + 2\ell\pi) = f(t), & \text{else}
\end{cases}
\]

where \( \ell \) is any integer. This function can be computed using the MATLAB function square, or simply using

\[ f = @(t) 1 - 2*\text{double}(\text{mod}(t, 2*\pi)) > \pi). \]

Generate \( N = 101 \) equally spaced samples of this function over \([0, 2\pi]\) and interpolate these samples to \( m = 10001 \) equally spaced points over \([0, 6\pi]\) (note the larger interval).
Plot the interpolated values (not the error) vs. \( t \) together with the true value of the square wave. Your plot should look similar to that shown in Figure 1.

The overshoots around the points of discontinuity of \( f \) is known as the Gibbs’ Phenomenon and is a fundamental problem in signal processing. The amplitude of the overshoots will not decrease with increasing \( N \), but will instead approach the fixed constant \( 1.281140725187307 \ldots \). Report the maximum interpolated value from your code to see how close your results are to this constant.

5. **(Piecewise cubic Hermite interpolation, 15pts)** Union Pacific has decided to reopen the Boise Train Depot for rail travel. They have contracted your computational science consulting company to design a new switching path between two of the rail lines (Track A and B) that enter the Depot. The requirements for the path are that it pass through the points \((0, 0)\), \((9, 3)\), and \((18, 6)\) (see Figure below). Furthermore, the path should be tangent to the line \( y = 0 \) at \((0, 0)\), tangent to the line \( y = 6 \) at \((18, 6)\), and have a slope of \(21/23\) at \( x = 9\).

(a) Since you are given both the function and derivative values at each of the node points, you decide to use a piecewise cubic Hermite polynomial to model the switching path. Construct the piecewise polynomial for the intervals \([-2, 0], [0, 9], [9, 18], [18, 20]\) (see Section 3.3 of the NCM book) and report the coefficients of the polynomial for each interval.

(b) Use the `mkpp` function in to make a piecewise polynomial representation for the train track.

(c) Use `ppval` with the piecewise polynomial function you created from (b) to make a nice smooth plot of the solution from part (a) for Union Pacific. If you did everything correctly, the value of the piecewise polynomial at \( x = 12 \) should be \(1034/207\).