1. In this problem, you will write a multigrid solver for the linear system of equations generated by \texttt{fempoi}, \texttt{pmesh} from problem set 6. Note that in 2-D it is very hard to be faster than the built-in backslash function, at least without using a compiled language. However, for large problems in 3-D, any multigrid solver should be superior to Gaussian elimination, so here we are more concerned about getting the right convergence behavior rather than a fast solver. To begin with, add two output arguments to the \texttt{fempoi} function to get access to the matrices \( A, b \) in the linear system:

\[
\text{function } [u,A,b]=\text{fempoi}(p,t,e)
\]

a) Write a MATLAB function

\[
\text{function } [u,res]=\text{gauss\_seidel}(A,b,u,niter)
\]

that makes \texttt{niter} iterations using the Gauss-Seidel method for \( Au = b \):

\[
u_{m+1} = u_m + (D - L)^{-1}(b - Au_m),
\]

starting from the input \( u \) and returning the last iterate \( u \). \( D - L \) is the lower triangular part of \( A \), including the diagonal (see the \texttt{tril} command). The output \texttt{res} is a vector of length \texttt{niter} + 1 with the infinity norms of the residuals \( b - Au_m \) at each iteration (including the initial and the final iterates). Try the function using the commands:

\[
\begin{align*}
\text{pv} & = [0,0;2,0;1.5,1;5,1;0,0]; \\
[p,t,e] & = \text{pmesh(pv,0.5,3)}; \\
[u0,A,b] & = \text{fempoi(p,t,e)}; \\
[u,res] & = \text{gauss\_seidel(A,b,0*b,1000)}; \\
\text{semilogy}(0:1000,res)
\end{align*}
\]

b) Write a MATLAB function

\[
\text{function } \text{data=}\text{mginit(pv,hmax,nref)}
\]

that computes all the required arrays for a multigrid solution of the Poisson problem using the mesh parameters \( \text{pv}, \text{hmax}, \text{nref} \). Start from the \texttt{pmesh} function, and make appropriate modifications and additions.

(a) \texttt{data(i).p.data(i).t.data(i).e} contain the mesh arrays \( p, t, e \) after \( i - 1 \) refinements, for \( i = 1, \ldots, \text{nref} \).

(b) \texttt{data(i).T} contains the interpolation matrix \( T^{(i)} \) from grid \( i \) to grid \( i + 1 \), for \( i = 1, \ldots, \text{nref} \). Use linear interpolation for all the new midpoints (that is, averaging of the neighboring nodes). The second output argument of \texttt{unique} might be useful.

(c) \texttt{data(i).R} contains the restriction matrix \( R^{(i)} \) from grid \( i + 1 \) to grid \( i \). Use the transpose of \( T^{(i)} \), but with the rows scaled to have sums of 1.

(d) \texttt{data(nref+1).A.data(nref+1).b} contain \( A, b \) for the finest grid (the actual linear system)

(e) \texttt{data(i).A} contains the projected matrices \( A^{(i)} = R^{(i)} A^{(i+1)} T^{(i)} \) for \( i = 1, \ldots, \text{nref} \).

c) Write a MATLAB function

\[
\text{function } [u,res]=\text{mgsolve(data,vdown,vup,tol)}
\]
that solves the problem precomputed in \texttt{data}, using multigrid V-cycles with \texttt{vdown/vup}
pre/post-smoothing iterations using Gauss-Seidel, until the infinite norm of the residual
is less than \texttt{tol}. The outputs are the solution \texttt{u} and the residuals \texttt{res} after each V-cycle
(including the residual for the initial solution \texttt{u} = 0).

Test the function using the commands

\begin{verbatim}
pv=[0,0;2,0;1.5,1;5,1;0,0];
for iref=1:5
    data=mginit(pv,0.5,iref);
    [u,res]=mgsolve(data,2,2,1e-10);
    semilogy(res),hold on
end
hold off
\end{verbatim}

If everything works correctly, you should see a very fast convergence compared to pure
Gauss-Seidel. More importantly, the number of iterations should not increase much when
the grid is refined. This leads to the optimal \( O(n) \) computational cost of the algorithm.

2. a) Modify the \texttt{dg1.m} function on the course web page to handle general polynomial orders \( p \).
Call the new function \texttt{dg2.m}. You can create the elementary matrices by computing a
Vandermonde matrix, and then either working with the MATLAB polynomial commands \texttt{polyder}, \texttt{polyint}, \texttt{conv}
or using Gaussian quadrature.

b) Write a MATLAB function

\begin{verbatim}
function [e,slopes]=dg2conv
    \end{verbatim}

that runs your function \texttt{dg2.m} using \( n = 5, 10, 20, 40 \), \( p = 1, 2, 3, 4, 5 \), \( \Delta t = 10^{-3} \), and
\( T = 1 \), and computes the infinity norm of each error. Note that the exact solution is
equal to the initial solution. Return the errors in the 5-by-4 array \texttt{e}, and estimate 5
slopes in the array \texttt{slopes}. Test the function using the commands:

\begin{verbatim}
    [e,slopes]=dg2conv
    loglog(1./[5,10,20,40],e)
\end{verbatim}

Code Submission: E-mail all requested and supporting MATLAB files to David at
\texttt{anderson@math.berkeley.edu} as a zip-file named \texttt{lastname_firstname_7.zip}, for example
\texttt{anderson_david_7.zip}. 