Instructions: Create a new MATLAB .m for each part. Each .m file should include comments that indicate problem number and names of the group members. Choose descriptive variable names and use comments as needed to describe your solutions.

Each group should hand in a copy of your .m files and results for the problems below to the TA in room G255 by 4 PM Friday on 10/15/04. No late submissions will be accepted.

Objectives: Simulating signals and systems; using difference equations and the MATLAB conv function

1. Population simulation: The population system model discussed in class was described by the difference equation

   \[ y[n] = (1 + p) y[n-1] + u[n] \]

   where \( y[n] \) is population at time index \( n \), \( p \) is the population growth rate per time step, and \( u[n] \) represents the population added at time \( n \).

   Create a new MATLAB function \texttt{popsim} (for population simulation) in file \texttt{popsim.m}. As shown below, \texttt{popsim} should take five arguments and return a vector representing the population \( y[n] \). The first part of the definition for \texttt{popsim} should look like:

   ```matlab
   function y = popsim(yn1, p, u)
   \%
   \% Population simulation function. y = popsim(yn1, p, u)
   \%
   \% Simulates a population with growth factor p with
   \% initial condition yn1 (for y[n - 1]) and input u.
   \% This function returns an output vector that is the same
   \% length as u with the corresponding population values.
   \%
   \% For example, if the input vector u is of length 3 and
   \% the first element of u corresponds to time n0, then
   \% the initial condition yn1 gives the population at
   \% time n0-1. The output vector will be length 3,
   \% and the first value in the output gives the population
   \% at time index n0, the second value gives the population
   \% at index n0+1, and the third value contains the population
   \% at index n0+2.
   \%
   
   You should compute \( y[n] \) by evaluating the system difference equation with input \( u \) and initial condition \( yn1 \). Hand in a printout of your \texttt{popsim.m} file.

Use the \texttt{stem} command and hand in plots of your simulated \( y[n] \) for these difference conditions:

(a) Find the system impulse response for \( p=0.25 \) by running the simulation with \( n0=0, n1=20, u[n] = \delta[n] \), and \( yn1=0 \) (relaxed system). This is the case of at 25% population growth per time index. Hand in a plot of the system response plotted versus time index.

(b) Compare the result from (a) to the case with \( u[n] = q[n] \). This will give the step response.

(c) Repeat (a) for \( p=-0.25 \), which models a 25% population loss each time index.

(d) Repeat (b) for \( p=-0.25 \), which models a 25% population loss each time index.
2. MATLAB has a built-in convolution function called \texttt{conv}. \texttt{conv} takes two input vectors and returns a new vector that represents their convolution.

Start with the impulse response $h[n]$ you found above in part 1(a). Use the \texttt{conv} function to compute the system output for a step input. (See the comments on the next page that describe the need to truncate the output of the \texttt{conv} function.) Hand in a MATLAB .m called \texttt{doconv.m} that shows your solution. Hand in a plot of the \texttt{conv} function output for $n = 0$ to $n = 20$. 
Handy MATLAB Tips

• How to use the for loop:

```matlab
for i = 1 : 10
    j = 3 * i;
end
```

• Making a 30-element vector of zeros:

```matlab
Z = zeros(30);
```

• Making a new vector of zeros that is the same size as another vector A:

```matlab
Z = zeros(size(A));
```

• How to use the conv function. conv takes two vector arguments. Example of how to use conv to convolve impulse response h with input u:

```matlab
h = [ 1, 0.5, 0.25, 0.125, 0.0625, 0.03125 ];
u = [ 1, 1, 1 ];
y = conv(h, u);
```

Notice that the output vector y is of length 7 for this case (the length of h plus the length of u minus 1). MATLAB assumes that the input vectors are zero outside the ranges defined by the user, and performs the convolution in a way that gives a result similar to the graphical convolutions we discussed in class. This would make sense if the input u was intended to represent a pulse that was non-zero only for \( n = 0, 1, \) and \( 2 \).

If we want to use the conv function to compute the output for impulse responses or input signals that are infinite in length (like the step function), we should discard the output of the conv function beyond the time index we are interested in. For example, if we want to use conv to compute the step response for the system given above for \( n = 0, \ldots, 5 \), we should do it this way:

```matlab
h = [ 1, 0.5, 0.25, 0.125, 0.0625, 0.03125 ];
u = [ 1, 1, 1, 1, 1, 1 ];
yorig = conv(h, u);
y = yorig(1:6);
```

In this case we are truncating the output of the conv function after the first six sample values because they were computed based on the assumption that the input u is zero after \( n = 5 \), which is not the case for the step input. To see this difference explicitly, try this demo:

```matlab
h = [ 1, 0.5, 0.25, 0.125, 0.0625, 0.03125 ];
u1 = [ 1, 1, 1 ];
u2 = [ 1, 1, 1, 1, 1, 1 ];
y1 = conv(h, u1);
y2 = conv(h, u2);
figure(1);
stem(y1);
figure(2);
stem(y2);
```

Notice that only the first three values of \( y_1 \) and \( y_2 \) match. Remember, \( y_1 \) is computed based on the assumption that \( u_1(n) \) is zero for \( n > 3 \).