Complete this exercise and submit your code to CMS by Sunday, 5/10, at 9pm EDT. At the top of your submitted files, add comments to answer each of the boxed questions on this worksheet.

1. Efficient calculation of \( x^n \) where \( n \) is large

If you cannot use MATLAB’s power operator \(^\) how would you calculate \( x \) to the \( n \)-th power? One way is to use iteration—a loop that executes \( n - 1 \) times. Another strategy is recursion—repeated squaring in this case. The idea is illustrated with the following schematic that shows how to compute \( x^{21} \):

\[
\begin{align*}
x^{21} &= (x^{10})^2 \cdot x \\
x^{10} &= (x^5)^2 \\
x^5 &= (x^2)^2 \cdot x \\
x^2 &= (x^2)^2
\end{align*}
\]

The recursive definition behind the scenes is given by

\[
f(x, n) = \begin{cases} 
1 & \text{if } n = 0 \\
f(x, n/2) \cdot f(x, n/2) & \text{if } n > 0 \text{ and } n \text{ is even} \\
f(x, (n-1)/2) \cdot f(x, (n-1)/2) \cdot x & \text{if } n > 0 \text{ and } n \text{ is odd}
\end{cases}
\]

Write the following function based on the recursive strategy. Do not use loops.

\[
\begin{verbatim}
function y = Power(x, n)
  \% y = x^n where n is an integer >=0
  \% Use recursion. No loops or the ^ operator.
\end{verbatim}
\]

How many times will your function \texttt{Power()} be called in total if you evaluate the expression \texttt{Power(2,5)}? \hfill \boxed{0}

Check your answer by adding the statement \texttt{disp(‘Pow!’)} as the first line of your function. If you see “Pow!” printed more than 5 times, how could you modify your code to reduce the number of recursive calls? (still without using loops or ^)
2 Writing efficient code

1. Download the script `LargestTriangle` from the Exercises page. The script (also shown below) is a first attempt at finding the largest triangle that can be formed from \( n \) points on a unit circle. Add code (\texttt{tic}, \texttt{toc}) to the script to determine how long it takes to find the answer for \( n = 100, 150, 200 \). Store the results (time) in vector \( t1 \) such that \( t1(i) \) corresponds to \( n(i), i = 1, 2, 3 \). Print the values in \( t1 \).

   ```
   for n=100:50:200
   theta = rand(n,1)*2*pi; % Angle of n random pts on the unit circle
   % Compute the area of the largest triangle that can be formed by 3 of the n points
   A = 0; % max area so far
   for i=1:n
       for j=1:n
           for k=1:n
               % Triangle with vertices at points i,j,k. Calculate Cartesian coordinates
               ci = cos(theta(i)); si = sin(theta(i));
               cj = cos(theta(j)); sj = sin(theta(j));
               ck = cos(theta(k)); sk = sin(theta(k));
               % Calculate area using Heron's Formula
               dij = sqrt((ci-cj)^2 + (si-sj)^2); % distance btw points i,j
               dik = sqrt((ci-ck)^2 + (si-sk)^2); % distance btw points i,k
               djk = sqrt((cj-ck)^2 + (sj-sk)^2); % distance btw points j,k
               s = (dij+dik+djk)/2; Aijk = sqrt((s-dij)*(s-dik)*(s-djk)*s);
               A = max(A, Aijk);
           end
       end
   end
   end
   ```

2. We now start to make the computation more efficient. \textit{Append} the script rather than modify directly—copy and paste your code from Part 1 to Part 2 of the script and make the modification in Part 2.

   Notice that there are several levels of inefficiency. The area for each combination of \( i, j, k \) is computed 6 times.
   
   \begin{itemize}
   \item \textbf{Improvement 1:} Modify the loop ranges to eliminate \textit{duplicate} and improper combinations (e.g., \((1,3,2)\) is a duplicate of \((1,2,3); (1,1,2)\) is not a proper combination for forming a triangle.)
   \item Also, there are a lot of redundant sine and cosine evaluations. \textbf{Improvement 2:} Address this issue by moving the \texttt{ci}, \texttt{si}, \texttt{cj} and \texttt{sj} assignments.
   \end{itemize}

   Store the time taken to do the computation in vector \( t2 \) such that \( t2(i) \) corresponds to \( n(i) \). Print the values in \( t2 \). How much speed-up did you get?


   \begin{itemize}
   \item Even with the change in \textit{where} we compute \texttt{ci}, \texttt{si}, \texttt{cj} and \texttt{sj} as done in Part 2, we are still doing more sine and cosine evaluations than necessary—given \( n \) values of \texttt{theta} we should only need to make \( n \) sine evaluations and \( n \) cosine evaluations. This suggests that we can reduce the time further by \textit{precomputing} the sine and cosine of \texttt{theta}. \textbf{Improvement 3a:} Compute and store the \( n \) sine values in a vector; compute and store the \( n \) cosine values in a vector. You will use these later.
   \item There is a similar redundancy associated with the repeated side length computations \texttt{a}, \texttt{b}, and \texttt{c}. \textbf{Improvement 3b:} Eliminate this redundancy by \textit{precomputing} an \( n \times n \) array \( D \) with the property that \( D(i,j) \) is the distance from point \((\cos(\texttt{theta}(i)), \sin(\texttt{theta}(i)))\) to point \((\cos(\texttt{theta}(j)), \sin(\texttt{theta}(j)))\). Note that you only need to compute “half” of \( D \) since \( D(i,j) \) equals \( D(j,i) \). Be sure to use the \texttt{precomputed sine and cosine values} instead of making new calls to the functions \texttt{sin} and \texttt{cos}.
   \item \textbf{Improvement 3c:} Update the remaining code to \textit{use the precomputed distances instead of making extra side length calculations}. Store the time taken to do the computation in vector \( t3 \) such that \( t3(i) \) corresponds to \( n(i) \).
   \end{itemize}

4. Plot the computation time: draw three curves of time vs. \( n \) on one set of axes with a legend to identify the curves. (Read MATLAB documentation!) Also show in a table (just use \texttt{fprintf}) the ratio of \( t1 \) to \( t3 \) for all \( n \).

5. What is the expected computation time for the three methods for \( n = 1000 \)？

\textbf{Final note.} The speed-up that we get isn’t all “free.” The speed-up gained from precomputation has a cost in computer memory—from version 1 to version 3, the major memory requirement increases from \( n \) (length of \texttt{theta}) to \( n^2 \) (size of \( D \)). The problem at hand, the language, and the hardware are all considerations in the trade-off between speed and memory.