Problem 1

Euler’s sieve is a souped-up version of the sieve of Eratosthenes, which finds the prime numbers. It works as follows:

L = the list of numbers from 2 to N;
P = 2; /* The first prime */
while (P^2 < N) {
    L1 = the list of all X in L such that P <= X <= N/P.
    L2 = P*L1;
    delete everything in L2 from L;
P = the next value after P in L;
}
return L;

For instance, for N=27, successive iterations proceed as follows:

Initialization
L = [2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27]
P = 2

First iteration
L1 = [2 3 4 5 6 7 8 9 10 11 12 13]
L2 = [4 6 8 10 12 14 16 18 20 22 24 26]
L = [2 3 5 7 9 11 13 15 17 19 21 23 25 27]
P = 3

Second iteration
L1 = [3 5 7 9]
L2 = [9 15 21 27]
L = [2 3 5 7 11 13 15 17 19 23 25]
P = 5

Third iteration
L1 = [5]
L2 = [25]
L = [2 3 5 7 11 13 17 19 23]

A. Write a MATLAB function EulerSieve1(N) which constructs the Euler sieve, implementing L, L1, L2 as arrays of integers, as above.

B. Write a MATLAB function EulerSieve2(N) which constructs the Euler sieve, implementing L, L1, and L2 as Boolean arrays, where L[i] = 1 if i is currently in the set L. Thus, the final value returned in the above example would be the array

[0 1 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 1 0 0 0 1 0 0 0 0]
Problem 2:

There is a theorem that states that, if you carry out the following procedure:

\[ P = \text{any polygon (this can be concave or even cross itself)}. \]

\[ \text{loop} \{
\hspace{1cm} \text{compute the midpoint of each side of } P \\
\hspace{1cm} P = \text{the polygon formed by connecting these midpoints in sequence}; \\
\} \]

Then \( P \) will converge toward a series of points that lie on an ellipse. Picture on the next page.

A. Assume that \( P \) is represented as a \( 2 \times n \) matrix, where each column is the coordinates of one vertex of \( P \). For example, the polygon with vertices at \( (0,0), (2,8), (4,0), (-2,6), (6,6) \) would be represented as the array,

\[
\begin{bmatrix}
0 & 2 & 4 & -2 & 6 \\
0 & 8 & 0 & 6 & 6
\end{bmatrix}
\]

Write a MATLAB function \texttt{ConnectMidpoints(P)} that, given a polygon \( P \) constructs the polygon that results from connecting the midpoints of \( P \) in sequence. For instance if \( P \) is the matrix above then \texttt{ConnectMidpoints(P)} would return the array

\[
\begin{bmatrix}
1 & 3 & 1 & 2 & 3 \\
4 & 4 & 3 & 6 & 3
\end{bmatrix}
\]

Each column of \( Q \) is constructed by taking the average of two consecutive columns of \( P \) and dividing by 2; e.g. \( Q[:,1] = 1/2(P[:,1]+P[:,2]) \). The last column of \( Q \) is the average of the last and first column of \( P \); i.e. \( Q[:,1] = 1/2(P[:,5]+P[:,1]) \).

Your code should of course work for polygons with any number of points, not just polygons with 5 points.

B. Write a MATLAB function \texttt{ConvergingPolygons(P,N)} which takes as input a polygon \( P \) and a number \( N \) and draws pictures of the first \( N \) polygons in this sequence, starting with \( P \). Let MATLAB adjust the scale on each successive picture, or the picture will soon become too small to see. Also, as always with geometric drawings in MATLAB, call \texttt{axis equal} to make sure that the x and y axes have the same scale.

Note: To make multiple plots on a single figure, use \texttt{hold on} and \texttt{hold off}. To make multiple figure, use \texttt{figure()}. So the code inside the loop that generated each figure on the next page had the form

\[
\begin{align*}
\text{figure()} \\
h\text{old on } \\
\text{plot(A(1,:), A(2,:))} \\
\text{plot(B(1,:), B(2,:), '--')} \\
\text{axis equal} \\
h\text{old off}
\end{align*}
\]

where \( A \) and \( B \) are the appropriate matrices.
Output of ConvergingPolygons(P,6) with

\[ P = \begin{bmatrix} 1 & 5 & 8 & -4 & 6 & 2 \\ 2 & 7 & 16 & 10 & 3 & 9 \end{bmatrix} \]