The purpose of this project is to implement the subdivision version of the de Casteljau algorithm for approximating a Bézier curve by a polygonal line.

(1) Given a cubic Bézier curve $C$ specified by its control points $(b_0, b_1, b_2, b_3)$, for any $t$, the de Casteljau algorithm constructs points

\[
\begin{align*}
b_0^1, b_1^1, b_2^1 \\
b_0^2, b_1^2 \\
b_0^3
\end{align*}
\]

using the equations

\[
\begin{align*}
b_i^1 &= (1 - t)b_i + tb_{i+1} & i &= 0, 1, 2 \\
b_i^2 &= (1 - t)b_i^1 + tb_{i+1} & i &= 0, 1 \\
b_i^3 &= (1 - t)b_0^2 + tb_1^2 & i &= 0.
\end{align*}
\]

This process is conveniently depicted as follows.

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
b_0 &= b_0^0 & b_1^0 & b_2^0 & b_3^0 \\
b_1 &= b_1^1 & b_0^2 & b_1^2 & b_2^3 \\
b_2 &= b_2^2 & b_1^3 \\
b_3 &= b_3^3
\end{array}
\]

Then the point $C(t)$ is given by

\[C(t) = b_0^3.\]

The red cubic curve is tangent to the line segment $(b_0^2, b_1^2)$ at $b_0^3$; see Figure 1.

It turns out that the two sequences of points

\[ud = (b_0, b_0^1, b_0^2, b_0^3)\]

and

\[ld = (b_0^3, b_1^2, b_2^1, b_3)\]
are also control points for the curve $C$; see Figure 1.

Thus we can iterate the above method using the control points in $ud$ and $ld$, to obtain a sequence of four control polygons, and if we iterate this process $n$ times, we obtain $2^n$ control polygons which when linked together yield a polygonal curve that approximates very closely the segment of Bézier curve $C(t)$ for $t \in [0, 1]$. Usually, we perform subdivision for $t = 1/2$. This method is called the subdivision version of the de Casteljau algorithm.

Implement the subdivision version of the de Casteljau algorithm in Matlab, for a cubic specified by its control points $(b_0, b_1, b_2, b_3)$. Your program should take as input the control polygon $(b_0, b_1, b_2, b_3)$ and the number of times $M$ that your program subdivides. The control polygon $(b_0, b_1, b_2, b_3)$ should be represented in Matlab as a $2 \times 4$ matrix $cpoly$ whose first row consists of the $x$-coordinates of $b_0, b_1, b_2, b_3$ and whose second row consists of the $y$-coordinates of $b_0, b_1, b_2, b_3$. For example, given

$$
cpoly = [0 \ 1 \ 2 \ 3; 0 \ 4 \ 5 \ 0],
$$

we obtain the line green polygonal curve of Figure 2 which passes through the points $b_0 = (0, 0)$, $b_1 = (1, 4)$, $b_2 = (2, 5)$, and $b_3 = (3, 0)$. The advantage of this representation is that the polygonal line consisting of the line segments joining the control points $b_0, b_1, b_2, b_3$ is plotted using the command

$$
\text{plot}(cpoly(1,:),cpoly(2,:))
$$
Your goal is to implement the program show_decas_subdiv2 (in project zip file), which should take as input a control polygon cpoly and output the x and y coordinates of the cubic curve. It should also be compatible with the function run_decas_subdiv_g1(cpoly,M,flag) (also in project zip file).

To run the above function and plot your output, type

\[ [x, y] = \text{run\_decas\_subdiv\_g1}(\text{cpoly}, M, 0) \]

in the command window. Before this, make sure to initialize cpoly and M. You must output the final (row) vectors \( x \) and \( y \) after \( M \) iterations.

More specifically, the function show_decas_subdiv2(bx, by, n) returns two row vectors \( x \) and \( y \) of dimension \( 3 \times 2^n + 1 \) consisting of the \( x \)-coordinates and the \( y \)-coordinates of the sequence of nodes starting with \( b_0 \) and ending with \( b_3 \) in the polygonal line produced by the de Casteljau subdivision algorithm after \( n \) rounds of subdivision. This polygonal curve is the concatenation of the \( 2^n \) control polygons (each consisting of 4 nodes) produced after \( n \) rounds of subdivision after removing the duplicate first control point of each control polygon after the first one. For example, for \( n = 1 \) and \( \text{cpoly} = [0 \ 1 \ 2 \ 3; \ 0 \ 4 \ 5 \ 0] \), we get

\[
\begin{align*}
x &= [0 \ 0.5 \ 1 \ 1.5 \ 2 \ 2.5 \ 3] \\
y &= [0 \ 2 \ 3.25 \ 3.375 \ 3.5 \ 2.5 \ 0];
\end{align*}
\]

see Figure 3.
Figure 3: A visualization of the output of `show_decas_subdiv2(bx, by, 1)` applied to `cpoly = [0 1 2 3; 0 4 5 0]`. The output is the concatenation of the two new blue control polygons whose vertices have $x$ and $y$ coordinates given by the lists above.

We suggest that you first write a function `subdecas` that takes as input a control polygon `cpoly` (a $2 \times 4$ matrix) and returns the two control polygons `ud` and `ld` produced after one step of the de Casteljau subdivision algorithm. Then write a function `subdivstep` that takes a $2 \times 4 \times l$ array `lpoly` consisting of $l$ control polygons and produces a $2 \times 4 \times 2l$ array in which each control polygon `lpoly(:, :, i)` is subdivided into two control polygons using `subdecas`. Here $l$ is some power of 2. Finally iterate `subdivstep` $M$ times starting with the input control polygon `cpoly` to produce a $2 \times 4 \times 2^M$ array `lpoly` consisting of $2^M$ control polygons, and then write a function `makelist` that makes the vectors $x$ and $y$ described earlier from `lpoly`.

**Hint:** Note that you will be extending this function in the next part, so ideally it will be easy to generalize to $2 \times n$ input control polygons.

(1)(i) (30 points) In summary, your program must take as input the control polygons listed below.

```plaintext
cpoly1 = [0 1 2 3; 0 4 5 0]
cpoly2 = [0 1 3 4; -2 2 -2 0]
cpoly3 = [3 0 4 1; 0 3 3 0]
cpoly4 = [4 0 4 0; 0 1 1 0]
cpoly5 = [4 0 6 2; 0 6 6 0]
```
For each control polygon (cpoly), your program must output the final (row) vectors $x$ and $y$ after $M$ iterations, for $M = 1, 2, \ldots, 6$. For a test of visual correctness, we will also plot each curve. This is all done in the output script `get_output_1.m`, so you will not need to worry about writing the plotting code for this project.

You may test your program on control polygons that you generated yourself.

(1)(ii) (10 points) Use the subdivision method in which you specify the control points by clicking on the mouse (screen input). The driver function `run_decas_subdiv_g2(M,flag)` and function `getpoints` (both in project zip file) will be used to do this. The output script `get_output_1.m` will automatically prompt you to do the clicking - just click 4 points for this part (though you can do more if you wish). For some examples look at Figure 6.

(2) Given a Bézier curve $C$ of degree $m$ specified by its control points $(b_0, b_1, \ldots, b_m)$, for any $t$, the de Casteljau algorithm constructs points $b_k^i$ in $m$ stages

\[
\begin{align*}
    b_0^1, b_1^1, & \ldots, b_{m-2}^1, b_{m-1}^1 \\
    b_0^2, b_1^2, & \ldots, b_{m-2}^2 \\
    \vdots \\
    b_{m-1}^m, b_1^m \\
    b_0^m.
\end{align*}
\]

If we write $b_i^0 = b_i$ for $i = 0, \ldots, m$, then the $b_k^i$ are given by the following equations

\[
b_k^{i+1} = (1-t)b_k^i + tb_{i+1}^k \quad k = 0, \ldots, m-1, \ i = 0, \ldots, m-k-1,
\]

and as in the case $m = 3$, the point on the curve is

\[
C(t) = b_0^m.
\]

As in the case of cubic curves, the two sequences of points

\[
ud = (b_0, b_1^0, \ldots, b_{m-1}^0, b_m^0)
\]

and

\[
ld = (b_m^0, b_1^{m-1}, \ldots, b_{m-1}^1, b_m)
\]

are also control points for the curve $C$, so we can iterate the above method using the control points in $ud$ and $ld$, and we obtain a subdivision method that yields a polygonal line that approximates very closely the segment of Bézier curve for $t \in [0, 1]$.

Implement the subdivision version of the de Casteljau algorithm in Matlab, for a Bézier curve of degree $m$ specified by its control points $(b_0, b_1, \ldots, b_m)$. Your program should take as input the control polygon $(b_0, b_1, \ldots, b_m)$, and the number of times $M$ that your program
subdivides. The control polygon \((b_0, b_1, \ldots, b_m)\) should be represented in Matlab as a \(2 \times (m+1)\) matrix \(cpoly\) whose first row consists of the \(x\)-coordinates of \(b_0, b_1, \ldots, b_m\) and whose second row consists of the \(y\)-coordinates of \(b_0, b_1, \ldots, b_m\). For example,

\[
\text{cpoly} = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 0 & 4 & 5 & 3 & 2 & 0 \end{bmatrix}.
\]

Use the same driver function as in (1) but modify show decas subdiv2\((bx, by, n)\) so that it returns two row vectors \(x\) and \(y\) of dimension \(m \times 2^n + 1\) consisting of the \(x\)-coordinates and the \(y\)-coordinates of the sequence of nodes starting with \(b_0\) and ending with \(b_m\) in the polygonal line produced by the de Casteljau subdivision algorithm after \(n\) rounds of subdivision. This polygonal line is the concatenation of the \(2^n\) control polygons (each consisting of \(m+1\) nodes) produced after \(n\) rounds of subdivision, and removing the duplicate first control point of each control polygon after the first one. For example, with

\[
\text{cpoly} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 0 & 4 & 3 & 6 & 4 & 0 \end{bmatrix}
\]

and \(n = 1\), we get

\[
x = [1 1.5 2 2.5 3 3.5 4 4.5 5 5.5 6] \\
y = [0 2.75 3.375 3.875 4.0625 4.25 4.125 3.5 2 0].
\]

See Figure 4.

If you used the recommended solution, first adapt the function subdecas to take as input a \(2 \times (m + 1)\) control polygon \(cpoly\) to return the two control polygons \(ud\) and \(ld\). Also adapt the function subdivstep so that it takes a \(2 \times (m + 1) \times l\) array \(lpoly\) consisting of \(l\) control polygons and produces a \(2 \times (m + 1) \times 2l\) array in which each control polygon \(lpoly(:,:,i)\) is subdivided into two control polygons using subdecas.

(2)(i) \((50\ \text{points})\) In summary, your program must take as input the control polygons listed below.

\[
\begin{align*}
cpoly1 &= [1 2 3 4 5 6; 0 4 3 6 4 0] \\
cpoly2 &= [2.9255 0.9333 2.6161 6.6779 9.0571 7.1809; \\
&\quad 1.7041 3.9307 7.2510 7.7979 4.4385 2.0361] \\
&\quad 0.9768 8.9458 9.8064 8.7565 0.6325 1.3554 5.4174 3.9716] \\
cpoly4 &= [7.6168 5.7044 1.6606 1.6168 4.2445 5.8212 8.7847 9.4124 8.0693; \\
cpoly5 &= [7.1058 9.7190 7.3540 4.2591 8.6825 4.7263 0.8577 3.9964 2.9599 1.5438 5.2664; \\
\end{align*}
\]

For each control polygon \(cpoly\), your program must output the final (row) vectors \(x\) and \(y\) after \(M\) iterations, for \(M = 1, 2, \ldots, 6\). For a test of visual correctness, we also plot each curve. Once again this is all done in the output script \texttt{get_output1.m}. 

6
Figure 4: A visualization of the output of `show_decas_subdiv2(bx,by,1)` applied to `cpoly = [1 2 3 4 5 6; 0 4 3 6 4 0]`. The output is the concatenation of the pink and blue control polygons whose vertices have $x$ and $y$ coordinates given by the lists above.

The result of applying the subdivision method for $M = 6$ to the control polygon `cpoly5` is shown in Figure 5.

(2)(ii) (10 points) This is the same as (1)(ii). Use the subdivision method in which you specify the control points by clicking on the mouse (screen input). The driver function `run_decas_subdiv_g2(M,flag)` and function `getpoints` (both in project zip file) will be used to do this. The output script `get_output_1.m` will automatically prompt you to do the clicking as before - this time please click 5 or more points for the curves. For some examples of curves look at figure 6.
Figure 5: de Casteljau subdivision applied to the control polygon cpoly5.
Figure 6: Three Bézier curves (the first two are cubic, the third has degree 7).