Problem set 2b: M-file Programming

Name: Stanciu Iulia-Cristina

MATLAB release used: R2020a

Collaboration Info:

- https://de.mathworks.com/help/matlab/ref/load.html
- https://de.mathworks.com/help/matlab/ref/strings.html
- https://de.mathworks.com/help/matlab/ref/upper.html
- https://de.mathworks.com/help/matlab/ref/strsplit.html
- https://de.mathworks.com/matlabcentral/answers/625638-make-a-function-recursive-koch-snowflake
- https://de.mathworks.com/help/matlab/creating_plots/combine-multiple-plots.html
- https://de.mathworks.com/matlabcentral/answers/100459-how-can-i-insert-a-title-over-a-group-of-subplots

Exercise 4. Morse Code.

Explanation:

The morse_encoder function uses the following *functions*:

- ***nargout*** for the number of output arguments – the program works differently if called morse_encoder(message) or pulse_seq = morse_encoder(message), where message is a string to be converted in morse code. For 0 output arguments, the code message is translated to morse code as a string of dots, lines and spaces, while in the other case it is translated as a vector of 0 and 1.

- ***char*** to make sure that the message is a string
- ***upper*** to transform the given string into one with only uppercase letters, numbers and signs. This step is done because the morse.mat file contains the translation from latin alphabet to morse only for uppercase letters.
- ***length*** to get the length of the message. I needed this to not put a space/ " "/ "000"/ "0000000" at the end of the message.
- ***append*** to add characters to a string
- ***find*** to find the letter in the Morse cell array and then to add its translation to the output
- ***switch*** for the different cases: if two characters are one after another (part of the same word), they are separated only by a one dot silence, while if there is a space in between, that is translated to a 7 dots silence. Similar for the 0 and 1 translation.

The code takes the string message letter, it makes it uppercase, then by letter it and finds the translation in dots, dashes and spaces. Then, if there is an output argument, it translates it to 0 and 1.

The morse_beep function uses the following *functions*:

- ***plot*** to plot the morse code from 0s and 1s to waveform
- ***sound*** to make the waveform audible

The code translates the 0 and 1 vector to a continuous waveform of a duration equal to the length of the messages in dots multiplied by the period of a dot (2*pi*duration).

The script for the third part uses morse_beep function for the given values of:

- pulse_seq = morse_encoder("MAE - SPRING 2022");
- tone_freq = 750;
- dot_duration = 0.06;
- sampling_freq1 = 8000;
- sampling_freq2 = 4000;

The comparation is made through the two plots and sounds.

MATLAB Code:

1. **Write the Matlab function pulse_seq = morse_encoder(message)**

```
function pulse_seq = morse_encoder(message)

load('morse.mat')

if nargout==0
    pulse_seq = strings;
    string_message = char(message);
    new_message = upper(string_message);
    for i = 1:length(new_message)
        j = find(Alpha == new_message(i));
        str = char(Morse(j));
        disp(str);
        pulse_seq = append(pulse_seq, str);
        if( i ~= length(new_message))
            pulse_seq = append(pulse_seq, ' ');
        end
    end
else
    pulse_seq = [];
    string_message = char(message);
    new_message = upper(string_message);
    for i = 1:length(new_message)
        j = find(Alpha == new_message(i));
        string = char(Morse(j));
        disp(string);
        for k = 1:length(string)
            switch string(k)
                case "."
                    pulse_seq = [pulse_seq, 1];
                    if( k~= length(string) )
                        switch string(k+1)
                            case "."
                                pulse_seq = [pulse_seq, 0];
                            case "-"
                                pulse_seq = [pulse_seq, 0];
                        end
                    else
                        if( i~= length(new_message))
                            pulse_seq = [pulse_seq, 0, 0, 0];
                        end
                    end

                case "-"
                    pulse_seq = [pulse_seq, 1, 1, 1];
                    if( k~= length(string) )
                        switch string(k+1)
                            case "."
                                pulse_seq = [pulse_seq, 0];
                            case "-"
                                pulse_seq = [pulse_seq, 0];
                        end
                    else
                        if( i~= length(new_message))
```

```
                                    pulse_seq = [pulse_seq, 0, 0, 0, 0, 0, 0, 0];
                                end
                            end
                    end
                end
            end
        end
    end
```

2. **Write a Matlab function morse_beep(pulse_seq,tone_freq,dot_duration, sampling_freq)**

```
function morse_sound = morse_beep(pulse_seq, tone_freq, dot_duration,
sampling_freq)

    morse_sound = [];
    %pulse_seq = morse_encoder(message);

    sampling_period = 1/sampling_freq;
    t = 0 : sampling_period*2*pi : dot_duration*2*pi;

    message_length = length(pulse_seq);
    for i = 1:message_length
             morse_sound = [morse_sound, tone_freq*pulse_seq(i)*cos(t)];
    end

    sound(morse_sound)
    figure
    plot(morse_sound)
    xlabel('message duration = time (s)')
    ylabel('tone frequency (Hz)')
    title("Representation of message in morse code for the chosen sampling
frequency: " + sampling_freq + "Hz, tone frequency " + tone_freq + " and
duration of a pulse " + dot_duration + " seconds.")

end
```

3. **Write a Matlab script that calls the previous two functions to produce the Morse code of the text "MAE - SPRING 2022" with parameters:**
   dot_duration = 0.06sec tone_freq = 750 Hz sampling_freq = 8000Hz.
   Compare the sound with the same sequence and parameters, but with a sampling frequency of 4000Hz.

```
clear
clc

pulse_seq = morse_encoder("MAE - SPRING 2022");
tone_freq = 750;
dot_duration = 0.06;
sampling_freq1 = 8000;
sampling_freq2 = 4000;

morse_beep(pulse_seq, tone_freq, dot_duration, sampling_freq1)
morse_beep(pulse_seq, tone_freq, dot_duration, sampling_freq2)
```

Results:

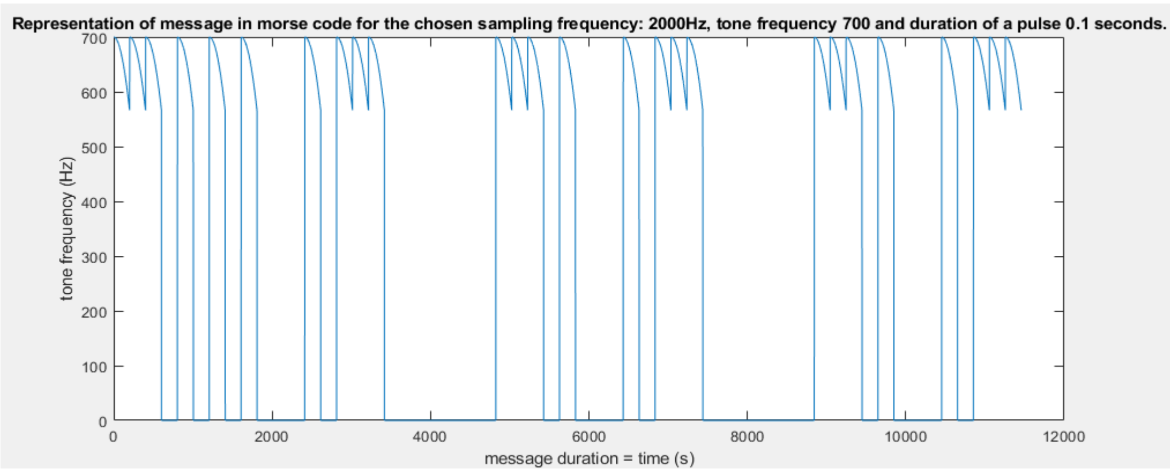
1. **Write the Matlab function pulse_seq = morse_encoder(message)**

```
>> pulse_seq = morse_encoder("I'm here now")

pulse_seq =

  Columns 1 through 15

     1     0     1     0     0     0     1     0     1     1     1     0     1     1     1

  Columns 16 through 30

     0     1     1     1     0     1     1     1     0     1     0     0     0     1     1

  Columns 31 through 45

     1     0     1     1     1     0     0     0     0     0     0     0     1     0     1

  Columns 46 through 60

     0     1     0     1     0     0     0     1     0     0     0     1     0     1     1

  Columns 61 through 75

     1     0     1     0     0     0     1     0     0     0     1     1     1     0     1

  Columns 76 through 90

     0     0     0     1     1     1     0     1     1     1     0     1     1     1     0

  Columns 91 through 105

     0     0     0     0     0     0     1     0     1     1     1     0     1     1     1
```

```
>> morse_encoder("I'm here")

ans =

    ".. .----. --  .... . .-. ."
```

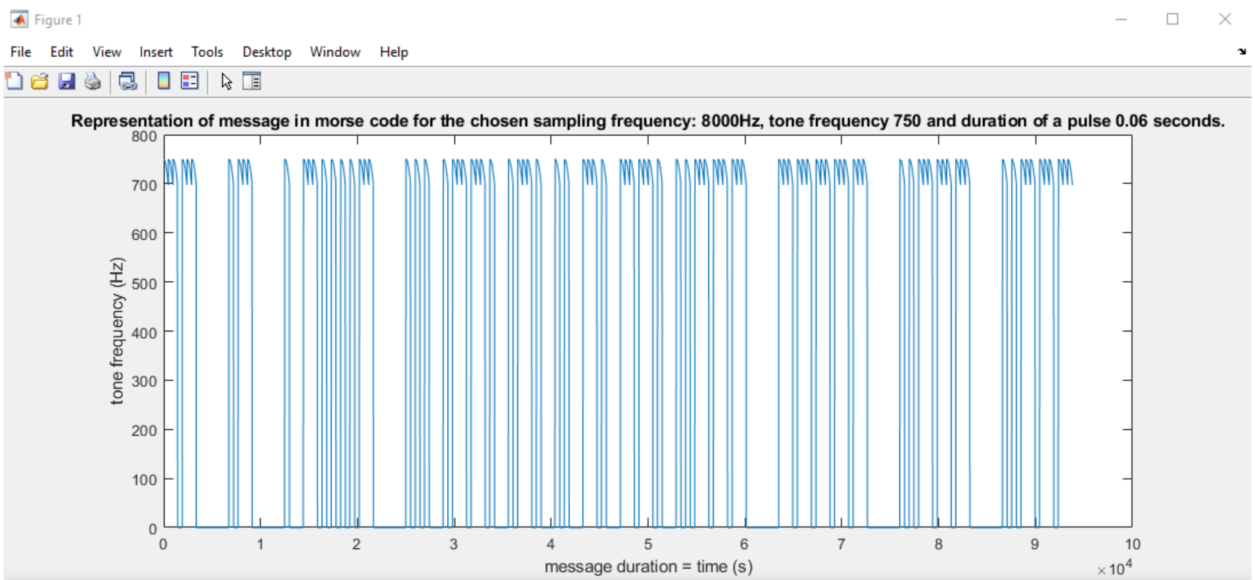
```
>> morse_encoder("Matlab is awesome")

ans =

    "-- .- - .-.. .- -...  .. ...  .- .-- . ... --- -- ."
```

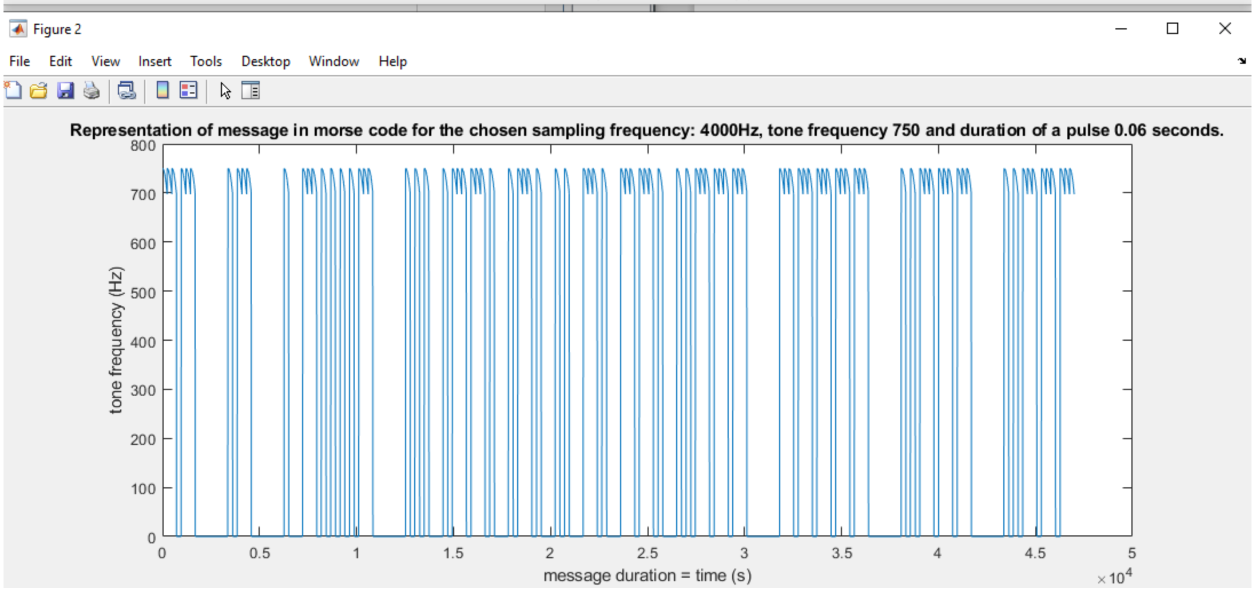
2. **Write a Matlab function morse_beep(pulse_seq,tone_freq,dot_duration, sampling_freq)**




```
>> pulse_seq = morse_encoder("banana");
>> morse beep(pulse seq, 700, 0.1, 2000)
```

3. **Write a Matlab script that calls the previous two functions to produce the Morse code of the text "MAE - SPRING 2022" with parameters:**
dot_duration = 0.06sec tone_freq = 750 Hz sampling_freq = 8000Hz.
Compare the sound with the same sequence and parameters, but with a sampling frequency of 4000Hz.





<u>Comments:</u>

A pause can be added in the script for a better comparison.

Exercise 4. Koch fractal curve.

Explanation:

The koch function uses the following *functions*:

- ***nargout*** for the number of output arguments – the program works differently for 0 or 1 output arguments.
- ***plot and subplot*** to plot multiple things in the same figure.
- ***size*** to find the size of a row/column.
- ***zeroes*** to initialize a matrix with a known number of rows and columns.

The code starts from the equilateral triangle (given as a matrix with two rows, one for x and one for y coordinates), then, using sin and cos and the coordinates of the margins, modifies every curve.

The genkoch function has a similar functionality as the previous one. Different from that, it has more arguments, and it starts from a given curve and a different way to modify it. The way the implemented program works is that is that it uses the M = [0,1;0,0]; line and the pattern specified to start in (0,0) and end in (0, 1) and modifies the M matrix accordingly, then, it uses M0, the initial curve and the given transformation to transform the initial curve accordingly:

$$u = u_0 + (u_1 - u_0)x - (v_1 - v_0)y$$

$$v = v_0 + (v_1 - v_0)x + (u_1 - u_0)y$$

.

In the code there can be always seen two matrices: the newM matrix that starts from M = [0,1;0,0] and the plotM matrix that starts from the given M0.

MATLAB Code:

1. **Create a function koch such that M=koch(n) outputs a two-row matrix M, containing the x and y-coordinates of the vertices of the n-th curve.**

```
function M = koch(n)
    M = [0 1 cos(-pi/3) 0; 0 0 sin(-pi/3) 0];

    if nargout==0
        clf;
        figure(1);
        suptitle("Koch fractal curve of orders 0 to " + n + " starting from a
equilateral triangle");

        subplot(n+1,1,n+1)
        plot( M(1,:), M(2,:) );

        axis equal;
        axis off;
    end
```

```
    for i = 1:n
        newM = zeros( 2, size(M,2)*4+1);

        for j = 1:size(M,2)-1
            newM(:, 4*j+1) = M(:, j);
            newM(:, 4*j+2) = (2*M(:, j) + M(:, j+1) )/3;
            link = M(:, j+1).'-M(:, j).';
            ang = atan2( link(2), link(1) );
            linkLeng = sqrt( sum(link.^2) );
            newM(:, 4*j+3) = newM(:, 4*j+2) + (linkLeng/3)*[ cos(ang+pi/3);
sin(ang+pi/3) ];
            newM(:, 4*j+4) = (M(:, j) + 2*M(:, j+1) )/3;
        end
        M = newM;

        if nargout==0
            subplot(n+1,1,n+1-i)
            plot( M(1,:), M(2,:) );
            axis equal;
            axis off;
        end
    end
end
```

2. **Modify the previous function, and call it genkoch, to admit two extra input parameters: An input matrix Patt containing an arbitrary polygonal curve with endpoints (0, 0) and (1, 0), and an extra matrix M0 defining the starting (order zero) curve.**

```
function M = genkoch(n, Patt, M0)

    M = [0,1;0,0];
    plotM = M0;

    if nargout==0
        clf;
        figure(1);
        suptitle("Koch fractal curve of orders 0 to " + n + " starting from a
the given curve M0 and folowing the pattern of Patt");


        subplot(n+1,1,n+1)
        plot( plotM(1,:), plotM(2,:) );
        %axis([0 1 0 0.5]);
        axis equal;
        axis off;
    end

    points = size(Patt,2);

    for i = 1:n
        newM = zeros( 2, size(M,2)*(points-1)+1);
```

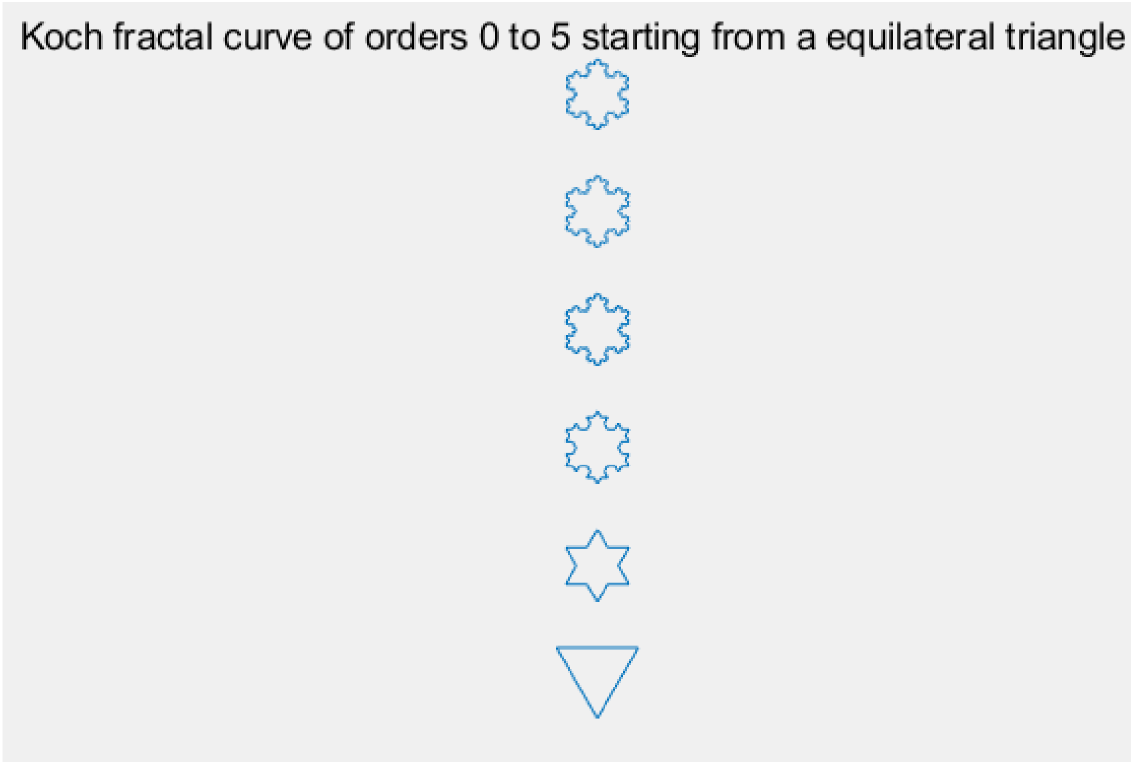
```
        plotM = zeros( 2, size(M,2)*(points-1)+1);

        for j = 1:size(M,2)-1
            u0 = M(1, j);
            u1 = M(1, j+1);
            v0 = M(2, j);
            v1 = M(2, j+1);

            newM(1, ((points-1)*j+1):((points-1)*j+points) )= u0 + (u1-
u0)*Patt(1, :) - (v1-v0)*Patt(2, :);
            newM(2, ((points-1)*j+1):((points-1)*j+points) ) = v0 + (v1-
v0)*Patt(1, :) + (u1-u0)*Patt(2, :);

            u0_plot = M0(1, 1);
            u1_plot = M0(1, 2);
            v0_plot = M0(2, 1);
            v1_plot = M0(2, 2);

            plotM(1, ((points-1)*j+1):((points-1)*j+points) )= u0_plot +
(u1_plot-u0_plot)*newM(1, ((points-1)*j+1):((points-1)*j+points)) - (v1_plot-
v0_plot)*newM(2, ((points-1)*j+1):((points-1)*j+points));
            plotM(2, ((points-1)*j+1):((points-1)*j+points) ) = v0_plot +
(v1_plot-v0_plot)*newM(1, ((points-1)*j+1):((points-1)*j+points)) + (u1_plot-
u0_plot)*newM(2, ((points-1)*j+1):((points-1)*j+points));

        end

        M = newM;
        if nargout==0
            subplot(n+1,1,n+1-i)
            plot( plotM(1,:), plotM(2,:) );
            %axis([0 1 0 0.5]);
            axis equal;
            axis off;
        end
    end
    M = plotM;
end
```

Results:

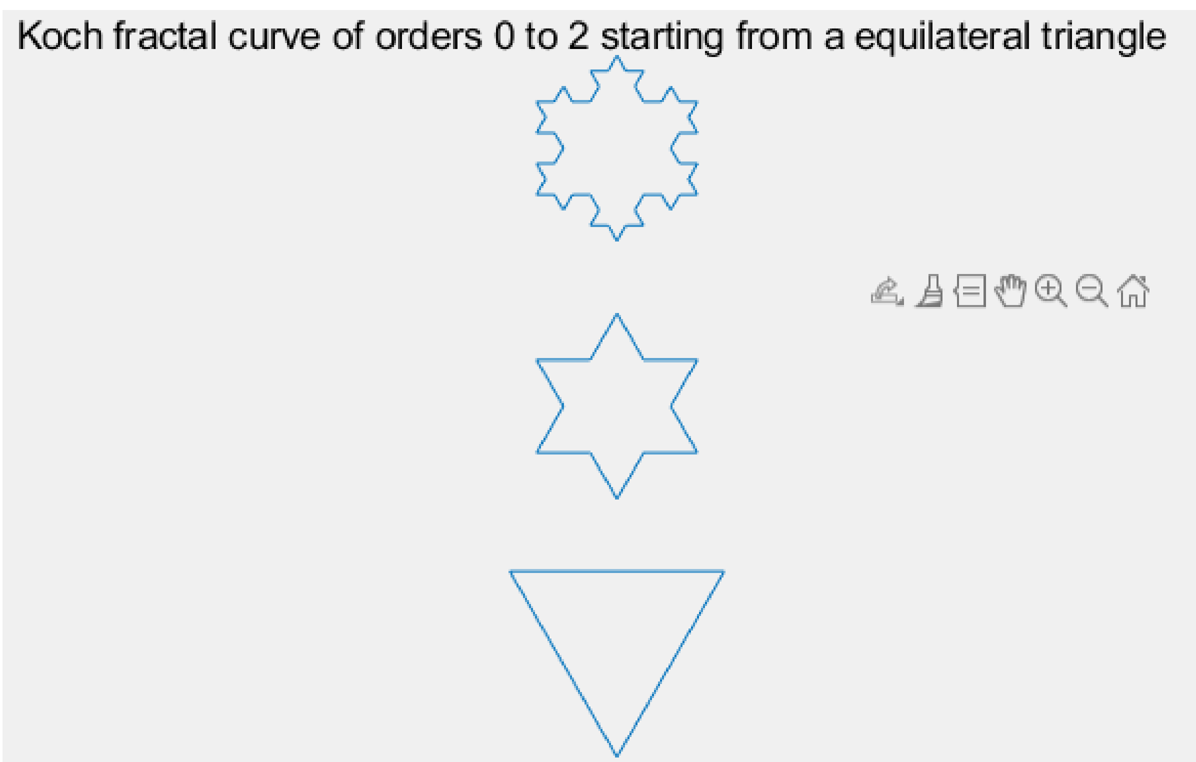
1. **Create a function koch such that M=koch(n) outputs a two-row matrix M, containing the x and y-coordinates of the vertices of the n-th curve.**

```
>> M = koch(2)

M =

  Columns 1 through 9

         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0

  Columns 10 through 18

         0         0         0         0         0         0         0         0         0
         0         0         0         0         0         0         0         0         0

  Columns 19 through 27

         0         0         0    0.1111    0.1667    0.2222    0.3333    0.3889    0.3333
         0         0         0         0    0.0962         0         0    0.0962    0.1925

  Columns 28 through 36

    0.4444    0.5000    0.5556    0.6667    0.6111    0.6667    0.7778    0.8333    0.8889
    0.1925    0.2887    0.1925    0.1925    0.0962         0         0    0.0962         0

  Columns 37 through 45

    1.0000    0.9444    1.0000    0.8889    0.8333    0.8889    1.0000    0.9444    1.0000
         0   -0.0962   -0.1925   -0.1925   -0.2887   -0.3849   -0.3849   -0.4811   -0.5774

  Columns 46 through 54

    0.8889    0.8333    0.7778    0.6667    0.6111    0.6667    0.5556    0.5000    0.4444
   -0.5774   -0.6736   -0.5774   -0.5774   -0.6736   -0.7698   -0.7698   -0.8660   -0.7698

  Columns 55 through 63

    0.3333    0.3889    0.3333    0.2222    0.1667    0.1111    0.0000    0.0556    0.0000
   -0.7698   -0.6736   -0.5774   -0.5774   -0.6736   -0.5774   -0.5774   -0.4811   -0.3849

  Columns 64 through 69

    0.1111    0.1667    0.1111    0.0000    0.0556         0
   -0.3849   -0.2887   -0.1925   -0.1925   -0.0962         0
```

```
>> koch(5)
```


Koch fractal curve of orders 0 to 5 starting from a equilateral triangle

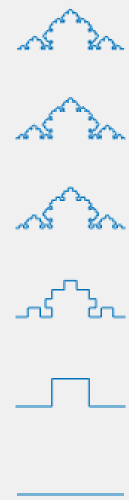
```
>> koch(2)
```


Koch fractal curve of orders 0 to 2 starting from a equilateral triangle

2. **Modify the previous function, and call it genkoch, to admit two extra input parameters: An input matrix Patt containing an arbitrary polygonal curve with endpoints (0, 0) and (1, 0), and an extra matrix M0 defining the starting (order zero) curve.**

```
Patt=[0,1/3,1/3,2/3,2/3,1;0,0,.25,.25,0,0];
M0=[0,1;0,0];

genkoch(5,Patt,M0)
```

Koch fractal curve of orders 0 to 5 starting from a the given curve M0 and folowing the pattern of Patt



```
>> Patt=[0,1/3,1/3,2/3,2/3,1;0,0,.25,.25,0,0];
>> M0=[0,1;0,1];
>> genkoch(5,Patt,M0)
```

Koch fractal curve of orders 0 to 5 starting from a the given curve M0 and folowing the pattern of Patt