Signal Progressing

MAS 3114 Spring 2023, Prof. Curtin

By

Kanye Smith

Contents

1 Abstract

Throughout this project, we will explore the idea of using RGB color vectors alongside vector coordinates to distinguish different colors.

2 Motivation

2.1 Main Problem

The text[4] provides us with the following data plot summation.

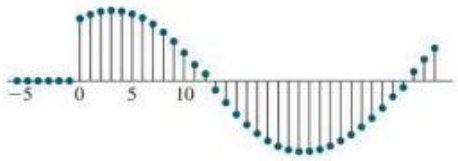


FIGURE 4 A discrete-time signal.

As seen in the overall shape of the graph, it is noticeable that for each k value on the x-axis, which represents instances in time, there exists a corresponding y_k value. Thus, by interpreting the individual entries along the graph, it is possible to establish conclusions about the trend of an event, as well as being able to distinguish certain results from each other. Although the figure presented above only accounts for the discrete-time signal, signal processing is a broad field with various techniques, such as Analog, Continuous Time, as well as Nonlinear Signal Processing. In particular, this project will focus mainly on applications related to image processing, which declutters data and complex algorithms, allowing for information to be extracted.

2.2 History and General Overview

The idea of signal processing dates back as far as 1807[2] when French Mathematician Joseph Fourier submitted a manuscript introducing the concept of Fourier Series, which allows periodic functions to be expressed as an infinite sum of sines and cosines[3]. Typically used in areas such as differential equations, the process of generalizing these functions and equations in a simpler form creates general solutions of differential equations, which then allows conclusions to be made in terms relative to said equations. Although Fourier is accredited as being the “founder” of the Fourier Series, the Fourier Transform was actually coined by Augustin-Louis Cauchy, another French mathematician. Similar to the Fourier Series, the related transform also allows for the decomposition of functions into their sines and cosines. Moreover, the text[1] states that its usage “replaces the description of a signal in the time or space domain by another one in the frequency domain.” Thus, the Fourier transform can be paralleled to the concept of discrete-time signals, which are signals that occur at specific instants of time.

3 Mathematical Description of Solutions

3.1 Problem 1

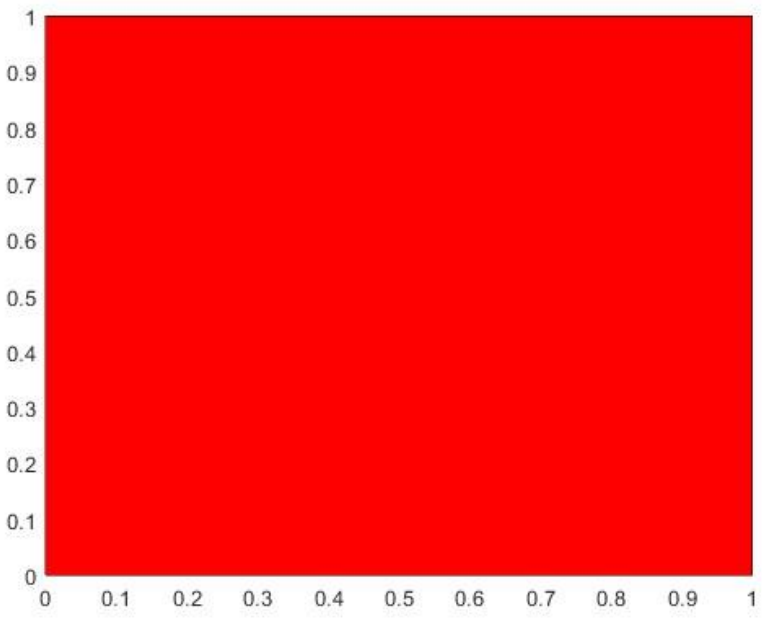
In the first problem, we are given two coordinate vectors" x" and "y", where x represents the x_1 coordinates [0 1 1 0] and y represents the x_2 coordinates [0 0 1 1]. Thus, considering the given vectors, we are able to interpret the potential graphing of a box.

Per the context of the Signal Processing Project, a color blind student is attempting to identify certain colors or images through the usage of RGB vectors. The text[6] provides a basic of RGBs by stating that these colors are the result of three numbers, ranging from 0 to 255, being enclosed in parentheses and viewed in a 3d vector space. However, rather than utilizing values between 0 and 255, the problem stops the range's upper limit at 1, allowing us to interpret the color vectors as being orchestrated similar to RGB Binary Code.

As such, MATLAB uses the patch(x,y,c) command to plot colored images within the constraints of the "x" and "y" coordinates on a plane, along with the RGB vector "c". To understand this idea, we are initially tasked with patching "x" and "y" with the RGB vector [1,0,0]. Thus, using MATLAB, we input the command

```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [1, 0, 0])
```

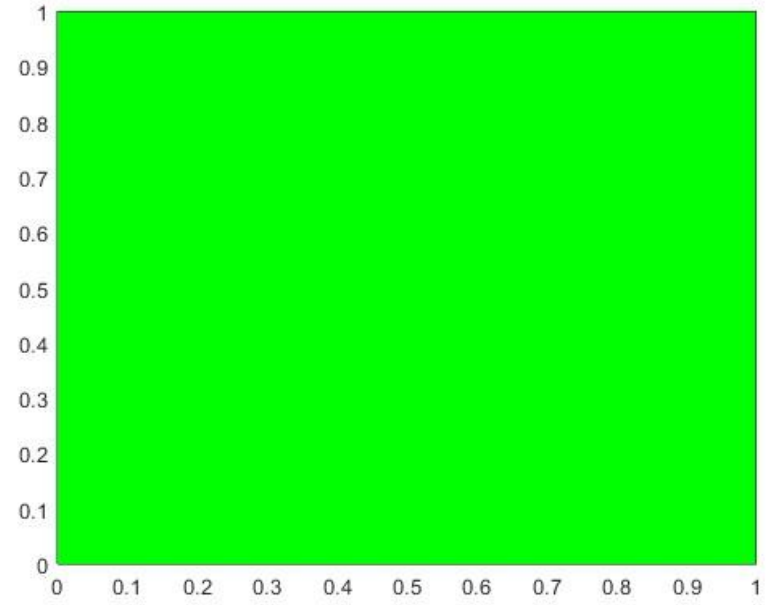
And receive the color polygon



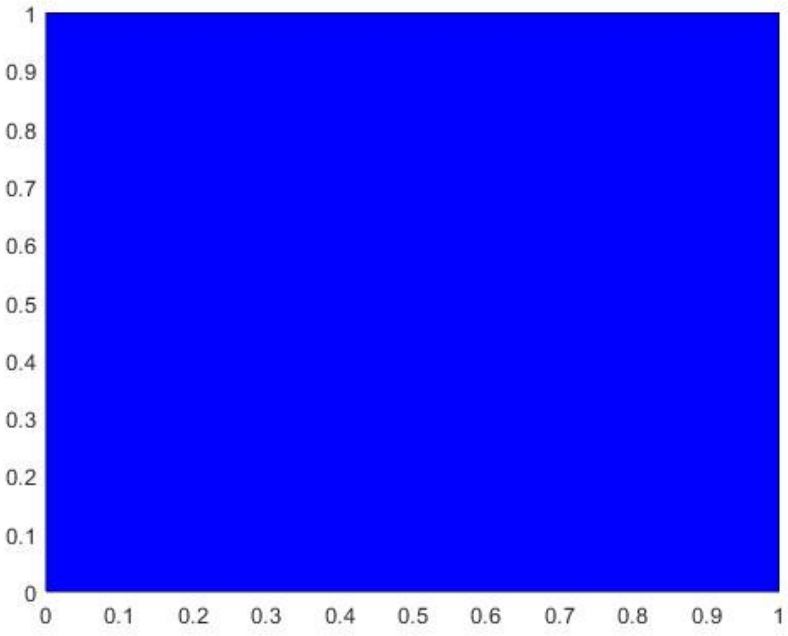
As seen in the MATLAB output, the triplet combination [1,0,0] produces a red box. To further emphasize the concept of different RGB triplet combinations signaling multiple colors, we are also given four other "c" vectors to patch within the box

coordinates, namely [0,1,0], [0,0,1], [0,0,0], [1,1,1]. Continuing the approach utilized with the first RGB vector, the following patch commands output these polygons.

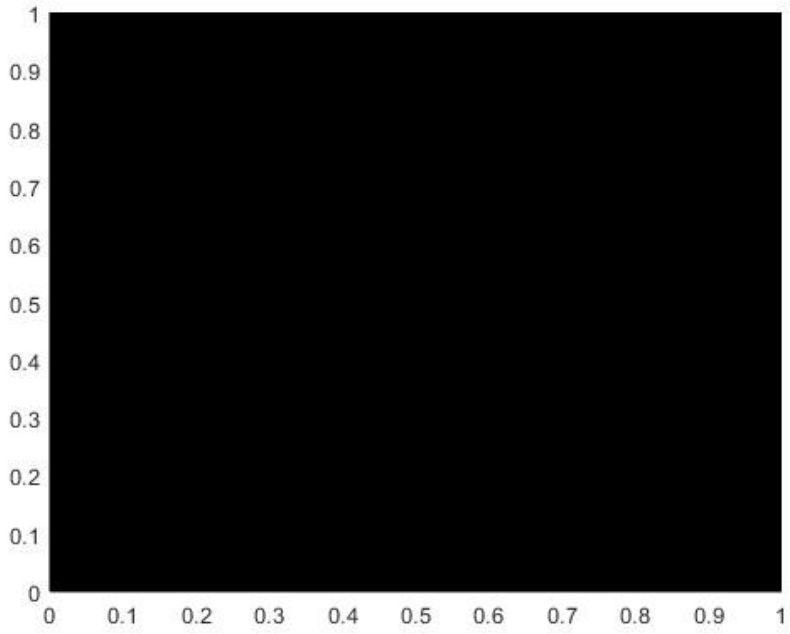
```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [0, 1, 0])
```



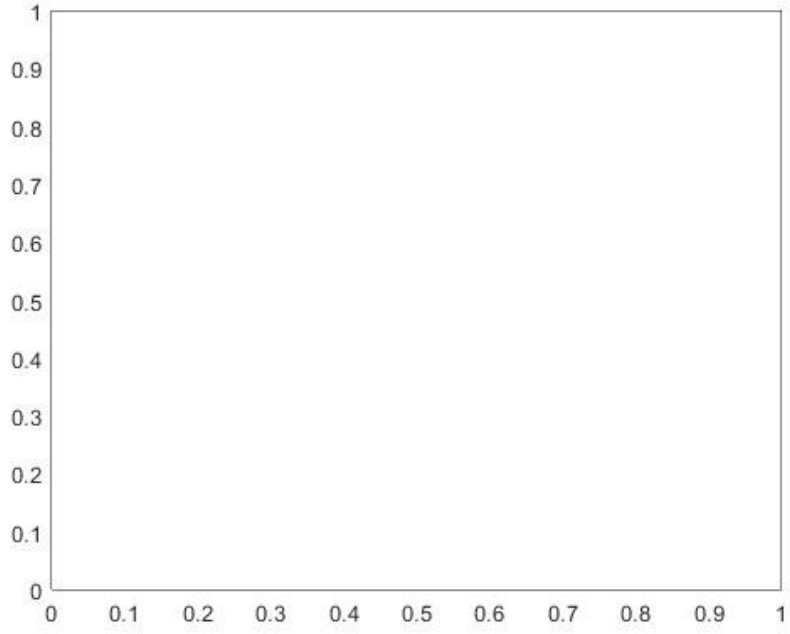

```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [0, 0, 1])
```



```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [0, 0, 0])
```



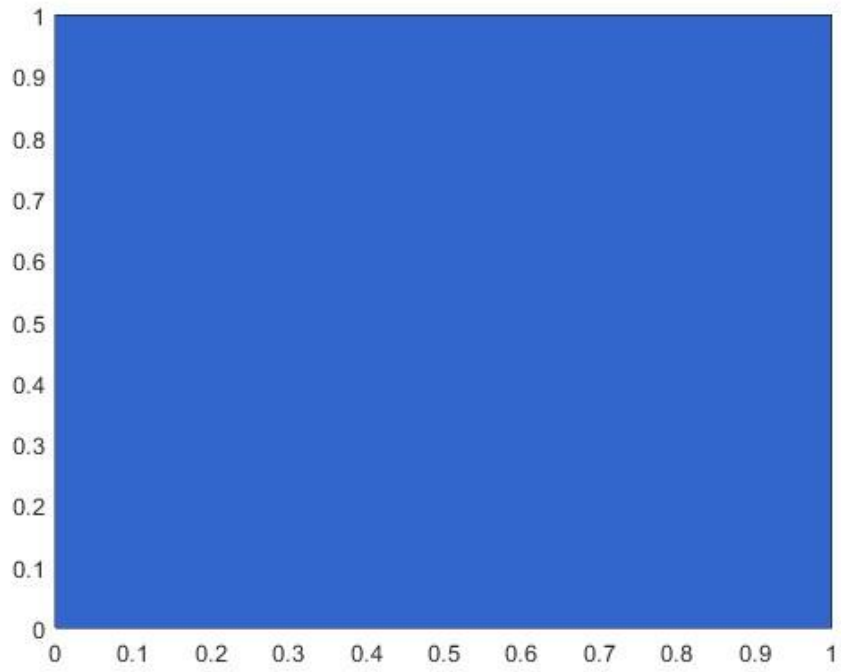

```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [1, 1, 1])
```



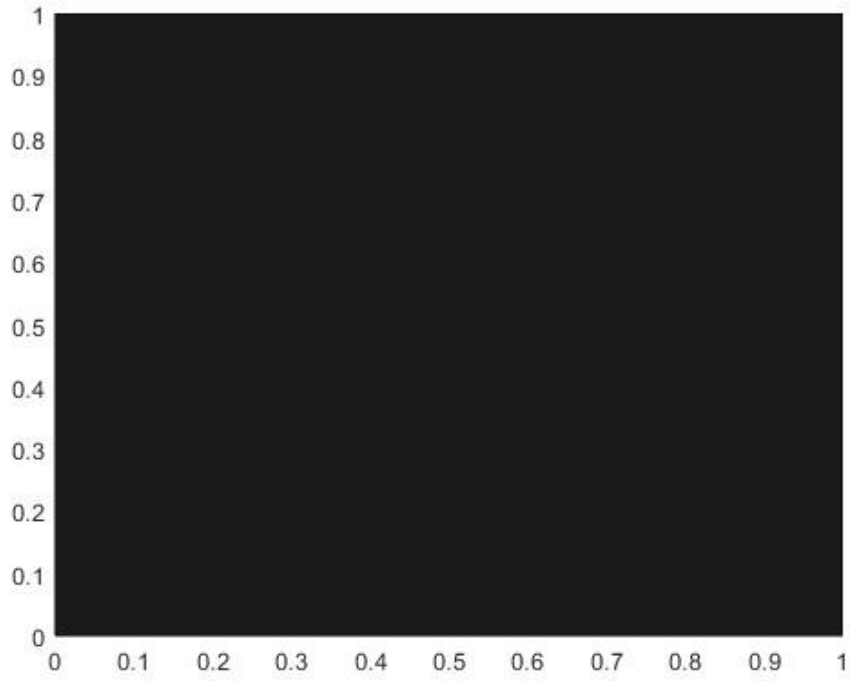
Thus, it is apparent that when both 0 and 1 entries were included in the vector, the resulting box produced colors green and blue. Due to the patch command explanation from the text[5], these plots were expected since the text uses the same number pattern, just with them upper limit 255 instead of 1. The latter experimentation results used either all zero entries or all one entries, which produced a black box for the zero entries, and a white box for the one entries. This can be interpreted as being in line with the concept of RGB Binary code which also relays the color black for all zero RGB entries, and the color white for all one entries.

Moreover, the problem attempts to experiment with different entry combinations by requesting triplet values between 0 and 1, rather than strictly those two entries. By utilizing the vectors [.2,.4,.8], [.1, .1, .1], [.05, .05, .05], and [.95, .95, .95], the following colors were produced within the x and y coordinates.

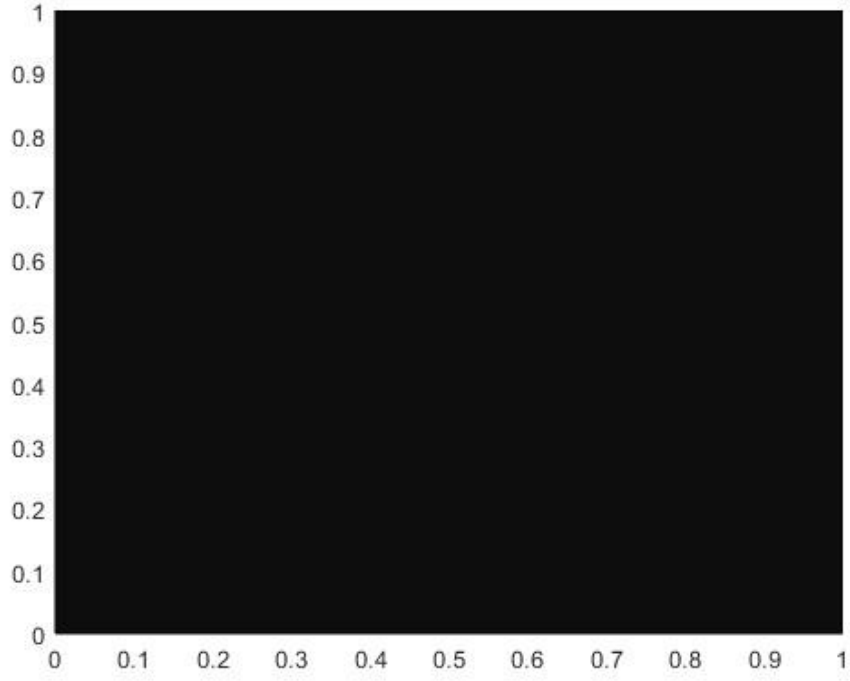
```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [.2, .4, .8])
```



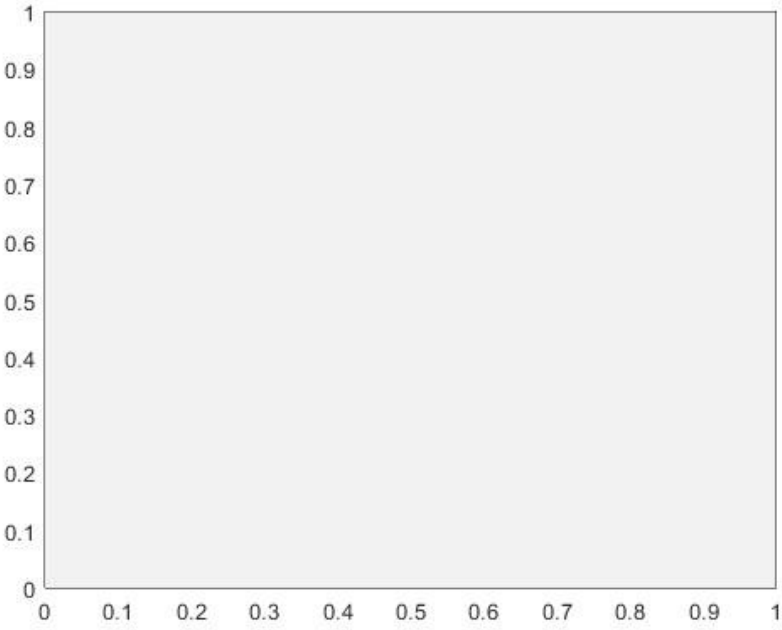

```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [.1, .1, .1])
```




```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [.05, .05, .05])
```




```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [.95, .95, .95])
```



Through the four previously given coordinates, it was already understood that when using the patch command, the 1x3 vector [0,0,0] produced a black square, whereas the vector [1,1,1] produced a white square. When using numbers between 0 and 1, assuming the same three numbers were used in the vector, numbers closer to 0, namely .1 and .05, produced a darker square, and values closer to 1, namely .95, produced a lighter square.

3.2 Problem 2

We further explore the idea of using unique RGB color vectors to signal specific colors in the second problem. However, rather than using 1x3 coordinate vectors, we switch to 4x4 vectors with the intention of creating a picture/image while retaining a sense of simplicity through the usage of low value positive numbers. Thus, the problem provides the x and y vectors:

$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 2 & 2 & 1 \\ 2 & 3 & 3 & 2 \\ 3 & 4 & 4 & 3 \end{bmatrix}$$

$$\mathbf{Y} = \begin{bmatrix} 3 & 3 & 4 & 4 \\ 2 & 2 & 3 & 3 \\ 1 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

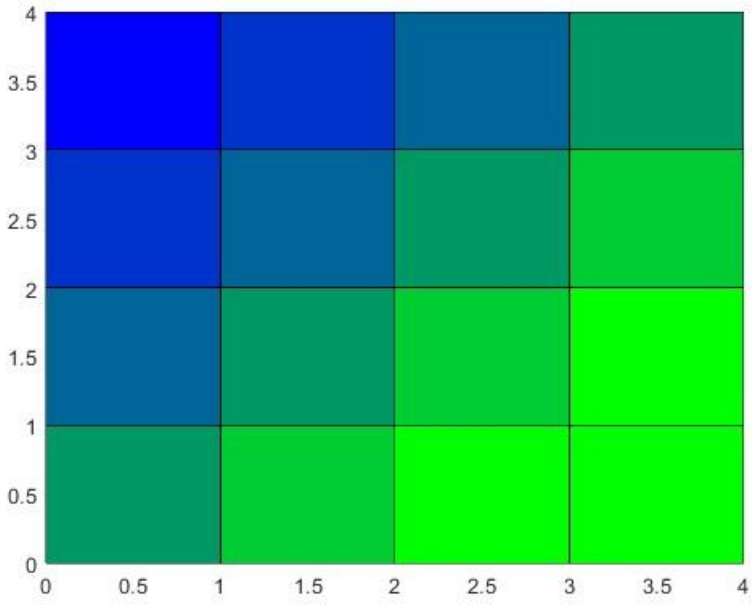
Moreover, we are given the following script code in order to plot an image.

```
function graphsignal(X, Y, code)
clf
for j=1:4
for i=1:4
patch( X(i,:), Y(j,:), code(i,3*j-2:3*j))
end
end
```

For the specific problem, we intend to graph multiple color boxes that shift from blue to green by inserting the following 4x4 color vector. It is important to note that while the vector is acknowledged as being a "4x4 vector", it still retains the RGB vector property of being composed of triplet entries between 0 and 1.

```
bluetogreen = [ 0 0 1,   0 .2 .8,  0 .4 .6,  0 .6 .4
                0 .2 .8, 0 .4 .6,  0 .6 .4,  0 .8 .2
                0 .4 .6, 0 .6 .4,  0 .8 .2,  0 1 0
                0 .6 .4, 0 .8 .2,  0 1 0,    0 1 0]
```

Afterwards, we refer to the given MATLAB script code, and call graphsignal(X,Y,bluetogreen) in order to view the image. While not being an official command, the function *graphsignal* aims to accomplish a job comparable to the patch command by plotting the colors within the constraints of the coordinate vectors. Therefore, we observe the plot:



Similar to the concept behind discrete-time signal processing in which certain signal values are sequentially outputted at their respective times, the above plot shows that, likewise, at certain areas of the plot, there are unique colors specific to that area. Also, returning to the results given from the first problem, it is noted that since none of the triplet entries were composed strictly of the same number 3 times, a significantly dark or light box wasn't not created.

Reverting back to the basic idea of signaling, the problem further aims to allow the bluetogreen RGB vector to be viewed as a signal. Thus, using the following MATLAB code, the following code, we create the "variable" b_k to account for k values between 1 and 48 due to the fact that there are 48 individual bluetogreen vector entries. As such we assign the variables b_1 to b_{48} the vector entries, and allow b_k to equal 0 for k values <1 and >48.

The MATLAB graphsignal output is as follows:

```
>> bluetogreen = [ 0 0 1, 0 .2 .8, 0 .4 .6, 0 .6 .4
0 .2 .8, 0 .4 .6, 0 .6 .4, 0 .8 .2
0 .4 .6, 0 .6 .4, 0 .8 .2, 0 1 0
0 .6 .4, 0 .8 .2, 0 1 0, 0 1 0]
M=bluetogreen
Mt=M';
for k=1:48
c(k) = Mt(k);
end
c

bluetogreen =

         0         0    1.0000         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000
         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000
         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0
         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0         0    1.0000         0


M =

         0         0    1.0000         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000
         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000
         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0
         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0         0    1.0000         0


c =

  Columns 1 through 12

         0         0    1.0000         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000

  Columns 13 through 24

         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000

  Columns 25 through 36

         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0

  Columns 37 through 48

         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0         0    1.0000         0

>>
```

3.3 Problem 3

In the final, we attempt to replicate the signal graphing procedure utilized in Problem 2 by creating a new image and signal with the same X and Y coordinates, but using different RGB vectors. Similar to the reasoning within the previous problem of maintaining simplicity, the plot consists of 4x4 boxes and RGB vectors with entries between 0 and 1.

Thus, using MATLAB, the following coordinates are input:

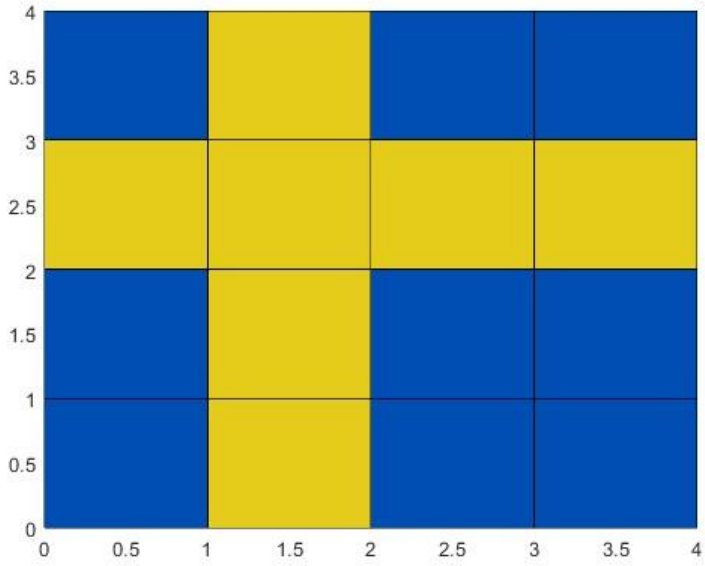
$$X = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 2 & 2 & 1 \\ 2 & 3 & 3 & 2 \\ 3 & 4 & 4 & 3 \end{bmatrix}$$

$$Y = \begin{bmatrix} 3 & 3 & 4 & 4 \\ 2 & 2 & 3 & 3 \\ 1 & 1 & 2 & 2 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

Then, the RGB color vector “sweden” is utilized with the triplet values (0, .3, .7) and (.9, .8, .1), where (0, .3, .7) produces a navy blue color and (.9, .8, .1) produces a gold color. As such, the intention behind using these specific colors is to create an image that emulates the Swedish flag.

After inserting the RGB values, the previous graphsignal command is executed in MATLAB with the variable “sweden" representing the “c” vector. The layout is as follows:

```
>> sweden = [ 0 .3 .7, .9 .8 .1, 0 .3 .7, 0 .3 .7
.9 .8 .1, .9 .8 .1, .9 .8 .1, .9 .8 .1
0 .3 .7, .9 .8 .1, 0 .3 .7, 0 .3 .7
0 .3 .7, .9 .8 .1, 0 .3 .7, 0 .3 .7]
```



As seen in the plot, the execution was successful since the blue and gold colors were produced and restricted to their respective areas.

To allow the “sweden” RGB vector to be observed as a signal, the same MATLAB code was used with M equaling sweden rather than bluetogreen. Therefore, the signal c is as follows:

```
>> sweden = [ 0 .3 .7, .9 .8 .1, 0 .3 .7, 0 .3 .7
.9 .8 .1, .9 .8 .1, .9 .8 .1, .9 .8 .1
0 .3 .7, .9 .8 .1, 0 .3 .7, 0 .3 .7
0 .3 .7, .9 .8 .1, 0 .3 .7, 0 .3 .7]
M=sweden
Mt=M';
for k=1:48
c(k) = Mt(k);
end
c

sweden =

         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000
         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000


M =

         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000
         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
```

```
c =

  Columns 1 through 16

         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000    0.9000    0.8000    0.1000    0.9000

  Columns 17 through 32

    0.8000    0.1000    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000

  Columns 33 through 48

    0.7000         0    0.3000    0.7000         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
```

4 Discussion and Conclusion

The signal processing concept has proved to be an effective method when attempting to distinguish specific RGB colors through the usage of vectors.

In Problem 2 where 4x4 colored box plots were used to demonstrate how slightly altering the triplet entries of an RGB vector could impact the color output, it's important to consider different potential approaches to the problem. For instance, it was obvious that when going down the entry list from first to last, the outputs were all of similar shades of blue or green. Despite accounting for these colors, more plots could've been used to include warmer colors such as red or orange. Comparing similar inputs to more obscure inputs would have proposed a more complete understanding of how different "signal frequencies" across various input instances. To illustrate this concept of altering the amount of "red", "green", and "blue" impact the color output, the text provides this example.

5 Nomenclature

Variable	Meaning
RGB	red, green and blue
patch(x,y,c)	MATLAB command that creates patches of colored polygons
graphsignal(x,y,code)	MATLAB function that expands the patch command

6 References

[1]Flandrin , P. (2022, September 7). *Signal Processing: A field at the heart of science and everyday life*. The Conversation. Retrieved March 26, 2023, from https://theconversation.com/signal-processing-a-field-at-the-heart-of-science-and-everyday-life-89267

[2]IEEE Pulse. (2022, March 4). *Highlights in the history of the Fourier transform*. IEEE Pulse. Retrieved March 26, 2023, from https://www.embs.org/pulse/articles/highlights-in-the-history-of-the-fourier-transform/#:~:text=Surprisingly%2C%20this%20method%20for%20derivation,institute%20in%201811%20%5B11%5D.

[3]Khan Academy. (n.d.). *Fourier series introduction (video)*. Khan Academy. Retrieved March 26, 2023, from https://www.khanacademy.org/science/electrical-engineering/ee-signals/ee-fourier-series/v/ee-fourier-series-intro

[4]Linear Algebra and its Applications, 6th edition. David Lay, Judi McDonals, Steven Lay, Pearson (2020).

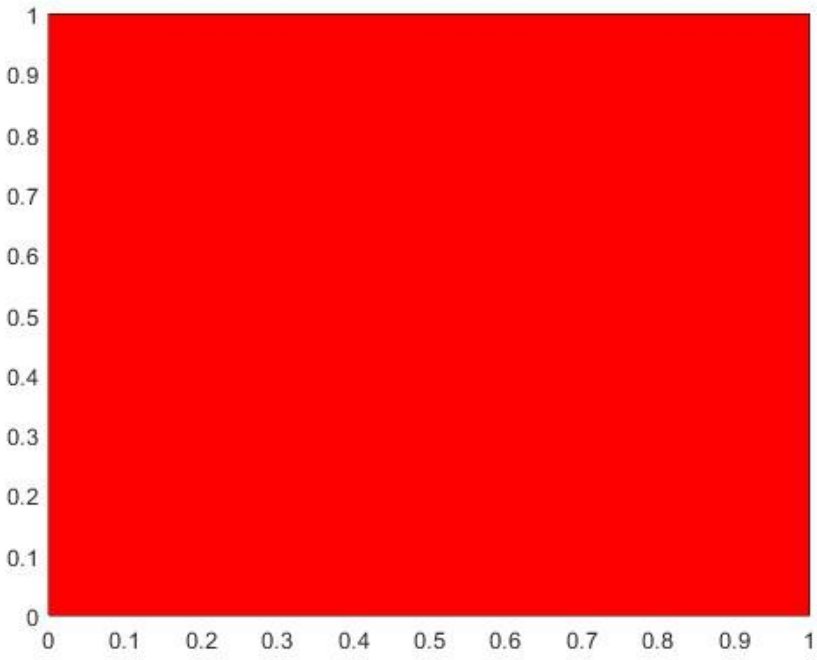
[5]patch. Create patches of colored polygons - MATLAB. (n.d.). Retrieved March 26, 2023, from https://www.mathworks.com/help/matlab/ref/patch.html

[6]Riva, S. (n.d.). *Linear combination of vectors and RGB colours*. GeoGebra. Retrieved March 26, 2023, from https://www.geogebra.org/m/Dq2A7aRv#:~:text=In%20computers%2C%20a%20RGB%20colour,combination%20of%20these%20basic%20colours.

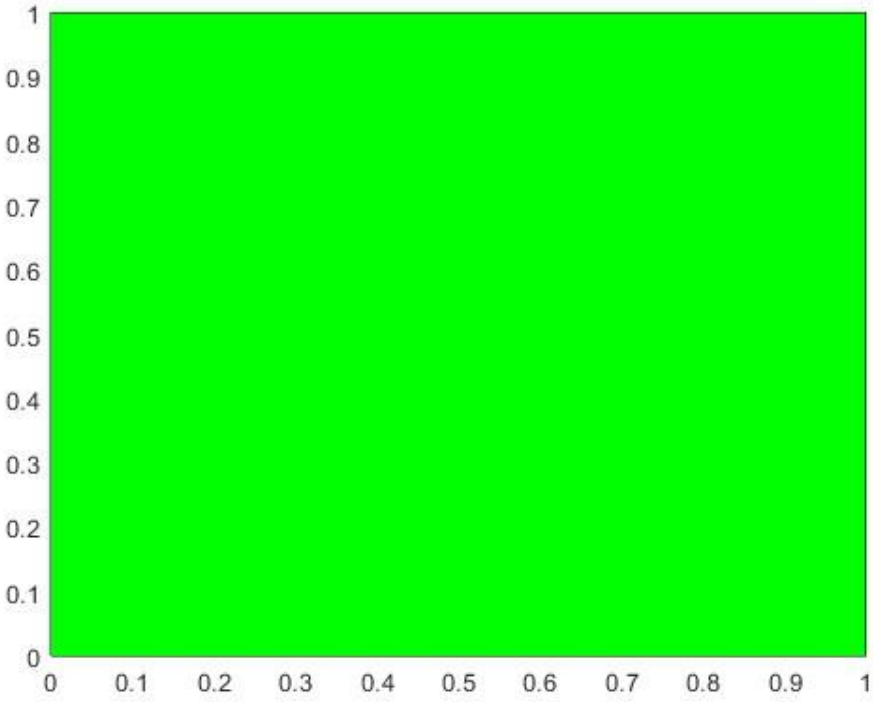
7 Appendix

Problem 1:

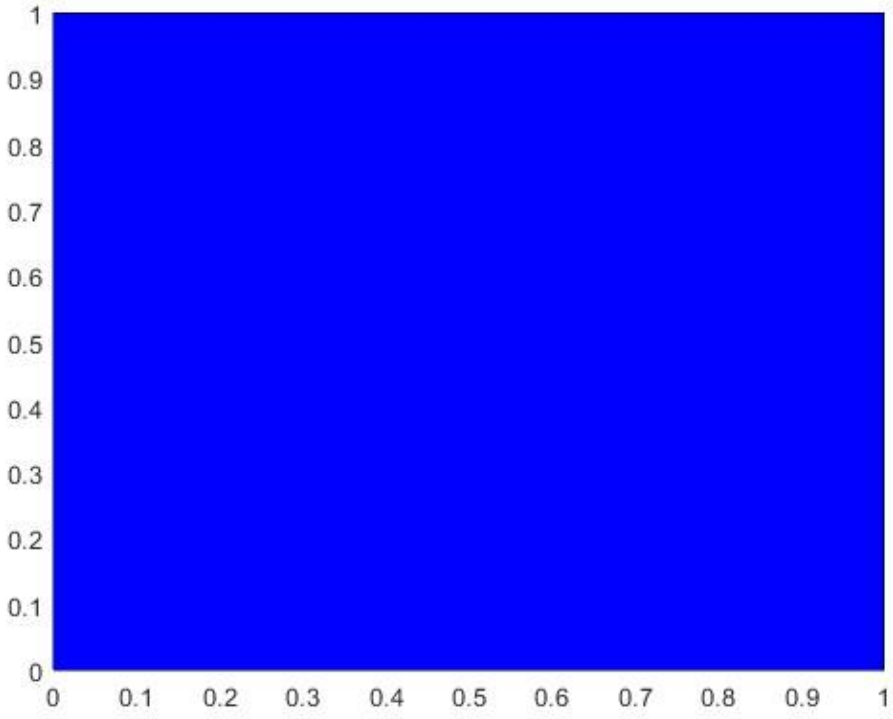
```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [1, 0, 0])
```



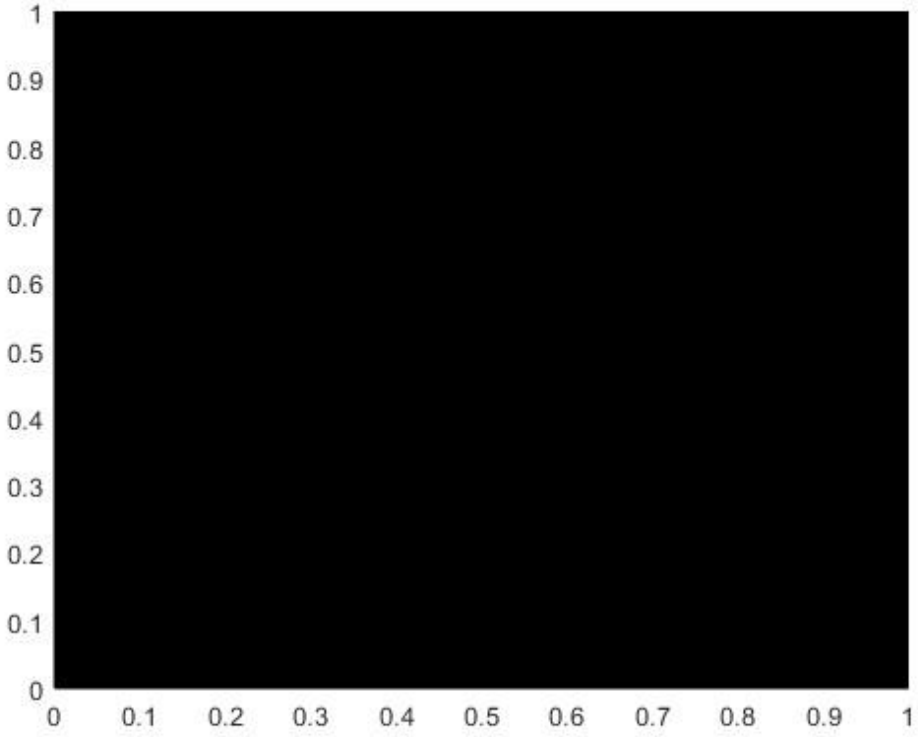

```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [0, 1, 0])
```



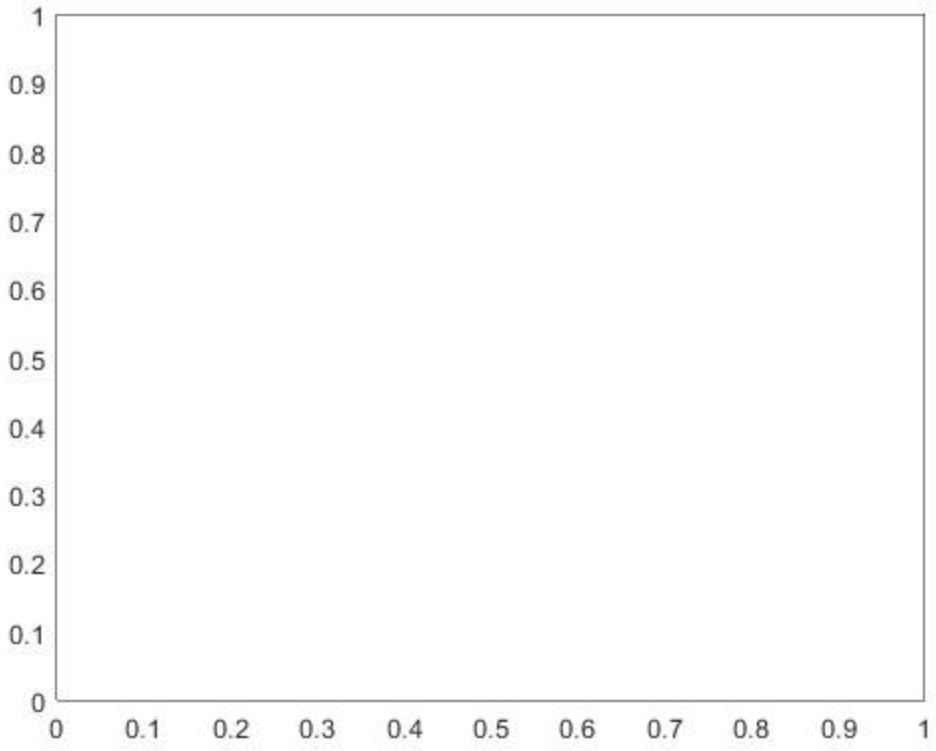

```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [0, 0, 1])
```



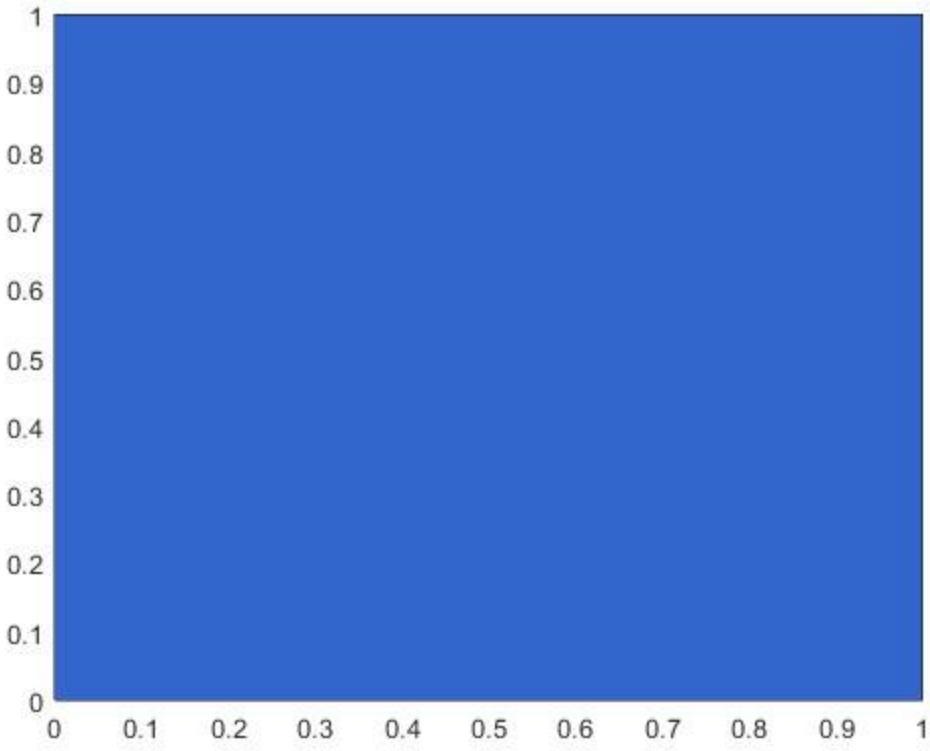
```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [0, 0, 0])
```



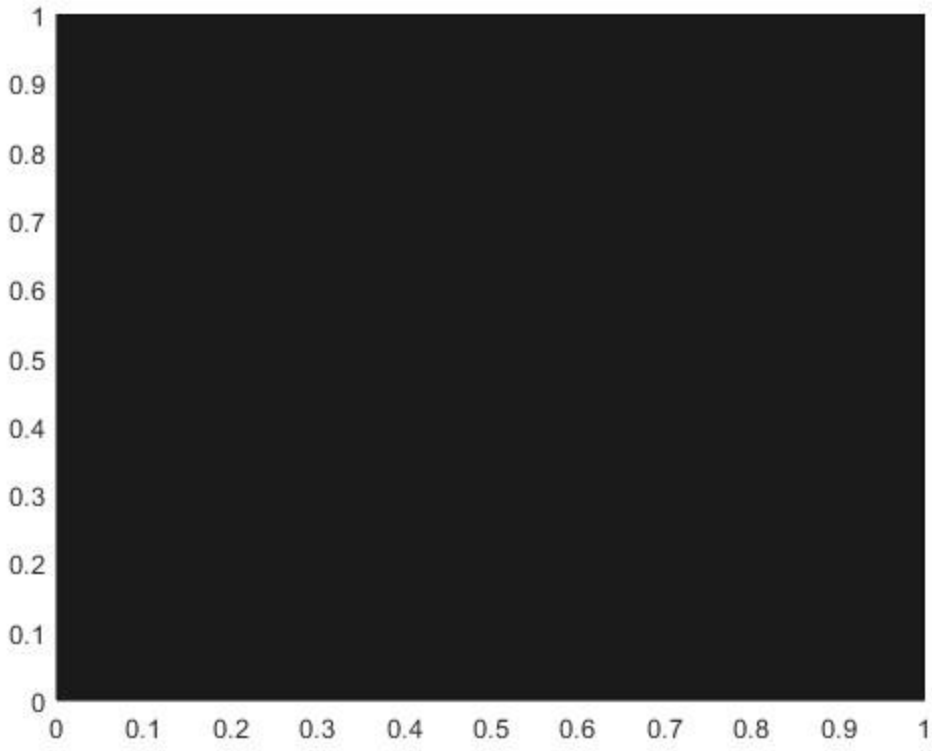

```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [1, 1, 1])
```



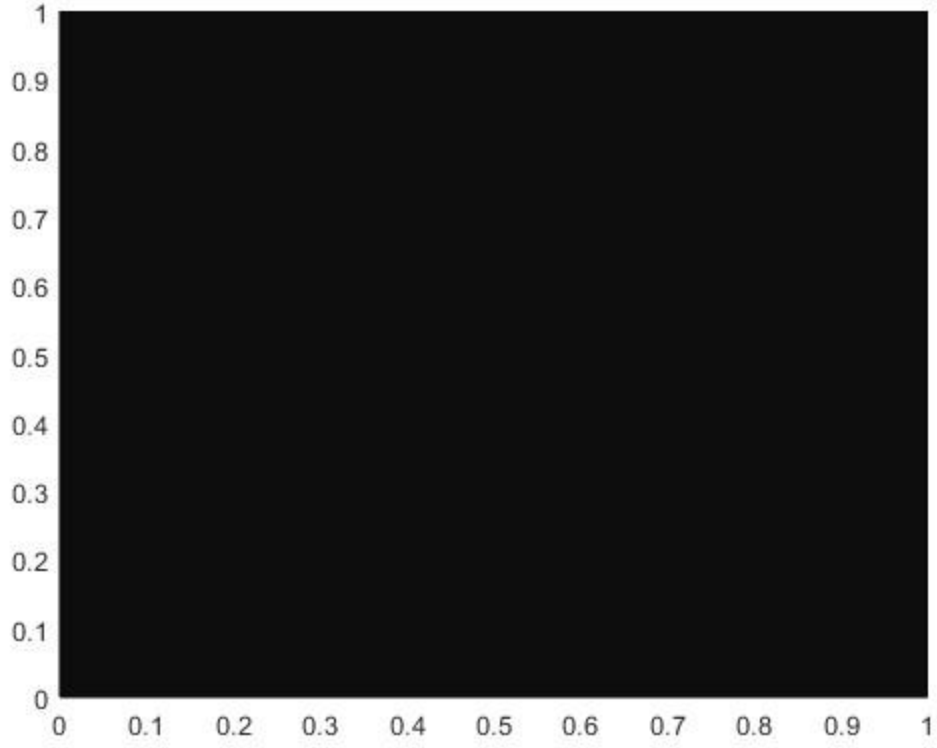

```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [.2, .4, .8])
```



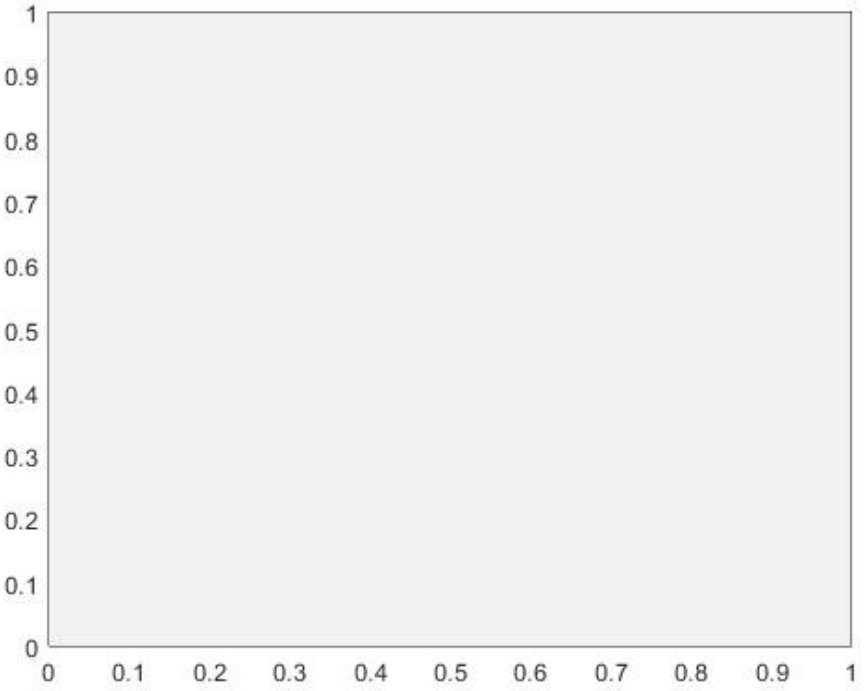

```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [.1, .1, .1])
```




```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [.05, .05, .05])
```

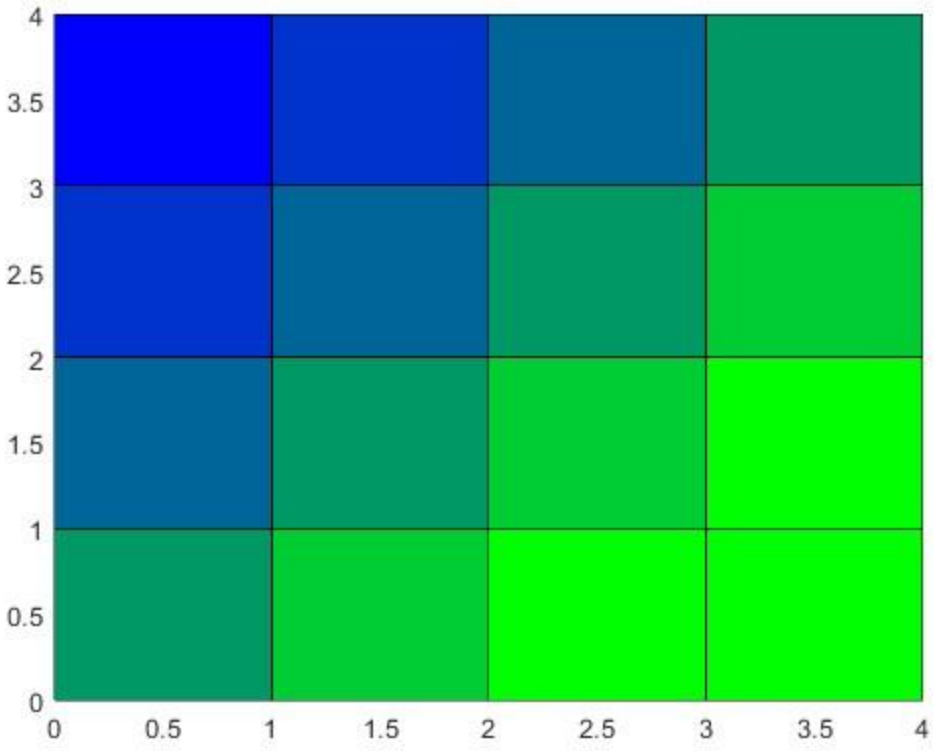



```
x = [0 1 1 0]; y = [0 0 1 1]; patch(x, y, [.95, .95, .95])
```



Problem 2:

```
>> bluetogreen = [ 0 0 1, 0 .2 .8, 0 .4 .6, 0 .6 .4
0 .2 .8, 0 .4 .6, 0 .6 .4, 0 .8 .2
0 .4 .6, 0 .6 .4, 0 .8 .2, 0 1 0
0 .6 .4, 0 .8 .2, 0 1 0, 0 1 0]
M=bluetogreen
Mt=M';
for k=1:48
c(k) = Mt(k);
end
c

bluetogreen =

         0         0    1.0000         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000
         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000
         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0
         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0         0    1.0000         0


M =

         0         0    1.0000         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000
         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000
         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0
         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0         0    1.0000         0



c =

  Columns 1 through 12

         0         0    1.0000         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000

  Columns 13 through 24

         0    0.2000    0.8000         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000

  Columns 25 through 36

         0    0.4000    0.6000         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0

  Columns 37 through 48

         0    0.6000    0.4000         0    0.8000    0.2000         0    1.0000         0         0    1.0000         0

>>
```



0

Problem 3:

```
>> sweden = [ 0 .3 .7, .9 .8 .1, 0 .3 .7, 0 .3 .7
.9 .8 .1, .9 .8 .1, .9 .8 .1, .9 .8 .1
0 .3 .7, .9 .8 .1, 0 .3 .7, 0 .3 .7
0 .3 .7, .9 .8 .1, 0 .3 .7, 0 .3 .7]
M=sweden
Mt=M';
for k=1:48
c(k) = Mt(k);
end
c

sweden =

         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000
         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000


M =

         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000
         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
```

```
c =

  Columns 1 through 16

         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000    0.9000    0.8000    0.1000    0.9000

  Columns 17 through 32

    0.8000    0.1000    0.9000    0.8000    0.1000    0.9000    0.8000    0.1000         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000

  Columns 33 through 48

    0.7000         0    0.3000    0.7000         0    0.3000    0.7000    0.9000    0.8000    0.1000         0    0.3000    0.7000         0    0.3000    0.7000
```