Project Report
On
Digital Image Processing using Linear Algebra using MATLAB

Submitted by:

1. 1pi14ec045 – Pradyumna Mukunda
2. 1pi14ec043 – Niraj Prasad
3. 1pi13ec063 – Nithin
4. 1pi14ec046 – Prajwal Manvi

Program: B.E. Electronics and Communication

Semester: IV

Section: F

Topics Covered:

1. Representation of Image using Matrices
2. Cropping the Image
3. Increasing & Decreasing Brightness
4. Increasing & Decreasing Contrast
5. Increasing/Decreasing Sharpness using Filters
6. HSV coordinates & Increasing Saturation
7. Color Temperature
8. Color Mapping

1. Representation of Image using Matrices

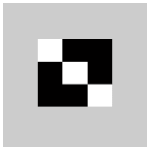
An image is basically a huge matrix. Each element of the matrix represents a pixel. The value of the element determines the brightness of the pixel

e.g $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$

Binary and Grayscale Images

In Binary image the pixel can be either black or white. Value 0 represents black and 1 represents white

e.g. Representation of identity matrix of order 3 as image



$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In grayscale image the pixel value may be from 0 to 255. As the value is increased the color of the pixel goes from black to gray to white.

RGB Image

In RGB (color) image each element of the matrix is a 3 dimensional vector with dimensions R, G & B. R, G and B stand for red, green and blue which are the primary colors from which all other colors can be created. The values of each color can be from 0 to 255.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = \begin{bmatrix} [r_{11} & g_{11} & b_{11}] & [r_{12} & g_{12} & b_{12}] \\ [r_{21} & g_{21} & b_{21}] & [r_{22} & g_{22} & b_{22}] \end{bmatrix}$$

MATLAB Denotations for RGB Image Matrices

Given matrix A as defined above,

Then

$$R = A(:,:,1) = \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{bmatrix} \quad G = A(:,:,2) = \begin{bmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{bmatrix} \quad B = A(:,:,3) = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

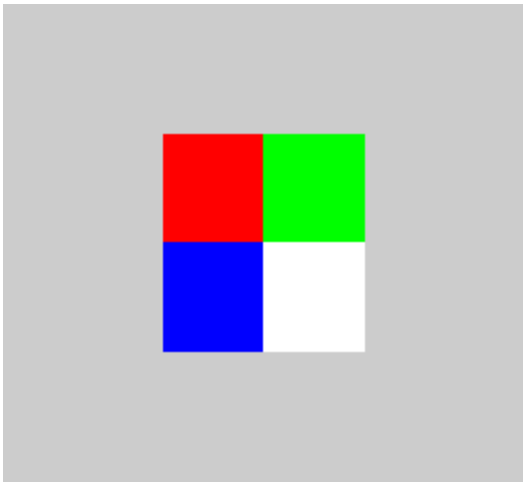
$$A(1,1,:) = a_{11} = \begin{bmatrix} r_{11} \\ g_{11} \\ b_{11} \end{bmatrix} \quad A(1,2,:) = a_{12} = \begin{bmatrix} r_{12} \\ g_{12} \\ b_{12} \end{bmatrix} \quad etc.$$

$$A(1,1,1) = r_{11} \quad A(1,2,2) = g_{12} \quad A(2,1,3) = b_{21} \quad etc.$$

Example:

Given here is an example of a 2 x 2 RGB image

$$A = \begin{bmatrix} [255 & 0 & 0] & [0 & 255 & 0] \\ [0 & 0 & 255] & [255 & 255 & 255] \end{bmatrix}$$



2. Cropping an Image

Cropping an image or a matrix basically means to take a slice out of it

MATLAB denotations for cropping matrices

$$Given\ A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

$$Then \quad A(1{:}2,:,:) = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \quad A(2{:}3,:,:) = \begin{bmatrix} a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad etc$$

$$A(:,1{:}2,:) = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} \quad A(:,2{:}3,:) = \begin{bmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \\ a_{32} & a_{33} \end{bmatrix} \quad etc$$

$$A(1{:}2,1{:}2,:) = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \quad A(1{:}2,2{:}3,:) = \begin{bmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{bmatrix} \quad etc$$

Example:

Here is an image of New York City (newyork.jpg) which we will be using for all our image operations

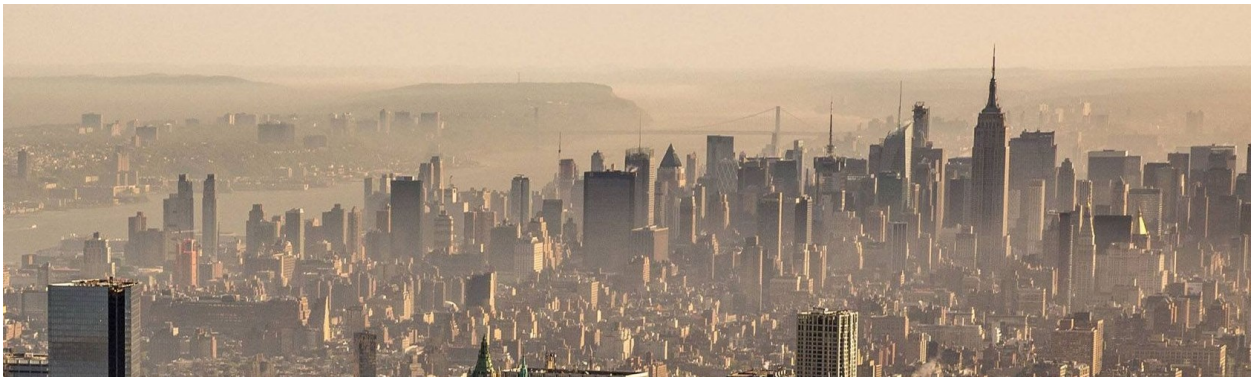
It was cropped using the following code in MATLAB:

```
NY = imread('newyork.jpg');
WTC = NY(1:1500, 900:1200, :);
MM = NY(200:700, 1200:2880, :);
imwrite(WTC, 'worldtradecenter.png')
imwrite(MM, 'midtown.png');
```

Output:

Image of World Trade Center
worldtradecenter.png

Image of Midtown Manhattan
midtown.png



3. Increasing and Decreasing Brightness

Brightness of image can be increased by using simple addition or multiplication operation on pixel values

Conversely brightness is decreased using subtraction or division

Addition:

$$\begin{bmatrix}[r_{11} & g_{11} & b_{11}] & [r_{12} & g_{12} & b_{12}] \\ [r_{21} & g_{21} & b_{21}] & [r_{22} & g_{22} & b_{22}]\end{bmatrix} + k = \begin{bmatrix}[r_{11}+k & g_{11}+k & b_{11}+k] & [r_{12}+k & g_{12}+k & b_{12}+k] \\ [r_{21}+k & g_{21}+k & b_{21}+k] & [r_{22}+k & g_{22}+k & b_{22}+k]\end{bmatrix}$$

The side effect of addition operation is a *decrease* in the contrast of the image

Example: Add 64

```
NY = imread('newyork.jpg');
NY1 = NY + 64;
imwrite(NY1, 'newyorkadd.png');
```

Subtraction:

$$\begin{bmatrix}[r_{11} & g_{11} & b_{11}] & [r_{12} & g_{12} & b_{12}] \\ [r_{21} & g_{21} & b_{21}] & [r_{22} & g_{22} & b_{22}]\end{bmatrix} - k = \begin{bmatrix}[r_{11}-k & g_{11}-k & b_{11}-k] & [r_{12}-k & g_{12}-k & b_{12}-k] \\ [r_{21}-k & g_{21}-k & b_{21}-k] & [r_{22}-k & g_{22}-k & b_{22}-k]\end{bmatrix}$$

The side effect of subtraction operation is an *increase* in the contrast of the image

Example: Subtract 64

```
NY = imread('newyork.jpg');
NY2 = NY - 64;
imwrite(NY2, 'newyorksub.png');
```

Multiplication:

$$\begin{bmatrix}[r_{11} & g_{11} & b_{11}] & [r_{12} & g_{12} & b_{12}] \\ [r_{21} & g_{21} & b_{21}] & [r_{22} & g_{22} & b_{22}]\end{bmatrix} \times k = \begin{bmatrix}[kr_{11} & kg_{11} & kb_{11}] & [kr_{12} & kg_{12} & kb_{12}] \\ [kr_{21} & kg_{21} & kb_{21}] & [kr_{22} & kg_{22} & kb_{22}]\end{bmatrix}$$

The side effect of multiplication operation is an *increase* in the contrast of the image

Example:

Multiply by 1.5

```
NY = imread('newyork.jpg');
NY3 = NY * 1.5;
imwrite(NY3, 'newyorkmul.png');
```

Division:

$$\begin{bmatrix}[r_{11} & g_{11} & b_{11}] & [r_{12} & g_{12} & b_{12}] \\ [r_{21} & g_{21} & b_{21}] & [r_{22} & g_{22} & b_{22}]\end{bmatrix} \div k = \begin{bmatrix}[r_{11}/k & g_{11}/k & b_{11}/k] & [r_{12}/k & g_{12}/k & b_{12}/k] \\ [r_{21}/k & g_{21}/k & b_{21}/k] & [r_{22}/k & g_{22}/k & b_{22}/k]\end{bmatrix}$$

The side effect of division operation is a *decrease* in the contrast of the image

Example:

Divide by 2

```
NY = imread('newyork.jpg');
NY4 = NY / 2;
imwrite(NY4, 'newyorkdiv.png');
```

4. Increasing and Decreasing Contrast

Contrast of the image is the variance of the pixel values. We have seen that addition and division operations *decrease* contrast whereas subtraction and multiplication operations *increase* contrast. To increase or decrease contrast without affecting brightness we use a combination of the two operations.

We assume a central value, say 128. The respective operations for increasing or decreasing contrast are manipulated such that this central value remains constant after both the operations, however the variance of the values changes

Increasing Contrast: Multiply and Subtract

Multiply the image by 1.5 and then subtract 64
The value 128 remains unchanged

NY = imread('newyork.jpg');
*NY1 = (NY * 1.5) - 64;*
imwrite(NY1, 'newyorkcont+.jpg');

Decreasing Contrast: Divide and Add

Divide the image by 2 and then add 64
The value 128 remains unchanged

NY = imread('newyork.jpg');
NY2 = (NY / 2) + 64 ;
imwrite(NY2, 'newyorkcont-.jpg');

5. Increasing/Decreasing Sharpness using Filters

To blur or sharpen an image we use another matrix known as a filter. We perform convolution operation between original image and a filter H to get a filtered image.

e.g. $A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$ $\quad H = \begin{bmatrix} r & s & t \\ u & \boldsymbol{v} & w \\ x & y & z \end{bmatrix}$

For image filtering we define convolution as the sum of the scalar products of the element $\mathbf{a_{ij}}$ of image matrix A with the filter matrix H as the element **v** located in center position of H 'slides' across all the elements of A in both dimensions

i.e $A * H = a_{11}\begin{bmatrix} \boldsymbol{v} & w & 0 \\ y & z & 0 \\ 0 & 0 & 0 \end{bmatrix} + a_{12}\begin{bmatrix} u & \boldsymbol{v} & w \\ x & y & z \\ 0 & 0 & 0 \end{bmatrix} + a_{13}\begin{bmatrix} 0 & u & \boldsymbol{v} \\ 0 & x & y \\ 0 & 0 & 0 \end{bmatrix} + a_{21}\begin{bmatrix} s & t & 0 \\ \boldsymbol{v} & w & 0 \\ y & z & 0 \end{bmatrix} + a_{22}\begin{bmatrix} r & s & t \\ u & \boldsymbol{v} & w \\ x & y & z \end{bmatrix} +$
$a_{23}\begin{bmatrix} 0 & r & s \\ 0 & u & \boldsymbol{v} \\ 0 & x & y \end{bmatrix} + a_{31}\begin{bmatrix} 0 & 0 & 0 \\ s & t & 0 \\ \boldsymbol{v} & w & 0 \end{bmatrix} + a_{32}\begin{bmatrix} 0 & 0 & 0 \\ r & s & t \\ u & \boldsymbol{v} & w \end{bmatrix} + a_{33}\begin{bmatrix} 0 & 0 & 0 \\ 0 & r & s \\ 0 & u & \boldsymbol{v} \end{bmatrix}$

Whether the filter is a *blurring* filter or a *sharpening* filter depends upon the nature of the matrix H

For *blurring* we will use the matrix $H_1 = \begin{bmatrix} 0.111 & 0.111 & 0.111 \\ 0.111 & 0.111 & 0.111 \\ 0.111 & 0.111 & 0.111 \end{bmatrix}$ and for *sharpening* we will use the matrix $H_2 = \begin{bmatrix} 0.04 & -0.2 & 0.04 \\ -0.2 & 1 & -0.2 \\ 0.04 & -0.2 & 0.04 \end{bmatrix}$

For an RGB image the matrix must be split into R, G and B components and convolution operation performed separately for each

NOTE: The image was scaled to 1/10 times its original size to make the blurring/sharpening effects more visible

```
NY = imread('newyork.jpg');
NY = imresize(NY, 0.1);
R = NY(:,:,1);
G = NY(:,:,2);
B = NY(:,:,3);
H1 = [0.111 0.111 0.111; 0.111 0.111 0.111; 0.111 0.111 0.111];
NY1(:,:,1) = imfilter(R,H1);
NY1(:,:,2) = imfilter(G,H1);
NY1(:,:,3) = imfilter(B,H1);
H2 = [0.04 -0.2 0.04; -0.2 1 -0.2; 0.04 -0.2 0.04];
NY2(:,:,1) = 2.75 * imfilter(R,H2);
NY2(:,:,2) = 2.75 * imfilter(G,H2);
NY2(:,:,3) = 2.75 * imfilter(B,H2);
imwrite(NY, 'newyorksmall.bmp');
imwrite(NY1, 'newyorkblur.bmp');
imwrite(NY2, 'newyorksharp.bmp');
```

Output:

Reduced size Image

Blurred:

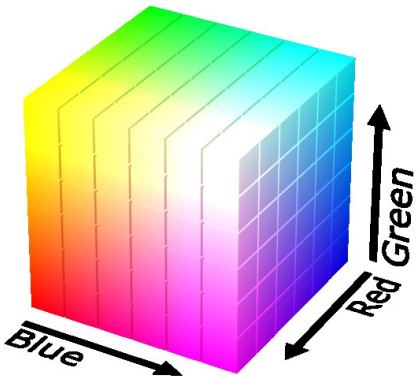
Sharpened:

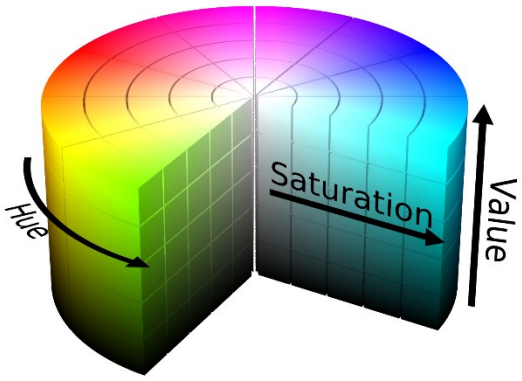
6. HSV coordinates and Increasing Saturation

Till now our pixels were expressed using RGB coordinates. R, G and B coordinates in the color space are analogous to Cartesian x, y and z coordinates in 3D space.

However there is another coordinate system known as HSV which we will be using for our next operation. H, S and V coordinates in the color space are analogous to cylindrical θ , r and h coordinates in 3D space.

RGB coordinates:





HSV Coordinates:

In HSV,

H stands for **Hue**. This coordinate is analogous to the cylindrical θ coordinate. Hue takes values from $0° \; to \; 360°$. As H increases, the color of the pixel varied from red to yellow to green to cyan to blue to magenta and back to red

S stands for **Saturation**. It is analogous to the cylindrical *r* coordinate. It varies from 0 to 100%. Saturation represents how vibrant the color of the pixel is. 0% saturation denotes neutral colors (black, white, grey, silver etc.) while 100% saturation denotes very vibrant colors. Pure red, pure green and pure blue all have 100% saturation.

V stands for **Value.** It is analogous to the cylindrical *h* coordinate. It varies from 0 to 100%. Value denotes the brightness of the pixel. As V drops from 100% to 0% the color of the pixel becomes darker. 0% value always represents black color irrespective of hue and saturation.

Increasing & Decreasing Saturation:

We increase Saturation of the image if it looks very dull to make it more colorful.

MATLAB has inbuilt functions to convert RGB to HSV and HSV to RGB coordinates. Here we multiply the S coordinate by 2 to increase saturation.

```
NY = imread('newyork.jpg');
NY1 = rgb2hsv(NY);
S = NY1(:,:,2);
S = S*2;
NY1(:,:,2) = S;
NY2 = hsv2rgb(NY1);
imwrite(NY2, 'newyorksat+.png');
```

Black and White:

It goes without saying that if we reduce the saturation to zero we get a black and white image

```
NY = imread('newyork.jpg');
NY1 = rgb2hsv(NY);
S = NY1(:,:,2);
S = NY1(:,:,2);
S = S*0;
NY1(:,:,2) = S;
NY2 = hsv2rgb(NY1);
imwrite(NY2, 'newyorksat0.png');
```

7. Color Temperature

By manipulating the R and B matrices we can make the colors of the image “warmer” or “cooler”. Redder hues are known as “warm” colors and bluer hues are known as “cooler” colors.

Decreasing color temperature:

Multiply the R matrix by 0.9 and the B matrix by 1.1. Then add 32. This cancels the natural red lighting of the picture due to sunset.

```
NY = imread('newyork.jpg');
R = NY(:,:,1);
G = NY(:,:,2);
B = NY(:,:,3);
NY1(:,:,1) = R*0.9;
NY1(:,:,2) = G;
NY1(:,:,3) = B*1.1;
imwrite(NY1 + 32, 'newyorkblue.jpg');
```

Increasing color temperature:

Multiply the R matrix by 1.1 and the B matrix by 0.9. This enhances the sunset effect

```
NY = imread('newyork.jpg');
R = NY(:,:,1);
G = NY(:,:,2);
B = NY(:,:,3);
NY2(:,:,1) = R* 1.1;
NY2(:,:,2) = G;
NY2(:,:,3) = B*0.9;
imwrite(NY2, 'newyorkred.jpg');
```

8. Color Mapping

Usually while representing data in statistics, topography, astronomy etc. we like to use colors to make our diagrams more attractive and readable. For this purpose we use color mapping.

Color Space

Color Space is like a vector space for colors. In 3D vector space we represent any point by a radius vector which can be broken down into 3 orthogonal components. We require 3 variables: x, y and z to span the entire space

$$a = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = x \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + y \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + z \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

In our color space also we use 3 variables: r, g and b to represent any color in the entire space

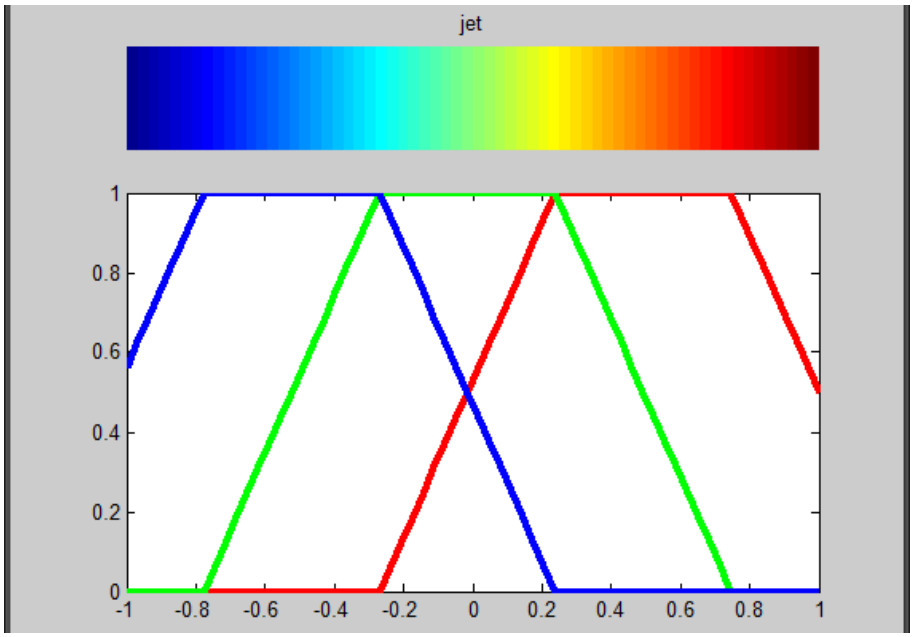
$$c = \begin{bmatrix} r \\ g \\ b \end{bmatrix} = r \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} + g \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + b \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

However since the color space is finite and discrete it is possible to represent any color in the entire space using a *single* variable k. This is the principle behind color mapping.

Jet Color Map:

It was mentioned that it is possible to represent the entire color space using a single variable k. However for our application, that is, representing data using colors, we are going to use only a part of the color space.

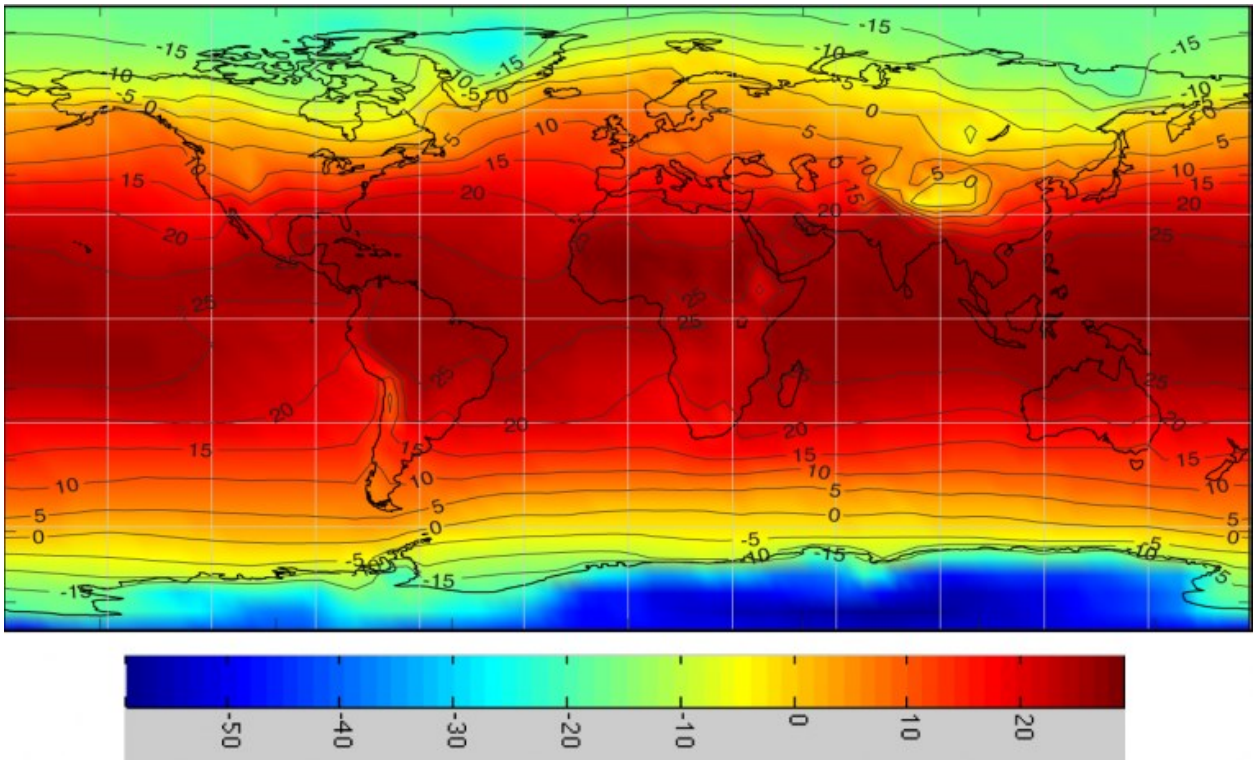
In Jet Color map we represent the maximum of the data in dark red color and the minimum of the data in dark blue. The jet color map is defined below:



The data values are converted to k values such that the maximum of the data is represented as 1 and the minimum of the data is represented as -1. The range of the variable k here is arbitrarily defined as -1 to 1 (double), however it may be defined as 0 to 1 (double), 0 to 255 (8-bit int), 0 to 65535 (16-bit int) etc.

Example:

Given below is a world map of the annual average temperature, taken from 1961 to 1990, that uses jet



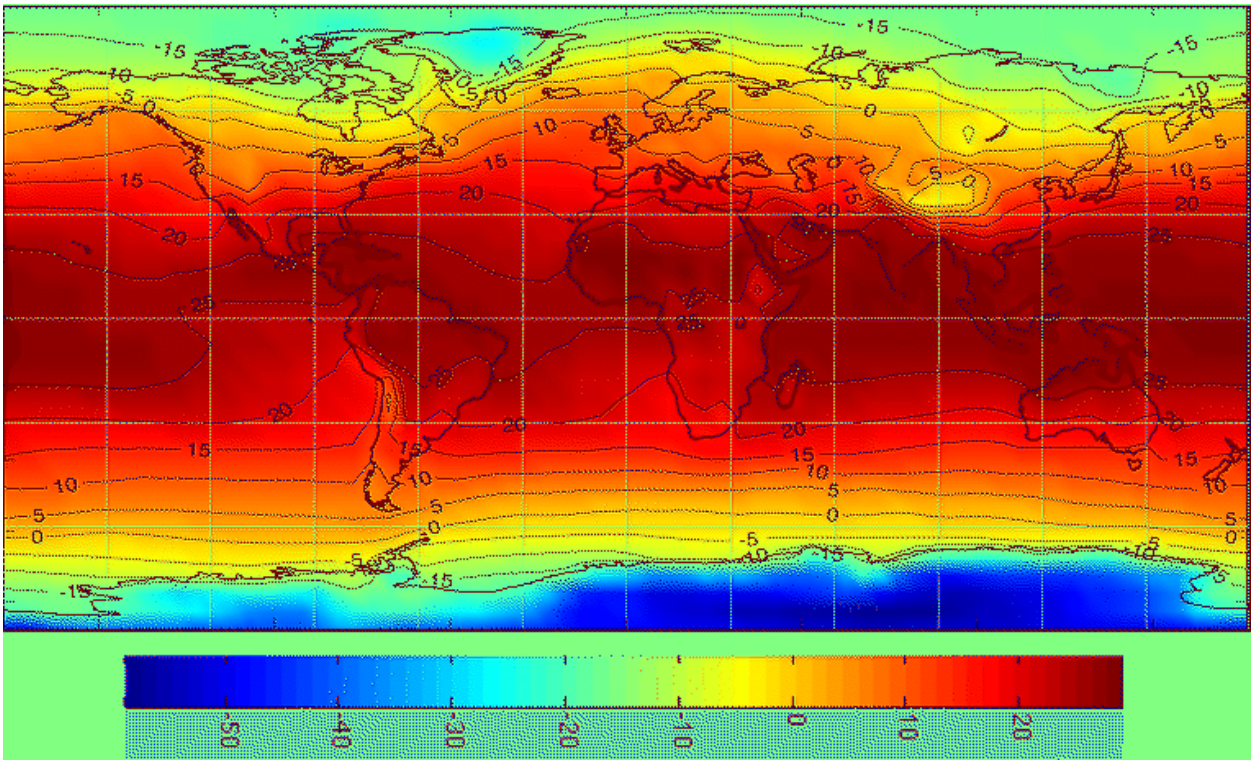
color map

This image was downloaded from the Internet. We can manipulate the colors of the image using MATLAB. In MATLAB the k values are 8 bit integers from 0 to 255 and are known as *indexes*. The colors of the RGB Image when converted to indexes form an *Indexed Image.* Any color outside the color space, such as white, are approximated to the nearest color using projection. (Here white is converted to green)

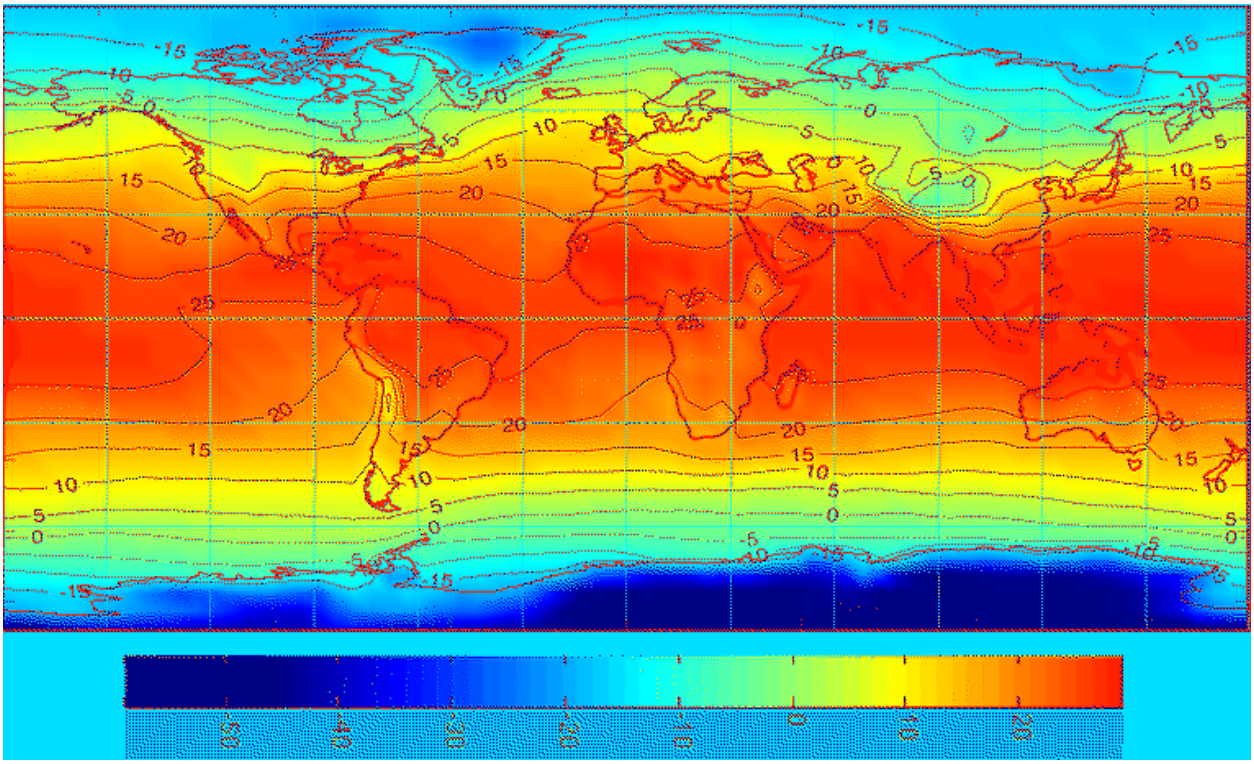
Notice that the scale at the bottom of the map also shifts with the colors.

A = imread('Temperature.png');
K = rgb2ind(A,jet);

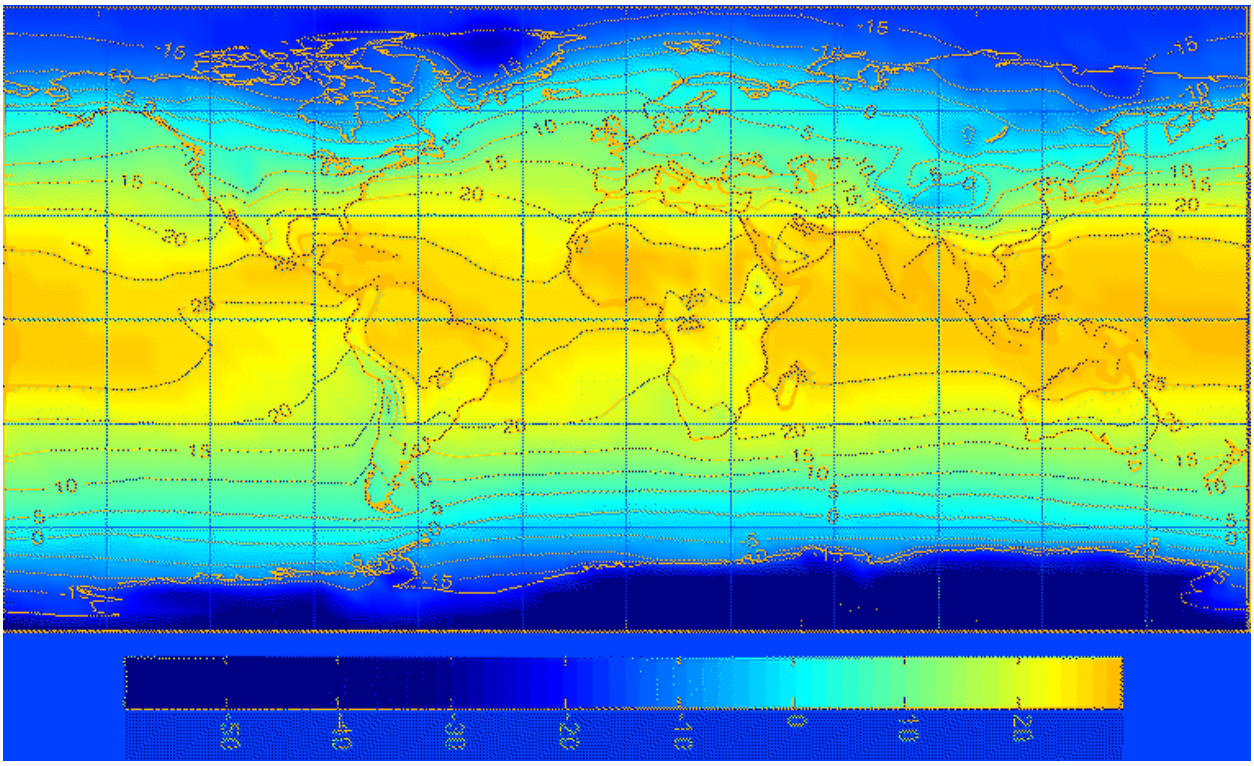
imwrite(ind2rgb(K,jet),'Temperature00.png');
imwrite(ind2rgb(K-40,jet),'Temperature40.png');
imwrite(ind2rgb(K-80,jet),'Temperature80.png');
*imwrite(ind2rgb((K-128)*2,jet),'Temperature99.png');*



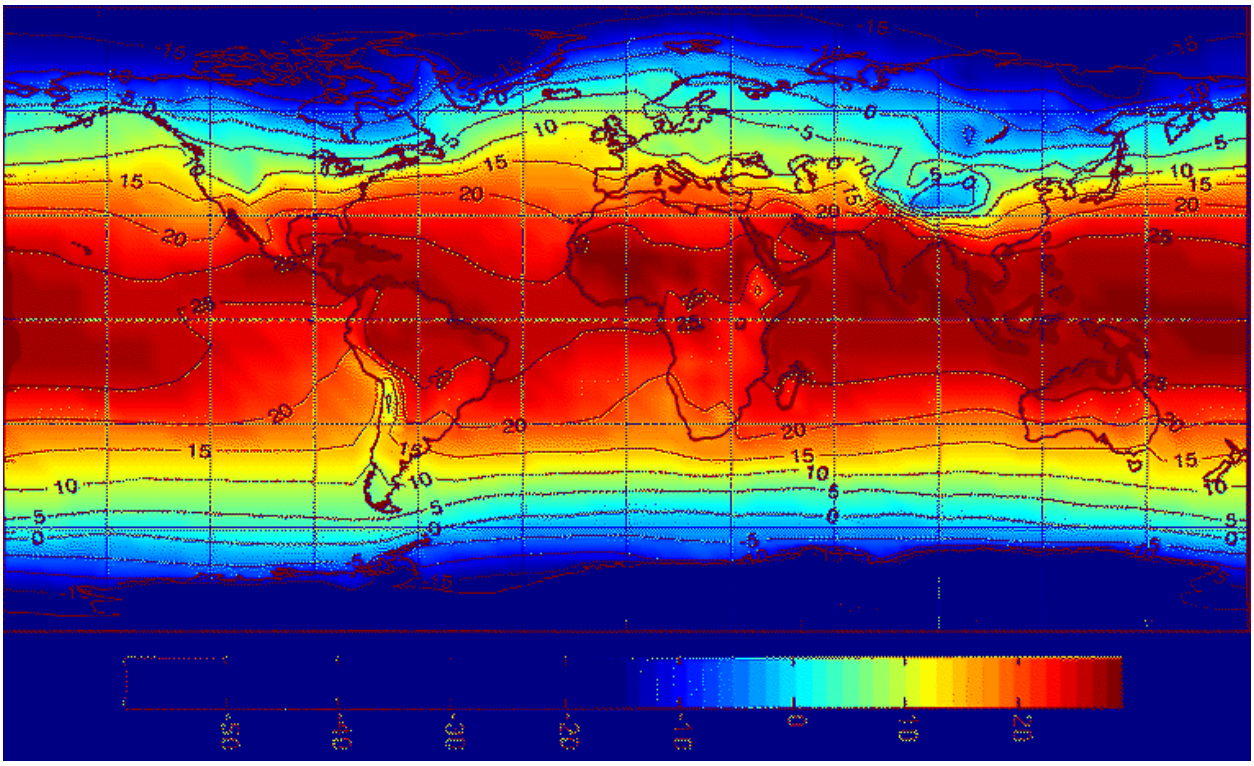
Original Image:



Subtract 40 from Indexed Image K:



Subtract 80 from Indexed image K:



Subtract 128 and multiply by 2: