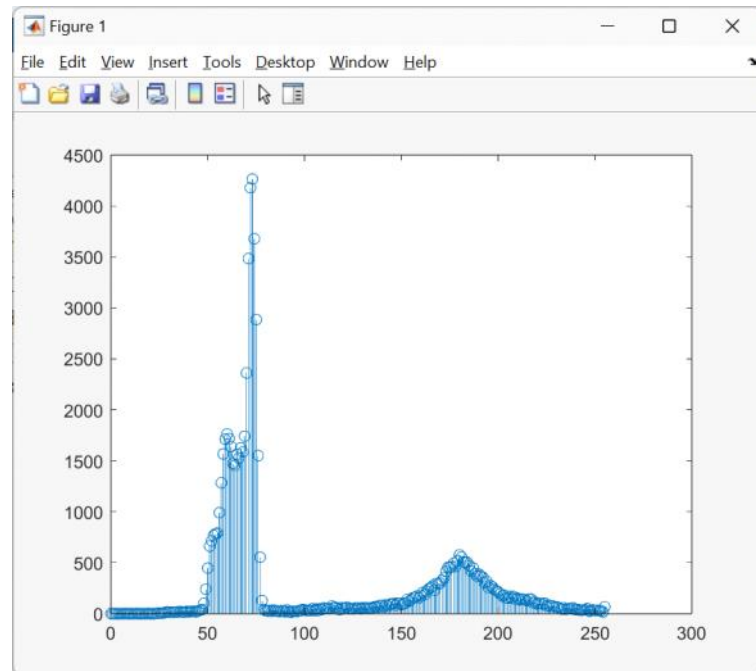


# Lecture 2

## Imhist function (cont'd)

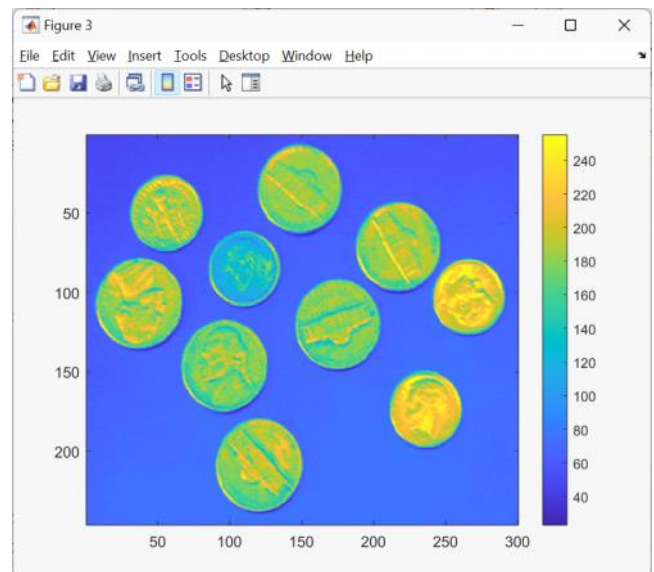
`[counts,binLocations] = imhist(I)` calculates the histogram for the grayscale image `I`. The `imhist` function returns the histogram counts in `counts` and the bin locations in `binLocations`. The number of bins in the histogram is determined by the image type.

```
>> [c, b] = imhist(I);  
>> stem(b, c)  
  
>> c(1:10)'  
  
ans =  
  
    0    0    0    0    0    0    0    0    0    0  
  
>> b(1:10)'  
  
ans =  
  
    0    1    2    3    4    5    6    7    8    9
```

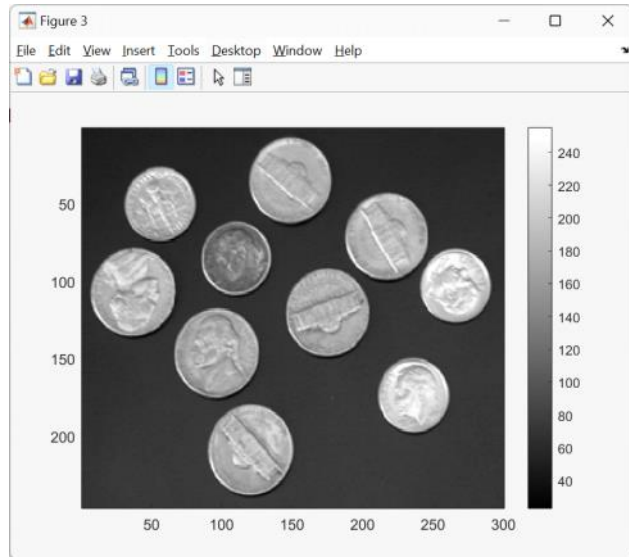


- `imagesc` function

```
>> figure; imagesc(I)  
>> colorbar
```



```
>> colormap('gray')
```



- Color Images

```
>> I = imread('football.jpg');  
>> imshow(I)
```

```
>> whos I  
Name      Size      Bytes Class  
Attributes
```

```
I        256x320x3    245760 uint8
```

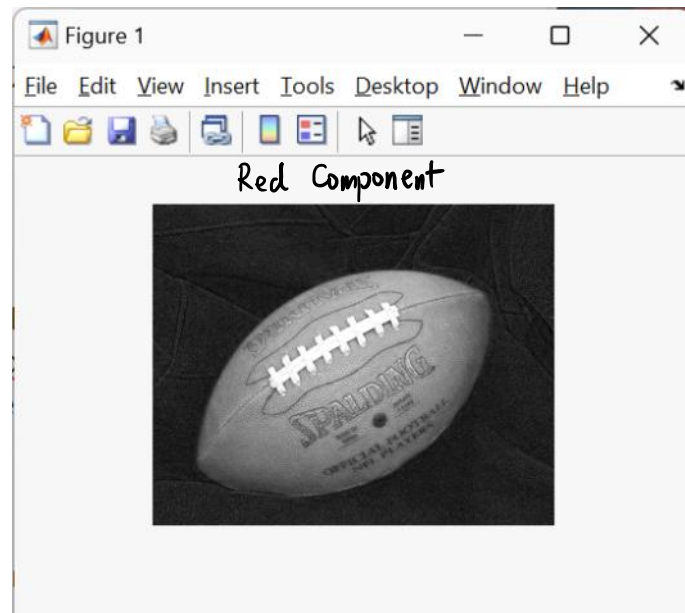
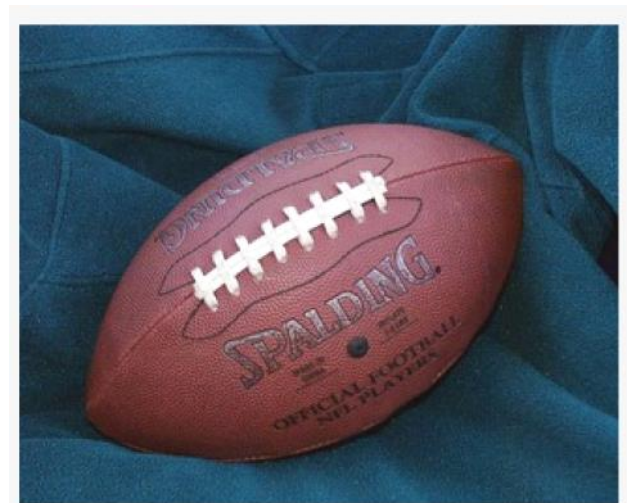
*R, G, B Components*

```
>> R = I(:,:,1);
```

```
>> whos R  
Name      Size      Bytes Class  Attributes
```

```
R        256x320      81920 uint8
```

```
>> figure; imshow(R)
```





```
>> G = I(:,:,2);
>> B = I(:,:,3);
>> whos G
Name      Size      Bytes Class  Attributes
G         256x320    81920 uint8
```

```
>> whos B
Name      Size      Bytes Class  Attributes
B         256x320    81920 uint8
```

```
>> R(146,146) + G(146,146) + B(146,146)
```

```
ans =
```

```
uint8
```

```
255
```

```
>> R(146,146)    >> G(146,146)    >> B(146,146)
ans =             ans =             ans =
uint8             uint8             uint8
119               129
```

```
uint8
```

```
185
```

```
>> (185 + 119 + 129)
```

```
433
ans
```

⇒ Truncated to 255 ( uint8 type )

**Solution: Need to convert uint8 type to the double type for arithmetic operations.**

```
>> R = ID(:,:,1); G = ID(:,:,2); B = ID(:,:,3);
```

```
>> AD = (R + G + B)/3;
```

```
>> AD(146, 146)
```

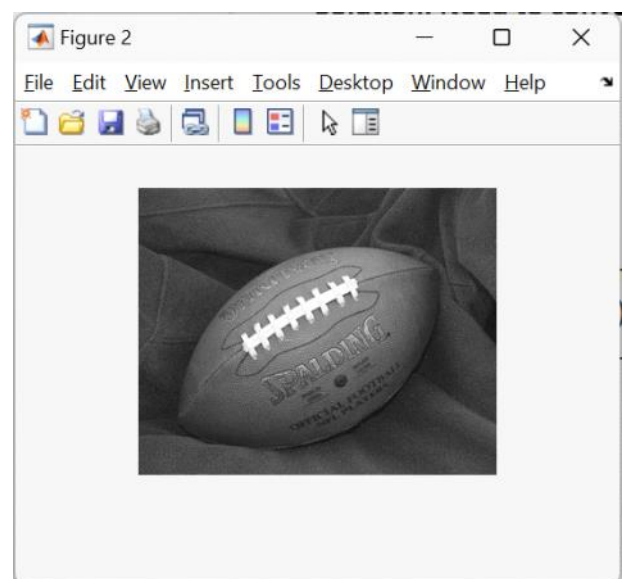
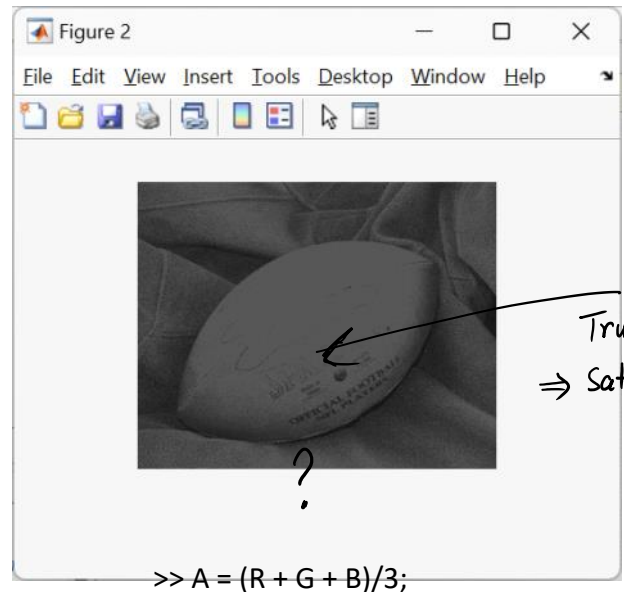
```
ans =
```

```
144.3333
```

Convert the result back to uint8 type

```
>> AD_uint8 = uint8(AD);
```

```
>> figure; imshow (AD_uint8)
```

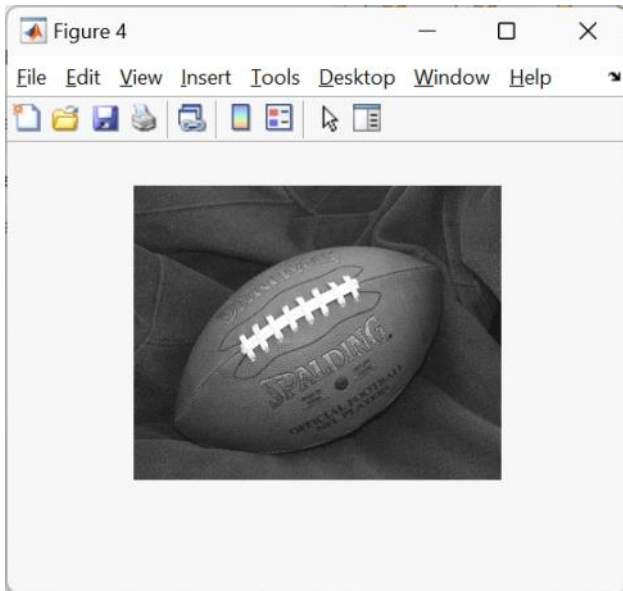




**rgb2gray** converts RGB values to grayscale values by forming a weighted sum of the *R*, *G*, and *B* components:

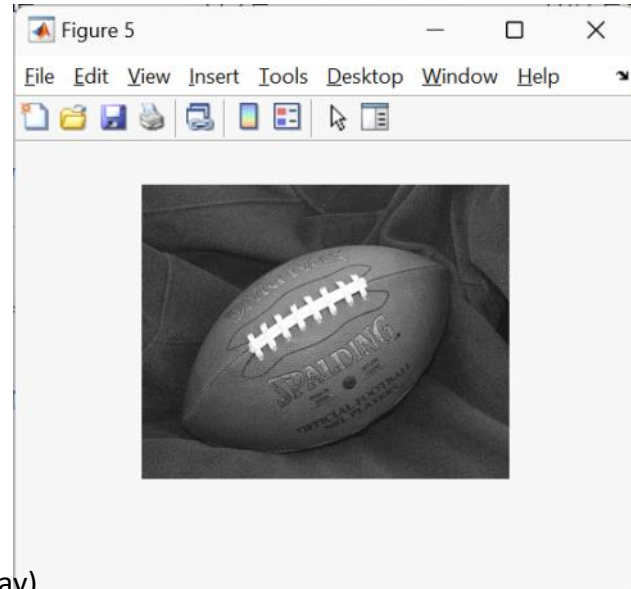
$$0.2989 * R + 0.5870 * G + 0.1140 * B$$

```
>> I_rgb2gray = rgb2gray(I);
>> figure; imshow(I_rgb2gray)
```



```
>> AW = 0.2989 * R + 0.5870 * G + 0.1140 * B;
>> whos AW
Name      Size      Bytes Class  Attributes
AW       256x320    655360 double
```

```
>> AW_uint8 = uint8(AW);
```



identical  
↔

```
>> isequal(AW_uint8, I_rgb2gray)
```

```
ans =
logical
1
```

- Size of the JPG image file

On the hard drive: 27,130 football.jpg

```
>> whos I
Name      Size      Bytes Class  Attributes
I       256x320x3    245760 uint8
           ~~~~~
           Raw Image Size
```

Raw image was compressed from 245760 bytes to 27130 bytes with a compression ratio about 9:1

```
>> 245760/27130
```

```
ans =
```

9.0586

<https://hexed.it/>

The screenshot displays the hexed.it web application interface. At the top, there is a navigation bar with icons for New file, Open file, Save as, Undo, Redo, Tools, Settings, and Help. Below this, the application shows the file 'football.jpg' with a size of 27,130 bytes (27 KiB). The main area is divided into two sections: 'File Information' and 'Data Inspector (Little-endian)'. The 'Data Inspector' shows a hex dump of the file's content, with the first few bytes highlighted in blue. A handwritten circle around the 'FF D8 FF' bytes is labeled 'JPEG'. The hex dump shows the following data:

Offset	Hex	ASCII
00000000	FF D8 FF E0 00 10 4A 46 49 46 00 01 01 00 00 01	+
00000010	00 01 00 00 FF DB 00 43 00 03 02 02 03 02 02 03	α JFIF
00000020	03 03 03 04 03 03 04 05 08 05 05 04 04 05 0A 07	
00000030	07 06 08 0C 0A 0C 0C 0B 0A 0B 0B 0D 0E 12 10 0D	
00000040	0E 11 0E 0B 0B 10 16 10 11 13 14 15 15 15 0C 0F	
00000050	17 18 16 14 18 12 14 15 14 FF DB 00 43 01 03 04	.C
00000060	04 05 04 05 09 05 05 09 14 0D 0B 0D 14 14 14 14	
00000070	14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14	
00000080	14 14 14 14 14 14 14 14 14 14 14 14 14 14 14 14	
00000090	14 14 14 14 14 14 14 14 14 14 14 14 14 14 FF C0	L
000000A0	00 11 08 01 00 01 40 03 01 22 00 02 11 01 03 11	@."
000000B0	01 FF C4 00 1F 00 00 01 05 01 01 01 01 01 01 00	-
000000C0	00 00 00 00 00 00 00 01 02 03 04 05 06 07 08 09	
000000D0	0A 0B FF C4 00 B5 10 00 02 01 03 03 02 04 03 05	-
000000E0	05 04 04 00 00 01 7D 01 02 03 00 04 11 05 12 21	} !
000000F0	31 41 06 13 51 61 07 22 71 14 32 81 91 A1 08 23	1A..Qa."q.2üæi.#
00000100	42 B1 C1 15 52 D1 F0 24 33 62 72 82 09 0A 16 17	B R=\$3bré...
00000110	18 19 1A 25 26 27 28 29 2A 34 35 36 37 38 39 3A	...%&'()*456789:
00000120	43 44 45 46 47 48 49 4A 53 54 55 56 57 58 59 5A	CDEFGHIJSTUVWXYZ
00000130	63 64 65 66 67 68 69 6A 73 74 75 76 77 78 79 7A	cdefghijstuvwxyz
00000140	83 84 85 86 87 88 89 8A 92 93 94 95 96 97 98 99	ââââçèèè#ôôôôùÿ