Numb3rs

Number and image types

CS112 Scientific Computation
Department of Computer Science
Wellesley College

Numb3rs

All of our numbers so far have been of type double for double-precision floating point

\[ 6.6260755 \times 10^{34} \]

double numbers use 8 bytes (64 bits) to store each number -- 52 for the mantissa (~ 16 significant digits), 11 bits for exponent, and one sign bit

High precision requires more memory space and processing time! And may not be needed....
Other number types

<table>
<thead>
<tr>
<th>Name</th>
<th>Size(bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>uint8</td>
<td>1</td>
<td>integers 0 to 255</td>
</tr>
<tr>
<td>uint16</td>
<td>2</td>
<td>integers 0 to 65,535</td>
</tr>
<tr>
<td>uint32</td>
<td>4</td>
<td>integers 0 to 4,294,967,295</td>
</tr>
<tr>
<td>int8</td>
<td>1</td>
<td>integers -128 to 127</td>
</tr>
<tr>
<td>int16</td>
<td>2</td>
<td>integers -32,768 to 32,767</td>
</tr>
<tr>
<td>int32</td>
<td>4</td>
<td>integers -2,147,483,648 to 2,147,483,647</td>
</tr>
<tr>
<td>single</td>
<td>4</td>
<td>single-precision floating point</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(fewer bits for mantissa &amp; exponent)</td>
</tr>
<tr>
<td>double</td>
<td>8</td>
<td>double-precision floating point</td>
</tr>
</tbody>
</table>

There is also a logical type to represent binary values 0 (false) and 1 (true), stored in 1 byte.

For each number type ...

... there is a built-in function of the same name that converts its input to the desired type:

```
>> bigNums = zeros(1, 100);
>> smallNums = uint8(zeros(1, 100));
```

> who

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Bytes</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigNums</td>
<td>1x100</td>
<td>800</td>
<td>double array</td>
</tr>
<tr>
<td>smallNums</td>
<td>1x100</td>
<td>100</td>
<td>uint8 array</td>
</tr>
</tbody>
</table>

zeros and ones can also be called with a third input:

```
>> smallNums = zeros(1, 100, 'uint8');
```

* What happens if we try to store a number in smallNums that is less than 0 or greater than 255? How are fractional numbers handled?
The silver screen

Some of our images have been black and white images stored in matrices of double type values spanning the range from 0.0 to 0.1

To conserve memory space, images are typically stored in files using formats that represent each image intensity using a small number of bits* (e.g., 1, 8, 16 bits)

*note that the human eye can only distinguish about 100 shades of gray at one time

MATLAB’s image processing toolbox

MATLAB’s imread and imwrite read and write images in many possible formats, such as JPEG, GIF, TIFF, BMP, PNG

Any of these image types can then be displayed using imshow

>> mona = imread('monaLisa.jpg');
>> imshow(mona);
>> imwrite(mona, 'monaLisa.png');

MATLAB uses the specified second filename to determine the format for the stored image
Storing a masterpiece

Recall that in an RGB color image, each picture element (pixel) consists of three values:

red  green  blue

Mona is three-dimensional

>> whos
Name  Size       Bytes  Class
mona  864x560x3  1451520  uint8 array

Third dimension has three indices, corresponding to the amount of red, green, and blue at each image location, specified as an integer between 0 and 255

RGB values at each location can be viewed with imtool(mona)
Exercise

How did I create this figure, where the gray-level images show red, green and blue components?

Suppose you want to show the components in shades of red/green/blue?

```matlab
imshow with a colormap

`imshow` can display an indexed image with a colormap:

```matlab
definitions = zeros(300, 300, 'uint8');
for pos = 50:50:200
    definition(pos:pos+50, pos:pos+50) = pos;
end

>> imshow(definition, [0 200])
```

>> imshow(definition, jet)
>> colorbar
```
A "real" indexed image

>> [image, cmap] = imread('trees.tif');
>> imtool(image, cmap);