MCDB/BCHM 4312 & 5312 – Quantitative Optical Imaging

Lecture 11:

Correcting uneven illumination and debugging code

Lecturer: Jian Wei Tay

Date: 17 September 2021

University of Colorado **Boulder**

Learning objectives

- Statistical functions
- Generating a normalized image intensities
- Understand the difference between array and matrix operations
- Array operators in MATLAB

Array and matrix operators in MATLAB

Operation	Array operator	Matrix operator
Multiplication	`.*`	`*`
Division	`./`	`/`
Power	`.^`	`^`
Addition	`+`	`+`
Subtraction	`-`	`-`

Note: The addition and subtraction operators are the same for array and matrix operations. `.+` and `.-` do not exist.

Operations between a matrix and a scalar

- Do you need an array operator between a matrix and a scalar?
  - `A * 2`
  - `A / 5`
  - `1 / A`

Questions?

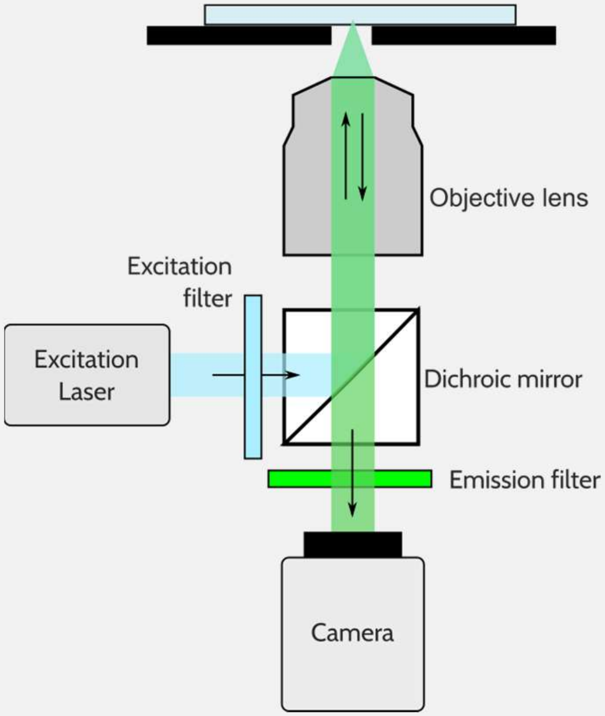
Application of array operations: Intensity corrections

- How is fluorescence generated?

Application of array operations: Intensity corrections

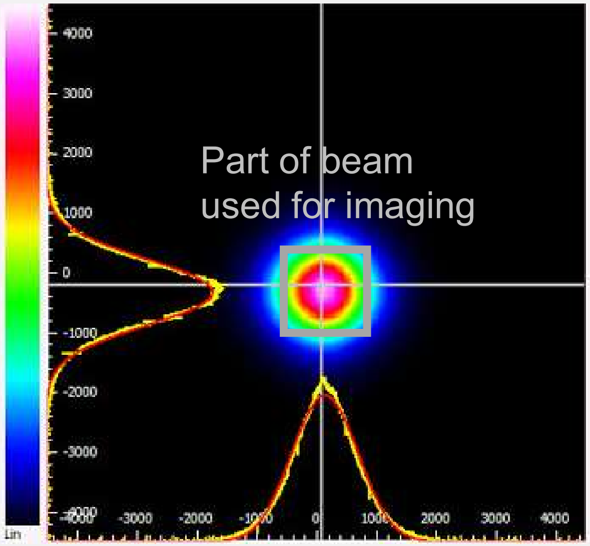
- Fluorescence is generated when a fluorophore absorbs a photon becoming excited. The excited fluorophore eventually decays to the ground state, emitting fluorescence
- See Lecture 2

Typical microscope setup



Application of array operations: Intensity corrections

- The excitation light typically has a spatially-dependent intensity pattern due to lens focusing (a problem for low magnification objectives)



Part of beam used for imaging

Typical illumination profile from objective lens

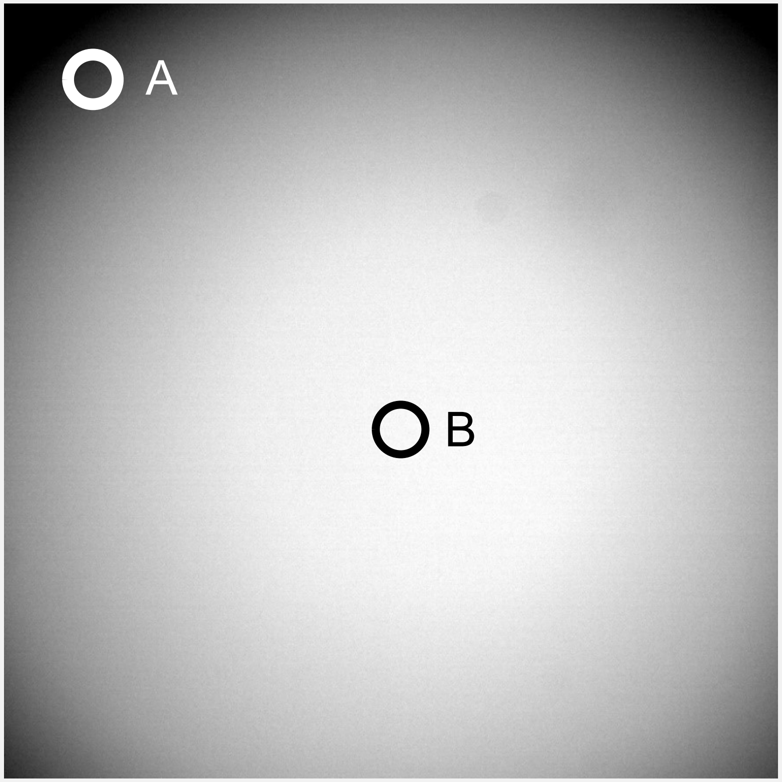
Part of the beam is blocked internally to remedy this

Measuring the illumination pattern

Fluorescent slides

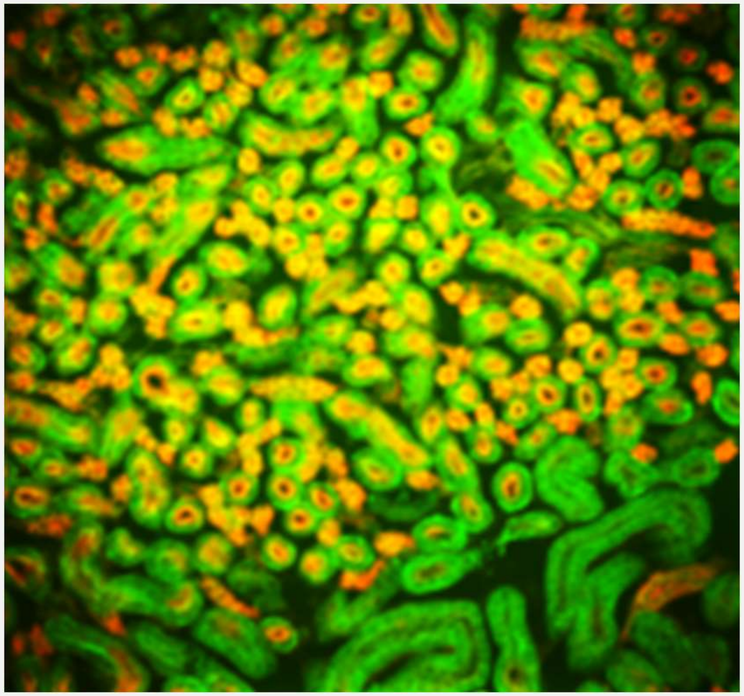
Calibration image captured on a widefield microscope, 10x objective

What is the effect of uneven illumination?



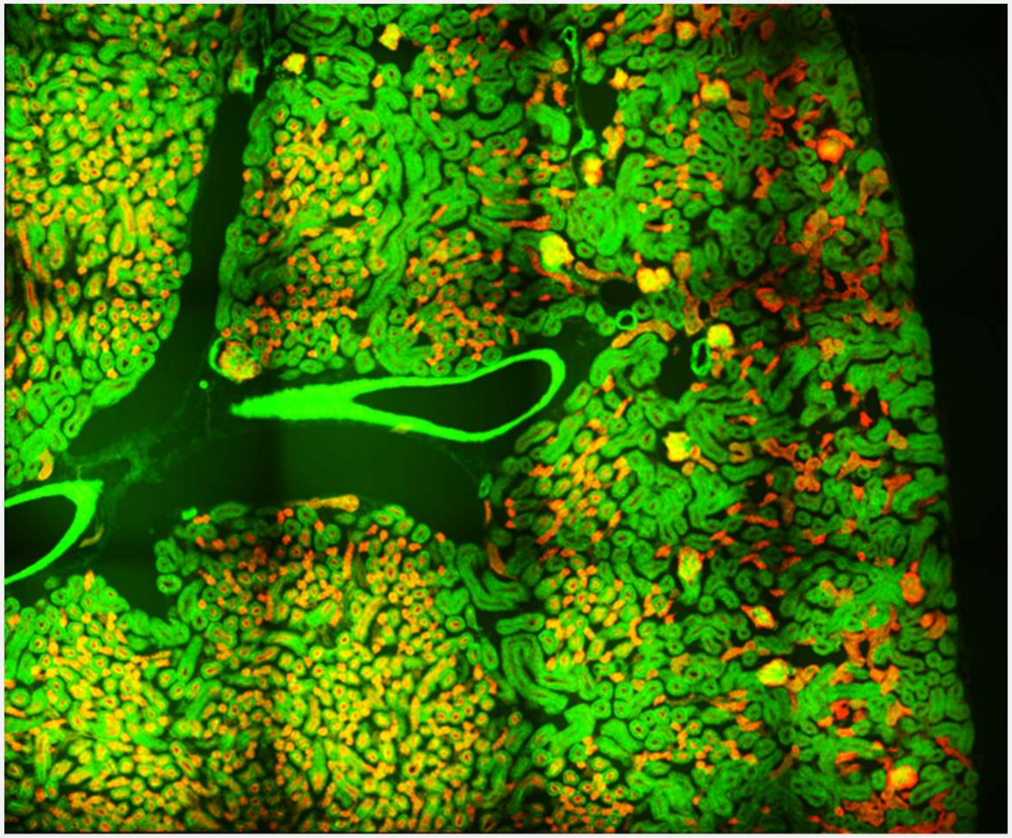
Assume there are identical fluorescent beads at points A and B. Which bead will appear brighter?

Uneven illumination causes “vignetting” or shading



http://nic.ucsf.edu/blog/2014/01/shading-correction-of-fluorescence-images/

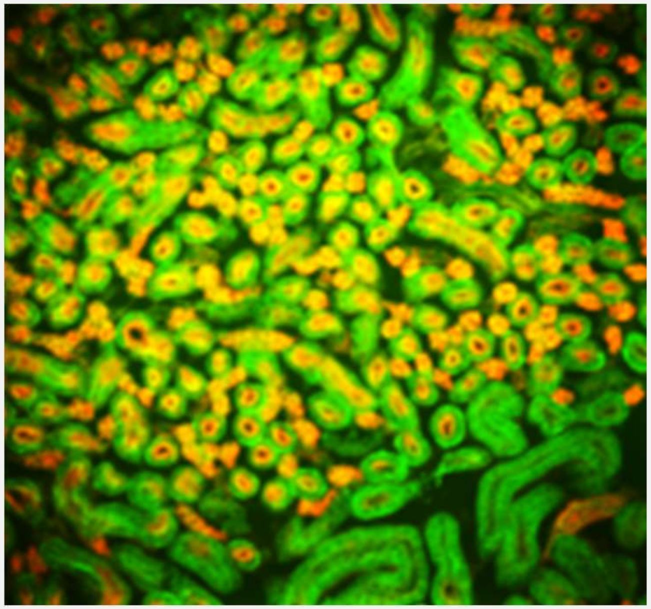
Uneven illumination causes “vignetting” or shading



Tiled image consisting of 4 x 3 individual images to illustrate shading

Correcting for uneven illumination

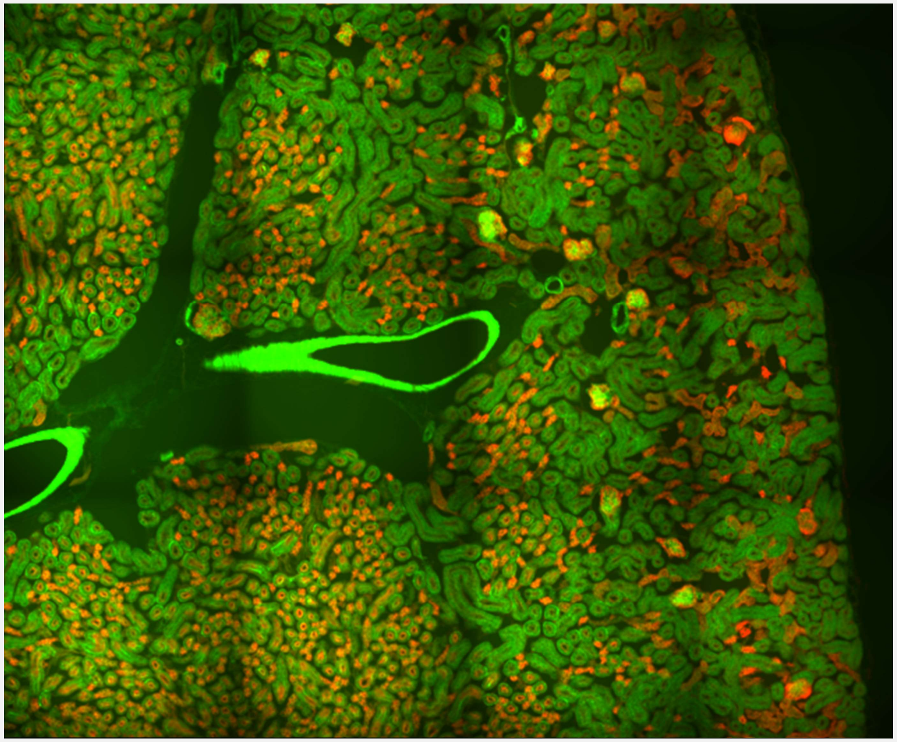
- Take an intensity calibration image (right)



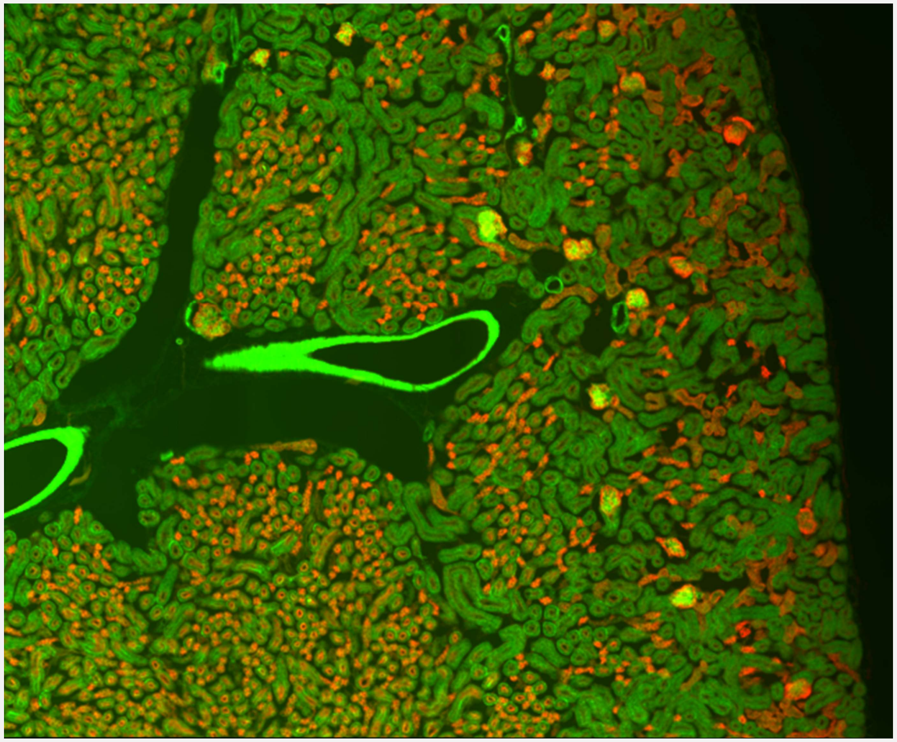
Correcting for uneven illumination

- Divide the cellular image by the calibration image

- Why divide?



Uneven illumination



Corrected

http://nic.ucsf.edu/blog/2014/01/shading-correction-of-fluorescence-images/

Example in Problem Set 4

Questions?

Debugging your code

- Mistakes are very common when programming
- Let’s get familiar with tools in the MATLAB Editor to recognize and fix errors

Types of errors

- Syntax errors
- Runtime errors
- Logic errors

Syntax errors

- Incomplete commands, e.g. missing brackets, parentheses
- Will be detected by MATLAB's built-in Code Analyzer **before** it runs the script

Examples

`A = [1 2 3`	Missing closing] bracket
`B = min(A`	Missing closing) parentheses
`B = min(A,)`	Missing argument? Or additional comma

Read the error messages

```
>> B = min(A
 B = min(A
         ↑
Invalid expression. When calling a function or
indexing a variable, use parentheses. Otherwise,
check for mismatched delimiters.
```

Note: If you don’t know what the error message means, feel free to email me

Runtime errors

- Errors that are **NOT detected** by MATLAB **until it runs the code**
- Causes program to terminate abnormally (i.e., MATLAB returns an error message)

Examples of runtime errors

```
A = [1 2 3 4];
A(5)
```

Indexing a non-existent element

```
A = [1 2 3 4];
A(1) = 1:3
```

Assignment size mismatch

```
A = 10
B = 20
C = a + b
C = mni(B)
```

Misspelled/Capitalized variables

Misspelled functions

Common runtime errors

- Capitalization matters in MATLAB
- Examples: Variable and function names, Filenames

Logic errors

- Errors that are **not detected** by MATLAB before running, and **do not cause the program to terminate abnormally**
- Results in incorrect operation (e.g. undesired/unintended outputs of behavior)
- These are the hardest to find

Examples of logic errors

```
A = [1 2 3; 4 5 6]
minRowsA = min(A, 2)
```

Incorrect argument
Check documentation

```
average = 1 + 2 + 3/3
```

Error in operator precedence

```
%Compute sine of 45 degrees
sin(45)
```

Incorrect units
Check documentation

Other mistakes to look out for

- Using the wrong type of operator (e.g. matrix instead of array)
- Entering equations incorrectly
- To minimize these, **test, test, test** your code
  - Use the "comment" function of the editor to comment blocks of code to test
  - If you can't find the error, talk to your classmates, reach out to us

Warnings

- Highlighted by the Code Analyzer in the editor
- May or may not cause errors
- Examples:
  - Unused variables
  - Not terminating lines with semicolons
  - Growing arrays in loops (we'll see this later in the course)

Practice

- Open the Editor and type the following commands in:

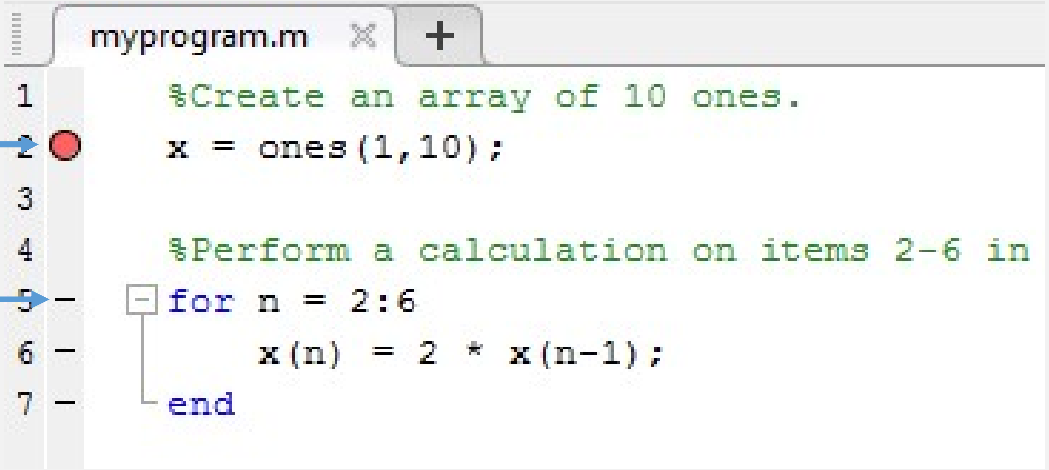
```
x = ones(1, 10);

for n = 2:6
        x(n) = 2 * x(n - 1);
end
```

Note: We haven’t covered for loops yet, but we will later in the course. For now, this code lets us test the debugging functions in MATLAB.

Debugging code

- Access the debugger by setting a breakpoint



```
%Create an array of 10 ones.
x = ones(1,10);

%Perform a calculation on items 2-6 in
for n = 2:6
    x(n) = 2 * x(n-1);
end
```

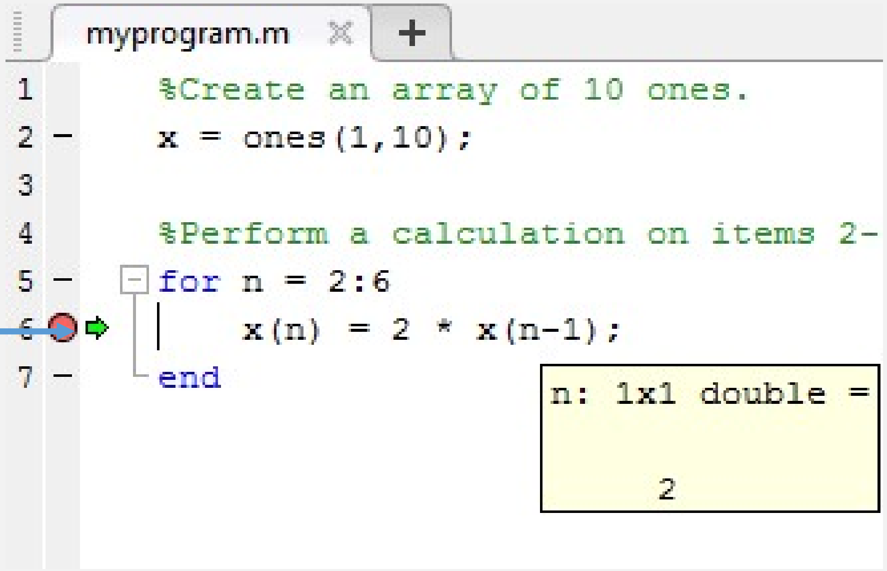
Red circle indicates breakpoint

Click on a line with code to set a breakpoint

Debugging code

- Run the code – the script will execute until it reaches a breakpoint

Green arrow indicates line where code has been paused



```
%Create an array of 10 ones.
x = ones(1,10);

%Perform a calculation on items 2-
for n = 2:6
    x(n) = 2 * x(n-1);
end
```

```
n: 1x1 double =

     2
```

Debugging mode

- The MATLAB prompt changes to `K>>`
- The status bar will read “Paused in debugger”
- You can inspect and change variables in this mode
  - I recommend turning on “Enable data tips in edit mode” under Preferences > Editor/Debugger > Display

Leaving the debugging session

- Click on the Quit Debugging button or click Continue and let the code run as usual