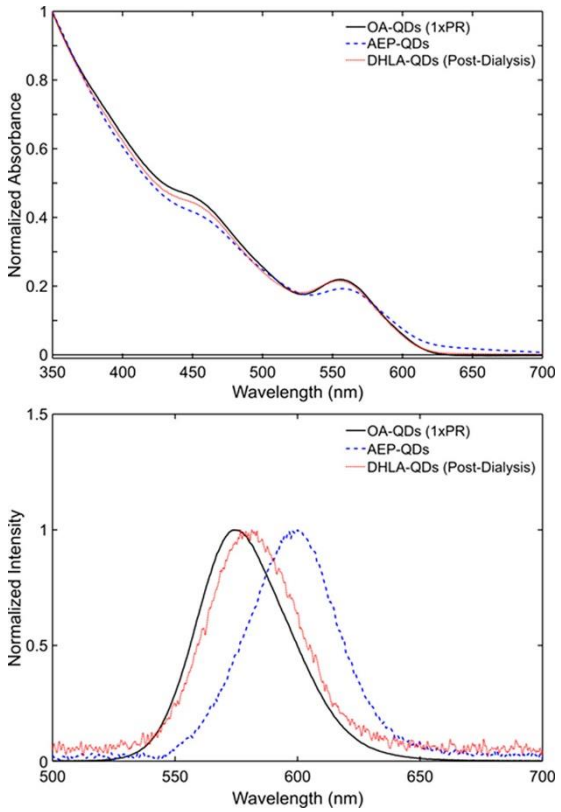
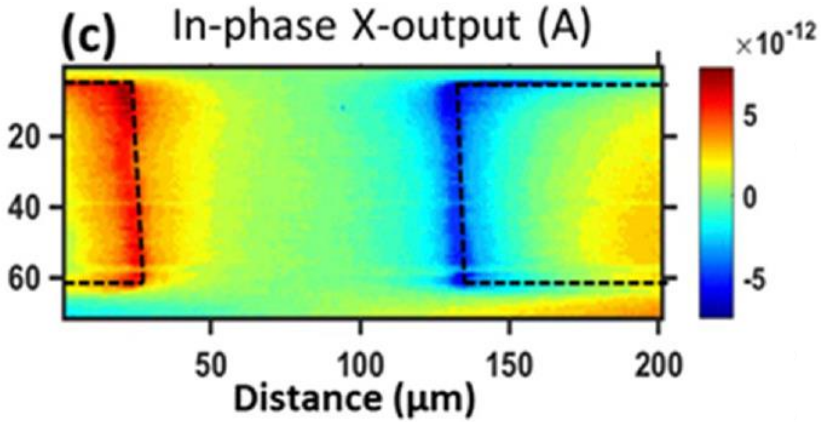
ChemCamp

Summer Graduate Enrichment Program

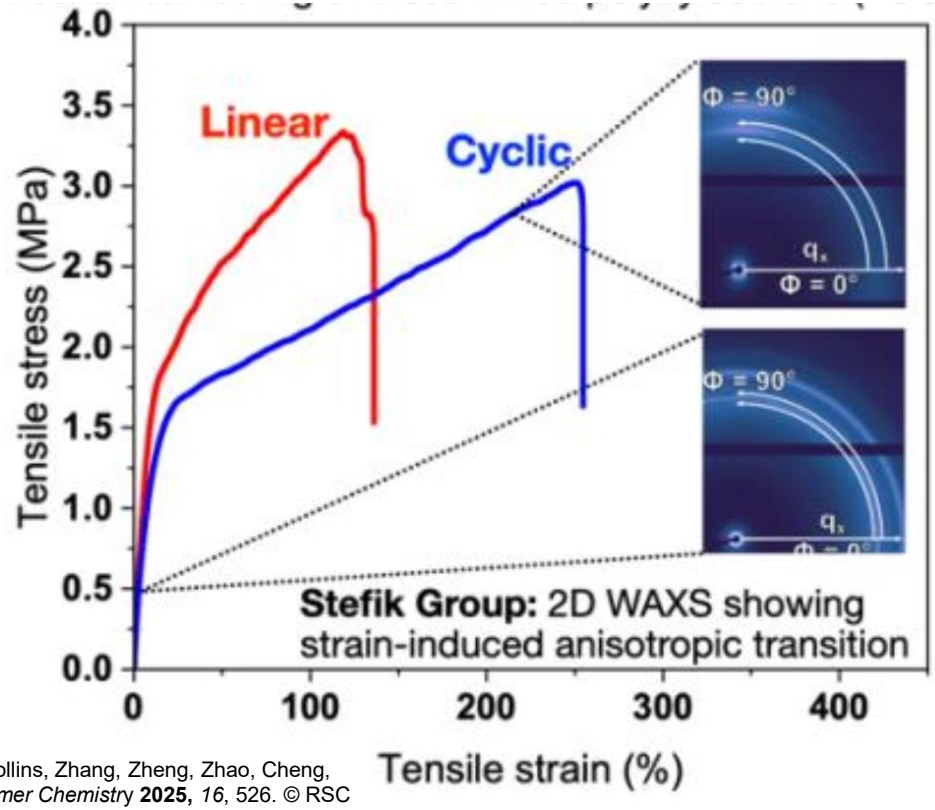
MATLAB Tutorial for Data Analysis & Presentation in Chemistry



Dunlap, Jayaweera, Pellechia & Greytak
J. Phys. Chem. C, **2022**, *126*, 17635–17646. © ACS



Ahmed, Kelley, Chandrashekhar & Greytak
J. Phys. Chem. C **2021**, *125*, 17796–17805. © ACS



Bension, Wijesekera, Collins, Zhang, Zheng, Zhao, Cheng, Stefik, Ge & Tang. *Polymer Chemistry* **2025,** *16*, 526. © RSC

Overview

What is Matlab?

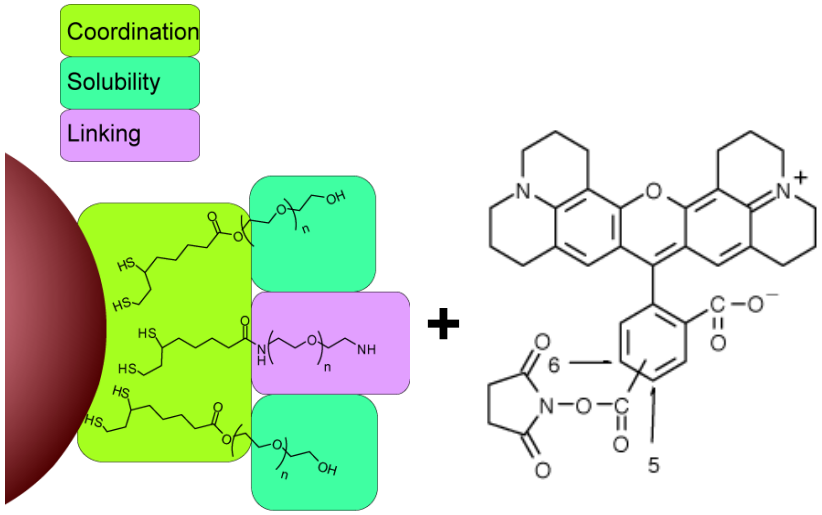
- *MATLAB* is commercial software developed by Mathworks
- Originally developed for matrix math calculations (the “matrix laboratory”) it has evolved into a complete set of tools for working with and plotting data, image analysis, and instrument control
- It is highly scriptable, making it possible to treat many sets of data in exactly the same way
- The Matlab commands make up a computer language with a very large number of built-in functions to do the sorts of things we frequently wish to do for interpreting quantitative data
- These days, Python and its various extensions (some replicating Matlab features) are used by many scientists and engineers: but Matlab provides a consistent environment, is extensively documented, and officially supported

This tutorial is meant as an introduction to Matlab for chemists. It is meant to teach some basic techniques in Matlab, but the examples are built around common problems in physical chemistry and spectroscopy. We hope it will help you think effectively about how to organize, display, and compare your results regardless of what tools you use.

See also:

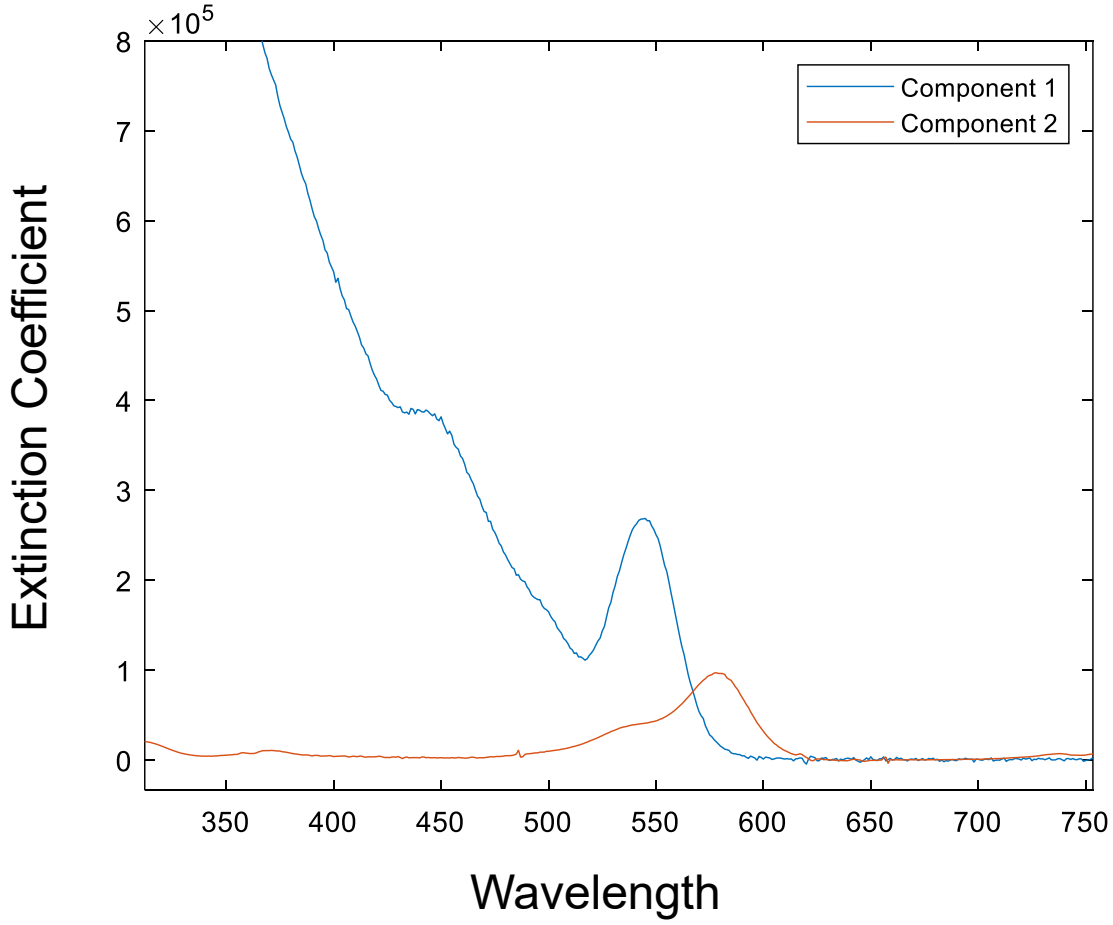
- Extensive help within Matlab: type “`doc`” or “`doc <command name>`” to launch
- Mathworks workshops on image processing, other topics
- Matlab books
- Octave: open source project implementing Matlab language

Some things you can do: Plotting and analyzing spectra of mixtures



The *extinction coefficient* (absorbance per molar per cm) for each separate component is known at a particular wavelength, so that you can scale a measured absorption spectrum to give the extinction spectrum of each component.

The spectrum of a sample with mixture, or bonded conjugate of QD and dye should be similar to a linear combination of these component waveforms, and the coefficients should give the total concentrations of each.

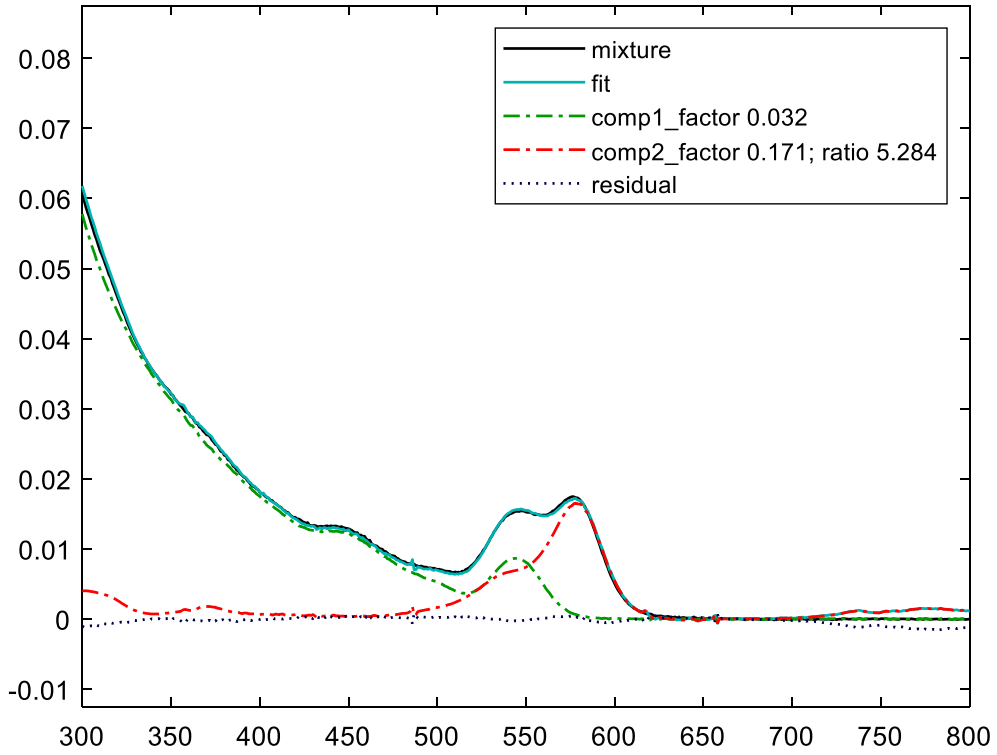


Example used in support of Liu, Howarth, Greytak, et al. *JACS* **2008** v. *130* p. 1274.

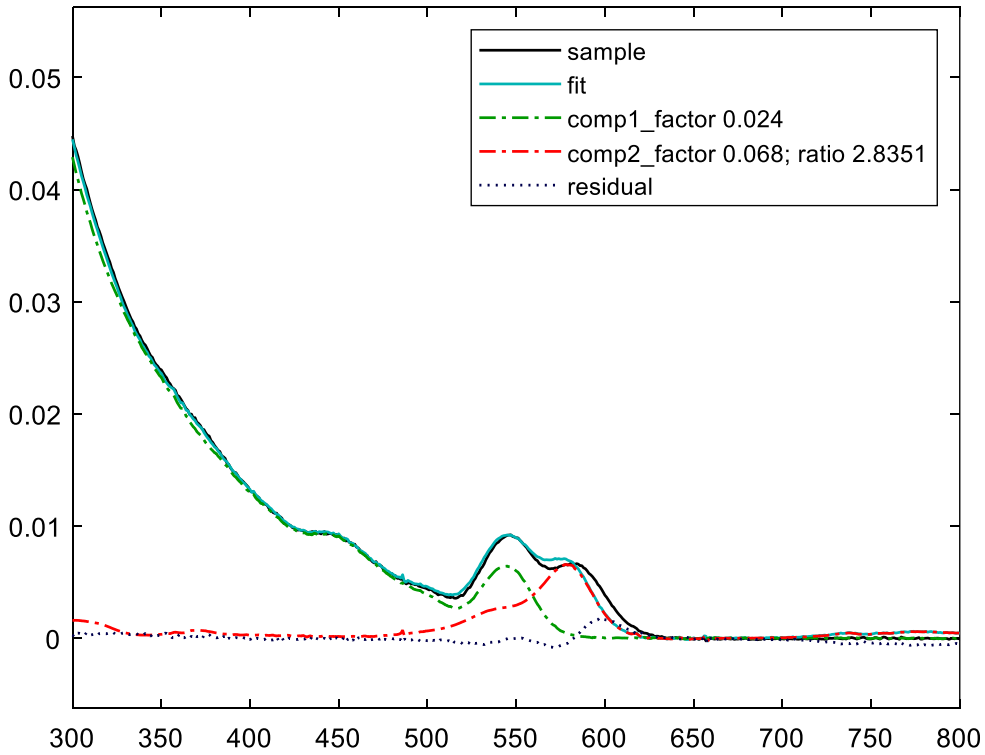
Absorbance of conjugate absorption spectrum to determine dye:QD ratio

- Purified dye ratio ~2.8
- Not a perfect fit: shows shift not seen for mixture

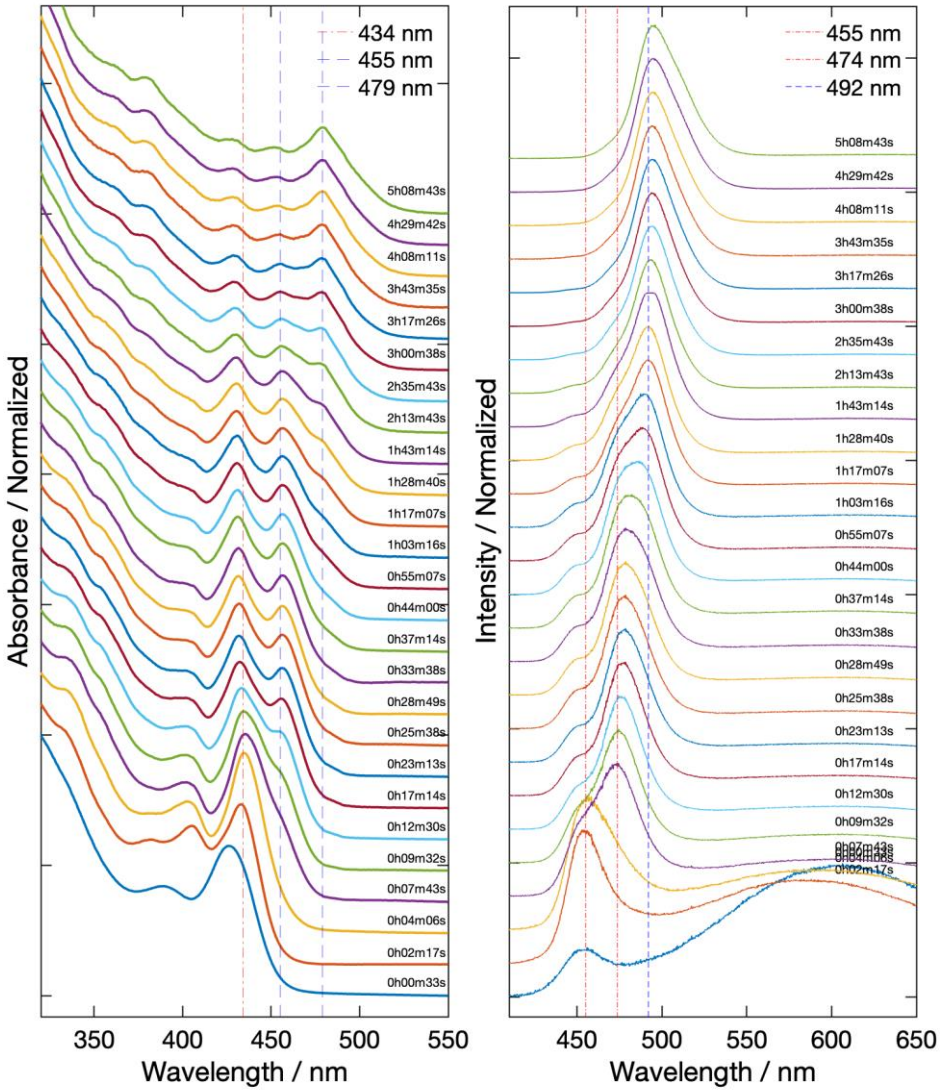
Fit mixture of QDs and free, unreactive dye (control, 4-8C) with reference spectra

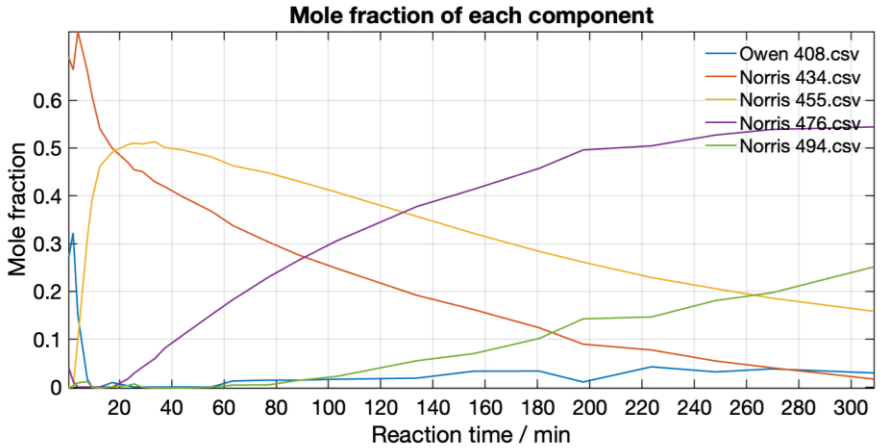


Fit purified QD-dye conjugate (4-8A) with reference spectra



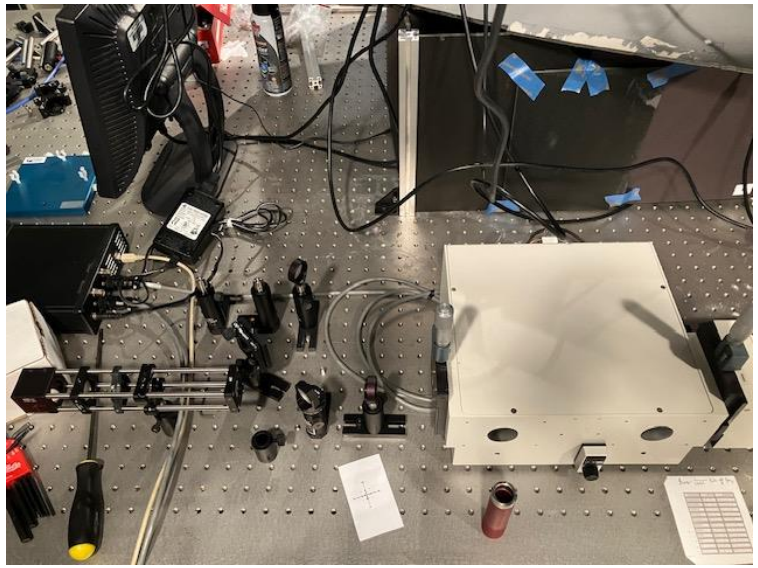
More Advanced: Fluorescence spectroscopy of “magic size” nanocrystal molecules

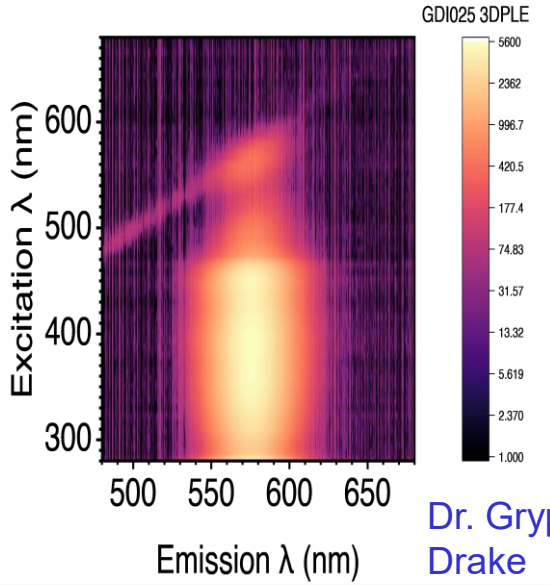




Moinul Islam

Also: Fluorescence spectroscopy & “excitation-emission maps” to identify components of quantum dot mixtures

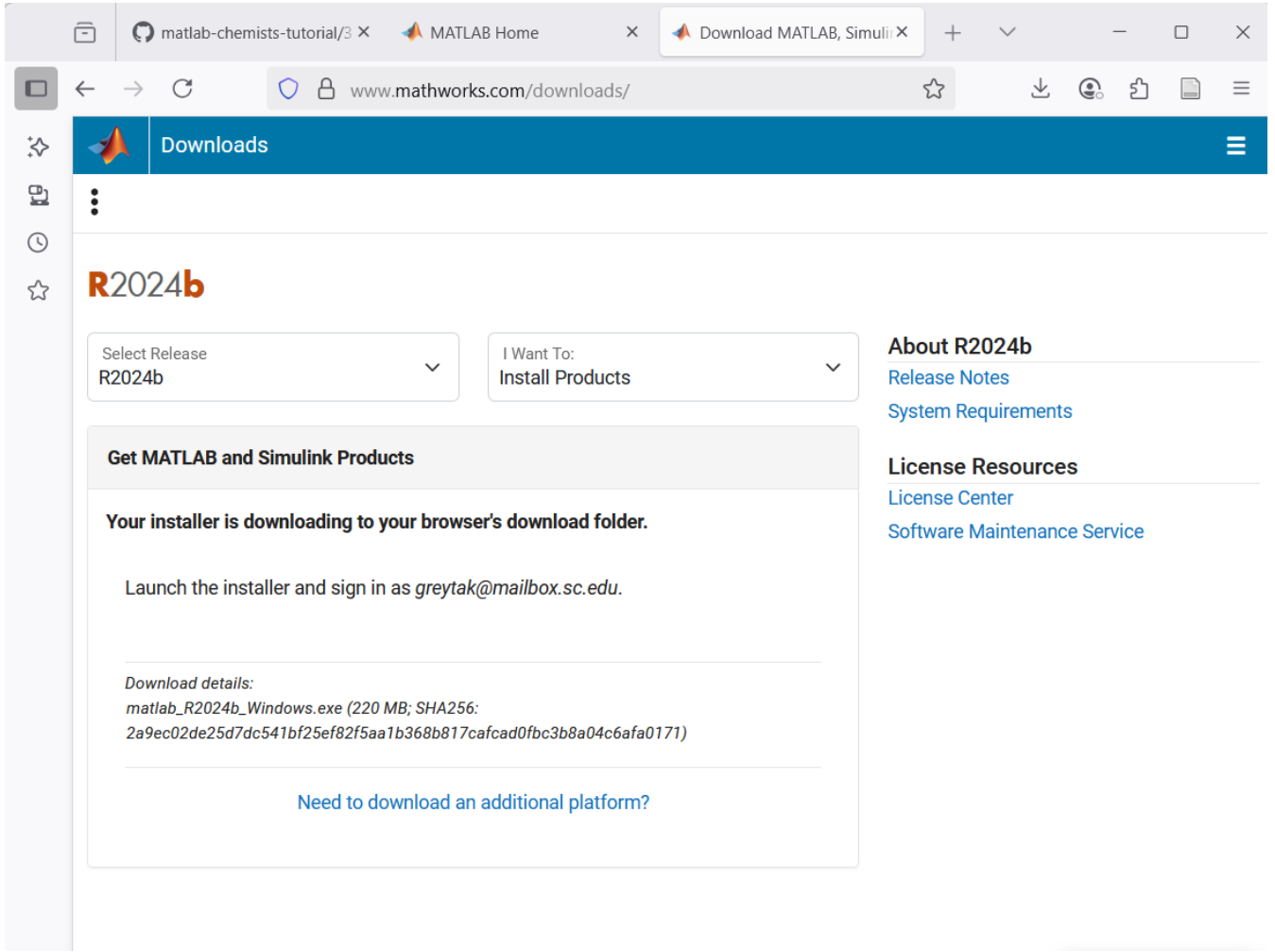




Dr. Gryphon Drake

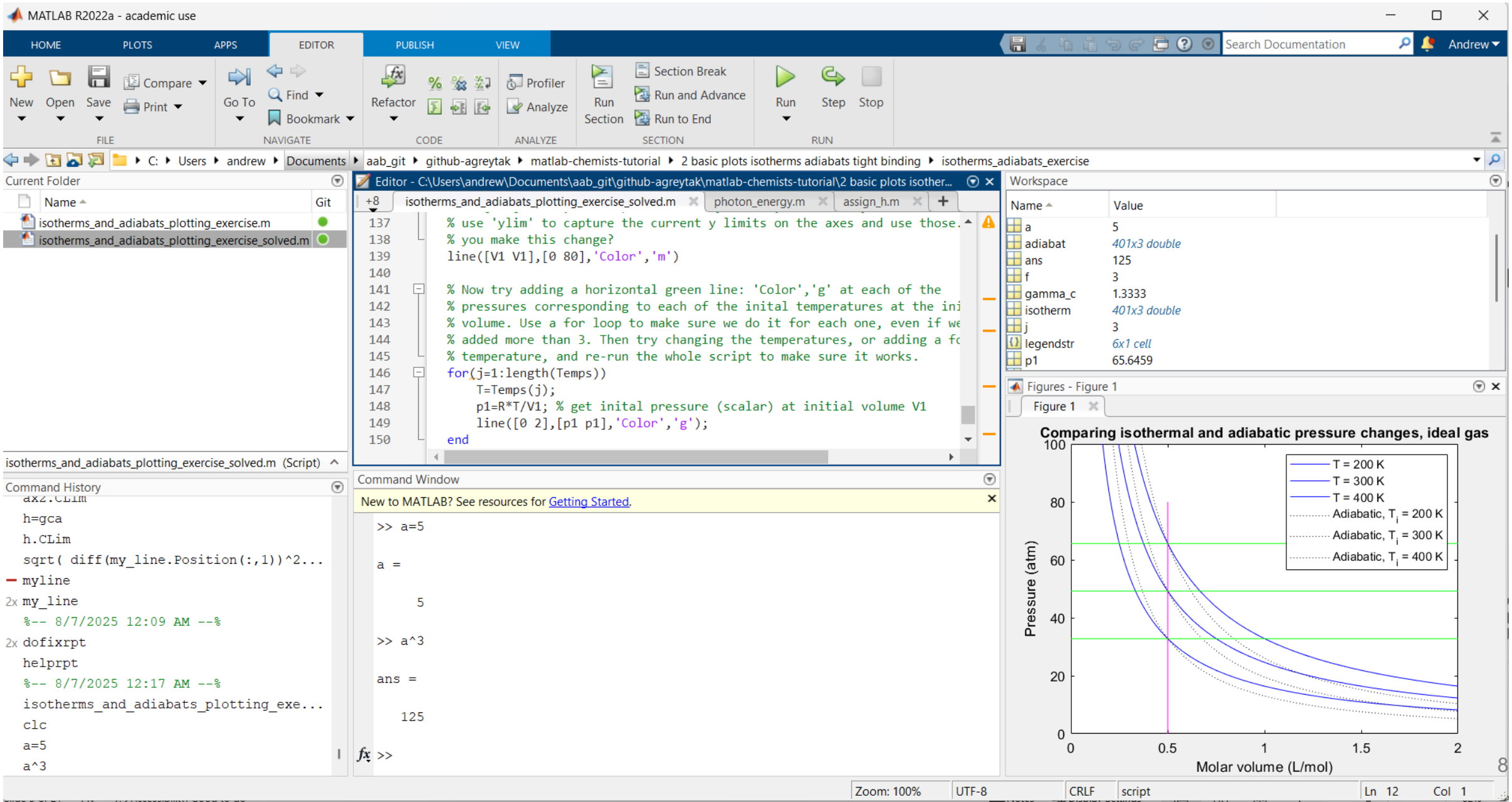
Running / Installing Matlab

- It is installed in these labs
- See my.sc.edu/software, under “statistics software”
  - “Purchase” for $0
- Versions and Toolboxes:
  - You are entitled to download any previous version of Matlab; Usually 2 per year; features and behavior change over time
  - You can install multiple versions at once
  - Some features are contained in “Toolboxes” that can be optionally installed
    - You want “Image Processing Toolbox” and “Statistics” for sure
    - Computer labs have most of them installed



Part 1: Basics: commands, variables
Part 2: Plotting Data, Loading Data
Part 3: Scripts and Functions
Part 6: Image Analysis Basics

The Matlab environment



Programming in a nutshell

Programs follow a sequence of commands in order

Variables store and retrieve information (numbers, letters, arrays, etc)

x=2, y=7, name='Bob', numbers=[1 2 3 4]

Conditional programming steers using logic

If-then-else: *If some condition is met then do this thing, otherwise do this other thing*

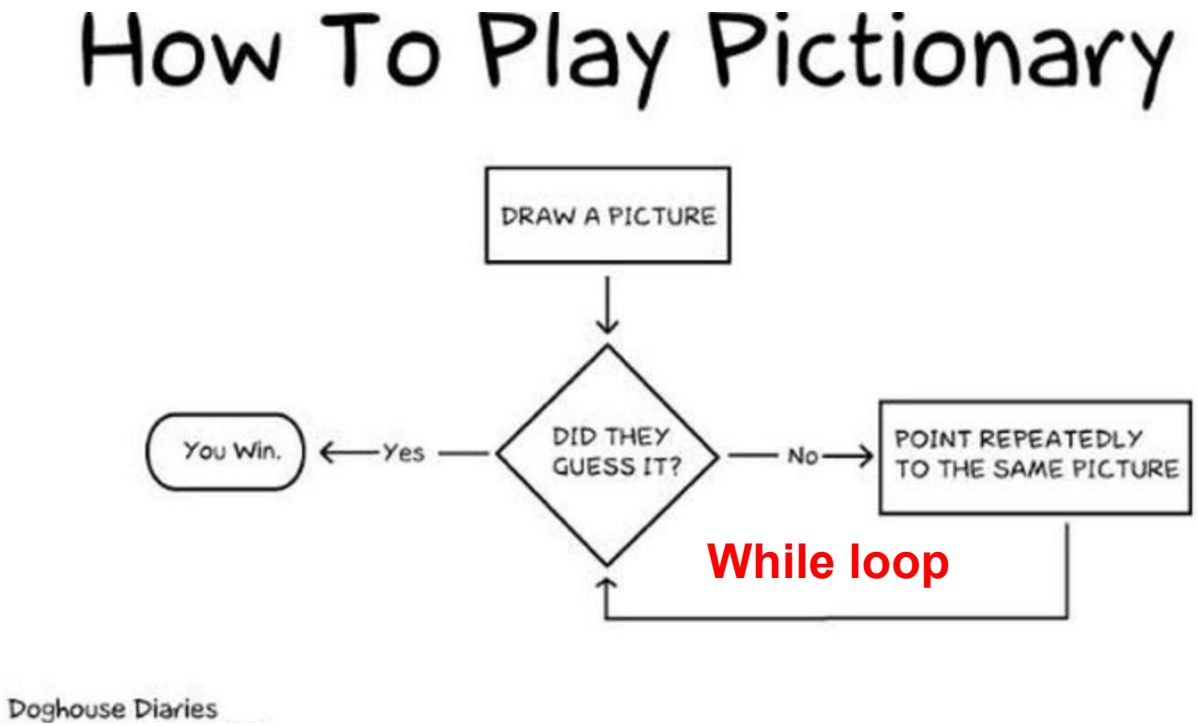
Switch-case: *For condition A do this thing, for condition B do this other thing,…*

Loops repeat sections of code

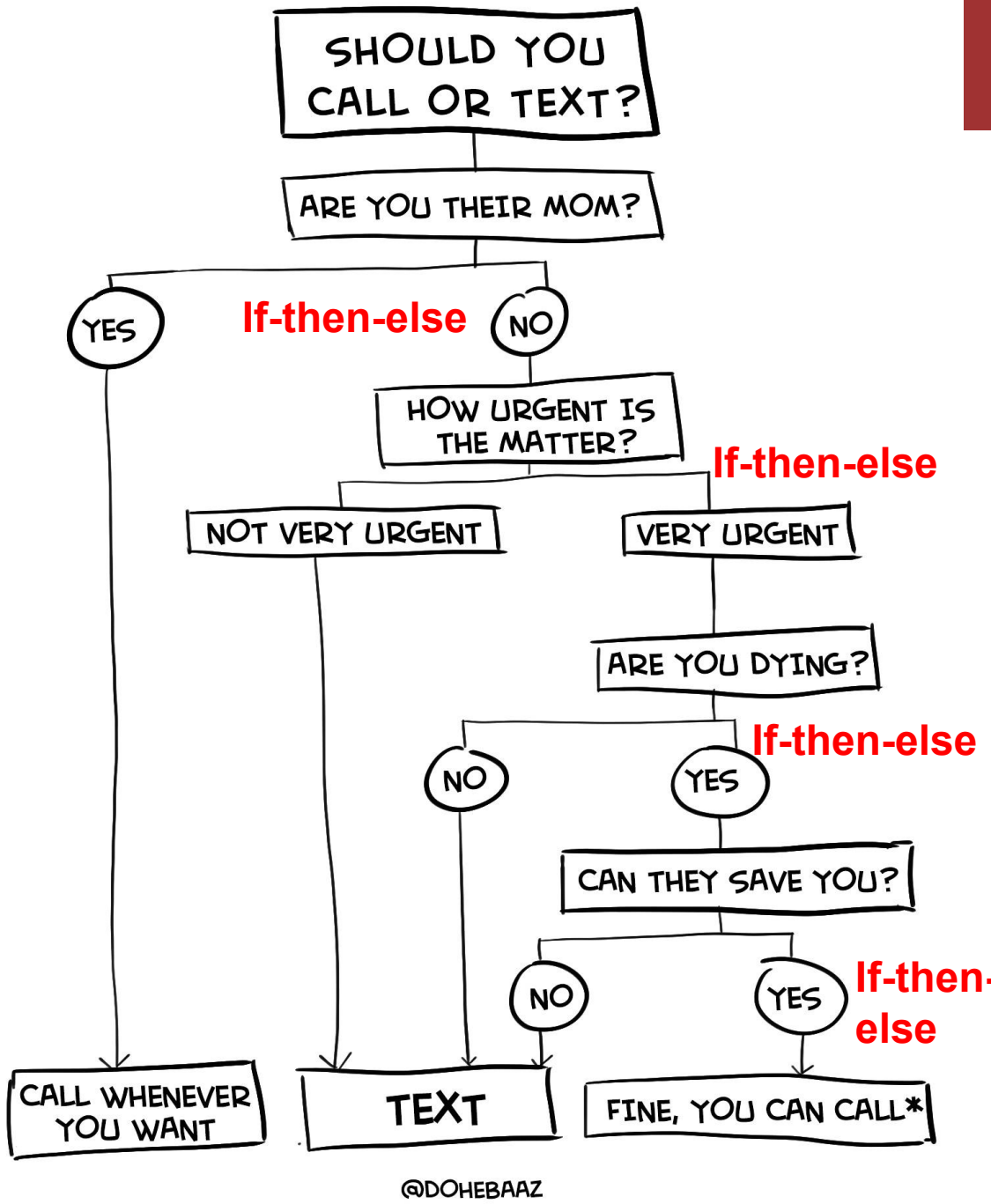
For Loops repeat a specified number of times: *Do this thing 5 times*

While Loops repeat until a condition is met: *Do this thing until the total is 100*

Visual Examples



While loop



If-then-else

If-then-else

If-then-else

If-then-else

Best Practices – get organized, don't overlook this part!

Plan for data management

x1=1, x2=2, x3=3, x4=4 is clunky and inflexible

x=1:4 is easily modified

Use data structures sample1.x, sample1.y, sample1.time, …
Here, all aspects of sample1 move together

Separate data from analysis

The original raw data will not change, never modify it or save over it.

Your analysis and calculations will change overtime so save all those variables elsewhere

Best Practices

“99% of programming issues occur between the chair and the keyboard”
Keep it human-friendly to avoid errors

*Specifically a*void nesting loops/conditions more than 3 layers deep

Functions are standalone chunks of code that are given inputs and return outputs. Functions allow complex programs to be made modularly from a collection of simpler codes

Validate your program output manually and/or on data that’s easy to interpret

Only use code that you understand

Comment your code!!

Comments are lines that aren't interpreted as commands.

In Matlab:

```
% comments everything on a line after the % sign
%% indicates the beginning of a "section" in a script, is also a comment
plot([1:10], …  three dots comments the rest of the line and continues command
        sin([1:10]*0.2/pi())
```

You can "comment out" code that isn't working for de-bugging, or that isn't needed right now but you don't want to throw away:

```
% plot([0:360], sin([0:360])
```

Comments help you or others understand what you were doing … comment as though a Martian needs to understand your code …
… in 6 months you will be that Martian!

Variables and workspace (tutorial 1): the command window

You can enter basic math operations directly in the command window. It knows order of operations:

```
>> 5+5
ans =
    10
>> 5+5*5
ans =
    30
>> a=5
a =
     5
>> a^2
ans =
    25
>> clc
```

Assign a variable

Clear command window

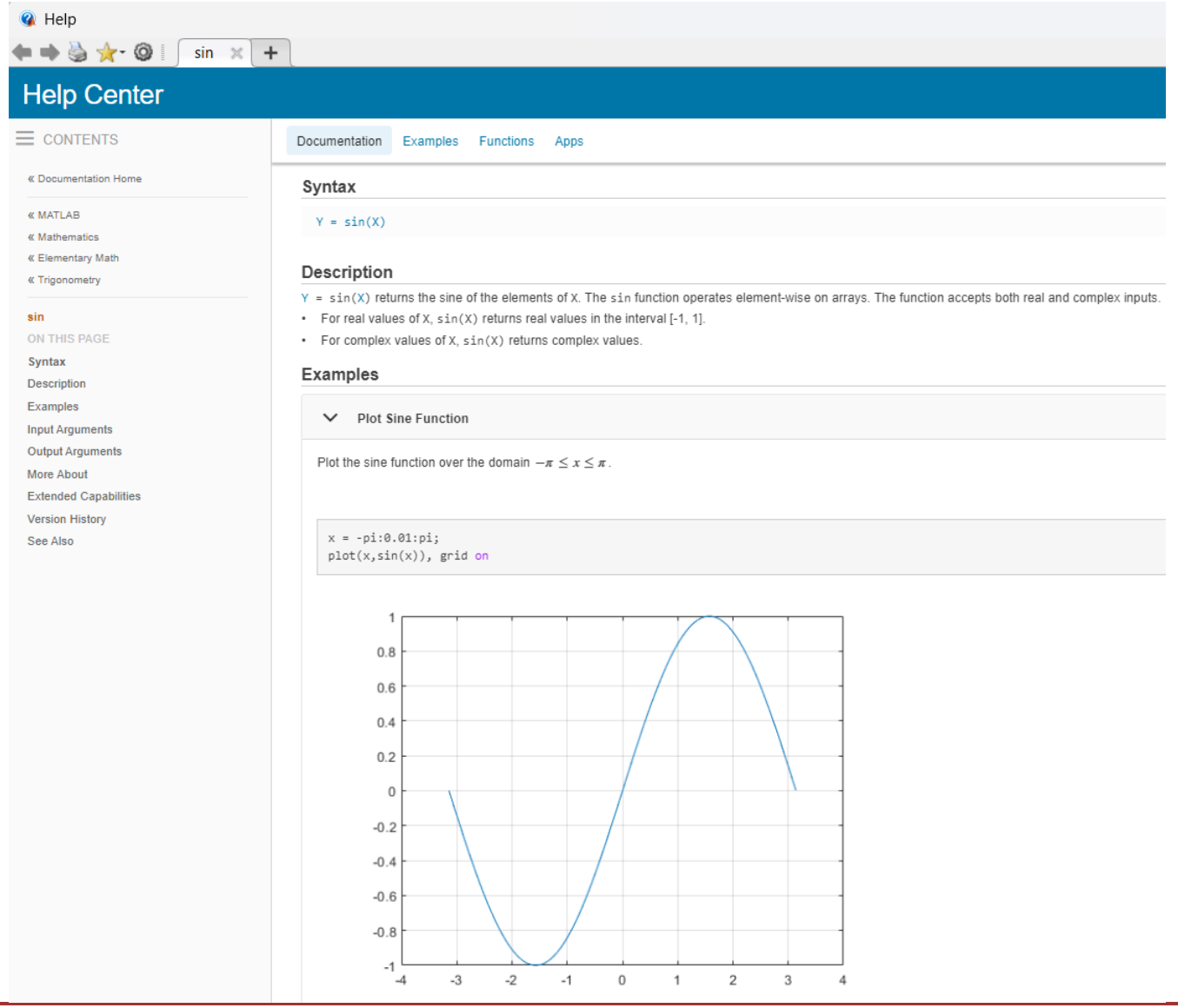
There are many Matlab commands: Learn to use the built-in help!

```
>> help sin
 sin    Sine of argument in radians.
    sin(X) is the sine of the elements of X.

    See also asin, sind, sinpi.

    Documentation for sin
>> doc sin

>> sqrt(2)
>> pi()
>> sin(pi()/2)
```



Arrays: vectors, matrices, matrix math

Matlab is the “matrix laboratory”. Most commands and functions are designed to work with arrays of numbers, including doing matrix math correctly and very rapidly:

```
% row vector:
[2 5]
% two column vectors: the ; separates rows in a matrix, ' transposes
[2 ; 0]
[2 0]’

% let's take the dot product (inner product)
[2 5]*[2 0] % this will generate an error
[2 5]*[2 ; 0] % this will work and give a scalar (single value) result
[2 ; 0]*[2 5] % this gives the 'outer product': order matters for matrices
[2 5]*6  % multiplication by a scalar always works

% you can do elementwise math by putting a . before most operators:
[2 5].*[2 0] % multiplies each element together for two matrices of equal size
```

Iteration operator ':' lets you specify a list

```
[0 1 2 3 4 5 6 7 8] % enter a list manually
[0:8] % values spaced by 1 by default
[0:1:8] % does the same as above
[0:2:8] % increment by 2 from the initial to the final value
[0:-2:-10] % increment can be negative
[0:8]' % if you wanted a column vector
sum(1:100) % add up numbers 1 thru 100

% linspace command automatically generates a series of evenly spaced values
linspace(0,10,11) % generate 11 values evenly spaced between 0 and 10
sin(linspace(0,8,9)*pi()/4)

% Make a plot: plot(x_data,y_data)
plot(linspace(0,8,101),sin(linspace(0,8,101)*pi()/4))
```

Variables

Variable **TYPE**: double (default), logical (0 or 1), uint8, uint16, char

```
a=5;
a=sqrt(-2);  % it can deal with complex numbers!
```

Variable **SIZE**: is it just one value, or an array? (2D array = matrix, 1D array = vector).

Matlab is always (row,column) when addressing arrays.

```
b=[2; 0]
size(b)
```

You can select (address) elements of a simple array using parentheses:

VERY IMPORTANT: in Matlab, unlike most computer languages, the first index is 1, not 0. This is because Matlab is designed for matrix math, and the first element is on the 1st row or column of the matrix!

```
b(1)
b(2)
b(0) % gives an error
```

Variable **SCOPE**: range of places from which variable is accessible (more on this later)

Variables: working with arrays

```
% For a 2-D array, the first number is the row, second number is the
% column:
a=[2 4 6 8]';
b=[3 5 7 9]';
c=[a b] % this concatentates (connects) two column vectors to get a matrix
c(2,1) % returns 4

% You can use the : operator within an index to pull out a row or column:
c(2,:) % gets the second row
c(:,2) % gets the second column

% the 'end' operator gives the last available index in that dimension
c(2:end,1) % second thru last element of 1st column
c([2 3 4],1) % get the same result manually, using an array of row indices
```

See also: Linear addressing, Logical addressing

More in: matlab-chemists-tutorial\1 general intro to variables and workspace

Part 1: Basics: commands, variables
Part 2: Plotting Data, Loading Data
Part 3: Scripts and Functions
Part 6: Image Analysis Basics

Loading and plotting data (tutorial 2)

```
% let's make some fake data
x1=linspace(0,5,21);
y1=x1.^2;

plot(x1,y1)

y2=x1.^3 / 5;
plot(x1,y1,'r',x1,y2,'b') % overlay 2 curves … ‘r’ and ‘b’ are “Linespec” values determining colors (doc plot for more)
plot(x1,[y1 y2]) % another way, for 2 curves with same x data (what’s wrong with this?)

y2=x1.^3 / 5;
x3=linspace(-5,5,21);
y3=x3.^3 / 5;
plot(x1,y1,'r',x3,y3,'b')
```

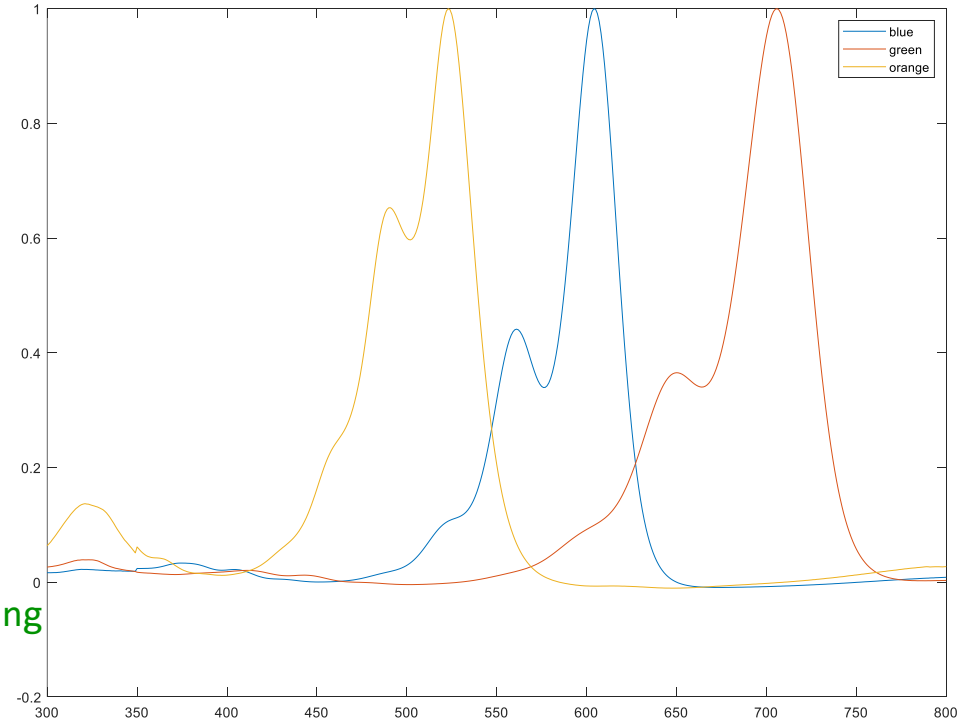
Importing data (tutorial 2)

We will import and plot some UV-visible absorbance curves from a .csv (comma separated variable) file exported from the Agilent Cary 5000 UV-visible spectrometer in Chem 542L, using the “Import Data” wizard.

→ Choose “column vectors”, name the columns appropriately

```
plot(wavelength,[blue/max(blue)  green/max(green)  orange/max(orange)])
```

Cyanine iodide dye spectra (normalized), Chem 542L



More on plots in

```
matlab-chemists-tutorial\2 basic plots isotherms adiabats tight binding
```

See example data in:

```
matlab-chemists-tutorial\2 basic plots isotherms adiabats tight binding\absorption_data_polyenes
```

Part 1: Basics: commands, variables
Part 2: Plotting Data, Loading Data
Part 3: Scripts and Functions
Part 6: Image Analysis Basics

Scripts and Functions (tutorial 3)

A **script** runs commands in order, in the main workspace

- Can see and change all variables in the main workspace
- Variables created in the script persist in the main workspace
- Can be broken into “cells” and run a piece at a time
- Select a portion and push “F9” to run
- Right-click any constant number value in a script window to pull up “increment value and run section” tab (Matlab 2021a or earlier, and 2023a and later)

A **function** takes **inputs** and returns **outputs**

- Operates in its **own workspace**; variables created disappear (but graphics and command outputs are visible)
- Behaves like a Matlab command
- Functions can be run by saving an appropriately named and formatted .m file somewhere in the Matlab **search path** (usually includes current directory**)**

An example function define in a .m file

```
function out=ev_demo(nm)
% return electron volts for input in nm


% the first comment line above will be printed when someone types "help ev"


h=4.135668e-15; % Planck's constant in eV-s
c=299792458; % m/s


% using element-wise notation
lambda=nm*1e-9;
out=h*c./lambda;
```

Scripts and Functions (tutorial 3)

Defined in .m file

```
function out=ev_demo(nm)
…
```

Referenced with a “function handle”:

```
>> myfunc=@sin ;
>> myfunc(pi/2)
ans =
    1
```

Defined in-line with “function handle”:

```
>> myfunc=@(x) x^2 - x;
```

“Anonymous function handle” – used in same line

```
>> fplot( @(x) x^2 - x,[0 5])
```

Much more in `matlab-chemists-tutorial\3 scripts and functions`

Part 1: Basics: commands, variables
Part 2: Plotting Data, Loading Data
Part 3: Scripts and Functions
Part 6: Image Analysis Basics

Image Analysis (Tutorial 6)

A true color image (fluorescent bead sensors, John Lavigne lab)

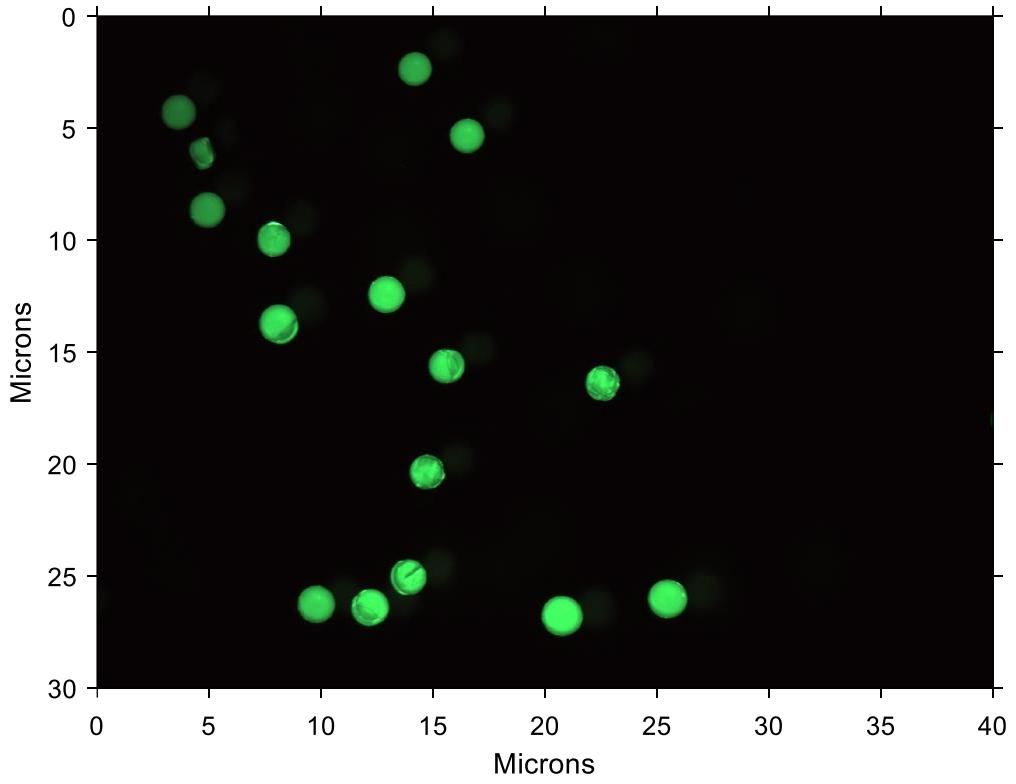
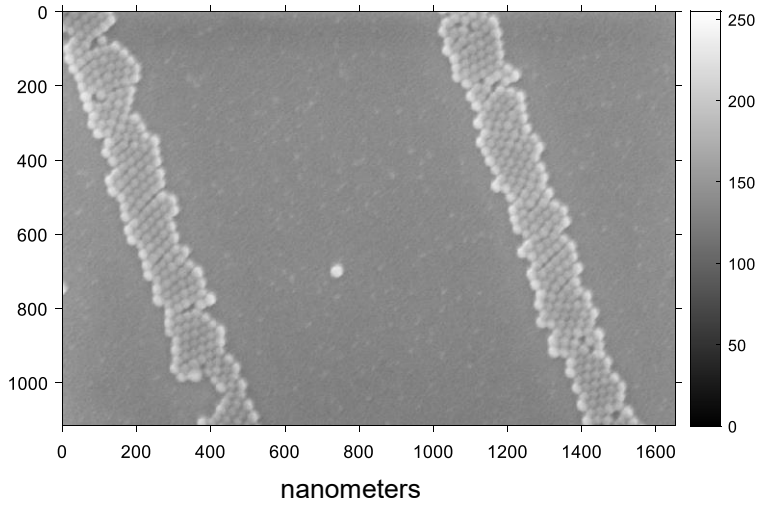
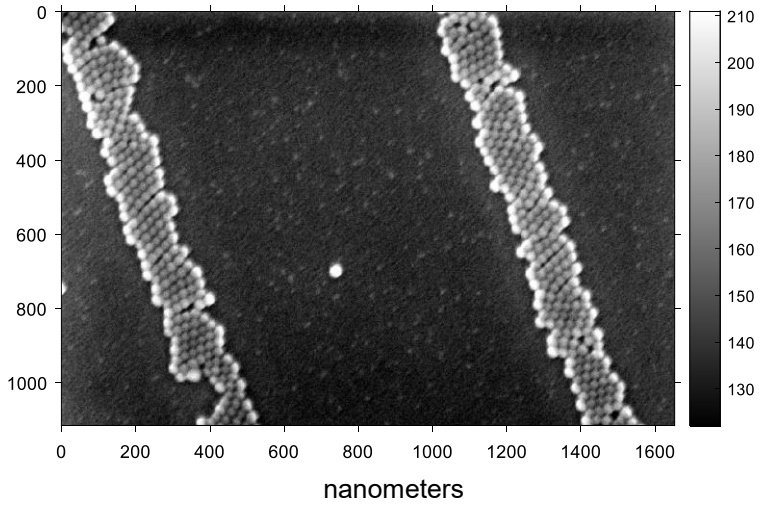
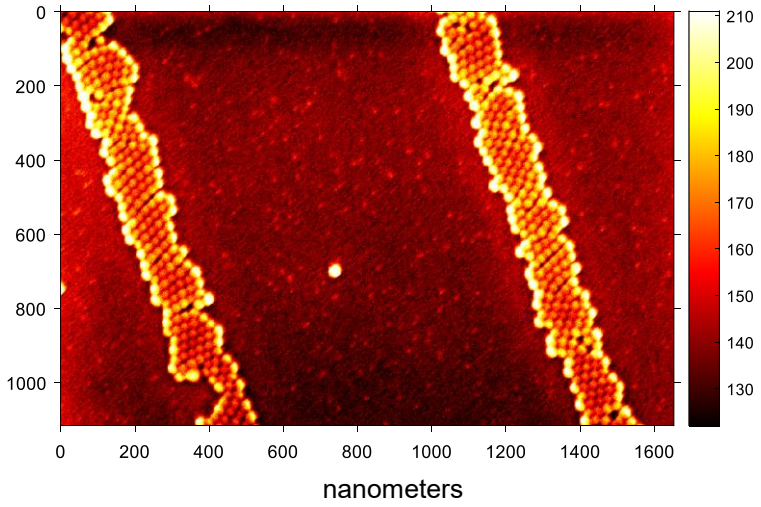


Image Analysis (Tutorial 6)

Grayscale Images: adjusting contrast and Lookup Tables (LUTs):

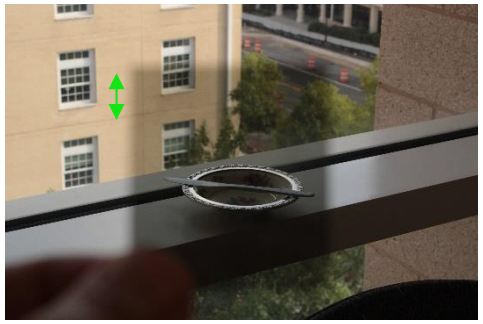






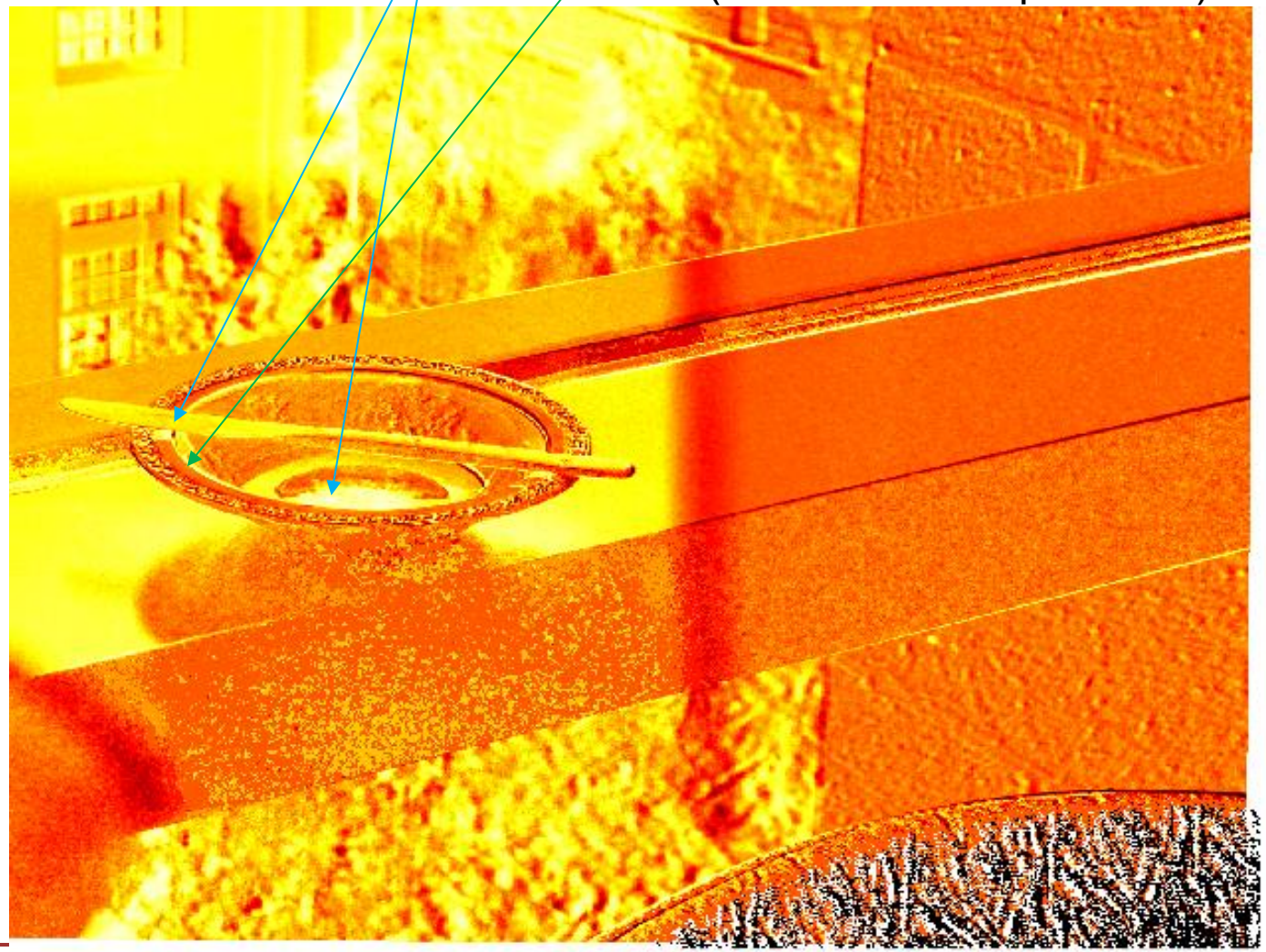
Advanced image analysis example: Image comparison (polarization demo)

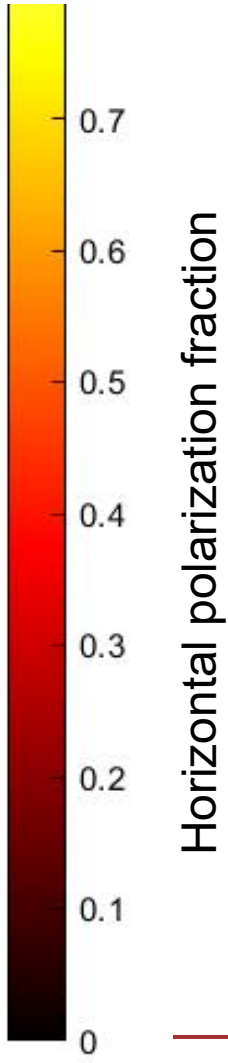
Use control points to register, overlay, and compare images



Stainless steel, ceramic glaze (dielectric surfaces: polarized)

Silver (noble metal: unpolarized)





Horizontal polarization fraction