Advice of the Day

- Why are you studying to be a scientist or engineer?
  - Money can’t buy happiness; a lot of people get into engineering because they know many engineering fields pay well, but at what opportunity cost?
    - Long hours
    - Mental/physical exhaustion
    - Time away from family/friends/loved ones
    - Is it worth it? That depends on who you are and what your passions are.

- Do you have alternative intrinsic motivators?

- Remember an earlier Advice of the Day: “If you do what you love, you’ll never work a day in your life.”

- The details of what exactly you want to do in your field may not be clear to you yet, but hopefully your path forward will become clearer as you take more narrow-focused major coursework.
Arrays vs. Matrices vs. ?

Array:
- holds “stuff:” Can hold numeric information, character data, symbolic data, etc.
- is “an orderly grouping of information”
- No special properties by virtue of its existence

Matrix:
- 2D numeric array used in linear algebra
- Used extensively in STEM fields
- Special properties!

What else is out there? This is the topic of the day!
Data Types

- Most Computer Science courses cover data types relatively early in their courses – why?
  - Most languages require you to declare a variable or a constant as a particular data type
  - As you might have noticed, MATLAB generally doesn’t care for the purposes of many operations, which is why we delayed talking about it up-front.

- The basic unit of information in computers is the bit
  - Bits can only have one of 2 values: 0 or 1 (binary!)
  - Bit is a portmanteau (blending) of the phrase “binary digit”

- One of the more commonly discussed units of information in computers is the byte
  - Typically is expressed as 8 bits/byte
  - Power of 2: allow’s values between 0 and 255 for 1 byte
Numeric Data Types

- Double-Precision Floating-Point Numbers (doubles)
  - MATLAB by default stores numeric data as doubles
  - Each value requires 8 bytes of space:
    \[ 8 \text{ bytes} \times 8 \text{ bits/byte} = 64 \text{ bits} \]

- Single-Precision Floating-Point Numbers (single)
  - Uses half the storage space of a double
    \[ \therefore \text{ that they have half the storage} \]
  - Each value requires 4 bytes of space:
    \[ 4 \text{ bytes} \times 8 \text{ bits/byte} = 32 \text{ bits} \]

- Complex Numbers (can be doubles, singles, or integers – more on integers on the next slide)
  - Requires twice the space of the base data type
  - Needs space for both the real-valued and complex-valued components
Numeric Data Types (Cont’d)

- Integer (whole numbers)
  - Often used to count, or as matrix/array indexes
  - Storage depends on if the integer is signed or unsigned

<table>
<thead>
<tr>
<th>Signed Integers</th>
<th>Unsigned Integers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>What is it?</td>
</tr>
<tr>
<td>int8</td>
<td>8-bit signed</td>
</tr>
<tr>
<td>int16</td>
<td>16-bit signed</td>
</tr>
<tr>
<td>int32</td>
<td>32-bit signed</td>
</tr>
<tr>
<td>int64</td>
<td>64-bit signed</td>
</tr>
</tbody>
</table>

Table Source: Moore, H., *MATLAB for Engineers*, 4th ed., p. 396
Examples of Numeric Data Types (1 of 3)

- **Input:**
  - A = realmax
  - B = realmin
  - C = 1E400  % defaults to inf
  - D = 1E-400  % defaults to 0
  - E = realmax('single')
  - F = realmin('single')
  - G = single(5)  % stores single

- **Output:**
  - A = 1.7977e+308
  - B = 2.2251e-308
  - C = Inf
  - D = 0
  - E = 3.4028e+38
  - F = 1.1755e-38
  - G = 5

Keep an eye on your calculations so that you don’t exceed the upper or lower storage limits for each numeric data type.
Examples of Numeric Data Types (2 of 3)

- Harmonic Series
  - Two Cases:
    - $\sum_{n=1}^{\infty} \frac{1}{n}$ (diverges)
    - $\sum_{n=1}^{\infty} \frac{1}{n^2}$ (converges)

- Input:

  ```matlab
  n_dbl = 1:1E7;
n_sgl = single(n_dbl);

  harm_dbl_a = 1./n_dbl;
  ptl_sum_dbl_a = cumsum(harm_dbl_a);
  clear harm_dbl_a

  harm_sgl_a = 1./n_sgl;
  ptl_sum_sgl_a = cumsum(harm_sgl_a);
  clear harm_sgl_a
  ```

- Input (Cont'd):

  ```matlab
  harm_dbl_b = 1./(n_dbl.^2);
  ptl_sum_dbl_b = cumsum(harm_dbl_b);
  clear harm_dbl_b

  harm_sgl_b = 1./(n_sgl.^2);
  ptl_sum_sgl_b = cumsum(harm_sgl_b);
  clear harm_sgl_b

  plot(n_dbl, ptl_sum_dbl_a, ... 
n_sgl, ptl_sum_sgl_a, ... 
n_dbl, ptl_sum_dbl_b, ... 
n_sgl, ptl_sum_sgl_b)
  legend('1/n double','1/n single', ... 
  '1/n^2 double','1/n^2 single', ... 
  'Location','Best')
  grid on
  xlabel('n')
  ylabel('partial sum')
  title('partial sum vs. n')
  ```
Examples of Numeric Data Types (3 of 3)

Observations:
Plot looks great for the convergent sequence! (they’re on top of each other)

One can see how the single precision calculations don’t keep up with the double precision calculations for the divergent sequence.

Which is more right? Single or double precision?
Character and String Data

Recall the difference between the two:
- Character: A single character (each element requires 2 bytes of space: 2 bytes * 8 bits/byte = 16 bits)
- String: A character array (important note: spaces do count as characters)

Examples:

Input:

A = 'B'
B = 'Brad Sottile'
C = A(1)
D = B(6)

Output:

A = B
B = Brad Sottile
C = B
D = S (why?)
How are characters represented in memory?

Two main ways:
- ASCII  (Mostly personally or “small” computers)
- EBCDIC (mostly mainframes and supercomputers)

Main idea:
- Every character stored in memory has a binary representation and a binary equivalent.
- When you convert from a character to a numeric data type (e.g. a double) you get the decimal equivalent in the ASCII coding system.

Input:

```plaintext
a = double('a')
b = char(98)
```

Output:

```plaintext
a = 97
b = b
```
Symbolic Data

- MATLAB has a symbolic toolbox that uses symbolic data to perform symbolic algebraic operations.

- One method of accomplishing this is with the `sym` function.

- Much, much more on this next week!

Input:

```matlab
my_symbol = sym('x^4/sqrt(y)')
```

Output:

```matlab
my_symbol =
  x^4/y^(1/2)
```
Logical Data

- We’ve discussed the idea of binary logic – 0’s and 1’s to represent binary ideas such as true and false.
- Can use the words `true` and `false` when programming (easier to remember than 0’s and 1’s).

**Input:**

```python
my_logic = [true; false; true]'
x = [1 2 5 3 9]
y = [2 6 3 4 9]
z = x >= 9
```

**Output:**

```python
my_logic = 1 0 1
x = 1 2 5 3 9
y = 2 6 3 4 9
z = 0 0 0 0 1
```
Sparse Arrays/Matrices

- Recall that sparse arrays/matrices are arrays/matrices that are sparsely populated
- To save memory, we can use the `sparse` command
  - If we store a matrix using doubles, it takes $8n$ bytes, where $n$ represents the number of elements in the matrix (example: 1,000 x 1,000 matrix = 1 million elements * 8 bytes = 8 million bytes = 8 MB)
  - The amount of memory saved is a function of how sparse the matrix is – for a 1,000 x 1,000 identity matrix (generated with `sparse(eye(1000))`), this results in a size of 16,004 bytes = 0.016004 MB
- Why do I care? All machines have a finite amount of memory available – particularly when you have multiple memory intensive operations going on you should consider the amount of memory available (MATLAB will throw an error if it runs out). Other strategies include clearing large variables once you don’t need them anymore.
Multidimensional Arrays

- So far we’ve talked about two different types of arrays (row or column, or 2D arrays)
- What if you have data that has multiple attributes? The alternative in this case is to use multidimensional arrays
- Students typically do okay with 3D arrays – but they get really hard to visualize if you go 4D or higher

Input:

```matlab
a = [1 3 7; 5 7 4; 8 4 2];
b = 2*a;
c = 3*b;
my_array(:,:,1) = a;
my_array(:,:,2) = b;
my_array(:,:,3) = c;
disp(my_array)
```

Output:

```
(:,:,1) =
    1     3     7
    5     7     4
    8     4     2
(:,:,2) =
    2     6    14
   10    14     8
   16     8     4
(:,:,3) =
    6    18    42
   30    42    24
   48    24    12
```
Think-Pair-Share

Consider the following code fragment. What will result if this code is run in MATLAB? Think about it for a minute, then find a partner/small group to discuss. We’ll come together then and talk about it as a group.

```matlab
x = 1; z = 1;
for w = 1:5
    while x <= 3
        for y = 1:2
            while z <= 4
                A(w,x,y,z) = w + x + y + z;
                z = z + 1;
            end
            z = 1;
        end
        x = x + 1;
    end
    x = 1;
end
disp(A)
```
Reshaping Arrays: `cat`

- The `cat` command concatenates input arrays along a given dimension `dim` (1 ↔ row, 2 ↔ column, etc.)

- Syntax:
  ```
  cat(dim, A1, A2, A3, …)
  ```

- Note that for non-empty arrays, `C = cat(1,A,B)` is the same effect as `C = [A;B]` and `D = cat(2,A,B)` has the same effect as `D = [A B]`

- Input:
  ```
  A = [1 2; 3 4]
  B = [5 6; 7 8]
  C = cat(1,A,B)
  D = cat(2,A,B)
  ```

- Output:
  ```
  A = 1  2
      3  4
  B = 5  6
      7  8
  C = 1  2
      3  4
      5  6
  D = 1  2  5  6
      3  4  7  8
  ```
Reshaping Arrays: `vertcat`

- **The `vertcat` command** concatenates vertically (up-down)
- **Syntax:**
  ```matlab
  vertcat(A1, A2, A3, ...)
  ```
- **Input:**
  ```matlab
  A = [1 2 3;
       4 5 6;
       7 8 9];
  B = 2*A;
  C = vertcat(A,B)
  ```
- **Output:**
  ```matlab
  C =   1     2     3     4     5     6     7     8     9
       2     4     6     8    10    12    14    16    18
  ```
Reshaping Arrays: `horzcat`

- **The `horzcat` command** concatenates horizontally (left-right)

- **Syntax:**
  `horzcat(A1, A2, ...)`

- **Input:**

  ```
  A = [1  2  3;  
       4  5  6;  
       7  8  9];  
  B = 2*A;  
  C = horzcat(A, B)
  ```

- **Output:**

  ```
  C = 1  2  3  2  4  6
      4  5  6  8 10 12
      7  8  9 14 16 18
  ```
Reshaping Arrays: \texttt{reshape}

- The \texttt{reshape} command reshapes the array.
- Syntax:
  \texttt{reshape(A, r,c \ldots)}
  \texttt{reshape(A,r,[[]])}

- Input:
  \begin{verbatim}
  A = [1  2  3  4; \\
       5  6  7  8; \\
       9 10 11 12]
  B = reshape(A,4,3)
  C = reshape(A,4,[])  
  \end{verbatim}

- Output:
  \begin{verbatim}
  A =  1   2   3   4 \\
       5   6   7   8 \\
       9  10  11  12 \\

  B =  1   6  11 \\
       5  10   4 \\
       9   3   8 \\
       2   7  12 \\

  C =  1   6  11 \\
       5  10   4 \\
       9   3   8 \\
       2   7  12 
  \end{verbatim}
Character Arrays

- Character arrays are arrays of characters (go figure)
- Key idea: The number of elements in each row has to be the same, or MATLAB will throw a warning.
- Remember: MATLAB is converting the letters to ASCII representations in memory!
- Can get around this using a command called `char`

**Input:**

```matlab
name1 = 'Brad'; name2 = 'Abdullah';
name3 = 'Kandhar'; name4 = 'Jake';
name5 = 'Jagruti'; name6 = 'Yibo';
name7 = 'Tian';
my_names1 = char(name1, name2, name3, ...
    name4, name5, name6, name7)
my_names2 = [name1; name2; name3; ...
    name4; name5; name6; name7]
```

**Output:**

```matlab
my_names1 = Brad
       Abdullah
       Kandhar
       Jake
       Jagruti
       Yibo
       Tian
```

Error using `vertcat`
CAT arguments dimensions are not consistent.
Character Arrays Cont’d

- `char` will also happily accept as input the ASCII representation of a number, letter or symbol.

Input:

```plaintext
my_ASCII_data = [92; 89; 94]
my_ASCII_data2 = char(my_ASCII_data)
my_str = num2str(my_ASCII_data)
my_ASCII_data3 = char(my_str)
```

Output:

```plaintext
my_ASCII_data =
    92
    89
    94

my_ASCII_data2 = \n    \Y
    ^
```
Questions?

- Quiz 7
  - Available on ANGEL between 9 a.m. today and 9 p.m. on Tuesday

- Wednesday’s Class
  - Interactive Lecture in Computer Labs

- Lab 7
  - Posted Thursday at noon; due Saturday at noon