Warmup

Assume vectors \( x, y \) contain the coordinates of the vertices of a rectangle:

\[
\begin{align*}
    x &= [3 \ 3 \ 7 \ 7]; \\
    y &= [1 \ 4 \ 4 \ 1];
\end{align*}
\]

Will the following code draw the four sides of the rectangle?

\[
\text{plot}(x, y, 'k-')
\]

A: Yes

B: No
Concatenation

- Concatenate two scalars into a (row-)vector:
  \[ u = \begin{bmatrix} 3 & 1 \end{bmatrix} \]

- Concatenate a scalar onto a (row-)vector:
  \[ v = \begin{bmatrix} u & 4 \end{bmatrix} \]
  \[ v = \begin{bmatrix} 3 & 1 & 4 \end{bmatrix} \]

- Application: repeat the first element of a vector at its end:
  \[ w = \begin{bmatrix} v & v(1) \end{bmatrix} \]
  \[ w = \begin{bmatrix} 3 & 1 & 4 & 3 \end{bmatrix} \]

- Application: append to a vector:
  \[ w = \begin{bmatrix} w & 5 \end{bmatrix} \]
  \[ w = \begin{bmatrix} 3 & 1 & 4 & 3 & 5 \end{bmatrix} \]
Previous Lecture:
- Discrete vs. continuous; finite vs. infinite
- Linear interpolation
- RGB color
- Floating-point arithmetic
- Introduction to vectorized computation

Today’s Lecture:
- Vectorized operations
- Introduction to 2-d array—matrix

Announcements:
- Survey season!
  - Please fill out “Mid-Semester Survey” on CMS
  - Please respond to ENG eval requests (course, TAs, etc.)
- See website for review materials. Optional review session on Sunday, 1:00-2:30pm in Phillips 203.
- Prelim 1 Tuesday 3/10 at 7:30pm, Olin Hall
  - Alt exam: 5:45pm; check e-mail
Studying for exams

1. Write your own solutions to examples from lecture
2. Re-do discussion problems un-aided
3. Answer review questions, using notes as needed
4. Do one old exam, using notes as needed
5. Do a second old exam un-aided – this is your best diagnostic
6. Review specific topics as necessary

Just reading code, solutions will not help!
Initialize arrays if dimensions are known (“pre-allocation”)

… instead of “building” the array one component at a time

% Initialize y
x = linspace(a,b,n);
y = zeros(1,n);
for k = 1:n
    y(k) = myF(x(k));
end

Faster for large n!
BUT you need to know n

% Build y on the fly
x = linspace(a,b,n);
for k = 1:n
    y(k) = myF(x(k));
    % OR
    % y = [y myF(x(k));
end

Totally fine if you don’t know ‘n’ (ignore Matlab’s warning)
Initialize arrays if dimensions are known ("pre-allocation")

... instead of "building" the array one component at a time

```
% Initialize y
x = linspace(a,b,n);
y = zeros(1,n);
for k = 1:n
    y(k) = myF(x(k));
end
```

```
% Build y on the fly
x = linspace(a,b,n);
for k = 1:n
    y(k) = myF(x(k));
    % OR
    %y = [y myF(x(k));
end
```
Vectorized code
—a Matlab-specific feature

- Code that performs element-by-element arithmetic/relational/logical operations on array operands in one step

- Scalar operation:  $x + y$
  where $x$, $y$ are scalar variables
  
  *Single Value (not containing multiple elements)*

- Vectorized code:  $x + y$
  where $x$ and/or $y$ are vectors. Generally, vectors $x$ and $y$ should have the same length and shape
  
  *rows/columns*
Vectorized addition

Matlab code: $z = x + y$

$$\begin{array}{c}
x = \begin{bmatrix} 2 & 1 & .5 & 9 \\ \end{bmatrix} \\
+y = \begin{bmatrix} 1 & 2 & 0 & 2 \\ \end{bmatrix} \\
\hline
= \begin{bmatrix} 3 & 3 & .5 & 11 \\ \end{bmatrix}
\end{array}$$

Observe:

$$z(1) = x(1) + y(1)$$
$$z(2) = x(2) + y(2)$$
$$\vdots$$
$$z(k) = x(k) + y(k)$$
## Vectorized multiplication (vector-vector)

Matlab code: 

\[
\text{c} = \text{a} .* \text{b}
\]
Vectorized element-by-element arithmetic operations on arrays

A dot (.) is necessary in front of these math operators

See full list of ops in §4.1
Shift (scalar-vector addition)

Matlab code: \( z = x + y \)
Reciprocate (scalar-vector division)

\[ \frac{x}{y} = z \]

Matlab code:  
```
z = x .\ (/ y
```
Vectorized

element-by-element arithmetic operations between an array and a scalar

A dot (.) is necessary in front of these math operators

Simplified rule: Use dot for these element-by-element ops: * / ^
When are functions vectorized?

- Many built-in functions (\(\sin()\), \(\text{abs}()\), …)
- When you only use vectorized operations to implement it
- When you loop over the length of the input
  - Note: Matlab treats scalars like length-1 vectors
    
    \[
    x = 3.1; \\
    \text{length}(x) = 1 \\
    x(1) = 3.1
    \]

Not all functions make sense to vectorize (users can always write their own loops, after all)
Can we plot this?

\[ f(x) = \frac{\sin(5x) \exp(-x/2)}{1 + x^2} \]

for

\[-2 \leq x \leq 3\]

Yes!

\[
x = \text{linspace}(-2,3,200);
\]

\[
y = \sin(5*x).*\exp(-x/2)./\left(1 + x.^2\right);
\]

\[
\text{plot}(x,y)
\]

Element-by-element arithmetic operations on arrays
Element-by-element arithmetic operations on arrays…
Also called “vectorized code”

\[
\begin{align*}
x &= \text{linspace}(-2,3,200); \\
y &= \sin(5x) \cdot \exp(-x/2) / (1 + x^2);
\end{align*}
\]

Contrast with scalar operations that we’ve used previously…

\[
\begin{align*}
a &= 2.1; \\
b &= \sin(5a);
\end{align*}
\]

The operators are (mostly) the same; the operands may be scalars or vectors.

When an operand is a vector, you have “vectorized code.”
End of Prelim 1 material
Storing and using data in **tables**

A company has 3 factories that make 5 products with these costs:

<table>
<thead>
<tr>
<th>Products</th>
<th>10</th>
<th>36</th>
<th>22</th>
<th>15</th>
<th>62</th>
</tr>
</thead>
<tbody>
<tr>
<td>factories</td>
<td>12</td>
<td>35</td>
<td>20</td>
<td>12</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>37</td>
<td>21</td>
<td>16</td>
<td>59</td>
</tr>
</tbody>
</table>

What is the best way to fill a given purchase order?
An array is a **named** collection of **like** data organized into rows and columns.

A 2-d array is a table, called a **matrix**.

Two **indices** identify the position of a value in a matrix, e.g.,

\[
\text{mat}(r,c)
\]

refers to component in row \( r \), column \( c \) of matrix \( \text{mat} \).

Array indices still start at 1.

**Rectangular**: all rows have the same # of columns.
Indexing example

<table>
<thead>
<tr>
<th></th>
<th>M(1,1)</th>
<th>M(1,2)</th>
<th>M(1,3)</th>
<th>M(1,4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

M refers to the whole matrix. M(3,2) refers to the element (value) in the third row, second column.
Creating a matrix

- Built-in functions: `ones()`, `zeros()`, `rand()`
  - E.g., `zeros(2, 3)` gives a 2-by-3 matrix of 0s
  - E.g., `zeros(2)` gives a 2-by-2 matrix of 0s
- “Build” a matrix using square brackets, `[ ]`, but the dimension must match up:
  - `[x  y]` puts `y` to the right of `x`
  - `[x; y]` puts `y` below `x`
  - `[4 0 3; 5 1 9]` creates the matrix
  - `[4 0 3; ones(1,3)]` gives
  - `[4 0 3; ones(3,1)]` doesn’t work
Working with a matrix: 

`size()` and individual components

Given a matrix \( M \)

\[
\begin{bmatrix}
2 & -1 & .5 & 0 & -3 \\
3 & 8 & 6 & 7 & 7 \\
5 & -3 & 8.5 & 9 & 10 \\
52 & 81 & .5 & 7 & 2
\end{bmatrix}
\]

\[
[\text{nr}, \text{nc}] = \text{size}(M) \quad \% \text{ nr is \# of rows}
\]

\[
\% \text{ nc is \# of columns}
\]

\[
\text{nr} = \text{size}(M, 1) \quad \% \# \text{ of rows}
\]

\[
\text{nc} = \text{size}(M, 2) \quad \% \# \text{ of columns}
\]

\[
\text{n} = \text{size}(M) \quad \% \text{ size}() \text{ \textbf{is weird like this}}
\]

\[
\% \text{n is length 2 vector since M is 2-d:}
\]

\[
\% \quad \text{n(1) is \# of rows, n(2) is \# of cols}
\]

\[
M(2,4) = 1;
\]

\[
\text{disp}(M(3,1))
\]
Working with a matrix: \( \text{size()} \) and individual components

Given a matrix \( M \) and the script below

Which statement(s) could make the update shown in purple on the diagram?

\[
\begin{align*}
[nr, \ nc] &= \text{size}(M) ; \\
n &= \text{size}(M) ; \\
M(1, nc) &= 4; \quad \text{\( \leftarrow A \)} \\
M(1, n(2)) &= 4; \quad \text{\( \leftarrow B \)} \\
M(0, 4) &= 4; \quad \text{\( \leftarrow C \)}
\end{align*}
\]

D: None of A, B, C

E: More than one of A, B, C
Example: minimum value in a matrix

```
function val = minInMatrix(M)
% val is the smallest value in matrix M
```

**Best-in-set pattern:**

- Initialize best-so-far
- Loop over set:
  - If current value is better than best-so-far:
    - Update best-so-far
Pattern for traversing a matrix $M$

\[
\begin{align*}
[nr, nc] &= \text{size}(M) \\
\text{for} &\quad r = 1:nr \\
&\quad \quad \% \text{ At row } r \\
&\quad \quad \text{for} \quad c = 1:nc \\
&\quad \quad \quad \% \text{ At column } c \text{ (in row } r) \\
&\quad \quad \quad \% \\
&\quad \quad \quad \% \text{ Do something with } M(r,c) \ldots \\
&\quad \quad \text{end} \\
&\text{end}
\end{align*}
\]