Initialize arrays if dimensions are known

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

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

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

Much faster for large n!

---

Drawing a polygon (multiple line segments)

```matlab
% Draw a rectangle with the lower-left corner at (a, b), width w, height h.
x = [a  a+w  a+w  a    a  ]; % x data
y = [b  b    b+h  b+h  b  ]; % y data
plot(x, y)
```

Fill in the missing vector values!

---

Coloring a polygon (fill)

```matlab
% Draw a rectangle with the lower-left corner at (a, b), width w, height h, and fill it with a color named by c.
x = [a  a+w  a+w  a    a  ]; % x data
y = [b  b    b+h  b+h  b  ]; % y data
fill(x, y, c)
```

Built-in function `fill` does the “wrap-around” automatically.

---

Coloring a polygon (fill)

```matlab
x = [0.1 -9.2 -7 4.4];
y = [9.4 7 -6.2 -3];
fill(x, y, 'g')
```

Can be a vector (RGB values)

---

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

- Vectorized code: `x + y`
  where `x` and/or `y` are vectors. If `x` and `y` are both vectors, they must be of the same shape and length

---

Announcements:
- Discussion this week in classrooms as listed in Student Center
- Prelim 1 on Oct 13 (Thursday) at 7:30pm
Vectorized addition

\[
\begin{align*}
x & = \begin{bmatrix} 2 & 1 & .5 & 8 \end{bmatrix} \\
y & = \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix} \\
z & = x + y = \begin{bmatrix} 3 & 3 & .5 & 9 \end{bmatrix}
\end{align*}
\]

Matlab code: \( z = x + y \)

Vectorized multiplication

\[
\begin{align*}
a & = \begin{bmatrix} 2 & 1 & .5 & 8 \end{bmatrix} \\
b & = \begin{bmatrix} 1 & 2 & 0 & 1 \end{bmatrix} \\
c & = a \times b = \begin{bmatrix} 2 & 2 & 0 & 8 \end{bmatrix}
\end{align*}
\]

Matlab code: \( c = a \times b \)

Vectorized element-by-element arithmetic operations on arrays

\[
\begin{align*}
+ & \quad \rightarrow \\
- & \quad \rightarrow \\
\times & \quad \rightarrow \\
/ & \quad \rightarrow
\end{align*}
\]

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

Shift

\[
\begin{align*}
x & = \begin{bmatrix} 3 \end{bmatrix} \\
y & = \begin{bmatrix} 2 & 1 & .5 & 8 \end{bmatrix} \\
z & = x + y = \begin{bmatrix} 5 & 4 & 3.5 & 11 \end{bmatrix}
\end{align*}
\]

Matlab code: \( z = x + y \)

Reciprocate

\[
\begin{align*}
x & = \begin{bmatrix} 1 \end{bmatrix} \\
y & = \begin{bmatrix} 2 & 1 & .5 & 8 \end{bmatrix} \\
z & = x / y = \begin{bmatrix} .5 & 1 & 2 & 125 \end{bmatrix}
\end{align*}
\]

Matlab code: \( z = x ./ y \)

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

\[
\begin{align*}
+ & \quad \rightarrow \\
- & \quad \rightarrow \\
\times & \quad \rightarrow \\
/ & \quad \rightarrow \\
\times & \quad \rightarrow \\
/ & \quad \rightarrow
\end{align*}
\]

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

Not necessary but OK to use dot: \( \times, \times, \times, / \)
Can we plot this?

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

for

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

Yes!

\[ x = \text{linspace}(-2, 3, 200); \]
\[ y = \sin(5x) \cdot \exp(-x/2) \div (1 + x.^2); \]
\[ \text{plot}(x, y) \]

Element-by-element arithmetic operations on arrays

Concatenating 2 vectors—copy 2 vectors into a new one

\% given column vectors \(x\) and \(y\)
\[ v = \text{zeros}((\text{length}(x)) + \text{length}(y), 1); \]
\[ \text{for } k = 1: \text{length}(x) \]
\[ v(k) = x(k); \]
\[ \text{end} \]
\[ \text{for } k = 1: \text{length}(y) \]
\[ v(\text{length}(x) + k) = y(k); \]
\[ \text{end} \]

This is non-vectorized code—operations are performed on one component (scalar) at a time

Split a vector in 2—copy values into 2 vectors

\% given row vector \(v\)
\[ s = \text{ceil}(\text{rand} \times \text{length}(v)); \% \text{split after } v(s) \]
\[ x = \text{zeros}(1, s); \]
\[ y = \text{zeros}(1, \text{length}(v) - s); \]
\[ \text{for } k = 1: s \]
\[ x(k) = v(k); \]
\[ \text{end} \]
\[ \text{for } k = 1: \text{length}(y) \]
\[ y(k) = v(s + k); \]
\[ \text{end} \]

This is non-vectorized code—operations are performed on one component (scalar) at a time

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

Connections between webpages:

- 0 0 1 0 0
- 1 0 0 1 1
- 0 1 0 1 1
- 1 0 1 0 1
- 0 0 1 1 0
- 0 1 0 1 0
- 1 1 0 1 0

What is the best way to fill a given purchase order?

---

**2-d array: matrix**

- 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., `mat(r,c)` refers to component in row r, column c of matrix `mat`.
- Array index starts at 1.
- **Rectangular:** all rows have the same # of columns.

---

**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`:

<table>
<thead>
<tr>
<th>2</th>
<th>-1</th>
<th>5</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>-3</td>
<td>8.5</td>
<td>9</td>
</tr>
<tr>
<td>52</td>
<td>81</td>
<td>5</td>
<td>7</td>
</tr>
</tbody>
</table>

- `[nr, nc] = size(M)`  % nr is # of rows, % nc is # of columns
- `nr = size(M, 1)`  % # of rows
- `nc = size(M, 2)`  % # of columns

- `M(2,4) = 1;`
- `disp(M(3,1))`
- `M(1,nc) = 4;`

---

**Example: minimum value in a matrix**

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

---

**Pattern for traversing a matrix `M`**

```matlab
[nr, nc] = size(M)
for r = 1:nr
    % At row r
    for c = 1:nc
        % At column c (in row r)
        %
        % Do something with M(r,c) ...
    end
end
```
Matrix example: Random Web

- N web pages can be represented by an N-by-N Link Array $A$.
- $A(i,j)$ is 1 if there is a link on webpage $j$ to webpage $i$.
- Generate a random link array and display the connectivity:
  - There is no link from a page to itself.
  - If $i \neq j$ then $A(i,j) = 1$ with probability $\frac{1}{1 + |i - j|}$.
  - There is more likely to be a link if $i$ is close to $j$.

```matlab
function A = RandomLinks(n)
% A is n-by-n matrix of 1s and 0s representing n webpages
A = zeros(n,n);
for i=1:n
    for j=1:n
        r = rand(1);
        if i~=j && r<= 1/(1 + abs(i-j));
            A(i,j) = 1;
        end
    end
end
```