• Today’s Lecture:
  – Vectorized logical operations and review

• Announcements:
  – Prepare for Test 2: review lab exercise 3 (char and string, vectorized code), cell arrays, and the functions you wrote (and given to you) for A2
  – Wednesday 11/16: no lecture; help session in Hollister 464 computer lab
  – Thursday 11/17: Assignment 2 due
  – Friday 11/18: Test 2 during lab session
  – Monday 11/21: Test 2, take 2, for those who need it during lecture time in Hollister 464 computer lab
Vectorized computation can simplify code

```matlab
function dMin = minDistance(x,y)
% Given a set of points, find the distance between the origin and the
% point nearest to the origin (0,0).
% x, y: vectors of the same length storing the x- and y-coordinates
% of a set of points.
% dMin: distance between (0,0) and the point nearest to (0,0).

n= length(x);
d= zeros(1,n);
for k= 1:n
    d(k)= sqrt(x(k)^2 + y(k)^2);
end
dMin= min(d);

d= sqrt(x.^2 + y.^2);
dMin= min(d);
```

```matlab
function dMin = minDistance(x,y)
% Given a set of points, find the distance between the origin and the
% point nearest to the origin (0,0).
% x, y: vectors of the same length storing the x- and y-coordinates
% of a set of points.
% dMin: distance between (0,0) and the point nearest to (0,0).

n= length(x);
d= zeros(1,n);
for k= 1:n
    d(k)= sqrt(x(k)^2 + y(k)^2);
end
dMin= min(d);

d= sqrt(x.^2 + y.^2);
dMin= min(d);
```
Strings and vectorized code

Given strings \texttt{a=‘hello’} and \texttt{b=‘jello’} ...

- No. of characters that are the same in \texttt{a} and \texttt{b}: \texttt{sum(a==b)}
- No. of characters that are different in \texttt{a} and \texttt{b}: \texttt{sum(a~=b)}
Vectorized (logical) operations – 1d

\[ a = [4 \ 2 \ 6 \ 1 \ 3] ; \]
\[ b = [5 \ 3 \ 6 \ 5 \ 3] ; \]

Relational ops:
\[ L = a == b \]
\[ M = a > b \]

Arithmetic ops:
\[ c = a - b \]

Extraction:
\[ d = a(a == b) \]
\[ e = b(b > 3 \ \& \ \text{rem}(b, 2) == 1) \]
Vectorized (logical) operations – 1d

\[ a = \begin{bmatrix} 4 & 2 & 6 & 1 & 3 \end{bmatrix}; \]
\[ b = \begin{bmatrix} 5 & 3 & 6 & 5 & 3 \end{bmatrix}; \]

**Relational ops:**
\[ L = a == b \quad \begin{bmatrix} 0 & 0 & 1 & 0 & 0 \end{bmatrix} \quad \text{Type: Logical} \]
\[ M = a > b \quad \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \end{bmatrix} \]

**Arithmetic ops:**
\[ c = a - b \quad \begin{bmatrix} -1 & -1 & 0 & -4 & 0 \end{bmatrix} \quad \text{Type: double} \]

**Extraction:**
\[ d = a(a == b) \quad \begin{bmatrix} 6 & 3 \end{bmatrix} \]
\[ e = b(b > 3 \&\& \text{rem}(b, 2) == 1) \quad \begin{bmatrix} 5 & 5 \end{bmatrix} \quad \text{for vectorized code} \]
Vectorized (logical) operations – 2d

\[
m = \begin{bmatrix}
  2 & 3 & 5 & 7; \\
  -2 & 1 & 0 & 7; \\
  5 & 2 & -1 & 8
\end{bmatrix}
\]

\[
L = m > 3
\]

\[
P = m > 3 \lor m < 0
\]

\[
a = m (m > 4)
\]

\[
b = (m > 4) \cdot m
\]
Vectorized (logical) operations – 2d

\[ m = \begin{bmatrix} 2 & 3 & 5 & 7; \, \ldots \\ -2 & 1 & 0 & 7; \, \ldots \\ 5 & 2 & -1 & 8 \end{bmatrix} \]

\[ L = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \]

\[ P = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix} \]

\[ a = \begin{bmatrix} 5 \\ 7 \\ 9 \end{bmatrix} \]

\[ b = \begin{bmatrix} 2 & 3 & 5 & 7 \\ -2 & 1 & 0 & 7 \\ 5 & 2 & -1 & 8 \end{bmatrix} \]

\[ \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 3 & 5 & 7 \\ -2 & 1 & 0 & 7 \\ 5 & 2 & -1 & 8 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 7 \\ 0 & 0 & 7 \\ 5 & 0 & 8 \end{bmatrix} \]
Replacing all occurrences of a character

• How do you (easily) replace all the non-letter characters in a string with a space?
• Vectorized code to the rescue!

• Find all non-upper-case letters in str:
  \[
  \text{str(str<‘A’ | str>’Z’)}
  \]
• Replace these non-upper-case letters with a space:
  \[
  \text{str(str<‘A’ | str>’Z’) = ‘ ’}
  \]
Array vs. Cell Array

• Simple array
  – Each component stores one scalar. E.g., one char, one double, or one logical value
  – All components have the same type

• Cell array
  – Each cell can store something “bigger” than one scalar, e.g., a vector, a matrix, a string (vector of chars)
  – The cells may store items of different types
\[ m = ['hello';...] \iff m = ['h' 'e' 'l' 'l' 'o';...] \iff \text{same} \iff ['j' 'e' 'l' 'l' 'o'] \]
## ASCII characters

(American Standard Code for Information Interchange)

<table>
<thead>
<tr>
<th>ascii code</th>
<th>Character</th>
<th>ascii code</th>
<th>Character</th>
</tr>
</thead>
<tbody>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>65</td>
<td>‘A’</td>
<td>48</td>
<td>‘0’</td>
</tr>
<tr>
<td>66</td>
<td>‘B’</td>
<td>49</td>
<td>‘1’</td>
</tr>
<tr>
<td>67</td>
<td>‘C’</td>
<td>50</td>
<td>‘2’</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
<tr>
<td>90</td>
<td>‘Z’</td>
<td>57</td>
<td>‘9’</td>
</tr>
<tr>
<td>:</td>
<td>:</td>
<td>:</td>
<td>:</td>
</tr>
</tbody>
</table>
Arithmetic and relational ops on characters

- ‘c’ – ‘a’ gives 2
- ‘6’ – ‘5’ gives 1
- letter1=‘e’; letter2=‘f’;
- letter1−letter2 gives -1
- ‘c’>‘a’ gives true
- letter1==letter2 gives false
- ‘A’ + 2 gives 67
- char(‘A’+2) gives ‘C’
Example: toUpper

Write a function toUpper(cha) to convert character cha to upper case if cha is a lower case letter. Return the converted letter. If cha is not a lower case letter, simply return the character cha.

Hint: Think about the distance between a letter and the base letter ‘a’ (or ‘A’). E.g.,

```
 a b c d e f g h ...
```

```
A B C D E F G H ...
```

\[ \text{distance} = \text{‘g’-‘a’} = 6 = \text{‘G’-‘A’} \]

Of course, do not use Matlab function upper!
function up = toUpper(cha)
% up is the upper case of character cha.
% If cha is not a letter then up is just cha.

up = cha;

cha is lower case if it is between 'a' and 'z'
function up = toUpper(cha)
% up is the upper case of character cha.
% If cha is not a letter then up is just cha.

up= cha;

if ( cha >= 'a' && cha <= 'z' )

    % Find distance of cha from 'a'
    offset= cha - 'a';

    % Go same distance from 'A'
    up= char('A' + offset);

end
Example: censoring words

function D = censor(str, A)
% Replace all occurrences of string str in
% character matrix A with X’s, regardless of
% case.
% Assume str is never split across two lines.
% D is A with X’s replacing str.

Function strcmpi does case-insensitive string comparison
function D = censor(str, A)
% Replace all occurrences of string str in character matrix A,
% regardless of case, with X's.
% A is a matrix of characters.
% str is a string. Assume that str is never split across two lines.
% D is A with X's replacing the censored string str.

D= A;
ns= length(str);
[nr,nc]= size(A);

% Build a string of X's of the right length

% Traverse the matrix to censor string str
function D = censor(str, A)
% Replace all occurrences of string str in character matrix A,
% regardless of case, with X's.
% A is a matrix of characters.
% str is a string. Assume that str is never split across two lines.
% D is A with X's replacing the censored string str.

D= A;
ns= length(str);
[nr,nc]= size(A);

% Build a string of X's of the right length
Xs= char( zeros(1,ns));for k= 1:ns
  Xs(k)= 'X';
end

% Traverse the matrix to censor string str
for r= 1:nr
  for c= 1:nc-ns+1
    if strcmpi( str , A(r, c:c+ns-1) )==1
      D(r, c:c+ns-1)= Xs;
    end
  end
end

Returns an array of type double
Changes the type to char

Case insensitive comparison of strings