

Vectors & Matrices

- Vectors & Matrices store sets of values, all of which have the same type.
- *row vector*
- *column vector*
- *scalar*
- *matrix*
- *elements*

Creating row vectors

- $v = [1,2,3,4]$
- $v = [1\ 2\ 3\ 4]$
- Colon operator (*iterator*): create equally spaced numbers
 - **from : step : to**
 - $v = 1:1:5$
 - $v = 1:5$
 - $v = 1:2:9$
 - $1:2:6$
 - $6:3$
 - $9:-2:1$
- $\text{linspace}(\text{from}, \text{to}, n)$
 - $\text{linspace}(1,5,5)$
 - $\text{linspace}(3,15,5)$

Concatenating vectors

- $a = [1\ 2]$
- $b = [3\ 4\ 5]$
- $b = [[3\ 4]\ 5]$
- $c = [a\ b]$
- $c = [a\ b\ 1\ 2\ 3]$

Indexing vectors

- **variable (index)**
- `v=10:15`
- `v(3)` "*v sub 3*"
- index itself can be a vector
 - `v([1 2 3])`
- the indexed entries can be modified:
 - `v(2)=30`
- Matlab automatically extends vector if indexed element does not exist.
 - `v(20)=1000`
 - Avoid automatic extension when you care about speed.

Exercise

- What is the value of `a` after the following statements are executed?
 - `a=2:2:8`
 - `a(4)=33`
 - `a(6)=11`
 - `a=a(3:6)`
 - `a=[a linspace(4,12,3)]`

Creating column vectors

- $c = [1; 2; 3; 4]$
- Row vectors can be transposed using $'$
- $r = 1:3$
- $c = r'$
- Exercise: what will the following produce?
– $1:3'$

Creating matrix variables

- $m=[4\ 3\ 1; 2\ 5\ 6]$
- $m=[4\ 3\ 1$
 $2\ 5\ 6]$
- There must always be the same number of elements in each row.
- $m=[2:4; 3:6]$

Row-Column Indexing Matrices

- $m = \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix}$
- $m(2,3)$
- $m(1:2,2:3)$
- $m(1,:)$
- $m(:,2)$
- $m(2:end,1)$

Linear Indexing

- Matlab stores and indexes matrices column-by-column.
- $m = \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix}$
- $m(1)$
- $m(2)$
- $m(\text{end}-1:\text{end})$

Modifying matrix elements

- $m = \begin{bmatrix} 2 & 3 & 4 \\ 5 & 6 & 7 \end{bmatrix}$
- $m(1,1) = 9$
- $m(1,1:2) = 9$
- $m(1,1:2) = [8 \ 11]$
- $m(1,:) = 9$
- $m(1,:) = [9 \ 9]$
- $m(5,:) = 1:3$

Matrix Dimensions

- `size`
- ~~`length`~~
- `numel`
- `m=rand(2,3)`
- `size(m)`
- `[r c] = size(m)`
- ~~`length(m)`~~
- `numel(m)`
- Exercise:
 - Write function that takes a matrix `m` as input, and returns a matrix of zeros with the same size as `m`.

Changing Dimensions

- reshape
 - `m=randi(100,3,4)`
 - `reshape(m,2,6)`
 - `reshape(m,4,3)`
 - `reshape(m,4,[])`
- `fliplr(m)`
- `flipud(m)`
- `rot90(m)d` *rotates counterclockwise*
 - `rot90(m,-1)`
 - `rot90(m,2)`

Replicating matrix

- `repmat(m, r, c)`
- `m=[1 2; 3 4];`
- `repmat(m,1,3)`
- `repmat(m,2,2)`
- `repmat(m,2,3)`

Using functions/operators with vectors & matrices

- `v=-3:4`
- `abs(v)`
- `m=round(rand(3,4)*10)`
- `sin(m)`
 - `a=randi(10,3,4)`
 - `b=randi(10,3,4)`
- `a+b`
- `a-b`
- `a*b`
- `a.*b`
- `a^2`
- `a.^2`

Empty vectors

- `e=[]`
 - `size(e)`
 - ~~`length(e)`~~
 - `numel(e)`
- Empty vectors can be used to delete elements from vectors/matrices
 - `m=randi(10,3,4)`
 - `m(:,4)=[]`
 - `m(1,:)=[]`
 - `m(2:3)=[]`

Three dimensional matrices

- `m=zeros(3,3,2)`
- `m(:,:,1)=randi(100,3,3)`
- `m(:,:,2)=randi(100,3,3)`

Exercise

- 1.32: Find an "efficient" way to generate the following matrix:

m =

7	8	9	10
12	10	8	6

- Increment the first row of the above matrix m with +1, and the second row of m with +2, in a single statement. If matrix m had more than two rows, your code should add +3 to the third row, +4 to the fourth row, etc.

Plotting

- `plot(X,Y)`
- `plot(X,Y,'r*')`

- `axis([xlow, xhigh, ylow, yhigh])`
- `xlabel('time (sec)')`
- `ylabel('temperature (Fahrenheit)')`
- `title('Temperature vs. time')`
- `grid on`
- `hold on/off`
- Saving plots as image files (File -> SaveAs).

Exercise

- Draw four random triangles using the `plot()` function. Use a different color, marker, and line type for each triangle.

Bar-plot

- `bar(Y)`
- `bar(X, Y)`