EEE 208 – Programming for EEE

Assist. Prof. Dr. Engin Mendi

Application: Polynomial Roots 1

In general, to solve the polynomial $a_n x^n + a_{n-1} x^{n-1} + a_{n-2} x^{n-2} + \ldots + a_1 x + a_0 = 0$, we can enter all the coefficients (including the 0 ones) in a row vector and then use the function roots.

```
>> coeff = [an, an-1, an-2 , … , a1 , a0]
>> roots(coefff)
```

Question: How many roots does a polynomial of order n have?

Application: Polynomial Roots 2

Example: To find the roots of $x^3 - 7x^2 + 40x - 34 = 0$, enter coefficients in a row vector (from the highest power to the lowest one)

```
>> coeff = [1,-7,40,-34];
>> roots(coeff)
ans =
     3.0000 + 5.000i
     3.0000 - 5.000i
     1.0000
```

So, the roots are $x = 1$ and $x = 3 \pm 5i$.

Element-by-element multiplication for Arrays, Example

```
» a=[1 2 3];b=[4;2;1];
» a.*b, a./b, a.^b → all errors
» a.*b', a./b', a.^(b') → all valid
```

$$\begin{bmatrix}1 & 2 & 3\end{bmatrix} .* \begin{bmatrix}4\\2\\1\end{bmatrix} = ERROR$$

$$\begin{bmatrix}1\\2\\3\end{bmatrix} .* \begin{bmatrix}4\\2\\1\end{bmatrix} = \begin{bmatrix}4\\4\\3\end{bmatrix}$$

$$3\times1 .* 3\times1 = 3\times1$$

$$\begin{bmatrix}1 & 1 & 1\\2 & 2 & 2\\3 & 3 & 3\end{bmatrix} .* \begin{bmatrix}1 & 2 & 3\\1 & 2 & 3\\1 & 2 & 3\end{bmatrix} = \begin{bmatrix}1 & 2 & 3\\2 & 4 & 6\\3 & 6 & 9\end{bmatrix}$$

$$3\times3 .* 3\times3 = 3\times3$$

$$\begin{bmatrix}1 & 2\\3 & 4\end{bmatrix} .\wedge 2 = \begin{bmatrix}1^2 & 2^2\\3^2 & 4^2\end{bmatrix}$$

Can be any dimension

Creating Matrices from Vectors

- Suppose `a = [2,4,6]` and `b = [8,10,12]` (two row vectors)
- What is the difference between the results given by [a b] and [a;b]?
-

```
>> c = [a b];
c =
    2   4   6   8   10   12
>> d = [a;b]
d =
    2   4   6
    8   10   12
```

Matrices: an Example

- Suppose we have 3 row vectors, each with 4 columns. Each vector shows a year and each column shows a season. The elements show the average temperature in that season. The whole data arrangement is called a Matrix; here called `mean_temp`

- Remember, spaces or commas separate elements in different columns, but semicolons separate elements in different rows.

Matrices – Concatenation

$y1 = [22\ 30\ 13\ 6]$

$y2 = [23\ 29\ 12\ 4]$

$y3 = [21\ 31\ 15\ 8]$

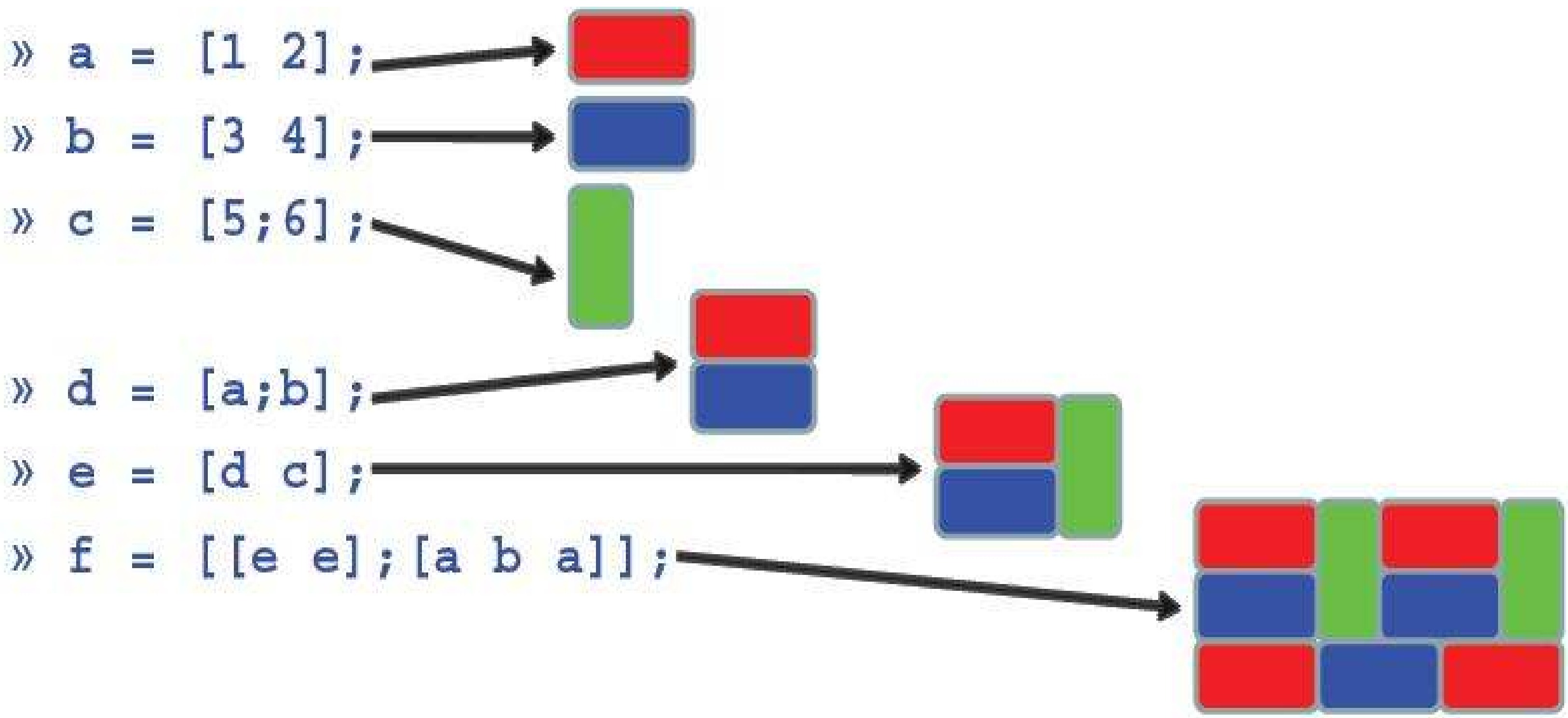
- Defining the matrix using row vectors
- `mean_temp = [y1;y2;y3]`

- Defining the matrix using column vectors

$$s1 = \begin{bmatrix} 22 \\ 23 \\ 21 \end{bmatrix} \quad s2 = \begin{bmatrix} 30 \\ 29 \\ 31 \end{bmatrix} \quad s3 = \begin{bmatrix} 13 \\ 12 \\ 15 \end{bmatrix} \quad s4 = \begin{bmatrix} 6 \\ 4 \\ 8 \end{bmatrix}$$

- `mean_temp = [s1,s2,s3,s4]`

Concatenation & Dimension



Size of Matrices

- Size of a matrix is in general expressed in m-by-n, where m is the number of rows and n is the number of columns
- `>> size(mean_temp) = 3 4`
- `size(mean_temp,1)` : number of rows
- `size(mean_temp,2)` : number of columns
- Length of a matrix= its largest dimension
- `>> length(mean_temp) = 4`

Element-by-element Multiplication for Matrices

- If we have two matrices A and B

$$\mathbf{A} = \begin{bmatrix} 11 & 5 \\ -9 & 4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -7 & 8 \\ 6 & 2 \end{bmatrix}$$

- Then C = A.*B equals

$$\mathbf{C} = \begin{bmatrix} 11(-7) & 5(8) \\ -9(6) & 4(2) \end{bmatrix} = \begin{bmatrix} -77 & 40 \\ -54 & 8 \end{bmatrix}$$

Matrix-Matrix Multiplication

- In the product of two matrices **AB**, the number of *columns* in **A** must equal the number of *rows* in **B**. The row-column multiplications form column vectors, and these column vectors form the matrix result. The product **AB** has the same number of *rows* as **A** and the same number of *columns* as **B**.

- In summary, if A is a matrix with m rows and n columns, and B is a matrix with n rows and p columns, then C=A*B is a matrix with m rows and p columns

Matrix-Matrix Multiplication: Examples

$$\begin{bmatrix} 6 & -2 \\ 10 & 3 \\ 4 & 7 \end{bmatrix} \begin{bmatrix} 9 & 8 \\ -5 & 12 \end{bmatrix} = \begin{bmatrix} (6)(9) + (-2)(-5) & (6)(8) + (-2)(12) \\ (10)(9) + (3)(-5) & (10)(8) + (3)(12) \\ (4)(9) + (7)(-5) & (4)(8) + (7)(12) \end{bmatrix}$$

$$= \begin{bmatrix} 64 & 24 \\ 75 & 116 \\ 1 & 116 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} * \begin{bmatrix} 4 \\ 2 \\ 1 \end{bmatrix} = 11$$

$$1 \times 3 * 3 \times 1 = 1 \times 1$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \wedge 2 = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

Must be square to do powers

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 9 \\ 6 & 12 & 18 \\ 9 & 18 & 27 \end{bmatrix}$$

$$3 \times 3 * 3 \times 3 = 3 \times 3$$

Inverse of a Matrix

>> `inv(A)` returns inverse of the square matrix A.
If A is not a square matrix or if it's a singular matrix, MATLAB returns an error message.

Application: Solving Linear Equations

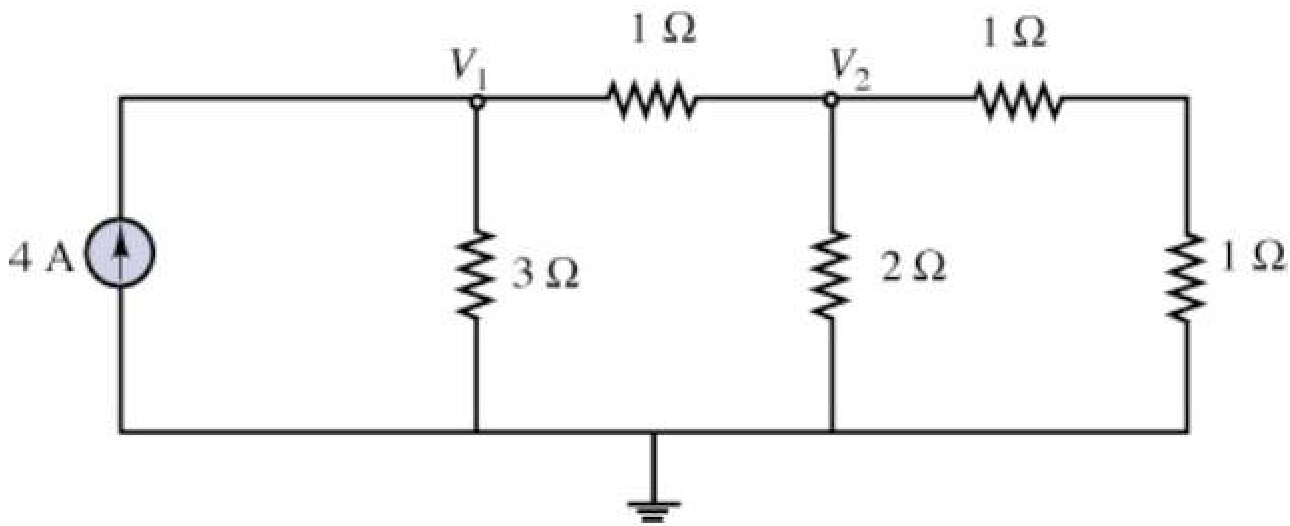
$6x + 12y + 4z = 70$
$7x - 2y + 3z = 5$
$2x + 8y - 9z = 64$

```
>> A = [6,12,4;7,-2,3;2,8,-9]; % a 3-by-3 matrix
>> b = [70;5;64]; % a 3-by-1 vector
>> Solution=inv(A)*b; % a 3-by-1 vector
Or
>> Solution = A\b
Solution =
     3
     5
     -2
```

The solution is $x = 3$, $y = 5$, and $z = -2$.

Class Exercise 1

Problem: Find voltages v1 and v2 in the following circuit using the KCL rule.



Class Exercise 1, Instructions

1. Write two KCL equations for nodes 1 and 2.
2. Simplify the equations to find V1 and V2 and write the equations in the matrix form.
3. Open a new m-file, define a matrix for coefficients (left-side) and another one for constants (right-side).
4. Use matrix operations to find the vector of unknowns.

Special Matrices, I

- `>> zeros(m,n)` or `zeros(n)`
- `>> ones(m,n)` or `ones(n)`
- `>> eye(m,n)` or `eye(n)`: Identity matrix

- Example:
- `>> A=[2 -3 0;-1 4 2]`
- `>> eye(2)*A`
- `>> B=zeros(size(A))`

Special Matrices, 2

- `>> diag(v,k)`: if v is a vector with n components, this is a square matrix of order n+abs(k) with the elements of v on the k-th diagonal. k = 0 is the main diagonal, k > 0 is above the main diagonal and k < 0 is below the main diagonal.

- `diag(2)`
- `diag (3,1)`
- `diag([2 3],0)`
- `diag([1 -1 2],2)`

Special Matrices, 3

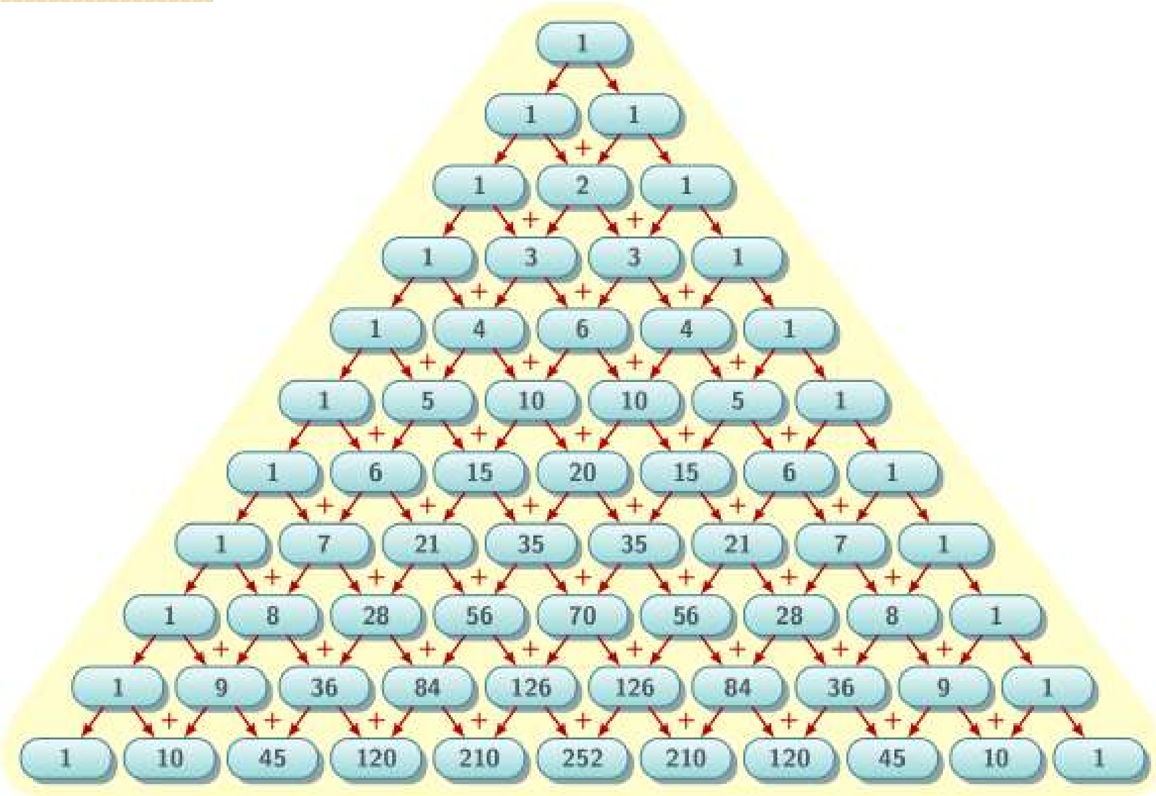
- `>> magic(n)`: an n-by-n matrix constructed from the integers 1 through n^2 with equal row, column, and diagonal sums. Produces valid magic squares for all n > 0 except n=2.
- `>> open magic` to see the algorithm and documentation

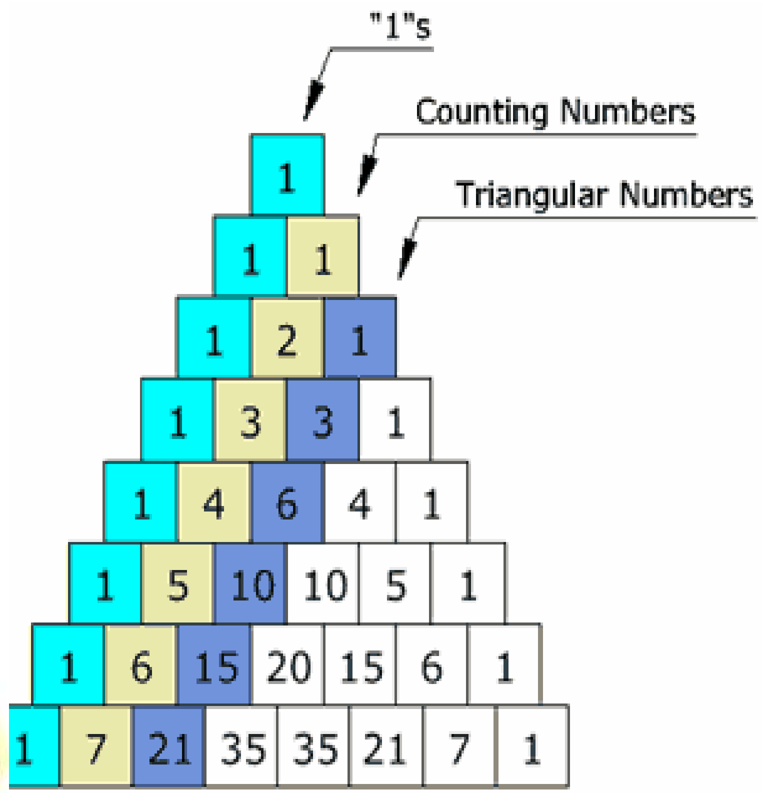
- Example:

```
for n=1:3
    magic(n)
end
```

Special Matrices, 4

- `>> pascal(n)` is the Pascal matrix of order N: a symmetric positive definite matrix with integer entries, made up from Khayyam-Pascal triangle. Its inverse has integer entries.





Array Addressing, I

- First generate two vectors
    - `>> u = 0:0.1:10;`
    - `>> w = 5*sin(u);`
- `>> length(u)`
- `plot(u,w)`

Accessing a Vector's Single Element

- Note: MATLAB indexing starts with **1, not 0**

- `>> u(21)`
- `>> w(90)`

- You can also assign values to a single element of a matrix:
- `u(50)=2*exp(-3);`

Array Addressing, 2

- The colon operator selects individual elements, rows, columns, or "subarrays" of arrays.

```
>> v = exp(linspace(1,2,10))
```

v(:) represents all the row or column elements of the vector v.

v(2:5) represents the second through fifth elements

Array Addressing, 3

- `>> D = [2  4  6; 8  10  12; 14 16 18]`
- `D(1,3)` shows the element on the 1st row and 3rd column
- `D(:,3)` denotes all the elements in the third column of the matrix D.
- `D(:,1:2)` denotes all the elements in the first to second columns of D.
- `D(2:3,1:3)` denotes all the elements in the second and third rows that are also in the first through third columns.
- `u = D(:)` creates a vector u consisting of all the columns of D stacked from first to last.
- `D(end,:)` shows the last row in D, and D(:,end) shows the last column.

Array Addressing, 4

You can use array indices to extract a smaller array from another array.

```
>> B = [2 4 10 -13; 16 -3 7 18; 8 4 9 0;
-2 12 15 -4.5]

>> C = B(2:3, 3:4)
```

Then `C =`

```
     7     18
     9      0
```

Array Functions, I

`A = [6 2;-10 -5;3 0]`

- `>> length(A)` computes either the number of elements of `A` if `A` is a vector or the largest value of `m` or `n` if `A` is an `m×n` matrix.

- `>> size(A)` returns a row vector [m n] containing the sizes of the m-by-n array `A`.

Try `size(A,1)` and `size(A,2)`

- `>> find` returns indices …

`find(A>0)` or `find(x)` where x is a logical array

Array Functions, 2

`A = [6 2;-10 -5;3 0]`

- `>> sort(A)` Sorts each column of the array A in ascending order and returns an array the same size as A.

- `>> sum(A)` Sums the elements in each column of the array A and returns a row vector containing the sums.

- `>> prod(A)` For vectors, `prod(A)` is the product of the elements of A. For matrices, `prod(A)` is a row vector with the product over each column.
Example: `prod(A)` and `prod(A(1, : ))`

Class Exercise 2

Problem. Open a new m-file and write the codes to create the following vectors:

- Vector a = 2, 4, 6, ..., 20
- Vector b = 1, 1/2, 1/3, ..., 1/9
- Vector c = 0, 1/2, 2/3, 3/4, ... , 8/9
- Vector d = $[10^0\ 10^1\ 10^2 \ldots 10^{0.99}\ 10^1]$

  Hint. Use the **logspace** command.
- Vector e = $\log_{10}(1/d)$

Class Exercise 3

Problem. Open a new m-file and write the codes to create the following vectors. Assign each value to the right variable.

Let x = [-3 2 6.5 -1]. Create the following vectors:

- x1. Add 13 to each element
- x2. Add 4 to just the odd-index elements
- x3. Compute the square root of each element
- x4. Compute the sum of vector elements.

Class Exercise 4

Problem: Open a new m-file and generate these matrices using zeros, ones, diag, or a combination of them.

- mat_E= $\begin{bmatrix} 2 & \cdots & 2 \\ \vdots & \ddots & \vdots \\ 2 & \cdots & 2 \end{bmatrix}$ a 5x5 matrix full of 2's
- mat_F= $\begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$ a 5x5 matrix with 0's everywhere, and elements 1 2 3 2 1 on the diagonal

Array Functions, 3

`A = [6 2;-10 -5;3 0]`

- `>> max(A)` Returns the algebraically largest element in `A` if `A` is a vector, and returns a row vector containing the largest elements in each column if **A** is a matrix.
- If any of the elements are complex, max(A) returns the elements that have the largest magnitudes.
- `>> [x,k] = max(A)` Similar to `max(A)` but stores the maximum values in the row vector `x` and their indices in the row vector `k`.

- `>> min(A)` and `[x,k] = min(A)` are similar to `max` but returns minimum values.

Array Functions, 4

`B = [2 3i; 2-4i 5]`

- `>> transpose(B)` or `B.'` returns the non-conjugate transpose of B.
- `>> B'` returns the transpose and the conjugate of complex elements

- `>> rand(m,n)` returns an m-by-n matrix with pseudorandom values drawn from the standard uniform distribution on the open interval(0,1)

- `>> nan(m,n)` or `NaN(m,n)` returns a matrix of NaNs (useful for representing uninitialized variables)

Array Functions, 5

- `>> reshape(x,m,n)` returns the m-by-n matrix whose elements are taken columnwise from x.

- Try `reshape(A,6,1)` and `reshape(A,2,3)`

- `reshape` changes the size, but not the values or order of the data in memory.
  - `>> C = [ 3  4.2  8 ; -6.7  12  0.75 ];`
  - `>> D = reshape(C,[3 2]);`

Name = 'C'	Name = 'D'
Size = [2 3];	Size = [3 2];

Array Functions, 6

- `>> round(3.4), round(5.65)`

`round(A)` rounds the elements of A to the nearest integers.

- `>> floor(3.4), floor(5.65)`

`floor(A)` rounds the elements of A to the nearest integers towards $-\infty$.

- `>> ceil(3.4), ceil(5.65)`

`ceil(A)` (ceiling) rounds the elements of A to the nearest integers towards $+\infty$.

Array Functions, 7

`>> rand(n)` or `rand(m,n)` return values drawn from the standard uniform distribution on the open interval(0,1).

Example:

```
>> mat = rand(4,5);
>> mat(end,end) % gives the last element
>> mat([1 end],[end-2:end]) % from the
first and last row, gives elements from
the last 3 columns
```

Array Functions, 8

>> `randn(n)` or `rand(m,n)` return values drawn from the standard normal distribution

Array Functions: Practice

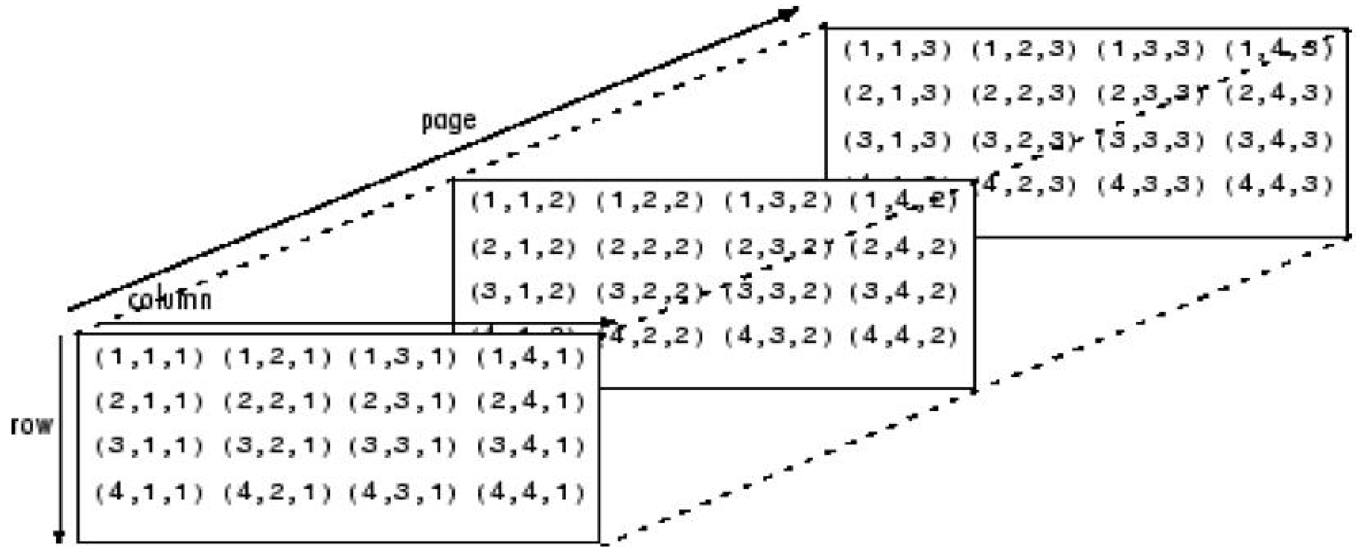
```
>> M = magic(4)
>> vec = reshape(M,1,prod(size(M)))
%Finds the number of elements in M, and
  puts them in a row vector
>> [min_value, min_index] = min(vec)
%extracts the values and index of minimum
  elements

>> N = rand(4,3)
>> ind=find(N>0.5)
>> [rowind,colind]=find(N>0.5)
```

Multidimensional Arrays - Pages

- The row and column vectors have one dimension (1D), like the axis of real number
- Matrices have two dimensions (2D), like the Cartesian plane
- But sometimes we need to use a database with more than 2 features (properties). In this case, we can create an array with 3 or more dimensions. Think of it as a book with several pages.

Multidimensional Arrays - Pages



Consists of two-dimensional matrices "layered" to produce a third dimension. Each "layer" is called a page

`cat(n,A,B,C, ...)` creates a new array by concatenating the arrays `A,B,C,` and so on along the dimension n.

Multidimensional Arrays: Example

```
>> A = eye(4)
>> B = magic(4)
>> page_mat = cat(3,A,B)

>> size(page_mat) = 4 4 2
```

This size shows the number of rows, columns, and pages

```
>> page_2 = cat(4,A,B)
```

What is the size now?

With these two matrices, one dimension becomes extra

Function rat

- `rat` returns the *rational fraction approximation*
- In Rational numbers, the nominator and denominators are integers.
- `>> rat(0.3333)`
- `>> rat(pi)`
- `>> rat(e)`

Function rats

- `rats` returns a string containing *simple rational approximations*
- `>> rats(0.3333)`
- `>> rats(pi)`
- `>> rats(e)`
- In CE_08, compare rat and rats:

```
>> b = 1./[1:9]
>> rat(b)
>> rats(b)
>> c = [0:8]./[1:9]
>> rat(c)
>> rats(c)
```