M'hamed Bougara University - Boumerdès
Faculty of Sciences
Computer Science Department

Course: Programming Tools for Mathematics Matlab
L1

Dr RAHMOUNE Nabila

2023/2024

Chapter 2
Arrays: Vectors and Matrices

Outline

- Vectors
- Array operations
- Matrices
- Entering Numeric Arrays
- Applications

Vectors

- A row vector in MATLAB can be created by an explicit list, starting with a left bracket, entering the values separated by spaces (or commas) and closing the vector with a right bracket.

- A column vector can be created the same way, and the rows are separated by semicolons.

Example:

```
>> x = [ 0   0.25*pi   0.5*pi   0.75*pi   pi ]
x =
         0    0.7854    1.5708    2.3562    3.1416
>> y = [ 0; 0.25*pi; 0.5*pi; 0.75*pi; pi ]

y =
         0
    0.7854
    1.5708
    2.3562
    3.1416
```

x is a row vector.

y is a column vector.

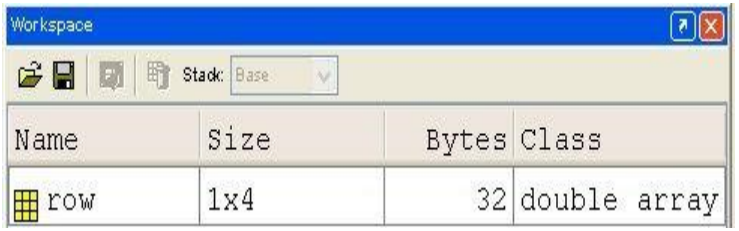
Vectors

- row = [1 2 3.2 4 6 5.4];
- row = [1, 2, 4, 7, 4.3, 1.1];

- Command window:

```
>> row=[1 2 5.4 -6.6]

row =

    1.0000    2.0000    5.4000   -6.6000
```

Workspace:

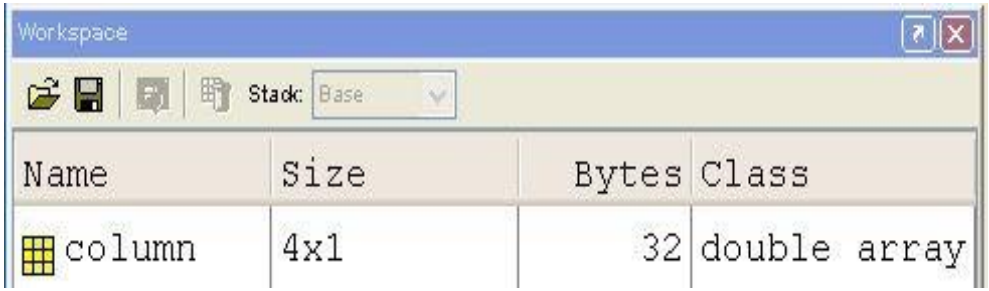


- col = [1; 2; 3.2; 4; 6; 5.4];

Command window:

```
>> column=[4;2;7;4]

column =

     4
     2
     7
     4
```

Workspace:



Vectors

Vector Addressing – A vector element is addressed in MATLAB with an integer index enclosed in parentheses.
Example:

```
>> x(3)
ans =
   1.5708
```

← 3rd element of vector x

- The colon notation may be used to address a block of elements.

(start : increment : end)

start is the starting index, increment is the amount to add to each successive index, and end is the ending index. A shortened format (start : end) may be used if increment is 1.

- Example:

```
>> x(1:3)
ans =
        0    0.7854    1.5708
```

← 1st to 3rd elements of vector x

NOTE: MATLAB index starts at 1.

Vectors

Some useful commands:

x = start:end	create row vector x starting with start, counting by one, ending at end
x = start:increment:end	create row vector x starting with start, counting by increment, ending at or before end
length(x)	returns the length of vector x
y = x’	transpose of vector x
dot (x, y)	returns the scalar dot product of the vector x and y.

Array Operations

Scalar-Array Mathematics

For addition, subtraction, multiplication, and division of an array by a scalar simply apply the operations to all elements of the array.

Example:

```
>> f = [ 1 2; 3 4]
f =
     1     2
     3     4

>> g = 2*f – 1
g =
     1     3
     5     7
```

Each element in the array f is multiplied by 2, then subtracted by 1.

Array Operations

Element-by-Element Array-Array Mathematics.

Operation	***Algebraic Form***	***MATLAB***
Addition	a + b	a + b
Subtraction	a – b	a – b
Multiplication	a x b	a .* b
Division	a ÷ b	a ./ b
Exponentiation	a^b	a .^ b

- **Example:**

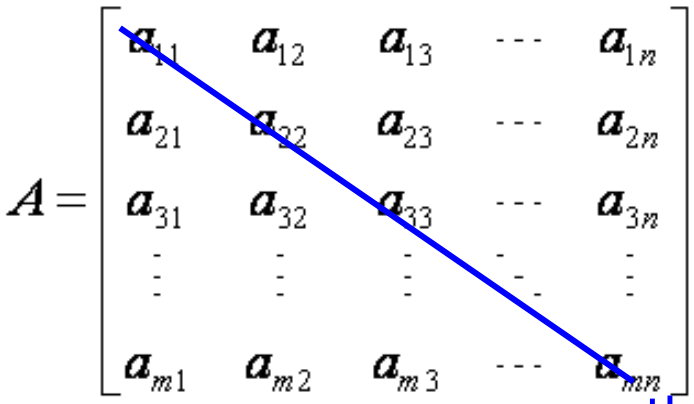
```
>> x = [ 1 2 3 ];
>> y = [ 4 5 6 ];
>> z = x .* y
z =
     4    10    18
```

Each element in x is multiplied by the corresponding element in y.

Matrices

- A Matrix array is two-dimensional, having both multiple rows and multiple columns, similar to vector arrays:
  - it begins with [, and end with]
  - spaces or commas are used to separate elements in a row
  - semicolon or enter is used to separate rows.

A is an m x n matrix.

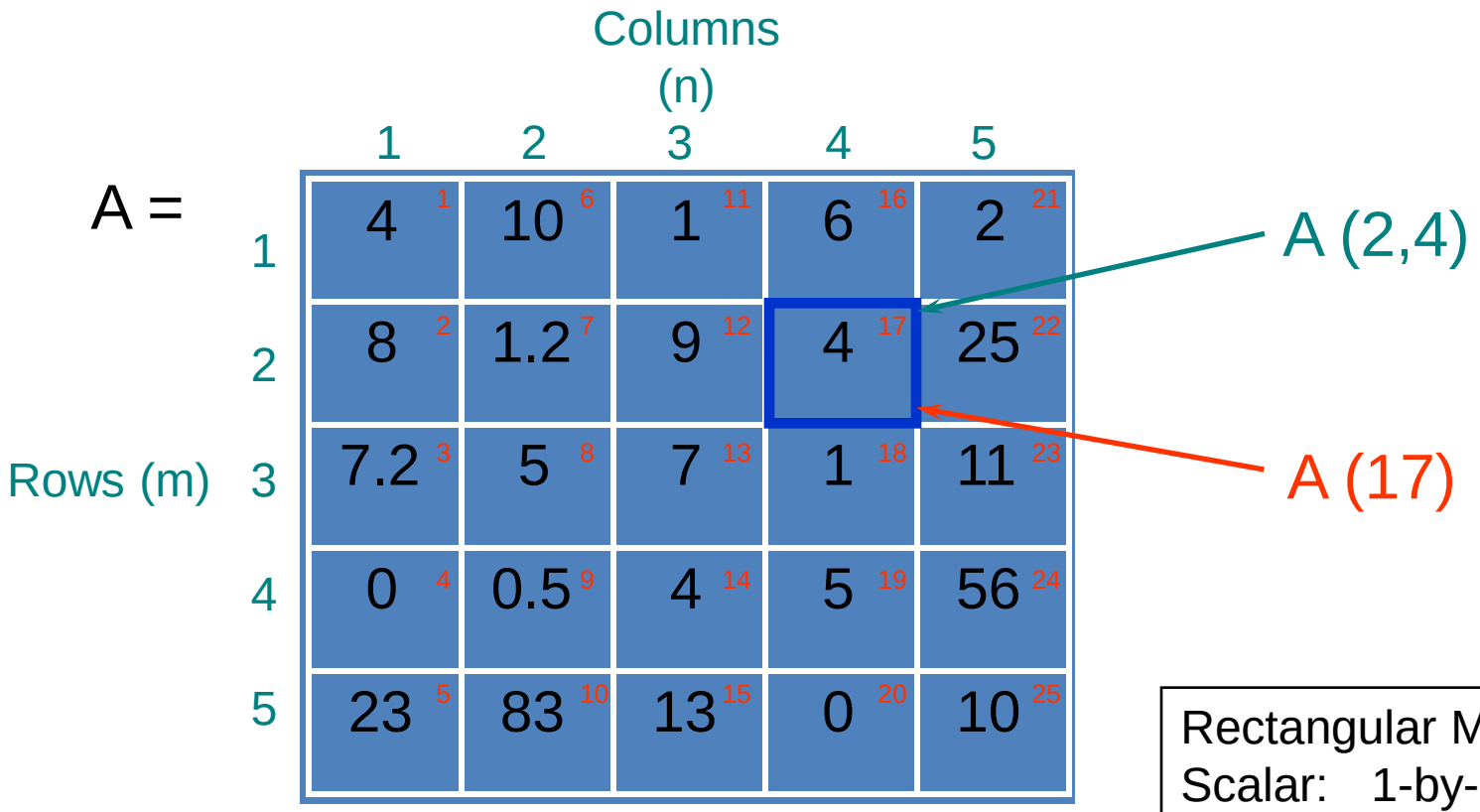


the main diagonal

•Example:

```
>> f = [ 1 2 3; 4 5 6]
f =
     1     2     3
     4     5     6
>> h = [ 2 4 6
1 3 5]
h =
     2     4     6
     1     3     5
```

Matrices



Matrix elements can be EITHER numbers OR characters

Rectangular Matrix:
Scalar: 1-by-1 array
Vector: m-by-1 array
1-by-n array
Matrix: m-by-n array

Matrices

Matrix Addressing:

-- *matrixname(row, column)*

-- **colon** may be used in place of a row or column reference to select the entire row or column.

- Example:

```
>> f(2,3)

ans =

    6

>> h(:,1)

ans =

    2

    1
```

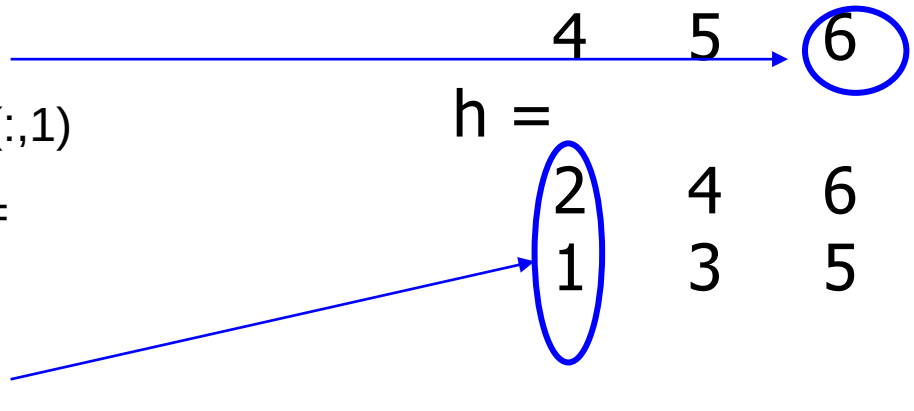
recall:

f =

1	2	3
4	5	6

h =

2	4	6
1	3	5



Matrices

Some useful commands:

zeros(n)	returns a n x n matrix of zeros
zeros(m,n)	returns a m x n matrix of zeros
ones(n)	returns a n x n matrix of ones
ones(m,n)	returns a m x n matrix of ones
size (A)	for a m x n matrix A, returns the row vector [m,n] containing the number of rows and columns in matrix.
length(A)	returns the larger of the number of rows or columns in A.

Matrices

More commands

Transpose	B = A’
Identity Matrix	eye(n) ➔ returns an n x n identity matrix eye(m,n) ➔ returns an m x n matrix with ones on the main diagonal and zeros elsewhere.
Addition and subtraction	C = A + B C = A – B
Scalar Multiplication	B = αA, where α is a scalar.
Matrix Multiplication	C = A*B
Matrix Inverse	B = inv(A), A must be a square matrix in this case. rank (A) ➔ returns the rank of the matrix A.
Matrix Powers	B = A.^2 ➔ squares each element in the matrix C = A * A ➔ computes A*A, and A must be a square matrix.
Determinant	det (A), and A must be a square matrix.

A, B, C are matrices, and m, n, α are scalars.

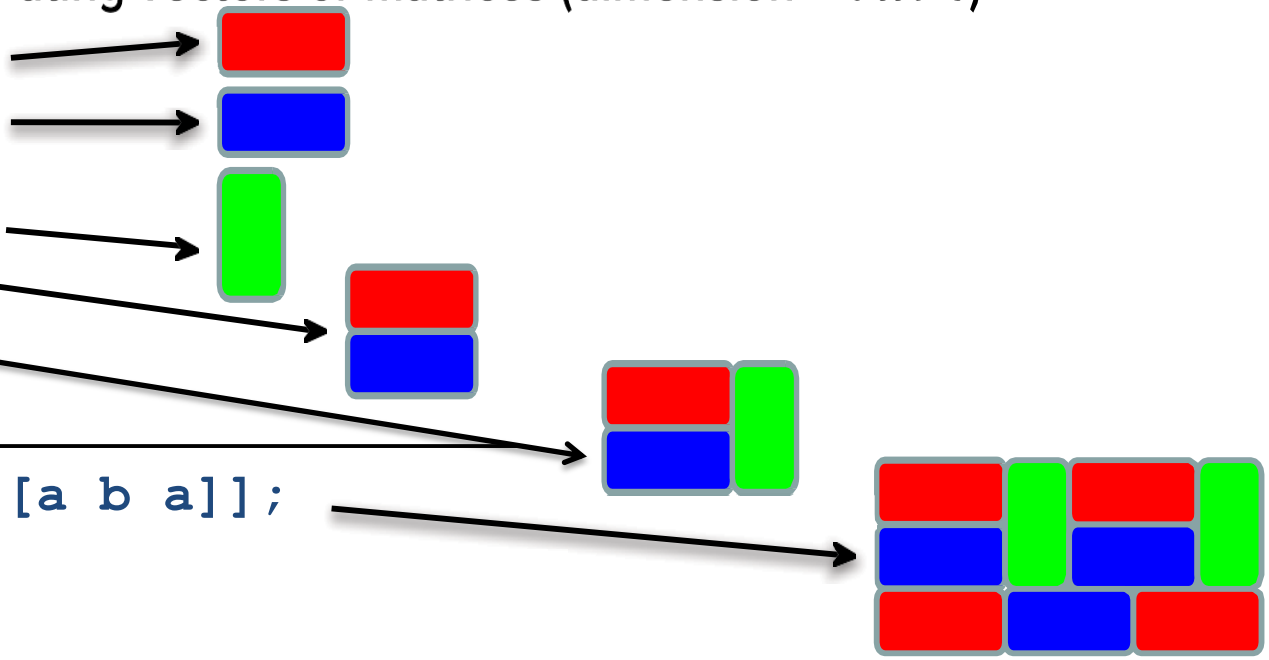
Matrices

- Make matrices like vectors
  - Element by element
    - a= [1 2;3 4];

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

- By concatenating vectors or matrices (dimension matters)

```
a = [1 2];
b = [3 4];
c = [5;6]
d = [a;b];
e = [d c];

f = [[e e];[a b a]];
```



```
str = ['Hello I am '];
```

Strings are character vectors

Entering Numeric Arrays

Row separator:
semicolon (;)

Column separator:
space / comma (,)

```
»  a=[1 2;3 4]
a =
     1     2
     3     4
»  b=[-2.8, sqrt(-7), (3+5+6)*3/4]
b =
   -2.8000   0 + 2.6458i   10.5000
»  b(2,5) = 23
b =
   -2.8000  0 + 2.6458i  10.5000   0         0
         0         0         0    0   23.0000
```

Use square brackets [ ]

Matrices must be rectangular. (Set undefined elements to zero)

Any MATLAB expression can be entered as a matrix element

Entering Numeric Arrays

Scalar expansion

Creating sequences:
colon operator (:)

Utility functions for creating matrices.
(Ref: Utility Commands)

```
»   w=[1 2;3 4] + 5
w =
    6    7
    8    9
»   x = 1:5
x =
    1    2    3    4    5
»   y = 2:-0.5:0
y =
  2.0000   1.5000   1.0000   0.5000   0
»   z = rand(2,4)
z =
  0.9501   0.6068   0.8913   0.4565
  0.2311   0.4860   0.7621   0.0185
```

Numerical Array Concatenation

Use [ ] to combine existing arrays as matrix "elements"

Row separator:
semicolon (;)

Column separator:
space / comma (,)

The resulting matrix must be rectangular.

```
»  a=[1 2;3 4]
a =
     1     2
     3     4
»  cat_a=[a, 2*a; 3*a, 4*a; 5*a, 6*a]
cat_a =
     1     2     2     4
     3     4     6     8
     3     6     4     8
     9    12    12    16
     5    10     6    12
    15    20    18    24
```

Use square brackets [ ]

4*a

Arrays : Vectors and Matrix

Example :

A= 1 2 3 4
5 6 7 8
9 10 11 12

```
>> A = [1,2,3,4 ; 5,6,7,8 ; 9,10,11,12]

>> A=[1:4;5:8;9:12]

>> A = [1,2,3,4

5,6,7,8

9,10,11,12] ;

>> A=[[1;5;9] , [2;6;10] , [3;7;11] , [4;8;12]] ;
```

Arrays : Vectors and Matrix

```
t =1:10

t =
         1   2   3   4  5  6   7  8   9   10

k =2:-0.5:-1

k =
         2  1.5  1  0.5  0  -0.5  -1

B = [1:4; 5:8]

B =
1    2    3    4
5    6    7    8
```

Referencing and access to elements of a vector

Access to the elements of a vector is done using the following syntax:

- parentheses ( ) are used for consultation

  <vectorName(position)>

  % position can be a simple number, a list of numbers (position vector)

- hooks [ ] for creation

Referencing and access to elements of a vector

```
V= 5  -1  13   -6  7

>> V(3)        % 3rd position
 ans =13
 >> V(2:4)     % from second position to fourth
 ans =  -1    13    -6

 >> V(4:-2:1)  % from the 4th position to the 1st with step = -2
 ans =   -6    -1

 >> V(3:end)     % from the 3rd position to the last
  ans =    13    -6    7

 >> V([1,3,4])   % 1st, 3rd and 4th position only
  ans =   5    13    -6
```

Array and Matrix Index

- Matrix indices begin from 1 (not 0!!!)
- Matrix indices must be positive integers

```
A =
     1     2     3
     4     5     6
     7     8     9
```

```
>> A(1,3)
ans =
     3
```

```
>> A(1)
ans =
     1
```

```
>> A(2)
ans =
     4
```

A(**1**) ↔ $A(\mathbf{1},1)$
A(**2**) ↔ $A(\mathbf{2},1)$

Column-Major Order

```
>> A(0)
Subscript indices must either be real positive integers or logicals.
>> A(1,4)
Index exceeds matrix dimensions.
>>
```

Matrix Index (example)

Given:

```
A =

     3     5     3
     6     8     2
     2     7     3
```

```
>> A(3,2)

ans =

     7
```

```
>> A(2,:)

ans =

     6     8     2
```

```
>> A(1:2,2)

ans =

     5
     8
```

A(-2), A(0)

Error: ??? Subscript indices must either be real positive integers or logicals.

A(4,2)
Error: ??? Index exceeds matrix dimensions.

Matrix Index (example)

```
A =
     1     2     3
     4     5     6
     7     8     9
```

```
>> A(2,2:3)

ans =

     5     6
```

```
>> A(2,1:end)

ans =

     4     5     6
```

```
>> A(2,:)

ans =

     4     5     6
```

```
>> A(2,1:2:3)

ans =

     4     6
```

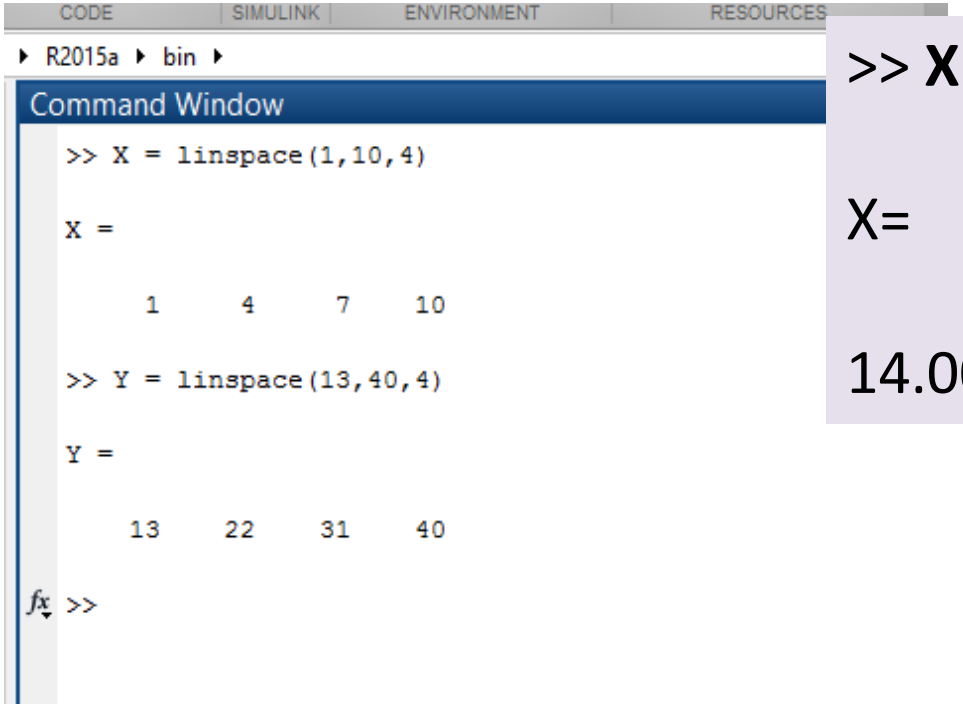
```
>> A(2,[1 3])

ans =

     4     6
```

```
>> A(:)

ans =

     1
     4
     7
     2
     5
     8
     3
     6
     9
```

Generating Vectors from functions

`>> X = linspace(1,10,4)` `%` a vector of four elements from 1 to 10

`>> Y = linspace(13,40,4)` % a vector of four elements from 13 to 40



>> **X = linspace(14,10,4)**

X=

14.0000 12.6667 11.3333 10.0000

Generating Vectors from functions

```
V= [5   -1        13    -6   7]
```

```
>> V(1) = 8      % give the value 8 to the first element
```

```
>> V(6) = -3     % add a sixth element with value -3
```

```
>> V(2) = []     % delete the second element
```

```
>> V(3:5) = []   % delete from 3rd to 5th element
```

Matrix Concatenation

```
>> x = [1 2], y = [4 5], z=[ 0 0]

    A = [ x y]

          1   2   4   5

  B = [x ; y]

        1 2
        4 5
```

```
C = [x y ;z]
```

Error:

??? Error using ==> vertcat CAT arguments dimensions are not consistent.

Numerical Array Concatenation - [ ]

Use [ ] to combine existing arrays as matrix “elements”

Row separator:
semicolon (;)

Column separator:
space / comma (,)

The resulting matrix must be rectangular.

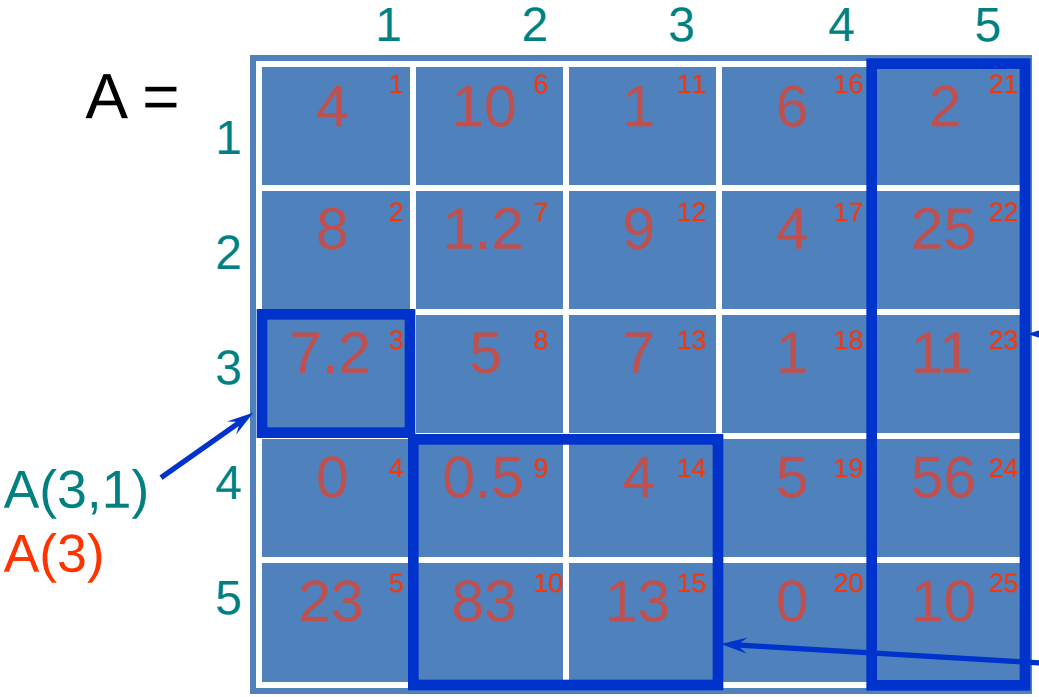
```
»   a=[1 2;3 4]
a =
    1    2
    3    4
»   cat_a=[a, 2*a; 3*a, 4*a; 5*a, 6*a]
cat_a =
    1    2    2    4
    3    4    6    8
    3    6    4    8
    9   12   12   16
    5   10    6   12
   15   20   18   24
```

Use square brackets [ ]

4*a

»num_cat

Array Subscripting / Indexing



- Use () parentheses to specify index
- colon operator (:) specifies range / ALL
- [ ] to create matrix of index subscripts
- ‘end’ specifies maximum index value

Exercise: Creating Matrices

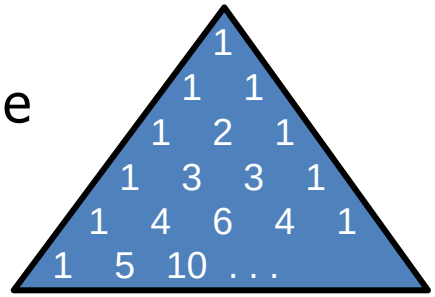
Create a 5x5 Pascal matrix ("P_mat")

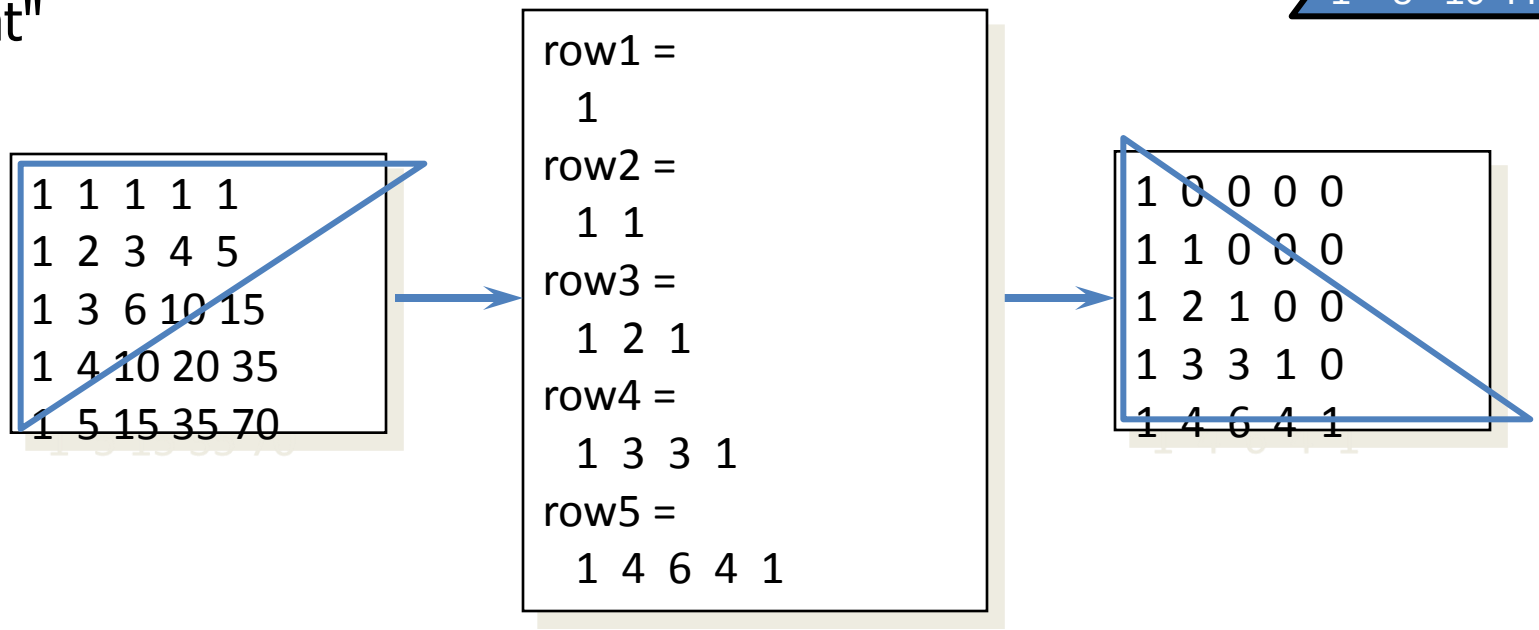
Extract elements to create row vectors for the first 5 rows of Pascal's Triangle

(Look for patterns in the element locations in "P_mat")

Combine the vectors into a single matrix with zeros above the diagonal

"P_mat"





Solution: Creating Matrices

```
» P_mat = pascal(5)                    % Create Pascal's Matrix

» row1 = P_mat(1)                      % Extract Rows
» row2 = P_mat(2:4:6) .....
» row5 = P_mat(5:4:21)

» new_mat = row1                       % Create New Matrix
» new_mat(2,1:2) = row2 .....
» new_mat(5,1:5) = row5
» % NOTE: GENERALIZED FORM:
» % Nmax = length(P_mat);
» % rowN = P_mat(N:Nmax-1:(N-1)*Nmax+1)
» % new_mat(N,1:N) = rowN
```

»create_mat

Solution: Creating Matrices

```
» P_mat = pascal(5)                    % Create Pascal's Matrix

%  Rearrange Matrix - Pascal Triangle in upper triangle
%  Flip Matrix Horizontally (LEFT-RIGHT):
%  Could also use FLIPLR():
» temp = P_mat(1:5, 5:-1:1)

% Extract diagonals (rows of Pascal's Triangle)
» row1 = diag(temp,4)'
» row2 = diag(temp,3)'
» row3 = diag(temp,2)'
» row4 = diag(temp,1)'
» row5 = diag(temp)'
% Create New Matrix as before
```

»create_mat

Solution: Creating Matrices (cont)

```
» P_mat = pascal(5)          % Create Pascal's Matrix


% Rearrange Matrix - Pascal Triangle in lower triangle
%  Flip Matrix Vertically (UP-DOWN):
%  Could also use FLIPUD():
» temp1 = P_mat(5:-1:1, 1:5)

% Extract lower triangular matrix:
» temp2 = tril(temp1)
% Create New Matrix by sorting columns of "temp2":
» new_mat = sort(temp2)
```

Matrices Operations

Add: +
Subtract: -

```
>> u = [-2, 6, 1] ;
>> v = [ 3, -1, 4] ;
```

```
>> u+2
Ans =
      0   8   3

>> u+v
ans =

       1     5      5
```

```
>> u-2
Ans =
      -4   4   -1

>> u-v
ans =

       -5     7      -3
```

Matrices Operations

Given A and B:

```
>> A = [1 2 3;4 5 6;7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9
```

```
>> B = [3 5 2; 5 2 8; 3 6 9]

B =

     3     5     2
     5     2     8
     3     6     9
```

Addition

```
>> X = A + B

X =

     4     7     5
     9     7    14
    10    14    18
```

Subtraction

```
>> Y = A - B

Y =

    -2    -3     1
    -1     3    -2
     4     2     0
```

Product

```
>> Z = A * B

Z =

    22    27    45
    55    66   102
    88   105   159
```

Transpose

```
>> T = A'

T =

     1     4     7
     2     5     8
     3     6     9
```

Matrices Operations

Divide: **./** % element by element

Multiply: **.*** % element by element

Power: **.^** (e.g. .^2 means squared) % element by element

You can use round brackets to specify the order in which operations will be performed

Matrices Operations

```
>> u = [-2, 6, 1] ;
>> v = [ 3, -1, 4] ;
```

```
>> u*2
ans =
      -4   12   2

>> u.*2
ans =

       -4     12      2

>> u.*v
 ans=
         -6     -6    4
```

```
>> u/2
  ans =
 -1.0000   3.0000 0.5000
>> u./2
  ans =
 -1.0000   3.0000 0.5000
>> u./v
  ans =
 -0.6667   -6.0000      0.2500
```

```
>> u.^2
ans =
   4   36       1
>> u.^v
ans =
 -8.0000   0.1667     1.0000
```

Matrices Operations

```
A = [1 2 3; 5 1 4; 3 2 1]
    A =
          1    2    3
          5    1    4
          3    2   -1
```

```
x = A(1,:)

x=
     1   2   3
```

```
y = A(3 ,:)

y=
     3  2  -1
```

```
b = x .* y

b=
     3  4 -3
```

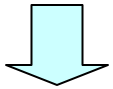
```
c = x . / y

c=
   0.33   0.5   -3
```

```
d = x .^2

d=
     1    4    9
```

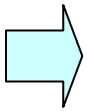
```
K= x^2
Erorr:
 ??? Error using ==> mpower  Matrix must be square.
B=x*y
Erorr:
??? Error using ==> mtimes Inner matrix dimensions must agree.
```

Matrices Operations

```
A = [1 2 3; 5 1 4; 3 2 1]
    A =
          1     2     3
          5     1     4
          3     2    -1
```



```
x = A(1,:)        y = A(3 ,:)

x=                y=
    1   2   3         3  4  -1
```



```
b = x .* y

b=
     3  8 -3
```

```
c = x . / y

c=
   0.33   0.5   -3
```

```
d = x
.^y

d=
       1    16    0.33
```

Logical functions and operators

- ✓any (vecA) returns 1 if one of the elements of vecA is not null
- ✓all (vecA) returns 1 if all elements of vecA are null

- ✓isequal (x,y) retourns 1 if x and y are equal
- ✓ischar (x) retourns 1 if x is a character

...

Special function

Commande	**Signification**
Sum(V)	La somme des éléments d'un vecteur V
prod (V)	Le produit des éléments d'un vecteur V
mean (V)	La moyenne des éléments d'un vecteur V
sort (V)	Ordonne les éléments du vecteur V par ordre croissant
length(V)	la taille du vecteur V
max(V)	plus grand élément du vecteur V
min(V)	Plus petit élément du vecteur V
fliplr(V)	renverse l'ordre des éléments du vecteur V

Exercises

Questions	
Create the line vector in two ways (9 7 5 3 1).	X=[9 7 5 3 1]; X=9:-2:1;
Create the column vector in two ways (10 9.5 9 8.5 8).	Y=[10; 9.5; 9; 8.5; 8]; Y=(10:-0.5:8)’;
Create the line vector (9 7 5 3 1 9 7 5 3 1 9 7 5 3 1) from the previous vectors.	X3=[X X X];
Create the column vector (10 9.5 9 8.5 8 10 9.5 9 8.5 8 10 9.5 9 8.5 8) from the previous vectors.	Y3=[Y;Y;Y];

Exercises

Questions
Enter polynomials $p = 2x^5 - x^2 + 5x$ et $q = x^2 + 1$.
Calculer les racines de p et q
Définir le vecteur V qui contient 50 valeurs compris entre 1 et 3
Evaluer le polynôme P sur les points de V.

```
p=zeros(1,6);
q=zeros(1,3);
 p(1)=2; p(4)=-1; p(5)=5;
q(1)=1; q(3)=1;

  roots(p) ;
   roots(q) ;


 V=linspace(1,3,50) ;


  Polyval(p,v) ;
```

Application 2

Compute :

$$\sum_{n=0}^{100} \sin^n x$$

$$\text{pour } x = \frac{\pi}{5}.$$

Solution :

1- create a vector pow with integer from 0 to 100,

2- create a second vector t each element is **sin ∏/5 power n**.

3- calculate sum of t' elements

```
>> pow = 0 : 100 ;
>> t = sin ( pi /5) .^ pow ;
>> s = sum( t )
```

User interaction

Reading : input(‘message’)
Writing : disp(‘message’)

Exercise: Calculate the sum of two integers entered by the user.

Solution 1 :

```
v1= input(' Enter the first value' );
v2= input(' Enter the second value' );
sm=v1+v2;
disp([‘the sum of the two values is ', num2str(sm)])
```

User interaction

Reading : input(‘message’)
Writing : disp(‘message’)

Exercise: Calculate the sum of two integers entered by the user.

Solution 2 :

```
v= input(‘Enter the two values : ', 's');
v = str2num(v) ;
sm=v(1)+v(2);
disp([‘the sum of the two values is ', num2str(sm)])
```

Application

1. Construct the vector $(1, \frac{1}{2}, \frac{1}{3} \ldots\ldots, \frac{1}{1000})$
2. Deduce the vector from it .
3. Use this final vector for calculations $\sum_{k=1}^{1000} \frac{1}{k^2}$

Solution :

1. Créating inverse of number vector from 1 to 1000

```
>> u = 1. / (1:1000) ;
```

2. Élévation square vector element :

```
>> v = u.^2 ;
```

3. function sum(v) sum vector éléments.

```
>>somme = sum( v )
```