Hands On: Introduction to MATLAB - Part 3 - Vectors & Matrices

This is the third MATLAB live script of the collection ***Hands On: Using MATLAB in the 267MI "System Dynamics" course***, devoted to introduce the MATLAB/Simulink environment and tools for solving practical problems related to the topics of the 267MI course, i.e. performance analysis of dynamic systems, parametric estimation, identification of models from data, and prediction of the evolution of dynamic systems.

Use this link to go back to the main live script of the collection.

Table of Contents

Objectives

The aim of this live script is

- to understand how to create and manipulate vectors in MATLAB;
- to understand how to create and manipulate matrices in MATLAB.

Introduction

One of the great powers of mathematics and MATLAB is the ability to work with both with vectors and matrices of virtually any size (2 by 2, 10 by 10, 100 by 100000). Matrices and vectors are essential tools in many applications, describing quantities like the concentrations of different chemicals at different points in time, the prices of assets (stocks, companies, equipment, etc) at different points in time, and the temperature at different points in a body. Most often matrices are associated with a linear system which we want to solve, if possible.

Row vectors

In MATLAB you can create a row vector using square brackets [ ]. Elements of the vector may be separated either by one or more blanks or a comma

```
xr1 = [1 -3 4]
```

```
xr1 = 1×3
     1    -3     4
```

```
xr2 = [1, -3, 4]
```

```
xr2 = 1×3
     1    -3     4
```

Elements of Vectors

To refer to elements in a vector MATLAB uses round brackets ( )

```
xr1(3) % the current value of the last element in xr1
```

```
ans =
     4
```

```

xr1(3) = 0; % This assigns the value 0 to the last element of xr1.
```

Accessing Several Elements of a Vector

The elements of a vector are indexed, starting with 1 continuing to the length of the vector.

MATLAB uses vectors of integers between 1 and the length of a vector to refer to several elements of a vector at once. For example

```
xr1([1 3]) % selecting the 1st and the 3rd element in xr1
```

```
ans = 1×2
     1     0
```

```


IDlist = [1 3 2 3 1];
sampleV = xr2(IDlist) % This produces a vector consisting of the elements of xr2
```

```
sampleV = 1×5
     1     4    -3     4     1
```

```
                       % corresponding to the values in IDlist and assigns the
                       % result to a variable called sampleV.
```

Number of Elements in a Vector

The number of elements in an array (for instance a vector) can be found using the MATLAB command `numel`

```
numel(sampleV)
```

```
ans = 
     5
```

The length of a vector can also be found using the MATLAB command `length`

```
length(sampleV)
```

```
ans = 
     5
```

The Colon Operator

Frequently you want to create vectors whose elements are defined by a simple expression or to access a sequence of elements in a vector. The colon operator : provides a very convenient way of doing this.

In its simplest form

```
x = a:b
```

the first element in the vector x is the a value, then the next element is $a+1$, and so on, untill b is reached

```
x = -12:4
```

```
x = 1×17
   -12   -11   -10    -9    -8    -7    -6    -5    -4    -3    -2    -1     0···
```

A slightly more general form of the colon operator is

```
y = a:step:b
```

which starts at a, then adds `step` repeatedly, until b is reached (or exceeded). In fact the value of `step` does not have to be an integer. It can even be negative.

```
 y1 = 5:2:10
```

```
y1 = 1×3
     5     7     9
```

```
y2 = 1 + 0.05*(0:10)
```

```
y2 = 1×11
   1.000000000000000   1.050000000000000   1.100000000000000   1.150000000000000 ⋯
```

```
y3 = 1:0.05:1.5 % y2 and y3 have the same values
```

```
y3 = 1×11
   1.000000000000000   1.050000000000000   1.100000000000000   1.150000000000000 ⋯
```

```
y4 = 2:-0.05:1.65
```

```
y4 = 1×8
   2.000000000000000   1.950000000000000   1.900000000000000   1.850000000000000 ⋯
```

The linspace Command

The task of creating a vector of equally (or linearly) spaced points between two limits occurs so commonly that MATLAB has a special command to do this:

```
linspace(a, b, n)
```

The command creates n equally spaced values between a and b, including both a and b.

```
z1 = linspace(1, 2, 8) % 7 intervals - why?
```

```
z1 = 1×8
   1.000000000000000   1.142857142857143   1.285714285714286   1.428571428571429 ⋯
```

```
z2 = linspace(1, 2, 9) % 8 intervals
```

```
z2 = 1×9
   1.000000000000000   1.125000000000000   1.250000000000000   1.375000000000000 ⋯
```

Indexing Vectors

The colon operator is very helpful for indexing elements of a vector

```
z2(1:5) % select the first 5 elements of the vector z2
```

```
ans = 1×5
   1.000000000000000   1.125000000000000   1.250000000000000   1.375000000000000 ⋯
```

```
y2(3:2:11) % select the 3rd, 5th, 7th, 9th and 11th elements of the vector y2
```

```
ans = 1×5
   1.100000000000000   1.200000000000000   1.300000000000000   1.400000000000000 ⋯
```

The MATLAB keyword end is used for several purposes. When referring to an element of a vector, it refers to the **last** element

```
y4(end-3:end) % the last 4 element in y4
```

```
ans = 1×4
   1.800000000000000   1.750000000000000   1.700000000000000   1.650000000000000
```

Vector Arithmetic

The standard vector operations of adding two vectors and multiplying a vector by a scalar work in MATLAB.

Addition of Two Vectors

The sum of two vectors of the same size is obtained by adding corresponding elements

```
a = [1 2 3];
b = [10 11 12];
c = a+b
```

```
c = 1×3
    11    13    15
```

Try adding vectors of different sizes

```
a1 = 1:3;
b1 = 10:15;
%a1+b1
```

A Vector Times a Scalar

Multiplying a vector by a scalar produces another vector of the same size in which each element of the original vector has been multiplied by the scalar

```
b2 = 2.5 *b1
```

```
b2 = 1×6
  25.000000000000000  27.500000000000000  30.000000000000000  32.500000000000000 ···
```

Adding a Scalar to a Vector

A scalar may be added to any vector producing a new vector with the scalar added to **each** element of the vector

```
b3 = b2+2.2
```

```
b3 = 1×6
  27.199999999999999  29.699999999999999  32.200000000000003  34.700000000000003 ···
```

Element by Element Operations

Sometimes it is very useful to apply arithmetic operations to each element of a vector (or even a matrix). These element by element operations are

- .* to multiply corresponding elements of two vectors
- ./ to divide corresponding elements of two vectors
- .^ to take powers of each element of a vector

Evaluate

$$x_k = k^2 , \quad k = 1 , 2 , \ \ldots 5$$

```
iv = (1:5);
xa = iv .* iv
```

```
xa = 1×5
     1     4     9    16    25
```

```
xb = iv .^2   % compare xa and xb
```

```
xb = 1×5
     1     4     9    16    25
```

Many MATLAB functions, for example `exp` and `sqrt`, work with vectors, applying the function to each element of the vector.

Column Vectors

In MATLAB you can also create a column vector using square brackets [ ]. However, elements of a column vector are separated either by a semicolon ;   or a newline

```
xv1 = [-1; 2; 3; 0]
```

```
xv1 = 4×1
    -1
     2
     3
     0
```

```

xv2 = [-1
        2
        3
        0]
```

```
xv2 = 4×1
    -1
     2
     3
     0
```

```
% they have the same elements
```

Operations with Column Vectors

Elements of a column vector are accessed using round brackets ( ), exactly the same as for row vectors.

Add Row and Column Vectors

By adding a row vector x and a colum vector y, you will obtain a matrix with each element (i, j) in the matrix equal to $x(i) + y(j)$

```
rv = [1 2 3];
cv = [4; 5; 6;7];
z = rv+cv
```

```
z = 4×3
     5     6     7
     6     7     8
     7     8     9
     8     9    10
```

Transpose of a Vector

You can convert a row vector into a column vector (and vice versa) using the transpose operator ' (an apostrophe)

```
x_c = [1 2 3 4]'
```

```
x_c = 4×1
     1
     2
     3
     4
```

```


y_r = [4; 3; 2; 1]'
```

```
y_r = 1×4
     4     3     2     1
```

Creating a Matrix

In MATLAB you can create a matrix using square brackets [ ]. Elements of a row are separated either by one or more blanks or a comma ,. Rows are separated by a semicolon ; or a newline.

A matrix must have the same number of elements in each row and the same number of elements in each column, thus an m by n matrix is a array of m rows each of n elements or equivalently n columns each with m elements.

```
A = [ 1 2 3 4; 5 6 7 8; 0 9 8 7]
```

```
A = 3×4
     1     2     3     4
     5     6     7     8
     0     9     8     7
```

Joining Matrices

You can join matrices together using square brackets [ ], provided the sizes match.

```
r1 = 1:3;
r2 = 4:6;
A = [r1; r2] % two row vectors, stacked on top of each other
```

```
A = 2×3
     1     2     3
     4     5     6
```

```
c1 = [1; 4]
```

```
c1 = 2×1
     1
     4
```

```
c2 = [2; 5]
```

```
c2 = 2×1
     2
     5
```

```
c3 = [3 6]'
```

```
c3 = 2×1
     3
     6
```

```
A = [c1 c2 c3] % The matrix A has three columns joined side by side.
```

```
A = 2×3
     1     2     3
     4     5     6
```

```
A = [1 2 3; 4 5 6]
```

```
A = 2×3
     1     2     3
     4     5     6
```

```
B = [10 11 12; 13 14 15]
```

```
B = 2×3
    10    11    12
    13    14    15
```

```
C = [A B] % As A and B have the same number of rows they may be joined side by side. C
```

```
C = 2×6
     1     2     3    10    11    12
     4     5     6    13    14    15
```

```

D = [A; B] % As A and B have the same number of columns they may be stacked. D is a 4
```

```
D = 4×3
     1     2     3
     4     5     6
    10    11    12
    13    14    15
```

Fast Ways to Create a Matrix

MATLAB provides efficient functions to create some commonly used matrices:

- The command `zeros(m,n)` creates an m by n array (matrix) of zeros.
- The command `ones(m,n)` creates an m by n array (matrix) of ones.
- The command `eye(n,n)` creates an n by n identity matrix.
- The command `rand(m,n)` creates an m by n matrix whose elements are random numbers, uniformly distributed between 0 and 1.
- The command `randn(m,n)` creates an m by n matrix whose elements are random numbers, gaussianly distributed with mean value 0 and variance 1.

```
A1 = zeros(5, 3)
```

```
A1 = 5×3
     0     0     0
     0     0     0
     0     0     0
     0     0     0
     0     0     0
```

```
A2 = ones(3, 5)
```

```
A2 = 3×5
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
```

```
I4 = eye(4) % compact form, equivalent to eye(4,4)
```

```
I4 = 4×4
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

```
R1 = rand(3,4)
```

```
R1 = 3×4
   0.709364830858073   0.679702676853675   0.118997681558377   0.340385726666133
   0.754686681982361   0.655098003973841   0.498364051982143   0.585267750979777
   0.276025076998578   0.162611735194631   0.959743958516081   0.223811939491137
```

```
G1 = randn(3,3)
```

```
G1 = 3×3
   1.117356138814467   0.552527021112224   0.085931133175425
  -1.089064295052236   1.100610217880866  -1.491590310637609
   0.032557464164973   1.544211895503951  -0.742301837259857
```

Size of a Matrix

The dimensions (number of rows, number of columns) of a matrix can be found using the MATLAB command `size`

```
size(I4)
```

```
ans = 1×2
```

```
     4     4
```

```
size(R1)
```

```
ans = 1×2
     3     4
```

Elements of a Matrix - Indexing

In MATLAB `A(i,j)` accesses the element $A_{i,j}$ in row i, column j of the matrix A

```
A = zeros(3); % compact form of zeros(3, 3)
A(1,3) = 3;
A(3,1) = -3;
A
```

```
A = 3×3
     0     0     3
     0     0     0
    -3     0     0
```

Rows/Columns of a Matrix

You can access a row or a column of a matrix using the colon : operator to refer to all of a row or all of a column

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

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

```
r2 = A(2, :)
```

```
r2 = 1×3
     4     5     6
```

```
c3 = A(:, 3)
```

```
c3 = 3×1
     3
     6
     9
```

Several Elements

The colon : referring to the whole row or column, can be replaced by a vector of indices

```
u = A(1, [1 3])
```

```
u = 1×2
     1     3
```

```
v = A([1 2], 1)
```

```
v = 2×1
     1
     4
```

Sub-matrices

If you refer to more than one row and more than one column of a matrix, then you get a sub-matrix consisting of all the elements in those rows and columns

```
B1 = A(1:2, [1 3])
```

```
B1 = 2×2
     1     3
     4     6
```

```
B2 = A(:, [1 3])
```

```
B2 = 3×2
     1     3
     4     6
     7     9
```

Matrix Arithmetic

Addition and subtraction are defined for matrices of the same dimensions, and work elementwise.

Multiplication of a matrix by a scalar is also defined elementwise, just as for vectors.

```
A = [1 2 3 0; 3 4 5 -1; 5 6 7 8]
```

```
A = 3×4
     1     2     3     0
     3     4     5    -1
     5     6     7     8
```

```
B = -2*A
```

```
B = 3×4
    -2    -4    -6     0
    -6    -8   -10     2
   -10   -12   -14   -16
```

```
C = 2*A + B
```

```
C = 3×4
     0     0     0     0
     0     0     0     0
     0     0     0     0
```

Matrix Multiplication

In MATLAB the multiplication operator * represents **matrix multiplication**.

If A and B are not scalars, then A*B is **only** defined if the number of columns in A is equal to the number of rows in B.

```
A = [1 2; 3 4; 5 6]
```

```
A = 3×2
     1     2
     3     4
     5     6
```

```
B = [5 6; 7 8]
```

```
B = 2×2
     5     6
     7     8
```

```
C = A*B
```

```
C = 3×2
    19    22
    43    50
    67    78
```

Remember: matrix multiplication is **not commutative**.

Connections Between Vectors and Matrices

A **row** vector with n elements is equivalent to a 1 by n matrix.

A **column** vector with m elements is equivalent to a m by 1 matrix.

```
A = [1 2 3 0; 3 4 5 -1; 5 6 7 8]
```

```
A = 3×4
     1     2     3     0
     3     4     5    -1
     5     6     7     8
```

```
b = [1;2;3;4]
```

```
b = 4×1
     1
     2
     3
     4
```

```
v = A*b
```

```
v = 3×1
    14
    22
    70
```

```

r = [3 2 1]
```

```
r = 1×3
     3     2     1
```

```
w = r*A
```

```
w = 1×4
    14    20    26     6
```

```
x = rand(3,1)
```

```
x = 3×1
   0.257508254123736
   0.840717255983663
   0.254282178971531
```

```
y = rand(3,1)
```

```
y = 3×1
   0.814284826068816
   0.243524968724989
   0.929263623187228
```

```
p = x'*y % the dot product of x and y
```

```
p =
   0.650715886333440
```

Matrix Powers

Just as * represents matrix multiplication, ^ represents the multiplication of matrices together

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

```
A = 2×2
     1     2
     3     4
```

```
B = A^2
```

```
B = 2×2
     7    10
    15    22
```

```
C = A^3
```

```
C = 2×2
    37    54
    81   118
```

Elementwise Operations

MATLAB provides the operators .* for element by element multiplication, ./ for element by element division and .^ for element by element powers. This works is the same way as with vectors.

```
A = [1 2 3; 4 5 6]
```

```
A = 2×3
     1     2     3
     4     5     6
```

```
B = 1./A
```

```
B = 2×3
   1.000000000000000   0.500000000000000   0.333333333333333
   0.250000000000000   0.200000000000000   0.166666666666667
```

Matrix Manipulation Functions

```
help elmat
```

```
  Elementary matrices and matrix manipulation.

  Elementary matrices.
    zeros       - Zeros array.
    ones        - Ones array.
    eye         - Identity matrix.
    repmat      - Replicate and tile array.
    repelem     - Replicate elements of an array.
    linspace    - Linearly spaced vector.
    logspace    - Logarithmically spaced vector.
    freqspace   - Frequency spacing for frequency response.
    meshgrid    - X and Y arrays for 3-D plots.
    accumarray  - Construct an array with accumulation.
    :           - Regularly spaced vector and index into matrix.

  Basic array information.
    size        - Size of array.
    length      - Length of vector.
    ndims       - Number of dimensions.
    numel       - Number of elements.
    disp        - Display matrix or text.
    isempty     - True for empty array.
    isequal     - True if arrays are numerically equal.
    isequaln    - True if arrays are numerically equal, treating NaNs as equal.
    height      - Number of rows.
    width       - Number of columns.

  Matrix manipulation.
    cat         - Concatenate arrays.
    reshape     - Reshape array.
    diag        - Diagonal matrices and diagonals of matrix.
    blkdiag     - Block diagonal concatenation.
    tril        - Extract lower triangular part.
    triu        - Extract upper triangular part.
    fliplr      - Flip matrix in left/right direction.
    flipud      - Flip matrix in up/down direction.
    flip        - Flip the order of elements.
    rot90       - Rotate matrix 90 degrees.
    :           - Regularly spaced vector and index into matrix.
    find        - Find indices of nonzero elements.
    end         - Last index.
    sub2ind     - Linear index from multiple subscripts.
    ind2sub     - Multiple subscripts from linear index.
    bsxfun      - Binary singleton expansion function.

  Multi-dimensional array functions.
```

```
      ndgrid      - Generate arrays for N-D functions and interpolation.
      permute     - Permute array dimensions.
      ipermute    - Inverse permute array dimensions.
      shiftdim    - Shift dimensions.
      circshift   - Shift array circularly.
      squeeze     - Remove singleton dimensions.

    Array utility functions.
      isscalar    - True for scalar.
      isvector    - True for vector.
      isrow       - True for row vector.
      iscolumn    - True for column vector.
      ismatrix    - True for matrix.

    Special variables and constants.
      eps         - Floating point relative accuracy.
      realmax     - Largest positive floating point number.
      realmin     - Smallest positive floating point number.
      intmax      - Largest positive integer value.
      intmin      - Smallest integer value.
      flintmax    - Largest consecutive integer in floating point format.
      pi          - 3.1415926535897....
      i           - Imaginary unit.
      inf         - Infinity.
      nan         - Not-a-Number.
      isnan       - True for Not-a-Number.
      isinf       - True for infinite elements.
      isfinite    - True for finite elements.
      j           - Imaginary unit.
      true        - True array.
      false       - False array.

    Specialized matrices.
      compan      - Companion matrix.
      gallery     - Test matrices.
      hadamard    - Hadamard matrix.
      hankel      - Hankel matrix.
      hilb        - Hilbert matrix.
      invhilb     - Inverse Hilbert matrix.
      magic       - Magic square.
      pascal      - Pascal matrix.
      rosser      - Classic symmetric eigenvalue test problem.
      toeplitz    - Toeplitz matrix.
      vander      - Vandermonde matrix.
      wilkinson   - Wilkinson's eigenvalue test matrix.

    Controlling multithreading setting.
      maxNumCompThreads - Controls the maximum number of computational threads.
```

Summary

Using this live script you have:

- learnt how to use square brackets [ ] to create row or column vectors;
- learnt how to access elements of a vector and to find the number of elements in a vector;
- learnt about fast ways to create vectors using the : operator;
- learnt about indexing to refer to some elements of a vector;

- learnt about vector arithmetic, both in the usual mathematical sense and element by element operations;
- learnt about the transpose operator ';
- learnt how to use square brackets [ ] to create matrices;
- learnt about fast ways to create matrices using `zeros`, `ones`, `rand` and `eye`;
- learnt about ways access elements, rows, columns or sub-matrices of a matrix;
- learnt about matrix arithmetic;
- learnt about elementary and specialised matrix functions.

Back to the Index

Use this link to go back to the main live script of the collection.

Back to the Previous Part: Numbers, Arithmetic Operations, Formats and Variables

Use this link to go back to the previous live script of this collection.

Go to the Next Part: Functions

Use this link to go to the next live script of the collection.