

Matrices II (plus): Objectives

By the end of this class you should be able to:

- Add interactive input to a script
- Explain and carry out vector and matrix multiplication
- Create multidimensional Arrays
- Find the zero of a single input function
- Minimize a single input function
- Begin preparing for Exam 1

Book Sections 2.4, (1.4), 3.2

quiz

Use a MATLAB script file to:

1. Create the following matrix in MATLAB

$$M = \begin{bmatrix} 4 & 1 & 8 \\ -3 & 6 & 2 \end{bmatrix}$$

2. Carry out the following calculation using element-by-element operations:

$$\frac{(M)}{(1+M)}$$

Print out a solution including both the script and its execution in your final answer. Please provide clear labeling.

Interactive Script

- Download file cyl1.m from website
- Save to your current MATLAB work directory
- Run `>> cyl1`
- Respond to questions (before we used $r = 1.3$ m , $h = 2$ m)
- look at m-file in editor - how does it work?

Interactive Input in a Script

- The input function:

`q = input('question in quotation marks ')`

The input command will ask a question and then wait for user input. Once user types a number and enter the number will be saved in the variable on the left-hand side of the assignment operator (=).

- The disp function:

`disp('text you want displayed')`

The disp command will display to the user text in single quotes given as an argument.

- Displaying Results:

Results can be displayed simply by leaving the ; off the final calculations

Vector Multiplication

Example:

$$u \cdot v = [4 \ 3] \cdot \begin{bmatrix} 5 \\ 1 \end{bmatrix} = (4 * 5) + (3 * 1) = 23$$

General Form:

$$u \cdot v = [u_1 \ u_2 \ \dots \ u_n] \cdot \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = (u_1 v_1 + u_2 v_2 + \dots + u_n v_n)$$

Vector Multiplication trial result

$$u \cdot v = [3 \ 1 \ 7] \cdot \begin{bmatrix} 4 \\ 6 \\ 5 \end{bmatrix} = ?$$

By Hand

$$\begin{aligned} u \cdot v &= 3(4) + 6(1) + 7(5) \\ &= 12 + 6 + 35 \\ &= 53 \end{aligned}$$

In MATLAB

$$\begin{aligned} >> [3 \ 1 \ 7]*[4 \ 6 \ 5]' \\ \text{ans} &= \\ &53 \end{aligned}$$

Can use this to calculate the magnitude of a vector.

For example

```
>> u = [3 5 6 2]
```

```
>> sqrt(u*u')
```

Yields the magnitude of this vector (the square root of the sum of squares of the elements).

Matrix Multiplication

$$U * V = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \end{bmatrix} \cdot \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{bmatrix} = \begin{bmatrix} u_{11}v_{11} + u_{12}v_{21} & u_{11}v_{12} + u_{12}v_{22} & \dots \\ u_{21}v_{11} + u_{22}v_{21} & \dots & \dots \\ \dots & \dots & u_{31}v_{13} + u_{32}v_{23} \end{bmatrix}$$

$$[m \times n] [n \times p] = [m \times p]$$

$$\begin{bmatrix} 6 & -2 \\ 10 & 3 \\ 4 & 7 \end{bmatrix} \cdot \begin{bmatrix} 9 & 8 \\ -5 & 12 \end{bmatrix} = \begin{bmatrix} (6)(9) + (-2)(-5) = 64 & 24 \\ (10)(9) + (3)(-5) = 75 & 116 \\ 1 & 116 \end{bmatrix}$$

Matrix Multiplication

$$U * V = \begin{bmatrix} u_{11} & u_{12} \\ u_{21} & u_{22} \\ u_{31} & u_{32} \end{bmatrix} \cdot \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{bmatrix} = \begin{bmatrix} u_{11}v_{11} + u_{12}v_{21} & u_{11}v_{12} + u_{12}v_{22} & \dots \\ u_{21}v_{11} + u_{22}v_{21} & \dots & \dots \\ \dots & \dots & u_{31}v_{13} + u_{32}v_{23} \end{bmatrix}$$

$$[m \times n] [n \times p] = [m \times p]$$

$$\begin{bmatrix} 6 & -2 \\ 10 & 3 \\ 4 & 7 \end{bmatrix} \cdot \begin{bmatrix} 9 & 8 \\ -5 & 12 \end{bmatrix} = \begin{bmatrix} (6)(9) + (-2)(-5) = 64 & 24 \\ (10)(9) + (3)(-5) = 75 & 116 \\ 1 & 116 \end{bmatrix}$$

Matrix Multiplication Trial

$$\begin{bmatrix} 4 & 3 \\ -1 & 6 \end{bmatrix} \cdot \begin{bmatrix} 5 & 1 \\ 3 & 7 \end{bmatrix} = ?$$

Solution by Hand: $\begin{bmatrix} 4(5) + 3(3) & 4(1) + 3(7) \\ -1(5) + 6(3) & -1(1) + 6(7) \end{bmatrix} = \begin{bmatrix} 29 & 25 \\ 13 & 41 \end{bmatrix}$

Solution in MATLAB:

```
>> u=[4 3;-1 6];v=[5 1; 3 7];
>> u*v
ans =
    29    25
    13    41
```

Note: with two square matrices (as in this case) it is legal to do either matrix operations or element-by-element operations (the `.` operators) with very different answers. Be careful to choose the correct one

Examples using Matrix Multiplication

Calculate Magnitude of a vector

```
>> v = [3 4 8 5]
>> mag = sqrt(v*v')
```

Solve a system of linear equations

$$\begin{aligned} x + 5y &= 4 \\ 3x + y &= 2 \end{aligned}$$

$$\begin{bmatrix} 1 & 5 \\ 3 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

Compare to

```
>> sqrt(sum(v.^2))
```

$$Ax = B$$

$$X = A \setminus B$$

Multi-Dimensional Matrices (Arrays)

- Can have more than two dimensions
- Add more indices
- Build up from lower dimension arrays
- Use indexing e.g.

```
>> A(:, :, 2) = B or
```

- Concatenation Function: `cat(dimension of assembly, array 1, array 2)`

Creating Multidimensional Arrays (Examples)

- using `x` & `y` from the array practice sheet (see handout)
 - These are both 3×3 arrays
- Creating a Multidimensional Array using indexing
 - `>> Q = x` copy `x` so I do not overwrite it

```
>> Q(:, :, 2) = y place y in a second page
```

This creates a $3 \times 3 \times 2$ matrix (3 rows, 3 columns, 2 pages). Notice the use of the `:` to represent all rows and all columns.

We could continue adding pages to this array, for example we could add another page as follows:

```
>> Q(:, :, 3) = [5, 3, 6; 2, -9, 4; 1, 3, -2]
```

Notice that all the pages must have the same number of rows and columns

- We can also make use of the cat function (concatenation)
cat(dimension of assembly, array 1, array 2)
- Lets look at how this works try each of the following
 - >> cat(1, x, y)
 - >> cat(2, x, y)
 - >> cat(3, x, y)
 Notice that in
 - the first case the y array is added as new rows (the 1st dimension),
 - in the second case it is added as new columns (the 2nd dimension)
 - & in the third case it is added as a new page (the 3rd dimension).
 This last case is a multidimensional array (3 dimensional).
- Sometimes we want to address the last indexed value in a row, column or page but do not know what it is. This can be accomplished by using "end" instead of an index. For example:
 - >> x(3, 2:end)
 will yield the third row of the x matrix from the second to the last column.



Extrema and Zeros of Functions

- fzero('name', estimate)
- fminbnd('name(vbl)', min, max)
- fminsearch('name', estimate)

fzero → finding the point where a functions output sign changes
 >> [x, fval, exitflag] = fzero('function', x0)

- 'function' → the name of the single variable function to minimize
- x0 = initial guess as to approximately where the zero will be.
- x = the input value where function result crosses zero,
- fval = function value at the point (will be close to but ≠ to zero)
- exitflag → indicates if zero was found

Try:

```
>> fzero('sin(x)', 3)
>> [x, y, flag]=fzero('sin(x)', 3), also try other 'x0's (e.g. 6)
>> [x, y, flag]=fzero('sec(x)', 3)
```

Notice: - finds the zero at π for all cases
 - for sec the function output flips from $+\infty$ to $-\infty$ (w/out crossing 0)
 - y value is within rounding error of 0
 - other zeros are found when x0 is adjusted

fminsearch → function minimum search
 >> [x, fval, exitflag] = fminsearch('function', x0)

- 'function' → the name of the single variable function to minimize
- x0 = initial guess as to approximately where the minimum will be.
- x = the value where function result is a local minimum
- fval = function value at the x point
- exitflag → indicates if minimum was found

Try:

```
>> fminsearch('sin(x)', 3)
>> [x, y, flag]=fminsearch('sin', 6) also try other values
>> [x, y, flag]=fminsearch('sec(x)', 0) also try 2
```

Notice: again function finds a local minimum. for a search of sec starting at 2 no answer is found because the function continues to $-\infty$

All of these functions can be used with user defined single input functions.

fminbnd → function minimum bounded
 >> [x, fval, exitflag] = fminbnd('function', xlow, xhigh)

- 'function' → the name of the single variable function to minimize
- xlow → lower bound of the possible range of x values
- xhigh → upper bound of the possible range of x values
- x = the value where tested function has the a local minimum within the range specified range
- fval = function value at the point
- exitflag → indicates if zero was found

e.g. Try:

```
>> fminbnd('sin(x)', 0, 6)
>> [x, y, flag] = fminbnd('sin(x)', 0, 12),
>> [x, y, flag] = fminbnd('sin(x)', 6, 12) also try other limits
```

Notice: finds only one minimum - there may be more, may be greater mins (global min). Computer routines suffer from this limit of finding only a local minimum, not guaranteeing that it is global min.