SECTION 2: VECTORS AND MATRICES

ENGR 112 – Introduction to Engineering Computing

Vectors and Matrices

The "MAT" in MATLAB

- **MATLAB**
  - The ***MAT****rix* (not *MAThematics*) ***LAB****oratory*
- MATLAB assumes all numeric variables are ***matrices***
  - ***Vectors*** and ***scalars*** are special cases of matrices
- This section of notes will introduce concept of vectors and matrices
  - Matrix math – linear algebra fundamentals
  - You'll cover this in much more detail in your Linear Algebra course

Matrices

- **Matrix**
  - Array of numerical values, e.g.:

$$\mathbf{A} = \begin{bmatrix} -7 & 0 & 1 & 4 \\ 4 & -2 & 9 & 5 \\ 8 & 3 & 4 & 0 \end{bmatrix}$$

  - The variable, $\mathbf{A}$, is a ***matrix***
- An $m \times n$ matrix has m ***rows*** and n ***columns***
- These are the ***dimensions*** of the matrix
  - $\mathbf{A}$ is a 3×4 matrix

Matrix Dimensions and Indexing

- An $m \times n$ matrix:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- Use indices to refer to individual elements of a matrix
  - a_{ij}: the element of $\mathbf{A}$ in the i^{th} row and the j^{th} column

Vectors

- ***Vectors***
  - A matrix with one dimension equal to one
  - A matrix with ***one row*** or ***one column***
- ***Row vector***
  - One row – a $1 \times n$ matrix, e.g.:

$$x = \begin{bmatrix} -9 & 1 & -4 \end{bmatrix}$$

  - A 1×3 row vector
- ***Column vector***
  - One column – an $m \times 1$ matrix, e.g.:

$$x = \begin{bmatrix} 5 \\ 1 \\ 8 \end{bmatrix}$$

  - A 3×1 column vector

Scalars

- **<u>Scalar</u>**
  - A 1×1 matrix
  - The numbers we are we are familiar with, e.g.:

$$b = 4, \qquad x = -3 + j5.8, \qquad y = -1 \times 10^{-9}$$

- We understand simple mathematical operations involving scalars
  - Can add, subtract, multiply, or divide any pair of scalars
  - Not true for matrices
    - Depends on the matrix dimensions

Mathematical Matrix Operations

Matrix Addition and Subtraction

- As long as matrices have the ***same dimensions***, we can add or subtract them
  - ***Addition*** and ***subtraction*** are done ***element-by-element***, and the ***resulting matrix is the same size***

$$\begin{bmatrix} 4 & 8 \\ 0 & 3 \end{bmatrix} + \begin{bmatrix} 1 & -4 \\ 6 & -1 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 6 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 8 \\ 0 & 3 \end{bmatrix} - \begin{bmatrix} 1 & -4 \\ 6 & -1 \end{bmatrix} = \begin{bmatrix} 3 & 12 \\ -6 & 4 \end{bmatrix}$$

- We can also add ***scalars*** to (or subtract from) matrices

$$\begin{bmatrix} 1 & -4 \\ 6 & -1 \end{bmatrix} + 5 = \begin{bmatrix} 6 & 1 \\ 11 & 4 \end{bmatrix}$$

Matrix Addition and Subtraction

- If matrices are not the same size, and neither is a scalar, addition/subtraction are not defined
  - The following operations cannot be done

$$\begin{bmatrix} 4 & 8 \\ 0 & 3 \end{bmatrix} + \begin{bmatrix} 1 & -4 & 6 \\ 6 & -1 & 9 \end{bmatrix} = ?$$

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

- Addition is commutative (order does not matter):

$$\mathbf{A} + \mathbf{B} = \mathbf{B} + \mathbf{A} = \mathbf{C}$$

$$\begin{bmatrix} 4 & 8 \\ 0 & 3 \end{bmatrix} + \begin{bmatrix} 1 & -4 \\ 6 & -1 \end{bmatrix} = \begin{bmatrix} 1 & -4 \\ 6 & -1 \end{bmatrix} + \begin{bmatrix} 4 & 8 \\ 0 & 3 \end{bmatrix} = \begin{bmatrix} 5 & 4 \\ 6 & 2 \end{bmatrix}$$

Matrix Multiplication

- In order to multiply matrices, their ***inner dimensions*** must agree
- We can multiply $\mathbf{A} \cdot \mathbf{B}$ only if the ***number of columns*** of $\mathbf{A}$ is equal to the ***number of rows*** of $\mathbf{B}$
- Resulting Matrix has same number of rows as $\mathbf{A}$ and same number of columns as $\mathbf{B}$

$$\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$$

$$(m \times n) \cdot (n \times p) = (m \times p)$$

Matrix Multiplication – $\mathbf{A} \cdot \mathbf{B} = \mathbf{C}$

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & \cdots & b_{1p} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{np} \end{bmatrix} = \begin{bmatrix} c_{11} & \cdots & c_{1p} \\ \vdots & \ddots & \vdots \\ c_{m1} & \cdots & c_{mp} \end{bmatrix}$$

- The (i, j^{th}) entry of $\mathbf{C}$ is the ***dot product*** of the i^{th} row of $\mathbf{A}$ with the j^{th} column of $\mathbf{B}$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

- Consider the multiplication of two 2×2 matrices:

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Matrix Multiplication – Examples

- A 2×2 and a 2×3 yield a 2×3

$$\begin{bmatrix} 1 & 4 \\ 2 & 1 \end{bmatrix} \cdot \begin{bmatrix} 3 & -1 & 5 \\ 6 & 2 & 0 \end{bmatrix} = \begin{bmatrix} 27 & 7 & 5 \\ 12 & 0 & 10 \end{bmatrix}$$

- A 3×3 and a 3×1 result in a 3×1

$$\begin{bmatrix} 1 & 5 & 0 \\ 0 & 4 & 8 \\ 2 & 7 & 3 \end{bmatrix} \cdot \begin{bmatrix} 6 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 11 \\ 20 \\ 25 \end{bmatrix}$$

Matrix Multiplication – Properties

- ***Matrix multiplication is not commutative***
  - Order matters
  - Unlike scalars
- In general,

$$\mathbf{A} \cdot \mathbf{B} \neq \mathbf{B} \cdot \mathbf{A}$$

- If A and/or B is not square then one of the above operations may not be possible anyway
  - Inner dimensions may not agree for both product orders

Matrix Multiplication – Properties

- ***Matrix multiplication is associative***
  - Insertion of parentheses anywhere within a product of multiple terms does not affect the result:

$$(\mathbf{A}\cdot\mathbf{B})\cdot\mathbf{C}=\mathbf{D}$$

$$\mathbf{A}\cdot(\mathbf{B}\cdot\mathbf{C})=\mathbf{D}$$

- ***Matrix multiplication is distributive***
  - Multiplication distributes over addition
  - Must maintain correct order, i.e. left- or right-multiplication

$$\mathbf{A}(\mathbf{B}+\mathbf{C})=\mathbf{AB}+\mathbf{AC}$$

$$(\mathbf{B}+\mathbf{C})\mathbf{A}=\mathbf{BA}+\mathbf{CA}$$

Identity Matrix

- Multiplication of a scalar by 1 results in that scalar

$$a \cdot 1 = 1 \cdot a = a$$

- The matrix version of 1 is the ***identity matrix***
  - Ones along the diagonal, zeros everywhere else
  - Square $(n \times n)$ matrix
  - Denoted as $\mathbf{I}$ or $\mathbf{I_n}$, where $\mathbf{n}$ is the matrix dimension, e.g.

$$\mathbf{I_3} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Left- or right-multiplication by an identity matrix results in that matrix, unchanged

$$\mathbf{A} \cdot \mathbf{I} = \mathbf{I} \cdot \mathbf{A} = \mathbf{A}$$

Identity Matrix

- Right-multiplication of an $n \times n$ matrix by an $n \times n$ identity matrix, $\mathbf{I_n}$

$$\begin{bmatrix} 1 & 5 & 0 \\ 0 & 4 & 8 \\ 2 & 7 & 3 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 0 \\ 0 & 4 & 8 \\ 2 & 7 & 3 \end{bmatrix}$$

- Same result if we left-multiply by $\mathbf{I_n}$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 5 & 0 \\ 0 & 4 & 8 \\ 2 & 7 & 3 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 0 \\ 0 & 4 & 8 \\ 2 & 7 & 3 \end{bmatrix}$$

Identity Matrix

- Right-multiplication of an $m \times n$ matrix by an $n \times n$ identity matrix

$$\begin{bmatrix} 1 & 5 & 0 \\ 0 & 4 & 8 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 0 \\ 0 & 4 & 8 \end{bmatrix}$$

- Same result if we left-multiply the $m \times n$ matrix by an $m \times m$ identity matrix

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 5 & 0 \\ 0 & 4 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 5 & 0 \\ 0 & 4 & 8 \end{bmatrix}$$

Vector Multiplication

- Vectors *are* matrices, so inner dimensions must agree
- Two types of vector multiplication:
- ***Inner product*** (***dot product***)
  - Result is a scalar

$$\begin{bmatrix} a_{11} & a_{12} \end{bmatrix} \cdot \begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix} = a_{11}b_{11} + a_{12}b_{21}$$

- ***Outer product***
  - Result for n-vectors is an n x n matrix

$$\begin{bmatrix} a_{11} \\ a_{21} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} \\ a_{21}b_{11} & a_{21}b_{12} \end{bmatrix}$$

Exponentiation

- As with scalars, raising a matrix to the power, n, is the multiplication of that matrix by itself n times

$$\mathbf{A}^3 = \mathbf{A} \cdot \mathbf{A} \cdot \mathbf{A}$$

- What must be true of a matrix for exponentiation to be allowable?
  - Inner matrix dimensions must agree
  - Rows of $\mathbf{A}$ must equal columns of $\mathbf{A}$ – n x n
  - ***Matrix must be square***

Matrix 'Division' – Multiplication by the Inverse

- Scalar division that we are accustomed to can be thought of as multiplication by an inverse:

$$a \div b = a \cdot \frac{1}{b} = a \cdot b^{-1}$$

- This is how we 'divide' matrices as well

$$\mathbf{A} \cdot \mathbf{B} \cdot \mathbf{B}^{-1} = \mathbf{A}$$

- Multiplication of a scalar by its inverse is equal to 1.
  - For a matrix, the result is the ***identity matrix***

$$\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I} = \begin{bmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{bmatrix}$$

Matrix Inverse

- Recall that matrix multiplication is not commutative
  - ***Right-*** and ***left-multiplication*** are different operations

$$\mathbf{A}\cdot\mathbf{B}\cdot\mathbf{B}^{-1}=\mathbf{A}\neq\mathbf{B}^{-1}\cdot\mathbf{A}\cdot\mathbf{B}$$

- The inverse does not exist for all matrices
  - ***Non-invertible*** matrices are referred to as ***singular***
  - Matrix must be ***square*** for its inverse to exist

Matrix Inverse

- Possible to calculate matrix inverses by hand
  - Simple for small matrices
  - Quickly becomes tedious as matrices get larger
- For example, the inverse of a 2×2 matrix:

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^{-1} = \frac{1}{ad - bc} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$$

- For example:

$$\mathbf{A} = \begin{bmatrix} 2 & 5 \\ 2 & 4 \end{bmatrix}$$

$$\mathbf{A}^{-1} = \frac{1}{8 - 10} \begin{bmatrix} 4 & -5 \\ -2 & 2 \end{bmatrix} = \begin{bmatrix} -2 & 2.5 \\ 1 & -1 \end{bmatrix}$$

Matrix Inverse - Example

- Multiplication of a matrix by its inverse yields the identity matrix
- For example:

$$\mathbf{A}\cdot\mathbf{A}^{-1}=\begin{bmatrix}2 & 5\\ 2 & 4\end{bmatrix}\cdot\begin{bmatrix}-2 & 2.5\\ 1 & -1\end{bmatrix}=\begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}$$

- Or, for a 3×3:

$$\mathbf{A}=\begin{bmatrix}2 & 0 & 2\\ 0 & 1 & 2\\ 0 & 0 & 2\end{bmatrix},\quad \mathbf{A}^{-1}=\begin{bmatrix}0.5 & 0 & -0.5\\ 0 & 1 & -1\\ 0 & 0 & 0.5\end{bmatrix}$$

$$\begin{bmatrix}2 & 0 & 2\\ 0 & 1 & 2\\ 0 & 0 & 2\end{bmatrix}\cdot\begin{bmatrix}0.5 & 0 & -0.5\\ 0 & 1 & -1\\ 0 & 0 & 0.5\end{bmatrix}=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix}$$

- You’ll learn more about this in Linear Algebra – not critical here

Matrix Transpose

- The ***transpose*** of a matrix is that matrix with ***rows and columns swapped***
  - First row becomes the first column, second row becomes the second column, and so on
- For example:

$$\mathbf{A} = \begin{bmatrix} 0 & 9 \\ 2 & 7 \\ 6 & 3 \end{bmatrix} \quad \mathbf{A}^{\mathbf{T}} = \begin{bmatrix} 0 & 2 & 6 \\ 9 & 7 & 3 \end{bmatrix}$$

- Row vectors become column vectors and vice versa

$$\mathbf{x} = \begin{bmatrix} 7 \\ -1 \\ -4 \end{bmatrix} \qquad \mathbf{x}^{\mathbf{T}} = \begin{bmatrix} 7 & -1 & -4 \end{bmatrix}$$

Why Do We Use Matrices?

- Vectors and matrices are used extensively in many engineering fields, for example:
  - Modeling, analysis, and design of dynamic systems
  - Controls engineering
  - Image processing
  - Etc. …
- Very common usage of vectors and matrices is to represent ***systems of equations***
  - These regularly occur in *all* fields of engineering

Systems of Equations

- Consider a system of three equations with three unknowns:

$$3x_1 + 5x_2 - 9x_3 = 6$$
$$-3x_1 + 7x_3 = -2$$
$$-x_2 + 4x_3 = 8$$

- Can represent this in ***matrix form***:

$$\begin{bmatrix} 3 & 5 & -9 \\ -3 & 0 & 7 \\ 0 & -1 & 4 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 6 \\ -2 \\ 8 \end{bmatrix}$$

- Or, more compactly as:

$$\mathbf{Ax} = \mathbf{b}$$

- Perform algebra operations as we would if $\mathbf{A}$, $\mathbf{x}$, and $\mathbf{b}$ were scalars
  - Observing matrix-specific rules, e.g. multiplication order, etc.

Vectors & Matrices in MATLAB

Defining Vectors and Matrices in MATLAB

- Let's say we want to assign the following matrix variable in MATLAB:

$$\mathbf{A} = \begin{bmatrix} 2 & 5 & 1 \\ -4 & 6 & 0 \end{bmatrix}$$

  - Enclose matrices in ***square brackets***
  - Elements on the same row are separated by ***spaces*** or ***commas***
  - Rows are separated by ***semicolons***

- In MATLAB:

```
A = [2, 5, 1; -4, 6, 0];
```

or

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

Ellipsis – Continuation Operator

- An ***ellipsis*** can be used as a ***continuation operator***
  - Tells MATLAB that a single command continues on the next line
- Improves readability
  - Long expressions
  - Large matrices

Command Window

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

A =

     1     3    -4     6
    -9     0     2    -7
     3    -1     5     4
    -2    -1     0     3
     6     8     7     1

>> 
```

Vector and Matrix Generation

- Often want to automatically generate vectors and matrices without having to enter them element-by-element

- A few of MATLAB's ***array-generation*** functions:
  - Colon operator (:)
  - linspace(…)
  - ones(…)
  - zeros(…)
  - diag(…)
  - eye(…)

Vector Generation – Colon operator

- Create vectors of evenly-spaced values using the ***colon*** (:) ***operator***

```
x = xstart:xstep:xstop;
```

  - `xstart`: value of the first element in the vector
  - `xstep`: *optional* increment value – default: `xstep` = `1`
  - `xstop`: maximum value of vector entries
  - `x`: vector of points that is created

- Number of elements in the vector:

$$N = \mathrm{floor}\left(\frac{(x_{stop} - x_{start})}{x_{step}}\right) + 1$$

- Value of the last element in the vector is

$$x_{last} = x_{start} + (N-1) \cdot x_{step}$$

Vector Generation – Colon operator

- Default increment is 1
- Can specify increment value
- Vector values will not exceed the stop value
  - May not include stop value
- Increment value can be negative

Command Window

```
>> x = 1:8

x =

     1     2     3     4     5     6     7     8

>> x = 2:0.5:4

x =

    2.0000    2.5000    3.0000    3.5000    4.0000

>> x = 0:4:22

x =

     0     4     8    12    16    20

>> x = 10:-1:2

x =

    10     9     8     7     6     5     4     3     2

fx >> 
```

Vector Generation – linspace(…)


```
x = linspace(xstart,xstop,N)
```

- `xstart`: value of the first element in the vector
- `xstop`: value of the last element in the vector
- `N`: Number of elements in the vector
- `x`: vector of linearly spaced points

- Colon operator:
  - Stop value may not be in the vector
  - Number of points not directly specified
- `linspace(…)`:
  - `x(end) = xstop`
  - Increment value not directly specified

Array Generation – `ones(…)`, `zeros(…)`

- Generate an $N \times N$ square matrix of all 1's or all 0's:

  `A = ones(N);` or `A = zeros(N)`

- Generate an $m \times n$ vector of all 1's or 0's

  `A = ones(m,n);` or `A = zeros(m,n)`

Command Window

```
>> A = ones(5)

A =

     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1
     1     1     1     1     1

>> A = ones(1,6)

A =

     1     1     1     1     1     1
```

Command Window

```
>> A = zeros(3)

A =

     0     0     0
     0     0     0
     0     0     0

>> A = zeros(2,4)

A =

     0     0     0     0
     0     0     0     0
```

Identity Matrix – eye(…)

$$\boxed{\texttt{I = eye(N)}}$$

- N: identity matrix dimension
- I: $N \times N$ identity matrix

Command Window

```
>> I5 = eye(5)

I5 =

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1

fx >> 
```

Random Number Generation – `rand(…)`

- Very often useful to generate ***random numbers***
  - Simulating the effect of noise
  - Monte Carlo simulation, etc.

```
x = rand(m,n)
```

  - `m`: number of rows in the matrix of random numbers
  - `n`: number of columns in the matrix of random numbers
  - `x`: $m \times n$ matrix of ***uniformly-distributed*** random values on the interval $[0,1]$

- If only one dimension specified (i.e. `rand(N)`), result is an $N \times N$ matrix of random values
- For ***normally-distributed*** (Gaussian) values, use:

```
x = randn(m,n)
```

Array Indexing in MATLAB

Array Indexing

- We've seen how we can refer to specific elements in an array by their ***row, column indices***, a_{ij}:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

- MATLAB allows us to do the same thing
  - Indices specified in parentheses immediately following the array variable name
  - Indices must be positive
  - Numbering begins at 1
- For example, `B(2,5)` refers to the element in the 2nd row and 5th column of the matrix B
- Also possible to specify ranges of elements within an array

Array Indexing

- Element of A in row `i`, column `j`:

  `A(i,j)`

- Elements of A in row `i`, all columns:

  `A(i,:)`

- Elements of A in all rows, column `j`:

  `A(:,j)`

- Elements of A in rows `i` through `k`, columns `j` through `q`:

  `A(i:k,j:q)`

- Elements of A in the second through last row and the last column:

  `A(2:end,end)`

Array Indexing – Single Index

- MATLAB also allows for indexing elements within an array with a ***single index*** – ***linear indexing***
  - Elements are counted down each column sequentially
  - Very useful for ***vectors***
  - Not often useful for matrices
- For example, for a 3×4 matrix:

$$A = \begin{bmatrix} a_1 & a_4 & a_7 & a_{10} \\ a_2 & a_5 & a_8 & a_{11} \\ a_3 & a_6 & a_9 & a_{12} \end{bmatrix}$$

- In MATLAB: `A(8) = A(2,3) =` a_8

Array Indexing


```
Command Window
>> A = [9 2 4 1 3; 6 3 1 0 9; 8 2 0 4 5]

A =

     9     2     4     1     3
     6     3     1     0     9
     8     2     0     4     5

>> A(2,5)

ans =

     9

>> A(3,:)

ans =

     8     2     0     4     5

>> A(:,3)

ans =

     4
     1
     0

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

ans =

     6     3
     8     2

>> A(1,3:end)

ans =

     4     1     3
```

- `A(2,5)` is the value in the 2^{nd} row, 5^{th} column of A
- A colon (:) indicates all rows or columns
- Can index over a range of rows and/or columns
- Use `end` to index the last row or column

Array Indexing

Command Window

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

A =

     9     2     4     1     3
     6     3     1     0     9
     8     2     0     4     5

>> A(3,3) = 8

A =

     9     2     4     1     3
     6     3     1     0     9
     8     2     8     4     5

>> b = [1;3;5]

b =

     1
     3
     5

>> A(:,1) = b

A =

     1     2     4     1     3
     3     3     1     0     9
     5     2     8     4     5

>> A(2:3,3:4) = 2

A =

     1     2     4     1     3
     3     3     2     2     9
     5     2     2     2     5
```

- Use indexing to redefine specific elements in an array
- Use colon indexing to replace entire row/column with a vector
- Can replace all elements within a range
  - Can set all equal to a scalar
  - Or, redefine as a matrix

Array Indexing – Single Index

Command Window

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

A =

     9     2     4     1     3
     6     3     1     0     9
     8     2     0     4     5

>> A(4)

ans =

     2

>> A(1:5)

ans =

     9     6     8     2     3

>> A(12:end)

ans =

     4     3     9     5

>> b = 0:0.5:3

b =

         0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000

>> b(6)

ans =

    2.5000
```

- Single indexing counts down successive columns
- end index indicates last row, last column
- Generally, more useful for vectors

Matrix Size Functions – `size`, `length`

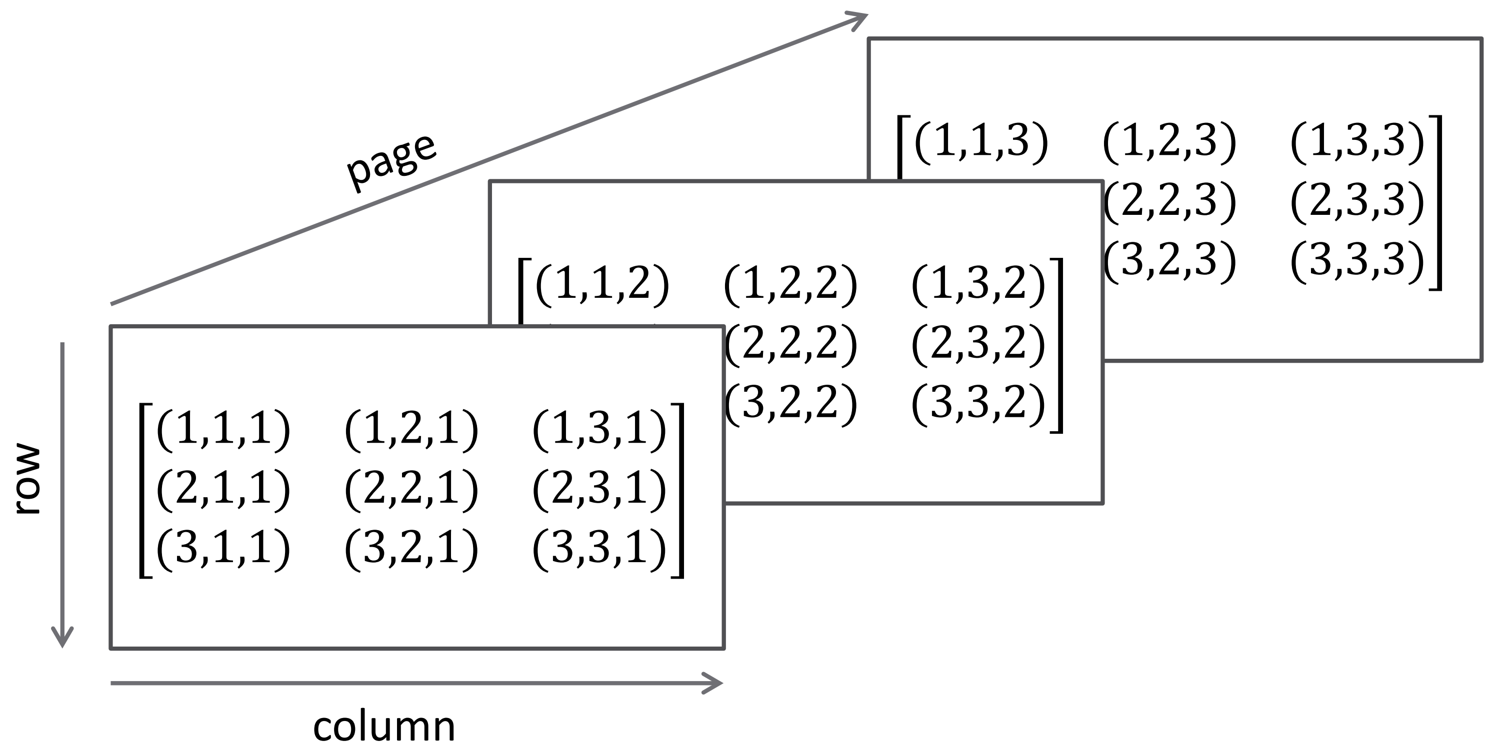
- `size(A)`
  - Returns a 1×2 row vector containing number of rows and columns of A
- `length(A)`
  - Returns a scalar equal to the greater of the number of rows or columns of A
  - `length(A)` = `max(size(A))`
  - Useful for vectors

Command Window

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

A =

     9     2     4     1     3
     6     3     1     0     9
     8     2     0     4     5

>> size(A)

ans =

     3     5

>> length(A)

ans =

     5

>> max(size(A))

ans =

     5

>> isequal(length(A),max(size(A)))

ans =

     1
```

Multidimensional Arrays

- MATLAB allows for the definition of arrays with more than two dimensions
  - Arbitrary number of dimensions allowed
  - Three dimensional arrays are common
  - Index an N-dimensional array with N indices
- For example, a $3 \times 3 \times 3$ array looks like this:



Multidimensional Arrays

- A did not exist prior to assignment
  - Size was undefined
  - Defined as smallest possible array allowing for assignment ($3 \times 3 \times 3$)
  - All other elements set to zero
- Three-dimensional array requires three indices

```
Command Window
>> A(3,3,3) = 5

A(:,:,1) =

     0     0     0
     0     0     0
     0     0     0


A(:,:,2) =

     0     0     0
     0     0     0
     0     0     0


A(:,:,3) =

     0     0     0
     0     0     0
     0     0     5

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

A(:,:,1) =

     0     0     0
     0     0     0
     0     0     0


A(:,:,2) =

     1     2     3
     0     0     0
     0     0     0


A(:,:,3) =

     0     0     0
     0     0     0
     0     0     5
```

Matrix Operations in MATLAB

Matrix Operations in MATLAB

- MATLAB treats all numeric variables as matrices
- Mathematical operations are matrix operations by default
  - Addition, subtraction, multiplication …
  - Matrix dimensions must be compatible
- Built-in functions designed to accept matrices as input arguments, e.g.:
  - Trigonometric functions
  - Exponential
  - Square root
  - Statistical functions, etc. …

Matrix Operations in MATLAB

- Matrices can be added, as long as they are the same size
- Multiplication is matrix multiplication
  - Inner dimensions must agree
  - Otherwise, an error results
- Here, transposing d satisfies inner dimension requirement

Command Window

```
>> A = [2 4; 3 5]

A =

     2     4
     3     5

>> B = ones(2)

B =

     1     1
     1     1

>> C = A + B

C =

     3     5
     4     6

>> d = [1 2]

d =

     1     2

>> E = d*C

E =

    11    17

>> E = C*d
Error using  *
Inner matrix dimensions must agree.

>> E = C*d'

E =

    13
    16
```

Passing Matrices to Functions

- Can pass vectors and matrices to most functions, just as we would a scalar
- The sine of a vector of angles calculated all at once
  - No need to pass one-at-a-time
  - Result is a vector of the same size
- `abs(…)` calculates the absolute value

Command Window

```
>> theta = [0:pi/4:2*pi]'

theta =

         0
    0.7854
    1.5708
    2.3562
    3.1416
    3.9270
    4.7124
    5.4978
    6.2832

>> y = sin(theta)

y =

         0
    0.7071
    1.0000
    0.7071
    0.0000
   -0.7071
   -1.0000
   -0.7071
   -0.0000

>> z = abs(y)

z =

         0
    0.7071
    1.0000
    0.7071
    0.0000
    0.7071
    1.0000
    0.7071
    0.0000
```

Array Operations

- Often, we want to operate on vectors and matrices ***element-by-element***
  - ***Array operations*** – not matrix operations
  - MATLAB's ***array operators***:   .*,   ./,   .^
- For example:

$$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 3 & 4 \\ 7 & 5 \end{bmatrix}$$

$$\mathbf{A} * \mathbf{B} = \begin{bmatrix} 17 & 14 \\ 37 & 32 \end{bmatrix}$$

but

$$\mathbf{A}.* \mathbf{B} = \begin{bmatrix} 3 & 8 \\ 21 & 20 \end{bmatrix}$$

Array Operations

- Matrices must be the same size to perform array operations
  - Not only inner dimensions must agree
- For example:

$$\mathbf{a} = [1 \quad 2] \qquad \mathbf{b} = [3 \quad 4]$$

$$\mathbf{a} * \mathbf{b} = ERROR$$

but

$$\mathbf{a}.* \mathbf{b} = [3 \quad 8]$$

Similarly,

$$\mathbf{b}/\mathbf{a} = ERROR$$

but

$$\mathbf{b}./\mathbf{a} = [3 \quad 2]$$

Array Operations

- *Matrix* exponentiation requires square matrix and scalar exponent
  - ***Array exponentiation*** by a scalar works for any matrix
  - Also allows for exponentiation by another matrix of the same size
- For example:

$$\mathbf{a} = [1 \quad 2 \quad 3 \quad 4] \quad \mathbf{b} = [4 \quad 3 \quad 2 \quad 1]$$

$$\mathbf{a}\hat{}2 = ERROR$$

but

$$\mathbf{a.\hat{}2} = [1 \quad 4 \quad 9 \quad 16]$$

And,

$$\mathbf{a}\hat{}\mathbf{b} = ERROR$$

but

$$\mathbf{a}.\hat{}\mathbf{b} = [1 \quad 8 \quad 9 \quad 4]$$