Introduction to Scientific Computing

Matrices

What is a matrix?

Multidimensional Array / Matrix

Scalar

$$a = 1$$

Vector

$$a = \begin{bmatrix}1 & 5 & 9\end{bmatrix}$$

$$a = \begin{bmatrix}1 \\ 5 \\ 9\end{bmatrix}$$

Matrix / Array

$$A = \begin{bmatrix}1 & 5 & 9 \\ 2 & 4 & 8 \\ 3 & 6 & 7\end{bmatrix}$$

Multidimensional Matrix / Array

$$A = \begin{bmatrix}1 & 5 & 9 \\ 2 & 4 & 8 \\ 3 & 6 & 7\end{bmatrix}$$

5
3
0 1 8

7
2
8 6 1

3^{rd} dimension

What is a matrix?

A 2D matrix consists of rows and columns

$$\begin{array}{cl} A = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 4 & 8 \\ 3 & 6 & 7 \end{bmatrix} & \begin{array}{l} \text{Row} \\ \\ \\ \end{array} \\ \text{Column} & \end{array}$$

What is the size of this matrix?

Ans:

Basic Array-related Operators

- **Operator** `[]`
  - Used to create arrays, data organized in rows and columns each having the same size (vectors, matrices)
  - Rows are formatted using ‘,’ or ‘ ’
  - Columns are formatted using ‘;’

- **Operator** `:`
  - Used to create arrays of large size & evenly spaced data

    `first_value:last_value` (default) step = 1

    `first_value:step:last_value` (customized step)

  example: `[f:n]` produces `[f,f+1,f+2,…,k]` where `k≤n` and `k+1>n`, `f≤n` or `[]` otherwise

Creating arrays

- **Creating a one-dimensional array (vector)**
  - Vectors with constant spacing

```
My_variable = [first:spacing:last]          A=[1:5]; A=[1:2:5];
My_variable = first:spacing:last            x=1:5; x=1:0.5:5
My_variable = linspace(first, last, num)    B=linspace(1,50,10)
```

  - Vectors in general

```
My_variable = [first second, …, last]
A = [1 3 0 -2 5 6 10];
```

Creating arrays (cont.)

- **Creating a two-dimensional array (matrix)**

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

- **Using built-in functions**

```
zeros, ones, and eye commands
A = ones(2,5); A = zeros(2,3); A = eye(3);
```

- **Transpose operator ' (e.g., A')**
  - Rows become columns and columns become rows
    - An `m` x `n` matrix results in an `n` x `m` matrix
    - A column vector results in a row vector
- **Expressions used to initialize arrays can include algebraic operations and (all or portions of) previously defined arrays**

```
a = [0 13*2];
b = [a(2) 13 a];
```

Accessing matrix indices

Specific values from a matrix can be accessed by specifying the row and column location using subscripts

$$A = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 4 & 8 \\ 3 & 6 & 7 \end{bmatrix}$$

What is the result of specifying `A(3,1)`?

Ans:

Accessing matrix indices

Matrix indices may also be accessed using a single subscript ("unwound") - in this case elements are evaluated starting in the upper left hand corner and down each column

$$A = \begin{bmatrix} 1^{1} & 5^{4} & 9^{7} \\ 2^{2} & 4^{5} & 8^{8} \\ 3^{3} & 6^{6} & 7^{9} \end{bmatrix}$$

Subarrays

- **Selecting subsets of rows and columns as separate arrays**
  - Array addressing using a colon (:)
    - Vectors

```
va(:)       %all elements                          va(m:n)    %elements m through n
```

    - Matrix

```
A(:,n)      %all elements in column n              A(n,:)     %all elements in row n
A(:,m:n)    %elements in columns m through n       A(m:n,:)   %elements in rows m through n
A(m:n,p:q)
```

- **Function `end` = highest value of corresponding dimension**

```
a = [1 2 3 4 5 6 7 8 9];                  a(7:end) is [7 8 9] and a(end) is 9
```

  - This function can return different values within the same expression

```
b = [1 2 3 4; 5 6 7 8; 9 10 11 12];       b(2:2:end,2:end) is [6 7 8]
```

Manipulating/creating Arrays

- **Adding elements (expanding)**
- **Deleting elements**
- **Built-in functions for arrays**

`length(v)`	number of elements
`size(A)`	returns [m, n] where m and n are the size of A
`reshape(A,m,n)`	rearrange A to have m rows and n columns
`diag(v)`	creates square matrix with elements on the diagonal
`diag(A)`	creates vector from the diagonal elements

Resizing Arrays

- **Arrays are resized automatically on assignment**

```
A(end+1) = 5
%automatically adds one more element to the array
```

- **Deleting rows or columns from an array**

```
A(:,2) = []
%deletes column 2 from A
```

- **Arrays can be concatenated to form larger arrays**

```
A = [B C; D E]
```

Accessing matrix indices

Often, the diagonal elements of a vector are important for engineering applications (ex. normal stresses from a stress tensor)

$$A = \begin{bmatrix} 1 & 5 & 9 \\ 2 & 4 & 8 \\ 3 & 6 & 7 \end{bmatrix}$$

How would one specify the diagonal elements of matrix A?

Ans: `a = [A(1,1) A(2,2) A(3,3)]`
or `a = A(1:4:9)`
or `a = diag(A)`

Strings

- **An array of characters typed between single quotes (‘)**
  - `'ad ef’`
  - `'CSCSI0040’`
  - `'Ruth Simmons’`
  - `'99.99% pure’`
  - `'Exclaiming text!’`
  - `'text with a quote''in it’`

- **char built-in function**

```
My_string = char('string 1','string 2','string 3’)
```

- **Work with strings**

```
lower; isspace; isletter; …
```

Matrix Math – Addition/Subtraction,

– Multiplication,

– Division,

Array Math,

Array Operations

Array (matrix) addition and subtraction

- **General rules for addition and subtraction with arrays of any nominal dimensions**

  - Scalars may be added to any array - the scalar value is added to all elements of the array

  - Arrays may be added to arrays - the arrays must have the same dimensions and elements in correlated positions are summed

Array (matrix) addition and subtraction

(1) Scalars may be added to any array - the scalar value is added to all elements of the array (scalar expansion)

$$A = \begin{bmatrix} 1 & 5 \\ 2 & 3 \end{bmatrix}$$

$$b = 2$$

$$b + A = 2 + \begin{bmatrix} 1 & 5 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} 2+1 & 2+5 \\ 2+2 & 2+3 \end{bmatrix}$$

$$= \begin{bmatrix} 3 & 7 \\ 4 & 5 \end{bmatrix}$$

Scalar expansion – automatically applied by MATLAB

Array (matrix) addition and subtraction

(2) Arrays may be added and subtracted so long as they have the same dimensions – subtraction occurs on an index-by-index basis

$$A=\begin{bmatrix} 1 & 5 & 9 \\ 2 & 4 & 8 \\ 3 & 6 & 7 \end{bmatrix}$$

$$B=\begin{bmatrix} 6 & 2 & 3 \\ 9 & 5 & 8 \\ 5 & 0 & 2 \end{bmatrix}$$

$$A+B=\begin{bmatrix} 7 & 7 & 12 \\ 11 & 9 & 16 \\ 8 & 6 & 9 \end{bmatrix}$$

Array (matrix) addition and subtraction

Say we define the following array

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

What is the result of `B = A + 2` **?**

```
B = [3 4; 5 6]
```

What is the result of `B = A + [3 4]` **?**

Error: `matrix dimensions must agree`

What is the result of `B = A + [3 4; 5 6]` **?**

```
B = [4 6; 8 10]
```

Element-by-Element Operations

- **Array operations, array sizes must agree**
  - Addition (`+`) and subtraction (`-`)
  - Multiplication (`.*`) and division (`./`)
  - Exponentiation (`.^`)

- **Functions applied over arrays (`sin`, `log`,…)**
  - When the argument (input) of a built-in function is an array (vector or matrix), the operation defined by the function is performed on each element of the array

- **Useful for evaluating a function**

$$y = x^2 - 4x \qquad\qquad y = \frac{z^3 + 5z}{4z^2 - 10}$$

Matrix / array multiplication

(1) Matrices and arrays may be multiplied by a scalar – MATLAB treats the scalar as an array of the same size as the N-dimensional array being multiplied

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

$$a * B = A .* B = \begin{bmatrix} 2 & 2 \\ 2 & 2 \end{bmatrix} .* \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} == \begin{bmatrix} 2 & 8 \\ 6 & 4 \end{bmatrix}$$

Scalar expansion

Matrix multiplication

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \\ A_{41} & A_{42} & A_{43} \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \\ B_{31} & B_{32} \end{bmatrix}$$

4x3 3x2

$$\mathbf{A} * \mathbf{B} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31} & A_{11}B_{12} + A_{12}B_{22} + A_{13}B_{32} \\ A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31} & A_{21}B_{12} + A_{22}B_{22} + A_{23}B_{32} \\ A_{31}B_{11} + A_{32}B_{21} + A_{33}B_{31} & A_{31}B_{12} + A_{32}B_{22} + A_{33}B_{32} \\ A_{41}B_{11} + A_{42}B_{21} + A_{43}B_{31} & A_{41}B_{12} + A_{42}B_{22} + A_{43}B_{32} \end{bmatrix}$$

4x2

$$\begin{bmatrix} 1 & 4 & 3 \\ 2 & 6 & 1 \\ 5 & 2 & 8 \end{bmatrix} \begin{bmatrix} 5 & 4 \\ 1 & 3 \\ 2 & 6 \end{bmatrix} = \begin{bmatrix} (1\cdot5+4\cdot1+3\cdot2) & (1\cdot4+4\cdot3+3\cdot6) \\ (2\cdot5+6\cdot1+1\cdot2) & (2\cdot4+6\cdot3+1\cdot6) \\ (5\cdot5+2\cdot1+8\cdot2) & (5\cdot4+2\cdot3+8\cdot6) \end{bmatrix} = \begin{bmatrix} 15 & 34 \\ 18 & 32 \\ 43 & 74 \end{bmatrix}$$

Array operators (.* versus *)

- **Say we define the following array/matrix**

  `A = [1 2; 3 4]`

- **What is the result of** `B = A*[1 2; 3 4]` **?**

  `B = [7 10; 15 22]`
  (aka “matrix product” or “matrix multiplication”)

- **What is the result of** `B = A.*[1 2; 3 4]` **?**

  `B = [1 4; 9 16]`
  (aka “array multiplication”, element by element multiplication)

Vector Multiplication

- **One row vector and a column vector**
  - row X column = scalar (dot)

$$\begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = a_1 b_1 + a_2 b_2 + a_3 b_3$$

  - column X row = matrix

$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} a_1 b_1 & a_1 b_2 & a_1 b_3 \\ a_2 b_1 & a_2 b_2 & a_2 b_3 \\ a_3 b_1 & a_3 b_2 & a_3 b_3 \end{bmatrix}$$

- **Cross product, special operation (`cross`)**

$$\begin{bmatrix} a_1 & a_2 & a_3 \end{bmatrix} \times \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} = \begin{bmatrix} (a_2 b_3 - a_3 b_2) & (a_3 b_1 - a_1 b_3) & (a_1 b_2 - a_2 b_1) \end{bmatrix}$$

Matrix division

•Identity matrix I

– Square (n x n) matrix of zeros with 1s on diagonal

$$\begin{bmatrix} 7 & 3 & 8 \\ 4 & 11 & 5 \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}=\begin{bmatrix} 7 & 3 & 8 \\ 4 & 11 & 5 \end{bmatrix}, \quad \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} 7 & 3 & 8 \\ 4 & 11 & 5 \end{bmatrix}=\begin{bmatrix} 7 & 3 & 8 \\ 4 & 11 & 5 \end{bmatrix}$$

$$\mathbf{AI} = \mathbf{IA} = \mathbf{A}$$

•Inverse of a matrix

– **B** is the inverse of **A** if **BA** = **AB** = **I**

– `inv` function or raise to -1 power

– matrix must be square and… invertible

Matrix division

•Left division

– Solve **AX = B** where **X** and **B** are column vectors.

– Multiply on the left both sides by inverse of **A**

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

– Left hand side is then

$\mathbf{A}^{-1}\mathbf{A}\mathbf{X} = \mathbf{I}\mathbf{X} = \mathbf{X}$

– Solution of **AX = B** is

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

– In MATLAB

```
X = A\B
```

Matrix division

•Right division

– Solve **XC = D** where **X** and **D** are row vectors.

– Multiply on the right both sides by inverse of **C**

$\mathbf{X \cdot C \cdot C^{-1} = D \cdot C^{-1}}$

$\mathbf{X = D \cdot C^{-1}}$

– In MATLAB

```
X = D/C
```

L9.m

Example Solving Three Linear Equations

- **Elimination of variables: Gaussian Elimination**

$$\begin{cases} 4x - 2y + 6z = 8 \\ 2x + 8y + 2z = 4 \\ 6x + 10y + 3z = 0 \end{cases}$$

Form AX = B

$$\begin{bmatrix} 4 & -2 & 6 \\ 2 & 8 & 2 \\ 6 & 10 & 3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 8 \\ 4 \\ 0 \end{bmatrix}$$

Form XC = D

$$\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} 4 & 2 & 6 \\ -2 & 8 & 10 \\ 6 & 2 & 3 \end{bmatrix} = \begin{bmatrix} 8 & 4 & 0 \end{bmatrix}$$

```
x=-1.8049
y= 0.2927
z= 2.6341
```

[L9.m]

Transposing arrays

The transpose function reflects a matrix over its main diagonal, which corresponds to the relationship

$$\left[A^T\right]_{ij} = \left[A\right]_{ji}$$

What are the transposed matrices for the following examples?

$$\begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}^T \longrightarrow \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}^T \longrightarrow \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Transposing arrays

What is the transpose of the following vector?

$$\begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix}^T \longrightarrow \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

Array Operations,

Built-in Arrays, Sparse Arrays,

Array Size,

Array Sums, Array Products, Array Concatenation

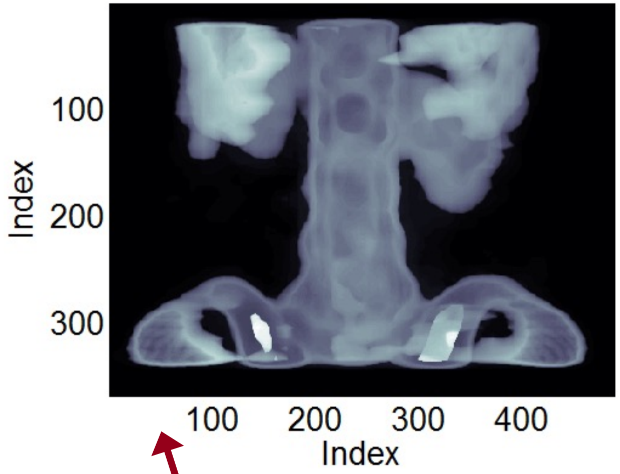
Rotating and flipping arrays

- **MATLAB permits array rotations using the** `rot90` **and flipping of matrices using** `fliplr, flipud,` **and** `flipdim`

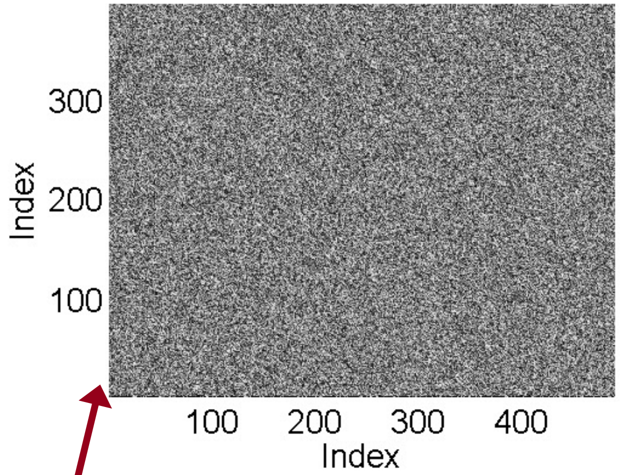
  - This capability is especially useful when interacting with combinations of images and arrays as the vertical axis of images is inverted from that of user-defined matrices

Rotating and flipping arrays

MATLAB’s built-in spine image



Matrix of randomly generated numbers



Note: the vertical indices are inverted in the “image”

Images will have to be flipped in order to interact with matrices of numbers

Rotating and flipping arrays

Code used to generate the images on the previous page

```
figure(1)
load('spine','X','map')
imagesc(X)
colormap(map)
set(gca,'FontSize',20)
set(gcf,'Color','w')
xlabel('Index')
ylabel('Index')
```

```
figure(2)
rndArr = rand(367,490);
surface(rndArr)
set(gca,'FontSize',20)
set(gcf,'Color','w')
xlabel('Index')
ylabel('Index')
axis tight
colormap('gray')
shading interp
```

Rotating and flipping arrays

Examples of rotating and flipping arrays:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \xrightarrow{\texttt{rot90(A)}} \begin{bmatrix} 3 & 6 \\ 2 & 5 \\ 1 & 4 \end{bmatrix}$$

$$\xrightarrow{\texttt{fliplr(A)}} \begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \end{bmatrix}$$

$$\xrightarrow{\texttt{flipud(A)}} \begin{bmatrix} 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix}$$

Rotating and flipping arrays

`rot90`, `fliplr`, and `flipud`, require 2D matrices - but `flipdim` can take arrays of any dimension

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

`flipdim(A,1)`

$$\begin{bmatrix} 4 & 5 & 6 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 10 & 11 & 12 \\ 7 & 8 & 9 \end{bmatrix}$$

Rotating and flipping arrays

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

`flipdim(A,2)`

$$\begin{bmatrix} 3 & 2 & 1 \\ 6 & 5 & 4 \end{bmatrix} \begin{bmatrix} 9 & 8 & 7 \\ 12 & 11 & 10 \end{bmatrix}$$

Rotating and flipping arrays

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix}$$

`flipdim(A,3)`

$$\begin{bmatrix} 7 & 8 & 9 \\ 10 & 11 & 12 \end{bmatrix} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Built-in matrices

- **MATLAB includes a lot of pre-defined matrices that may be scaled to fit your needs**

- **Generally, there are two ways to specify the dimensions of the matrices**
  - A single dimension (n) creates an n x n square matrix
  - Two dimensions (n, m) create a n x m matrix
  - Some pre-defined matrices support N-dimensional arrays (ex. `rand(m,n,p)` and `randi(m,n,p)`)

Built-in matrices

`zeros(3)`

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

`ones(2,3)`

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

`magic(3)`

$$\begin{bmatrix} 8 & 1 & 6 \\ 3 & 5 & 7 \\ 4 & 9 & 2 \end{bmatrix}$$

Equal row and column sums, only permits an n x n matrix, n>2

Built-in matrices

`eye(3)`

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Identity matrix

Matrices with random values

`rand(2,3,5)` and `randi(10,3,5,4)`

Try these out!

Creating sparse arrays

- **Sparse matrices are useful in situations such as:**
  - Search algorithms
  - Atomistics
  - Stress / strain mapping and FEM

- **In such cases, a sparse matrix may be defined to reduce memory load and speed calculations**
  - Sparse matrices are defined by a list of elements containing non-zero elements while all other elements are assumed to be zero

Creating sparse arrays

- **Consider a sparse matrix of dimensions** `1000 x 2000` **elements - that correlates to 2 million data points**

- **Assume that this matrix contains non-zero values at the following indices** `(3,4) = 15, (100,1500) = 5,` **and** `(1000,2000) = 9`

- **This matrix may be established using a conventional matrix subscripts**

```
A = zeros(1000,2000);
A(3,4) = 15;
A(100,1500) = 5;
A(1000,2000) = 9;
```

Creating sparse arrays

- **This matrix could also be established using a sparse matrix via the following**

```
A = sparse([3,100,1000],...
     [4,1500,2000],...
     [15,5,9],1000,2000);
```

- **The original (full) matrix A could also be converted to a sparse matrix via the following**

```
A = sparse(A);
```

Creating sparse arrays

The sparse matrix offers considerable savings for memory and computation time

Matrix type	Memory (MB)	Time to compute A.^2 (ms)
Full	15.2	4
Sparse	0.015	0.04

Array size

- **You can now create a lot of beautiful arrays of differing shapes and sizes, but you will often need to know the size of the array you working with**

- **MATLAB offers several methods to determine array size**
  - `size(A)` yields all of the dimensions of the array in order of typical dimensional specifications (row, column, page, etc.)
  - `length(A)` returns the largest dimension
  - `numel(A)` returns the total number of elements in the array

Array size

Consider the following array

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{bmatrix}$$

What would the following return?

`size(A)`

Ans: `[2, 8]`

`length(A)`

Ans: `8`

`numel(A)`

Ans: `16`

General / built-in array functions

- As was the case with vectors - many built-in MATLAB functions operate on any N-dimensional arrays element by element

- Assume A is a 100 x 200 matrix
  - `tan(A)` yields a 100 x 200 matrix containing the value of the tangent at each index location of A

  - The same holds true for most of the common, built-in MATLAB functions (ex. `abs(A)`, `sin(A)`, `log10(A)`)

Summing array elements

- **N-dimensional arrays may be summed across any of the dimensions or the entire array**

- **For an N-dimensional array**
  - The sum along the N^{th} dimension may be invoked using `sum(A,N)`, where N is the dimension number
  - The sum of all of the array may be output by nesting N number of sum calls (ex. `sum(sum(A))` for a 2D matrix) - the sum starts with the innermost dimension for the most highly nested sum call

Summing array elements

Consider the following 3D array

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

What is the result of calling `sum(A)`?

$$\begin{bmatrix} 2 & 2 & 2 \end{bmatrix} \quad \begin{bmatrix} 4 & 4 & 4 \end{bmatrix} \quad \begin{bmatrix} 6 & 6 & 6 \end{bmatrix}$$

Summing array elements

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

What is the result of calling `sum(sum(A))`?

$$[6] \quad [12] \quad [18]$$

Summing array elements

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

What is the result of calling `sum(sum(sum(A)))`?

$$\begin{bmatrix} 36 \end{bmatrix}$$

Summing array elements

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

What is the result of calling `sum(A,3)`?

$$\begin{bmatrix} 6 & 6 & 6 \\ 6 & 6 & 6 \end{bmatrix}$$

Cumulative sums

- `cumsum()` **sums cumulatively along a vector or some dimension of an array**
  - unless a dimension is specified, it works on the first non-singleton dimension

`cumsum([1 2 3 4 5]);`
yields `[1, 3, 6, 10, 15]`

`cumsum([1 2 3; 4 5 6]);`
yields `[1 2 3; 5 7 9]`

Product

- `prod()` **calculates the product of array elements along the first non-singleton dimension**

```
x(:,:,1) = [1 2 3; 4 5 6];
x(:,:,2) = [6 5 4; 3 2 1];
y = prod(x)
```

yields `y(:,:,1) = [4, 10, 18]` and
`y(:,:,2) = [18, 10, 4]`

Product

What does the following return?

```
x(:,:,1) = [1 2 9];
x(:,:,2) = [3 2 1];
y = prod(x)
```

yields `y(:,:,1) = 18` and `y(:,:,2) = 6`

Cumulative product

- `cumprod()` **calculates the cumulative product of array elements along the first non-singleton dimension**

```
x(:,:,1) = [1; 2; 3];
x(:,:,2) = [2; 2; 2];
y = cumprod(x)
```

yields `y(:,:,1) = [1; 2; 6]`
and `y(:,:,2) = [2; 4; 8]`