Matrices in Matlab

Arrays and Vectors

- An *array* is a collection of data objects of the same type, typically stored sequentially in memory.
- An array is an elementary *data structure*.
- Almost all programming languages provide support for arrays.
- Matlab is a language that has been particularly specialised to support arrays (and subsequently matrices).
- An array is the obvious way to represent a vector.
- A 3D vector with coordinates [1, 2.5, 5] would be represented as an array of 3 doubles arranged sequentially in memory.

Declaring (Constructing) Arrays

- In Matlab, we can construct an array and associate it with an identifier very easily. For example:

\[
\text{>> a = [1, 2.5, 5]}
\]

\[
\text{a =}
\begin{bmatrix}
1.0000 & 2.5000 & 5.0000
\end{bmatrix}
\]

- The commas are optional and can be omitted:

\[
\text{>> a = [1 2.5 5]}
\]

\[
\text{a =}
\begin{bmatrix}
1.0000 & 2.5000 & 5.0000
\end{bmatrix}
\]

- The square brackets indicate to Matlab that the contents represent an array:

\[
\text{>> whos}
\]

\[
\begin{array}{|c|c|c|}
\hline
\text{Name} & \text{Size} & \text{Bytes} \\
\hline
\text{a} & 1x3 & 24 \text{ double array} \\
\hline
\end{array}
\]

Grand total is 3 elements using 24 bytes

Operations on Arrays

- This process of constructing an array involves a segment of memory being allocated and associated with the variable name, and the elements of memory being set to the specified values.
- In most programming languages you would have to declare an array and assign the values one at a time. In Matlab this is automatic.
- You can then perform arithmetic on arrays as simply as you can with scalars. For example:

\[
\text{>> b = a*2}
\]

\[
\text{b =}
\begin{bmatrix}
2 & 5 & 10
\end{bmatrix}
\]
Working with Array Elements

- You can extract individual values from an array by specifying the index within the array using round brackets. For example:

  ```matlab
  >> c = b(1)  % c is set to the value of the first element of b
  c =
  2
  ```

- You can also assign new values to individual elements of an array. For example:

  ```matlab
  >> b(3) = 6  % Set the value of the 3rd element of b to 6
  b =
  2   5   6
  ```

- In Matlab, the index of the first element of an array is always 1.

  **Note:** this differs from languages such as C or Java where the index of the first element is always 0.

Size of an Array

- Matlab keeps track of the size of arrays and ensures you do not try to go beyond their bounds. For example:

  ```matlab
  >> b(4)
  ??? Index exceeds matrix dimensions.
  >> b(0)
  ??? Index into matrix is negative or zero.
  ```

Matrices

- An array is a collection of data objects of the same type.
- The data objects of the array can themselves be arrays.
- A **matrix** is typically represented by an array of arrays, or a 2D array. Matlab supports matrices in the same way that it supports vectors.
- Matlab uses the semi-colon (\( ; \)) operator to distinguish between the different rows of a matrix. For example:

  ```matlab
  >> a = [1 2 3; 4 5 6]  % The ; separates the individual 1D arrays.
  a =
  1 2 3
  4 5 6
  ```

Everything in Matlab is a Matrix

- Matlab also allows rows to be entered on different lines.
- Once an array is started by a square bracket (\([\]\)), Matlab assumes that a new line means a new row of the matrix. For example:

  ```matlab
  >> a = [1 2 3; 4 5 6]  % A matrix consisting of two rows
  ```

  ```matlab
  a =
  1 2 3
  4 5 6
  ```

  **As far as Matlab is concerned, _everything is a matrix!_**
- A vector is a 1xN (or Nx1) matrix; a scalar is a 1x1 matrix.
Matrix and vector operators

- The standard mathematical operators can be applied to vectors and matrices. Matlab handles all the details automatically.
- For example, suppose we have two matrices defined as:

\[
\begin{bmatrix}
1 & 2 \\
3 & 4
\end{bmatrix},
\begin{bmatrix}
5 & 6 \\
7 & 8
\end{bmatrix}
\]

- The transpose operator switches the rows and columns of a matrix.
- The transpose operator is denoted with the single apostrophe (') symbol.
- Transpose has higher precedence than multiplication.

Addition and subtraction

- Matrix addition and subtraction is same as linear algebra. For example:

\[
\begin{align*}
&\text{c = a + b} & \text{Matrix addition (and subtraction).} \\
&c = \\
&6 & 8 \\
&10 & 12
\end{align*}
\]

\[
\begin{align*}
&\text{c = a + 2} & \text{Addition of a scalar results in the scalar being added to the matrix elements.} \\
&c = \\
&3 & 4 \\
&5 & 6
\end{align*}
\]

- For matrix addition or subtraction to work, the dimensions of the two matrices must match.

Matrix multiplication

- Matrix multiplication is defined as in standard linear algebra. For example:

\[
\begin{align*}
&\text{c = a * b} & \text{Matrix multiplication.} \\
&c = \\
&19 & 22 \\
&43 & 50
\end{align*}
\]

\[
\begin{align*}
&\text{c = a * 2} & \text{Matlab recognises this as scalar multiplication.} \\
&c = \\
&2 & 4 \\
&6 & 8
\end{align*}
\]

- For matrix multiplication to work, the number of columns in the first matrix must match the number of rows in the second matrix.

Point-wise Multiplication

- Associated with matrices and vectors are a number of special operators, many of which are unique to Matlab.
- The * operator performs point-wise multiplication on each corresponding pair of elements of two matrices (sometimes called “array multiplication”).
- For example:

\[
\begin{align*}
&\text{c = a .* b} & \text{Point-wise multiplication.} \\
&c = \\
&5 & 12 \\
&21 & 32
\end{align*}
\]
**Point-wise Division and Exponentiation**

- The `/` operator performs point-wise division on each corresponding pair of elements of two matrices.
  
  For example:
  ```matlab
  >> c = a ./ b          % Point-wise division.
  c =
  0.2000    0.3333
  0.4286    0.5000
  ```

- The `.^` operator performs point-wise exponentiation on each corresponding pair of elements of two matrices.
  
  For example:
  ```matlab
  >> c = a .^ b        % Point-wise exponentiation.
  c =
  1          64
  2187       65536
  ```

**Matrix Division**

- Matrix division implies solving for matrix inverses. Matlab handles this automatically!
  
  Note that because matrix multiplication is not commutative, we require the concept of left and right division.
  
  Right division is post-multiplication by the inverse of a matrix:
  ```matlab
  >> c = a / b;            % c = a * b^{-1}
  c =
  3    -2
  2    -1
  ```

- Left division is pre-multiplication by the inverse of a matrix:
  ```matlab
  >> c = a \ b             % c = a^{-1} * b
  c =
  -3    -4
  4     5
  ```

**Checking...**

- Left division is the most common.
  
  The expression `c = a\b` above would solve the equation `b = a*c`.
  
  Double check:
  ```matlab
  >> a*c
  ans =
  5     6
  7     8
  ```
  which is the value of the `b` matrix.

- Most of these expressions would require at least 5 lines of code if programmed in some other language such as C or Java. Matlab's syntax yields very concise and readable code.

**Array constructors - the colon operator**

- It is easy to construct small arrays by explicitly specifying all the elements, but this is not practical for large arrays.
  
  Matlab provides the `colon` operator (:) for constructing sequences of values.
  
  The colon operator produces an array equivalent to the elements of an arithmetic sequence.
  
  Arithmetic sequences are defined in terms of the first value in the series, the increment between successive values, and the last value in the series.
  
  The syntax for building an array using the colon operator is:
  ```matlab
  array = first : increment : last
  ```
Colon Operator

- For example:
  ```matlab
  >> x = 3 : 2 : 11
  x =
  3   5   7   9   11
  ```
- If the increment is 1, it can be omitted.
- For example:
  ```matlab
  >> x = 1 : 10
  x =
  1   2   3   4   5   6   7   8   9   10
  ```
- The colon operator is enormously useful, not only for array creation but also for loop control.

Subarrays

- As well as selecting individual elements from arrays, Matlab allows for the selection of sections of an array.
- For example:
  ```matlab
  >> a = [10:-1:1]
  a =
  10   9   8   7   6   5   4   3   2   1
  >> a(4:9) % Use the colon operator (with a default increment of 1) to select elements 4 to 9 from the array.
  ans =
  7   6   5   4   3   2
  ```

Subarrays (cont.)

- You can also assign values to subarrays.
  ```matlab
  >> a(1:3) = [8 9 10]
  a =
  8   9   10   7   6   5   4   3   2   1
  ```
- The size of the array being assigned must match the size of the array selected.

Extracting Data from 2D Arrays

Matrix selection operations are extended to 2D arrays in a natural way.

- For example:
  ```matlab
  >> a = [1 2 3; 4 5 6; 7 8 9]
  >> a(2, 3) % Get the value at row 2, column 3.
  ans =
  6
  >> a(2,1:3) % Get row 2, column 1 to 3.
  ans =
  4   5   6
  >> a(2:3, 1:2) % Get the values from rows 2-3 in columns 1-2.
  ans =
  4   5
  7   8
  ```
Extracting all rows or columns

- If you want to extract all the rows (or all the columns) from a matrix, you can use the empty colon operator to specify the row or column.

- For example:

```matlab
>> a(:, 2)          % Get the values from every row in column 2.
    ans =
        2
        5
        8

>> a(2:3, :)        % Get the values from rows 2-3 in all columns.
    ans =
        4     5     6
        7     8     9
```

Matrix concatenation

- Matlab has a very convenient syntax for concatenating matrices - just stick the matrices side by side, or on top of each other, within a set of enclosing square brackets.

```matlab
>> a = [1 2 3]
>> b = [a 7 8]  % Concatenate 7 and 8 onto the end of a.
    b =
        1     2     3     7     8

>> a = [a a(1:2) % Construct a new matrix a. The first row is the "old a" with % elements 1:2 of a concatenated to the end. The second row is made from array b.
        b ]
    a =
        1     2     3     1     2
        1     2     3     7     8
```

Assigning to arrays and subarrays

- Note that in the last example, the memory required to store the new matrix will not "fit into" the old space occupied by the original matrix.
- Matlab will handle any memory allocation needed to make matrices fit.
- Matlab handles assignment to subarrays and arrays differently.
  - For assignment statements involving subarrays, the shapes of the subarrays on either side of the equal sign must match. Otherwise, Matlab will produce an error. The assignment will only replace the specified elements.
  - In contrast, assigning to an existing array will replace the entire contents of the array, may even resize the array.

Summary of Arithmetic Operations
### Arithmetic Operations between Two Scalars

<table>
<thead>
<tr>
<th>Operation</th>
<th>Algebraic Form</th>
<th>Matlab Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>$a + b$</td>
<td>a+b</td>
</tr>
<tr>
<td>Subtraction</td>
<td>$a - b$</td>
<td>a-b</td>
</tr>
<tr>
<td>Multiplication</td>
<td>$a \times b$</td>
<td>a*b</td>
</tr>
<tr>
<td>Division</td>
<td>$a / b$</td>
<td>a/b</td>
</tr>
<tr>
<td>Exponentiation</td>
<td>$a^b$</td>
<td>a^b</td>
</tr>
</tbody>
</table>

### Common “Array” and Matrix Operations (1)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Matlab Form</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Addition</td>
<td>a + b</td>
<td>Array addition and Matrix addition are identical</td>
</tr>
<tr>
<td>Array Subtraction</td>
<td>a - b</td>
<td>Array subtraction and matrix subtraction are identical</td>
</tr>
<tr>
<td>Array Multiplication</td>
<td>a .* b</td>
<td>Element-by-element multiplication of $a$ and $b$. Both arrays must of the same shape, or one of them must be a scalar.</td>
</tr>
<tr>
<td>Matrix Multiplication</td>
<td>a * b</td>
<td>Matrix multiplication of $a$ and $b$. The number of columns in $a$ must be equal to number of rows in $b$.</td>
</tr>
</tbody>
</table>

### Common “Array” and Matrix Operations (2)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Matlab Form</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array Right Division</td>
<td>a ./ b</td>
<td>Element-by-element division of $a$ and $b$: $a(i,j)/b(i,j)$. Both arrays must of the same shape, or one of them must be a scalar.</td>
</tr>
<tr>
<td>Array Left Division</td>
<td>a \ b</td>
<td>Element-by-element division of $a$ and $b$: $b(i,j)/a(i,j)$. Both arrays must of the same shape, or one of them must be a scalar.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Matlab Form</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Right Division</td>
<td>a / b</td>
<td>Matrix division is defined by $a*inv(b)$, where $inv(b)$ is the inverse of matrix $b$.</td>
</tr>
<tr>
<td>Matrix Left Division</td>
<td>a \ b</td>
<td>Matrix division is defined by $inv(a)*b$, where $inv(a)$ is the inverse of matrix $a$.</td>
</tr>
<tr>
<td>Array Exponentiation</td>
<td>a .^ b</td>
<td>Element-by-element exponentiation of $a$ and $b$: $a(i,j)^{b(i,j)}$. Both arrays must of the same shape, or one of them must be a scalar.</td>
</tr>
</tbody>
</table>