MATLAB Arrays and Matrices

Trevor Spiteri
trevor.spiteri@um.edu.mt
http://staff.um.edu.mt/trevor.spiteri

Department of Communications and Computer Engineering
Faculty of Information and Communication Technology
University of Malta

13 February, 2008

Outline

Outline

Writing vectors inline

- We want to store a row vector rv $= \begin{bmatrix} 11 & 12 & 13 \end{bmatrix}$ and a column vector cv $= \begin{bmatrix} 11 \\ 21 \\ 31 \end{bmatrix}$.
- The row vector can be written as: `>> rv = [11 12 13]`
- It can also be written as: `>> rv = [11, 12, 13]`
- The column vector can be written as: `>> cv = [11; 21; 31]`
- Or we can use the transpose operator: `>> cv = [11 21 31]'`
- It can also be written as:

```
>> cv = [11
21
31]
```

Regularly spaced elements

- The colon operator `(:)` generates regularly spaced elements. The expression is: `>> r = [j:d:k]`
- `j` is the first element.
- `k` is the last element.
- `d` is the spacing.
- For example, `1:2:9` gives us $\begin{bmatrix}1 & 3 & 5 & 7 & 9\end{bmatrix}$.
- If the `d` term is omitted, the spacing is presumed to be 1.
- For example, `1:5` gives us $\begin{bmatrix}1 & 2 & 3 & 4 & 5\end{bmatrix}$.
- `d` can also be negative.
- For example, `0.5:-0.2:0.1` gives us $\begin{bmatrix}0.5 & 0.3 & 0.1\end{bmatrix}$.
- To find the length of a row vector we use the `length` function.
- For example, `>> r = 1:4; length(r)` returns 4.

Creating matrices

- When writing row and column numbers, the row always comes first.
- We want to store a matrix `m` $= \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix}$.
- This matrix has 2 rows and 3 columns.
- We say that this is a 2 by 3 matrix.
- A matrix can be written as:
  ```
  >> m = [11 12 13; 21 22 23]
  ```
- It can also be written as:
  ```
  >> m = [11 12 13
  21 22 23]
  ```
- The size of the matrix can be obtained using the `size` function. For example, `size(m)` returns $\begin{bmatrix} 2 & 3 \end{bmatrix}$, i.e., `m` is a 2 by 3 matrix.

Accessing a single element

- We can refer to an element of the matrix using indexing.
- Suppose we have the matrix $\mathtt{m} = \begin{bmatrix} 1 & 2 & 4 \\ 1 & 4 & 9 \end{bmatrix}$ and we want to change the element in row 1, column 3.
- We write: `>> m(1, 3) = 3`
- Now, the matrix is $\mathtt{m} = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 4 & 9 \end{bmatrix}$.

Accessing whole vectors

- Suppose we have the matrix $\mathtt{m} = \begin{bmatrix} 1 & 2 & 4 & 4 \\ 1 & 4 & 16 & 16 \\ 1 & 4 & 16 & 16 \end{bmatrix}$ and we want to change all of row 3.
- We write: `>> m(3, :) = [1 8 64 64]`
- Now, the matrix is $\mathtt{m} = \begin{bmatrix} 1 & 2 & 4 & 4 \\ 1 & 4 & 16 & 16 \\ 1 & 8 & 64 & 64 \end{bmatrix}$.
- To change column 3, we write: `>> m(:, 3) = [3; 9; 27]`
- The final matrix is $\mathtt{m} = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 1 & 4 & 9 & 16 \\ 1 & 8 & 27 & 64 \end{bmatrix}$.

Accessing parts of a matrix

- Suppose we have the matrix `m` $= \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 4 & 0 & 0 & 25 \\ 1 & 8 & 0 & 0 & 125 \end{bmatrix}$ and we want to fill in values for the zeros.
- We can write: `>> m(2:3, 3:4) = [9 16; 27 64]`
- The matrix becomes `m` $= \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 4 & 9 & 16 & 25 \\ 1 & 8 & 27 & 64 & 125 \end{bmatrix}$.
- Another way of doing this change is:
  `>> m(2:end, 3:4) = [9 16; 27 64]`
- `1` is index of the first element, and `end` is the index of the last element.

Appending and prepending vectors

- Start with the matrix $\mathtt{m} = \begin{bmatrix} 11 & 12 \\ 21 & 22 \end{bmatrix}$.
- To append a column, we write: `>> m = [m, [13; 23]]`
- Now, $\mathtt{m} = \begin{bmatrix} 11 & 12 & 13 \\ 21 & 22 & 23 \end{bmatrix}$.
- To add some rows, we write: `>> m = [1 2 3; m; 31 32 33]`
- Now, $\mathtt{m} = \begin{bmatrix} 1 & 2 & 3 \\ 11 & 12 & 13 \\ 21 & 22 & 23 \\ 31 & 32 & 33 \end{bmatrix}$.

Growing a matrix by indexing

- Start with the matrix `m` $= \begin{bmatrix} 11 & 12 \\ 21 & 22 \end{bmatrix}$.
- If we want to resize the array to be a 4 by 6 array, we write:
  `>> m(4, 6) = 1`
- The new elements are automatically filled with zeros.
- The new matrix is `m` $= \begin{bmatrix} 11 & 12 & 0 & 0 & 0 & 0 \\ 21 & 22 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$.

Deleting vectors from a matrix

- Start with the matrix `m` $= \begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 21 & 22 & 23 & 24 & 25 \\ 31 & 32 & 33 & 34 & 35 \end{bmatrix}$.
- To delete the second row, we write: `>> m(2, :) = []`
- Now the matrix is `m` $= \begin{bmatrix} 11 & 12 & 13 & 14 & 15 \\ 31 & 32 & 33 & 34 & 35 \end{bmatrix}$.
- To delete the first column, we write: `>> m(:, 1) = []`
- Now the matrix is `m` $= \begin{bmatrix} 12 & 13 & 14 & 15 \\ 32 & 33 & 34 & 35 \end{bmatrix}$.
- To delete columns 3 onwards: `>> m(:, 3:end) = []`
- Now the matrix is `m` $= \begin{bmatrix} 12 & 13 \\ 32 & 33 \end{bmatrix}$.
- This works on whole rows or columns only.

Creating special matrices

- The `zeros` function creates a matrix of zeros.
- `zeros(2, 4)` $= \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$
- The `ones` function creates a matrix of ones.
- `ones(2, 3)` $= \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$
- The `eye` function creates an identity matrix ($\mathbf{I}$).
- `eye(3)` $= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- The `rand` function creates a matrix of random numbers in $[0, 1)$.
- `rand(2, 4)` $= \begin{bmatrix} 0.1576 & 0.9572 & 0.8003 & 0.4218 \\ 0.9706 & 0.4854 & 0.1419 & 0.9157 \end{bmatrix}$

Outline

- Creating and Editing Arrays
  - Creating Arrays
  - Editing Arrays
  - Special Matrices
- Mathematical Operations
  - Element by Element Operations
  - Functions on Many Values
  - Matrix Operations
- Polynomial Algebra
- Structure Arrays

Adding and subtracting arrays

- We start with two matrices, $\mathtt{a} = \begin{bmatrix} 15 & 52 \\ 23 & 23 \end{bmatrix}$ and $\mathtt{b} = \begin{bmatrix} 52 & 15 \\ 61 & 12 \end{bmatrix}$.
- To add, we write: `>> apb = a + b`
- $\mathtt{apb} = \begin{bmatrix} 67 & 67 \\ 84 & 35 \end{bmatrix}$
- To subtract, we write: `>> amb = a - b`
- $\mathtt{amb} = \begin{bmatrix} -37 & 37 \\ -38 & 11 \end{bmatrix}$
- Matrix addition is element-by-element addition.
- So array addition is matrix addition and array subtraction is matrix subtraction.

Multiplicating arrays

- Matrix multiplication is not element-by-element multiplication.
- For array multiplication, we have a special operator: `.*`
- We start with two matrices, $a = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $b = \begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix}$.
- To array-multiply, we write: `>> m = a .* b`
- To multiply by a scalar, we may use matrix multiplication (`*`) or array multiplication (`.*`).
- $\texttt{a * 2} = \texttt{a .* 2} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$
- $\texttt{2 * a} = \texttt{2 .* a} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$

Dividing arrays

- Start with $\texttt{a} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $\texttt{b} = \begin{bmatrix} 2 & 3 \\ 1 & 2 \end{bmatrix}$.
- To perform array right division, we write: `>> rd = a ./ b`
- $\texttt{rd} = \begin{bmatrix} 0.5 & 0.6667 \\ 3 & 2 \end{bmatrix}$
- To perform array left division, we write: `>> ld = a .\ b`
- $\texttt{ld} = \begin{bmatrix} 2 & 1.5 \\ 0.3333 & 0.5 \end{bmatrix}$
- Scalar division can be performed as follows:
- $\texttt{a / 2} = \texttt{a ./ 2} = \texttt{2 \textbackslash\ a} = \texttt{2 .\textbackslash\ a} = \begin{bmatrix} 0.5 & 1 \\ 1.5 & 2 \end{bmatrix}$
- $\texttt{2 ./ a} = \texttt{a .\textbackslash\ 2} = \begin{bmatrix} 2 & 1 \\ 0.6667 & 0.5 \end{bmatrix}$

Array exponentiation

- Start with $\mathtt{a} = \begin{bmatrix} 1 & 2 \\ 0 & 3 \end{bmatrix}$ and $\mathtt{b} = \begin{bmatrix} 0 & 1 \\ 2 & 4 \end{bmatrix}$.
- To perform element-by element exponentiation, we use the `.^` operator.
- `a .^ b` $= \begin{bmatrix} 1 & 2 \\ 0 & 81 \end{bmatrix}$
- This can also be used with a scalar.
- `2 .^ a` $= \begin{bmatrix} 2^1 & 2^2 \\ 2^0 & 2^3 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 1 & 8 \end{bmatrix}$
- `a .^ 2` $= \begin{bmatrix} 1^2 & 2^2 \\ 0^2 & 3^2 \end{bmatrix} = \begin{bmatrix} 1 & 4 \\ 0 & 9 \end{bmatrix}$
- The matrix exponentiation operator `^` cannot be used for element-by-element exponentiation, not even with a scalar.

Functions operating on all elements

- Some functions can be performed on a whole array with one command.
- Start with `angles` $= \begin{bmatrix} 0 & \frac{\pi}{2} \\ \frac{\pi}{3} & \pi \end{bmatrix} = \begin{bmatrix} 0 & 1.5708 \\ 1.0472 & 3.1416 \end{bmatrix}$.
- We can find the sine of these values with one function call.
- `sin(angles)` $= \begin{bmatrix} 0 & 1 \\ 0.8660 & 0 \end{bmatrix}$

Four-quadrant inverse tangent

- Suppose we have some x and y coorinates.
- We have $\mathtt{x} = \begin{bmatrix} -1 & 1 \\ -1 & 1 \end{bmatrix}$ and $\mathtt{y} = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$.
- To find the angles, we can try to use the `atan` function.
- `atan(y ./ x)` $= \begin{bmatrix} -0.7854 & 0.7854 \\ 0.7854 & -0.7854 \end{bmatrix} = \begin{bmatrix} -\frac{\pi}{4} & \frac{\pi}{4} \\ \frac{\pi}{4} & -\frac{\pi}{4} \end{bmatrix}$
- The results are only in the two quadrants $(-\frac{\pi}{2}, \frac{\pi}{2})$.
- The `atan2` function gives the four-quadrant inverse tangent.
- `atan2(y, x)` $= \begin{bmatrix} 2.3562 & 0.7854 \\ -2.3562 & -0.7854 \end{bmatrix} = \begin{bmatrix} \frac{3\pi}{4} & \frac{\pi}{4} \\ -\frac{3\pi}{4} & -\frac{\pi}{4} \end{bmatrix}$
- The results can be in the four quadrants $(-\pi, \pi)$.

Multiplying matrices

- We have two matrices, `a` $= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and `b` $= \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$.
- Matrix multiplication can be performed with the $*$ operator.
- `a * b` $= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} = \begin{bmatrix} 10 & 13 \\ 22 & 29 \end{bmatrix}$
- `b * a` $= \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 11 & 16 \\ 19 & 28 \end{bmatrix}$

Dividing matrices (inverse)

- Start with $\mathtt{a} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $\mathtt{b} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix}$.
- Matrix division is equivalent to multiplying by the inverse.
- $\mathtt{a\ /\ b} = \mathtt{a\ *\ inv(b)} = \begin{bmatrix} 1.5 & -0.5 \\ 0.5 & 0.5 \end{bmatrix}$
- $\mathtt{b\ \backslash\ a} = \mathtt{inv(b)\ *\ a} = \begin{bmatrix} 2 & 1 \\ -1 & 0 \end{bmatrix}$
- $\mathtt{a\ \backslash\ b} = \mathtt{inv(a)\ *\ b} = \begin{bmatrix} 0 & -1 \\ 1 & 2 \end{bmatrix}$
- $\mathtt{b\ /\ a} = \mathtt{b\ *\ inv(a)} = \begin{bmatrix} 0.5 & 0.5 \\ -0.5 & 1.5 \end{bmatrix}$

Matrix exponentiation

- To multiply a square matrix by itself, we can use matrix exponentiation.
- For example, let us cube the matrix `a` $= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$.
- `a ^ 3` $= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 37 & 54 \\ 81 & 118 \end{bmatrix}$

Matrix transposition

- To find the transpose of a matrix, we can use the `'` operator.
- For example, let `a` $= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$.
- `a'` $= \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^T = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix}$
- The `'` operator also performs the complex conjugate.
- Suppose that `z` $= \begin{bmatrix} 1+i & 2-i \\ 3+2i & 4 \end{bmatrix}$
- `z'` $= \begin{bmatrix} 1+i & 2-i \\ 3+2i & 4 \end{bmatrix}^{T*} = \begin{bmatrix} 1-i & 3-2i \\ 2+i & 4 \end{bmatrix}$
- To perform transposition and no conjugation, use the `.'` operator.
- `z.'` $= \begin{bmatrix} 1+i & 2-i \\ 3+2i & 4 \end{bmatrix}^T = \begin{bmatrix} 1+i & 3+2i \\ 2-i & 4 \end{bmatrix}$

Outline

Storing polynomials

- Polynomials are stored as vectors.
- The polynomial $f(x) = a_1 x^n + a_2 x^{n-1} + \ldots + a_n x + a_{n+1}$ can be stored as `f` $= \begin{bmatrix} a_1 & a_2 & \ldots & a_n & a_{n+1} \end{bmatrix}$
- The first element is the coefficient of the highest power.
- If a particular power of x is missing, we use 0.
- $f_1(x) = x^3 + 8x^2 + 1$
- The coefficient of x^1 is 0.
- We can write this as: `f1 = [1 8 0 1]`

Adding polynomials

- To add polynomials, make sure they have the same degree.
- $f_1(x) = x^3 + 8x^2 + 1$
  $f_2(x) = 3x^2 + x + 4$
  $f_s(x) = f_1(x) + f_2(x)$
- Since f_1 has degree 3, all polynomials have to be of degree 3.
- ```
  >> f1 = [1 8 0 1]
  >> f2 = [0 3 1 4]
  >> fs = f1 + f2
  ```
- This gives us `fs` $= \begin{bmatrix} 1 & 11 & 1 & 5 \end{bmatrix}$.
- $f_s(x) = x^3 + 11x^2 + x + 5$
  ```

Multiplying polynomials

- To multiply two polynomials, we use the `conv` function.
- $f_1(x) = x^2 + 1$
  $f_2(x) = x^3 + 2x + 1$
  $f_p(x) = f_1(x) f_2(x)$
- ```
  >> f1 = [1 0 1]
  >> f2 = [1 0 2 1]
  >> fp = conv(f1, f2)
  ```
- This gives us `fp` $= \begin{bmatrix} 1 & 0 & 3 & 1 & 2 & 1 \end{bmatrix}$.
- $f_p(x) = x^5 + 3x^3 + x^2 + 2x + 1$
  ```

Dividing polynomials

- To divide two polynomials, we use the `deconv` function.
- $num(x) = x^3 + 2x^2 - x + 3$
  $den(x) = x^2 + 2x + 1$
- We need to find the quotient $q(x)$ and the remainder $r(x)$, such that $num(x) = q(x)den(x) + r(x)$
- ```
  >> num = [1 2 -1 3]
  >> den = [1 2 1]
  >> [q r] = deconv(num, den)
  ```
- This gives us q $= \begin{bmatrix} 1 & 0 \end{bmatrix}$ and r $= \begin{bmatrix} 0 & 0 & -2 & 3 \end{bmatrix}$
- $q(x) = x$
  $r(x) = -2x + 3$
- To check the result, we write: `>> conv(q, den) + r`
  ```

Working with roots

- We can calculate a polynomial from the roots.
- $f(x) = (x-1)(x+2)(x+5)$
- The roots of $f(x)$ are 1, -2 and -5.
- `>> r = [1 -2 -5]`
  `>> f = poly(r)`
- Now `f` is a row vector with the polynomial coefficients, `f` $= \begin{bmatrix} 1 & 6 & 3 & -10 \end{bmatrix}$.
- $f(x) = x^3 + 6x^2 + 3x - 10$
- To find the roots of a polynomial, we write `>> r2 = roots(f)`
- This gives us a column vector with the roots, `r2` $= \begin{bmatrix} -5 \\ -2 \\ 1 \end{bmatrix}$.

Evaluating polynomials

- Sometimes we need to evaluate a polynomial over a range of inputs.
- Suppose we want to evaluate $f(x) = x^3 - 9x + 1$ over a range of x.
- We can do this using the `polyval` function.
- ```
  >> f = [1 0 -9 1]
  >> x = 0:5
  >> y = polyval(f, x)
  ```
- Now, $\mathrm{x} = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \end{bmatrix}$
  and $\mathrm{y} = \begin{bmatrix} 1 & -7 & -9 & 1 & 29 & 81 \end{bmatrix}$.
  ```

Outline

Creating structure arrays

- Structure arrays are suitable to hold data.
- For example, to hold strings and lengths:

```
>> s.string = 'Hello'
>> s.length = 5
```

- To add another structure to the array:

```
>> s(2).string = 'Hi'
>> s(2).length = 2
```

- Now `s` is a vector.

  $\mathtt{s} = \begin{bmatrix} \mathtt{s(1)} & \mathtt{s(2)} \end{bmatrix}$

  `s(1)` = string: `'Hello'`, length: 5

  `s(2)` = string: `'Hi'`, length: 2

Accessing structure data

- To access fields in the data structure, we use the `.` operator.
- `s(1).string` returns the `string` field of the first element.
- We can also use an expression after the `.` operator.
```
>> field_name = strcat('str', 'ing') % 'string'
>> field = s(1).(field_name)
```
- Now, `field` contains the string `'Hello'`.
- If we try to access a new field, the field will be added to all the elements of the array.
- `>> s(1).other = 34`
- `s(1)` = string: `'Hello'`, length: 5, other: 34
  `s(2)` = string: `'Hi'`, length: 2, other: []
- Now, `s(2).other` contains an empty matrix, [].