Worksheet 4. Matrices in Matlab

Creating matrices in Matlab

Matlab has a number of functions for generating elementary and common matrices.

`zeros`	Array of zeros
`ones`	Array of ones
`eye`	Identity matrix
`repmat`	Replicate and tile array
`blkdiag`	Creates block diagonal array
`rand`	Uniformly distributed
`randn`	Normally distributed random number
`linspace`	Linearly spaced vector
`logspace`	Logarithmically spaced vector
`meshgrid`	X and Y arrays for 3D plots
`:`	Regularly spaced vector
`:`	Array slicing

If given a single argument they construct square matrices.

```
octave:1> eye(4)
ans =
   1   0   0   0
   0   1   0   0
   0   0   1   0
   0   0   0   1
```

If given two entries n and m they construct an $n \times m$ matrix.

```
octave:2> rand(2,3)
ans =
   0.42647   0.81781   0.74878
   0.69710   0.42857   0.24610
```

It is also possible to construct a matrix the same size as an existing matrix.

```
octave:3> x=ones(4,5)
x =
   1   1   1   1   1
   1   1   1   1   1
   1   1   1   1   1
   1   1   1   1   1
```

```
octave:4> y=zeros(size(x))
y =
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
   0   0   0   0   0
```

- Construct a 4 by 4 matrix whose elements are random numbers evenly distributed between 1 and 2.
- Construct a 3 by 3 matrix whose off diagonal elements are 3 and whose diagonal elements are 2.
- Construct the matrix $\begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{pmatrix}$

The `size` function returns the dimensions of a matrix, while `length` returns the largest of the dimensions (handy for vectors).

```
octave:5> size(x)
ans =
   4   5

octave:6> length(x)
ans =  5
```

`repmat` builds up matrices from blocks composed of smaller matrices.

```
octave:7> A=[1 2; 3 4]
A =
   1   2
   3   4

octave:8> B=[A A A ; A A A]
B =
   1   2   1   2   1   2
   3   4   3   4   3   4
```

```
   1   2   1   2   1   2
   3   4   3   4   3   4

octave:9> C=repmat(A,2,3)
C =
   1   2   1   2   1   2
   3   4   3   4   3   4
   1   2   1   2   1   2
   3   4   3   4   3   4
```

`Blkdiag` constructs a block diagonal matrix from submatrices.

```
octave:10> A
A =
   1   2
   3   4

octave:11> B=[5 6; 7 8]
B =
   5   6
   7   8

octave:12> C=blkdiag(A,B)
C =
   1   2   0   0
   3   4   0   0
   0   0   5   6
   0   0   7   8
```

`linspace(begin,end,number)` produces `number` points equally distributed between `begin` and `end`.

```
octave:44> linspace(0,1,5)
ans =
   0.00000   0.25000   0.50000   0.75000   1.00000
```

Subscripting matrices in Matlab

A single element of a matrix can be accessed using its i, j indices.

```
octave:13> A
A =
   1   2
   3   4

octave:14> A(1,2)
ans =  2

octave:15> A(1,2)=5
A =
   1   5
   3   4
```

It is also possible to access submatrices using ranges. Ranges can be specified as `first:last` for unit increments (`first`<`last`) or `first:step:last` where step can be negative

```
octave:31> 1:10
ans =
   1   2   3   4   5   6   7   8   9  10

octave:32> -10:-1:-20
ans =
  -10  -11  -12  -13  -14  -15  -16  -17  -18  -19  -20

octave:23> A=1:16
A =
   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16

octave:25> A=reshape(A,4,4)
A =
    1    5    9   13
    2    6   10   14
    3    7   11   15
    4    8   12   16

octave:26> A(1:2,2:3)
ans =
    5    9
```

```
    6   10

octave:27> A(1:2,2:3)=zeros(2)
A =
    1    0    0   13
    2    0    0   14
    3    7   11   15
    4    8   12   16
```

`end` can be used to specify the last entry in cases where it can't be deduced from the context: e.g. iterating backwards. `reshape` is used to change the shape of a matrix.

```
octave:30> A(end:-1:1,end)
ans =
   16
   15
   14
   13
```

A single ':' as an argument to a matrix produces a column vector of all the columns

```
octave:37> a=[1 3; 2 4]
a =
   1   3
   2   4

octave:38> a(:)
ans =
   1
   2
   3
   4
```

[] denotes an empty matrix: rows and columns can be deleted from a matrix by overwriting them with an empty matrix.

```
octave:41> a=[1 2 3; 4 5 6; 7 8 9]
a =
   1   2   3
   4   5   6
```

```
   7   8   9

octave:42> a(:,2)=[]
a =
   1   3
   4   6
   7   9
```

Matrix and Array Operations

Matlab carries out operations on matrices either in a matrix sense or an array sense.

Operation	Matrix sense	Array Sense
Addition	`a+b`	`a+b`
Subtraction	`a-b`	`a-b`
Multiplication	`a*b`	`a.*b`
Left division	`a\b`	`a.\b`
Right division	`a/b`	`a./b`
Exponentiation	`a^2`	`a.^2`

The matrix and array senses of addition, subtraction and multiplication by a scalar are the same.

$$(A+B)_{ij} = A_{ij} + B_{ij} \tag{1}$$

$$(A-B)_{ij} = A_{ij} - B_{ij} \tag{2}$$

$$(\lambda A)_{ij} = \lambda A_{ij} \tag{3}$$

A and B are matrices, λ is a scalar.

Matrix and array multiplication of two matrices are quite different. Matrix multiplication `A*B` = $\sum_k A_{ik}B_{kj}$; array multiplication `A.*B` = $A_{ij}B_{ij}$ (no Einstein summation convention).

```
octave:45> a=[1 2; 3 4]; b=[5 6; 7 8];
octave:46> a*b
ans =
   19   22
   43   50
octave:47> a.*b
ans =
    5   12
   21   32
```

Similarly for exponentiation. Matrix sense `A^2` = $\sum_k A_{ik}A_{kj}$. Array sense `A.^2` = A_{ij}^2.

We postpone a discussion of the meaning of dividing by a matrix until later.

- The Pauli spin matrices $\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, $\sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}$ and $\sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ are used in non-relativistic quantum mechanical descriptions of spin-$\frac{1}{2}$ particles (e.g. electrons).
- Verify that the Pauli matrices satisfy $\sigma_x^2 = \sigma_y^2 = \sigma_z^2 = 1$, $\sigma_y\sigma_z = i\sigma_x$, $\sigma_z\sigma_x = i\sigma_y$, $\sigma_x\sigma_y = i\sigma_z$ and the commutation relation $\sigma_i\sigma_j + \sigma_j\sigma_i = 2\delta_{ij}$