MATRICES & MATRIX COMPUTATIONS

in R, Python 2.7, SAS and Julia

Creating a Matrix

matrix(data, ncol, nrow, byrow)

```
> mat1 <- matrix(1:6,2,3)
> mat1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> mat2 <- matrix(1:6,2,3,byrow=T)
> mat2
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

numpy.matrix()

```
>>> import numpy
>>> mat1 = numpy.matrix('1 3 5; 2 4 6')
>>> mat1
matrix([[1, 3, 5],
        [2, 4, 6]])
>>> mat2 = numpy.matrix('1 2 3; 4 5 6')
>>> mat2
matrix([[1, 2, 3],
        [4, 5, 6]])
```

proc IML; {} or shape()

```
proc IML;
a = {1 3 5, 2 4 6};
b = shape(1:6,2);
print a, b;
```

a		
1	3	5
2	4	6

b		
1	2	3
4	5	6

2D array

```
a=[1 3 5; 2 4 6]
2x3 Array{Int64,2}:
 1  3  5
 2  4  6
b=[1 2 3; 4 5 6]
2x3 Array{Int64,2}:
 1  2  3
 4  5  6
```

Accessing Elements

matrix[row#, col#]

```
> mat3 <- matrix(1:9,3,3)
> mat3
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> #access an element
> mat3[1,2]
[1] 4
> #access a row
> mat3[1,]
[1] 1 4 7
> #access a column
> mat3[,1]
[1] 1 2 3
> #access a submatrix
> mat3[1:2,1:3]
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
```

matrix[args]

```
>>> import numpy
>>> mat3 = numpy.matrix('1 4 7; 2 5 8; 3 6 9')
>>> mat3
matrix([[1, 4, 7],
        [2, 5, 8],
        [3, 6, 9]])
>>> #access an element
>>> mat3[0,1]
4
>>> mat3[1,2]
8
>>> #access a row
>>> mat3[0]
matrix([[1, 4, 7]])
>>> #access a column
>>> mat3[:,0]
matrix([[1],
        [2],
        [3]])
>>> #access a submatrix
>>> mat3[0:2]
matrix([[1, 4, 7],
        [2, 5, 8]])
>>> mat3[[0,1],:]
matrix([[1, 4, 7],
        [2, 5, 8]])
>>> mat3[[0,1],:][:,[0,1,2]]
matrix([[1, 4, 7],
        [2, 5, 8]])
>>> mat3[0:2][:,[0,1]]
matrix([[1, 4],
        [2, 5]])
>>> mat3[0:2:1,0:2:1]
matrix([[1, 4],
        [2, 5]])
```

matrix[row#, col#]

```
proc IML;
c = {1 4 7, 2 5 8, 3 6 9};
/*access an element*/
element = c[1,2];
/*access a row*/
row = c[1,];
/*access a column*/
col = c[,1];
/*access a submatrix*/
submat =c[1:2,1:3];
print c, element, row, col, submat;
```

c		
1	4	7
2	5	8
3	6	9

element
4

row		
1	4	7

col
1
2
3

submat		
1	4	7
2	5	8

matrix[row#, col#]

```
c=[1 4 7; 2 5 8; 3 6 9]
3x3 Array{Int64,2}:
 1  4  7
 2  5  8
 3  6  9
#access an element
c[1,2]
4
#access a row
c[1,:]
1x3 Array{Int64,2}:
 1  4  7
#access a column
c[:,1]
Vector Int64, 3
 1
 2
 3
#access a submatrix
c[1:2,1:3]
2x3 Array{Int64,2}:
 1  4  7
 2  5  8
```

Finding Transpose, Inverse, and Determinant

t(), solve(), det()

```
> mat4 <- matrix(rep(1:4,2,9),3,3, byrow=T)
> mat4
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    1    2
[3,]    3    4    1
> #transpose
> t(mat4)
     [,1] [,2] [,3]
[1,]    1    4    3
[2,]    2    1    4
[3,]    3    2    1
> #inverse
> solve(mat4)
            [,1]        [,2]        [,3]
[1,] -0.19444444  0.27777778  0.02777778
[2,]  0.05555556 -0.22222222  0.27777778
[3,]  0.36111111  0.05555556 -0.19444444
> #determinant
> det(mat4)
[1] 36
```

numpy.transpose(), numpy.linalg.inv(), numpy.linalg.det()

```
>>> import numpy
>>> mat4 = numpy.matrix('1 2 3; 4 1 2; 3 4 1')
>>> #transpose
>>> numpy.transpose(mat4)
matrix([[1, 4, 3],
        [2, 1, 4],
        [3, 2, 1]])
>>> #inverse
>>> numpy.linalg.inv(mat4)
matrix([[-0.19444444,  0.27777778,  0.02777778],
        [ 0.05555556, -0.22222222,  0.27777778],
        [ 0.36111111,  0.05555556, -0.19444444]])
>>> #determinant
>>> numpy.linalg.det(mat4)
36.0
```

matrix`, inv(), det()

```
proc IML;
d = {1 2 3, 4 1 2, 3 4 1};
/*transpose*/
transpose = d`;
/*inverse*/
inverse = inv(d);
/*determinant*/
determinant = det(d);
print d, transpose, inverse, determinant;
```

d		
1	2	3
4	1	2
3	4	1

transpose		
1	4	3
2	1	4
3	2	1

inverse		
-0.194444	0.2777778	0.0277778
0.0555556	-0.222222	0.2777778
0.3611111	0.0555556	-0.194444

determinant
36

matrix', inv(), det()

```
d=[1 2 3; 4 1 2; 3 4 1]
3x3 Array{Int64,2}:
 1  2  3
 4  1  2
 3  4  1
#transpose
d'
3x3 Array{Int64,2}:
 1  4  3
 2  1  4
 3  2  1
#inverse
inv(d)
3x3 Array{Float64,2}:
 -0.194444   0.277778   0.0277778
  0.0555556 -0.222222   0.277778
  0.361111   0.0555556 -0.194444
#determinant
det(d)
36.0
```

Addition, Subtraction, Multiplication, Outer Product, & Linear Equation

```
> #addition
> mat3 + mat4
     [,1] [,2] [,3]
[1,]    2    6   10
[2,]    6    6   10
[3,]    6   10   10
> #subtraction
> mat3 - mat4
     [,1] [,2] [,3]
[1,]    0    2    4
[2,]   -2    4    6
[3,]    0    2    8
> #multiplication
> mat3 %*% mat4
     [,1] [,2] [,3]
[1,]   38   34   18
[2,]   46   41   24
[3,]   54   48   30
> #solve system of linear equations
> A <- matrix(c(3,2,-1,2,-2,4,-1,0.5,-1),3,3,byrow=T)
> A
     [,1] [,2] [,3]
[1,]    3  2.0   -1
[2,]    2 -2.0    4
[3,]   -1  0.5   -1
> b <- c(1,-2,0)
> b
[1]  1 -2  0
> solve(A,b)
[1]  1 -2 -2
```

```
>>> #addition
>>> mat3 + mat4
matrix([[ 2,  6, 10],
        [ 6,  6, 10],
        [ 6, 10, 10]])
>>> #subtraction
>>> mat3 - mat4
matrix([[ 0,  2,  4],
        [-2,  4,  6],
        [ 0,  2,  8]])
>>> #multiplication
>>> mat3 * mat4
matrix([[38, 34, 18],
        [46, 41, 24],
        [54, 48, 30]])
>>> #solve system of linear equations
>>> A = numpy.matrix('3 2 -1; 2 -2 4; -1 0.5 -1')
>>> A
matrix([[ 3. ,  2. , -1. ],
        [ 2. , -2. ,  4. ],
        [-1. ,  0.5, -1. ]])
>>> b = numpy.array([1,-2,0])
>>> b
array([ 1, -2,  0])
>>> numpy.linalg.solve(A,b)
array([ 1., -2., -2.])
```

```
proc IML;
c = {1 4 7, 2 5 8, 3 6 9};
d = {1 2 3, 4 1 2, 3 4 1};
/*addition*/
addition = c + d;
/*subtraction*/
subtraction = c - d;
/*multiplication*/
multiplication = c * d;
/*solve system of linear equations*/
A = {3 2 -1, 2 -2 4, -1 0.5 -1};
b = {1, -2, 0};
x = inv(A) * b;
print addition, subtraction, multiplication, x;
```

addition		
2	6	10
6	6	10
6	10	10

subtraction		
0	2	4
-2	4	6
0	2	8

multiplication		
38	34	18
46	41	24
54	48	30

x
1
-2
-2

```
#addition
c + d
3x3 Array{Int64,2}:
 2   6  10
 6   6  10
 6  10  10
#subtraction
c - d
3x3 Array{Int64,2}:
  0  2  4
 -2  4  6
  0  2  8
#multiplication
c * d
3x3 Array{Int64,2}:
 38  34  18
 46  41  24
 54  48  30
#solve system of linear equations
A = [3 2 -1; 2 -2 4; -1 0.5 -1]
3x3 Array{Float64,2}:
  3.0   2.0  -1.0
  2.0  -2.0   4.0
 -1.0   0.5  -1.0
b = [1; -2; 0]
Vector Int64, 3
1
-2
0
\(A,b)
Vector Float64, 3
0.9999999999999994
-1.9999999999999984
-1.9999999999999984
```

by BULLETBYTE
bulletbyte.weebly.com