# Indexing

### Subscripts

Podemos acceder a los elementos de una matriz usando dos subíndices (renglón, columna). Recordemos que los índices en Matlab empiezan en 1.

```
A = magic(4)
A(3,2)
```

### Colon operator ":"

You can refer to the elements of a matrix with a single subscript, A(k). Matlab stores matrices as a single column of elements. This single column is composed of all of the columns from the matrix, each appended to the last.

```
A = magic(4)
A(:)    % todos los elementos de A en un único vector columna
% A(:)'
A(5)
[minVal,indice] = min(A(:))
[row,col] = ind2sub(size(A),indice)
```

### Slicing operator

You can reduce the size of expressions using the colon operator ":".  Subscript expressions involving colons refer to portions of a matrix. The expression A(1:m, n) refers to the elements in rows 1 through m of column n of matrix A. Using this notation, you can compute the sum of the third column of A more succinctly:

```
A = magic(4)
A(1:4,3)
sum(A(1:4,3))
% sum(A(3,1:4)
i = 1:4;
sum(A(i, 3))
sum(A(1:end, 3))
sum(A(:,3))
sum(A(:,end-1))
A(:,end-1) = -10
```

### Multiplication

```
A = [1,2,-1;2,1,-2;-3,1,1]
x = [3;1;2]
A*x
suma=0;
for j=1:3
    suma=suma+A(1,j)*x(j);
end
suma
A(1,1:3)*x(1:3)
A(1,:)*x
```

```
suma=0;
for j=1:1:3
    suma=suma+A(3,j)*x(j);
end
suma
A(3,:)*x
A(2,2:3)*x(2:3)
```

## Row exchange

### Select some rows with a "list" of indexes

```
A;
Temp = A
```

```
Temp = 4×4
    16     2   -10    13
     5    11   -10     8
     9     7   -10    12
     4    14   -10     1
```

```
Temp([2,3],:) = Temp([3,2],:)
```

```
Temp = 4×4
    16     2   -10    13
     9     7   -10    12
     5    11   -10     8
     4    14   -10     1
```

## Permutation matrix

A permutation matrix (transposition matrix) is an identity matrix with rows and columns interchanged. It consistes of zeros and ones, and each row and column has exactly one nonzero element.

A permutation matrix when used to multiply another matrix A, results in permuting the rows (when pre-multiplying, i.e., PA) or columns (when post-multiplying, AP) of the matrix A.

### Rows permutation PA

```
A = magic(4)
```

```
A = 4×4
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
```

```
P = [0,0,0,1;1,0,0,0;0,0,1,0;0,1,0,0]
```

```
P = 4×4
     0     0     0     1
     1     0     0     0
     0     0     1     0
     0     1     0     0
```

```
PA=P*A  % permutes rows
```

```
PA = 4×4
```

```
     4    14    15     1
    16     2     3    13
     9     7     6    12
     5    11    10     8
```

The resulting matrix has the fourth row of A as its first row, the first row of A as its second row, and so on.

```
r=[4,1,3,2]
```

```
r = 1×4
     4     1     3     2
```

```
A(r,:)
```

```
ans = 4×4
     4    14    15     1
    16     2     3    13
     9     7     6    12
     5    11    10     8
```

P*A and A(r,:) are equal.

The P*A notation is closer to traditional mathematics, PA, while the A(r,:) notation is faster and uses less memory.

**Columns permutation AP**

```
A = magic(4)
P = [0,0,0,1;1,0,0,0;0,0,1,0;0,1,0,0]
P=A*P  % permutes columns
```

The resulting matrix has the second column of A as its first column, the fourth column of A as its second column, and so on.

A*P and A(:,c) produce the same permutation of the columns of A.

```
c = [2,4,3,1]
A(:,c)
```

A*P' and A(:,r) produce the same permutation of the columns of A.

```
A*P'
A(:,r)
```

## Logical (boolean) indexing

https://www.mathworks.com/help/matlab/math/matrix-indexing.html

https://www.mathworks.com/help/matlab/matlab_prog/vectorization.html

A logical array index designates the elements of an array A based on their position in the indexing array, B. In this masking type of operation, every true element in the indexing array is treated as a positional index into the array being accessed.

In the following example, B is a matrix of logical ones and zeros. The position of these elements in B determines which elements of A are designated by the expression A(B)

```
A = magic(4)
B = isprime(A)
A(~B) = 0
```

## Indexing using vectors

```
x = 1:10
```

Mask: indices of elements satisfing a condition

```
v = x>6
```

Elements satisfing a condition

```
x(v)
x(x>6)
```

### Ejemplos

```
x = randperm(20)
target = 5;
x < target
ind = find(x < target)  % returns the linear indices of nonzero elements
xTarget = x(ind)        % elements matching the criteria
iseven = @(x) ~logical(rem(x,2))
compoundCondInd = (x < target) & iseven(x)
x(compoundCondInd)
```

```
N = 10;
Mx = 2:2:2*N
r = randi(N,1,N) - N/2
indices1 = find(r>0)
vectorLogico = r>0
indicesMx = Mx(indices1)
indicesMx = Mx(vectorLogico)
```

Substract 3 from each element which is greater than 3

```
x(x>3)=x(x>3)-3;
```

Determine the number of items > 8

```
M = magic(5)
M>8
sum(M>8)
sum(sum(M>8))
sum(M(:)>8)
```

## Vectorization

```
for i = 1:10
    for j = 1:10
        r(i,j) = sqrt(i^2 + j^2);
    end
end
[i,j] = meshgrid(1:10, 1:10);
r = sqrt(i.^2 + j.^2)
```

bsxfun applies the element-wise binary operation specified by the function handle to two arrays.

Square root of sum of squares (hypotenuse)

```
r = bsxfun(@hypot,i,j)
```