ESERCITAZIONI
MATLAB – Nozioni fondamentali

MathWorks®

Matrici e array

Funzioni

Creazione e abbinamento di array

`zeros`	Creare array di tutti zeri
`ones`	Creare array di tutti numeri uno
`rand`	Numeri casuali distribuiti uniformemente
`true`	Logical 1 (true)
`false`	Logical 0 (false)
`eye`	Identity matrix
`diag`	Create diagonal matrix or get diagonal elements of matrix
`blkdiag`	Block diagonal matrix
`cat`	Concatenare gli array
`horzcat`	Concatenate arrays horizontally
`vertcat`	Concatenate arrays vertically
`repelem`	Repeat copies of array elements
`repmat`	Ripetere le copie dell'array
`combinations`	Generate all element combinations of arrays *(Da R2023a)*

https://it.mathworks.com/videos/working-with-arrays-in-matlab-101637.html

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Funzioni

Creazione di griglie

`linspace`	Generare un vettore a spaziatura lineare
`logspace`	Generare un vettore a spaziatura logaritmica
`freqspace`	Frequency spacing for frequency response
`meshgrid`	Griglie bidimensionali e tridimensionali
`ndgrid`	Rectangular grid in N-D space

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Funzioni

Determinazione della dimensione, della forma e dell'ordine

`length`	Lunghezza della dimensione più grande dell'array
`size`	Dimensione dell'array
`ndims`	Number of array dimensions
`numel`	Numero degli elementi dell'array
`isscalar`	Determine whether input is scalar
`isvector`	Determine whether input is vector
`ismatrix`	Determine whether input is matrix
`isrow`	Determine if input is row vector
`iscolumn`	Determine if input is column vector
`isempty`	Determinare se un array è vuoto
`issorted`	Determine if array is sorted
`issortedrows`	Determine if matrix or table rows are sorted
`isuniform`	Determine if vector is uniformly spaced *(Da R2022b)*

ESERCITAZIONI
MATLAB – Nozioni fondamentali

MathWorks®

Matrici e array

Funzioni

Ridimensionamento, rimodellamento e riorganizzazione

Ridimensionamento

`head`	Get top rows of array or table
`tail`	Get bottom rows of array or table
`resize`	Resize data by adding or removing elements *(Da R2023b)*
`paddata`	Pad data by adding elements *(Da R2023b)*
`trimdata`	Trim data by removing elements *(Da R2023b)*

Rimodellamento

`permute`	Permutare le dimensioni degli array
`ipermute`	Inverse permute array dimensions
`shiftdim`	Shift array dimensions
`reshape`	Rimodellare l'array riorganizzando gli elementi esistenti
`squeeze`	Rimuovere le dimensioni di lunghezza 1

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Funzioni

Riorganizzazione

`sort`	Ordinare gli elementi dell'array
`sortrows`	Sort rows of matrix or table
`flip`	Ordine di capovolgimento degli elementi
`fliplr`	Capovolgere l'array da sinistra a destra
`flipud`	Flip array up to down
`rot90`	Rotate array 90 degrees
`transpose`	Trasporre un vettore o una matrice
`ctranspose`	Complex conjugate transpose
`circshift`	Shift array circularly

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Funzioni

Indicizzazione

`colon`	Vector creation, array subscripting, and `for`-loop iteration
`end`	Terminate block of code or indicate last array index
`ind2sub`	Convert linear indices to subscripts
`sub2ind`	Convert subscripts to linear indices

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Creating, Concatenating, and Expanding Matrices

The most basic MATLAB® data structure is the matrix. A matrix is a two-dimensional, rectangular array of data elements arranged in rows and columns. The elements can be numbers, logical values (`true` or `false`), dates and times, strings, `categorical` values, or some other MATLAB data type.

Even a single number is stored as a matrix. For example, a variable containing the value 100 is stored as a 1-by-1 matrix of type `double`.

```
A = 100;
whos A
```

```
  Name      Size            Bytes  Class     Attributes

  A         1x1                 8  double
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Creating, Concatenating, and Expanding Matrices

Constructing a Matrix of Data

If you have a specific set of data, you can arrange the elements in a matrix using square brackets. A single row of data has spaces or commas in between the elements, and a semicolon separates the rows. For example, create a single row of four numeric elements. The size of the resulting matrix is 1-by-4 because it has one row and four columns. A matrix of this shape is often referred to as a row vector.

```
A = [12 62 93 -8]
```

```
A = 1×4

    12    62    93    -8
```

```
sz = size(A)
```

```
sz = 1×2

     1     4
```

Now create a matrix with the same numbers, but arrange them in two rows. This matrix has two rows and two columns.

```
A = [12 62; 93 -8]
```

```
A = 2×2

    12    62
    93    -8
```

```
sz = size(A)
```

```
sz = 1×2

     2     2
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

MathWorks®

Matrici e array

Argomenti

Creating, Concatenating, and Expanding Matrices

Specialized Matrix Functions

MATLAB has many functions that help create matrices with certain values or a particular structure. For example, the `zeros` and `ones` functions create matrices of all zeros or all ones. The first and second arguments of these functions are the number of rows and number of columns of the matrix, respectively.

```
A = zeros(3,2)
```

```
A = 3×2

     0     0
     0     0
     0     0
```

```
B = ones(2,4)
```

```
B = 2×4

     1     1     1     1
     1     1     1     1
```

The `diag` function places the input elements on the diagonal of a matrix. For example, create a row vector `A` containing four elements. Then, create a 4-by-4 matrix whose diagonal elements are the elements of `A`.

```
A = [12 62 93 -8];
B = diag(A)
```

```
B = 4×4

    12     0     0     0
     0    62     0     0
     0     0    93     0
     0     0     0    -8
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Creating, Concatenating, and Expanding Matrices

Specialized Matrix Functions

The `createArray` function *(since R2024a)* can return arrays of almost any MATLAB data type. The function uses the same syntax as `zeros` and `ones` to define the size of the array, but it also offers several options for specifying the contents of the array. For example, you can specify the fill value.

```
C = createArray(2,3,FillValue=duration(1,15,0))
```

```
C = 2x3 duration
   01:15:00   01:15:00   01:15:00
   01:15:00   01:15:00   01:15:00
```

Concatenating Matrices

You can also use square brackets to append existing matrices. This way of creating a matrix is called *concatenation*. For example, concatenate two row vectors to make an even longer row vector.

```
A = ones(1,4);
B = zeros(1,4);
C = [A B]
```

```
C = 1×8

     1     1     1     1     0     0     0     0
```

To arrange `A` and `B` as two rows of a matrix, use the semicolon.

```
D = [A; B]
```

```
D = 2×4

     1     1     1     1
     0     0     0     0
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Creating, Concatenating, and Expanding Matrices

Concatenating Matrices

To concatenate several matrices, they must have compatible sizes. In other words, when you concatenate matrices horizontally, they must have the same number of rows. When you concatenate them vertically, they must have the same number of columns.

For example, create two matrices that both have two rows. Horizontally append the second matrix to the first by using square brackets.

```
A = ones(2,3)
```

```
A = 2×3

     1     1     1
     1     1     1
```

```
B = zeros(2,2)
```

```
B = 2×2

     0     0
     0     0
```

```
C = [A B]
```

```
C = 2×5

     1     1     1     0     0
     1     1     1     0     0
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Creating, Concatenating, and Expanding Matrices

Concatenating Matrices

An alternative way to concatenate compatible matrices is to use concatenation functions, such as `horzcat`, `vertcat`, and `cat`. Horizontally append the second matrix to the first by using `horzcat`.

```
D = horzcat(A,B)
```

```
D = 2×5

     1     1     1     0     0
     1     1     1     0     0
```

Generating a Numeric Sequence

The `colon` is a handy way to create matrices whose elements are sequential and evenly spaced. For example, create a row vector whose elements are the integers from 1 to 10.

```
A = 1:10
```

```
A = 1×10

     1     2     3     4     5     6     7     8     9    10
```

You can use the colon operator to create a sequence of numbers within any range, incremented by one.

```
A = -2.5:2.5
```

```
A = 1×6

   -2.5000   -1.5000   -0.5000    0.5000    1.5000    2.5000
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Creating, Concatenating, and Expanding Matrices

Generating a Numeric Sequence

To change the value of the sequence increment, specify the increment value in between the starting and ending range values, separated by colons.

```
A = 0:2:10
```

```
A = 1×6

     0     2     4     6     8    10
```

To decrement, use a negative number.

```
A = 6:-1:0
```

```
A = 1×7

     6     5     4     3     2     1     0
```

You can also increment by noninteger values. If an increment value does not evenly partition the specified range, MATLAB automatically ends the sequence at the last value it can reach before exceeding the range.

```
A = 1:0.2:2.1
```

```
A = 1×6

    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Creating, Concatenating, and Expanding Matrices

Expanding a Matrix

You can add one or more elements to a matrix by placing them outside of the existing row and column index boundaries. MATLAB automatically pads the matrix with zeros to keep it rectangular. For example, create a 2-by-3 matrix and add an additional row and column to it by inserting an element in the (3,4) position.

```
A = [10  20  30; 60  70  80]
```

```
A = 2×3

    10    20    30
    60    70    80
```

```
A(3,4) = 1
```

```
A = 3×4

    10    20    30     0
    60    70    80     0
     0     0     0     1
```

You can also expand the size by inserting a new matrix outside of the existing index ranges.

```
A(4:5,5:6) = [2 3; 4 5]
```

```
A = 5×6

    10    20    30     0     0     0
    60    70    80     0     0     0
     0     0     0     1     0     0
     0     0     0     0     2     3
     0     0     0     0     4     5
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

MathWorks®

Matrici e array

Argomenti

Creating, Concatenating, and Expanding Matrices

Expanding a Matrix

To expand the size of a matrix repeatedly, such as within a `for` loop, it is a best practice to preallocate space for the largest matrix you anticipate creating. Without preallocation, MATLAB has to allocate memory every time the size increases, slowing down operations. For example, preallocate a matrix that holds up to 10,000 rows and 10,000 columns by initializing its elements to zero.

```
A = zeros(10000,10000);
```

If you need to preallocate additional elements later, you can expand it by assigning outside of the matrix index ranges or concatenate another preallocated matrix to `A`.

Empty Arrays

An empty array in MATLAB is an array with at least one dimension length equal to zero. Empty arrays are useful for representing the concept of "nothing" programmatically. For example, suppose you want to find all elements of a vector that are less than 0, but there are none. The `find` function returns an empty vector of indices, indicating that it did not find any elements less than 0.

```
A = [1 2 3 4];
ind = find(A<0)
```

```
ind =

  1x0 empty double row vector
```

Many algorithms contain function calls that can return empty arrays. It is often useful to allow empty arrays to flow through these algorithms as function arguments instead of handling them as a special case. If you do need to customize empty array handling, you can check for them using the `isempty` function.

```
TF = isempty(ind)
```

```
TF = logical
   1
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Indicizzazione di array

In MATLAB® sono disponibili tre approcci principali per accedere agli elementi di un array in base alla loro posizione (indice) nell'array. Questi approcci sono l'indicizzazione per posizione, l'indicizzazione lineare e l'indicizzazione logica.

Indicizzazione con le posizioni degli elementi

Il modo più comune è quello di specificare gli indici degli elementi in modo esplicito. Ad esempio, per accedere a un singolo elemento di una matrice, specificare il numero di riga seguito dal numero di colonna dell'elemento.

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

```
A = 4×4

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

```
e = A(3,2)
```

```
e = 10
```

`e` è l'elemento nella posizione 3,2 (terza riga, seconda colonna) di `A`.

È inoltre possibile fare riferimento a più elementi alla volta, specificando i loro indici in un vettore. Ad esempio, accedere al primo e al terzo elemento della seconda riga di `A`.

```
r = A(2,[1 3])
```

```
r = 1×2

     5     7
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

MathWorks®

Matrici e array

Argomenti

Indicizzazione di array

Indicizzazione con le posizioni degli elementi

Per accedere agli elementi di un intervallo di righe o di colonne, utilizzare il `colon`. Ad esempio, accedere agli elementi dalla prima alla terza riga e dalla seconda alla quarta colonna di `A`.

```
r = A(1:3,2:4)
```

```
r = 3×3

     2     3     4
     6     7     8
    10    11    12
```

Un modo alternativo per calcolare `r` è quello di utilizzare la parola chiave `end` per specificare dalla seconda colonna fino all'ultima. Questo approccio consente di specificare l'ultima colonna senza sapere esattamente quante colonne sono presenti in `A`.

```
r = A(1:3,2:end)
```

```
r = 3×3

     2     3     4
     6     7     8
    10    11    12
```

Se si desidera accedere a tutte le righe o le colonne, utilizzare l'operatore dei due punti da solo. Ad esempio, restituire tutta la terza colonna di `A`.

```
r = A(:,3)
```

```
r = 4×1

     3
     7
    11
    15
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Indicizzazione di array

Indicizzazione con un indice singolo

Un altro metodo per accedere agli elementi di un array è quello di utilizzare un solo indice, indipendentemente dalla grandezza o dalle dimensioni dell'array. Questo metodo è noto come *indicizzazione lineare*. Sebbene MATLAB visualizzi gli array in base alle loro grandezze e forme definite, in realtà vengono archiviati nella memoria come una singola colonna di elementi. Un modo efficace per visualizzare questo concetto è con una matrice. Sebbene il seguente array sia visualizzato come una matrice 3x3, MATLAB lo memorizza come una singola colonna composta dalle colonne di A aggiunte una dopo l'altra. Il vettore memorizzato contiene la sequenza degli elementi 12, 45, 33, 36, 29, 25, 91, 48, 11, che può essere visualizzata utilizzando un due punti singolo.

```
A = [12 36 91; 45 29 48; 33 25 11]
```

```
A = 3×3

    12    36    91
    45    29    48
    33    25    11
```

```
Alinear = A(:)
```

```
Alinear = 9×1

    12
    45
    33
    36
    29
    25
    91
    48
    11
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Indicizzazione di array

Indicizzazione con un indice singolo

Ad esempio, l'elemento 3,2 di A è 25 ed è possibile accedervi utilizzando la sintassi `A(3,2)`. È inoltre possibile accedere a questo elemento utilizzando la sintassi `A(6)`, poiché 25 è il sesto elemento della sequenza vettoriale memorizzata.

```
e = A(3,2)
```

```
e = 25
```

```
elinear = A(6)
```

```
elinear = 25
```

Sebbene l'indicizzazione lineare possa essere meno intuitiva dal punto di vista visivo, può essere molto efficace per eseguire alcuni calcoli che non dipendono dalla grandezza o dalla forma dell'array. Ad esempio, è possibile sommare facilmente tutti gli elementi di A senza dover fornire un secondo argomento alla funzione `sum`.

```
s = sum(A(:))
```

```
s = 330
```

Le funzioni `sub2ind` e `ind2sub` aiutano a convertire gli indici originali dell'array nella loro versione lineare. Ad esempio, calcolare l'indice lineare dell'elemento 3,2 di A.

```
linearidx = sub2ind(size(A),3,2)
```

```
linearidx = 6
```

Riconvertire l'indice lineare nella sua forma a righe e colonne.

```
[row,col] = ind2sub(size(A),6)
```

```
row = 3
```

```
col = 2
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

MathWorks®

Matrici e array

Argomenti

Indicizzazione di array

Indicizzazione con valori logici

L'utilizzo degli indicatori logici true e false è un altro modo utile per indicizzare gli array, in particolare quando si lavora con le dichiarazioni condizionali. Ad esempio, si supponga di voler sapere se gli elementi di una matrice A sono minori dei corrispondenti elementi di un'altra matrice B. L'operatore minore-di restituisce un array logico i cui elementi sono 1 quando un elemento in A è più piccolo del corrispondente elemento in B.

```
A = [1 2 6; 4 3 6]
```

```
A = 2×3

     1     2     6
     4     3     6
```

```
B = [0 3 7; 3 7 5]
```

```
B = 2×3

     0     3     7
     3     7     5
```

```
ind = A<B
```

```
ind = 2×3 logical array

   0   1   1
   0   1   0
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Indicizzazione di array

Indicizzazione con valori logici

Ora che si conosce la posizione degli elementi che soddisfano la condizione, è possibile esaminare i singoli valori utilizzando `ind` come array di indici. MATLAB abbina le posizioni del valore 1 in `ind` agli elementi corrispondenti di A e B ed elenca i loro valori in un vettore colonna.

```
Avals = A(ind)
```

```
Avals = 3×1

     2
     3
     6
```

```
Bvals = B(ind)
```

```
Bvals = 3×1

     3
     7
     7
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Indicizzazione di array

Indicizzazione con valori logici

Le funzioni "is" di MATLAB restituiscono inoltre array logici che indicano quali elementi di input soddisfano una determinata condizione. Ad esempio, verificare quali elementi di un vettore `string` sono mancanti utilizzando la funzione `ismissing`.

```
str = ["A" "B" missing "D" "E" missing];
ind = ismissing(str)
```

```
ind = 1x6 logical array

   0   0   1   0   0   1
```

Si supponga di voler trovare i valori degli elementi che *non* sono mancanti. A tale scopo, utilizzare l'operatore ~ con il vettore indice `ind`.

```
strvals = str(~ind)
```

```
strvals = 1x4 string
    "A"    "B"    "D"    "E"
```

Per ulteriori esempi sull'utilizzo dell'indicizzazione logica, vedere Find Array Elements That Meet a Condition.

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Indicizzazione di array

Find Array Elements That Meet a Condition

This example shows how to filter the elements of an array by applying conditions to the array. For instance, you can examine the even elements in a matrix, find the location of all 0s in a multidimensional array, or replace NaN values in data. You can perform these tasks using a combination of the relational and logical operators. The relational operators (>, <, >=, <=, ==, ~=) impose conditions on the array, and you can apply multiple conditions by connecting them with the logical operators and, or, and not, respectively denoted by the symbols &, |, and ~.

Apply a Single Condition

To apply a single condition, start by creating a 5-by-5 matrix that contains random integers between 1 and 15. Reset the random number generator to the default state for reproducibility.

```
rng default
A = randi(15,5)
```

```
A = 5×5

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

Use the relational *less than* operator, <, to determine which elements of A are less than 9. Store the result in B.

```
B = A < 9
```

```
B = 5×5 logical array

   0   1   1   1   0
   0   1   0   1   1
   1   0   0   0   0
   0   0   1   0   0
   0   0   0   0   0
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

MathWorks®

Matrici e array

Argomenti

Indicizzazione di array

Find Array Elements That Meet a Condition

Apply a Single Condition

The result is a logical matrix. Each value in `B` represents a logical `1` (`true`) or logical `0` (`false`) state to indicate whether the corresponding element of `A` fulfills the condition `A < 9`. For example, `A(1,1)` is 13, so `B(1,1)` is logical `0` (`false`). However, `A(1,2)` is 2, so `B(1,2)` is logical `1` (`true`).

Although `B` contains information about *which* elements in `A` are less than 9, it doesn't tell you what their *values* are. Rather than comparing the two matrices element by element, you can use `B` to index into `A`.

```
A(B)
```

```
ans = 8×1

     2
     2
     5
     3
     8
     3
     7
     1
```

The result is a column vector of the elements in `A` that are less than 9. Since `B` is a logical matrix, this operation is called **logical indexing**. In this case, the logical array being used as an index is the same size as the other array, but this is not a requirement. For more information, see Array Indexing.

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Indicizzazione di array

Find Array Elements That Meet a Condition

Apply a Single Condition

Some problems require information about the *locations* of the array elements that meet a condition rather than their actual values. In this example, you can use the `find` function to locate all of the elements in `A` less than 9.

```
I = find(A < 9)
```

```
I = 8×1

     3
     6
     7
    11
    14
    16
    17
    22
```

The result is a column vector of linear indices. Each index describes the location of an element in `A` that is less than 9, so in practice `A(I)` returns the same result as `A(B)`. The difference is that `A(B)` uses logical indexing, whereas `A(I)` uses linear indexing.

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Indicizzazione di array

Find Array Elements That Meet a Condition

Apply Multiple Conditions

You can use the logical `and`, `or`, and `not` operators to apply any number of conditions to an array; the number of conditions is not limited to one or two.

First, use the logical `and` operator, denoted `&`, to specify two conditions: the elements must be **less than 9** and **greater than 2**. Specify the conditions as a logical index to view the elements that satisfy both conditions.

```
A(A<9 & A>2)
```

```
ans = 5×1

     5
     3
     8
     3
     7
```

The result is a list of the elements in `A` that satisfy both conditions. Be sure to specify each condition with a separate statement connected by a logical operator. For example, you cannot specify the conditions above by `A(2<A<9)`, since it evaluates to `A(2<A | A<9)`.

Next, find the elements in `A` that are **less than 9** and **even numbered**.

```
A(A<9 & ~mod(A,2))
```

```
ans = 3×1

     2
     2
     8
```

The result is a list of all even elements in `A` that are less than 9. The use of the logical NOT operator, `~`, converts the matrix `mod(A,2)` into a logical matrix, with a value of logical `1` (`true`) located where an element is evenly divisible by 2.

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Indicizzazione di array

Find Array Elements That Meet a Condition

Apply Multiple Conditions

Finally, find the elements in `A` that are **less than 9** and **even numbered** and **not equal to 2**.

```
A(A<9 & ~mod(A,2) & A~=2)
```

```
ans = 8
```

The result, 8, is even, less than 9, and not equal to 2. It is the only element in `A` that satisfies all three conditions.

Use the `find` function to get the index of the element equal to 8 that satisfies the conditions.

```
find(A<9 & ~mod(A,2) & A~=2)
```

```
ans = 14
```

The result indicates that `A(14) = 8`.

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Indicizzazione di array

Find Array Elements That Meet a Condition

Replace Values That Meet a Condition

Sometimes it is useful to simultaneously change the values of several existing array elements. Use logical indexing with a simple assignment statement to replace the values in an array that meet a condition.

Replace all values in A that are greater than 10 with the number 10.

```
A(A>10) = 10
```

```
A = 5×5

    10     2     3     3    10
    10     5    10     7     1
     2     9    10    10    10
    10    10     8    10    10
    10    10    10    10    10
```

Next, replace all values in A that are not equal to 10 with a NaN value.

```
A(A~=10) = NaN
```

```
A = 5×5

    10   NaN   NaN   NaN    10
    10   NaN    10   NaN   NaN
   NaN   NaN    10    10    10
    10    10   NaN    10    10
    10    10    10    10    10
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Matrici e array

Argomenti

Indicizzazione di array

Find Array Elements That Meet a Condition

Replace Values That Meet a Condition

Lastly, replace all of the `NaN` values in `A` with zeros and apply the logical NOT operator, `~A`.

```
A(isnan(A)) = 0;
C = ~A
```

```
C = 5x5 logical array

   0   1   1   1   0
   0   1   0   1   1
   1   1   0   0   0
   0   0   1   0   0
   0   0   0   0   0
```

The resulting matrix has values of logical `1` (`true`) in place of the `NaN` values, and logical `0` (`false`) in place of the 10s. The logical NOT operation, `~A`, converts the numeric array into a logical array such that `A&C` returns a matrix of logical `0` (`false`) values and `A|C` returns a matrix of logical `1` (`true`) values.

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Alcune operazioni sulle matrici

det

Matrix determinant

Syntax

```
d = det(A)
```

Description

`d = det(A)` returns the determinant of square matrix `A`.

Examples

Calculate Determinant of Matrix

Create a 3-by-3 square matrix, `A`.

```
A = [1 -2 4; -5 2 0; 1 0 3]
```

```
A = 3×3

     1    -2     4
    -5     2     0
     1     0     3
```

Calculate the determinant of `A`.

```
d = det(A)
```

```
d = -32
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Alcune operazioni sulle matrici

inv

Inverso delle matrici

Sintassi

```
Y = inv(X)
```

Descrizione

`Y = inv(X)` l'inverso della matrice quadrata `X`.

- `X^(-1)` equivale a `inv(X)`.
- `x = A\b` è calcolato in modo diverso rispetto a `x = inv(A)*b` ed è consigliato per la risoluzione di sistemi di equazioni lineari.

Esempi

Matrice inversa

Calcolare l'inverso di una matrice 3x3.

```
X = [1 0 2; -1 5 0; 0 3 -9]
```

```
X = 3×3

     1     0     2
    -1     5     0
     0     3    -9
```

```
Y = inv(X)
```

```
Y = 3×3

    0.8824   -0.1176    0.1961
    0.1765    0.1765    0.0392
    0.0588    0.0588   -0.0980
```

ESERCITAZIONI
MATLAB – Nozioni fondamentali

Alcune operazioni sulle matrici

eig

Eigenvalues

Examples

Eigenvalues of Matrix

Use `gallery` to create a symmetric positive definite matrix.

```
A = gallery("lehmer",4)
```

```
A = 4×4

    1.0000    0.5000    0.3333    0.2500
    0.5000    1.0000    0.6667    0.5000
    0.3333    0.6667    1.0000    0.7500
    0.2500    0.5000    0.7500    1.0000
```

Calculate the eigenvalues of `A`. The result is a column vector.

```
e = eig(A)
```

```
e = 4×1

    0.2078
    0.4078
    0.8482
    2.5362
```

Riferimenti Bibliografici

[1] https://it.mathworks.com