Matrici e Operazioni Vettoriali

Informatica, AA 2021/2022

Francesco Trovò

15 Ottobre 2021

https://trovo.faculty.polimi.it/

francesco1.trovo@polimi.it

Esercizio di Stretching Mentale

Scrivere un programma che determina se una parola è palindoma

Matrici: Array Bidimensionali

Le Matrici

Le matrici sono array 2-D

Hanno quindi due indici:

- L'indice di riga (il primo)
- L'indice di colonna (il secondo)

Esempio:

```
A = 16     2     3    13
     5    11    10     8
     9     7     6    12
```

Le Matrici

Le matrici sono array 2-D

Hanno quindi due indici:

- L'indice di riga (il primo)
- L'indice di colonna (il secondo)

Esempio:

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

Elemento alla riga 2 colonna 3

```
>> A(2, 3)

ans = 10
```

Creazione di Matrici

Le matrici vengono definite **affiancando vettori di dimensioni compatibili**

- Usiamo sempre gli operatori **,** (spazio) e **;** (vai a capo)
- L'operazione di **trasposizione** inverte le righe e le colonne della matrice

Es:

```
>> a = [1 , 2 ; 3 ,4 ]
a =
     1     2
     3     4
```

```
a' =
     1     3
     2     4
```

Le dimensioni di una matrice

Per scorrere una matrice è spesso necessario conoscere le sue dimensioni

Il comando size(A, dim) restituisce il numero di elementi di A lungo la dimensione dim

```
A = 16     2     3    13
     5    11    10     8
     9     7     6    12
righe = size(A, 1);
colonne = size(A, 2);
```

Le Matrici: operatore CAT

La concatenazione dei vettori avviene mediante operatore **CAT** che richiede **dimensioni consistenti** dei vettori

```
>> a = [1 : 3]
a =
     1     2     3

>>  b = [4; 5; 6]
b =
     4
     5
     6
```

```
>> A = [a; b]
Error using vertcat
CAT arguments dimensions are
not consistent.
>> A =[a, b]
Error using horzcat
CAT arguments dimensions are
not consistent.
>> A =[a; b']
A =
     1     2     3
     4     5     6
```

Accedere agli Elementi di una Matrice

Per accedere agli elementi di una matrice occorre specificare un valore per ogni indice

`nomeMatrice(indice1, indice2)`

Seleziona il valore alla riga **`indice1`** colonna **`indice2`** nella variabile **`nomeMatrice`**

Es

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

```
>> A(2, 3)
ans =
     6
>> A(3,5)
Index exceeds
matrix dimensions.
```

Memorizzazione degli Array

Gli array vengono salvati linearmente in memoria.

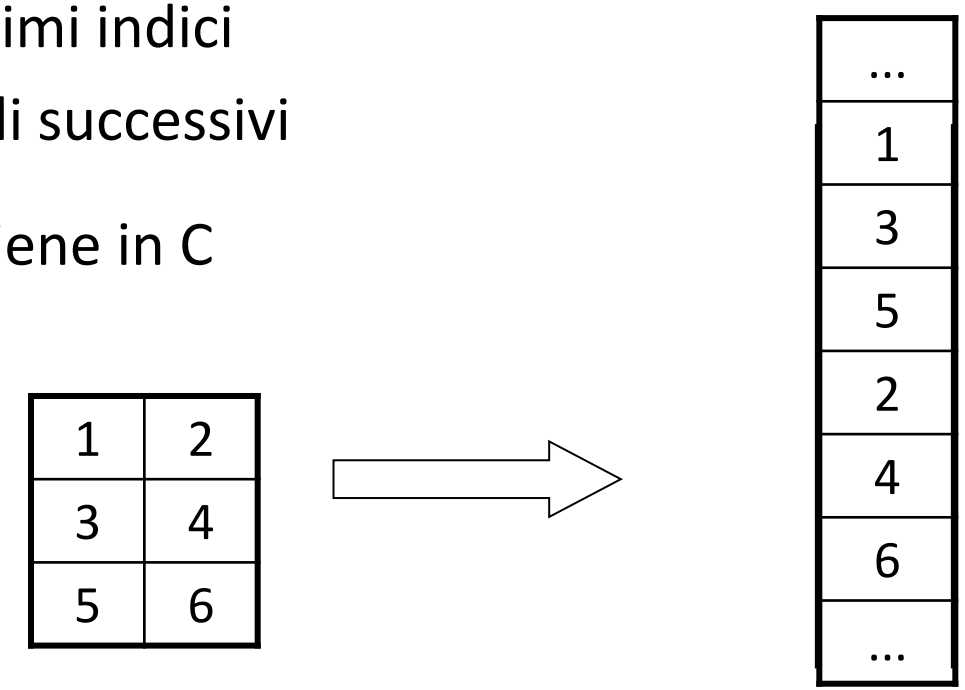
In particolare le matrici sono memorizzate

- per colonna: colonna 1, poi colonna 2, 3, etc.
- ogni colonna memorizzata per indici di riga crescenti

Array memorizzati in forma lineare nella RAM variando

- più velocemente i primi indici
- più lentamente quelli successivi

NB: opposto a quanto avviene in C



Array: forma *linearizzata*

Si può accedere a un array a più dimensioni come se ne avesse una sola

Usando un unico indice si segue l'ordine della memorizzazione

```
>> a = [1 2 3; 4 5 6; 7 8 9; 10 11 12]
a =
     1     2     3
     4     5     6
     7     8     9
    10    11    12
>> a(3, 2)
ans =
     8
>> a(10)
ans =
     6
```

TODO: Esercizio su Matrici

Si scriva un programma che prende in ingresso una matrice quadrata e controlla che sia simmetrica.

Una matrice è simmetrica se per ogni elemento vale la seguente proprietà: L'elemento alla riga i, colonna j coincide con l'elemento alla riga j colonna i

Es di matrice simmetrica

1	**12**	**1**
12	**0**	**3**
1	**3**	**23**

Array Multidimensionali

È possibile specificare una terza (quarta, quinta…) dimensione lungo la quale indicizzare un array

Ad esempio le immagini a colori sono definite con tre piani colore (RGB), quindi

- un'immagine a colori 10 Mpixels, aspect ratio (3:4) richiede una matrice 3D di 2736 x 3648 x 3 elementi
- 10 sec di video full HD (1080 x 768) a 24fps richiede una matrice 4D di 1080 x 768 x 3 x (10 x 24) elementi

Altre Operazioni sugli Array

Subarray

Cancellazione elementi

Esercizietto di riscaldamento

Dato un vettore **`v`** di interi ed un vettore **`indx`** di posizioni ammissibili per **`v`** copiare in un secondo vettore **`t`** tutti i valori di **`v`** che compaiono nelle posizioni **`indx`**

Domanda: quali sono le posizioni ammissibili per **`v`**?

- Interi positivi (**>0**) che sono minori o uguali alla lunghezza di **`v`**

Esercizietto di riscaldamento

Dato un vettore **v** di interi ed un vettore **indx** di posizioni ammissibili per **v** copiare in un secondo vettore **t** tutti i valori di **v** che compaiono nelle posizioni **indx**

```
t = [];
for ii = indx
    t = [t, v(ii)];
end
```

Nella pratica…

```
v = [10 : 2 : 16]
indx = [2, 3]


t = [];
for ii = indx
    t = [t, v(ii)];
end
disp(t);
```

oppure

```
t = [];
ii = 1;
while (ii <= length(indx))
    t(ii) = v(indx(ii));
    ii = ii + 1;
end
disp(t);
```

```
>> t =
    12    14
```

Oppure..

```
>> v(indx)

     12     14
```

Sottoarray (un vettore come indice di un vettore)

Si denota un sottoinsieme di un array usando vettori per valori degli indici

`nomeVettore(vettoreIndici)`

restituisce un vettore che comprende gli elementi di **`nomeVettore`** che compaiono nelle posizioni **`vettoreIndici`**

Estende il modo l'accesso ad un singolo elemento

`nomeVettore(indice)`

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v = 6   8   4   2   4   5   1   3

>> v([1 4 7])
ans =     6     2     1

>> v(2:2:6)
ans =     8     2     5
```

primo, quarto settimo elemento

2:2:6 è il vettore [2, 4, 6]: indica secondo, quarto, sesto elemento

Sottovettori definiti da vettori di indici

Quindi, la notazione

```
a(v)
```

corrisponde a

```
[a(v(1)), a(v(2)),…, a(v(end))]
```

Attenzione che i valori di **`v`** devono essere interi positivi e minori delle dimensioni di **`a`**, devono essere indici validi.

La Keyword `end`

All'interno di **`vettoreIndici`** si può usare la keyword **`end`** che assume il valore dell'ultimo indice disponibile su una specifica dimensione di **`nomeVettore`.**

In questo modo non occorre conoscere le dimensioni del vettore

Esempi

```
>> a = [10 : 10: 100]
a =
    10   20   30   40   50   60   70   80   90  100
```

Toglie l'ultimo elemento

```
>> b = a(1 : end - 1)
b =
    10   20   30   40   50   60   70   80   90
```

Legge il vettore dall'ultimo elemento al primo

```
>> b = a(end : -1 : 1)
b =
   100   90   80   70   60   50   40   30   20   10
```

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =     6    8    4    2    4    5    1    3
>> v([1 4 7])


>> v(2:2:6)


>> v(3:end-2)


>> v(v)


>> v([1, 1, 1, 2, end])
```

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =     6    8    4    2    4    5    1    3
>> v([1 4 7])
ans =      6      2      1
```

primo, quarto settimo elemento

```
>> v(2:2:6)
```

```
>> v(3:end-2)
```

```
>> v(v)
```

```
>> v([1, 1, 1, 2, end])
```

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =     6    8    4    2    4    5    1    3
>> v([1 4 7])
ans =      6      2      1

>> v(2:2:6)
ans =      8      2      5

>> v(3:end-2)



>> v(v)



>> v([1, 1, 1, 2, end])
```

primo, quarto settimo elemento

2:2:6 è il vettore [2, 4, 6]: indica secondo, quarto, sesto elemento

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =     6    8    4    2    4    5    1    3
>> v([1 4 7])
ans =      6      2      1

>> v(2:2:6)
ans =      8      2      5

>> v(3:end-2)
ans =      4      2      4      5

>> v(v)


>> v([1, 1, 1, 2, end])
```

primo, quarto settimo elemento

2:2:6 è il vettore [2, 4, 6]: indica secondo, quarto, sesto elemento

dal terzo al terz'ultimo elemento

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =     6    8    4    2    4    5    1    3
>> v([1 4 7])
ans =      6      2      1

>> v(2:2:6)
ans =      8      2      5

>> v(3:end-2)
ans =      4      2      4      5

>> v(v)
ans =     5    3    2    8    2    4    6    4

>> v([1, 1, 1, 2, end])
```

primo, quarto settimo elemento

2:2:6 è il vettore [2, 4, 6]: indica secondo, quarto, sesto elemento

dal terzo al terz'ultimo elemento

i *valori* di v usati come indice

Esempi

```
> v=[6 8 4 2 4 5 1 3]
v =    6    8    4    2    4    5    1    3
>> v([1 4 7])
ans =     6     2     1
```

primo, quarto settimo elemento

```
>> v(2:2:6)
ans =     8     2     5
```

2:2:6 è il vettore [2, 4, 6]: indica secondo, quarto, sesto elemento

```
>> v(3:end-2)
ans =     4     2     4     5
```

dal terzo al terz'ultimo elemento

```
>> v(v)
ans =    5    3    2    8    2    4    6    4
```

i *valori* di v usati come indice

```
>> v([1, 1, 1, 2, end])
ans = 6    6    6    8    3
```

Indici ripetuti fanno replicare i valori nel sottovettore

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

```
v1(vettoreIndici) = v2
```

Viene però richiesto che **`v2`** abbia **le stesse dimensioni** di **`v1(vettoreIndici)`**

```
>> a = [1 : 10]
a =
     1  2  3  4  5  6  7  8  9  10

>> a(1 : 3) = [0 0 0]
a =
     0  0  0  4  5  6  7  8  9  10
```

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

```
v1(vettoreIndici) = v2
```

Viene però richiesto che **`v2`** abbia **le stesse dimensioni** di **`v1(vettoreIndici)`**

```
>> a =
     0  0  0  4  5  6  7  8  9  10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
```

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

```
v1(vettoreIndici) = v2
```

Viene però richiesto che **`v2`** abbia **le stesse dimensioni** di **`v1(vettoreIndici)`**

```
>> a =
     0  0  0  4  5  6  7  8  9  10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
a =
     0  0  0  8  5  12  7  16  9  20
```

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

`v1(vettoreIndici) = v2`

Viene però richiesto che **`v2`** abbia **le stesse dimensioni** di **`v1(vettoreIndici)`**

```
>> a =
     0  0  0  4  5  6  7  8  9 10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
a =
     0  0  0  8  5 12  7 16  9 20
>> a(1 : 2 : end) = a(end : -2 : 1)
```

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

`v1(vettoreIndici) = v2`

Viene però richiesto che **`v2`** abbia **le stesse dimensioni** di **`v1(vettoreIndici)`**

```
>> a =
      0  0  0  4  5  6  7  8  9  10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
a =
      0  0  0  8  5  12  7  16  9  20
>> a(1 : 2 : end) = a(end : -2 : 1)
```

Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

```
v1(vettoreIndici) = v2
```

Viene però richiesto che **v2** abbia **le stesse dimensioni** di **v1(vettoreIndici)**

```
>> a =
     0  0  0  4  5  6  7  8  9  10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
a =
     0  0  0  8  5  12  7  16  9  20
>> a(1 : 2 : end) = a(end : -2 : 1)
                    20 16 12 8 0
```

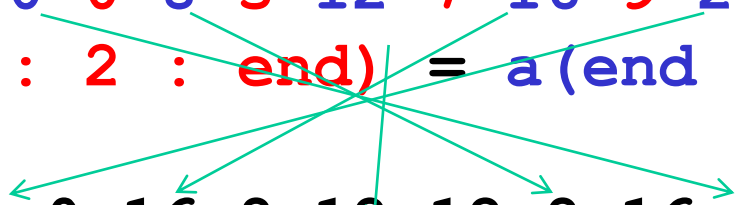
Modificare un Sotto-Array

È possibile **effettuare l'assegnamento tra sottovettori** per modificare una parte del vettore

```
v1(vettoreIndici) = v2
```

Viene però richiesto che **v2** abbia **le stesse dimensioni** di **v1(vettoreIndici)**

```
>> a =
     0  0  0  4  5  6  7  8  9 10
>> a(2 : 2 : end) = 2 * a(2 : 2 : end)
a =
     0  0  0  8  5 12  7 16  9 20
>> a(1 : 2 : end) = a(end : -2 : 1)
a =
    20  0 16  8 12 12  8 16  0 20
```



Sottoarray: Applicazione a Matrici

Si denota un sottoinsieme di un array usando vettori per valori degli indici

`nomeMatrice(vettore1,vettore2)`

restituisce una matrice che comprende gli elementi di **`nomeMatrice`** alle righe di indice in **`vettore1`** e alle colonne di indice in **`vettore2`**.

Sottoarray: Applicazione a Matrici

```
m =  9     8     7
     6     5     4
     3     2     1
     0    11    12
     0     0     0

>> m([1 4], [2 3])
```

Sottoarray: Applicazione a Matrici

```
m =  9     8     7
     6     5     4
     3     2     1
     0    11    12
     0     0     0
```

```
>> m([1 4], [2 3])
ans =   8     7
       11    12
```

tutti gli elementi sulle
righe 1 e 4 e sulle colonne 2 e 3

Sottoarray: Applicazione a Matrici

```
m =  9     8     7
     6     5     4
     3     2     1
     0    11    12
     0     0     0
```

```
>> m([1 4], [2 3])
ans =  8     7
      11    12
```

tutti gli elementi sulle
righe 1 e 4 e sulle colonne 2 e 3

```
>> m(1:2:5, 1:end)
```

Sottoarray: Applicazione a Matrici

```
m =  9     8     7
     6     5     4
     3     2     1
     0    11    12
     0     0     0
```

```
>> m([1 4], [2 3])
ans =  8     7
      11    12
```

tutti gli elementi sulle
righe 1 e 4 e sulle colonne 2 e 3

```
>> m(1:2:5, 1:end)
ans =  9     8     7
       3     2     1
       0     0     0
```

tutti gli elementi delle
righe 1, 3 e 5

Sottoarray: Applicazione a Matrici

```
m =  9     8     7
     6     5     4
     3     2     1
     0    11    12
     0     0     0

>> m(1:2:5, :)
ans =  9     8     7
       3     2     1
       0     0     0
```

notazione ‘:’ abbreviata per 1:end, cioè tutti i valori di quell’indice

Sottoarray: Applicazione a Matrici

```
    m = 9     8     7
        6     5     4
        3     2     1
        0    11    12
        0     0     0
>> m(1:2:5, :)
ans =  9     8     7
       3     2     1
       0     0     0
```

notazione ':' abbreviata per 1:end, cioè tutti i valori di quell'indice

```
>> m(2:2:4, :) = [-1 -2 -3; -4 -5 -6]
m =   9     8     7
     -1    -2    -3
      3     2     1
     -4    -5    -6
      0     0     0
```

uso della notazione dei sottoarray per individuare elementi oggetto di assegnamento

Assegnamenti con Scalari

È possibile associare a qualsiasi sotto array un valore scalare

`nomeVettore(vettoreIndici) = k`

Fa si che a tutti gli elementi di **`nomeVettore`** alle posizioni **`vettoreIndici`** venga assegnato il valore **`k`**

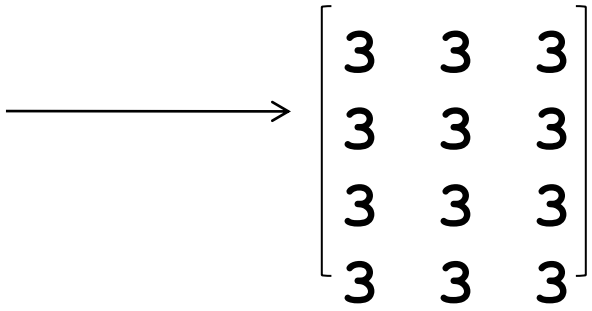
In questo modo è possibile inizializzare nuovi vettori.

```
>> a = [1 : 10]
a =
   1   2   3   4   5   6   7   8   9   10

>> a(1 : 3) = 0
a =
   0   0   0   4   5   6   7   8   9   10
```

Assegnamenti con Scalari

Esempio

```
m(1:4, 1:3) = 3
```

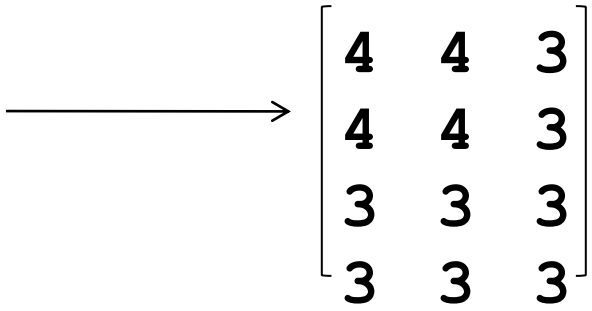


$$\begin{bmatrix} 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

Il modo con cui uno scalare viene assegnato a un array dipende dalla forma dell'array che viene specificata a sinistra dell'assegnamento

Esempio

```
m(1:2, 1:2) = 4
```



$$\begin{bmatrix} 4 & 4 & 3 \\ 4 & 4 & 3 \\ 3 & 3 & 3 \\ 3 & 3 & 3 \end{bmatrix}$$

Esempio

```
% inizializzare una matrice 5x5 con tutti valori a zero

% modificare la colonna centrale in 1

% modificare la riga centrale in 3

% sommare 2 ai valori della colonna centrale

% porre a 2 gli elementi nel primo quadrante

% copiare nell'ultima riga la prima riga letta al
contrario
```

Esempio

```
% inizializzare una matrice 5x5 con tutti valori a zero
A(5,5) = 0;
% modificare la colonna centrale in 1
A(: , 3) = 1;
% modificare la riga centrale in 3
A(3 , : ) = 3;
% sommare 2 ai valori della colonna centrale
A(: , 3) = A(: , 3) + 2; % NB termini a dx e sx
dell'uguale hanno la stessa dimensione
% porre a 2 gli elementi nel primo quadrante
A(1 : 2 , 1 : 2) = 2;
% copiare nell'ultima riga la prima riga letta al
contrario
A(end , :) = A(1 , end : -1 : 1)
```

Assegnamenti con Scalari su matrici

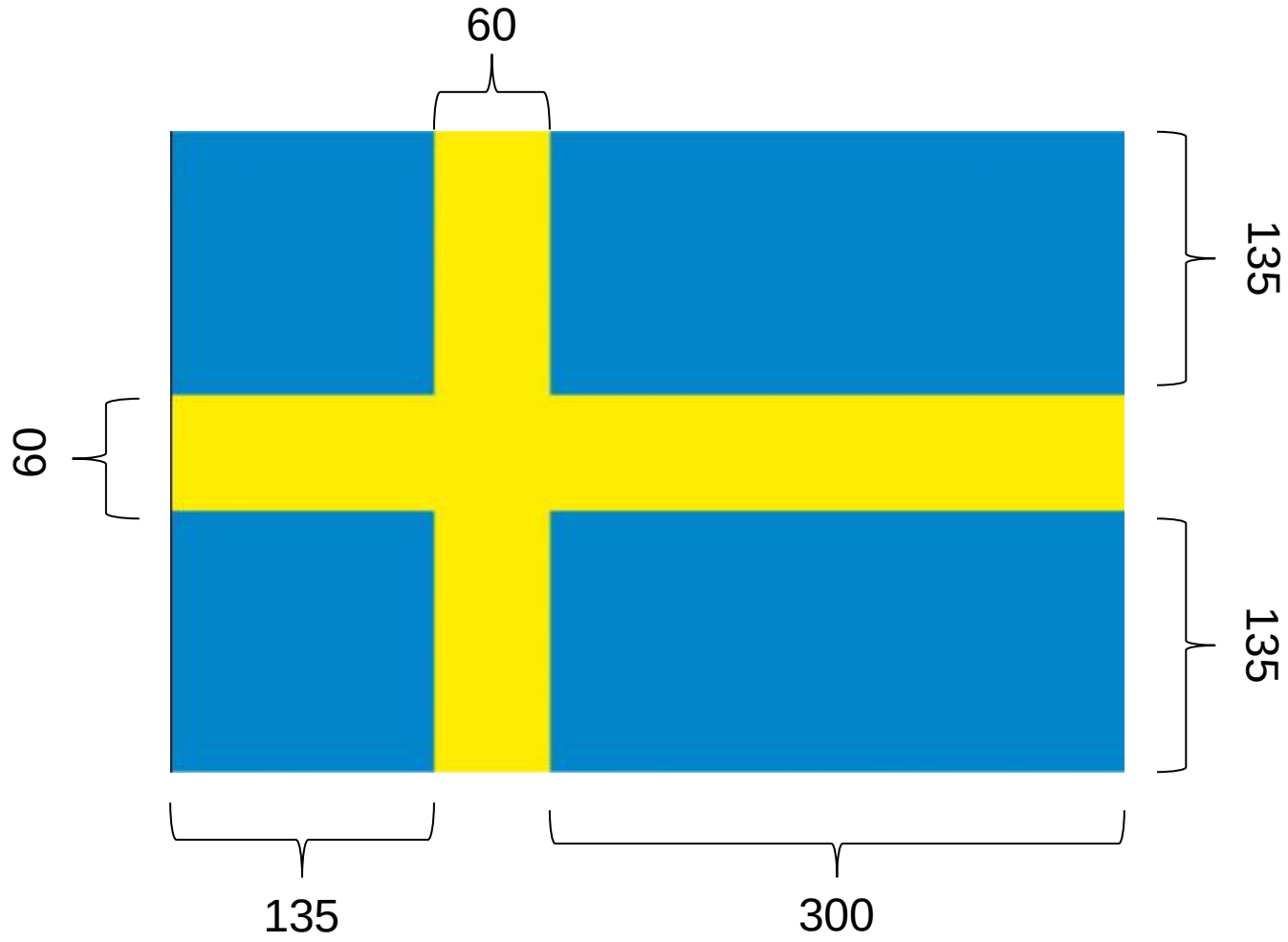
Il modo con cui uno scalare viene assegnato a un array dipende dalla forma dell'array che viene specificata a sinistra dell'assegnamento

Es.

```
>> clear m;

>> m(4, 3) = 3;

>> m(1:2, 1:2) = 4

ans =

     4     4     0

     4     4     0

     0     0     0

     0     0     3
```

Esercizio

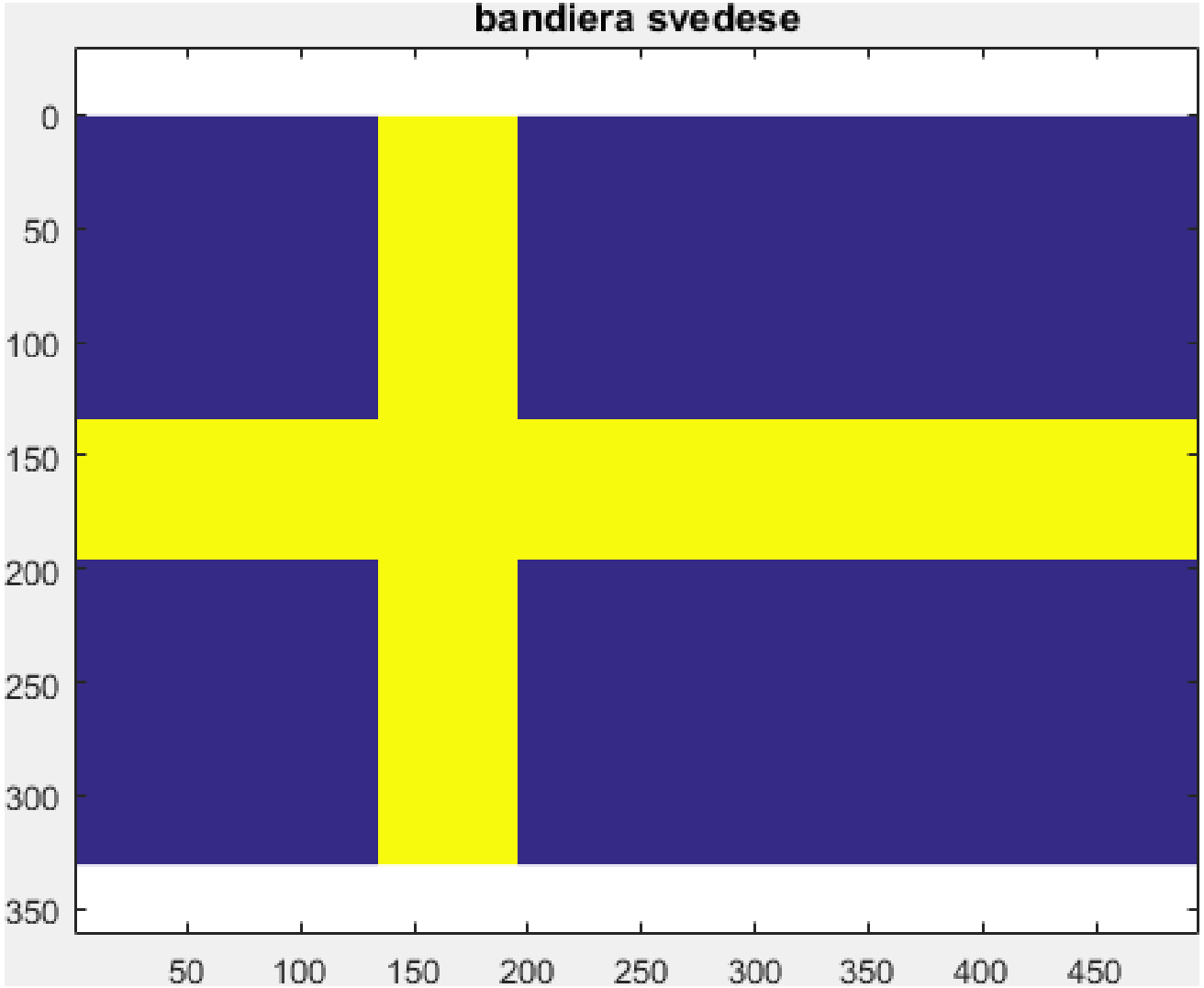
Disegnare la bandiera svedese



Bandiera Svedese

```
clear;
clc;
close all;

A(330, 495) = 0;
A(:, 135:195) = 1;
A(135:195, :) = 1;

figure();
imagesc(A);
title('Bandiera svedese');
axis equal;
```

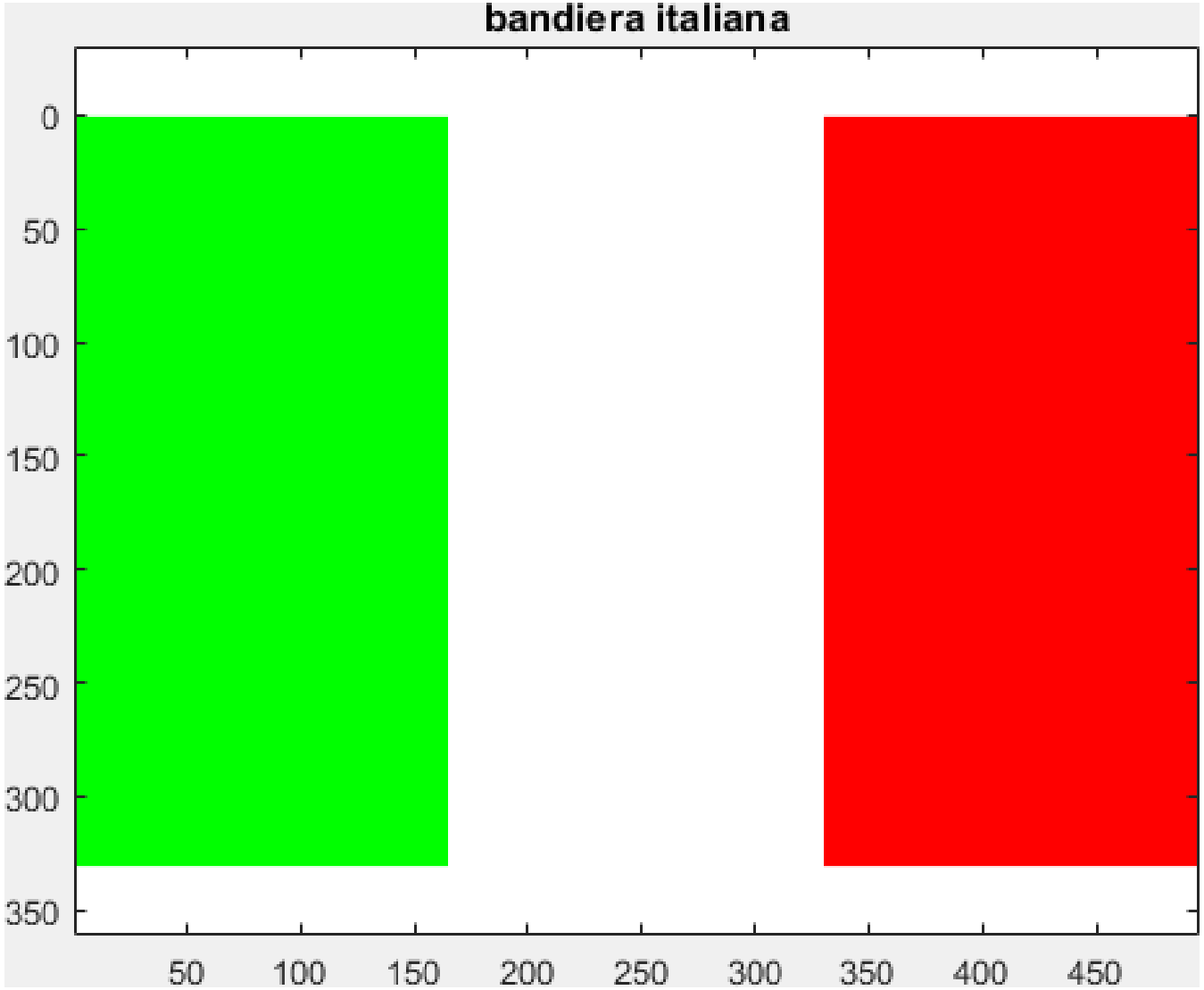
Ecco il Risultato (immagine 2D)



Bandiera Italiana (immagine 3D)

```
clear;
clc;
close all;

A(330, 495, 3) = 0;
A(:, 1 : 495/3, 2) = 1; % verde
A(:, 495/3 : (2*495)/3, :) = 1; % bianco
A(:, (2*495)/3 : end, 1) = 1; % rosso
A = uint8(255 * A);
figure(); imagesc(A);
title('Bandiera italiana');
axis equal;
```

Ecco il Risultato



GRAN CONTEST DELLE BANDIERINE

Come creare le immagini a colori:

- E’ possibile riprodurre i colori in due modi:
  - creando una matrice 3D B che ha N_righe x N_colonne x 3 dove il terzo piano indica il colore (B(:,:,1) è il rosso, B(:,:2) il verde, B(:,:,3) il blu) e visualizzando con imshow
  - creando una matrice 2D e modificando la colormap
  - **Cercate su internet i colori corretti in RGB e ricordate di convertire in uint8 prima di visualizzare la matrice**

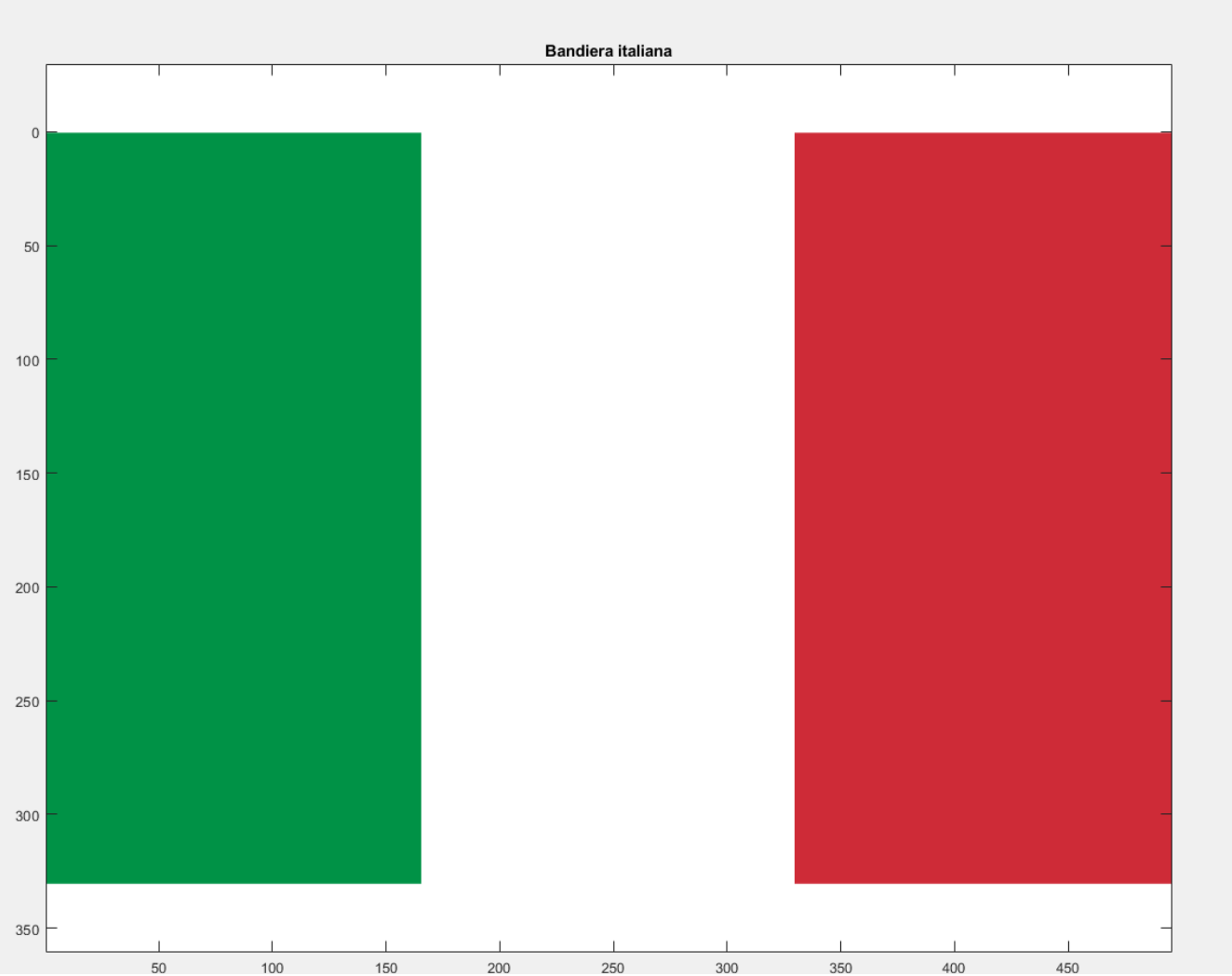
Colors in Palette

Color	Hex	RGB
	#f2f2f2	(242,242,242)
	#009246	(0,146,70)
	#ffffff	(255,255,255)
	#ce2b37	(206,43,55)
	#f2f2f2	(242,242,242)

Uso i colori corretti

```
A = ones(330, 495, 3) * 255 ;

A(:, 1 : 495/3, 1) = 0; % R: banda
verde

A(:, 1 : 495/3, 2) = 146; % G: banda
verde

A(:, 1 : 495/3, 3) = 70; % B: banda
verde


A(:, (2*495)/3 : end, 1) = 206; % rosso

A(:, (2*495)/3 : end, 2) = 43; % rosso

A(:, (2*495)/3 : end, 3) = 55; % rosso


A = uint8(A);

figure(); imagesc(A);
```

Bandiera italiana con i colori corretti



GRAN CONTEST DELLE BANDIERINE

Regole:

- La bandiera deve essere realizzata interamente con uno script MatLab, senza interazione con l'utente
- È necessario usare operazioni vettoriali, si possono usare cicli
- Verrà valutata sia la veridicità della bandiera, sia la struttura del codice utilizzato per realizzarla
- Potete presentare una sola bandiera a persona
- Inviate via mail a gfrancesco1.trovo@polimi.it
  - un'immagine (png) della bandiera
  - lo script che la genera (opportunamente indentato e commentato)
- **Deadline: Domenica 21 Novembre 2021 (23.59 UTC+1)**

GRAN CONTEST DELLE BANDIERINE

Premi:

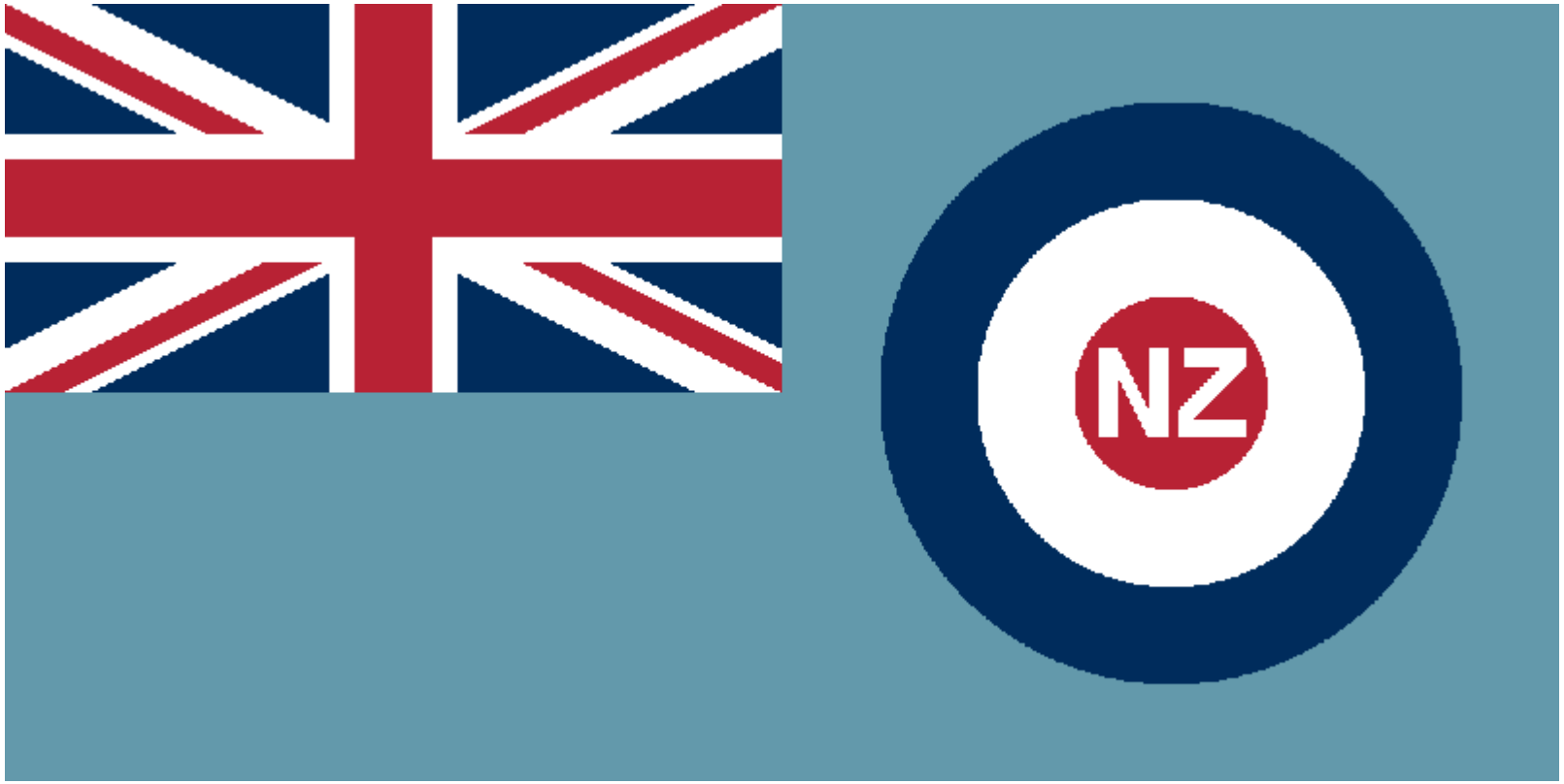
Tre punti alla bandierina/codice migliore e premi a scalare (a patto che lo studente dimostri di aver capito il codice scritto)

Deadline: Domenica 21 Novembre 2020 (23.59 UTC+1)

Esempi di bandiere realizzate dai vostri colleghi…

Bandiera della Catalogna

Esempi di bandiere realizzate dai vostri colleghi…



New Zealand Airforce