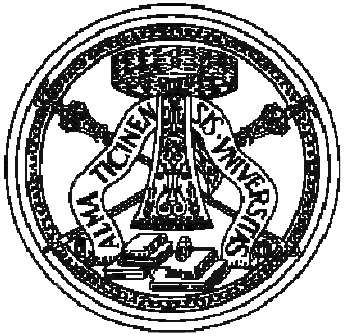
UNIVERSITÀ DEGLI STUDI DI PAVIA
FACOLTÀ DI INGEGNERIA



Matlab: esempi ed esercizi

Sommario e obiettivi

- Sommario
  - Esempi di implementazioni Matlab di semplici algoritmi
  - Analisi di codici Matlab

- Obiettivi
  - Tradurre un algoritmo in un programma Matlab
  - Comprendere algoritmi già implementati in Matlab

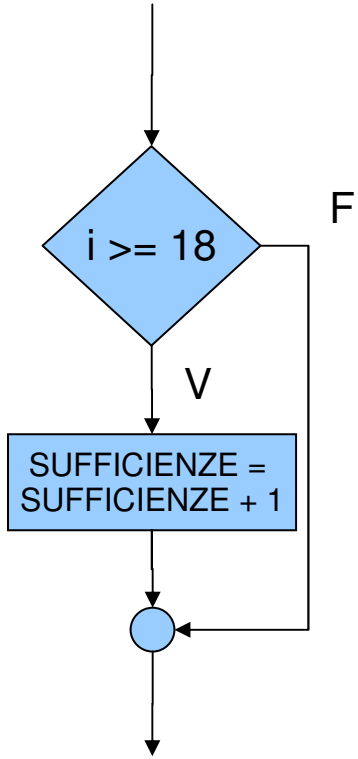
Esempio 1 (1)

- Dato un vettore contenente una serie di voti, contare quante sono le sufficienze (cioè i voti maggiori o uguali a 18)
  - Per ogni voto del vettore devo verificare se è verificata la condizione di sufficienza
    - Nel caso un cui il voto sia sufficiente, dovrò incrementare un contatore
  - Il controllo deve essere ripetuto per tutti i voti del vettore

Esempio 1 (2)

- Per ogni i-esimo voto del vettore, devo verificare se questo è sufficiente o meno
  - Il digramma di flusso a lato può essere implementato in un programma Matlab utilizzando il costrutto *if*

```
if (i >= 18)
     sufficienze = sufficienze + 1;
end;
```



Esempio 1 (3)

- Il controllo deve quindi essere ripetuto per tutti gli elementi del vettore voti
  - Per ripetere un blocco di operazioni un certo numero di volte si può impiegare il costrutto di programmazione *for*

```
for (i = voti)
      % controllo sull'i-esimo voto
end;
```

Esempio 1 (4)

- Il programma completo diventa quindi:

```
voti = [13 19 17 24 30 11 16];
sufficienze = 0;

for i = voti
   if (i >= 18)
      sufficienze = sufficienze + 1;
   end;
end;
```

Esercizio 1 (1)

- **Dato il seguente codice Matlab, si presenti il significato di ogni statement e si dica che cosa stampa:**

```
voti = [11 19 17 24 30 21 16 9];
insufficienze = 0;
for i = voti
   if (i < 18)
      insufficienze = insufficienze + 1;
   end;
end;
insufficienze
```

Esercizio 1 (2)

```
% viene creato un vettore 'voti' composto
dagli elementi sotto riportati
voti = [11 19 17 24 30 21 16 9];

% la variabile 'insufficienze' viene impostata
a 0
insufficienze = 0;

% ciclo for: la variabile 'i' assume in
sequenza tutti i valori del vettore 'voti'. Il
ciclo verrà ripetuto tante volte quanti sono
gli elementi del vettore (cioè 8). Al primo
passo 'i' varrà quindi 11, al secondo 19 e
così via
for i = voti
```

Esercizio 1 (3)

```
    % se l'elemento i-esimo del vettore 'voti'
    soddisfa la condizione (nell'esercizio
    i<18), allora si incrementa di 1 il
    contatore delle insufficienze
    if (i < 18)
          insufficienze = insufficienze + 1;
    end;
end;

% stampa il valore finale del contatore delle
insufficienze
insufficienze
```

Esercizio 1 (4)

- Il programma proposto conta e visualizza quanti valori del vettore ‘voti’ sono insufficienti (ovvero inferiori a 18). L’esercizio proposto al termine dell’esecuzione stamperà quindi:

```
insufficienze=4
```

Esempio 2 (1)

- Data una base ed un esponente, calcolare la potenza utilizzando una serie di prodotti.
  - Ad esempio, $3^5 = 3 * 3 * 3 * 3 * 3 = 243$
  - Poiché devo ripetere un'operazione (il prodotto) un numero di volte noto a priori, il problema può essere risolto impiegando la struttura di programmazione *for*
    - *All'interno del ciclo for devo effettuare il prodotto*
    - *Il ciclo deve essere ripetuto tante volte quanto vale l'esponente*

Esempio 2 (2)

```
base = 3;
esponente = 5;
potenza = 1;

for (i = 1:esponente)
    potenza = potenza * base;
end
```

		base	esponente	potenza	i
base = 3		3			
esponente = 5			5		
potenza = 1				1	
potenza = potenza * base	ciclo for			3 * 1 = 3	1
potenza = potenza * base				3 * 3 = 9	2
potenza = potenza * base				9 * 3 = 27	3
potenza = potenza * base				27 * 3 = 81	4
potenza = potenza * base				81 * 3 = 243	5

Esercizio 2 (1)

- **Dato il seguente codice Matlab, si presenti il significato di ogni statement e si dica che cosa stampa:**

```
base = 2;
esponente = 10;
potenza = 1;

for (i = 1:esponente)
    potenza = potenza * base;
end
potenza
```

Esercizio 2 (2)

```
% viene creata una variabile 'base'
pari a 2
base = 2;

% viene creata una variabile
'esponente' pari a 10
esponente = 10;

% viene creata una variabile 'potenza'
pari a 1
potenza = 1;
```

Esercizio 2 (3)

```
% ciclo for: la variabile 'i' assume in
sequenza tutti i valori interi da 1 al valore
di 'esponente', ovvero 10. Al primo passo 'i'
varrà quindi 1, al secondo 2 e così via
for (i = 1:esponente)

   % la variabile 'potenza' assume come nuovo
   valore il prodotto del suo valore corrente
   per la base
   potenza = potenza * base;
end

% stampa il valore finale della variabile
'potenza'
potenza
```

Esercizio 2 (4)

- Il programma proposto calcola e stampa il valore della variabile ‘base’ elevata all’esponente dato. L’esercizio proposto al termine dell’esecuzione stamperà quindi:

```
potenza=1024
```

Esempio 3 (1)

- Creare una matrice quadrata di dimensione data e avente ogni elemento pari al prodotto del numero di riga e colonna corrispondenti alla posizione dell'elemento stesso
  - scorro tutti le i-esime righe della matrice
  - per ogni riga, scorro tutti i j-esimi elementi
    - l'elemento in posizione (i,j) vale i*j, ovvero ogni elemento della matrice è pari al prodotto del numero di riga e colonna corrispondenti alla posizione dell'elemento stesso

Esempio 3 (2)

- Es: Creare una matrice 3x3
  - Scorro la prima riga (*i = 1*)
    - Imposto il valore nella prima colonna *(j=1)* a: (*i * j) = 1*1 = 1*

$$\begin{bmatrix} 1 & .. & .. \\ .. & .. & .. \\ .. & .. & .. \end{bmatrix}$$

    - Imposto il valore nella seconda colonna *(j=2)* a: (*i * j) = 1*2 = 2*

$$\begin{bmatrix} 1 & 2 & .. \\ .. & .. & .. \\ .. & .. & .. \end{bmatrix}$$

    - Imposto il valore nella terza colonna *(j=3)* a: (*i * j) = 1*3 = 3*

$$\begin{bmatrix} 1 & 2 & 3 \\ .. & .. & .. \\ .. & .. & .. \end{bmatrix}$$

Esempio 3 (3)

- Scorro quindi la seconda riga (*i = 2*)
  - Imposto il valore nella prima colonna *(j=1)* a: *(i * j) = 2*1 = 2*

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & .. & .. \\ .. & .. & .. \end{bmatrix}$$

  - Imposto il valore nella seconda colonna *(j=2)* a: *(i * j) = 2*2 = 4*

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & .. \\ .. & .. & .. \end{bmatrix}$$

  - Imposto il valore nella terza colonna *(j=3)* a: *(i * j) = 2*3 = 6*

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ .. & .. & .. \end{bmatrix}$$

Esempio 3 (4)

- Per impostare tutti i j-esimi elementi di una riga dal primo fino all'ultimo, pari alla dimensione della matrice, è possibile utilizzare un ciclo for

```
for j = 1:dimensione
    A(i,j) = i*j;
end;
```

- Allo stesso modo, l'operazione effettuata lungo una riga deve essere ripetuta per tutte le i-esime righe della matrice

```
for i = 1:dimensione
    % imposta tutti gli elementi della riga
end;
```

Esempio 3 (5)

- Inserendo il primo ciclo all'interno del secondo, si ottiene quindi il programma definitivo per la creazione della matrice A:

```
for i = 1:dimensione
   for j = 1:dimensione
      A(i,j) = i*j;
   end;
end;
```

Esercizio 3 (1)

- **Dato il seguente codice Matlab, si presenti il significato di ogni statement e si dica che cosa stampa:**

```
dimensione = 3;
for i = 1:dimensione
   for j = 1:dimensione
      A(i,j) = 2*i*j;
   end;
end;
A
```

Esercizio 3 (2)

```
% viene creata una variabile
'dimensione' pari a 3
dimensione = 3;


% ciclo for: la variabile 'i' assume
in sequenza tutti i valori interi da 1
al valore di 'dimensione', ovvero 3.
Al primo passo 'i' varrà quindi 1, al
secondo 2 e infine varrà 3
for (i = 1:dimensione)
```

Esercizio 3 (3)

```
% ciclo for: la variabile 'j' assume in
sequenza tutti i valori interi da 1 al valore
di 'dimensione', ovvero 3. Al primo passo 'j'
varrà quindi 1, al secondo 2 e infine varrà 3
for (j = 1:dimensione)

   % L'elemento in posizione (i,j) della
   matrice A viene impostato al doppio del
   prodotto di 'i' per 'j'
   A(i,j)= 2*i*j;
end

% stampa il valore finale della matrice A
A
```

Esercizio 3 (4)

- Il programma proposto crea una matrice quadrata, ovvero avente lo stesso numero di righe e di colonne, A. Ogni elemento di A è pari al doppio del prodotto del numero di riga e colonna corrispondenti alla posizione dell'elemento stesso. L'esercizio proposto al termine dell'esecuzione stamperà quindi:

```
A = [2  4  6
     4  8 12
     6 12 18]
```

Esempio 4 (1)

- Data una matrice quadrata A di dimensione nota, creare una seconda matrice B
  - avente la stessa dimensione di A;
  - contenente una copia degli elementi di A se questi sono maggiori del valore dato *x*
  - zero se questi sono minori o uguali ad *x*.

Esempio 4 (2)

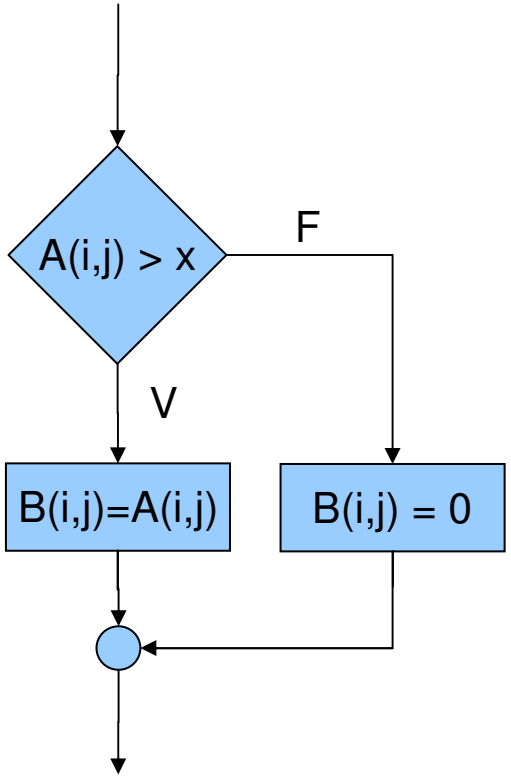
- Come visto nell'esempio precedente, per scorrere tutte le righe e, per ogni riga, tutti gli elementi lungo le colonne, devono essere impiegati due cicli *for* annidati

```
for i=1:dimensione
    for j=1:dimensione
        % imposta gli elementi di B
    end
end
```

Esempio 4 (3)

- Il valore dei singoli elementi di B deve invece essere determinato sulla base di un confronto.
  - La scelta viene implementata in Matlab utilizzando il costrutto *if-else*

```
if A(i,j) > x
        B(i,j) = A(i,j);
else
      B(i,j) = 0;
end
```



Esempio 4 (4)

- Il programma completo diventa quindi:

```
for i=1:dimensione
  for j=1:dimensione
     if A(i,j) > x
           B(i,j) = A(i,j);
  else
           B(i,j) = 0;
  end
end
```

Esercizio 4 (1)

- **Dato il seguente codice Matlab, si presenti il significato di ogni statement e si dica che cosa varrà 'negative' al termine dell'esecuzione:**

```
misure = [-3 4 0 -1 -2 11 1];
lunghezza = length(misure);

for i = 1:lunghezza
    if misure(i) < 0
        negative(i) = misure(i);
    else
        negative(i) = 0;
    end
end
```

Esercizio 4 (2)

```
% viene creato un vettore 'misure' composto
dagli elementi sotto riportati
misure = [-3 4 0 -1 -2 11 1];

% alla variabile 'lunghezza' viene assegnato
il valore restituito dalla funzione
'lenght()'. La funzione, a cui viene passato
come parametro la variabile 'misure',
restituisce quindi il numero di elementi
presenti nel vettore 'misure' (cioè 7)
lunghezza = length(misure)

% ciclo for: la variabile 'i' assume in
sequenza tutti i valori interi da 1 fino al
valore della variabile 'lunghezza' (in questo
esercizio 7)
for i = 1:lunghezza
```

Esercizio 4 (3)

```
    % se l'elemento i-esimo del vettore 'misure'
    soddisfa la condizione (nell'esercizio se è
    minore di zero), allora l'elemento del vettore
    'negative' nella i-esima posizione assume il
    medesimo valore
    if misure(i) < 0
        negative(i) = misure(i);

    % altrimenti l'elemento i-esimo del vettore
    'negative' viene posto uguale a zero
    else
        negative(i) = 0;
    end
end
```

Esercizio 4 (4)

- Il programma proposto crea un vettore ‘negative’ (avente lo stesso numero di elementi del vettore ‘misure’) contenente una copia degli elementi di ‘misure’ quando queste sono negative (cioè minori di 0), uno zero altrimenti. Al termine dell’esecuzione il vettore ‘negative’ varrà quindi:

```
negative = [-3 0 0 -1 -2 0 0]
```

Esempio 5 (1)

- Dato il numero intero *n*, calcolare il corrispondente numero di Fibonacci
  - Per definizione il numero di Fibonacci di 1 è esattamente pari a 1
  - Allo stesso modo, il valore di 2 è sempre 1.
  - Per i valori successivi, il numero di Fibonacci è definito **ricorsivamente**: dato un numero *n*, il numero di Fibonacci è pari alla somma dei numeri di Fibonacci di *n-1* e *n-2*.
    - fibonacci(*n*) = fibonacci(*n-1*) + fibonacci(*n-2*)

Esempio 5 (2)

- Per esempio, detta fibonacci(n) la funzione di Fibonacci, per *n=3* vale:
  - fibonacci(3) = fibonacci(3-1) + fibonacci(3-2)
= fibonacci(2) + fibonacci(1)
= 1 + 1 = 2
- Analogamente, per *n=4, si ha:*
  - fibonacci(4) = fibonacci(*4-1*) + fibonacci(*4-2*)
= fibonacci(*3*) + fibonacci(*2*)
= [fibonacci(*3-1*) + fibonacci(*3-2*)] + fibonacci(*2*)
= [fibonacci(*2*) + fibonacci (*1*)] + fibonacci(*2*)
= [1 + 1] + 1 = 2 + 1 = 3
- E così via al crescere di n

Esempio 5 (3)

- Dalla definizione, è immediato scrivere la funzione Matlab che calcola il numero di Fibonacci.
  - E' importante notare come questo sia un classico esempio di funzione ricorsiva

```
function [ Fib ] = fibonacci( n )
   if n == 1
          Fib = 1;
   elseif n == 2
          Fib = 1;
   else
          Fib = fibonacci(n-1) + fibonacci(n-2);
   end
return
```

Esercizio 5 (1)

- **Dato il seguente codice Matlab, si presenti il significato di ogni statement e si dica che cosa stampa:**

```
function p = potenza(base, esponente)
    if (esponente == 1)
        p = base;
    else
        p = base * potenza(base, esponente -1);
    end
return
```

Esercizio 5 (2)

```
% viene dichiarata una funzione 'potenza' che
accetta in ingresso due parametri ('base' ed
'esponente') e restituisce una variabile ('p')
function p = potenza(base, esponente)

% se il valore della variabile 'esponente' passato
alla funzione è pari a 1, 'p' viene posto uguale a
'base'
if (esponente == 1)
    p = base;
% altrimenti 'p' diventa pari al prodotto di 'base'
moltiplicata per il valore restituito dalla
funzione 'potenza', richiamata passando come
parametri 'base' ed il valore di 'esponente' meno 1
else
    p = base * potenza(base, esponente -1 );
end
```

Esercizio 5 (3)

```
% torna al programma che chiama la funzione e
restituisce in 'p' il valore calcolato
return
```

- La funzione implementata nel programma proposto, dati una base ed un esponente, calcola e stampa la potenza.
- Per effettuare il calcolo viene utilizzata la ricorsione.
- Se ad esempio *base=2* e *esponente=3*, *potenza(2,3)* calcolerebbe il risultato come *2*potenza(2,2)* e, a sua volta, *potenza(2,2)* sarebbe calcolato come *2*potenza(2,1).* Si avrebbe quindi:
  - potenza(2,3) = 2*potenza(*2,2*) =
= 2*[2*potenza(2,1)] =
= 2*[2*2] = 8

Esempio 6 (1)

- Dato un numero *n*, calcolare la radice quadrata con il metodo della bisezione.
  - La radice viene ricercata per approssimazioni successive accettando nel calcolo effettuato un errore minore di un valore prefissato. Si procede quindi considerando il numero dato, calcolandone il quadrato e l'errore commesso ($n^2 - n$):
    - se l'errore è nullo o minore del valore prefissato, allora la radice è *n*;
    - altrimenti si divide l'intervallo [0; n] in due parti eguali e si calcola il valore della potenza nel punto medio.
      - Se il risultato è esatto, allora *n/2* è la radice cercata;
      - altrimenti si sceglie tra i due intervalli in base all'errore di approssimazione commesso:
        - se il quadrato del numero trovato è maggiore del valore dato, allora la radice calcolata è approssimata per eccesso e si considera solo l'intervallo di sinistra;
        - viceversa, se il quadrato della radice è inferiore a n, la radice è approssimata per difetto e viene quindi considerato l'intervallo a destra.

Esempio 6 (2)

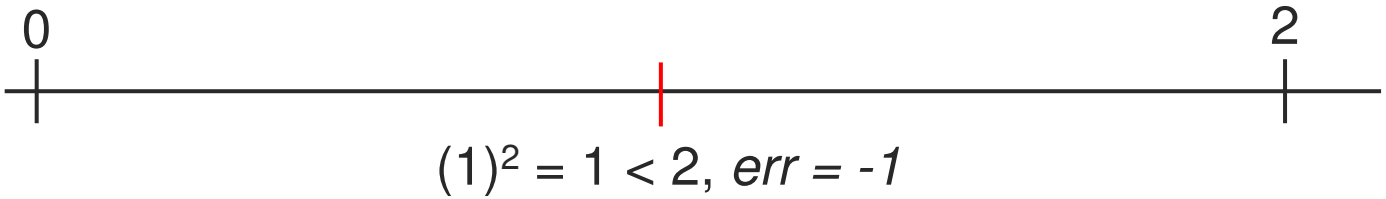
- Si ripete per il nuovo sotto-intervallo il procedimento di dimezzamento.
  - Analogamente a quanto accade per l'algoritmo di ricerca con bisezione, la dimensione dell'intervallo di ricerca si dimezza ad ogni iterazione e, con approssimazioni successive, ci si avvicina sempre più alla soluzione cercata.
- L'algoritmo viene arrestato quando il quadrato della radice trovata è abbastanza vicino al valore *n*, cioè quando l'errore di approssimazione commesso è inferiore ad una certa soglia di tolleranza predefinita.

Esempio 6 (3)

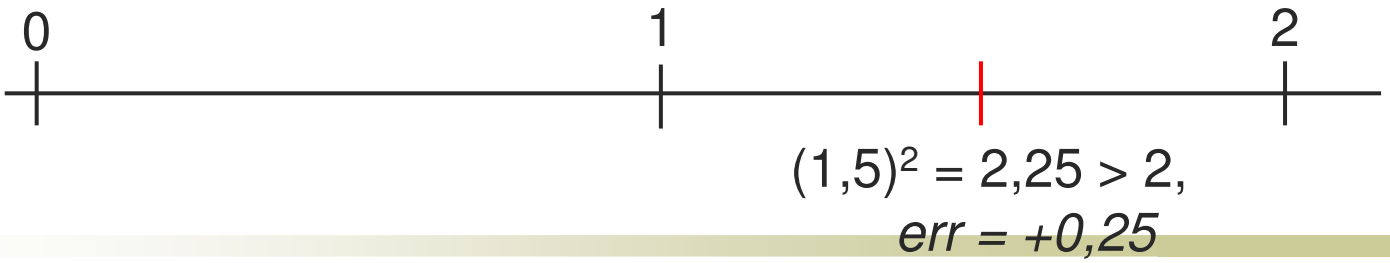
- Es. Calcolare la radice quadrata di 2 (errore di approssimazione non superiore a 0,1)
  - In prima approssimazione considero esattamente il numero dato, cioè 2, e calcolo il quadrato e l'errore commesso

Esempio 6 (4)

- Poiché 2^2 approssima per eccesso 2, si ripete il procedimento considerando il centro dell'intervallo [0; 2], cioè 1

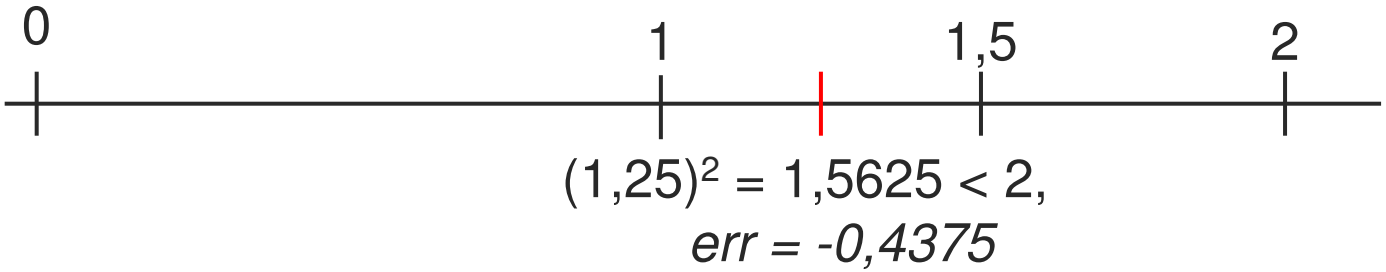


- Il valore 1 approssima per difetto la radice di 2. Devo quindi utilizzare l'intervallo a destra. Ripeto quindi il procedimento considerando il centro dell'intervallo [1; 2]

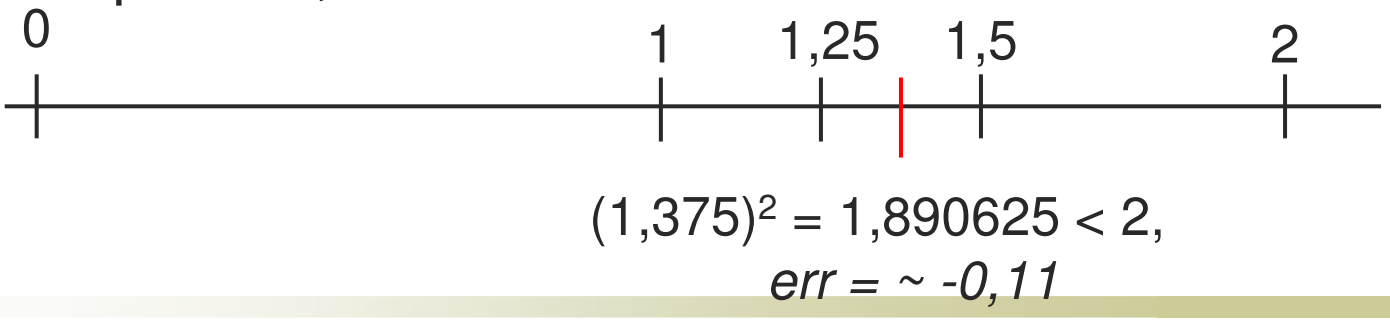


Esempio 6 (5)

- Il valore 1,5 approssima invece per eccesso la radice. Ora considero quindi il centro dell'intervallo di sinistra [1; 1,5]

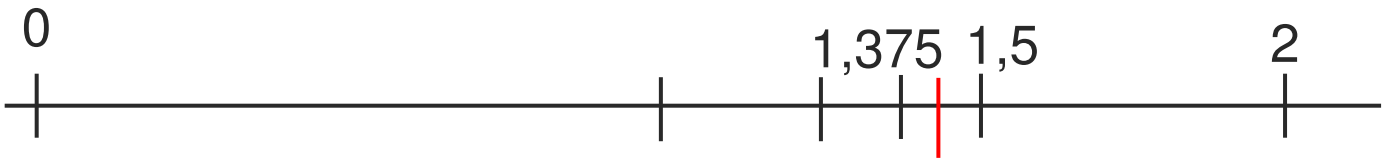


- La radice di 2 è quindi compresa tra 1,25 e 1,5. Il nuovo valore approssimato che considero è quindi 1,375



Esempio 6 (6)

- Il valore 1,375 approssima ancora per difetto la radice. Considerando quindi 1,4375, ovvero il centro dell'intervallo di destra [1,375; 1,5], si ha:



$(1{,}4375)^2 = \sim 2{,}0664 > 2,$

$err = +0{,}0664$

- A questo punto l'errore di approssimazione commesso è inferiore a 0,1 (il valore trovato è effettivamente simile al valore esatto 1,41421..) e posso fermarmi. Valori più precisi potrebbero essere comunque ottenuti iterando il procedimento in maniera analoga con soglia inferiore a quella scelta.

Esempio 6 (7)

```
n = 2;
soglia = 0.1;

radice = n;
intervallo = n/2;
errore = radice * radice - n;

   while (abs(errore) > soglia)
       if errore > 0
          radice = radice - intervallo;
       else
          radice = radice + intervallo;
       end
   intervallo = intervallo / 2;
   errore = radice * radice - n;
end
```