COMPUTER SKILLS

LESSON 5

Valeria Cardellini
cardellini@ing.uniroma2.it
A.Y. 2015/16

Objectives of this lesson

We'll discuss

- How to create matrices
- How to refer to and modify elements of matrices
- Vectors and matrices as function arguments

Matrices

A matrix is a table of values. The dimensions of a matrix are *m* x *n*, where *m* is the number of rows and *n* is the number of columns.

$$A_{(m \times n)} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & & \ddots & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

Transposed matrix

$$A'_{(n \times m)} = \begin{bmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{m2} \\ \vdots & & \ddots & \\ a_{1n} & a_{2n} & \dots & a_{mn} \end{bmatrix}$$

Matrix manipulation

We consider the following basic operations on matrices:

- Create a matrix
- Read and extract data from a matrix by indexing
- Shorten a matrix
- Mathematical and logical operations on matrices
- In the past lesson: determine the matrix size

- Note: when the operators apply to both vectors and matrices with the same syntax, we use the term array
  - Vectors are 1-d arrays
  - Matrices are 2-d arrays

Creating a matrix: constant values

- Entering the values directly: the semicolon identifies the next row
- Be careful: always the same number of elements in each row

- Using the functions **`zeros(rows, cols)`**, **`ones(rows, cols)`**, **`rand(rows, cols)`** and **`randn(rows, cols)`** to create matrices filled with 0, 1, or random values between 0 and 1 (from uniform or normal distribution)

```
>> A=[2 4 5 6; 3 5 6 7]

A =

     2     4     5     6

     3     5     6     7

>> rand(3,2)

ans =

   8.1472e-01   9.1338e-01

   9.0579e-01   6.3236e-01

   1.2699e-01   9.7540e-02
```

Indexing a matrix

- The process of referring to and modifying elements in a matrix
- Syntax:
  - `A(row, col)` returns the element(s) at the location(s) specified by the matrix row and column indices (called *subscripted indexing*)
  - `A(row, col) = value` replaces the elements at the location(s) specified by the matrix row and column indices
- The index array may contain either numerical or logical values
- In MATLAB `end` is a built-in expression to refer to the last element of the row or column, e.g., the last element of 1st row

```
>> A(1,end)
ans = 6
```

```
>> A(2,3)
ans =
     6
>> A(2,:)   % 2nd row
ans =
     3     5     6     7
>> A(2,2:4)
ans =
     5     6     7
>> B=[3 4; 4 7]
B =
     3     4
     4     7
>> C=[true false; true
true]
C =
     1     0
     1     1
>> B(C) % apply C as mask
ans =
     3
     4
     7
```

See A values on previous slide

Numerical indexing

- The index vector may be of any length
- When it contains numerical values, use only integer (non-fractional) numbers
- The values in the index vector are constrained by the following rules:
  - To refer to elements, all index values must be
    1 <= element <= length (row or column dimension)
  - To replace elements, all index values must be
    1 <= element
- In MATLAB we can also use *linear indexing*: the matrix is unwound column by column

```
>> A(3)  % see A values on slide 6
ans =
     4
>> A(3)  % see A values on slide 6
ans =
     3
```

Replacement rules for matrix elements

1. Either:
   - There must be a single element on the right side of the instruction, or
   - All dimensions of the blocks on either side of the assignment must be equal
2. If you replace beyond the end of any dimension of the matrix, its size is automatically increased
   - Any element not specifically replaced remains unchanged
   - Elements beyond the existing dimension length not replaced are set to 0

```
>> A(1,3)=8
A =
    2    4    8    6
    3    5    6    7
>> A(:,1)=[9 7] %replace
1st column
A =
    9    4    8    6
    7    5    6    7
>> A(:,6) = [3 8]' %add new
column
A =
    9    4    8    6    0    3
    7    5    6    7    0    8
```

See A values on slide 6

Logical indexing

- Logical arrays use relational expression that result in `true/false` values
- The index array length must be less than or equal to the original array dimension
- It must contain logical values (`true` or `false`)
- Access to the matrix elements is by their relative position in the logical array
  - When reading elements, only the elements corresponding to true index values are returned
  - When replacing elements, the elements corresponding to true index values are replaced

Logical indexing

```
>> vec = [5 9 3 4 6 11]
vec =
     5     9     3     4     6    11
>> isg = vec > 5          %if vec[i]>5 then isg[i]=true
isg =
    0    1    0    0    1    1
>> vecisg = vec(isg)      %only elements > 5
vecisg =
     9     6    11
```

- Note that in MATLAB logical vectors echo in the Command window as 1 or 0, but they are not the same thing! See next slide

Logical data type is different from double data type!

```
>> mask = [0 1 0 1]
mask =
     0     1     0     1

>> whos mask
  Attr Name        Size              Bytes  Class
   ==== ====         ====              =====  =====
        mask        1x4                32  double
>> A
A =
     1     4     7    10
>> A(mask)
error: subscript indices must be either positive integers
less than 2^31 or logicals.
```

Operating on vectors and matrices

Three techniques extend directly from operations on scalar values:

- Arithmetic operations
- Logical operations
- Applying library functions

Two techniques are unique to arrays in general, and to vectors in particular:

- Concatenation
- Slicing (generalized indexing)

Operators precedence

- Operator description and precedence

www.mathworks.it/help/matlab/matlab_prog/operator-precedence.html

Table 5.2: Operator precedence

PRECEDENCE	OPERATOR
1	Parentheses ()
2	Transpose (.'), power (.^), matrix power (^)
3	Unary plus (+), unary minus (−), logical negation (~)
4	Multiplication (.*), right division (./), left division (.\), matrix multiplication (*), matrix right division (/), matrix left division (\)
5	Addition (+), subtraction (−)
6	Colon operator (:)
7	Less than (<), less than or equal to (≤), greater (>), greater than or equal to (≥), equal to (==), not equal to (~=)
8	Element-wise AND, (&)
9	Element-wise OR, (|)

Arithmetic operations

- Arithmetic operations (e.g., addition and multiplication) can be done on entire vectors or matrices
- Some examples of scalar operations (add a scalar, multiply or divide by a scalar)

```
>> A = [2 5 7 1 3];
>> A + 5          % add 5 to every element of A
ans =
7 10 12 6 8
>> A * 2          % multiply by 2 every element of A
ans =
4 10 14 2 6
```

Arithmetic operations (continue)

- Some examples of array operations, that are performed on vectors or matrices *element by element*
  - Be careful: a dot must be placed in front of the operator for array operations (`.*`, `.^`, `./`, `.\`)

```
>> B = -1:1:3
B =
-1  0  1  2  3
>> A .* B % element-by-element multiplication
ans =
-2  0  7  2  9
>> C = [1 2 3]
C =
1  2  3
>> A .* C % A and C must have the same length
error: product: nonconformant arguments (op1 is 1x5,
  op2 is 1x3)
```

Matrix multiplication

- Matrix multiplication does not mean multiplying term by term
  - You'll study it in the Mathematics course
- Be careful to the dimension of the matrices
  - In MATLAB: `C = A * B`
  - The number of columns of A must be equal to the number of rows of B

$$[A]_{m \times n} * [B]_{n \times p} = [C]_{m \times p}$$

- Each element c_{ij} of the product matrix C is defined as:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Matrix multiplication (continue)

```
>> A = [2 5 7 1 3];
>> B = -1:1:3
B =
-1 0 1 2 3
>> A * B
error: operator *: nonconformant arguments (op1 is
1x5, op2 is 1x5)
>> A = [2 4; 1 5];
>> B = [1 3; 2 7];
>> C=A*B
C =
   10   34
   11   38
```

Matrix multiplication for vectors

- To multiply vectors they must have the same number of elements, but one must be a row vector and the other a column vector

```
>> r = [6 2 3 4];      % r is a row vector
>> c = [5 3 7 1]';     % r is a column vector
>> c * r  % the result is a 4x4 matrix
ans =
    30    10    15    20
    18     6     9    12
    42    14    21    28
     6     2     3     4
>> r * c  % dot product: the result is a scalar
ans =  61
```

Logical operations

```
>> A = [2 5 7 1 3];
>> B = [0 6 5 3 2];
>> A >= 5
ans =
0 1 1 0 0
>> A >= B
ans =
1 0 1 0 1
>> C = 1:3;
>> A > C
error: mx_el_gt: nonconformant arguments (op1 is
 1x5, op2 is 1x3)
```

Logical operations (continue)

```
>> A = [true true false false];
>> B = [true false true false];
>> A & B
ans =
1 0 0 0
>> A | B
ans =
1 1 1 0
>> C = [1 0 0];  % C is NOT a logical vector
>> A(C) % you can index logical vectors, but ...
error: subscript indices must be either positive
 integers less than 2^31 or logicals
```

A footnote: the `find` function

- Continuing the code in the previous slide:

```
>> C = find(B)  % C is a vector of indices of
  elements of B which are true
ans =
      [1 3]
```

- The **`find(...)`** built-in function returns the indices of a vector that meet some given criterium; it also works for matrices

```
>> vec = [11   -5   33    2    8   -4   25];
>> find(vec<0) % find the indices of the negative
elements of vec
ans =
   2   6
```

Applying library functions

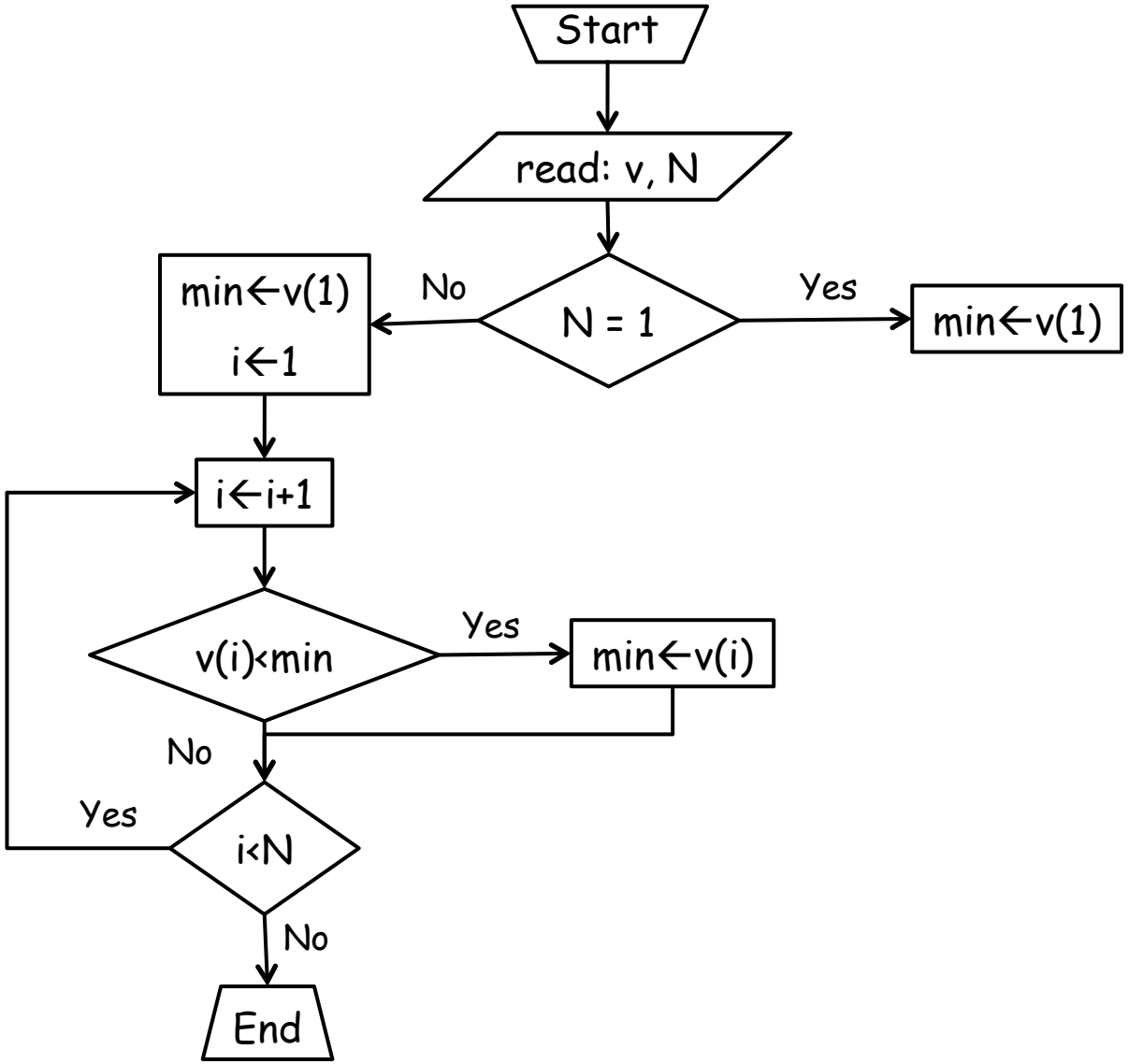
All MATLAB functions accept vectors of numbers and return a vector of the same length.

Special functions:

- **`sum(v)`** and **`mean(v)`** return a number, which is either the sum or the mean of the vector elements
- **`min(v)`** and **`max(v)`** return a number, which is either the minimum or maximum value in a vector
- **`round(v)`**, **`ceil(v)`**, **`floor(v)`**, and **`fix(v)`** remove the fractional part of the numbers in a vector by conventional rounding (round), rounding up (ceil), rounding down (floor), and rounding toward zero (fix)

Flow diagram: min(v)

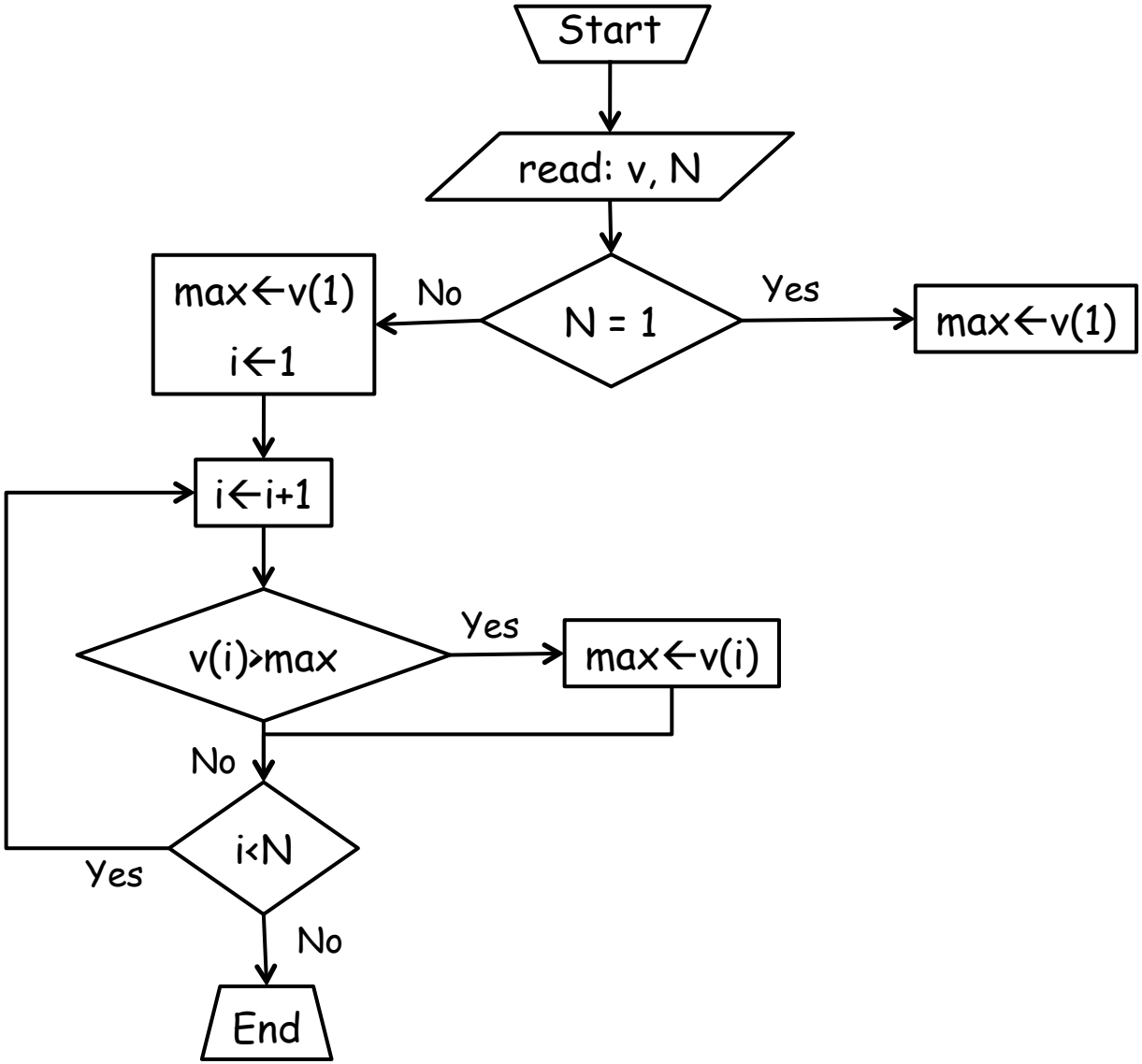
Find the minimum value of a vector



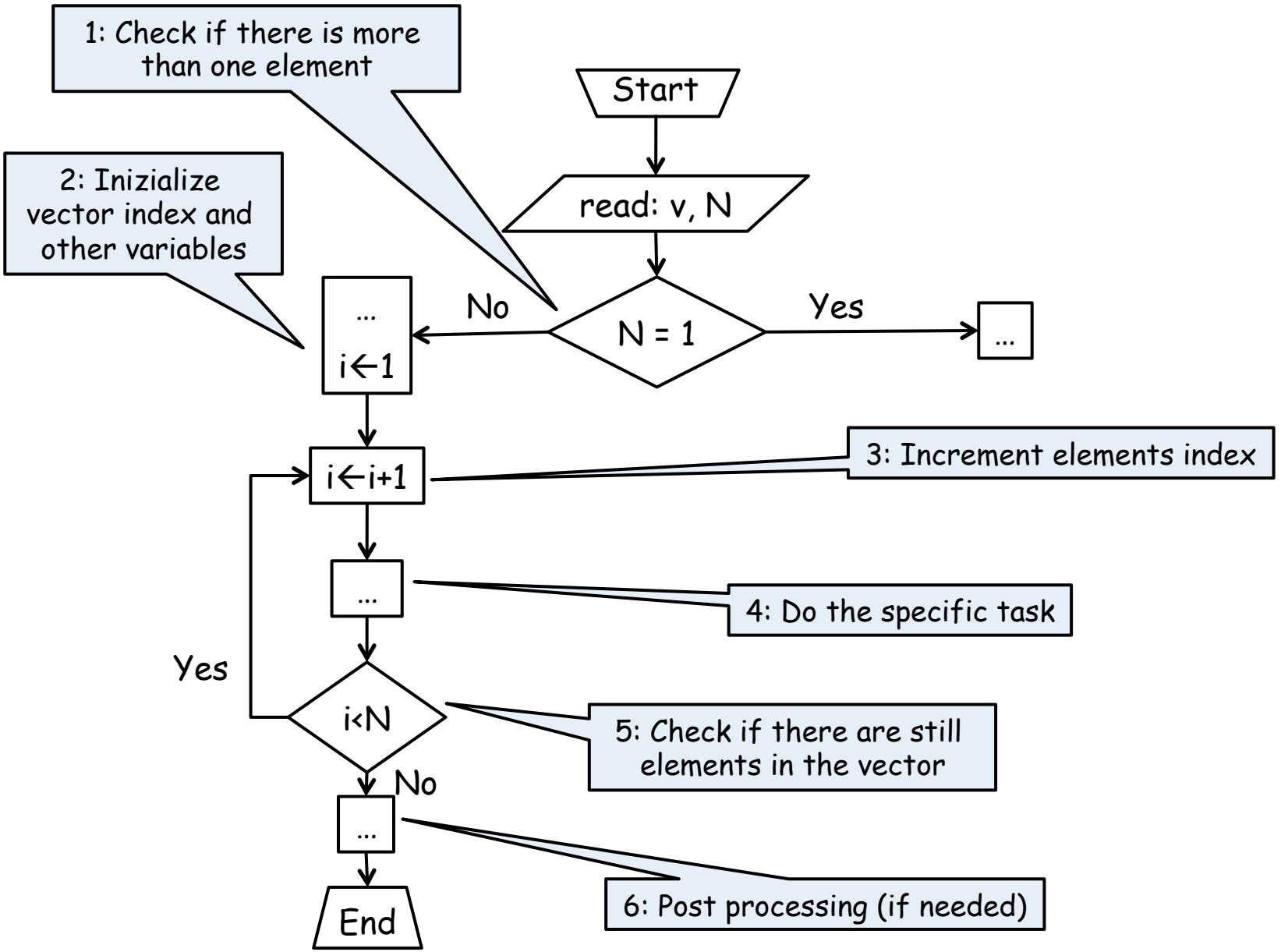
- v is the vector, N its length
- The algorithm assumes first element as minimum and then compares it with other elements
- If an element is smaller than the current minimum, it becomes the new minimum
- This process is repeated till complete array is scanned

Flow diagram: max(v)

Find the maximum value of a vector

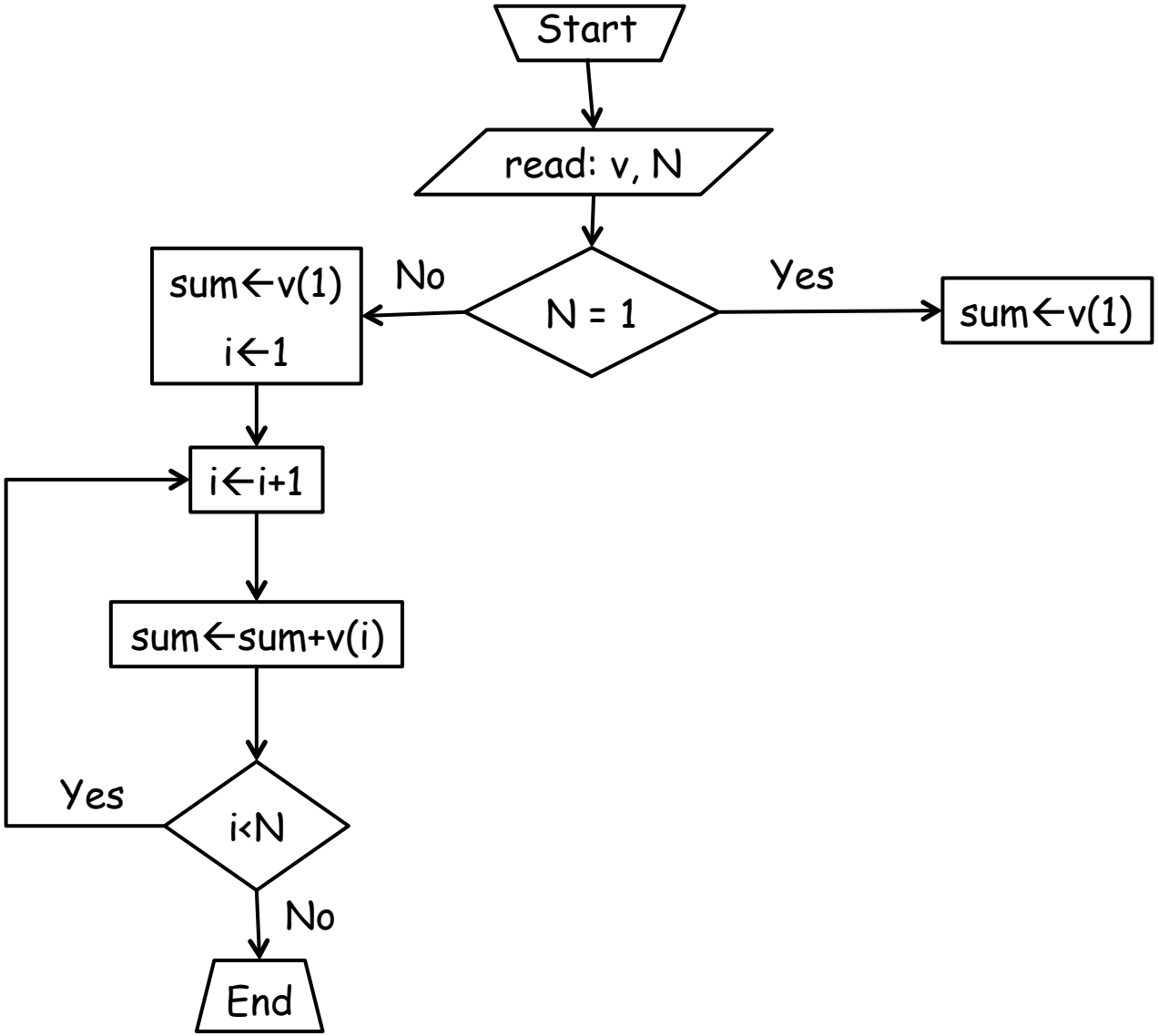


A skeleton to process vector's elements



Flow diagram: sum(v)

Sum the elements of a vector



Exercise: mean of the vector elements

- Write the flow diagram to compute the mean of the elements of vector v; the length of v is N
  - Read v and N from input
- Suggestion: start from the **sum** algorithm

Vector concatenation

- In MATLAB we can build a new vector by concatenating other vectors:
  - **A = [B C ... Y Z]**

    Individual items within brackets may be vectors: the length of A will be the sum of the lengths of the individual vectors

```
>> B = 3:-2:1;
>> C = [4 9 2];
>> D = linspace(1,2,3);
>> A = [B C D]
A =
3.0000    1.0000    4.0000    9.0000    2.0000
1.0000    1.5000    2.0000
```

  - **A = [1 2 3 42]**

    is a special case of concatenation where all the elements are scalars

Replacement rules for vectors

1. Either:
   - All dimensions of the blocks on either side of the replacement instruction must be equal, or
   - There must be a single element on the right side of the replacement
2. If you replace beyond the end of the existing vector, the vector length is automatically increased
   - Any element not specifically replaced remains unchanged
   - Elements beyond the existing length not replaced are set to 0

```
>> A=1:3:12
A =
     1     4     7    10
>> A(7)
error: A(I): index out
of bounds; value 7 out
of bound 4
>> B=[2,4];
>> A(B)=[0,0]
A =
     1     0     7     0
>> A(4)=99
A =
     1     0     7    99
>> A(7)=99
A =
     1     0     7    99
0     0    99
```

Generalized indexing for arrays

- A(4) actually creates an anonymous 1×1 index array, 4, and then use it to extract the specified element from array A
- In general,

`B(<rangeB>) = A(<rangeA>)`

where **`<rangeA>`** and **`<rangeB>`** are index arrays, A is an existing array, B can be an existing array or a new one

The values in B at the indices in **`<rangeB>`** are assigned the values of A from **`<rangeA>`**

Generalized indexing for arrays

```
>> A = [5 3; 2 4];
>> B = ones(2);
>> B(:,2) = A(:,1)
>> B =
    1    5
    1    2
>> C = -1:2:7
C =
   -1    1    3    5    7
>> C(2:3)=A(1,:)
>> C =
   -1    5    3    5    7
```

Operating on matrices

Four techniques extend directly from operations on vectors:

- Arithmetic operations
- Logical operations
- Applying library functions
- Generalized indexing

Concatenation and reshaping need some additional words

Matrix concatenation

- Matrix concatenation can be accomplished either horizontally or vertically:
  - `R = [A B C]` succeeds as long as A, B and C have the same number of rows; the columns in R will be the sum of the columns in A, B and C.
  - `R = [A; B; C]` succeeds as long as A, B and C have the same number of columns; the rows in R will be the sum of the rows in A, B and C.

Do as exercise

Reshaping matrices

- Matrices are actually stored in column order in MATLAB. So internally, a 2 × 3 matrix is stored as a column vector:

A(1,1)
A(2,1)
A(1,2)
A(2,2)
A(1,3)
A(2,3)

- Any n × m matrix can be reshaped into a p × q matrix as long as n*m = p*q using the built-in `reshape` function

```
>> A=[2 4 5 6; 3 5 6 7]
A =
     2     4     5     6
     3     5     6     7
>> idx=find(A>5)
idx =
     6
     7
     8
>> A(idx)=A(idx)+3
A =
     2     4     5     9
     3     5     9    10
>> A(6)
ans =
     9
>> B=rand(2)
B =
   2.7850e-01   9.5751e-01
   5.4688e-01   9.6489e-01
>> reshape(B,1,4)
ans =
   2.7850e-01   5.4688e-01
9.5751e-01   9.6489e-01
```