Module 05: Built-in Functions

Chul Min Yeum
Assistant Professor
Civil and Environmental Engineering
University of Waterloo, Canada

UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING

Module 5: Learning Outcomes

- Relate an array as an input argument of built-in functions
- Print elements in variables and format output texts
- Apply built-in arithmetic functions and replicate the same capability using selection and loop statements
- Use a sort function to sort array elements

Built-in Function (Array as an Input Argument)

round

R2018b

Round to nearest decimal or integer

Syntax

```
Y = round(X)
Y = round(X,N)
Y = round(X,N,type)

Y = round(t)
Y = round(t,unit)
```

Description

`Y = round(X)` rounds each element of `X` to the nearest integer. In the case of a tie, where an element has a fractional part of exactly `0.5`, the `round` function rounds away from zero to the integer with larger magnitude. example

`Y = round(X,N)` rounds to `N` digits: example

- `N > 0`: round to `N` digits to the *right* of the decimal point.
- `N = 0`: round to the nearest integer.
- `N < 0`: round to `N` digits to the *left* of the decimal point.

`Y = round(X,N,type)` specifies the type of rounding. Specify `'significant'` to round to `N` significant digits (counted from the leftmost digit). In this case, `N` must be a positive integer. example

Input Arguments

x — Input array

scalar | vector | matrix | multidimensional array

📖: It is frequently useful to conduct the function over each elements in an array. We do not have to implement a loop statement. You should check if the function support the array format.

Built-in Function (Array as an Input Argument) (Continue)

- Entire arrays (vectors or matrices) can be used as input arguments to functions; This is very powerful!
- The result will have the same dimensions as the input

Q. Write a code to round to the nearest integer of each values in the vector.

```
vec = [1.1 2.3 -3.1 4.7 8.9];
n_vec = numel(vec);

vec_r = zeros(1, n_vec);
for ii=1:n_vec
    vec_r(ii) = round(vec(ii));
End
```

```
vec = [1.1 2.3 -3.1 4.7 8.9];

vec_r = round(vec);
```

Name	Value
vec	[1.1 2.3 -3.1 4.7 8.9]
n_vec	5
vec_r	[1 2 -3 5 9]

Example: Built-in Function

Q. Write a code to compute a square root of each value in the vector.

```
vec = [3 2 4 1 2 4];
n_vec = numel(vec);

for ii=1:n_vec
    vec(ii) = sqrt(vec(ii));
end
```

```
vec = [3 2 4 1 2 4];

vec = sqrt(vec);
```

Q. Write a code to compute a sign of each values in a matrix

```
mat1 = [3 2 4; -1 -2 4; 3 0 -1];
[nr, nc] = size(mat1);
for ii=1:nr
    for jj = 1: nc
        val= sign(mat1(ii, jj));
        mat1(ii, jj) = val;
    end
end
```

```
mat1 = [3 2 4; -1 -2 4; 3 0 -1];
mat1 = sign(mat1);
```

3	2	4
-1	-2	4
3	0	-1

1	1	1
-1	-1	1
1	0	-1

Array Operation

Function	Description
`reshape(x, sz)`	Changes dimensions of a matrix to any matrix with the same number of elements
`flip(x, dim)`	Flips a row vector left to right, column vector or matrix up to down, the elements in each column or row in a matrix
`cat(dim, A, B)`	Concatenate arrays along specified dimension
`diag(x)` `diag(A)`	Create diagonal matrix or get diagonal elements of matrix

Function: `reshape(x, sz)`

`B = reshape(A,sz)` reshapes A using the size vector, `sz`, to define `size(B)`. `sz` must contain at least 2 elements, product of elements in `sz` must be the same as `numel(A)`.

```
vec = 3:14; % 12 elements

mat0 = reshape(vec, [2 6]);
mat1 = reshape(vec, [3 4]);
mat2 = reshape(mat1, [6 2]);
```

`mat1`

3	6	9	12
4	7	10	13
5	8	11	14

`vec`

3	4	5	6	7	8	9	10	11	12	13	14

`mat0`

3	5	7	9	11	13
4	6	8	10	12	14

`mat2`

3	9
4	10
5	11
6	12
7	13
8	14

Function: `flip(x, dim)`

`B = flip(A,dim)` reverses the order of the elements in A along dimension `dim`.

```
mat0 = reshape(3:14, [3 4]);
mat1 = flip(mat0, 1);
mat2 = flip(mat0, 2);
```

📖: `dim` - If no value specified, then the default is 1. 1 indicates the vertical direction and 2 indicates the horizontal direction.

2: horizontal →

1: vertical ↓

`mat0`

3	6	9	12
4	7	10	13
5	8	11	14

`mat1`

5	8	11	14
4	7	10	13
3	6	9	12

`mat2`

12	9	6	3
13	10	7	4
14	11	8	5

Function: `cat(dim, A, B)`

`C = cat(dim, A, B)` concatenates the arrays A and B along array the dimension specified by the number of dimensions.

```
mat0 = [1 1; 2 2];
mat1 = zeros(2,2);

mat2 = cat(1, mat0, mat1);
mat3 = cat(2, mat0, mat1);
mat4 = cat(3, mat0, mat1);
```

Name	Value
mat0	[1 1; 2 2]
mat1	[0 0; 0 0]
mat2	[1 1; 2 2; 0 0; 0 0]
mat3	[1 1 0 0; 2 2 0 0]
mat4	2 x 2 x 2 double

mat0

1	1
2	2

mat1

0	0
0	0

mat2

1	1
2	2
0	0
0	0

mat4

		0	0
			0
1	1		
2	2		

mat3

1	1	0	0
2	2	0	0

Function: `diag(x), diag(A)`

- `D = diag(v)` returns a square diagonal matrix D with the elements of vector v on the main diagonal.
- `x = diag(A)` returns a column vector of the main diagonal elements of matrix A.

```
vec = [1 2 3];
mat0 = [2 1 3; 4 5 2; 2 3 4];

D = diag(vec);
v = diag(mat0)
```

vec

1	2	3

mat0

2	1	3
4	5	2
2	3	4

D

1	0	0
0	2	0
0	0	3

v

2
5
4

Output Functions

- There are two basic output functions:
  - `disp`, which is a quick way to display things
  - `fprintf`, which allows formatting

- The `fprintf` function uses format specifiers which include place holders; these have conversion characters:

```
fprintf(formatspec, A1, A2)
```

  - %d : integers
  - %f : floats (real numbers)
  - %c : single characters
  - %s : string of characters

- \n newline character

📖: This function and syntax is to print out numeric values or texts in a command window.

Output Functions (Continue)

```
x = 1234.5678
v = [ 1 2 3 4]
```

`disp(x)`
`disp(v)`
`disp('Hello! Matlab')`
	`1.2346e+03`
	`1  2  3  4`
	`Hello! Matlab`

`fprintf('Hello! Matlab')`
`fprintf('Hello! %catlab', 'M')`
`fprintf('Hello! %slab', 'Mat')`
`fprintf('Hello! \nMatlab')`
	`Hello! Matlab`
	`Hello! Matlab`
	`Hello! Matlab`
	`Hello!`
	`Matlab`

```
x = 1234.5678
y = 10;
```

`fprintf('case1: x is %f.', x);` `fprintf('case2: y  is %d.', y);`	`case1: x is 1234.567800.` `case2: y is 10.`

fprintf(formatspec, A1, A2)

Sum, Average, Min & Max, and Sorting

Sum of array elements

- `S = sum(A)`
- `S = sum(A,'all')`
- `S = sum(A,dim)`

Average or mean value of array

- `S = mean(A)`
- `S = mean(A,'all')`
- `S = mean(A,dim)`

Median value of array

- `S = median(A)`
- `S = median(A,'all')`
- `S = median(A,dim)`

Maximum elements of an array

- `M = max(A)`
- `M = max(A, [], dim)`
- `[M, I] = max(A)`
- `C = max(A, B)`

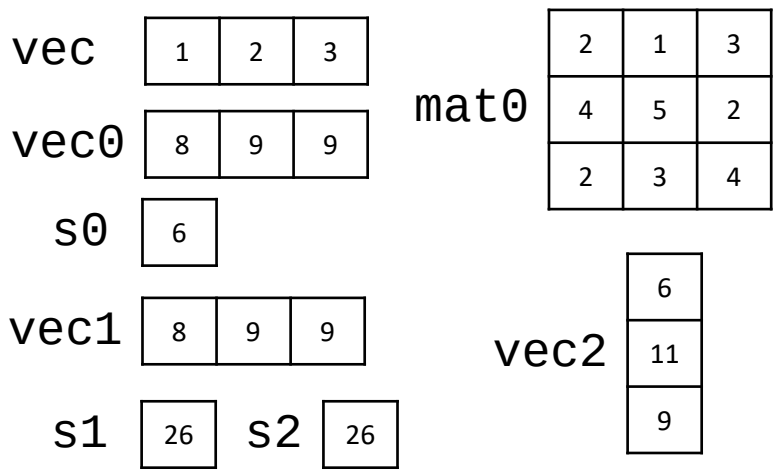
Sort array elements

- `B = sort(A)`
- `B = sort(A,dim)`
- `B = sort(A,direction)`
- `[B,I] = sort(A)`

Function: `sum`

- `S = sum(A)` returns the sum of the elements of A
  - If A is a vector, then `sum(A)` returns the sum of the elements.
  - If A is a matrix, then `sum(A)` returns a row vector containing the sum of each column (= `sum(A, 1)`).
- `S = sum(A,dim)` returns the sum along dimension `dim`.
- `S = sum(A,'all')` computes the sum of all elements of A.

```
vec = [1 2 3];
mat0 = [2 1 3; 4 5 2; 2 3 4];

s0 = sum(vec);
vec0 = sum(mat0);

vec1 = sum(mat0, 1);
vec2 = sum(mat0, 2);

s1 = sum(mat0, 'all');
s2 = sum(mat0(:));
```



Function: `mean`

- `M = mean(A)` returns the mean of the elements of A
  - If `A` is a vector, then `mean(A)` returns the mean of the elements.
  - If `A` is a matrix, then `mean(A)` returns a row vector containing the mean of each column.
- `M = mean(A,'all')` computes the mean over all elements of A.
- `M = mean(A,dim)` returns the mean along dimension `dim`.

```
vec = [1 2 3];
mat0 = [2 1 3; 4 5 2; 2 3 4];

s0 = mean(vec);
vec0 = mean(mat0);

vec1 = mean(mat0, 1);
vec2 = mean(mat0, 2);

s1 = mean(mat0, 'all');
s2 = mean(mat0(:));
```

vec

1	2	3

mat0

2	1	3
4	5	2
2	3	4

vec0

2.67	3	3

s0

2

vec1

2.67	3	3

vec2

2
3.67
3

s1

2.89

s2

2.89

Example: `sum` and `mean`

Q. Write a code to compute `vec1`, `vec2`, `s1`, `s2` from `mat0` without using `sum` and `mean`

```
mat0 = [2 1 3; 4 5 2; 2 3 4];

[nr, nc] = size(mat0);

s1 = 0;
vec1 = zeros(nr, 1);
for ii=1:nr
    for jj=1:nc
        s1 = s1 + mat0(ii,jj);
        vec1(ii) = vec1(ii) + mat0(ii, jj);
    end
end

s2 = s1/(nr*nc);
vec2 = vec1/nc;
```

```
mat0 = [2 1 3; 4 5 2; 2 3 4];

vec1 = sum(mat0, 2);
vec2 = mean(mat0, 2);

s1 = sum(mat0, 'all');
s2 = mean(mat0, 'all');
```


`mat0`

2	1	3
4	5	2
2	3	4

`vec1`

6
11
9

Function: `median`

- `M = median(A)` returns the median of the elements of A
  - If `A` is a vector, then `median(A)` returns the median of the elements.
  - If `A` is a matrix, then `median(A)` returns a row vector containing the median of each column.
- `M = median(A,'all')` computes the median over all elements of A.
- `M = median(A,dim)` returns the median along dimension `dim`.

```
vec = [1 2 3];
mat0 = [2 1 3; 4 5 2; 2 3 4];

s0 = median(vec);
vec0 = median(mat0);

vec1 = median(mat0, 1);
vec2 = median(mat0, 2);

s1 = median(mat0, 'all');
s2 = median(mat0(:));
```

vec	1	2	3
vec0	2	3	3
s0	2		
vec1	2	3	3
s1	3		
s2	3		

mat0	2	1	3
	4	5	2
	2	3	4

vec2	2
	4
	3

Function: `max`

- `M = max(A)` returns the maximum elements of an array.
  - If A is a vector, then `max(A)` returns the maximum of A.
  - If A is a matrix, then `max(A)` is a row vector containing the maximum value of each column.
- `M = max(A,[],dim)` returns the largest elements of A along dimension `dim`. For example, if A is a matrix, then `max(A,[],2)` is a column vector containing the maximum value of each row.
- `M = max(A,[],'all')` finds the maximum over all elements of A.
- `[M,I] = max(____)` finds the indices of the maximum values of A and returns them in output vector `I`, using any of the input arguments in the previous syntaxes. If the maximum value occurs more than once, then max returns the index corresponding to the <u>first occurrence.</u>
- `C = max(A,B)` returns an array with the largest elements taken from A or B.

Function: `max` (Continue)

```
vec = [1 3 2];
mat0 = [2 1 3; 4 5 2; 2 4 4];

s0 = max(1, 2);
s1 = max(vec);
[M, I1] = max(vec);

vec1 = max(mat0)
vec2 = max(mat0, [], 1);
vec3 = max(mat0, [], 2);

[M, I2] = max(mat0, [], 1);
[M, I3] = max(mat0, [], 2);

s2 = max(mat0, [], 'all')
s3 = max(mat0(:))
```

Name	Value
s0	2
s1	3
M	3
I1	2
vec1	[4 5 4]
vec2	[4 5 4]
vec3	[3; 5; 4]
I2	[2 2 3]
I3	[3; 2; **2**]
s2	5
s3	5

mat0

2	1	3
4	5	2
2	4	4

vec

1	3	2

I1

3

I2

2	2	3

I3

3
2
2

Example: `max`

Q. Write a code to find a maximum number of a given array without using `max`.

```
vec = [2 1 5 7 4 2 3 9 4 2];

n_v = numel(vec);

max_val = vec(1);

for ii=2:n_v
    if vec(ii) > max_val
        max_val = vec(ii);
    end
end
```

📖: Check if each number in an array is bigger than the current maximum value, `max_val`. If a new value is larger, the current maximum value is replaced to the new value. The current maximum value is initialized with the first value.

Q. How to change the code if the given array is a matrix?

2	1	5	7	4	2	3	9	4	2

Function: `min`

```
vec = [1 3 2];
mat0 = [2 1 3; 4 5 2; 2 2 4];

s0 = min(1, 2);
s1 = min(vec);
[M, I1] = min(vec);

vec1 = min(mat0)
vec2 = min(mat0, [], 1);
vec3 = min(mat0, [], 2);

[M, I2] = min(mat0, [], 1);
[M, I3] = min(mat0, [], 2);

s2 = min(mat0, [], 'all')
s3 = min(mat0(:))
```

vec

1	3	2

mat0

2	1	3
4	5	2
2	2	4

Name	Value
s0	1
s1	1
I1	1
vec1	[2 1 2]
vec2	[2 1 2]
vec3	[1; 2; 2]
I2	[1 1 2]
I3	[2; 3; 1]
s2	1
s3	1

I1

1

I2

1	1	2

I3

2
3
1

Function: `sort`

- `B = sort(A)` sorts the elements of A in ascending order.
  - If A is a vector, then `sort(A)` sorts the vector elements.
  - If A is a matrix, then `sort(A)` treats the columns of A as vectors and sorts the elements within each vector column.
- `B = sort(A,dim)` returns the sorted elements of A along dimension `dim`.
- `B = sort(___,direction)` returns sorted elements of A in the order specified by direction using any of the previous syntaxes. `'ascend'` indicates ascending order (the default) and `'descend'` indicates descending order.
- `[B,I] = sort(___)` also returns a collection of index vectors for any of the previous syntaxes. `I` is the same size as A and describes the arrangement of the elements of A into B along the sorted dimension.

Function: `sort` (Continue)

```
vec = [2 1 3];
mat0 = [2 1 3; 4 5 2; 2 3 4];

v0 = sort(vec);
v1 = sort(vec, 'descend');
[B1, I1] = sort(vec, 'descend');

mat2 = sort(mat0, 1);
[B2, I2] = sort(mat0, 1);

mat3 = sort(mat0, 2);
[B3, I3] = sort(mat0, 2);

mat4 = sort(mat0, 2, 'descend');
```

vec

2	1	3

v0

1	2	3

B1

3	2	1

v1

3	2	1

I1

3	1	2

mat2

2	1	2
2	3	3
4	5	4

I2

1	1	2
3	3	1
2	2	3

mat0

2	1	3
4	5	2
2	3	4

mat4

3	2	1
5	4	2
4	3	2

mat3

1	2	3
2	4	5
2	3	4

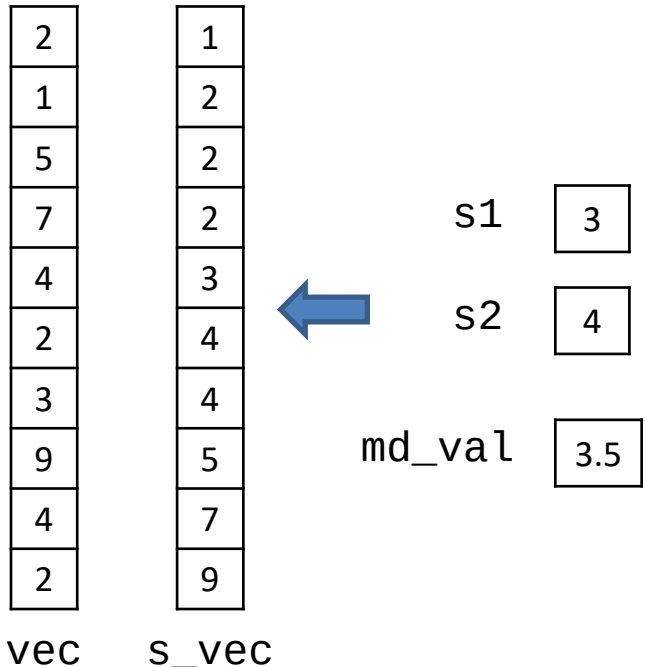
I3

2	1	3
3	1	2
1	2	3

Example: `median`

Q. Write a code to find a median of a given vector without using `median` (use `sort`).

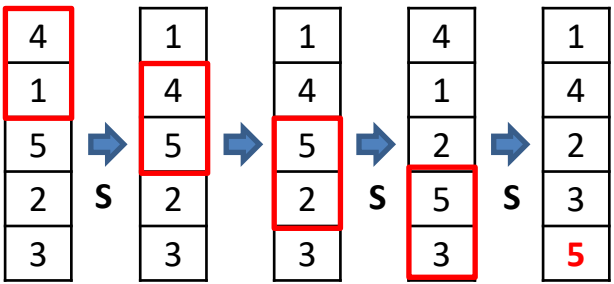
```
vec = [2 1 5 7 4 2 3 9 4 2]';

n_v = numel(vec);
s_vec = sort(vec);

if rem(n_v, 2) == 1
    idx = ceil(n_v/2);
    md_val = s_vec(idx);
else
    idx = n_v/2;
    s1 = s_vec(idx);
    s2 = s_vec(idx+1);
    md_val = (s1+s2)/2;
end
```



Sort Algorithm: Bubble Sort

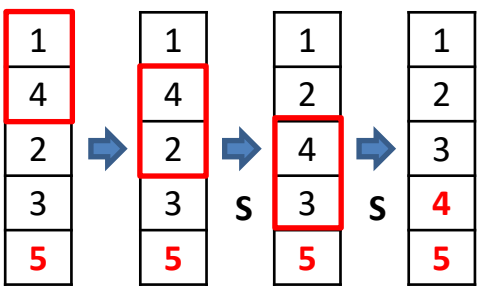
Challenging

Bubble sort, is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.



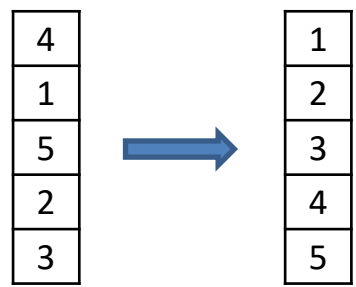
Step1: There are 4 conditions statements performed. At the end of the first step, we can find the last value.

Here '**S**' means swapping the values.



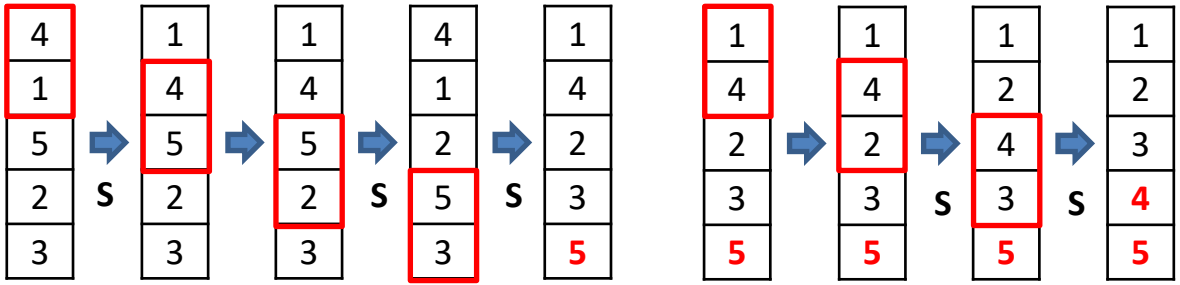
Step2: There are 3 conditions statements performed. At the end of the second step, we can find the 2nd last value.

Q. How to sort 'vec'



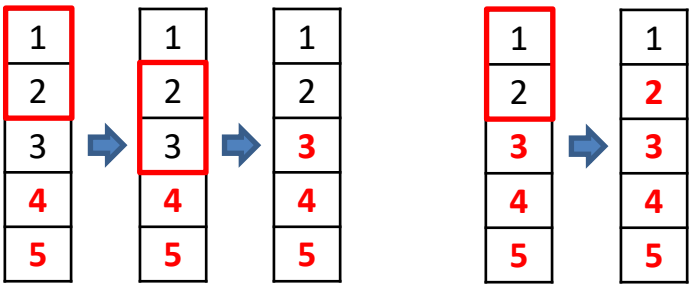
Sort Algorithm: Bubble Sort (Continue)

Challenging



Step1

Step2



Step3

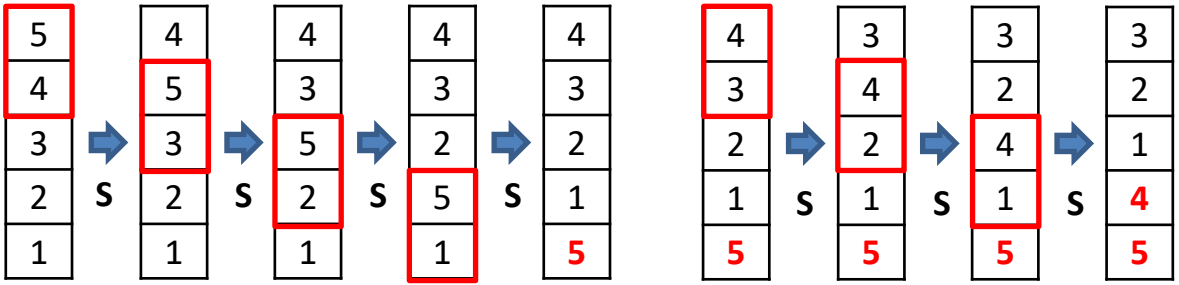
Step4

Summary

- 5 elements
- 4 steps needed
- Each steps, the number of condition statements is decreased by one.

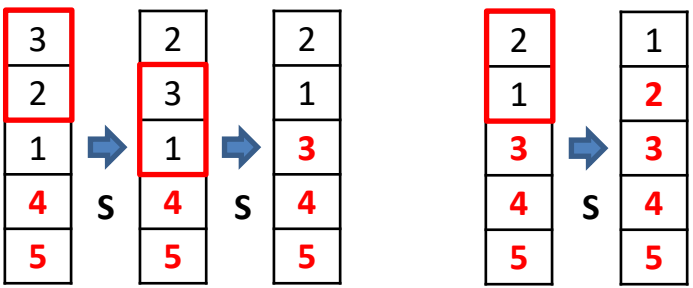
Sort Algorithm: Bubble Sort (Another Example)

Challenging



Step1

Step2



Step3

Step4

Summary

- 5 elements
- 4 steps needed
- Each steps, the number of actions is decreased by one.

Sort Algorithm: Bubble Sort (Implementation)

Challenging

Bubble sort, is a simple sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order. The pass through the list is repeated until the list is sorted.

```
vec = [4 1 5 2 3]';
nv = numel(vec);

for ii=1:nv-1
    for jj=1:nv-ii
        if vec(jj)>vec(jj+1)
            tmp = vec(jj);
            vec(jj) = vec(jj+1);
            vec(jj+1) = tmp;
        end
    end
end
```

4	1	1	4	1
1	4	4	1	4
5	5	5	2	2
2	2	2	5	3
3	3	3	3	**5**

At ii=1

1	1	1	1
4	4	2	2
2	2	4	3
3	3	3	**4**
5	**5**	**5**	**5**

At ii=2

😂: The script in italic & bold can be replaced as `vec([jj jj+1]) = vec([jj+1 jj]);`