

**Date: 30-10-2021**

## **Experiment 8**

**Aim:** To work with sparse arrays, cell arrays and structures in MATLAB.

**Apparatus:** MATLAB Software

**Objective:** To deal with special types of array like sparse arrays, cell arrays and structures.

**Problems:**

**Q-1.** Write a MATLAB function that will accept a cell array of strings and sort them into ascending order according to the lexicographic order of the ASCII character set. (You may use function `c_strncmp` from Chapter 6 for the comparisons if you wish.)

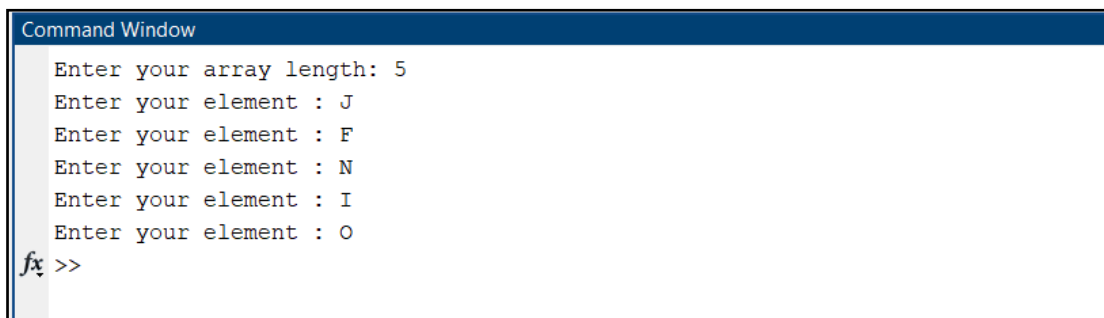
**Code:**

```
clc;
clear all;
close all;

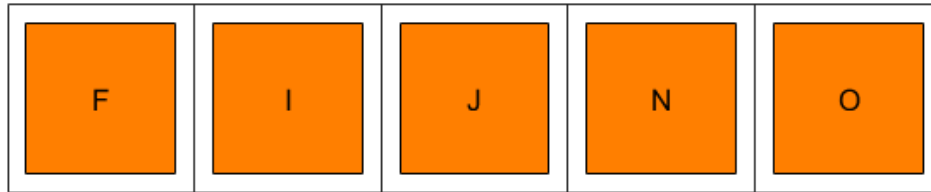
A = {};
c = input('Enter your array length: ');
array = sortcell(A,c);
cellplot(array)

function B = sortcell(A,n)
    for i = 1:n
        A{i} = input('Enter your element : ','s');
    end
    B = sort(A);
end
```

**Output:**



```
Command Window
Enter your array length: 5
Enter your element : J
Enter your element : F
Enter your element : N
Enter your element : I
Enter your element : O
fx >>
```



**Q-2.** Write a MATLAB function that will accept a cell array of strings and sort them into ascending order according to *alphabetical order*. (This implies that you must treat 'A' and 'a' as the same letter.)

### Code:

```
clc;
clear all;
close all;

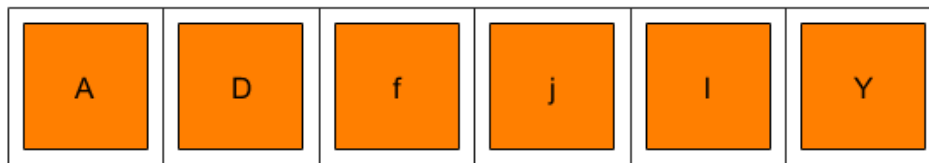
A = {};
c = input('Enter your array length: ');
array = sortcell(A,c);
cellplot(array)

function B = sortcell(A,n)
    for i = 1:n
        A{i} = input('Enter your element : ','s');
    end
    [~,idx]=sort(upper(A));
    B = A(idx);
end
```

### Output:

```
Command Window

Enter your array length: 6
Enter your element : A
Enter your element : j
Enter your element : I
Enter your element : Y
Enter your element : D
Enter your element : f
fx >>
```



**Q-3.** Create a sparse 100 x 100 array *a* in which about 5% of the elements contain normally distributed random values, and all of the other elements are zero (use function `sprandn` to generate these values). Next, set all of the diagonal elements in the array to 1. Next, define a 100-element sparse column array *b*, and initialize that array with 100 uniformly distributed values produced by function `rand`. Answer the following questions about these arrays:

- Create a full array *a\_full* from the sparse array *a*. Compare the memory required to store the full array and the sparse array. Which is more efficient?
- Plot the distribution of values in array *a* using function `spy`.
- Create a full array *b\_full* from the sparse array *b*. Compare the memory required to store the full array and the sparse array. Which is more efficient?
- Solve the system of equations  $a * x = b$  using both the full arrays and the sparse arrays. How do the two sets of answers compare? Time the two solutions. Which one is faster?

### Code:

```
clc
clear all
close all
```

*% Sparse array 5% random elements*

```
✓ a=sprandn(100, 100, 0.05);
figure(1)
spy(a, '.', 4) ✓
```

*% Mask for diagonal elements*

```
mask=logical(eye(100));
```

*% Assigning 1 to diagonal elements*

```
a(mask)=1;
```

```
b=sparse(rand(100, 1));
```

```

a_full=full(a);
whos a;
whos a_full;

figure(2)
spy(a, '*', 3)
title('Sparse Matrix Distribution');

b_full=full(b);
whos b;
whos b_full;

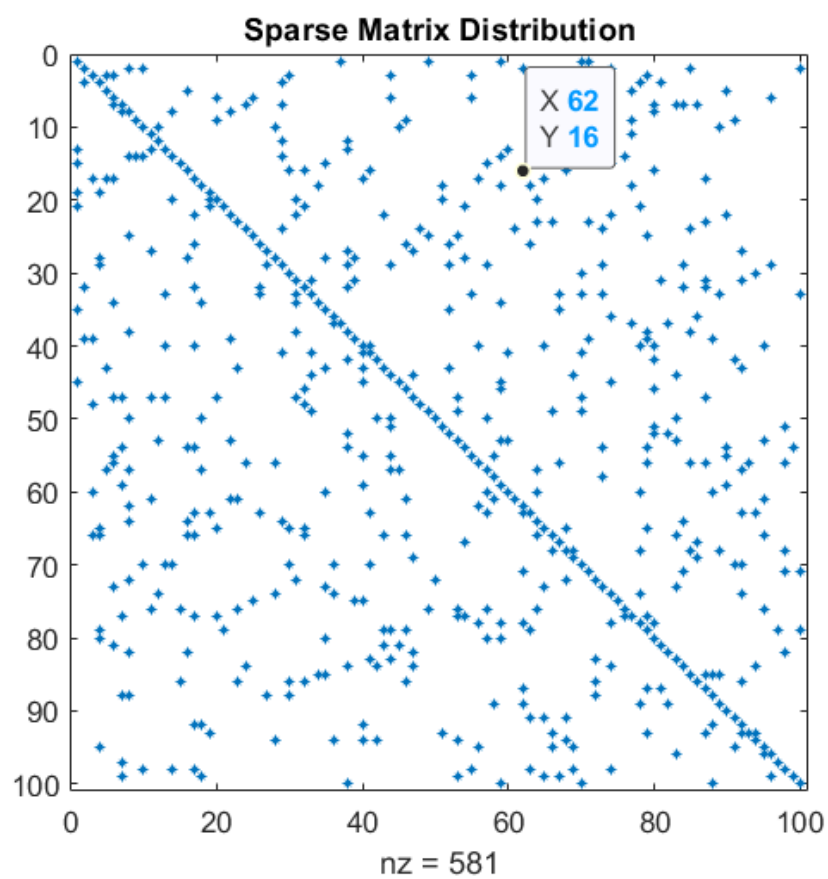
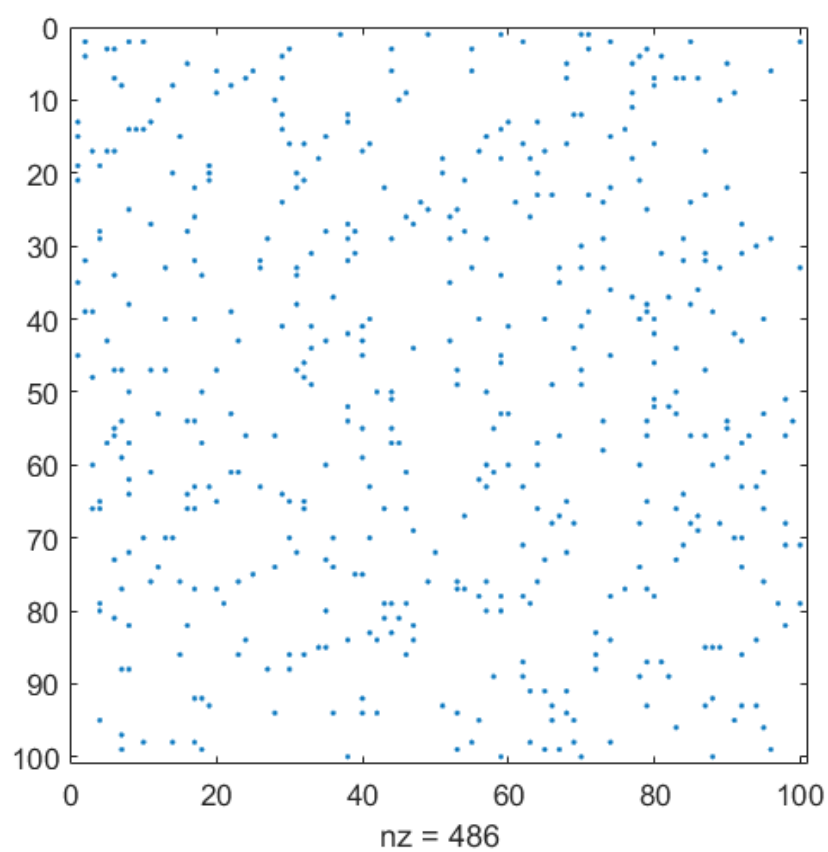
tic;
x1=b_full \ a_full;
t1=toc;
tic;
x2=b\a;
t2=toc;

t1
t2

```

## Output:

Command Window				
Name	Size	Bytes	Class	Attributes
a	100x100	10216	double	sparse
Name	Size	Bytes	Class	Attributes
a_full	100x100	80000	double	
Name	Size	Bytes	Class	Attributes
b	100x1	1616	double	sparse
Name	Size	Bytes	Class	Attributes
b_full	100x1	800	double	
t1 =				
	0.0034			
t2 =				
	0.0198			



✓ **Q-4.** Create a function that accepts any number of numeric input arguments and

sums up all of individual elements in the arguments. Test your

function by passing it the four arguments  $a=10$ ,  $b=\begin{bmatrix} 4 \\ -2 \\ 2 \end{bmatrix}$ ,  $c =$

$\begin{bmatrix} 1 & 0 & 3 \\ -5 & 1 & 2 \\ 1 & 2 & 0 \end{bmatrix}$  and  $d=[1 \ 5 \ -2]$ .

### Code:

```
clc
clear all
close all

a={10};
b={[4;-2;2]};
c={[1 0 3;-5 1 2;1 2 0]};
d={[1 5 -2]};
ans1=varsum(a, b, c, d);
fprintf("Sum: %d\n\n",ans1);

function ans1=varsum(varargin)
    ans1=0;
    for i=varargin
        e=i{:};
        celldisp(e)
        if(iscell(e))
            for j=e
                e2=j{:};
                ans1=ans1+sum(e2, "all");
            end
        end
    end
end
```

### Output:

```
Command Window

e{1} =

    10

e{1} =

     4
    -2
     2

e{1} =

     1     0     3
    -5     1     2
     1     2     0

e{1} =

     1     5    -2

fx Sum: 23
```

✓ **Q-5.** Modify the function of the previous exercise so that it can accept either ordinary numeric arrays or cell arrays containing numeric values. Test your

function by passing it the two arguments a and b, where  $c = \begin{bmatrix} 1 & 4 \\ -2 & 3 \end{bmatrix}$ ,  $b\{1\} = [1 \ 5 \ 2]$ , and  $b\{2\} = \begin{bmatrix} 1 & -2 \\ 2 & 1 \end{bmatrix}$ .

### Code:


```
clc
clear all
close all

a=[1 4: -2 3];
b{1}=[1 5 2];
b{2}=[1 -2; 2 1];
```

```
ans1=varsum(a, b);
fprintf("Sum: %d\n\n",ans1);
```

```
function ans1=varsum(varargin)
    ans1=0;
    for i=varargin
        w=i{:};
        if(iscell(w))
            for j=w
                w2=j{:}
                ans1=ans1+sum(w2, "all"); ✓
            end
        else
            ans1=ans1+sum(w, "all"); ✓
        end
    end
end
```

## Output:



```
Command Window

w2 =

     1     5     2

w2 =

     1    -2
     2     1

Sum: 14
```

**Q-6.** Create a structure array containing all of the information needed to plot a data set. At a minimum, the structure array should have the following fields:

- x\_data- x-data (one or more data sets in separate cells)
- y\_data -y-data (one or more data sets in separate cells)
- type -linear, semilogx, and so forth
- plot\_title -plot title
- x\_label -x-axis label
- y\_label -y-axis label
- x\_range -x-axis range to plot
- y\_range -y-axis range to plot



You may add additional fields that would enhance your control of the final plot.

After this structure array has been created, create a MATLAB function that accepts an array of this structure and produces one plot for each structure in the array. The function should apply intelligent defaults if some data fields are missing. For example, if the `plot_title` field is an empty matrix, then the function should not place a title on the graph. Think carefully about the proper defaults before starting to write your function!

To test your function, create a structure array containing the data for three plots of three different types and pass that structure array to your function. The function should correctly plot all three data sets in three different figure windows.

### Code:

```
clc
clear all
close all

graph(1).x_data=[1 2 3 4 5 6 7 8];
graph(1).y_data=[1 2 3 4 5 6 7 8];
graph(1).type="linear";
graph(1).title="Linear of y=x";

graph(1).x_label="X";
graph(1).y_label="Y";

graph(1).x_range=[0 9];
graph(1).y_range=[0 9];

graph(2).x_data=[10 10^2 10^3 10^4 10^5 10^6 10^7 10^8];
graph(2).y_data=[1 2 3 4 5 6 7 8];
graph(2).type="semilogx";
graph(2).title="Semilogx of y=x";

graph(2).x_label="X";
graph(2).y_label="Y";
graph(2).x_range=[0 10^9];
graph(2).y_range=[0 9];

graph(3).x_data=[1 2 3 4 5 6 7 8];
graph(3).y_data=[1 2 3 4 5 6 7 8];
graph(3).type="stem";
graph(3).title="Stem of y=x";
```

```

graph(3).x_label="X";
graph(3).y_label="Y";

graph(3).x_range=[0 9];
graph(3).y_range=[0 9];
plt(graph);

function k=plt(graph)
    n=size(graph);
    k=n(2);
    for i=1:k
        figure(i)
        if(strcmp(graph(i).type,"linear"))
            for j=1:length(graph(i).x_data)

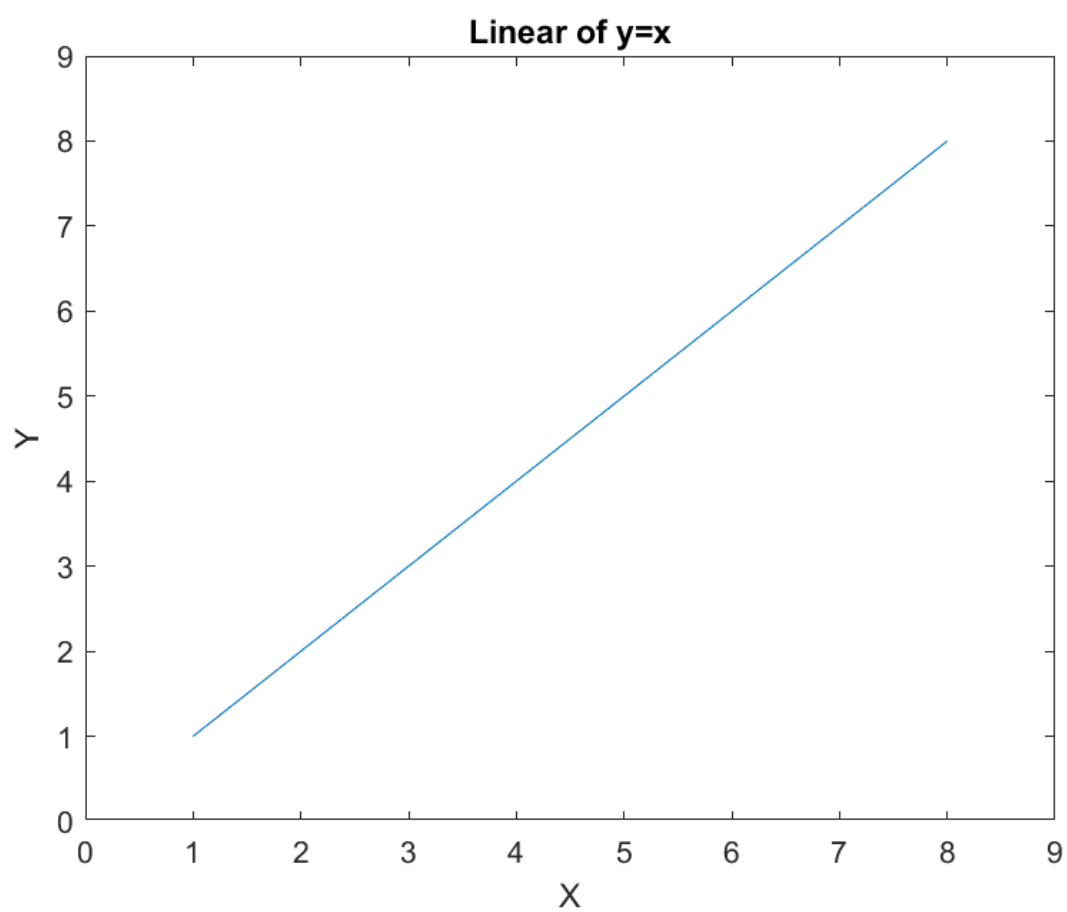
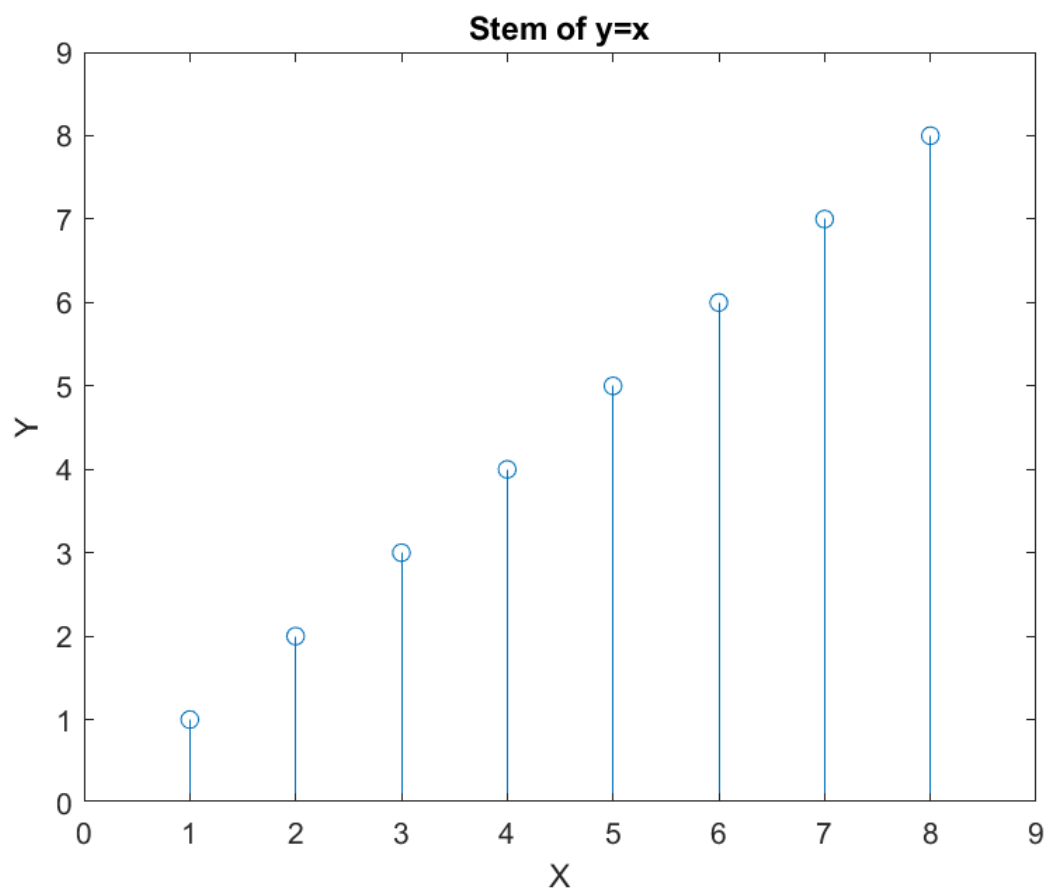
plot(graph(i).x_data{j},graph(i).y_data{j})
            end
            % a=1
            elseif(strcmp(graph(i).type,"semilogx"))
                for j=1:length(graph(i).x_data)

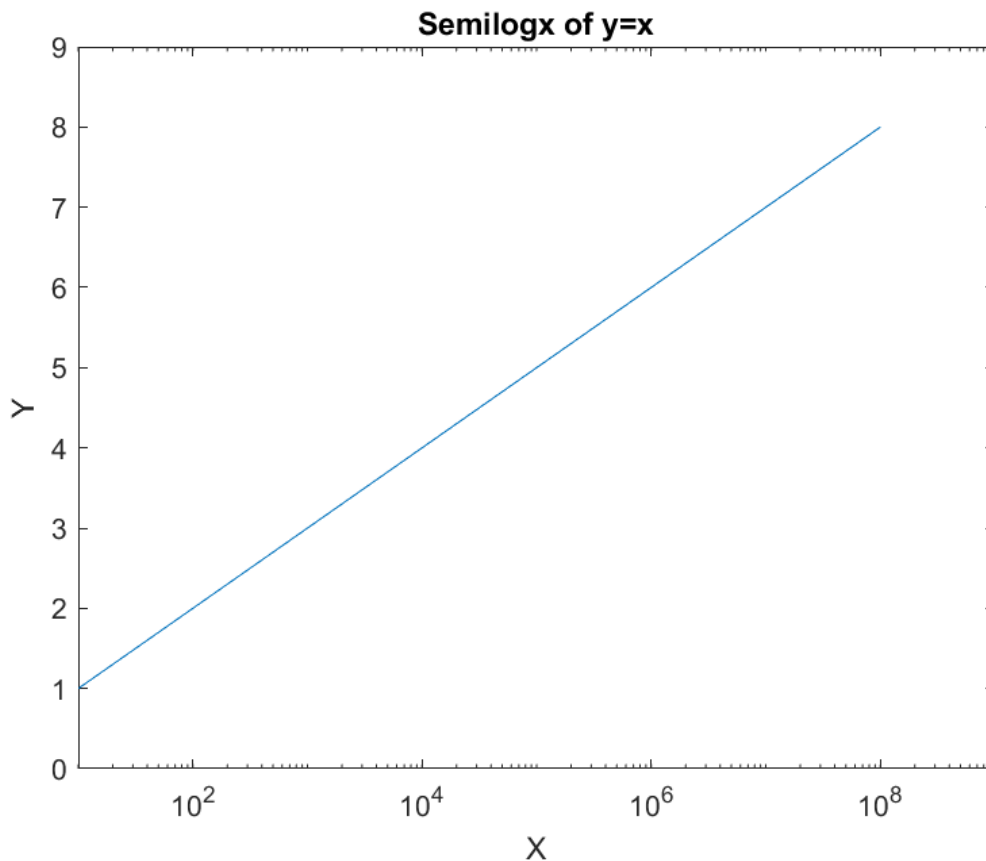
semilogx(graph(i).x_data{j},graph(i).y_data{j})
                end
            elseif(strcmp(graph(i).type,"stem"))
                for j=1:length(graph(i).x_data)

stem(graph(i).x_data{j},graph(i).y_data{j})
                end
            end
        title(graph(i).title)
        xlabel(graph(i).x_label)
        ylabel(graph(i).y_label)
        xlim(graph(i).x_range)
        ylim(graph(i).y_range)
    end
end

```

**Output:**





✓ **Q-7.** Define a structure **point** containing two fields x and y. The x field will contain the x-position of the point, and the y field will contain the y-position of the point. Then write a function **dist3** that accepts two points and returns the distance between the two points on the Cartesian plane. Be sure to check the number of input arguments in your function.

### Code:

```
clc
clear all
close all

point.x=[3 6 9 5 2];
point.y=[4 8 12 12 2];

for i=1:length(point.x)
    ans(i)=dist3(point.x(i), point.y(i));
end

disp(ans)

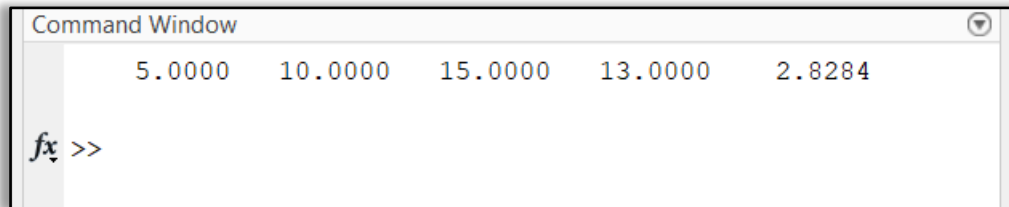
function dist=dist3(x, y)
```

```

    if(nargin==2)
        dist=sqrt(x^2+y^2);
    else
        dist=-1;
    end
end

```

### Output:



**Q-8.** Write a function that will accept a structure as an argument and return two cell arrays containing the names of the fields of that structure and the data types of each field. Be sure to check that the input argument is a structure and generate an error message if it is not.

### Code:

```

clc
clear all
close all

```

```

struct.name='Kanisha';
struct.roll=253;

```

```

struct.marks=[99 97 100];

```

```

[s1, s2]=fun(struct);
s1
s2

```

```

function [s1, s2] = fun(struct)
    msg=nargchk(1,1,nargin);
    error(msg);
    if(isstruct(struct))
        s1=fieldnames(struct);
        n=size(s1);
        s2={};
        for i=1:n(1)
            s2{i}=class(struct.(s1{i}));
        end
    end

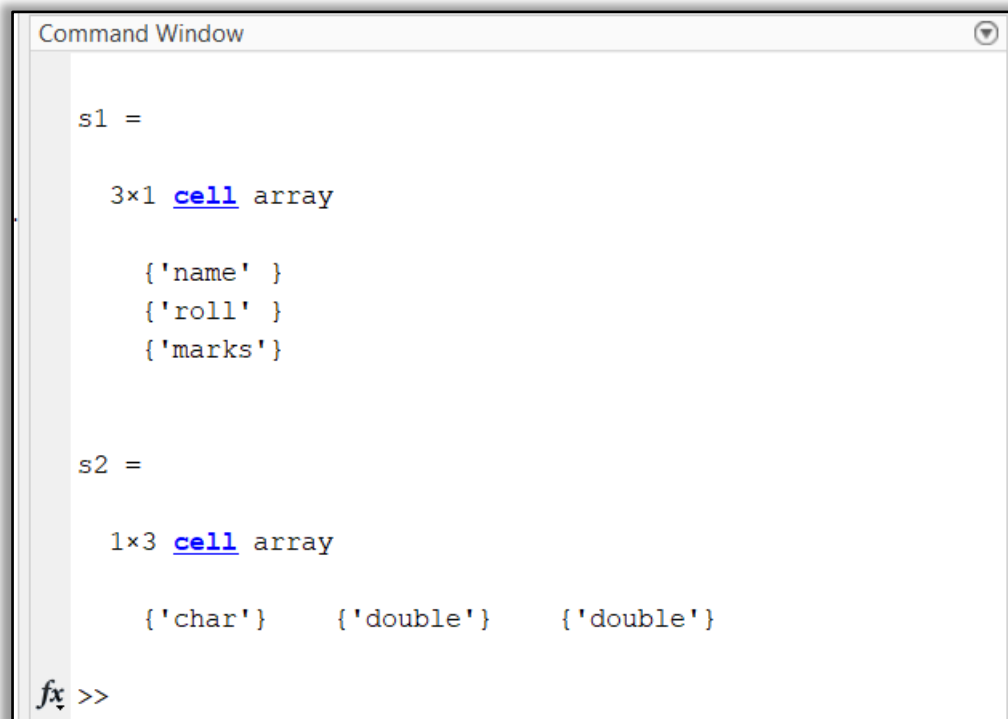
```

```

        end
        s2=s2;
    else
        error("Only Structures allowed.");
    end
end

```

## Output:



```

Command Window

s1 =

3x1 cell array

    {'name' }
    {'roll' }
    {'marks' }

s2 =

1x3 cell array

    {'char'}    {'double'}    {'double'}

fx >>

```

**Q-9.** Write a function that will accept a structure array of student as defined in this chapter, and calculate the final average of each one, assuming that all exams have equal weighting. Add a new field to each array to contain the final average for that student, and return the updated structure to the calling program. Also, calculate and return the final class average.

## Code:

```

clc
clear all
close all

student(1).marks=[90 92 95];
student(2).marks=[85 93 89];
student(3).marks=[93 94 99];

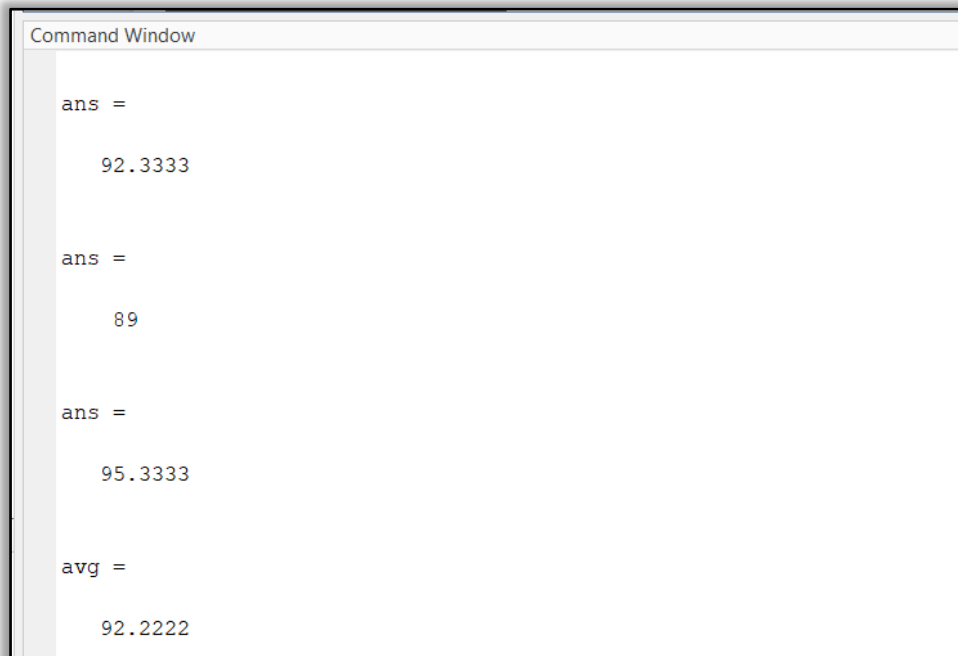
```

```
[student, avg]=fun(student);  
student.average
```

```
avg
```

```
function [student, avg] = fun(student)  
    msg=nargchk(1,1,nargin);  
    error(msg);  
    avg=0;  
    for i=1:length(student)  
  
student(i).average=sum(student(i).marks)/length(student(i).marks);  
    avg=avg+sum(student(i).marks);  
    end  
    avg=avg/(length(student)*length(student(1).marks));  
end
```

### Output:



```
Command Window  
  
ans =  
  
    92.3333  
  
ans =  
  
    89  
  
ans =  
  
    95.3333  
  
avg =  
  
    92.2222
```

✓ **Q-10.** Write a function that will accept two arguments—the first a structure array, and the second a field name stored in a string. Check to make sure that these input arguments are valid. If they are not valid, print out an error message. If they are valid and the designated field is a string, concatenate all of the strings in the specified field of each element in the array, and return the

resulting string to the calling program. Be sure that the function works properly for both lowercase and uppercase characters.

### Code:

```
clc
clear all
close all

student(1).name="Kanisha";
student(1).address="Paldi";
student(1).city="Ahmedabad";

student(1).marks=[93 94 88];

student(2).name="Prapti";
student(2).address="Satellite";
student(2).city="Ahmedabad";
student(2).marks=[91 79 86];

student(3).name="Kahan";
student(3).address="Vasna";
student(3).city="Ahmedabad";

student(3).marks=[100 96 81];

struct_strcat(student,'names');
struct_strcat(student,'marks');

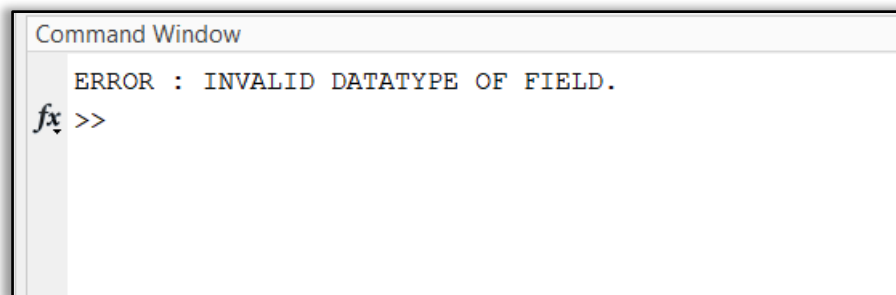
function out_str=struct_strcat(structure,field)
    msg=nargchk(2,2,nargin);
    error(msg);
    arr=' ';
    if isstruct(structure)
        if ischar(field)
            if isfield(structure,field)
                if ischar(structure(1).(field))
                    for ii=1:length(structure)
                        arr=strcat(arr,structure(ii).(field));
                    end
                    out_str=arr;
                end
            else
                disp("ERROR : INVALID DATATYPE OF FIELD.")
            end
        else
            % Empty field
        end
    end
```



```
        disp("ERROR : INVALID FIELD OF STRUCTURE.")
    end

    else
        disp("ERROR : INVALID INPUT SECOND INPUT SHOULD BE STRING.")
    end
end
```

### Output:



### Conclusion:

From this experiment we learnt about sparse matrix, variety of functions are executed for cell arrays and also, the Structure and structure arrays.