DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

LAB MANUAL

Academic Year: 2015-16 ODD SEMESTER

Programme (UG/PG)	**:**	**UG**
Semester	**:**	**I**
Course Code	**:**	**15CS101L**
Course Title	**:**	**PROGRAMMING LAB**

Prepared By

G. SIVAGAMI

(Assistant Professor, Department of Computer Science and Engineering)

FACULTY OF ENGINEERING AND TECHNOLOGY
SRM UNIVERSITY
(Under section 3 of UGC Act, 1956)
SRM Nagar, Kattankulathur- 603203
Kancheepuram District

LIST OF EXPERIMENTS & SCHEDULE

Course Code: 15CS101L

Course Title: PROGRAMMING LAB

Exp. No.	Title	Week No.
1	STUDY OF BASIC SCILAB COMMANDS	
2	MATRIX CONSTRUCTORS AND OPERATIONS	
3	MATRIX BITWISE, RELATIONAL & LOGICAL OPERATIONS	
4	CONTROL STRUCTURES (If-Else, If-elseif –else, Select)	
5	CONTROL STRUCTURES (for, while, break and continue)	
6	GRAPHICS - 2D PLOTS	
7	SCILAB – CIVIL APPLICATION PROGRAM (1)	
8	SCILAB – CIVIL APPLICATION PROGRAM (2)	
9	SCILAB – ELECTRONICS APPLICATION PROGRAM (1)	
10	SCILAB – ELECTRONICS APPLICATION PROGRAM (2)	

COURSE COORDINATOR **HOD**

HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS:

No Specific Hardware Requirements

SOFTWARE REQUIREMENTS:

SCILAB Software – Version 5.5.2

INTERNAL ASSESSMENT MARK SPLIT UP

Observation	:	20 Marks
Mini Project with the Report (Max. 8 Pages & 3 Students per Batch)	:	20 + 5 (Report) Marks
Model Exam	:	15 Marks
TOTAL MARKS	:	60 Marks

STUDY OF BASIC SCILAB COMMANDS

EXP. NO: 1

i. OBJECTIVE: Practicing MATLAB environment with simple exercises to familiarize Command Window, History, Workspace, Current Directory, Figure window, Edit window, Shortcuts, Help files.

ii. SOURCE CODE :

Working on general commands on scilab environment

Help – detailed help menu for scilab commands
Who – list all the variables from the variable browser window
Whos - list all the variables with byte size, variable type etc.
Clc – Clears command window
Clear – Removes the variable from memory
Quit – to close the session
Pwd – present working directory
Ls – list the files
Ls -ltr – list the detailed view on the files
Cd - to change the directory
Mkdir – to create a new directory

To work on Special variables / Pre-Defined Variables
%pi – 3.14
Ans
% e = 2.718
%eps – epsilon
%inf – infinity

Basic Scalar & Vector Operations

Creation of Scalar Elements
Y = [1 4 6] ➔ Declares a row vetor
yT = [1; 4; 6] ➔ Declares a Column vector
Creation of Vector Elements
Y = [1 4 6; 2 7 3; 4 1 1]
➔ Creates a 3*3 matrix

To determine the size / order of the vectors.
Size(y)

To change the elements in the given vector
vector(i,j) = value

Performing element by element operations using dot operator

.*, ./, .^,

Linspace[a,b,N] ➔ Vector can be created by evenly spaced points. From a to b the vector is created by 'N' evenly spaced points

Eg: linspace[0,1,5] → 0 0.25 0.5 0.75 1

Transpose of a matrix – y'

ans =

1. 2. 4.
4. 7. 1.
6. 3. 1.

RESULT

Study of basic SCILab commands are worked and executed

MATRIX CONSTRUCTORS AND OPERATIONS

EXP. NO: 2

i. OBJECTIVE: To study on basic Matrix Constructors and Operations.

ii. SOURCE CODE :

Zeros(m,n) – creates m rows with n cols

```
zeros(3,2)
 ans  =
    0.    0.
    0.    0.
    0.    0.
```

Eye(m,n) – creates identity matrix

```
eye(2,3)
 ans  =
    1.    0.    0.
    0.    1.    0.
```

Ones(m,n) – creates matrix with all 1's for all m rows and n cols

```
ones(2,2)
 ans  =
1.    1.
1.    1.
```

rand(m,n) – creates matrix with random numbers

```
rand(4,4)
 ans  =
    0.2113249    0.6653811    0.8782165    0.7263507
    0.7560439    0.6283918    0.0683740    0.1985144
    0.0002211    0.8497452    0.5608486    0.5442573
    0.3303271    0.6857310    0.6623569    0.2320748
```

Max(z) -- returns the largest element in a vector.

```
Y  =
    1.    4.    6.
```

```
    2.    7.    3.
    4.    1.    1.
max(Y)
 ans  =
     7.
```

Min(z) - returns the smallest element in a vector.

```
min(Y)
 ans  =
    1.
```

Sum(z) – returns the sum of elements in a vector

```
sum(Y)
 ans  =
     29.
```

prod(z) – returns the product of all elements in a vector.

```
prod(Y)
 ans  =
   4032.
```

Mathematical operations

Sin(z) – Retrieve the sine value for the given matrix / vector

```
sin(Y)
 ans  =

    0.8414710  - 0.7568025  - 0.2794155
    0.9092974    0.6569866    0.1411200
  - 0.7568025    0.8414710    0.8414710
```

Similar operation can be obtained for Cos, tan, sec, csc, cot. The hyperbolic for sine, cosine etc can be retrieved using sinh, cosh etc.

Inverse of cosine, sine can be obtained using the acos, asin etc

Exp(z) – Returns the exponential, element wise

```
exp(Y)
 ans  =

    2.7182818   54.59815    403.42879
    7.3890561   1096.6332    20.085537
    54.59815    2.7182818   2.7182818
```

Log10(z) and log(z) provides the base 10 & natural logarithm for the given vector and matix

```
log(Y)
 ans  =
    0.          1.3862944   1.7917595
    0.6931472   1.9459101   1.0986123
    1.3862944   0.          0.
```

Sqrt(z) provides the square root for the matrix elements.

```
sqrt(Y)
 ans  =
    1.          2.          2.4494897
    1.4142136   2.6457513   1.7320508
    2.          1.          1.
```

Performing addition, subtraction, multiplication and division for array vectors or matrix elements

Floating Point operations

Working on floating point operations like ceil, floor, fix, round for both vectors and matrices

nthroot(x,n) -- Is the real vector/matrix of the nth root of the x elements

```
nthroot(Y,4)
 ans  =
    1.          1.4142136   1.5650846
```

1.1892071 1.6265766 1.316074

1.4142136 1. 1.

Sign(z) -- returns the matrix made of the signs of z(i,j)

sign(Y)

ans = 1. 1. 1.

1. 1. 1.

1. 1. 1.

Modulo(n,m) – computes n\m and gives the remained.

pmodulo(n,m) – positive arithmetic remainder modulo .

Cat -- Concatenate several arrays

Cat(1,y,z) – concatenates the array / vector ‘y’ with ‘z’ row wise

Y = 1. 4. 6.

2. 7. 3.

4. 1. 1.

Z =

9. 8. 7.

5. 6. 4.

3. 2. 1.

cat(1,Y,Z)

ans =

1. 4. 6.

2. 7. 3.

4. 1. 1.

9. 8. 7.

5. 6. 4.

3. 2. 1.

Cat(2,y,z) – concatenates the array / vector ‘y’ with ‘z’ column wise

Matrix Analysis Commands

It helps in finding determinant, Rank and sum of eigen values

y = [1 -2; -2 0]

det(y) = -4

rank(y) = 2

trace(y) = 1 [sum(diag(x))]

spec(y) ➔ will provide the eigen values of matrix

= -1.5615528

2.5615528

RESULT :

Thus the Matrix constructors and operations are successfully executed

MATRIX BITWISE, RELATIONAL & LOGICAL OPERATIONS

EXP. NO: 3

i. **OBJECTIVE: To study on Matrix Bitwise operations, Relational Operations and Logical Operations.**

ii. **SOURCE CODE :**

Relational operators: < <= > >= == ~=

```
X=5;       % X=5*ones(3,3);

X >=[1 2 3;4 5 6;7 8 9];
Output:
T  T  T
T  T  F
F  F  F


x<=[1 2 3; 4 5 6; 7 8 9];
Output:

F F F
F T T
T T T

x<[1 2 3; 4 5 6; 7 8 9];
Output:

F F F
F F T
T T T

x~=[1 2 3; 4 5 6; 7 8 9];
 Output:

T T T
T F T
T T T
```

LOGICAL OPERATORS:

```
a=0;b=10;
if a and b
   disp("Condition is true");
else
   disp("Condition is false");
end
```

```
if a or b
   disp("Condition is true");
end
if (~a)
   disp("Condition is true");
end

exec('C:\Users\admin\Documents\relational.sce', -1)
```

Condition is false

Condition is true

BITWISE OPERATORS:

```
U = [0 0 1 1 0 1];
V= [0 1 1 0 0 1];
>> U |V
Output:

Ans =
F T T T F T

a = 60; % 0011 1100
b = 13; % 0000 1101

c=bitand(a,b);        % 12 = 0000 1100
ans  =

   12.

d=bitor(a,b); % 61 = 00111101
ans  =

   61.

e= bitxor(a,b); % 49 = 00110001
ans  =

   49.
```

RESULT :

The study on Relational, logical and bitwise operations on matrices is performed.

CONTROL STRUCTURES (If-Else, If-elseif –else, Select)

EXP. NO: 4

i. OBJECTIVE: To write and execute programs that demonstrate on Control Structures (If-Else, If-elseif –else, Select) using SCI Notes.

ii. ALGORITHM:

STEP 1: Start the program

STEP 2: Get the input from user using input() method.

STEP 3: pmodulo()_gives the positive remainder of the number.

STEP 4: pmodulo(number,2) tells if the number is divisible by 2. By which the the given number odd or even is determined.

STEP 5: Using select statement multiple cases are executed.

STEP 6: The dayNum returns the number for the given system date ie. Sun is considered 1, Mon = 2 , Tue = 2 etc.

STEP 7: The dayString returns the day ie. Mon,Tue etc.

STEP 8: Using dayString, the different cases are dealt and the statements are dealt accordingly.

STEP 9: A number is taken as input and checked for positive or negative or zero using the if-elseif – else condition.

iii. SOURCE CODE :

If- Else Program

To find whether a number is an even number or not

```
a=input("Enter a number:");
if pmodulo(a,2)==0
disp("Number is even");
else
disp("Number is odd");
end
```

SELECT STATEMENTS:

To print on what day we are in a week

```
[dayNum,dayString]=weekday(datenum());
 select dayString
        case "Mon" then
              disp("Start of work week");
        case "Tue" then
              disp("Day2");
        case "Wed" then
              disp("Day3");
        case "Thu" then
              disp("Day4");
       case "Fri" then
              disp("Last day of work week");
 else
       disp("Weekend");
  end
```

If-elseif –else condition:

To determine whether a number is +ve or –ve or zero

```
number = input("Enter a number:");
if number >0
   disp("positive");
elseif number< 0
   disp("negative");
else
   disp("Zero");
end
```

iv. SAMPLE INPUTS & OUTPUTS:

If- Else Program

```
exec('C:\Users\admin\Documents\if.sce', -1)
Enter a number:5

 Number is odd

-->exec('C:\Users\admin\Documents\if.sce', -1)
Enter a number:6

 Number is even
```

SELECT STATEMENTS:

```
-->exec('C:\Users\admin\Documents\select.sce', -1)

 Start of work week
```

If-elseif –else condition:

```
exec('C:\Users\admin\Documents\nestedif.sce', -1)
Enter a number:4

 positive

-->exec('C:\Users\admin\Documents\nestedif.sce', -1)
Enter a number:0

 Zero

-->exec('C:\Users\admin\Documents\nestedif.sce', -1)
Enter a number:-7

 negative
```

RESULT :

The programs are executed using if-else, select, if-elseif-else statements.

CONTROL STRUCTURES (for, while, break and continue)

EXP. NO: 5

i. **OBJECTIVE: To write and execute programs that demonstrate on Control Structures (for, while, break and continue) using SCI Notes.**

ii. **ALGORITHM:**

STEP 1: Start the program.

STEP 2: For the given user input number, the factorial is to be found.

STEP 3: The variable fact is initialized to 1 and the input is stored in variable n.

STEP 4: The for loop is executed till and is multiplied with fact variable repeatedly.

STEP 5: The same functionality is executed using while loop.

STEP 6: For break and continue statement execution, the input is got from the user.

STEP 7: If the input is a positive number, the sum is calculated repeatedly, else it prompts for input if a negative number is entered.

STEP 8: If the input is zero, the program is stopped by printing the final sum.

iii. **SOURCE CODE :**

FOR LOOP:

To find factorial of given number

```
function fact = factorial(n)
fact =1;
for i=1:n
        fact=fact*i;
end
fprintf('Factorial of %d is %d\n',n,fact);
end
```

WHILE LOOP:

To find factorial of given number

```
function fact = factorial(n)
fact =1;
i=1;
while i<=n
   fact=fact*i;
   i=i+1;
end
fprintf('Factorial of %d is %d\n',n,fact);
end
```

BREAK AND CONTINUE STATEMENTS:

To find sum of all positive numbers entered by user (enter '0' to terminate)

```
a=1;
sum=0;
while a
   n=input('Enter a number:');
   if n>0
      sum=sum+n;
   elseif n<0
      disp('Enter a positive number.');
      continue;
   else
      break;
   end
end
printf('Sum of all positive numbers is %d',sum);
```

iv. SAMPLE INPUTS & OUTPUTS:

For loop:

```
>> factorial(5)

ans  =

    120.
```

While loop:

```
>> factorial(5)

ans  =

    120.
```

BREAK AND CONTINUE STATEMENTS:

```
Enter a number:2

Enter a number:-2

 Enter a positive number.

Enter a number:1

Enter a number:0

Sum of all positive numbers is 3
```

RESULT:

The programs are executed using for, while and break-continue statements.

GRAPHICS - 2D PLOTS

EXP. NO: 6

i. OBJECTIVE: To work on basic graphics -- 2D Plots

ii. ALGORITHM:

STEP 1: The x-axis and y-axis range is defined.

STEP 2: Using plot function, the values are plotted.

STEP 3: The title for the graph, x-axis and y-axis label can also be provided.

STEP 4: On top of the previous, various graphs can also be plotted using multiple plot commands.

STEP 5: The graph gets populated in the figure window.

STEP 6: The sub plots can also be plotted such that the figure window can be divided either row or column wise and the graphs can be populated.

STEP 7: The pie chart can be generated using the command pie.

iii. SOURCE CODE :

Plotting a single plot on the graph

Defining the x axis range

```
X = 0:1:10
Y = X.^2-10.*X+15
```

Plotting the graph

```
plot(X,Y);
```

Providing title, x & y axis label

```
title("X^2-10X+15");
xlabel("X");
ylabel("Y");
```

Multiple plots on the same graph

```
x = 0:1:10;
y = cos(x);
y1 = sin(x);
plot(x,y,x,y1);
plot(x,y,'ys:',x,y1,'gd--');
```

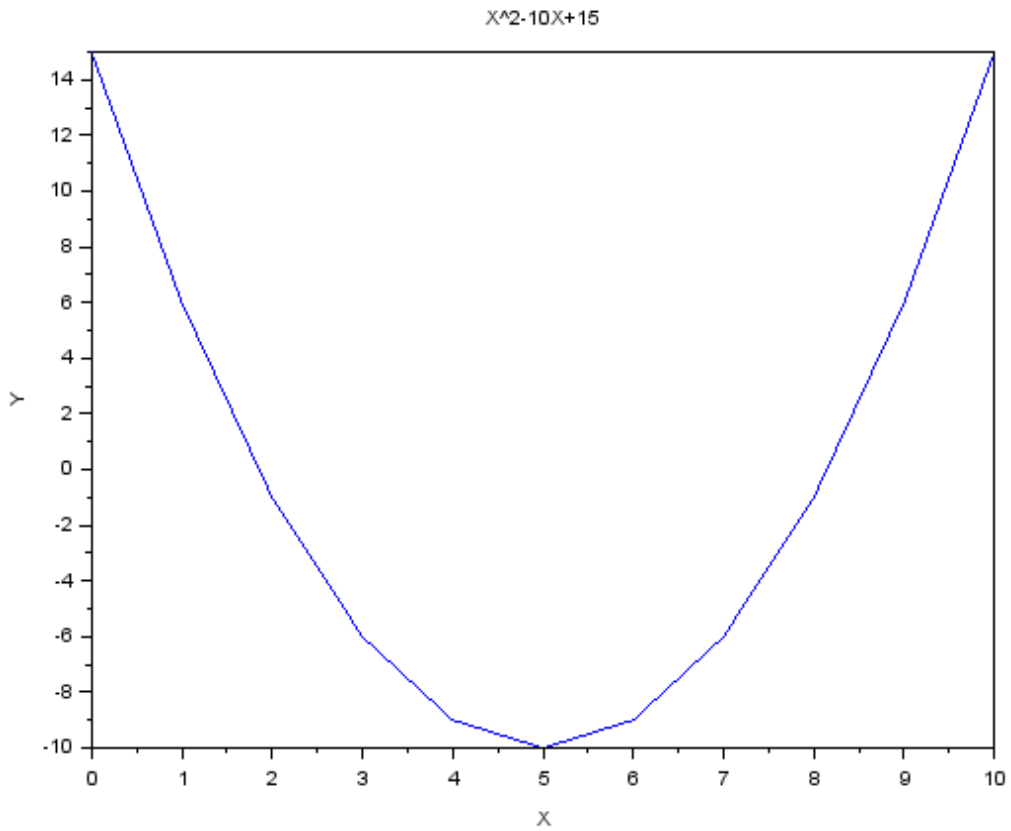
SUB PLOTS:

```
f=input('enter the frequency');
a=input('enter the amplitute');
t=0:.01:2;
y=a*sin(2*%pi*f*t);
y1=a*cos(2*%pi*f*t);
y2=t;
subplot(3,1,1);
plot(t,y);
subplot(3,1,2);
plot(t,y1);
subplot(3,1,3);
plot(t,y2);
```

Pie:

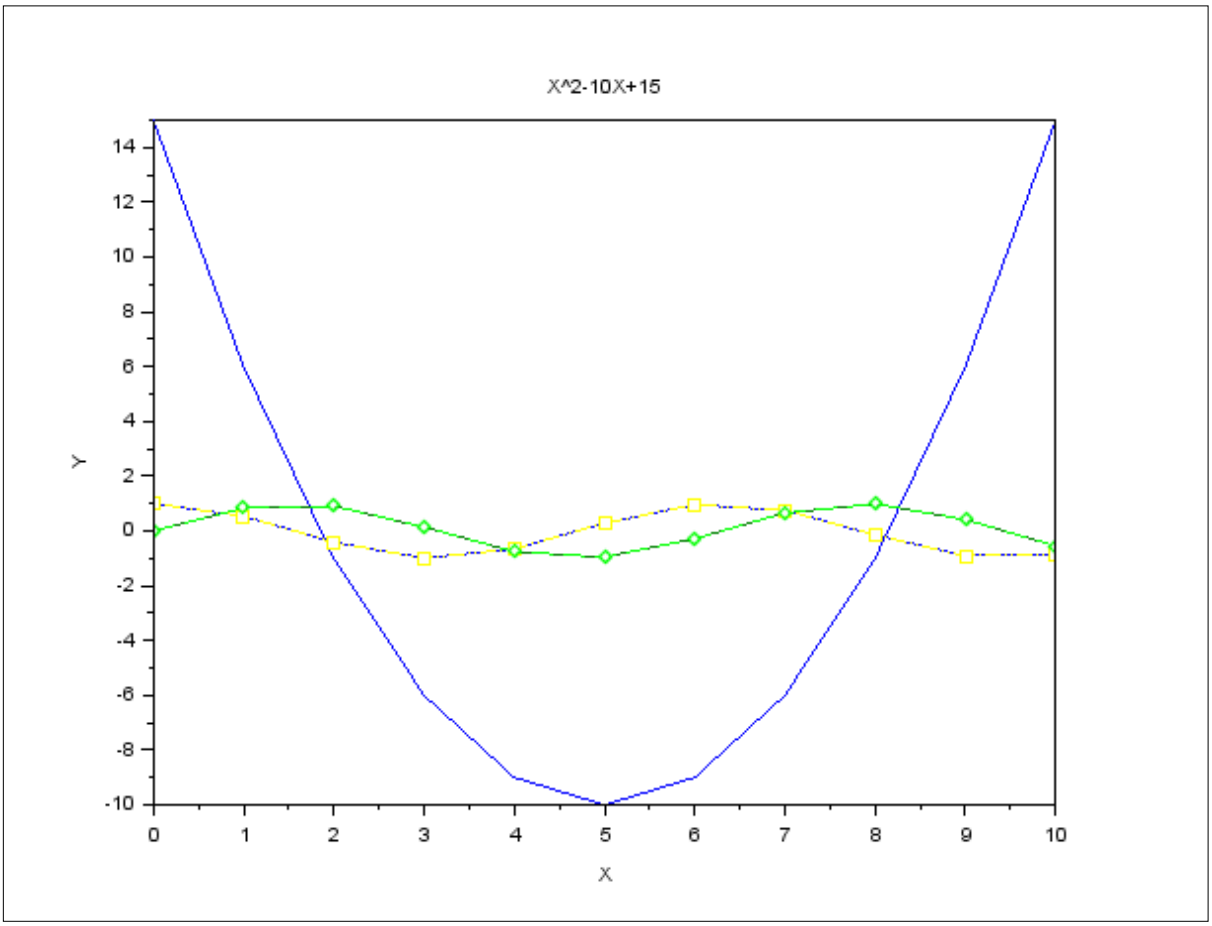
```
pie([1 2 5]);
```

iv. SAMPLE INPUTS & OUTPUTS:



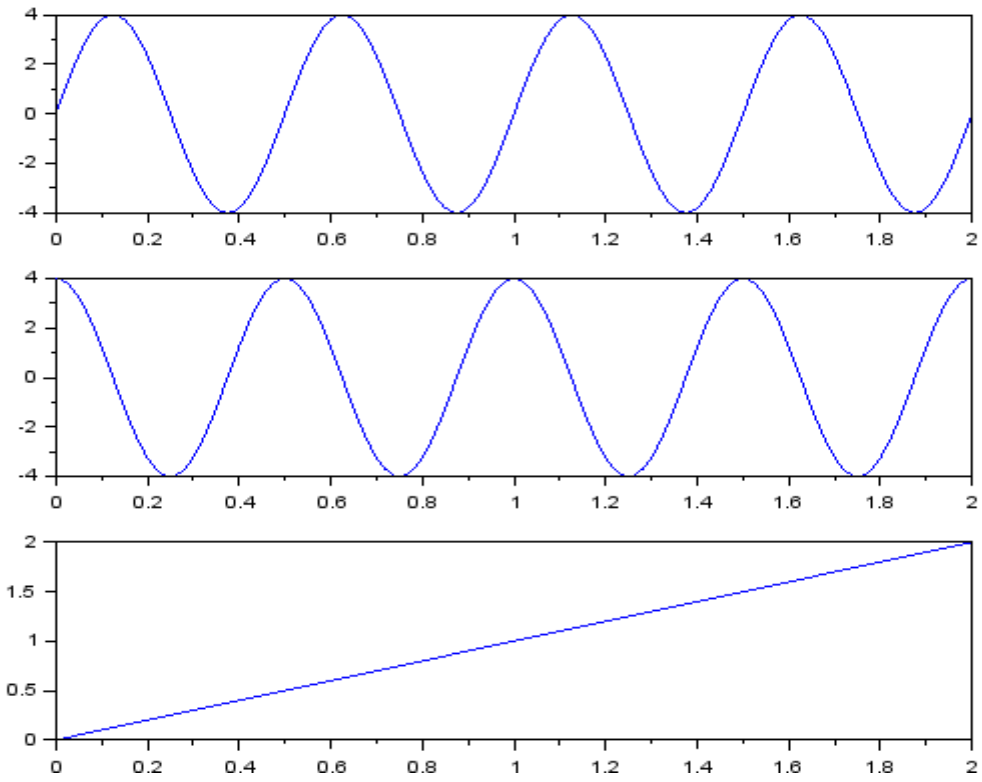
PLOTTING A SINGLE PLOT ON THE GRAPH

MULTIPLE PLOTS ON THE SAME GRAPH

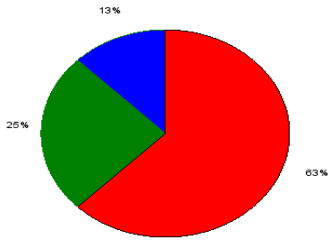


SUBPLOTS:

enter the frequency 2
enter the amplitude 4



PIE:



RESULT :

Thus the two dimensional graphs are plotted using SCILAB commands.

SCILAB – CIVIL APPLICATION PROGRAM (1)

EXP. NO: 7

i. **OBJECTIVE** To develop a program that finds out whether a tank is overflowing or not wrt the shape of the tank, its dimensions and rate of flow.

ii. **ALGORITHM**

STEP 1: Assume tank of shape rectangular, cylindrical or any other shape. Assume its dimensions also.

STEP 2: Calculate volume of the tank. For e.g. Assuming the tank is cylindrical, then *V_tank = h Where ‘r’ – radius of tank (m) Where ‘h’ – height of tank (m)

STEP 3: Calculate volume of liquid. V_liq = F x t Where ‘F’ - rate of flow (m3 /min) Where ‘t’ – time taken (min)

STEP 4: Conditions If V_liq > V_tank Tank is Overflow If V_liq < V_tank Tank is not Overflow

iii. **SOURCE CODE**

```
F=input('Enter the Value of Flow Rate:');
t=input('Enter the time to fill the Tank:');
r=input('Enter the Radius of the Tank:');
h=input('Enter the Height of the Tank:');
Vtank=%pi*r*r*h;
disp('Vtank:');
disp(Vtank);
Vliquid=F*t;
disp('Vliquid');
disp(Vliquid);
if Vliquid>Vtank then
   disp('Tank is Overflow');
else
   disp('Tank is not Overflow');
end
```

iv. **SAMPLE INPUTS & OUTPUTS**

INPUT

Enter the Value of Flow Rate: 10

Enter the time to fill the Tank: 2

Enter the Radius of the Tank: 3

Enter the Height of the Tank: 4

OUTPUT

Vtank:

113.09734

Vliquid

20.

Tank is not Overflow

RESULT :

Thus the program is executed for checking if the tank is over flown or not.

SCILAB – CIVIL APPLICATION PROGRAM (2)

EXP. NO: 8

i. OBJECTIVE: Write a program to find the structural stability of the given truss.

ii. ALGORITHM:

STEP 1: Assume any definite shape (shapes made of straight lines). Examples are: M

STEP 2: Stability of the truss shall be determined using the formula, m = 2j-3

Where 'm' – No. of members in the given structure (nos.)

'j' – No. of joints in the given structure (nos.)

STEP 3: Conditions

m < 2j-3 Unstable [Deficient Truss]

m = 2j-3 Stable or Statically determinate [Perfect Truss]

m > 2j-3 Statically indeterminate [Redundant Truss]

iii. SOURCE CODE :

```
M=input('Enter the Number of Members:');
J=input('Enter the Number of Joints:');
N=2*J-3;
if M==N then
   disp('The Given Struture is Stable:');
elseif M>N then
   disp('The Given Struture is Indetermine:');
else
   disp('The Given Struture is Unstable:');
end
```

iv. SAMPLE INPUTS & OUTPUTS:

INPUT (Assuming triangle)

Enter the Number of Members:3

Enter the Number of Joints:3

OUTPUT: The Given Structure is Stable

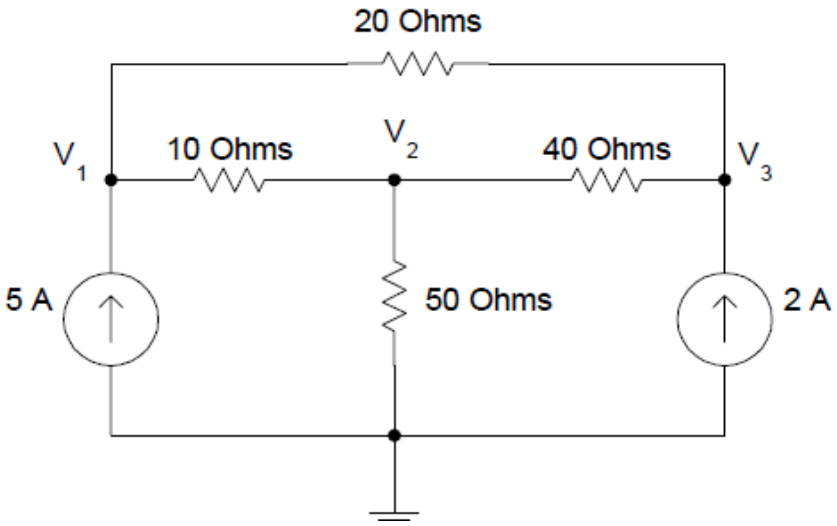
RESULT :

For any given structure, the stability of the structure will be calculated and displayed

SCILAB – ELECTRONICS APPLICATION PROGRAM (1)

EXP. NO : 9

Write a scilab program to find the nodal voltages *V1*, *V2* and *V*3 for the circuit shown below.



ii: **ALGORITHM:**

STEP 1: **using** Kirchhoff's current law the algebraic sum of all the currents at any node in the circuit equals zero.

STEP2: In nodal analysis, if there are n nodes in a circuit select a reference node, the other nodes can be numbered from *V1* through *Vn-1*.

STEP3: With one node selected as the reference node, there will be n-1 independent equations. If we assume that the admittance between nodes i and j is given as *Yij* , we can write the nodal equations:

STEP4: Y11 V1 + Y12 V2 + … + Y1m Vm = ∑□I1
Y21 V1 + Y22 V2 + … + Y2m Vm = ∑□I2
Ym1 V1 + Ym2 V2 + … + Ymm Vm = ∑□Im (4.1)
Where $m = n - 1$

V1, *V2* and *Vm* are voltages from nodes 1, 2 and so on ..., *n* with respect to the reference node.

STEP5: ∑*Ix* is the algebraic sum of current sources at node x.

Equation (4.1) can be expressed in matrix form as

□*Y*□□*V* □□□□□□*I* □□ (4.2)

The solution of the above equation is

□*V* □□□□□*Y*□$^{□□}$□□*I* □where (4.3)

□*Y*□$^{□□}$ is an inverse of □*Y*□□.

In MATLAB, we can compute [V] by using the command

$V = inv(Y) * I$ (4.4)

where

$inv(Y)$ is the inverse of matrix Y

The matrix left and right divisions can also be used to obtain the nodal voltages. The following Scilab commands can be used to find the matrix [V]

$V=I\backslash Y$ (4.5)

or

$V = Y \backslash I$ (4.6)

iii. SOURCE CODE :

PROGRAM

```
Y = [0.15 -0.1 -0.05;-0.1 0.145 -0.025;-0.05 -0.025 0.075];
I = [5; 0; 2];
V = inv(Y)*I
fid= mopen('volatage.txt','w');
mfprintf (fid, 'Nodal voltages V1,V2 and V3 are\n');
mclose(fid);
```

iv: SAMPLE INPUTS & OUTPUTS:

INPUT: Y, I (the inputs are there in the program)

OUTPUT: (to see the output just type V and enter in console screen)

```
V  =

  404.28571
  350.
  412.85714
```

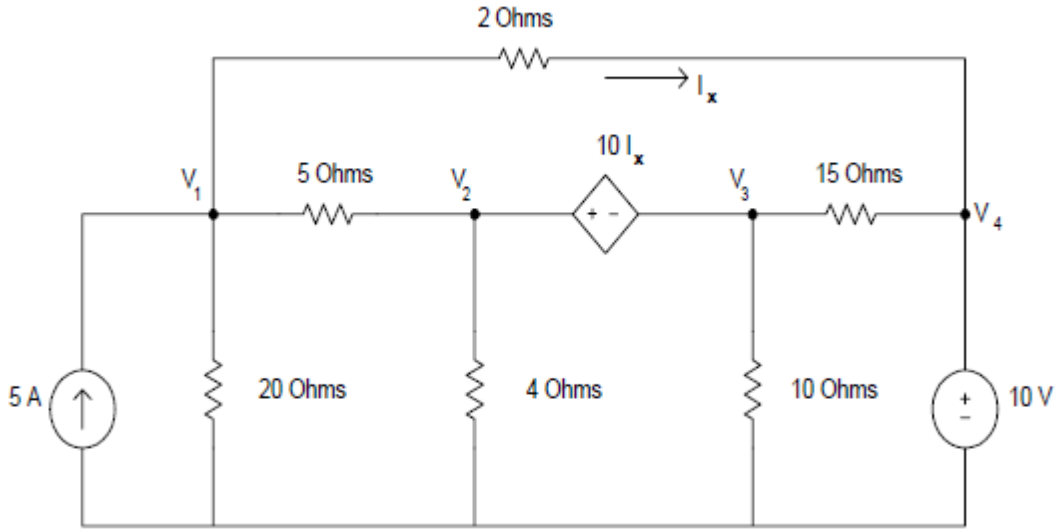
RESULT

The nodal voltages *V1, V2* and *V*3 for the circuit using KCL is verified .

SCILAB – ELECTRONICS APPLICATION PROGRAM (2)

EXP. NO: 10

i : **OBJECTIVE:** Write a scilab program to find the nodal voltages *V1, V2* and *V*3 for the circuit shown below.



ii: **ALGORITHM:**

STEP 1: using Kirchhoff's current law the algebraic sum of all the currents at any node in the circuit equals zero.

STEP2: In nodal analysis, if there are n nodes in a circuit select a reference node, the other nodes can be numbered from *V1* through *Vn-1*.

STEP3 : With one node selected as the reference node, there will be n-1 independent equations. If we assume that the admittance between nodes i and j is given as *Yij* , we can write the nodal equations:

STEP4: Y11 V1 + Y12 V2 + … + Y1m Vm = ∑□I1
Y21 V1 + Y22 V2 + … + Y2m Vm = ∑□I2
Ym1 V1 + Ym2 V2 + … + Ymm Vm = ∑□Im (4.1)
Where $m = n - 1$

V1, *V2* and *Vm* are voltages from nodes 1, 2 and so on ..., *n* with respect to the reference node.

STEP5: ∑*Ix* is the algebraic sum of current sources at node x.

Equation (4.1) can be expressed in matrix form as

□*Y*□□*V* □□□□□□*I* □□ (4.2)

The solution of the above equation is

□*V* □□□□□□*Y*□□□□□*I* □where (4.3)

□*Y*□□□ is an inverse of □*Y*□□.

In MATLAB, we can compute [V] by using the command

V □□*inv*(*Y*) * *I* (4.4)

where

inv(*Y*) is the inverse of matrix *Y*

The matrix left and right divisions can also be used to obtain the nodal voltages. The following Scilab commands can be used to find the matrix [V]

V=I\Y (4.5)

or

V □□*Y* \ *I* (4.6)

iii. SOURCE CODE :

PROGRAM

```
Y = [0.75 -0.2 0 -0.5;-5 1 -1 5;-0.2 0.45 0.166666667 -0.0666666667;0 0 0 1];
I = [5 ;0 ;0 ;10];
V = inv(Y)*I
fid= mopen('volatage.txt','w');
mfprintf (fid,'Nodal voltages V1,V2,V3 and V4 are\n');
mclose(fid);
```

iv: SAMPLE INPUTS & OUTPUTS:

INPUT: Y, I (the inputs are there in the program)

OUTPUT: (to see the output just type V and enter in console screen)

OUTPUT

Nodal voltages V1,V2,V3,V4 are

V =

18.110749
17.915309
- 22.638436
10.

RESULT :

The nodal voltages *V1, V2, V*3 and *V4* for the circuit using KCL is verified .