MMTE-003
PATTERN RECOGNITION AND IMAGE PROCESSING

Indira Gandhi National Open University
School of Sciences

Block

5

SCILAB MANUAL

Course Design Committee

Dr. B.D. Acharya
Dept. of Science & Technology, New Delhi

Prof. Adimurthi
School of Mathematics
TIFR, Bangalore

Prof. Archana Aggarwal
CESP, School of Social Sciences
JNU, New Delhi

Prof. R.B. Bapat
Indian Statistical Institute
New Delhi

Prof. M.C. Bhandari
Dept. of Mathematics
IIT, Kanpur

Prof. R. Bhatia
Indian Statistical Institute, New Delhi

Prof. A.D. Dharmadhikari
Dept. of Statistics
University of Pune

Prof. O.P. Gupta
Dept. of Financial Studies
University of Delhi

Prof. S.D. Joshi
Dept. of Electrical Engineering
IIT, Delhi

Dr. R.K. Khanna
Scientific Analysis Group, DRDO, Delhi

Prof. Susheel Kumar
Dept. of Management Studies
IIT, Delhi

Prof. Veni Madhavan
Scientific Analysis Group
DRDO, Delhi

Prof. J.C. Misra
Dept. of Mathematics
IIT, Kharagpur

Prof. C. Musili
Dept. of Mathematics and Statistics
University of Hyderabad

Prof. Sankar Pal
ISI, Kolkata

Prof. A.P. Singh
PG Dept. of Mathematics
University of Jammu

Faculty Members
School of Sciences, IGNOU
Prof. Poornima Mital
Dr. Atul Razdan
Prof. Parvin Sinclair
Prof. Sujatha Varma
Dr. S. Venkataraman

Course Design Committee

Dr.A.K Sinha
DRDO, Delhi

Prof. Vipula Singh
RNSIT, Bangalore

Prof. Muthukumar Subramanyaom
NIT, Puducherry

Faculty Members
School of Sciences, IGNOU
Prof. Sujatha Varma
Dr. S. Venkataraman
Dr. Deepika

Block Preparation Team

Prof. K.K. Biswas (Retd.) (Editor)
IIT, Delhi

Dr. Deepika
School of Sciences, IGNOU

Course Coordinator: Dr. Deepika

Acknowledgements: Mr. Santosh Kumar Pal for preparing the CRC.

Disclaimer – Any material adapted from web-based resources in this course are being used for educational purposes only and not for commercial purpose.

March, 2021

ISBN-

Further information on the Indira Gandhi National Open University courses may be obtained from the University's Office at Maidan Garhi, New Delhi-110 068.

Printed and published on behalf of the Indira Gandhi National Open University, New Delhi by the Director, School of Sciences.

BLOCK 5 LABORATORY MANUAL

Programming cannot be learnt by just reading books. It can be learnt only by constant practice, by creating, compiling and executing programs. Because of this, we have a substantial practical component for this course. You will recall that in the 70% marks carmarked for the term end examination, 50% is for theory and 20% is for practicals. In the 30% of the continuous assessment, 20% is for the assignment and 10% is for practical component. This lab manual will serve as a guide for the practical component. This lab manual consists of two units and ten lab sessions.

Unit 15 of this manual is an introduction to Scilab. It lists various steps to download scilab environment and discusses basic commands of scilab related to various computations, matrices, plot, etc.

In Unit 16, we discuss the important tool box of scilab known as Scilab Image and Video Processing (SIVP) toolbox. We are giving the steps to add on this tool box in your scilab environment. We also discuss the various command available in it for image processing.

This course has 10 practical sessions of 3 hours each. We are giving the sessions with list of programs you have to write in each of the 10 sessions along with instructions. In some of the sessions, input/output are also given.

UNIT 15

INTRODUCTION TO SCILAB

Structure **Page No.**

15.1 INTRODUCTION

Scilab is a programming language and has many algorithms available for numerical computations and scientific computations. Scilab is an open source application and is an alternative to MATLAB. Here is a list of features of Scilab:

- It is open source software which can be downloaded and installed without any restriction and the user does not pay for it.
- Operating systems like Windows, Linux and MAC are used to run scilab.
- The development processes are faster as Scilab directly uses the access of high level languages.

Scilab was created in 1990 by researchers of French Institute for Research in Computer Science and Automation. Initially, it was named as ψlab.

This unit provides you an introduction to Scilab. We shall also discuss some basic operations performed using Scilab such as working with matrices, functions, plotting functions etc.

And now, we will list the objectives of this unit. After going through the unit, please read this list again make sure you have achieved the objectives.

Objectives

After studying this unit you should be able to

- download and install Scilab in WINDOWS
- compute various functions
- perform operations on matrices etc.
- perform numerical computations
- plot 2D graphs

15.2 DOWNLOADING & INSTALLING SCILAB

You can easily download Scilab on different operating systems such as GNU/Linux platforms, Windows 2007/XP/VISTA platforms, MacOSX 10.5 Intel platforms. Scilab is distributed freely and is available on open source. Scilab is currently being used in educational and industrial environments around the world, Scilab includes hundreds of mathematical functions.

The home page of Scilab is www.scilab.org. Here is the stepwise procedure for downloading and installing Scilab in WINDOWS.
Step – I: (Downloading the Scilab): Search for download Scilab on any of the search engine e.g. google, yahoo, etc. You will get the window in google, given in Fig. 1:

Fig 1: Search Engine results for Scilab

Now select www.scilab.org/download in the window given above and double click it. You will get the window as shown in Fig. 2:

Fig 2: Downloading Scilab

You are seeing an arrow on which **Download Scilab** is written. Double click the arrow. It will start downloading scilab. When the download is complete you will get the window shown in Fig. 3:

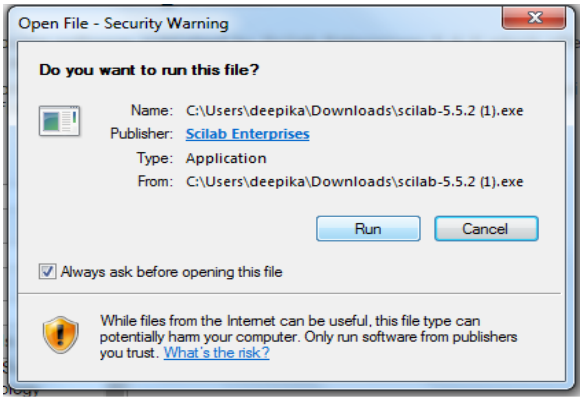


Fig 3: Run exe file

Step – II: (Installing Scilab): Select Run. It will open select start menu folder where the application to be saved as given in the window of Fig. 4:

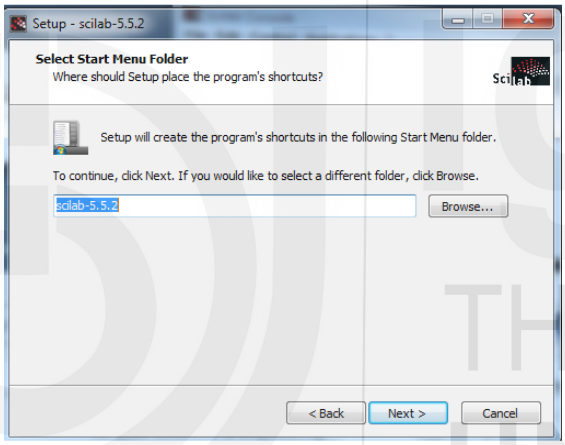


Fig 4: Selecting folder

Click on the button marked **Next** in the window for select start menu folder in Fig. 4. It will open scilab setup (Fig. 5):

Fig 5: Setup wizard

Now, you can see the setup wizard in Fig. 5. Click again on **Next** button. This will bring up the license agreement window shown in Fig. 6:

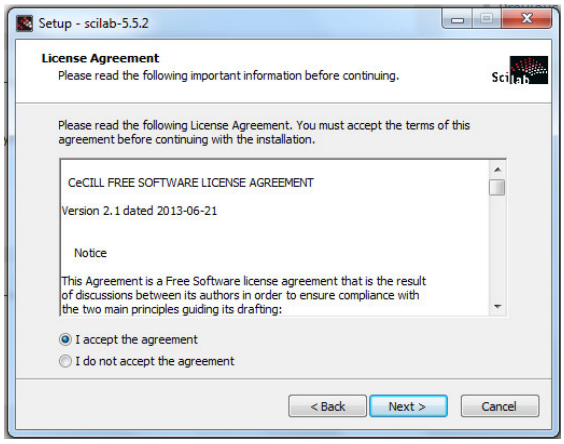


Fig 6: License agreement

Accept the terms of the license agreement. Select and press Next after selecting 'I accept the agreement', and get the window as given in Fig. 7:

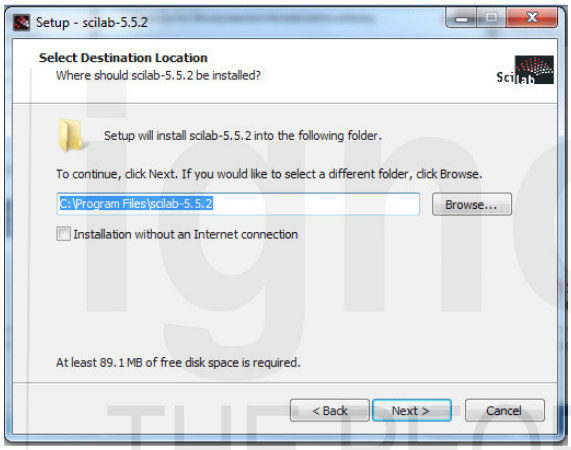


Fig 7: Selecting Destination Location

Select designation location gives you an opportunity to select the desired location for saving the application. You may enter the destination location and press next. It will take you to window shown in Fig. 8:

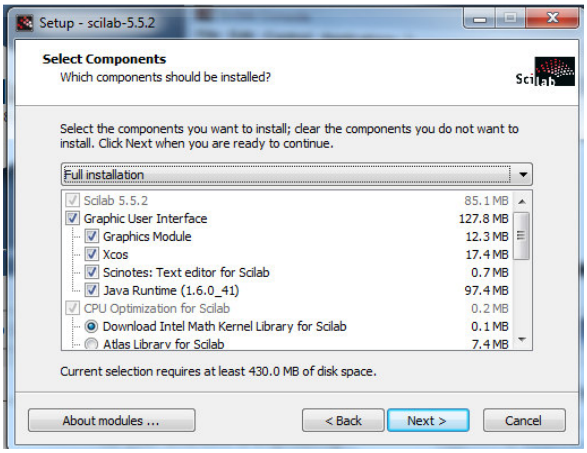


Fig 8: Selecting Components

Here, you see a list of components, from which you can choose required components or can continue with the default ones. Again press Next to get the window shown in Fig. 9:

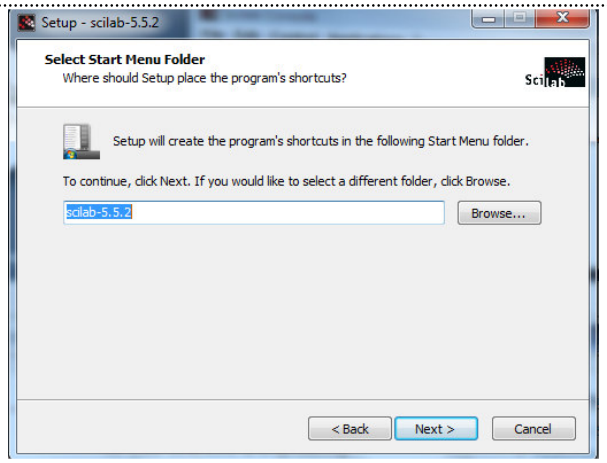


Fig 9: Selecting start menu folder

Once again press Next, you will get a window, which provides you options to select additional task, if any. You can always select for the default ones as shown in Fig. 10:

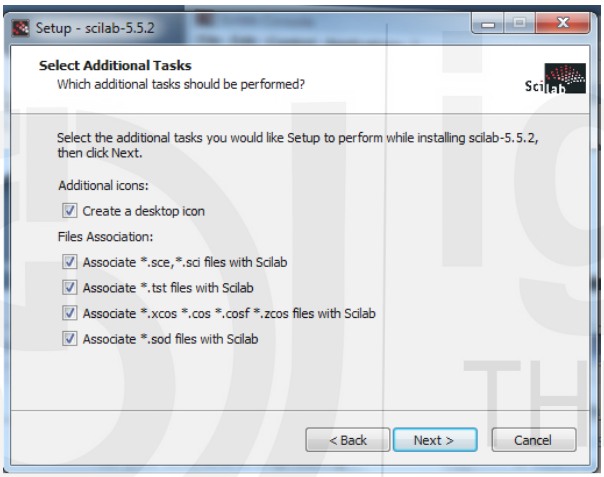


Fig 10: Selecting Additional Tools

After selecting additional tools for additional tasks, select and press next. You will come across another window ready to install as given in Fig. 11:

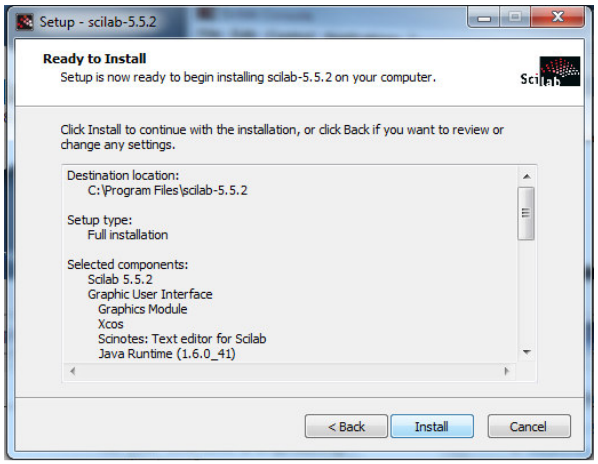


Fig 11: Install Command

Again press **Next**. This will open installing window as shown in Fig. 12. Please wait till the installation is complete.

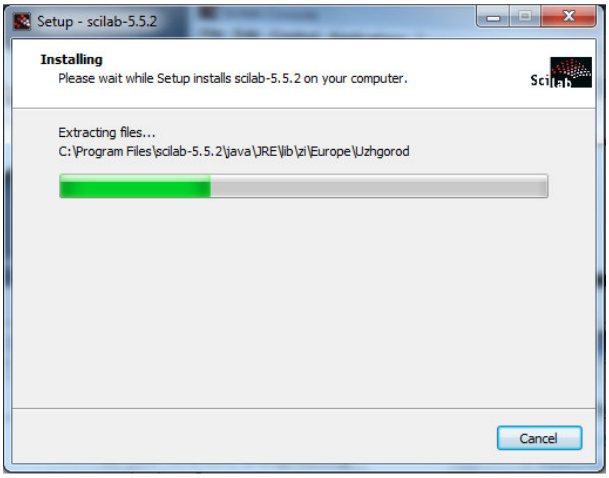


Fig 12: Installing Scilab

You will now see the completing setup wizard window given in Fig. 13 with message to click finish to exit setup.

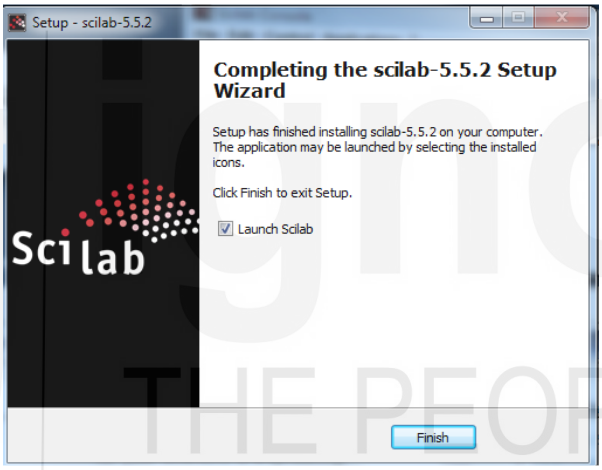


Fig 13: Completion of Installation

Press finish now. Scilab has been downloaded and the installation procedure is complete.

You will see the Scilab shortcut icon on the desktop as shown in Fig. 14.

Fig. 14 Shortcut icon of the Scilab

Now, you have successfully installed Scilab but later we shall be adding a toolbox called SIVP, which will be discussed in unit 16.

Next we shall see how to start Scilab.

15.3 HOW TO START SCILAB

There are several ways in which the Scilab can be used. For example, Scilab can be started using the console window of Scilab in an

interactive mode. You can also start Scilab by using the `exec` function against a file. Scilab can also be started by using batch processing. You can start Scilab by any of the listed method. In this unit, we are going to starting Scilab by using the console window.

In console window, commands are typed, results are analysed and process is continued till the final form of the output in terms of result is computed.
To start Scilab, double click on the shortcut of the Scilab icon and launch the Scilab window which is called console window, as shown in Fig. 15. You will enter Scilab commands at the prompt (-->).

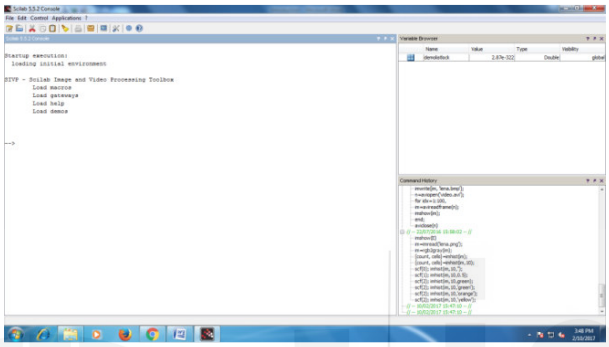


Fig 15: Starting with Scilab

So far we have discussed installation of Scilab and starting Scilab. Let us now discuss the commands to be used in Scilab. Scilab can be used as a simple calculator to perform numerical computations. It also has the ability to define variables and store values in them so that they can be used later.

Let us demonstrate some examples.

Example 1: Type `x = 1` followed by Enter key, followed by `y = 2` followed by Enter key, followed by `x*y`. On the Scilab window, the commands would appear as follows. You can also see that the result 2, is displayed by Scilab.

```
--> x = 1 [Create and set a real variable x to 1]
 x =
    1
--> y = 2 [Create and set a real variable y to 2]
 x =
    1
--> z = x*y [Performing multiplication on variables]
 z =
    2
```

Example 2:

```
--> 2 + 3
 ans =
    5.
--> 2/3
 ans =
    .666667
--> 2^3
```

```
ans =
    8.
```

It may be noted here that Scilab creates a variables named "ans" to store results of calculations whenever we do not supply a variable for this.

Here is a list of some elementary mathematical operators:

Table 1: Some Mathematical Operators

Operations	Symbol used for
Addition	+
Subtraction	–
Multiplication	*
Division	/
Power Transpose	^
Conjugate	'

We have so far been discussing examples where Scilab is used to compute numbers. Scilab can manage different kind of data, digital images, etc. In the unit 16, we shall discuss how to analyse an image. As you know, a digital image is an array of numbers represented in a matrix. In fact, every digital object, text, images, sound, video or software ultimately boil down to numbers, to be specific, zeros and ones.

Now we discuss some very basic features of the Scilab environment which are helpful in working with Scilab effectively and efficiently.

- In Scilab, a command line is entered by typing after the prompt or by copying it from other windows to the prompt in the Scilab window.
- To delete characters of a command, we use backspace.
- We can also use left and right arrow keys to move back and forth along the command and insert (by typing) or delete (by using the delete key) specific characters.
- The Up and Down arrow keys help us move back and forth among previous commands of the current session, enabling command history management.
- We can enter more than one command on the same line by separating the commands by semicolons (;).

  For example, — → `a = 5; b = 2.`

  In addition, the following key shortcuts are also supported by Scilab:

- Ctrl-a move to beginning of line
- Ctrl-b move backward one character
- Ctrl-c interrupts Scilab and pause after carriage return. Clicking on the Control/stop button enters a Ctrl-c.
- Ctrl-d delete one character (at cursor)

- Ctrl-e move to end of line
- Ctrl-f move forward one character
- Ctrl-h delete previous character
- Ctrl-k delete to the end of the line
- Ctrl-n recall next line
- Ctrl-p recall previous line
- Ctrl-u cancel current line
- Ctrl-y yank the text previously deleted

Now in the following section, we shall discuss Scilab menu bar.

15.4 SCILAB MENU BAR

The scilab window always displays a menu bar with the following menus: file, Edit, control, applications, ? and toolbox. Let us see the features available in these.

File Menu

The file menu has the following menu options which are self-explanatory. These are identical to File menus in most other generic application software as shown in Fig. 16:

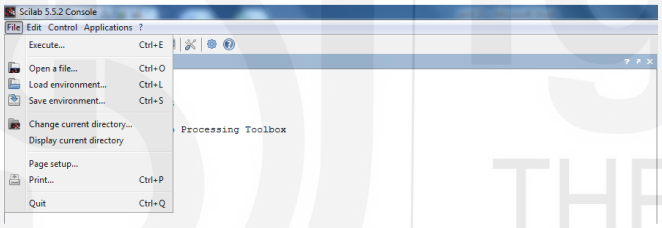


Fig 16: The File menu

Edit Menu

The common edit options like Cut, Coy, Paste, and Empty/Clipboard & Select all are available in this menu. The list of edit menu is given in Fig.17.

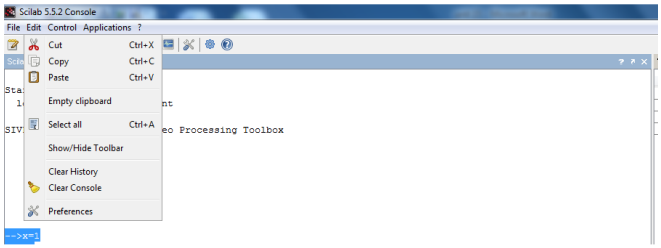


Fig 17: The Edit menu

Control Menu

The control menu includes **Interrupt**, which interrupts execution of Scilab and enters in pause mode; **Resume**, continues execution after a pause entered as a command in function or generated by the Stop button or Control C and Abort, which aborts execution after one (or several) pause, and returns to top-level prompt. All these are shown in Fig. 18:

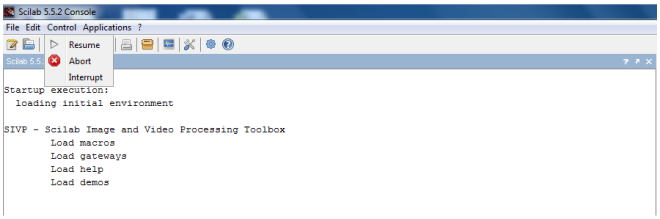


Fig 18: The Control menu

Applications Menu

We can use Editor, Scicos, Matlab to Scilab translator or Variable Editor in the Applications menu as shown in Fig. 19:

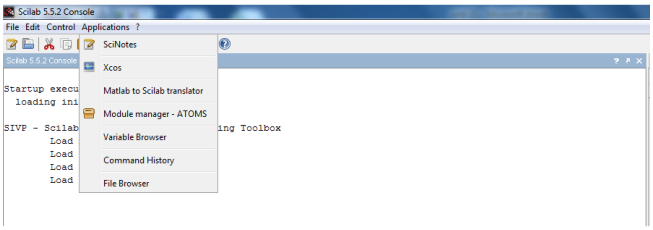


Fig 19: The Applications menu

? Menu

This menu presents interesting option Scilab Demonstrations. Selecting this will displays a large number of demos in various areas which are very useful for the beginners. The list of demos is shown in Fig. 20. The required demo can be selected and viewed.

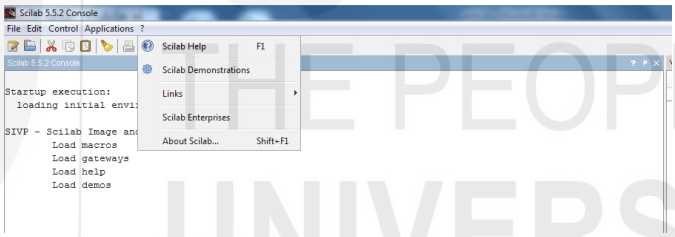


Fig 20: The ? menu

Toolboxes

Tool boxes are a collection of specialized commands/functions catering to specific area. To use any tool box, it has to be downloaded and installed. See the following screenshot of Scilab with SIVP installed. The tool box can be launched by clicking on the menu option in the Toolbox Menu.

In the following sections, we shall discuss various other functions and their operations done in Scilab.

15.5 FUNCTION USING SCILAB

Before we start discussion on various functions, let us discuss about name of the variable.

Variable name can be as long as you want. Scilab is case sensitive, therefore upper and lower case letters are considered differently. For example N and n are two different variables.

If any comment is to be added which is to be ignored while processing, it is started with "//".

For example

```
--> // this is example.
```

Scilab uses the following inbuilt mathematical functions:

sin	cos	tan	cot	cosec	sec
sinh	cosh	tanh	coth	cosech	sech
exp	log	log 10	log 2	log m	max
min	modulo	sign	sqrt		

Let us now discuss few examples on these:

For example:

Example 3:

```
— → x = sin (2)
 x =
    0.9092974
— → y = cos (2)
 y =
    - 0.4161468
— → x^2 + y^2
 ans =
    1.
```

Complex numbers are stored in pairs of floating point numbers. The pre-defined variable % i represents the imaginary number i for which $i^2 = -1$.

For example,

```
— → x = 2 + % i
 x =
   1 + i
```

Complex numbers are shown by following functions:

Real part	real
Imaginary part	imag
Multiplicating by i	imult

Strings can be stored in variables. In the following example you will understand it.

For example,

Example 4:

```
— → x = "string"
 x =
   string
```

Example 5: (Concatenation)

```
— → x = “tool”
      x =
         tool
— → y = “box”
      x =
           box
— → x + y
      ans =
            toolbox
```

Again, you may note that, ans variable has been used whenever we make a computation and do not store the result into an output variable. To define a new function, we use the function and endfunction which are Scilab keywords. Let us take an example by defining a new function named myfunction, which takes the input argument x, multiplies it by 2, and returns the value in the output argument y.

```
function  y  =  myfunction  ( x )
  y  = 2  *  x
endfunction
```

The statement function y = myfunction (x) is the header of the function while the body of the function is made of the statement y = 2 * x. The body of a function may contain one, two or more statements.

There are at least three possibilities to define the previous function in Scilab.

- The first solution is to type the script directly into the console in an interactive mode. Notice that, once the “function y = myfunction (x)” statement has been written and the **enter** key is typed in, Scilab creates a new line in the consle, waiting for the body of the function. When the “endfunction” statement is typed in the console, Scilab returns back to its normal edition mode.
- Another solution is available when the source code of the function is provided in a file. This is the most common case, since functions are generally quite long and complicated. We can simply copy and paste the function definition into the console. When the function definition is short (typically, a dozen lines of source code), this way is very convenient. With the editor, this is very easy, thanks to the *Load into Scilab* feature.
- We can also use the exec function. Let us consider a Windows system where the previous function is written in the file “C: \myscripts\examples-functions. sce". The following example shows the use of exec to load the previous function.

```
-->exec (“C: \myscripts\examples-functions. sce")
-->function y = myfunction ( x )
-->  y  =  2 * x
-->endfunction
```

Now, let us discuss the operations on matrices.

15.6 MATRICES USING SCILAB

Matrices play a central role in image processing. The basic data type is the matrix which requires the number of rows, the number of columns, the type of data.The data type can be real, integer, boolean, string or polynomial. When two matrices have the same number of rows and columns, we say that the two matrices have the same shape.
Matrix operations that are built-in into Scilab are addition, subtraction, multiplication, transpose, inversion, determinant, logarithmic, exponential and many others.

There is a simple and efficient syntax to create a matrix with given values. The following is the list of symbols used to define a matrix.
* square brackets "[“and”]" mark the beginning and the end of the matrix,

* commas " ," separate the values in different columns,
* semicolons " ; " separate the values of different rows.

The following syntax can be used to define a matrix, where blank spaces are optional (but make the line easier to read) and " … " denotes intermediate values:

```
A = [a11, a12, ..., a1n; a21, a22, ..., a2n; ...; an1, an2, ..., ann].
```

For example, we create a 2×3 matrix of real values.

```
-->A = [1, 2, 3; 4, 5, 6; 7, 8, 9]
A =
   1   2   3
   4   5   6
   7   8   9
```

Several Scilab commands allow to create matrices from a given size, i.e. from a given number of rows and columns. The most commonly used are eye, zeros and ones. The identity matrix is represented by `eye`.

These commands take two input arguments, the number of rows and the number of columns.

The matrix with each element 1 is represented by the function `ones`.

For example,

```
-->A = ones (2, 4)
 A =
    1.    1.    1.    1.
    1.    1.    1.    1.
```

The identify matrix is represented by eye.

For example,

```
-->A = eye (3, 3)
 A =
  1   0   0
  0   1   0
  0   0   1
```

A diagonal matrix can be created as follows:

```
-->d = eye (3,3) * 5,
-->d =
      5   0   0
      0   5   0
      0   0   5
```

The zero matrix: A zero matrix can be created by using zeros.

```
-->A = zeros (3,4)
 A =
      0   0   0   0
      0   0   0   0
      0   0   0   0
      0   0   0   0
```

Similarly ones function creates the matrix with all elements being 1. For example,

```
-->A = ones (2, 3)
 A  =
      1.    1.    1.
      1.    1.    1.
```

The `size` function returns the number of rows or the number of columns of a matrix.

```
nr  =  size  (A  , sel)
```

where, `sel` has to be given values:

- `sel =1` or `sel ="r"`, returns the number of rows,
- `sel =2` or `sel ="c"`, returns the number of columns.
- `sel ="*"`, returns the total number of elements, that is, the number of columns times the number of rows.

We use the size function in order to compute the total number of elements of a matrix.

For example,

```
--> size (A, "*")
ans  =
      6.
```

Transpose of a matrix is shown by ' .

For example,

```
--> a = [ 1, 2, 3; 4, 5, 6]
--> b = a' ;
--> b =
      1   4
      2   5
      3   6
```

Operations in matrices are done in the following way:

- `a + b` adds a to b provided a and b are of same size.
- `a - b` gives subtraction of b from a, if both a and b are of same size.
- `a * b` give product of two matrices a and b, if they are computable to multiply.
- `inv (a)` inverts matrix a, if a is positive definite. A warning is displayed is not.
- `det (a)` gives determinant of matrix `a`, provided a is a square matrix.
- `log (a)` returns a matrix, where each element is log of the corresponding element of matrix `a`.
- `a. * b` gives element by element multiplication.
- `a^2` gives `a * a`, if a is square matrix.
- `a. ^2` gives element by element square.

There are several methods to access the elements of a matrix A:

- the whole matrix, with the `A` syntax
- element by element with the `A(i , j)` syntax,
- a range of index values with the colon ":" operator.

Now let us demonstrate by taking a few examples:

Example 6:

```
-->A = ones (2, 3)
      A  =
          1.    1.    1.
          1.    1.    1.
-->B = 2   *  ones (2, 3)
      B  =
          2.    2.    2.
          2.    2.    2.
-->A+B
```

```
ans  =
    3.    3.    3.
    3.    3.    3.
```

Example 7:

```
-->A = ones (2, 3)
 A  =
    1.    1.    1.
    1.    1.    1.
-->A (1,1)
 ans  =
    1.
-->A (12,1)
 !-error 21
Invalid index
-->A (0,1)
 !-error 21
Invalid index.
```

Example 8:

```
-->A = ones (3, 3)
 A  =
    1.    1.    1.
    1.    1.    1.
    1.    1.    1.
-->B = A  +  3*eye()
 B  =
    4.    1.    1.
    1.    4.    1.
    1.    1.    4.
```

Example 9:

```
-->A = ones (2, 2)
 A  =
    1.    1.
    1.    1.
-->B = eye (A)
 B  =
    1.    0.
    0.    1.
```

Example 10:

```
-->A = [1  2
-->3   4]
 A   =
    1.          2.
    3.          4.
-->B = [5  6
```

```
-->7   8]
      B    =
           5.          6.
           7.          8.
-->A + B
       ans  =
           6.      8.
           10.     12.
```

Example 11:

```
-->A = [1   2
-->3    4]
      A    =
           1.     2.
           3.     4.
-->A + 1
      ans   =
           2.     3.
           4.     5.
```

Example 12:

```
-->A = [1   2
-->3    4]
      A    =
           1.     2.
           3.     4.
-->A + 1
      ans   =
           2.     3.
           4.     5.
```

The addition is possible only if the two matrices are conformable to addition. In the following session, we try to add a 2×3 matrix with a 2×2 matrix and check that this is not possible.

Example 13:

```
-->A = [1   2
-->3    4]
     A    =
          1.     2.
          3.     4.
-->B = [1   2   3
-->4   5   6]
     B    =
       1.      2.      3.
       4.      5.      6.
-->A + B
```

```
        ! -- error 8
Inconsistent addition.
```

In the following example, an unsymetric matrix of doubles containing complex values is used, so that the difference between the two operators is obvious.

Example 14:

```
-->A = [1  2; 3  4]  + %i  * [5  6; 7 8]
 A  =
   1. + 5 .i      2. + 6 .i
   3. + 7 .i      4. + 8. I
-->A '
 ans  =
   1. - 5. i      3. - 7. i
   2. - 6. i      4. - 8. i
-->A .'
 ans  =
   1. + 5. i      3. + 7. i
   2. + 6. i      4. + 8. i
```

In the following example, we define an unsymetric matrix of doubles containing real values and see that the results of the " ' " and " .' " are the same in this particular case.

Example 15:

```
-->B = [1  2; 3  4]
 B  =

   1.    2.
   3.    4.
-->B  '
 ans  =
   1.    2.
   3.    4.
-->B .  '
 ans  =
   1.    3.
   2.    4.
```

In the following example, we multiply the column vector u by the row vector v and store the result in the variable A.

Example 16:

```
-->u = [1
-->2
-->3]
  u  =
      1.
      2.
      3.
```

```
-->v = [4   5   6]
   v =
      4.   5.   6.
-->u*v
   ans  =
      4.    5.    6.
      8.    10.   12.
      12.   15.   18.
```

In the following Scilab example, we create a matrix A and test if elements are more than 3 or less than 3.

Example 17:

```
-->A = [1   2   7
-->6   9   8]
   A  =
      1.   2.   7.
      6.   9.   8.
-->A>3
 ans  =
      F   F   T
      T   T   T
-->B = [4   5   6
-->7   8   9]
   B  =
      4.   5.   6.
      7.   8.   9.
-->A>B
 ans  =
      F   F   T
      F   T   F
-->or (A>B, "r")
 ans  =
      F   T   T
```

In Scilab, it is possible to generate a vector given a range of numbers.

- `a = [ 1: 6]` creates a vector with 6 elements as follows:

  `[ 1, 2, 3, 4, 5, 6]`
- `b = [ 0:0. 5:5]` creates a vector with 11 elements as follows:

  `[0, 0.5, 1.0, 1.5, …..4.5, 5.0].`
- r and `(4, 5) * 100` generates elements are generated as random numbers.

In the following section, we shall discuss looping and branching.

15.7 LOOPING AND BRANCHING

In this section, we describe how to write conditional statements.

The `if` statement

The `if` statement allows to perform a statement if a condition is satisfied. In the following script, we display the string "Hello!" if the condition `%t`, which is always true, is satisfied.

```
If ( %t ) then
disp ( "Hello  !" )
end
```

The previous script produces:

```
"Hello!"
```

If the condition is not satisfied, the else statement allows to perform an alternative statement, as in the following script.

```
If ( %t ) then
disp ( "Hello  !" )
      else
            disp( "Goodbye !" )
        end
```

The previous script produces:

```
Goodbye !
```

In order to get a boolean, any comparison operator can be used, e.g. "==", ">", etc... or any function which returns a boolean.

In the following example, we use the "==" operator to display the message "Hello !".

For example,

```
      i = 2
      if ( i ==2 ) then
   disp ("Hello !")
else
   disp ( "Goodbye  !" )
end
```

The `Select` Statement

In the following script, we want to display a string which corresponds to the given integer `i`.

For example,

```
i = 2
select  i
case 1
   disp ( “One” )
case 2
   disp ( “Two” )
case 3
   disp ( “Three” )
else
   disp ( “Other” )
end
```

The `for` statement

The `for` statement allows to perform loops, i.e. allows to perform a given action several times.

For example: In the following Scilab script, we display the value of `i`, from 1 to 5.

```
for i = 1 : 5
   disp (i)
end
```

The `while` statement

The `while` statement allows to perform a loop as long as a boolean expression is true.

In the following script, we compute the sum of the numbers `i` from 1 to 10 with a `while` statement.

For example:

```
s  =  0
i  =  1
while ( i<= 10 )
  s  =  s  +  i
  i  =  i  +  1
end
```

At the end of the algorithm, the values of the variables `i` and `s` are:

```
s =
   55.
i =
   11.
```

In the next section we show how plotting can be done in Scilab.

15.8 PLOTTING THROUGH SCILAB

Let us learn to plot simple graphs. We first have to generate the data to be used for the graph. Let us assume we want to draw the graph of cos (x) and sin (x) for one full cycle (2π radius). Let us first generate the values for the x-axis with the following command:

```
-->x = [0 : % pi/16:2 * % pi];
```

In the above command, % pi is a predefined constant representing the value of π. The command requires a start value, an increment and an end value, which are 0, $\pi/16$ and 2π respectively in the above example. Increment is optimal. If not given, it takes default value 1.

Now let us create the value for y-axis.

```
-->y = [ cos (x)   sin (x)];
```

For plotting these, we use

```
-->plot 2d (x,y)
```

Scilab can create x-y plots with the plot function, contour plots with the contour function, 3D plots with the surf function, histograms with the histplot function and many other types of plots.

In order to get an example of a 3D plot, we can simply type the statement surf () in the Scilab console.

```
-->surf ()
```

The various functions used to plot are given below:

plot	2D plot
surf	3D plot
contour	contour plot
pie	pie chart
histplot	histogram
bar	bar chart

Let us see simple instances such as plot (x,y) which plots the variable x against y, with default graphic settings.

```
-->x=[10, 12, 14, 16, 19]
 x  =
    10.    12.    14.    16.    19.
 -->y=[1, 2, 3, 4, 5]
 y  =
    1.    2.    3.    4.    5.
 -->plot 2d(y,x)
```

Let us see the output of the plot in the following Fig. 21.

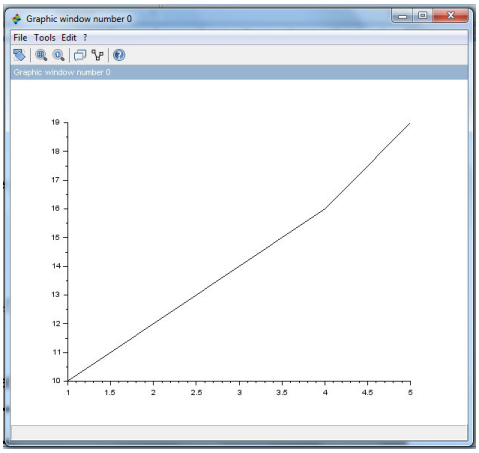


Fig. 21: Plotting (y,x)

You may note that the window title bar displays “Scilab Graphic number (0)”. Scilab can open up 20 such windows at the same time.

Let us see more examples.

Example 18: Plotting sine function.

```
-->x=[0: 0.1: 6.28];
-->plot2d(sin(x))
```

The output is shown in Fig.22.

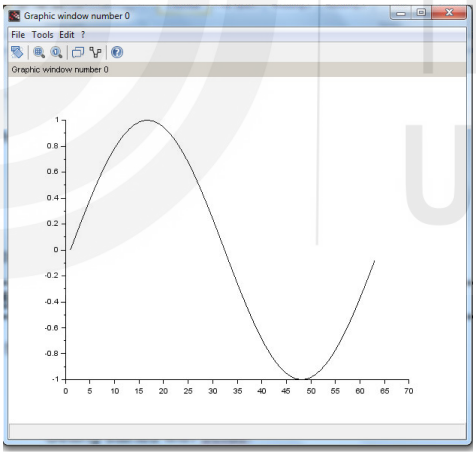


Fig. 22: Sine Function

Now let us summaries what we have studied in this unit.

15.9 SUMMARY

In this unit we have discussed following points:

1. The applications of Scilab.
2. Installation of Scilab.
3. Getting started with Scilab.

4. Basic command of Scilab.
5. Menu bar and Tool box of Scilab.
6. Functions and their operations using Scilab.
7. Matrix operations and its operations through Scilab.
8. How to write conditional statements using looping and branching.
9. Plotting of 2D figures through Scilab.

UNIT 16

INTRODUCTION TO SIVP TOOLBOX

Structure **Page No.**

16.1 INTRODUCTION

SIVP stands for Scilab Image and Video Processing toolbox. It is a toolbox provided by Scilab to implement number of Image and Video processing tasks. We can carry out image processing projects using SIVP commands. This unit gives a very brief introduction to some popular image processing related commands of SIVP toolbox.

It can easily read and process images stored in various formats such as BMP, PNG, JPEG, TIFF, PBM, PGM, PPM and FORMATS. Here, in this unit, we shall discuss image processing commands.

Objectives

After studying this unit you should be able to

- download and install SIVP toolbox in Scilab
- apply various commands for reading an image, writing an image, matrix of an image, adding and subtracting images, adding noise to images, filtering, edge detection, histogram, etc.

16.2 INSTALLING SIVP

The main purpose of SIVP is to provide a useful, efficient, and free image and video processing toolbox for academic researchers. SIVP is free software and licensed under GPL (GNU General Public License). Anyone can get the source code from SIVP homepage modify it and improve it.

You can download SIVP toolbox from the website address http://sivp.sourceforge.net:

Download SIVP for Linux:

I. Download SIVP sourcecode from sourceforge and uncompress it.
II. Run**./configure**.
III. Run **make** to compile the source code.
IV. Run **make install** as root.

Downloading SIVP for WINDOWS:

I. Double click on the download button in the homepage enables us to browse all the files at http://sourceforge.net/projects/sivp/files.
II. Double clicking on sivp 0.5.3.exe starts installation procedure.
III. After installation is complete, you will see that SIVP will appear in toolboxes menu.

After installing SIVP, when you start Scilab, you will see the window shown in Fig.1.

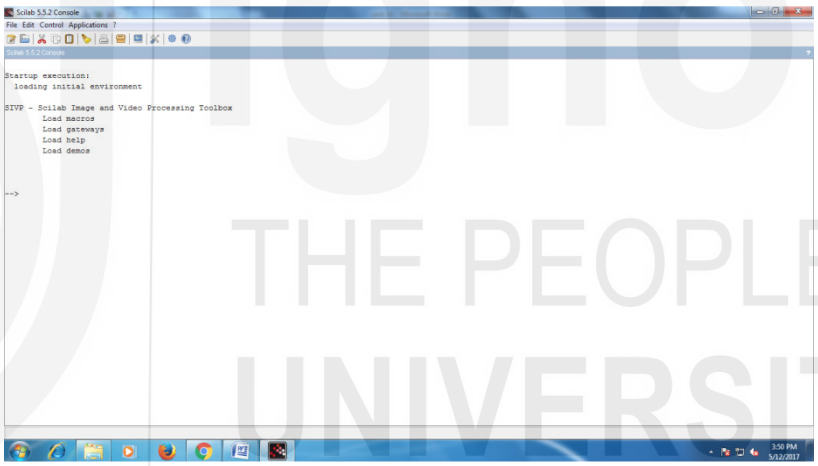


Fig. 1: SIVP homepage

The easiest way to add on SIVP toolbox is that first you click the tab of module manager – ATOMS, which is located on the left top of the scilab window as shown in Fig. 2.

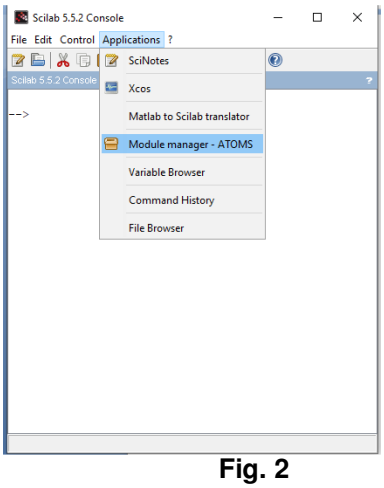


Fig. 2

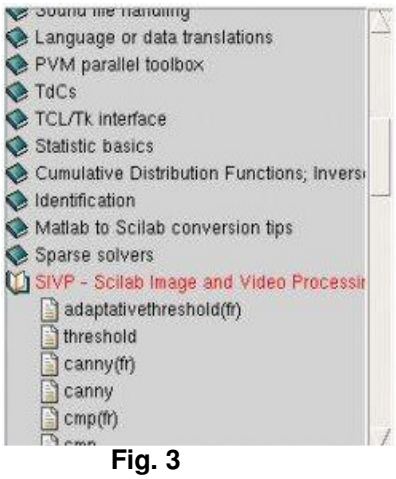


Fig. 3

Double click module manager, it will pop up another window in the left side of the scilab, which looks like as given in Fig. 3.

Click the Image Processing option from the list. Again you will get a popup window listing options. Double click Scilab Image and Video Processing toolbox and install it.

Now in the following section we shall discuss the basic commands used for image processing.

16.3 BASIC COMMANDS

Scilab image processing toolbox has some pre-stored images which researchers have been using for last 30 years or so, such as lena.png, peppers.png, people.png, etc. Either you can try your commands on any one of these images or you can download any other image of your choice. Remember to save image of your choice to the default folder or give the complete path to read the image every time. Also, ensure that the image file size is not too big in size.

Let us discuss following commands:

Reading an image

We shall start with how images are opened and viewed. We use imread() and imshow() to open and view an image respectively. Suppose the image you want to analysis is in F: and the file name is flower -1 jpg, then the image can be displayed by writing the complete path.

For Example:

```
-->fp = ('F:/flower-1.jpg');
-->imshow(fp);
```

After you press enter, you will see a pop up window as shown in Fig. 4.

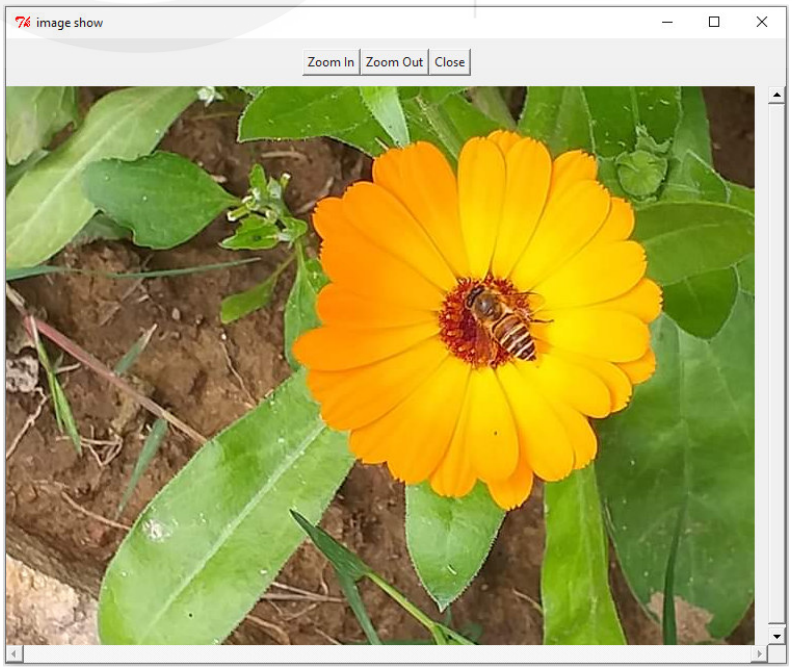


Fig. 4: Viewing flower-1

The details of this image can be seen by the command imfinfo(). These details are about filename, filesize, width, height, bitdepth and colourtype. These can be edited after copying and are shown in Fig. 5. If you type info=imfinfo ('F: /image/flower/.jpg'); in the console window, and after pressing **enter** type info, you will get the following result:

```
-1->info=imfinfo('F:/images/flower1.jpg');
 -1->info
 info  =
        info(1)
 !V         !
!           !
!Filename   !
!           !
!FileSize   !
!           !
!Width      !
!           !
!Height     !
!           !
!BitDepth   !
!           !
!ColorType  !
       info(2)
F:/images/flower1.jpg
       info(3)
      47571.
       info(4)
      720.
        info(5)
      540.
       info(6)
      8.
       info(7)
  truecolor

F:/images/flower 1.jpg
91225.
ans(4)
     512.
ans(5)
     512.
ans(6)
     8.
```

```
ans(7)
 grayscale
```

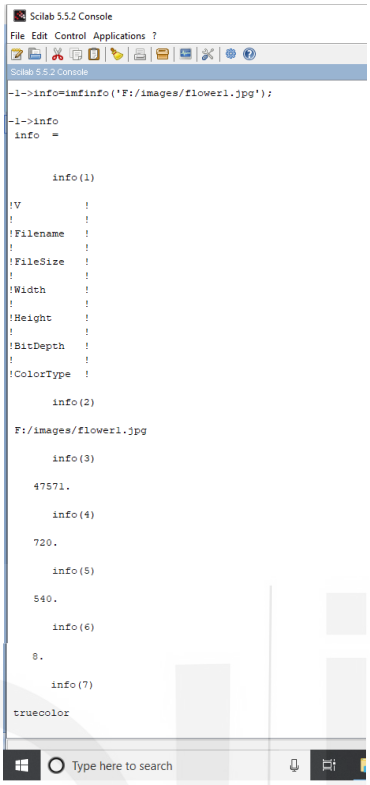


Fig. 5: Details of an image

The size of an image can be seen by the command size.
The matrix of any image can be seen by the following command:

```
-->a=( 'F:/images/flower 1.jpg' )
```

The output would scroll as it is a big matrix, this ends in the window as shown in Fig. 6.

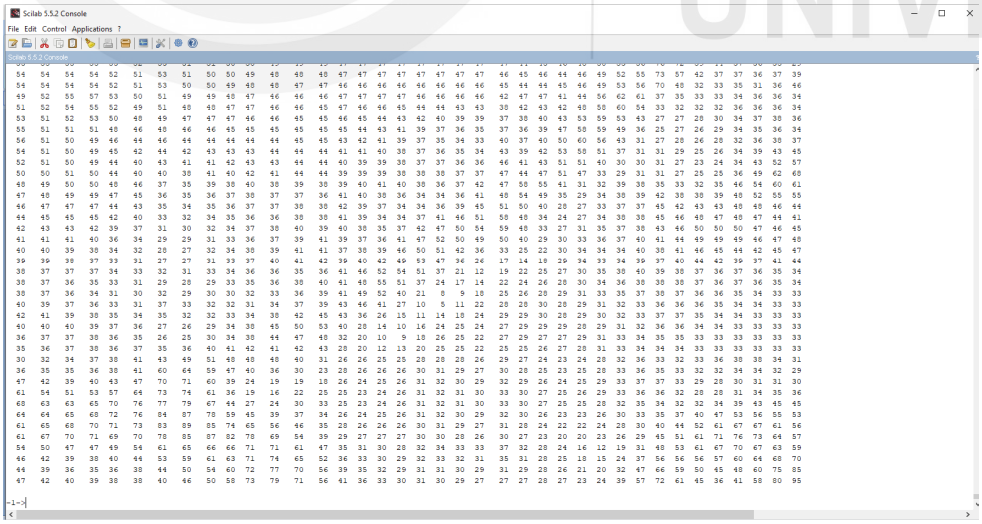


Fig. 6: Matrix of flower 1.jpg

Generating a Random Image

You can also generate an image with randoms gray values. For example, if the desired image is of the size 100×150, the following command will result in the output as shown in Fig. 7.

```
--> r=rand (100, 150);
--> imshow(r);
```

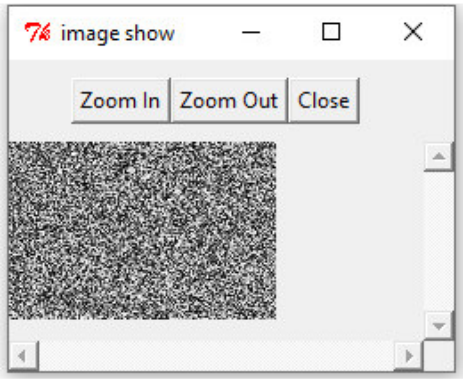


Fig. 7: Random Image of size 100 × 150.

Conversion to Gray image

Most of the commands for SIVP work on gray images, therefore it is important to convert coloured images into gray images. The command used for this purpose is rgb2gray() and the respective output is shown in Fig. 8.

```
-->a=imread( 'F:/images/flower/.jpg' );  //a is variable here.
-->b=rgb2gray(a);    //b is the variable, which stores
converted gray image.
-->imshow(b);
```

Fig. 8: Gray Image

If you want to work with this gray image, you need to store it as a new variable giving it a name. This can be done using the command `inwrite()`:

`-->imwrite(w, 'F:/images/flower 1.jpg');` //stores gray image of `flower 1.jpg` in variable w.

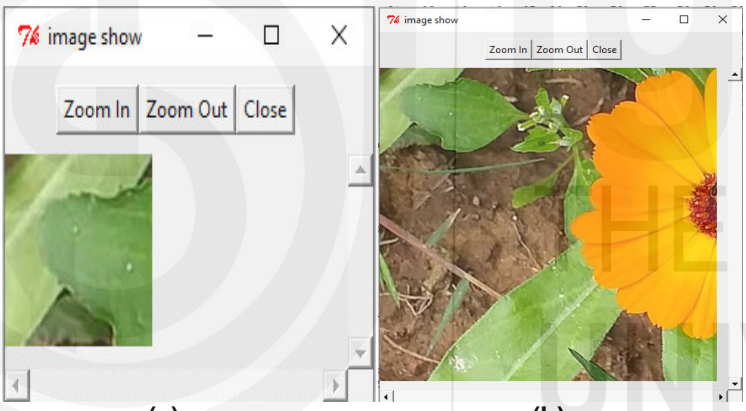
Cropping an Image

The command used for cropping an image is `imcrop(file,a)`, where file is name of the file containing the image saved in the default folder and "a" is the rectangular area to which the image is to be cropped. This area is defined with four parameters separated by `comma(,)`, and are

coordinates of the left top corner, width and height. Please note that these coordinates follow axis according to the coordinate system of computer graphics i.e. the origin is at top left corner of the image and +Y axis is in downward direction. Also, while selecting width and height, ensure that they fall inside the image area. The syntax for cropping an image is as given below:

```
-1->a=imread('F:/images/flower1.jpg');
 -1->b=rgb2gray(a);
 -1->imshow(b);
 -1->c=[50,50,100,100]; // (50,50) is the coordinate of left
top corner and 100, 100 are the width and height respectively.
 -1->d=imcrop(a,c);
 -1->imshow(d);
 -1->c1=[50,50,400,400];
 -1->d1=imcrop(a,c1);
 -1->imshow(d1);
```

The output in image show window is given in Fig.9 (a) and Fig. 9 (b) for the variables d and d_1 respectively.



(a) (b)

Fig. 9: Cropping Image

Now let us resize image.

Resizing an Image

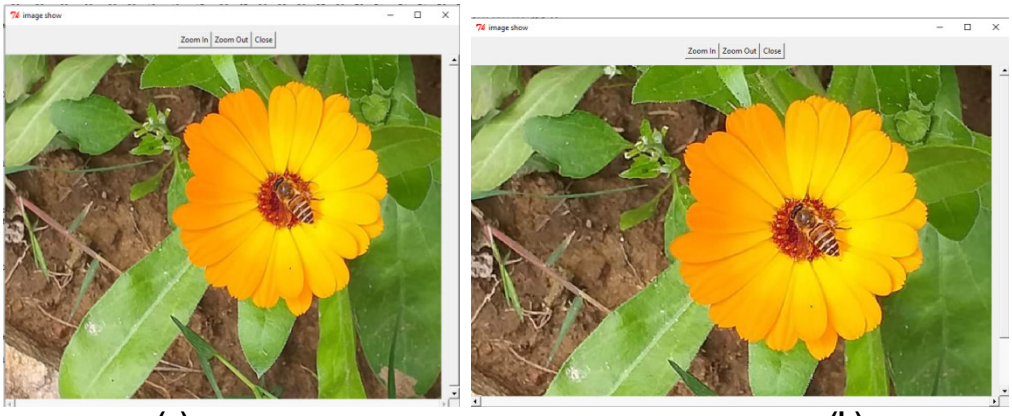
Magnification and compression of any image can be done by the command imresize ("file variable", multiple, optional subtle effect), where file variable is the name of the variable which stores image, **multiple** is the factor by which the image is to be magnified or compressed and subtle effects are optional. For example, multiple 1.5 means that the image will be increased by 50% and 0.5 means that image will be compressed by 50%.The default subtle effects are such as 'nearest', 'bilinear', 'bicubic', etc. You may like to try follow commands:

Example 1(Resizing with a scale):

```
-1->a=imread('F:/images/flower1.jpg');
-1->e=imresize(a,1.175); // image will be magnified by 75%.
```

```
-1->imshow(e);
```

The output is shown in Fig. 10 (b) and the original image is shown in Fig. 10 (a).



(a) (b)

Fig. 10

Example 2(Resizing with Rows and Columns):

```
-1->a=imread('F:/images/flower1.jpg');
-1->f=imresize(a,[100,100]); //100 and 100 are the number of
rows
 -1->imshow(f);
```

This output is shown in the corresponding Fig. 11.

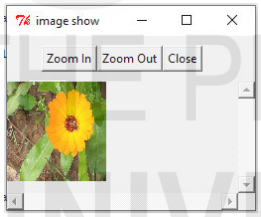


Fig.11: Image with 100 Rows and 100 Columns

Example 3(Resizing with scale and subtle effect):

```
-->a=imread('F:/images/flower1.jpg');
-->x=imresize(im,1.25,'bilinear');
-->y=imresize(im,1.25,'bicubic' );
-->imshow(a);
-->imshow(x);
-->imshow(y);
```

Operations on Images

Images can be added, subtracted and multiplied as given in the following examples.

Example 4: Consider the following commands

```
-->a=imread('F:/images/flower1.jpg');
-->imshow(a);
```

```
-->a=imresize (a,[100,100];
-->b=imresize(a,[100,100]);
-->max(b); //This provides the maximum intensity.
-->max(b)
ans  =
    255.
-->imshow(b);//This shows the image with maximum intensity.
-->c=243*ones(100,100);  // All members have maximum intensity.
-->c=b-a; //while adding or subtracting images, ensure that the
size of the images are same, here all intensities are
subtracted from the highest one.
-->imshow(c);
```

The output is like negative of image as shown in Fig. 12.

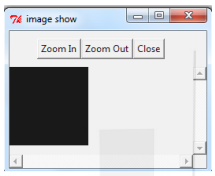


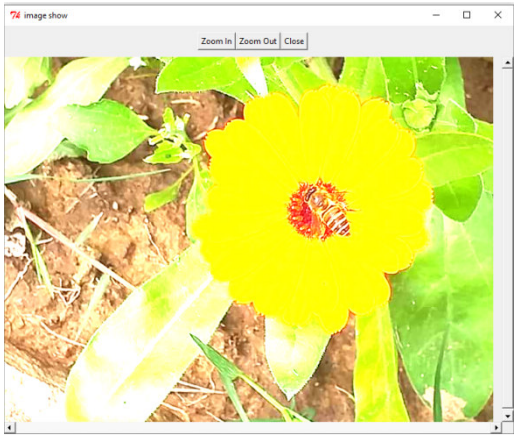
Fig. 12: Subtraction from Maximum intensity

***

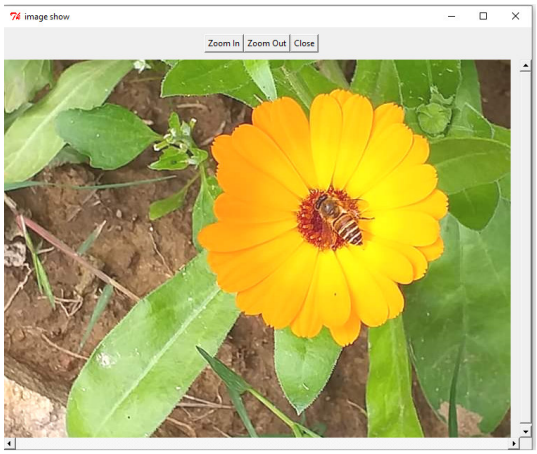
Example 5(Addition in Images):

```
-->a=imread('F:/images/flower1.jpg');
-->a1=imadd(a,a);   //Adding a and a.
-->imshow(a1);
-->a2=imadd(a,20);      //Adding 20 to each member.
-->imshow(a2);
-->a3=imadd(a,100);     //Adding 100 to each member.
-->imshow(a3);
-->a4=immultiply(a,a);   //Multiply a with a
-->imshow(a4)
```

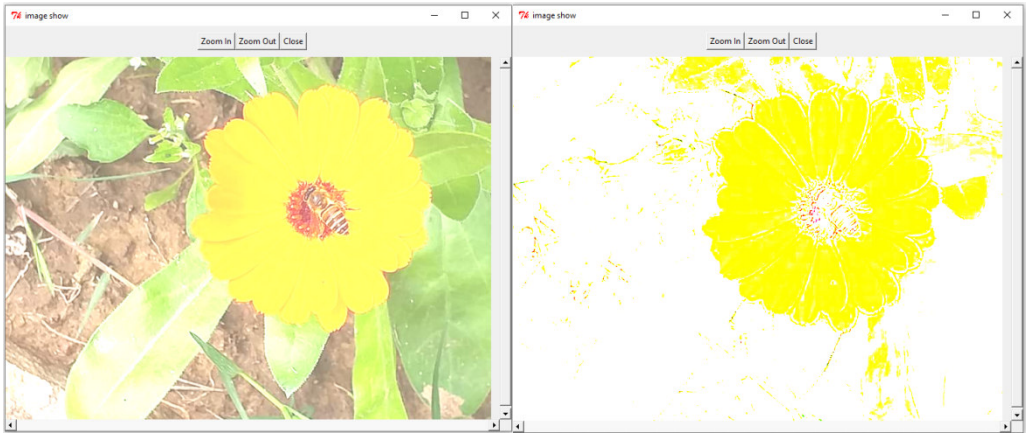
The output can be viewed in Fig. 13.



(a) Variable a1



(b) Variable a2



(c) Variable a3 (d) Variable a4

Fig. 13: Output of Example 5.

***

Example 6(Subtraction in Imges):

```
-->a=imread('F:/images/flower1.jpg');
-->ab=imread('F:/images/flower1/with spot.jpg');//save after
   inserting a spot in flower 1.jpg in paint brush and save with
   flower/with spot.jpg
-->af=imsubtract(a,as);
-->imshow(af);
```

This will simple show the spot inserted as shown in Fig.14.

Fig. 14: Subtraction

So far, we have discussed basic commands. Now in the following section, we shall discuss other commands like edge detection, adding noise to images, etc.

16.4 OTHER COMMANDS

We shall now discuss SIVP commands to detect images, filtering images, etc.

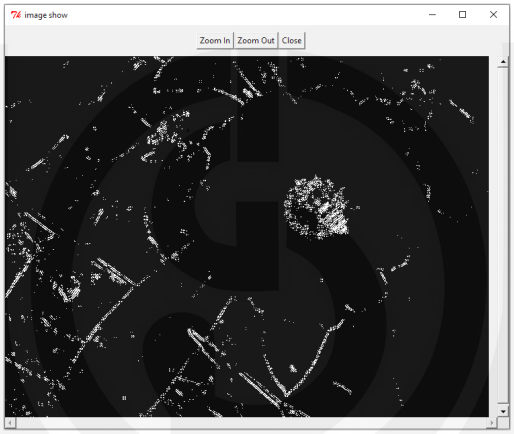
Edge Detection

Function edge('file variable',operator), where operator may be selected from the default operators available such as sobel, prewitt, log, canny, etc. We can see the following illustration to understand this well.

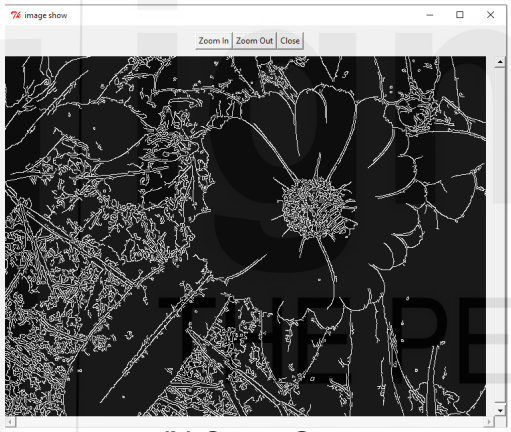
Example 7:

```
-->a=imread('F:/images/flower1/.jpg');
-->a=rgb2gray(a);
-->e1=edge(a,'sobel');
-->e2=edge(a,'canny',[0.05,0.2]);
-->e3=edge(a,'sobel',-1);
-->imshow(e1);
-->imshow(e2);
-->imshow(e3);
```

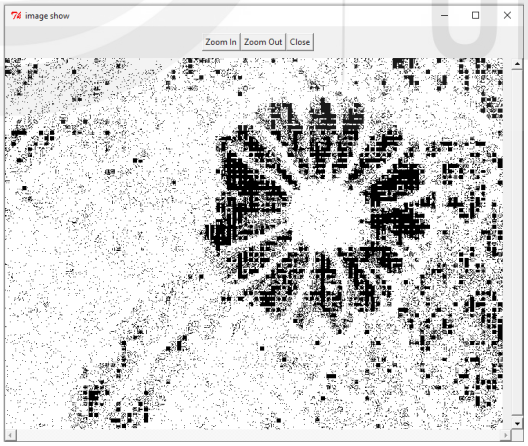
The output is given in Fig. 15, (a) for Sobel operator and (b) for Canny operator and (c) for Sobel Operator with parameter-1.



(a) Sobel Operator



(b) Canny Operator



(c) Sobel Operator with Parameter -1

Fig.15: Edge Detection

***

Now let us see how to plot histogram of images.

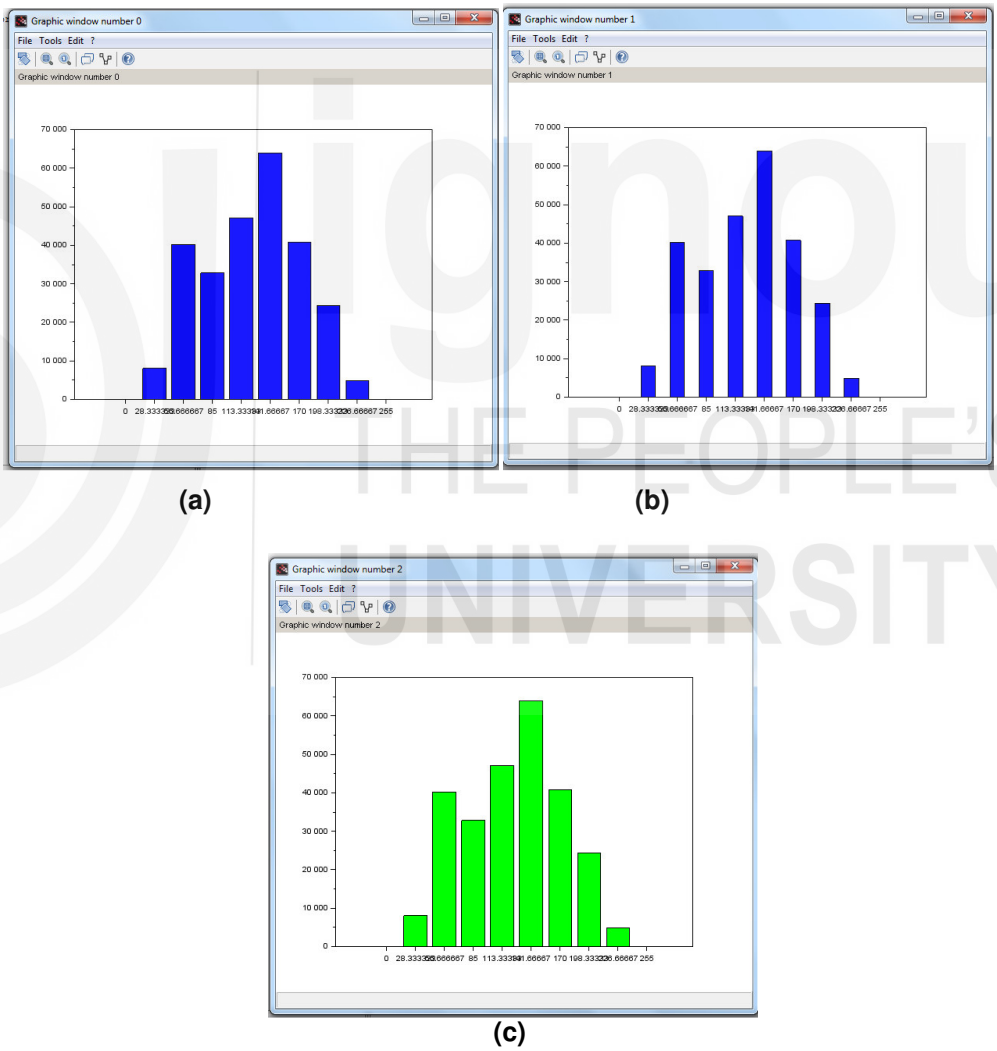
Histogram

In-built function imhist() is used to get histogram of an image as given in the following example:

Example 8:

```
-->a=imread;('F:/images/flower1/.jpg');
-->a=rgb2gray(a);  //histogram requires gray image
-->[count,cells]=imhist(a);
-->[count,cells]=imhist(a,10);    //scale is 10.
-->scf(0);   //separate graphic window will show histogram.
-->imhist(a,10,'');
-->scf(1);
-->imhist(a,10,0.5);   //width is reduces to 50%.
-->scf(2);
-->imhist(a,10,'green');   //Colour is changed to green.
```

These three outputs are shown in Fig. 16 (a), (b) and (c) respectively for default width, reduced width and changed colour.



(a) (b)

(c)

Fig. 16: Histogram

***

Adding Noise to the Image

In order to examine how image enhancement can be carried out for noisy images, the first step would be to take a clean image and selectively add noise to it. The next step would be to apply various filters to carry out image enhancement. Scilab provides functions for a

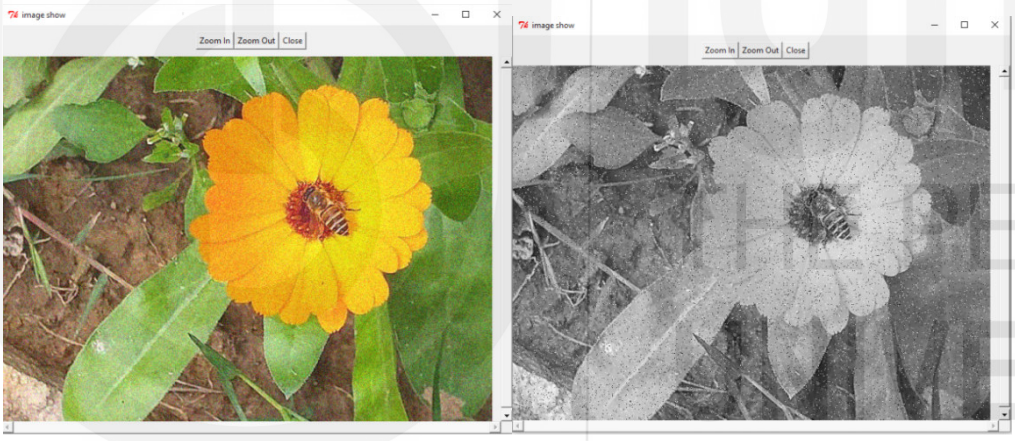
variety of noise types such as Gaussian, salt & pepper, speckle. These are obtained with default parameters, m=0 and v=0.01, d=0.05, v=0.04 respectively.

Let us illustrate with same examples.

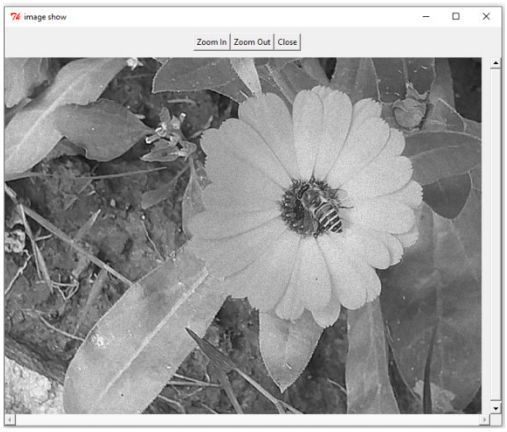
Example 9:

```
-->a=imread('F:/images/flower1/.jpg');
-->n1=imnoise(a,'gaussian');//imnoise (in,'gaussian',m,v)adds
   Gaussian noise of mean m and variance v;
-->imshow(n1);
-->a=rgb2 gray(a);
-->n2=imnoise(a,'salt & pepper',0.05);//imnoise.(im,'salt &
   pepper',d); adds drop-out noise, where d is the noise
   density, that is probability of swapping a pixel.
-->imshow(n2);
-->n3=imnoise (a,'speckle');
-->imshow(n3);
```

Outputs are in Fig. 17.



(a) Variable n1 **(b) Variable n2**



(c) variable n3

Fig. 17: Adding Noise

***

Now, let us discuss filtering.

Filtering

We shall cite following examples to view filtering.

Example 10:

```
-->a=imread('F:/images/flower1/.jpg');
-->F=[0 0 0; 0.33 0.33 0.33; 0 0 0]
 F  =
    0.        0.        0.
    0.33      0.33      0.33
    0.        0.        0.
 -->f1=imfilter(im,F);
 -->imshow(f1);
```

The corresponding output is shown in Fig. 18.

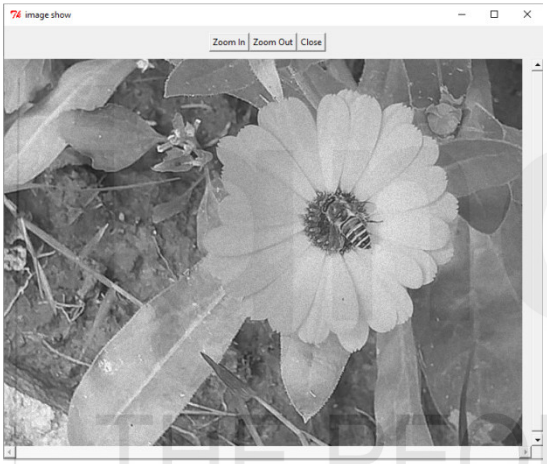


Fig. 15: Variable f1

***

Example 11: Let us do the following commands to see more filtering results on images.

```
-->a=imread('F:/images/flower1/.jpg');
-->F=[0.04 0.04 0.04 0.04 0.04; 0.04 0.04 0.04 0.04 0.04; 0.04
0.04 0.04 0.04 0.04; 0.04 0.04 0.04 0.04 0.04; 0.04 0.04 0.04
0.04 0.04];
 -->F=[0.04 0.04 0.04 0.04 0.04; 0.04 0.04 0.04 0.04 0.04; 0.04
0.04 0.04 0.04 0.04; 0.04 0.04 0.04 0.04 0.04; 0.04 0.04 0.04
0.04 0.04]
 F  =
    0.04      0.04      0.04      0.04      0.04
    0.04      0.04      0.04      0.04      0.04
    0.04      0.04      0.04      0.04      0.04
    0.04      0.04      0.04      0.04      0.04
    0.04      0.04      0.04      0.04      0.04
 -->f2=imfilter(a,F);
 -->imshow(f2);
```

The output is shown in Fig. 19.

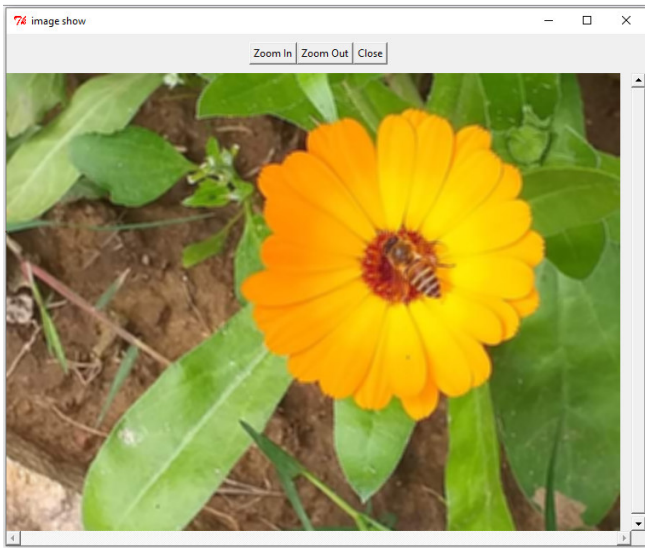


Fig.19: Variable f2

Example 12: Consider the different filter as given in the following commands:

```
-->a=imread('F:/images/flower1/.jpg');
-->F=[-1 -1 -1; -1 9 -1; -1 -1 -1];
-->f3=imfilter(im,F);
-->imshow(f3);
```

Fig. 20 depicts output.

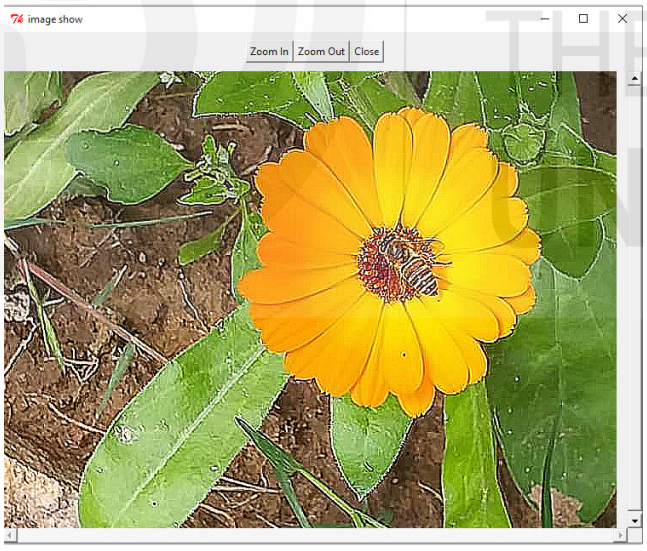


Fig. 20: Variable f3

Example 13: See the output for the following commands;

```
-->im=imread('F:/images/flower1/.jpg');
-->F=-1*ones(9,9);
-->F(5,5)=81;
-->f4=imfilter(a,F);
-->imshow(f4);
```

The output can be viewed in Fig. 21.

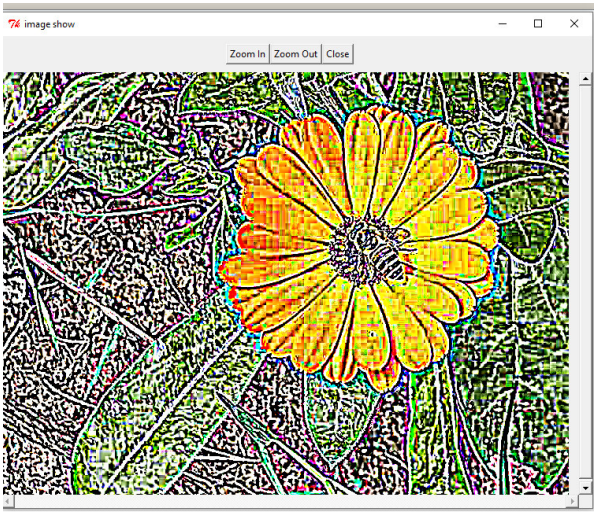


Fig. 21: Variable f4

Example 14: Consider the following commands and see the output.

```
-->a=imread('F:/images/flower1/.jpg');
 -->F=fspecial('sobel');
 -->f5=imfilter(a,F);
 -->imshow(f5);
```

The output is shown in Fig. 22.

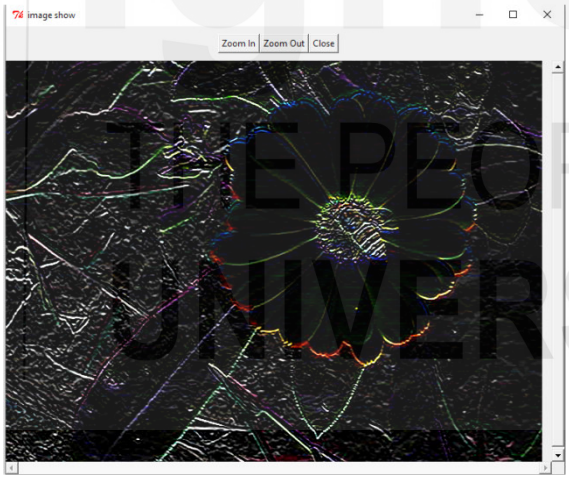


Fig. 22: Variable f5

You can try different values of the parameters and can view the corresponding outputs.

Now let us summaries what we have covered in this unit.

16.4 SUMMARY

In this unit, we have covered following points:

1. How to download SIVP.
2. Reading, writing and displaying an image.
3. Addition and Subtraction of images.
4. Filtering in images.
5. Adding noises to images.
6. Histogram of images.

Session 1 BASIC COMMANDS

Practical Assignment 1: Finding the roots of the polynomial equations.

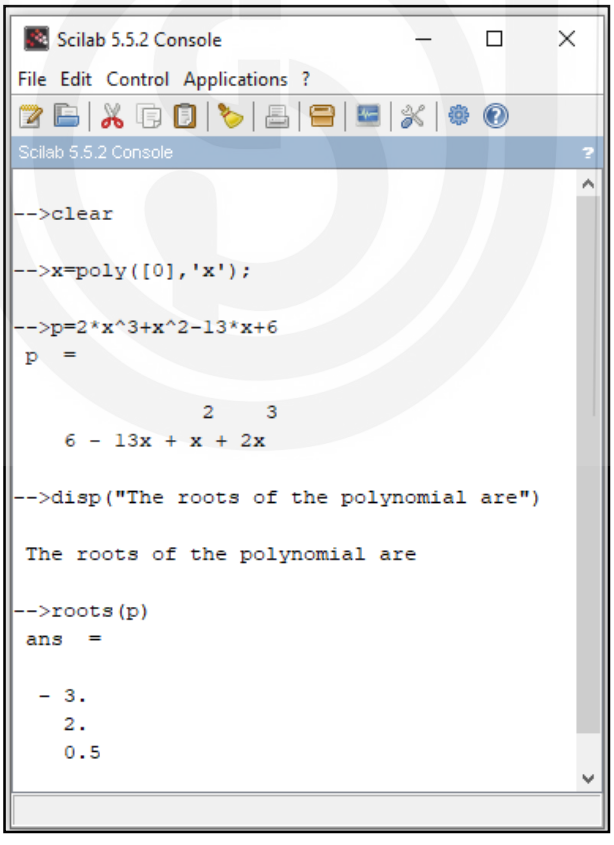
Solution: The source code is given as below:

```
--> clear;
-->clc;     //clears the console window
-->x = poly ([0] , ' x ' ) ; //defining the variable
-->p =2*(x^3)+x^2-13*x+6; //writing the polynomial equation
-->disp("the roots of above equations are" )
-->roots(p)
```

The output:

```
The roots of the polynomial are
ans  =
     - 3.
     2.
     0.5
```

The corresponding window will look like



***

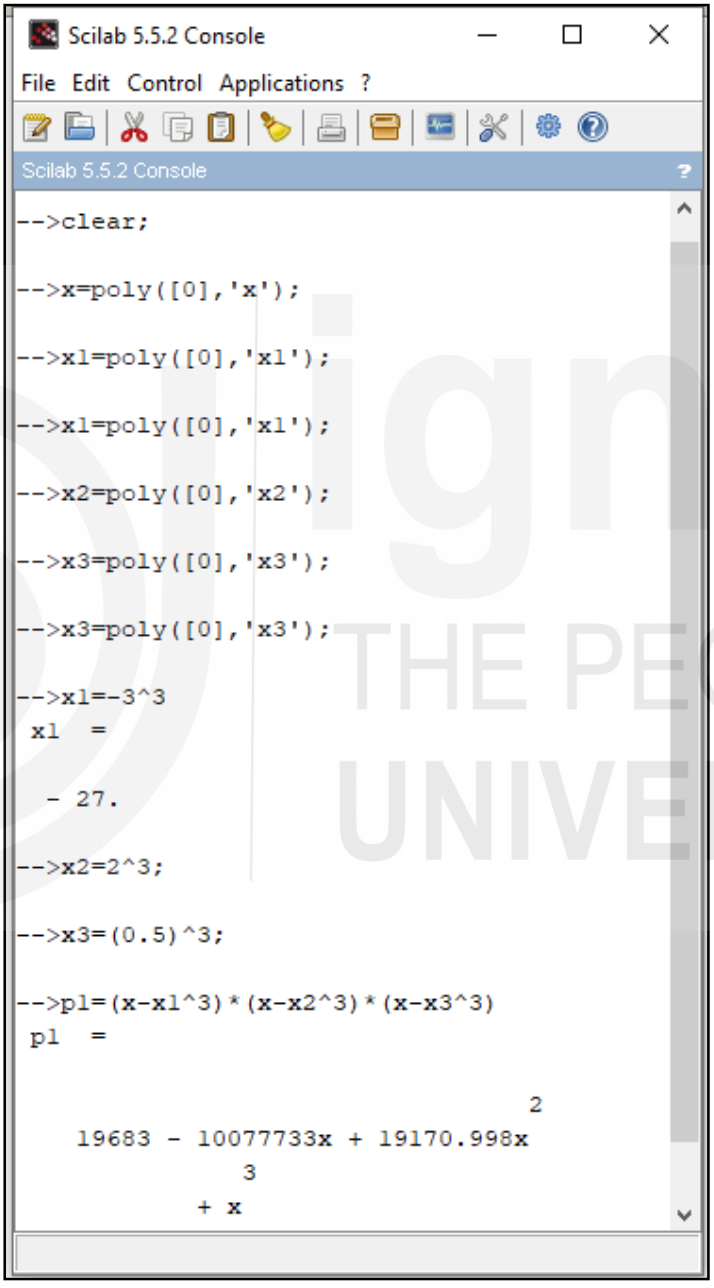
Practical Assignment 2: Finding the polynomial equations with roots are the cubes of the roots of the polynomial given in practical assignment 1.

Solution: The source Code is as given below:

```
-->clear;
```

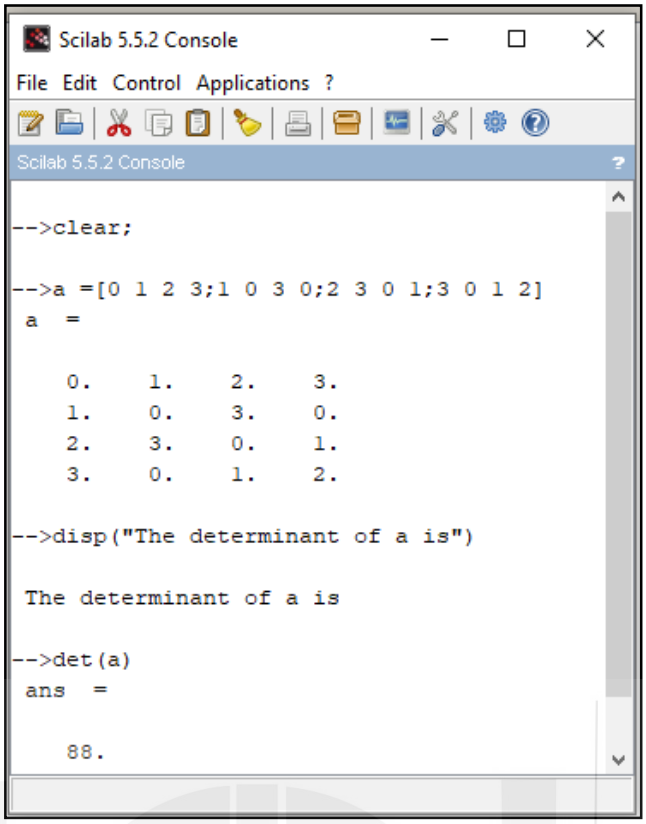
```
-->x1=poly([0],'x1');
-->x2=poly([0],'x2');
-->x3=poly([0],'x3');
-->disp("Let")
-->x1=-3^3
-->x2=2^3
-->x3=(0.5)^3
-->p1=(x-x1^3)*(x-x2^3)*(x-x3^3)
```

The console window is:



***

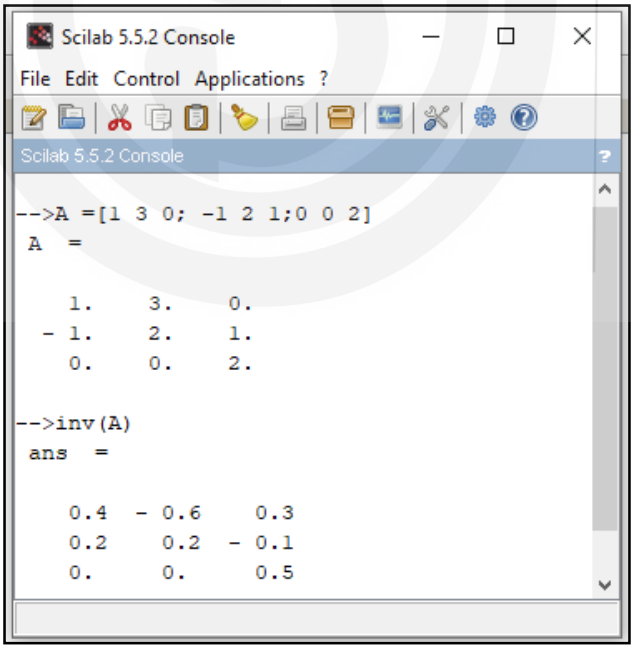
Practical Assignment 3: Calculating the value of a determinant.
Solution: The console window along with output of each of the step is given below:



***

Practical Assignment 4: Finding inverse of the matrix.

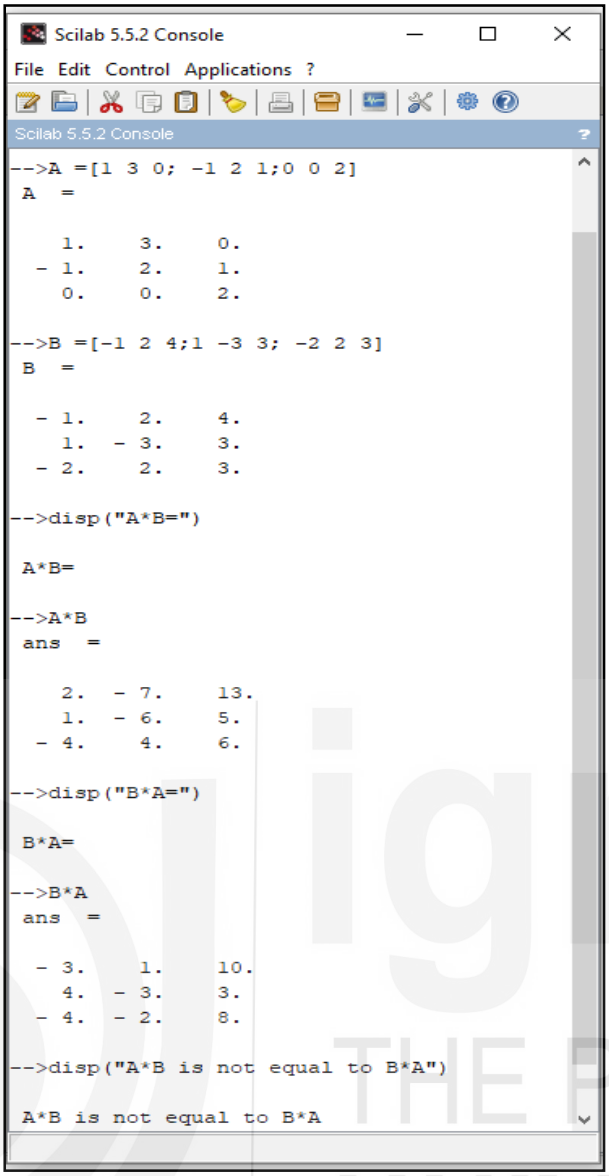
Solution:



***

Practical Assignment 5: Finding product of two matrices.

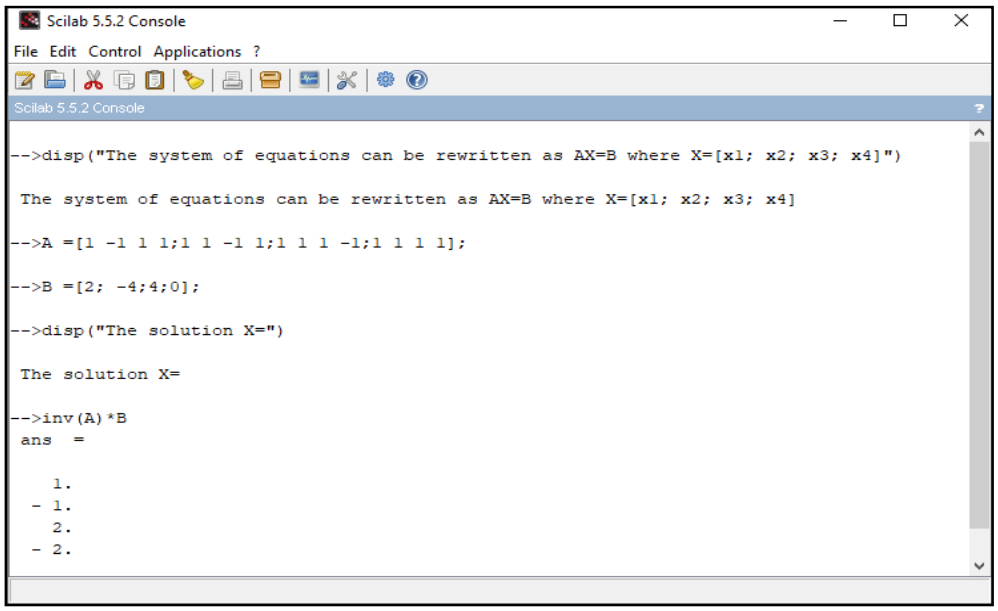
Solution:



***

Practical Assignment 6: Solving system of equations using matrices.

Solution:

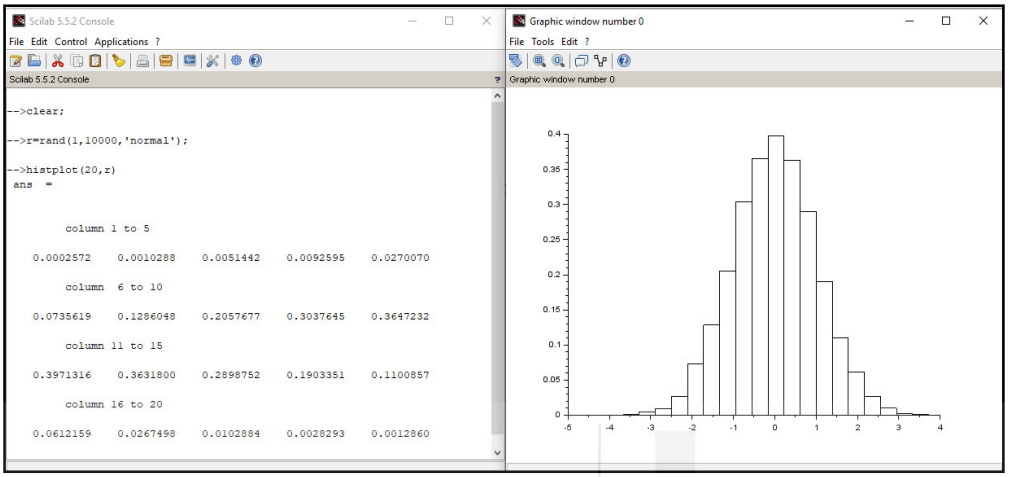


***

Session 2 PLOTTING IN SCILAB

Practical Assignment 1: Plotting a histogram.

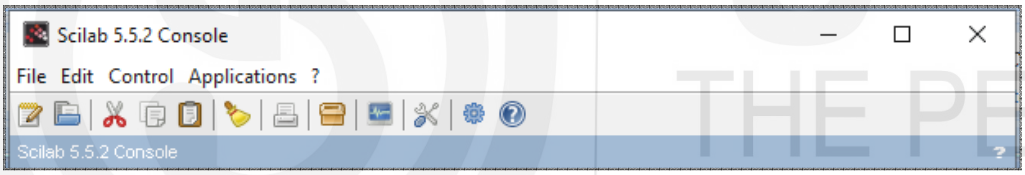
Source Code: The command *histplot(range of the variable, variable name)* is used to plot a histogram.



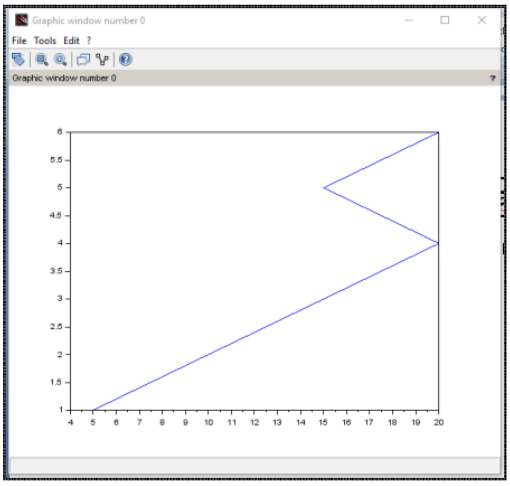
***

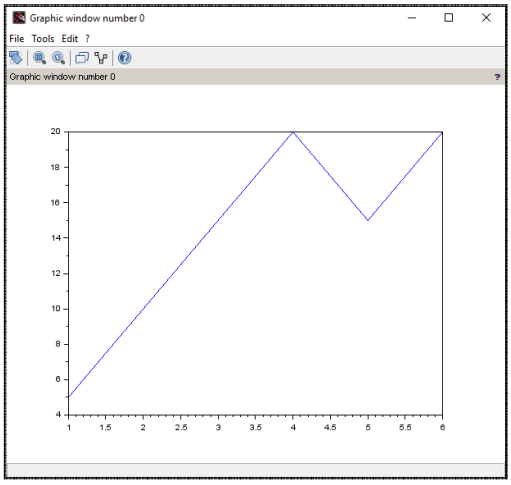
Practical Assignment 2: Plotting the points in graph and two graphs showing in the same window using subplot command.

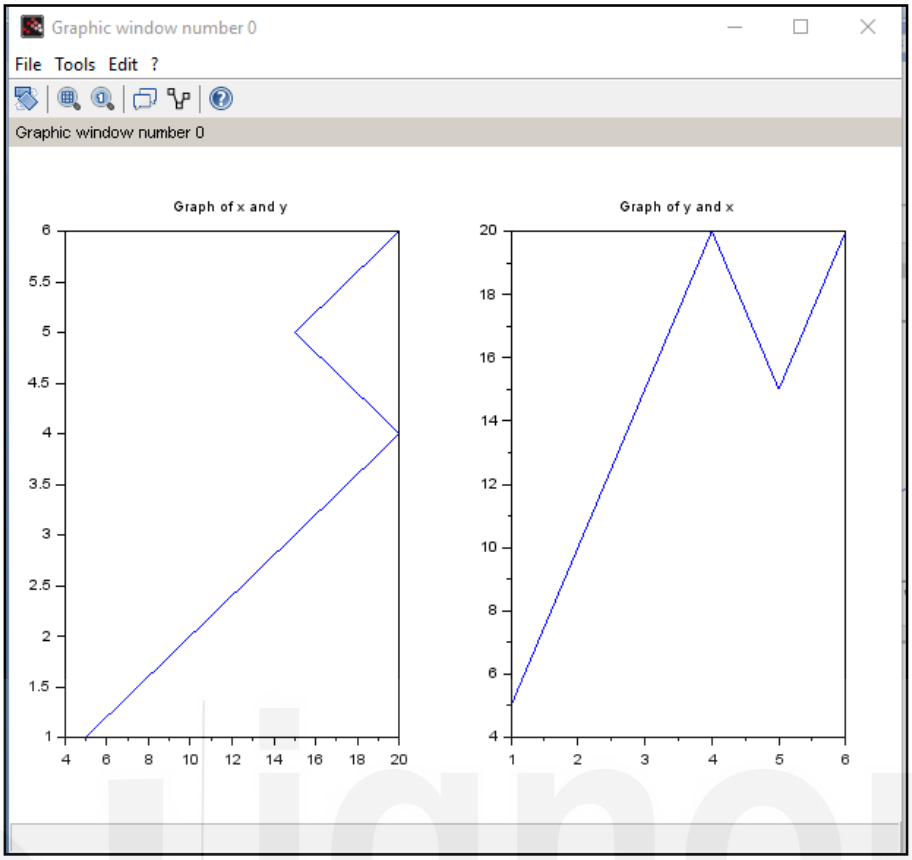
Source Code:




```
-->clear;
-->x=[5, 10, 15, 20, 15, 20];
-->y=[1, 2, 3, 4, 5, 6];
-->plot (x,y)
-->plot (y,x)
-->subplot (1,2,1); plot (x,y); title ("Graph of x and y")
-->subplot (1,2,2); plot (y,x); title ("Graph of y and x")
-->
```



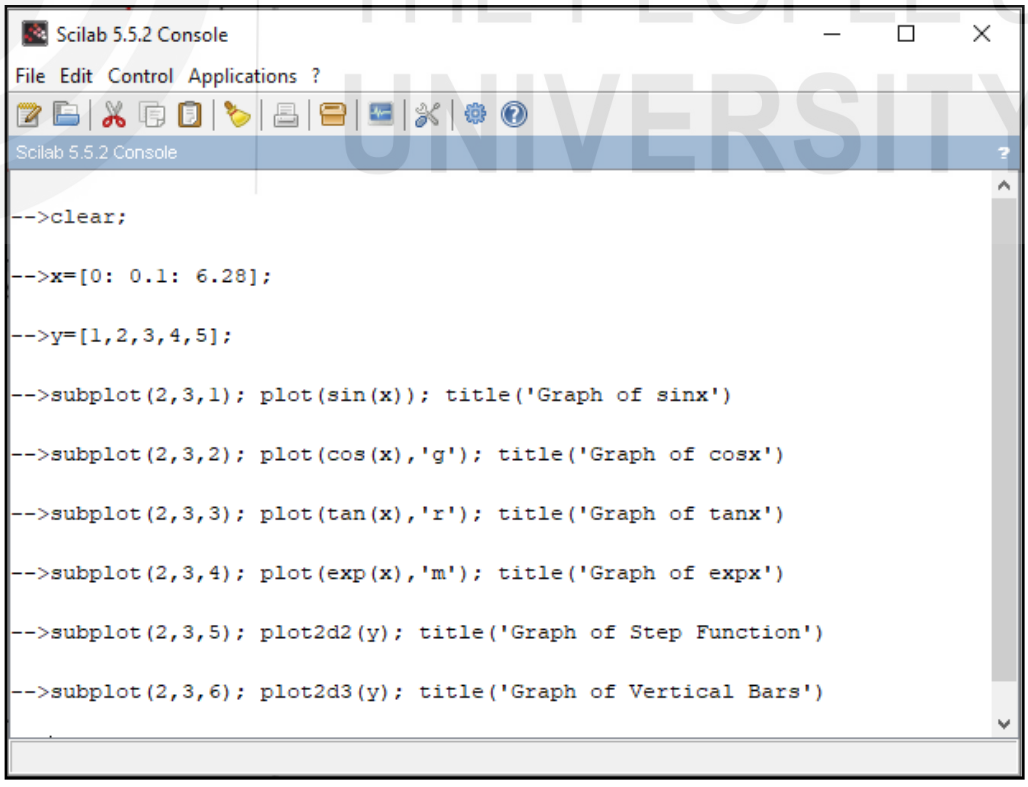


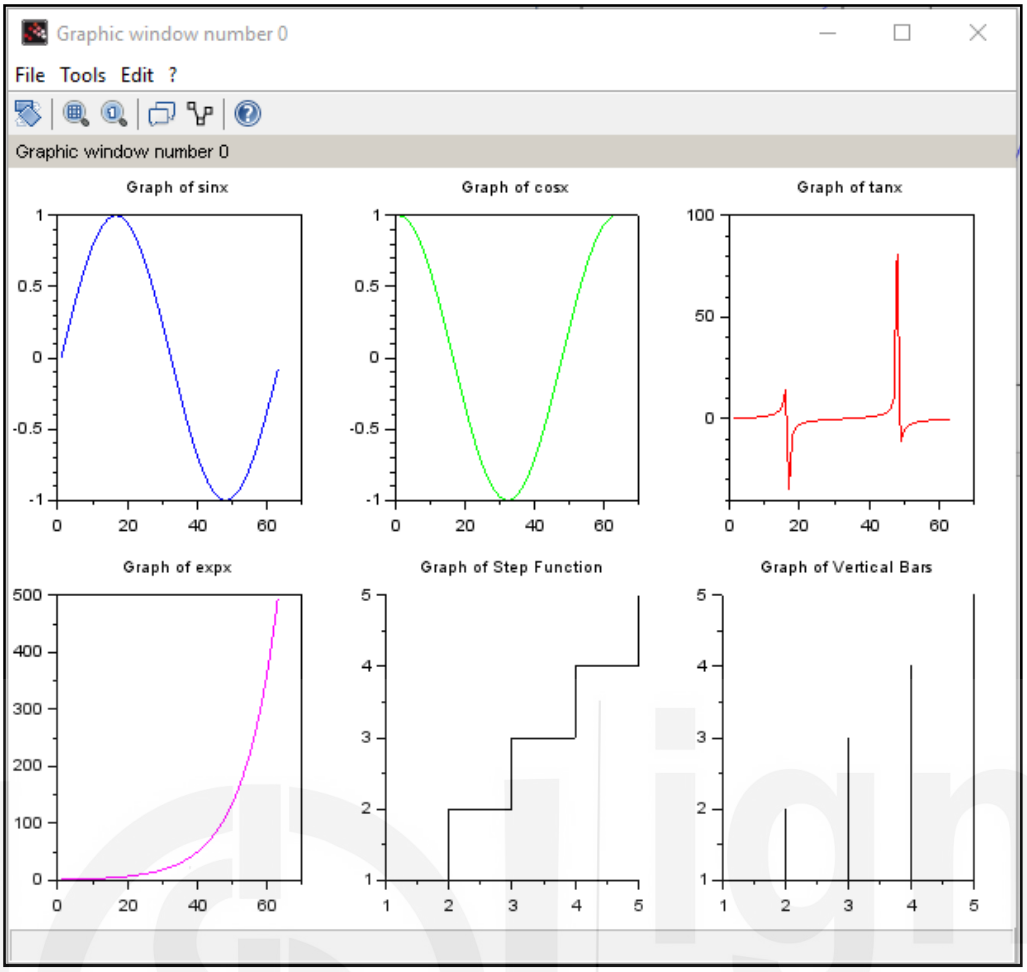


***

Practical Assignment 3: Plotting functions in one window.

Source Code: Several plots can be shown in one graphic window by defining the position in subplot function.

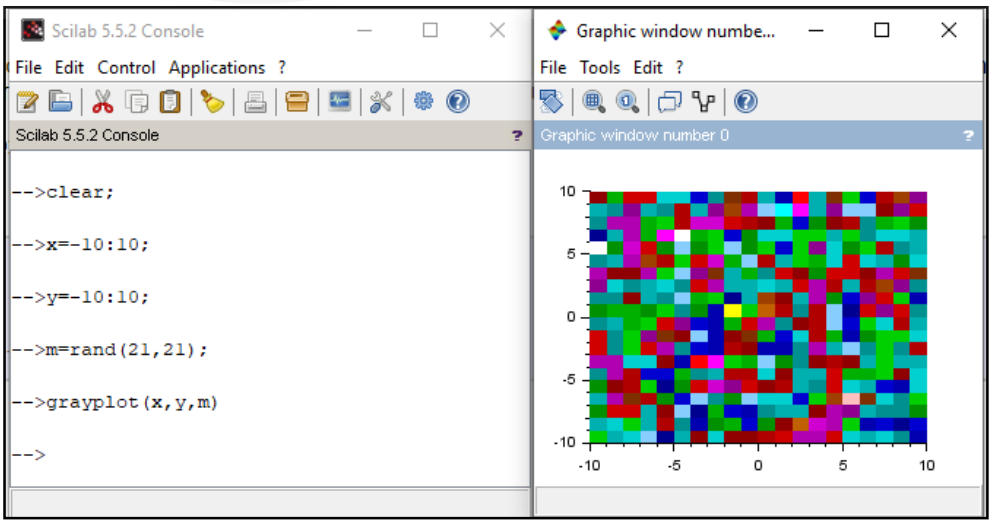




***

Practical Assignment 4: Plotting mat using grayplot command.

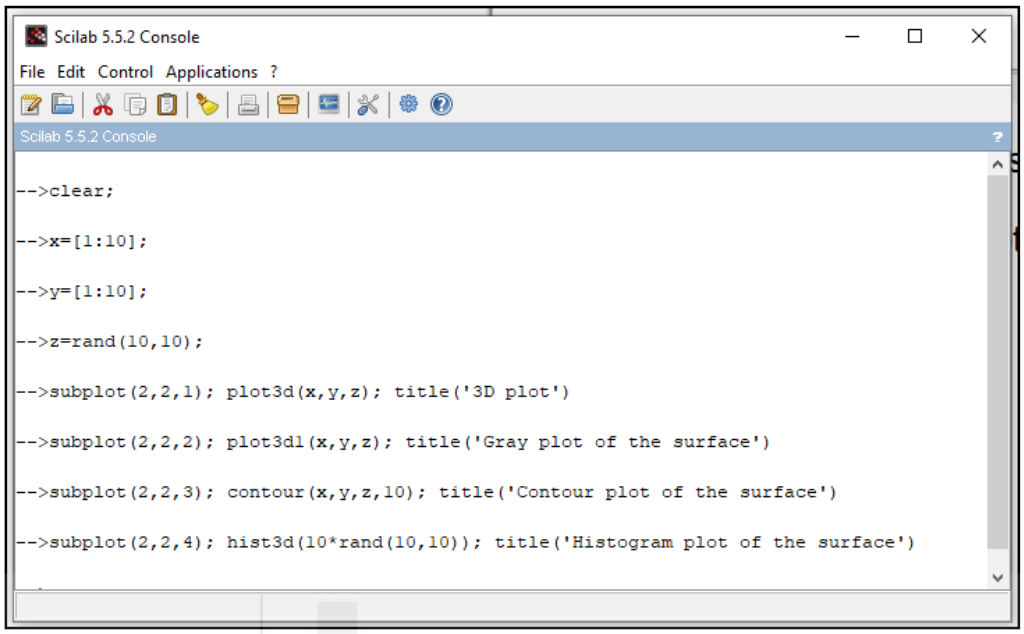
Solution: The command grayplot is a 2D plot of the defined surface into a mat grid, in which each rectangle is filled with a gray or a colour level depending on the average value of the random variable m on the corners of the rectangle. The command and the output window are as given below.

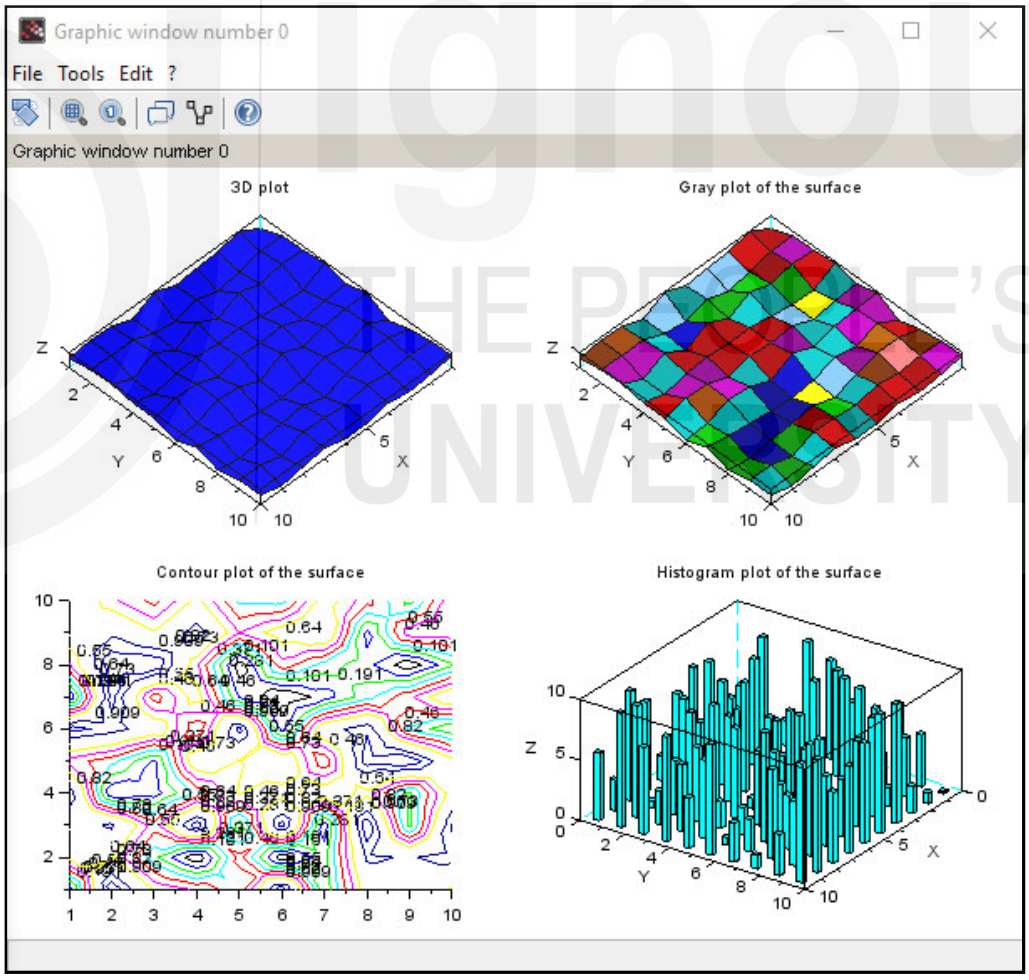


In these plots the for loop can also be used to let the variables vary.

***

Practical Assignment 5: 3D Plotting.
Solution:



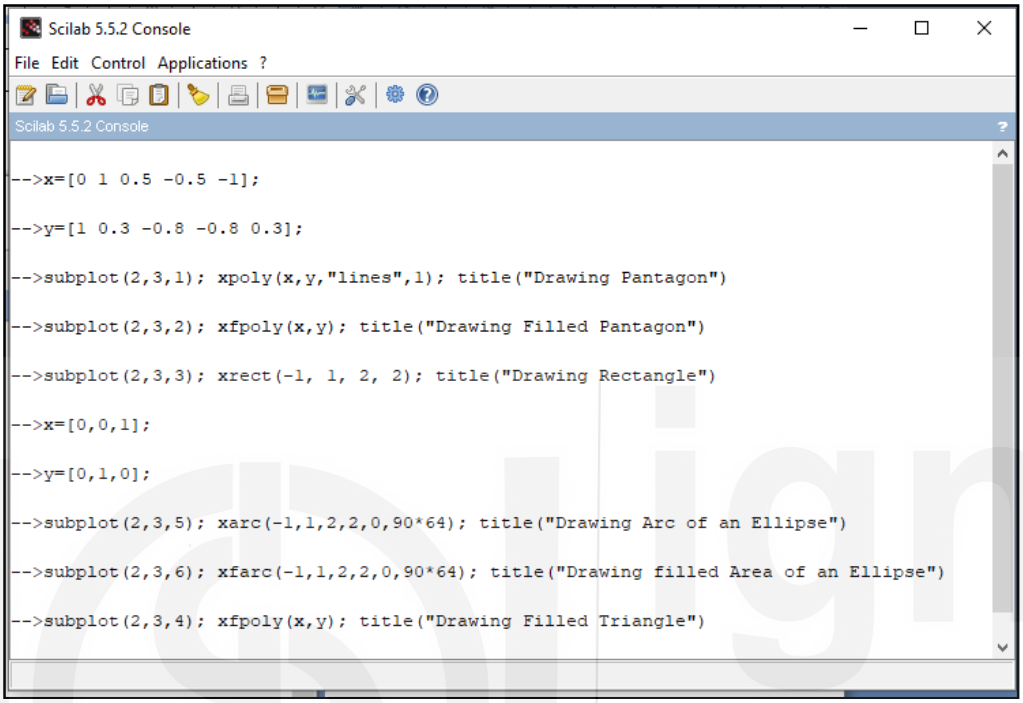


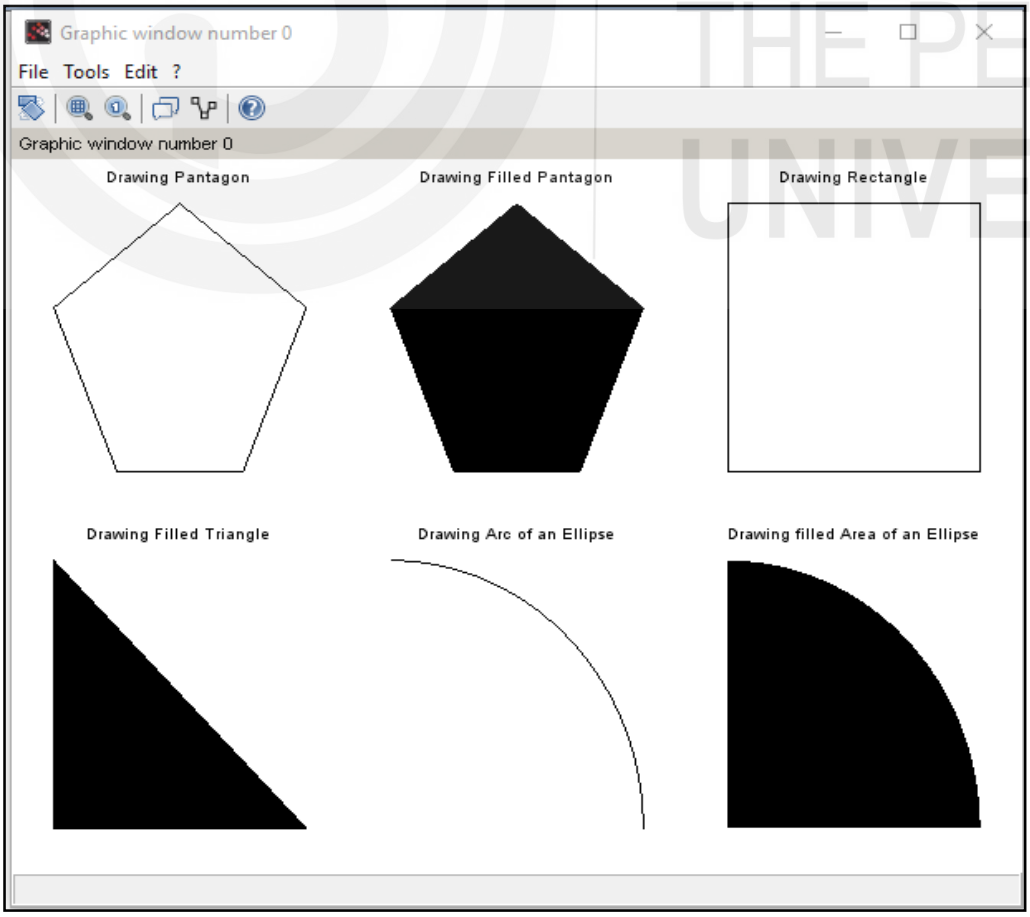
***

Practical Assignment 6: Plotting polygons.

Solution: The following commands are used for drawing polygons:

xpoly	Draws a regular polygon with n sides.
xfpoly	Draws a regular filled polygon with n sides.
xrect	Draws a rectangle with four values which are the corresponding values of pairs of parallel lines of x and y.
xarc	Draws an arc of an ellipse.
xfarc	Draws the filled area of an ellipse.





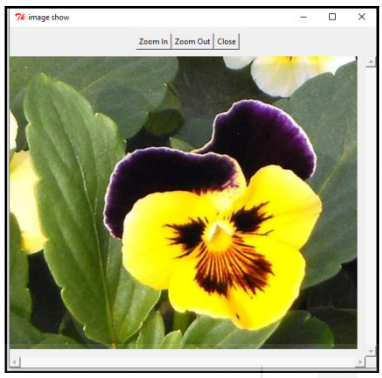
***

Session 3 SIVP BASIC COMMANDS

Practical Assignment 1: Reading and showing an image.

Solution:

```
-->a=imread('F:/images/flower1.jpg');
-->imshow(a);
```



The output of the command
`imfinfo('F:/images/flower1.jpg')`
is given as:

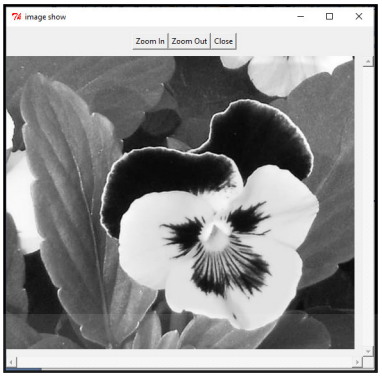
```
ans(1)
!V           !
!            !
!Filename    !
!            !
!FileSize    !
!            !
!Width       !
!            !
!Height      !
!            !
!BitDepth    !
!            !
!ColorType   !
         ans(2)
  F:/images/flower1.jpg
         ans(3)
      75537.
         ans(4)
      543.
         ans(5)
      457.
         ans(6)
      8.
         ans(7)
  truecolor
```

***

Practical Assignment 2: Load an image (e.g. tress or clown) in Scilab in gray shades. Change the number of gray levels to 64, 16, 8, 4, 2 to display the same image.

Solution:

```
im=imread('F:/images/flower1.jpg');
img=rgb2gray(im);
imshow(img)
```



Try to change the gray levels and show the effect.

***

Practical Assignment 3: Change the resolution of the image. Explain the effect of sampling error.

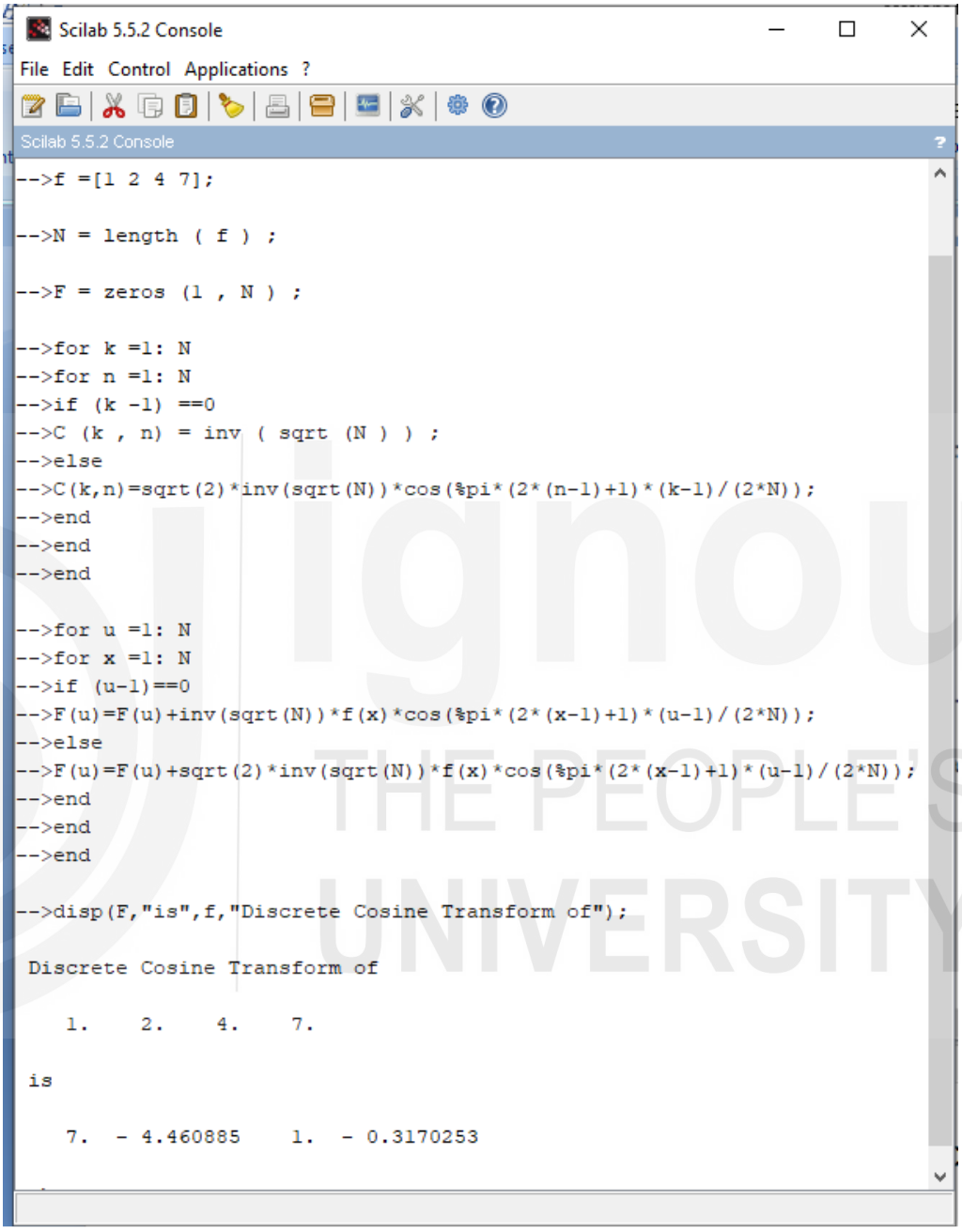
Solution: Refer to Sec. 16.3.

***

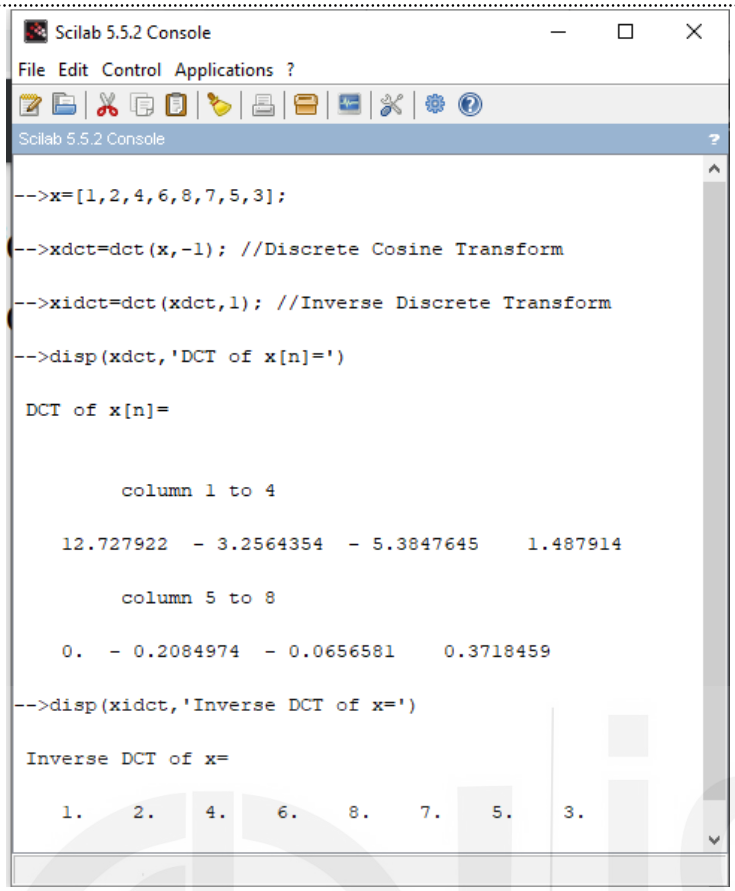
Session 4 IMAGE TRANSFORMS

Practical Assignment 1: Find its Discrete Cosine Transform.

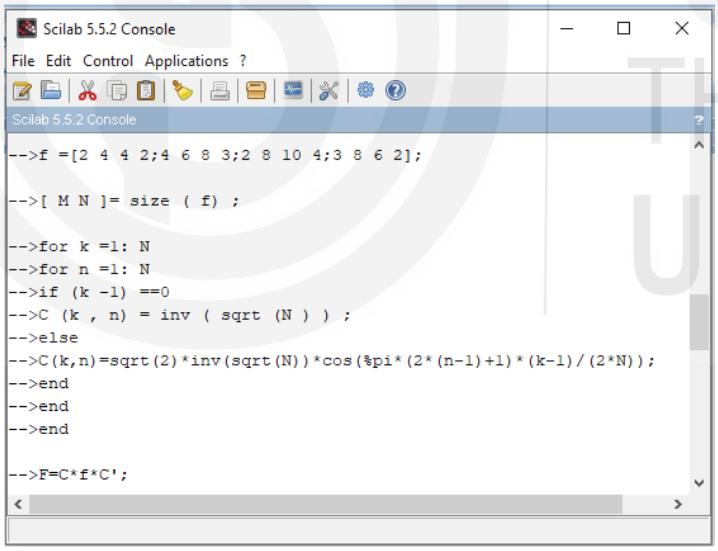
Solution: Refer to Sec. 3.3



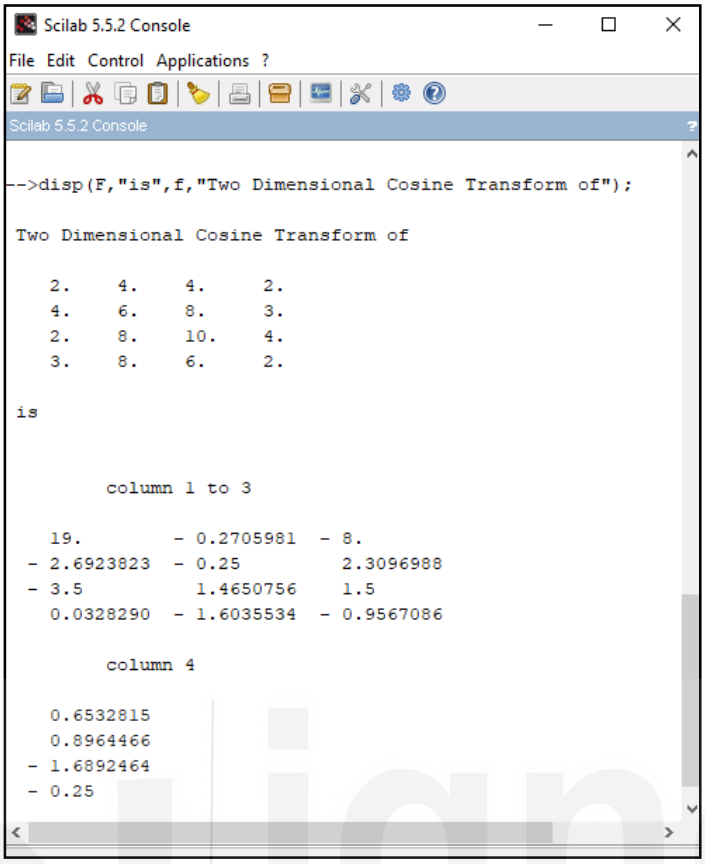
Using the direct command we get the following output:



The 2-D DCT is as given below



The corresponding output is as follows:



***

Practical Assignment 2: Load an image. Find its digital negative and perform thresholding.

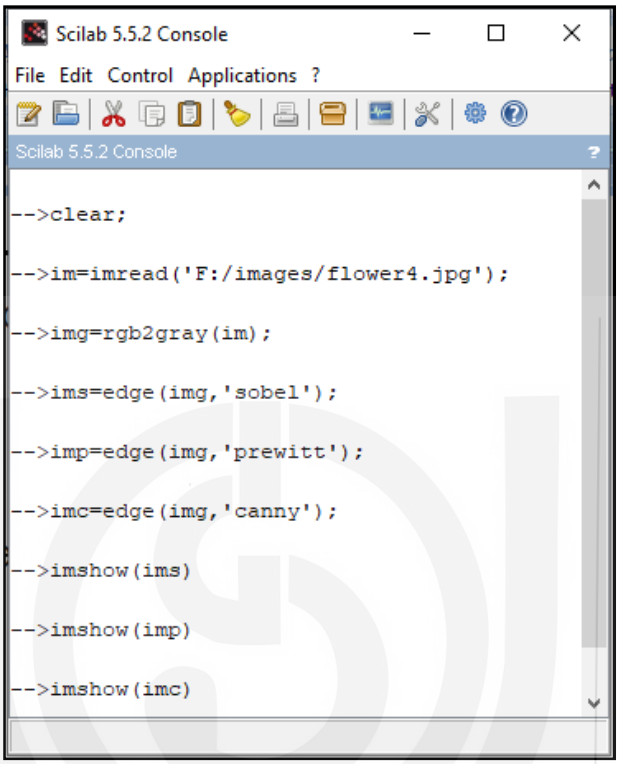
Solution: You may like to do it yourself.

***

Session 5 EDGE DETECTION

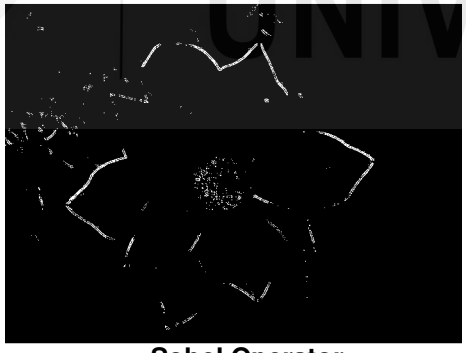
Practical Assignment 1: Take an image. Carry out edge detection in both directions using
Sobel operator
Prewitt operator
Canny operator
Use the corresponding masks and then carry out edge detection.

Solution:

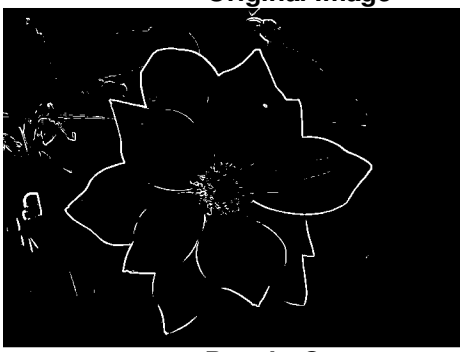


The corresponding output is as shown below:

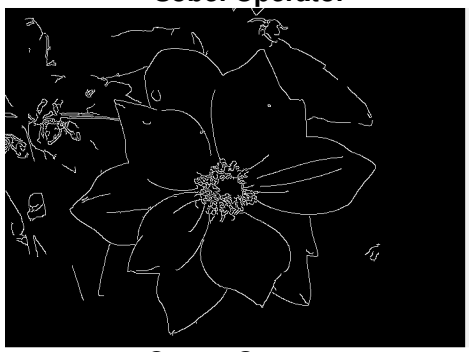
Original Image



Sobel Operator



Prewitt Operator



Canny Operator

***

Practical Assignment 2: Take an image and pass it through a low pass and high pass filter. Comment on your observation.

Solution: Low Pass Filters

```
-->clear;
-->im=imread('F:/images/bird1.jpg');
-->img=rgb2gray(im);
-->imshow(im)
-->h1=1/9*ones(3,3); //3x3 mask
-->im1=imfilter(im,h1);
-->imshow(im1)
-->h2=1/25*ones(5,5); //5x5 mask
-->im2=imfilter(im,h2);
-->imshow(im2)
```

Original image **Low pass filter 3x3** **Low pass filter 5x5**

The effect of low pass filters is blurring. This is because these filters allow low intensity to pass through.

High Pass Filters

```
-->im=imread('F:/images/bird1.jpg');
-->img=rgb2gray(im);
-->h1=-1*ones(3,3); //3x3 mask
-->h2=-1*ones(9,9); //9x9 mask
-->h1(2,2)=10;
-->h2(5,5)=75;
-->im1=imfilter(img,h1);
-->im2=imfilter(img,h2);
-->imshow(im1)
-->imshow(im2)
```

Original image **High pass filter h1** **High pass filter h2**

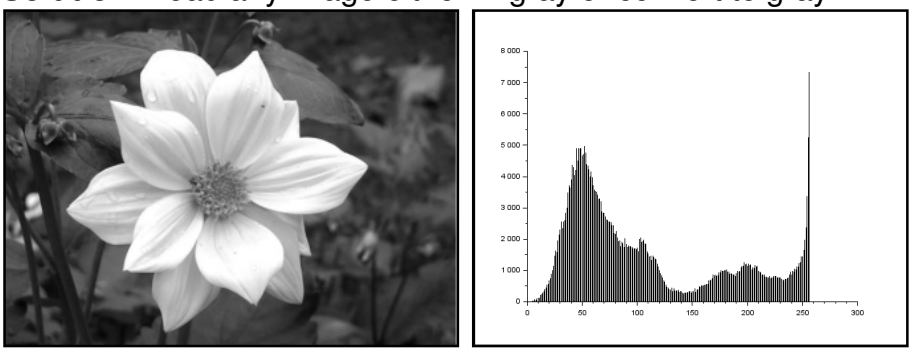
The high pass filters are exactly opposite to the low pass filters. These filters allow the high intensity to pass through.

***

Session 6 HISTOGRAM

Practical Assignment 1: Take a low contrast image and improve the quality of the image using histogram modification.

Solution: Load any image either in gray or convert to gray.



Perform the histogram equilisation and show the image again. The change in the image after histogram equilisation to be interpreted.

Refer to Sec. 16.4.

***

Practical Assignment 2: Load an image, add noise to it, display the noisy image, remove the noise using filtering operation.

Solution: Refer to Sec. 16.4

***

Session 7 DFFT AND IFFT

Practical Assignment 1: Take a small image. Compute its 2 – D FFT and display the magnitude in the form of image.

Solution:

```
-->f=zeros(30,30);
-->f(5:24,13:17)=1;
-->imshow(f)
-->F=fft2(f,256,256);
-->imshow(abs(F))
-->F2=fftshift(F);
-->imshow(abs(F2))
```

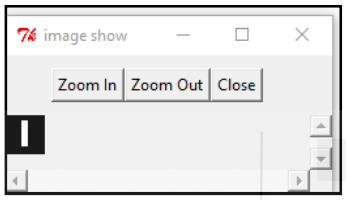


Image f

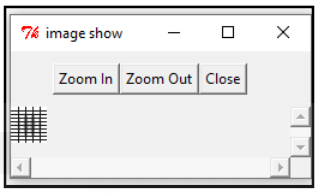


Image F

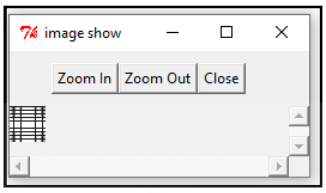


Image F2

***

Session 8 NOISE

Practical Assignment 1: Load an image. Choose a blurring function to blur this image and add noise to the blurred image. Now try to restore this image using

a) Wiener filter
b) Median filter

Comment on the result.

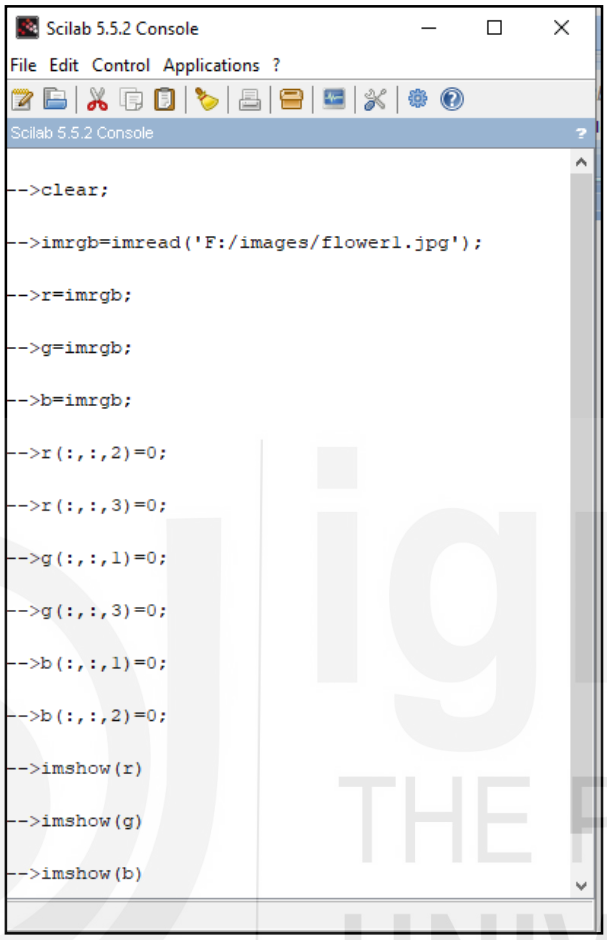
Solution: Steps are imread, imnoise, fspecial, imfilter, imshow functions, write code for Weiner filter.
Refer to Sec. 16.4.

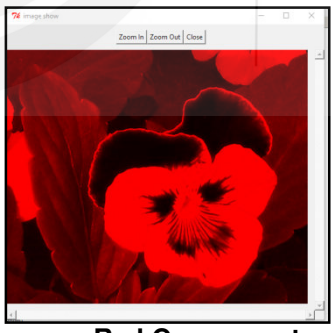
***

Session 9 COLOUR TRANSFORMATION

Practical Assignment 1: Load a colour image. Show R, G and B component present or missing in the image.

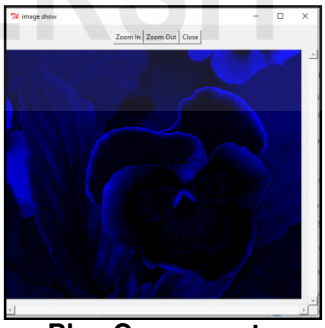
Solution:





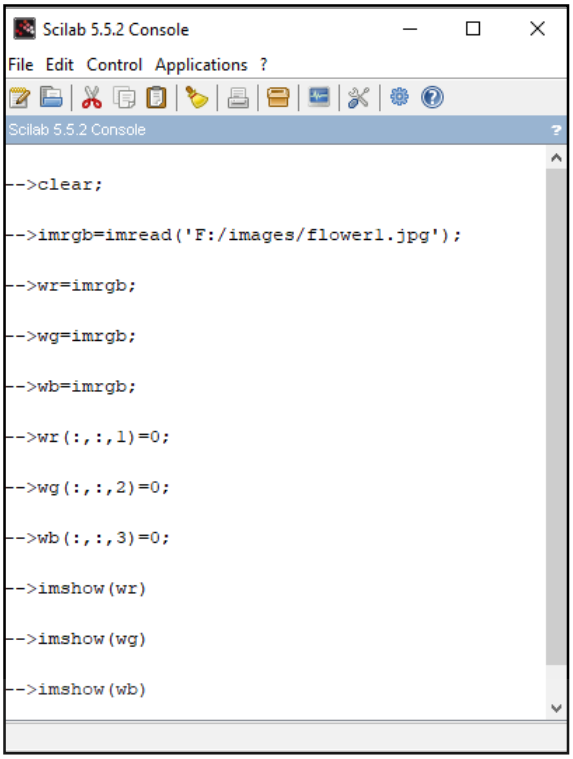
Red Component

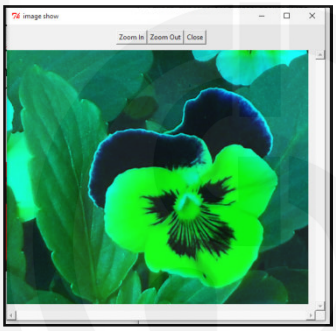
Green Component



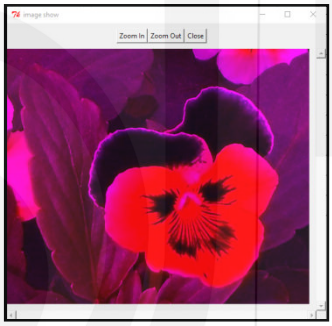
Blue Component

The following is the console window to get the output for missing R, G and B components.

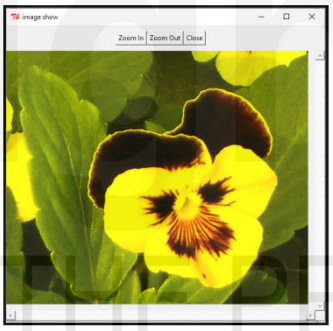




Missing Red Component



Missing Green Component



Missing Blue Component

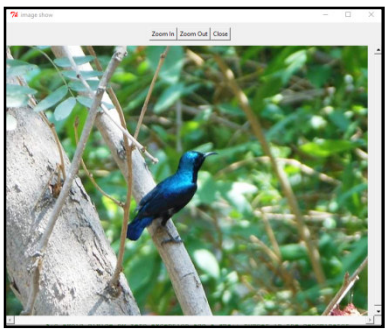
***

Practical Assignment 2: Load a colour image. Show RGB to HSI and vice versa colour transformations.

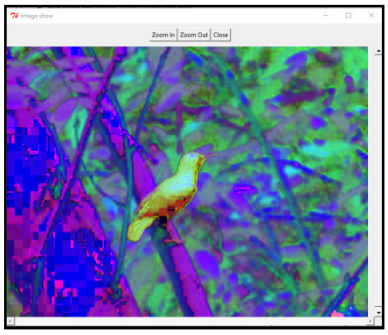
Solution: The steps are as follows:

- Read a RGB image using 'imread' function.
- Each RGB component will be in the range of [0 255].
- Represent the image in [0 1] range by dividing the image by 255.
- Find the theta value. If B<=G then H= theta. If B>G then H= 360-theta
- Use 'acosd' function to find inverse cosine and obtain the result in degrees.
- Divide the hue component by 360 to represent in the range [0 1]
- Similarly, find the saturation and the intensity components.
- Display the image.

Also, the direct command *rgb2hsv()* can be used.



Original Image



HSI Image

***

Practical Assignment 3: Load a colour image. Show RGB to CMYK and vice versa colour transformations.

Solution: You may like to try it yourself.

***

Practical Assignment 4: Load a grayscale, do pseudo-colouring.

Solution: You may like to try it yourself.

***

Session 10 IMAGE SEGMENTATION

Practical Assignment 1: Load an image and count the number of objects present in the image.

Solution: (i) Point Detection

```
-->f=imread('F:/images/flower4.jpg');
-->imshow(g)
-->w=[-1 -1 -1; -1 8 -1; -1 -1 -1];
-->g=abs(imfilter(double(f), w));
-->T=max(g(:));
-->g=g>=T;
-->imshow(w)
-->imshow(g)
```

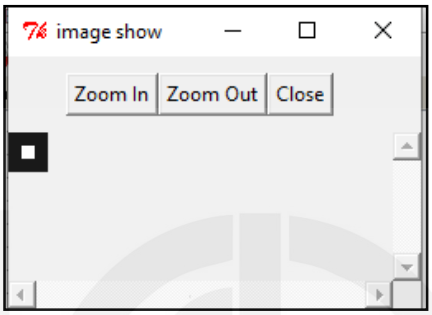


Image w

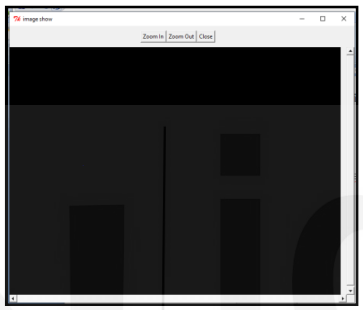


Image g

(ii) Line Detection

```
-->f=imread('F:/images/flower4.jpg');
-->w=[2 -1 -1; -1 2 -1; -1 -1 2];
-->imshow(w)
-->g=(imfilter(double(f), w));
-->imshow(g)
-->gtop=g(1:120, 1:120);
-->imshow(gtop)
```

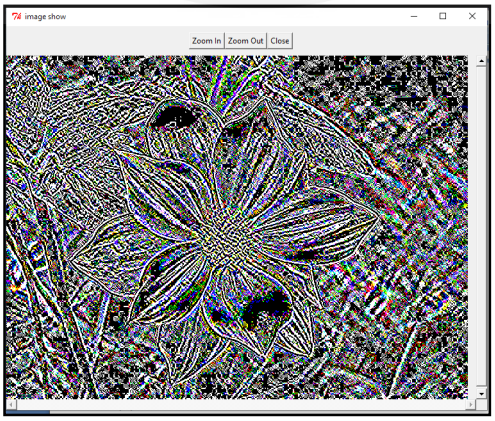


Image g

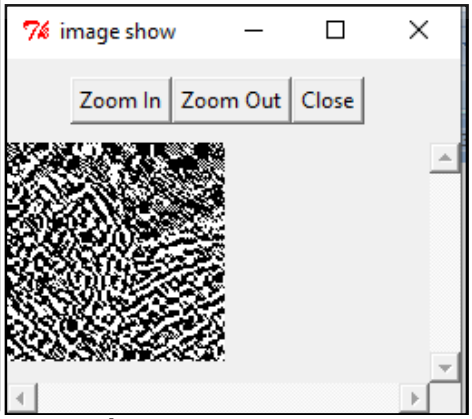


Image gtop

(iii) For edge detection, use sobel operator as given in Sec. 16.4.

***

References

[1] Digital Image Processing, Gonzalez and Woods, Pearson, third edition.

[2] Fundamentals of Digital Image Processing, A K Jain, Pearson.

[3] Digital Image Processing, S Sridhar, Oxford Higher Education.

[4] Digital Image Processing, Jayaraman, Esakkirajan, Veerakumar, McGraw Hill Education

[5] Digital Image Processing using Matlab, Gonzalez, Woods, McGraw Hill Education

[6] Digital Image Processing with Matlab and LavView, Vipula Singh, Elsevier.

[7] Pattern Classification, Duda, Hart, Stork, Wiley.

[8] Pattern Recognition and Machine Learning, Christopher Bishop, Springer.

[9] Pattern Recognition and Machine Analysis, Gose, Johnsonbaugh, Jost, Pearson

[10] Pattern Recognition Theodoridis, Koutroumbas, Elsevier.