GURU TEGH BAHADUR INSTITUTE OF TECHNOLOGY

NEW DELHI

SCILAB LAB MANUAL

BS-252

Lesson Plan for Probability,Statistic and Linear Programming Lab

Course Name: B.Tech. **Semester: 4th** **PAPER CODE: BS-252**

S.No	Topic Details	No of Hours	Reference/text book
First Term			
1.	**Unit-I:** **Introduction Basics: Probability and Statistical models, Sample Spaces and Events,Counting** **Techniques,Interpretations and Axioms of Probability, Unions of Events and Addition Rules,Conditional Probability, Intersections of Events and Multiplication and Total Probability Rules, Independence, Bayes' Theorem,** Random Variables. Discrete and Continuous Random Variables and Distributions: Probability Distributions and Probability Mass/ density Functions, Cumulative Distribution Functions, Mean and Variance of a Random Variable, Discrete and continuous Uniform Distribution Binomial Distribution, Geometric and Negative Binomial Distributions, Hypergeometric Distribution, Poisson Distribution. Normal Distribution, Normal Approximation to the Binomial, and Poisson Distributions; Exponential Distribution, Erlang and Gamma Distributions, Weibull Distribution, Lognormal Distribution, Beta Distribution,	2 3 2 3	T1, T2
2.	**UNIT-II :** Joint Probability Distributions for Two Random Variables, Conditional Probability Distributions and Independence, Joint Probability Distributions for Two Random Variables Covariance and Correlation, Common Joint Distributions, Linear Functions of RandomVariables, General Functions of Random Variables, Moment- Generating Functions. Numerical Summaries of Data, Stem-and-Leaf Diagrams, Frequency Distributions and Histograms, Box Plots, Time Sequence Plots, Scatter Diagrams, Probability Plots Point Estimation, Sampling Distributions	2 3 2	T2,R3,R5

	Central Limit Theorem without proof, General Concepts of Point Estimation, Methods of Point Estimation, Statistical Intervals for a Single Sample	3	
S			
3.	**UNIT-III Hypotheses Testing for a SingleSample: Tests on the Mean of a Normal Distribution with Variance Known / Unknown, Tests on the Variance and Standard Deviationof a Normal Distribution,** **Tests on a Population Proportion, Testing for Goodness of Fit, Nonparametric tests (Signed, Wilcoxon), Similarly Statistical Inference forTwo Samples.** **Regression and Correlation: Linear Regression, Least Squares Estimators, Hypotheses testing for simple linear** **regression, Confidence Intervals, Adequacy of model, Correlation, Transformed Variables, Logistic Regression.** **Similarly, for multiple linear regression including aspects of MLR.**	2 3 2 4	T1,T2, R1, R3,R4
4.	Linear Programming: Introduction, formulation of problem, Graphical method, Canonical and Standard form of LPP, Simplex method, Duality concept, Dual simplex method, Transportation and Assignment problem	3 2	T2, R1, R6,

Text Books:

[T1] Applied Statistics and Probability for Engineers by Douglas G. Montgomery and Runger, Wiley, 2018 2. Linear Programming by G. Hadley, Narosa, 2002

[T2] Linear Programming by G. Hadley, Narosa, 2002

References Books:

[R1] Miller and Freund's Probability and Statistics for Engineers by Richard A. Johnson, Pearson, 10" Ed., 2018.

[R2] Probability & Statistics for Engineers & Scientists by Ronald E. Walpole, Raymond H. Myers, Sharon L. Myers and Keying Ye, Pearson, 2016.

[R3] Statistics and probability with applications for engineers and scientists using Minitab, R and JMP, C. Gupta, Irwin Guttman, and Kalanka P. Jayalath, Wiley, 2020.

[R4] Probability and Statistics for Engineering and the Sciences, Jay Devore, Cengage Learning, 2014.

[R5] Probability and Statistics in Emgineering, William W. Hines, Douglas C.Montgomery, David M. Goldman,and Connie M. Borror, Wiley, 2003.

[R6] Operations Research: An Introduction by Hamdy A. Taha, Pearson, 10th Edition, 2016

List of Experiments

1. Installation of Scilab and demonstration of simple programming concepts like transpose of a matrix, matrix addition, and matrix multiplication (using loops).

2. To find factorial of a Natural Number.

3. To generate n number of terms of Fibonacci Series.

4. To classify odd & even numbers upto a limit.

5. To find average of 10 numbers.

6. Fitting of Linear regression lines through given data set and testing of goodness of fit using mean errorProgram for demonstration of theoretical probability limits.

7. Fitting of binomial distributions for given n and p.

8. Fitting of binomial distributions after computing mean and variance.
9. Fitting of Poisson distributions for given value of lambda.

10. Fitting of normal distribution when parameters are given.

11. Solve a LPP of three variable using Simplex Method in three variables.

12. Solve a Transportation problem of three variables.

13. Solve an Assignment problem of three variables.

14. Program to plot normal distributions and exponential distributions for various parametric values.

INTRODUCTION

Scilab is a programming language associated with a rich collection of numerical algorithms covering many aspects of scientific computing problems.

From the software point of view, Scilab is an interpreted language. This generally allows to get faster development processes, because the user directly accesses to a high level language, with a rich set of features provided by the library. The Scilab language is meant to be extended so that user-defined data types can be defined with possibly overloaded operations. Scilab users can develop their own module so that they can solve their particular problems. The Scilab language allows to dynamically compile and link other languages such as Fortran and C and in this way, external libraries can be used as if they were a part of Scilab built-in features. Scilab is also a numerical computation software that anybody can freely download. Available under Windows, Linux and Mac OS X, Scilab can be downloaded at the following address http://www.scilab.org/.

An online help is provided in many local languages. From a scientific point of view, Scilab comes with many features. At the very beginning of Scilab, features were focused on linear algebra. But rapidly, the number of features extended to cover many areas of scientific computing such as matrices, statistics, Ordinary differential equations, signal processing, interpolation, approximation, linear, quadratic and non linear optimization and many more.

The Menu Bar

The following options in menu are particularly useful:

Applications

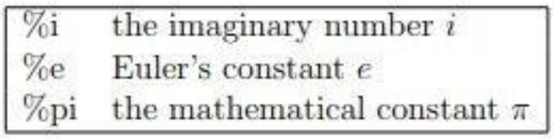
- The command history allows you to find all the commands from previous sessions to the current session.
- The variables browser allows you to find all variables previously used during the current session.

Editing

- Preferences (in Scilab menu under Mac OS X) allow you to set and customize colors, fonts and font size in the console and in the editor, which is very useful for screen projection.
- Clicking on Clear Console clears the entire content of the console. In this case, the command history is still available and calculations made during the session remain in memory. Commands that have been erased are still available through the keyboards arrow keys.

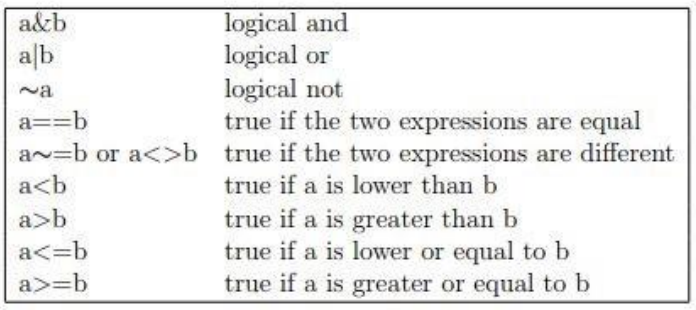
1.4 Pre-defined mathematical variables and operators

In Scilab, several mathematical variables are pre-defined variables, which name begins with a percent % character. The variables which have a mathematical meaning are summarized in the given table:



%i	the imaginary number i
%e	Euler's constant e
%pi	the mathematical constant π

Also apart from usual operators for summation, subtraction, multiplication and division, comparison operators are as given:



a&b	logical and
a|b	logical or
∼a	logical not
a==b	true if the two expressions are equal
a∼=b or a<>b	true if the two expressions are different
a<b	true if a is lower than b
a>b	true if a is greater than b
a<=b	true if a is lower or equal to b
a>=b	true if a is greater or equal to b

1.5 Booleans

Boolean variables can store true or false values. In Scilab, true is written with %t or %T and false is written with %f or %F.

1.6 Complex Numbers

Scilab provides complex numbers, which are stored as pairs of floating point numbers. The predefined variable %*i* represents the mathematical imaginary number *i* which satisfies $i^2 = -1$.

1.7 Strings

Strings can be stored in variables, provided that they are delimited by double quotes. The concatenation operation is available from the + operator. They are many functions which allow to process strings, including regular expressions.

1.8 Matrices

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:

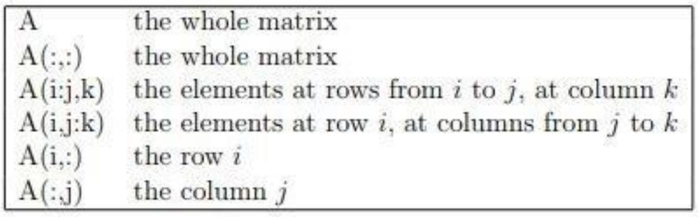
- square brackets ”[” and ”]” mark the beginning and the end of the matrix
- commas ”,” separate the values on different columns
- semicolons ”;” separate the values of different rows

The following syntax can be used to define an $m \times n$ matrix, where blank spaces are optional (but make the line easier to read) and ”...” are designing intermediate values:

$A = [a_{11},a_{12},...,a_{1n};a_{21},a_{22},...,a_{2n};...;a_{m1},a_{m2},...,a_{mn}]$

Use of colon operator in matrices

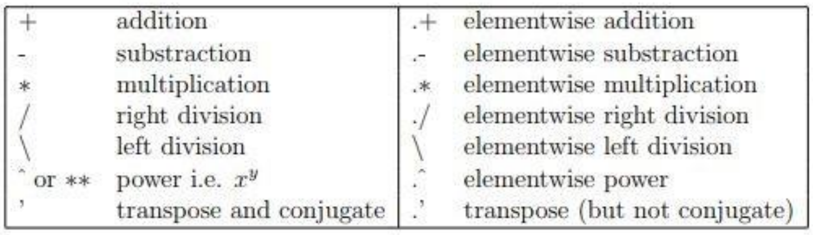
Following table depicts use of colon operator in matrices:



A	the whole matrix
A(:,:)	the whole matrix
A(i:j,k)	the elements at rows from i to j, at column k
A(i,j:k)	the elements at row i, at columns from j to k
A(i,:)	the row i
A(:,j)	the column j

Matrix and usual operators

Following table shows various operator used in matrix operations:



+	addition	.+	elementwise addition
-	substraction	.-	elementwise substraction
*	multiplication	.*	elementwise multiplication
/	right division	./	elementwise right division
\	left division	\	elementwise left division
ˆ or **	power i.e. x^y	.ˆ	elementwise power
’	transpose and conjugate	.’	transpose (but not conjugate)

1.9 Looping and Branching

In this section, we describe how to make conditional statements

The if statement

The if uses a boolean variable to perform its choice: if the boolean is true, then the statement is executed. A condition is closed when the end keyword is met. 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 ( %f ) 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 session, we use the == operator to display the message ”Hello ! ”.

```
i = 2 if ( i == 2 )
then disp (” Hello
! ”)

else
disp (” Goodbye !” ) end
```

TABLE OF CONTENTS

Experiment -1

Aim- Demonstration of simple programming concepts like transpose of a matrix, matrix addition, and matrix multiplication (using loops).

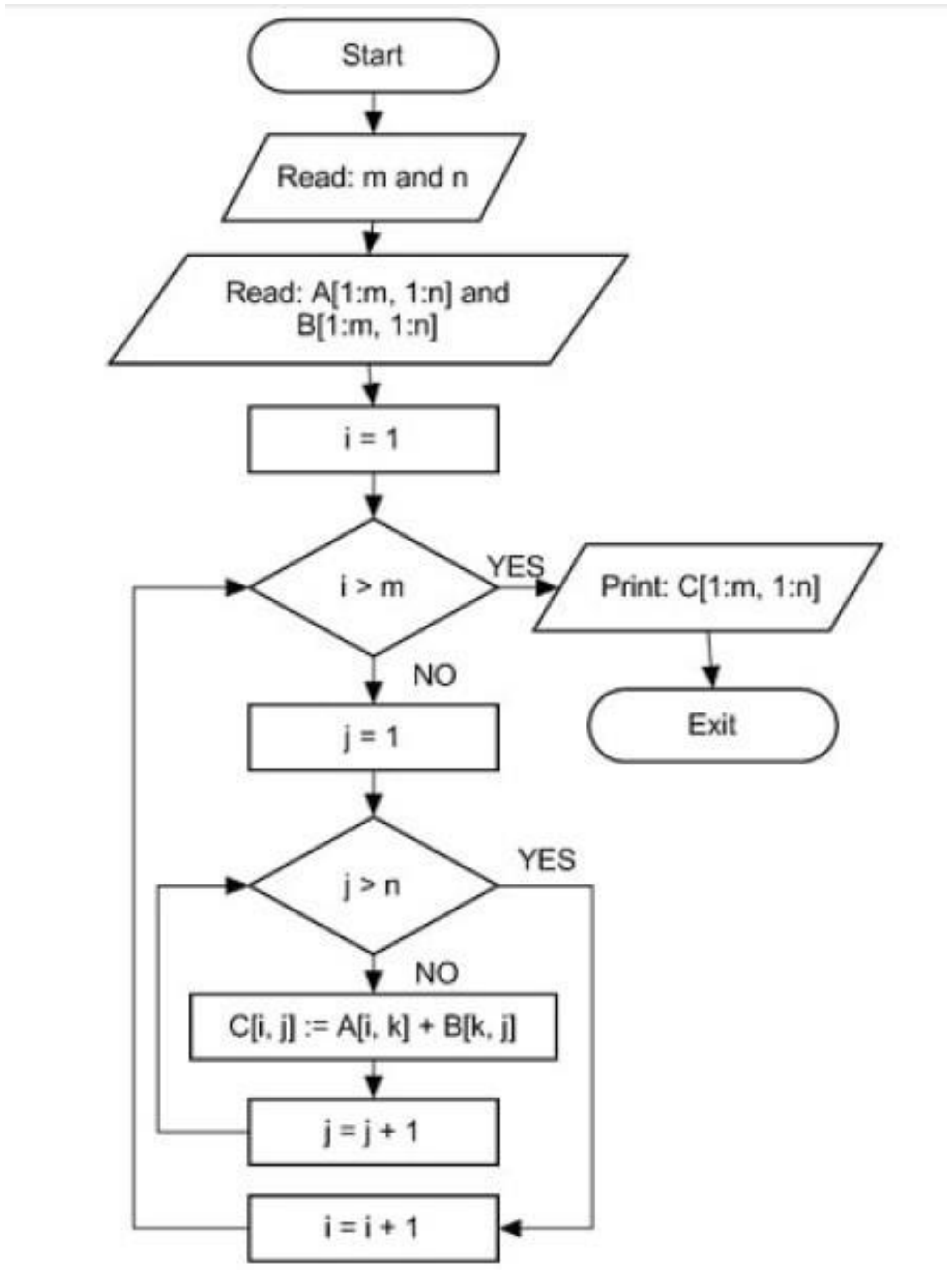
1.1 Matrix Addition

- $(A + B)_{i,j} = A_{i,j} + B_{i,j}$
- Order of the matrices must be the same
- Matrix addition is commutative

Matrix addition is associative **Matrix Addition Algorithm.**

1. Start
2. Declare variables and initialize necessary variables
3. Enter the elements of matrices row wise using loops
4. Add the corresponding elements of the matrices using nested loops
5. Print the resultant matrix as console output
6. Stop

Flow Chart for Matrix Addition



Scilab Code for Matrix Addition

clc

m=input(”enter number of rows in the Matrices: ”);
n=input(”enter number of columns in the Matrices: ”);
disp(’enter the first Matrix’) for i=1:m for j=1:n

```
A(i,j)=input('\'); end end
disp('enter the second Matrix')
for i=1:m for j=1:n

B(i,j)=input('\');
end end for i=1:m
for j=1:n

C(i,j)=A(i,j)+B(i,j); end end
disp('The first matrix is')
disp(A)
disp('The Second matrix is')
disp(B)
disp('The sum of the two matrices is')
disp(C)
```

Matrix Addition using functions

```
// Save file as addition.sce

    clc

function [ ]=addition(m, n, A, B)

C=zeros(m,n);

C=A+B;

disp('The first matrix is')
disp ( A )
disp('The Second matrix is')
disp ( B )
disp('The sum of two matrices is')
disp ( C )
endfunction
```

1.2 Matrix Multiplication

- $A_{m\times n} \times B_{n\times p} = AB_{m\times p}$

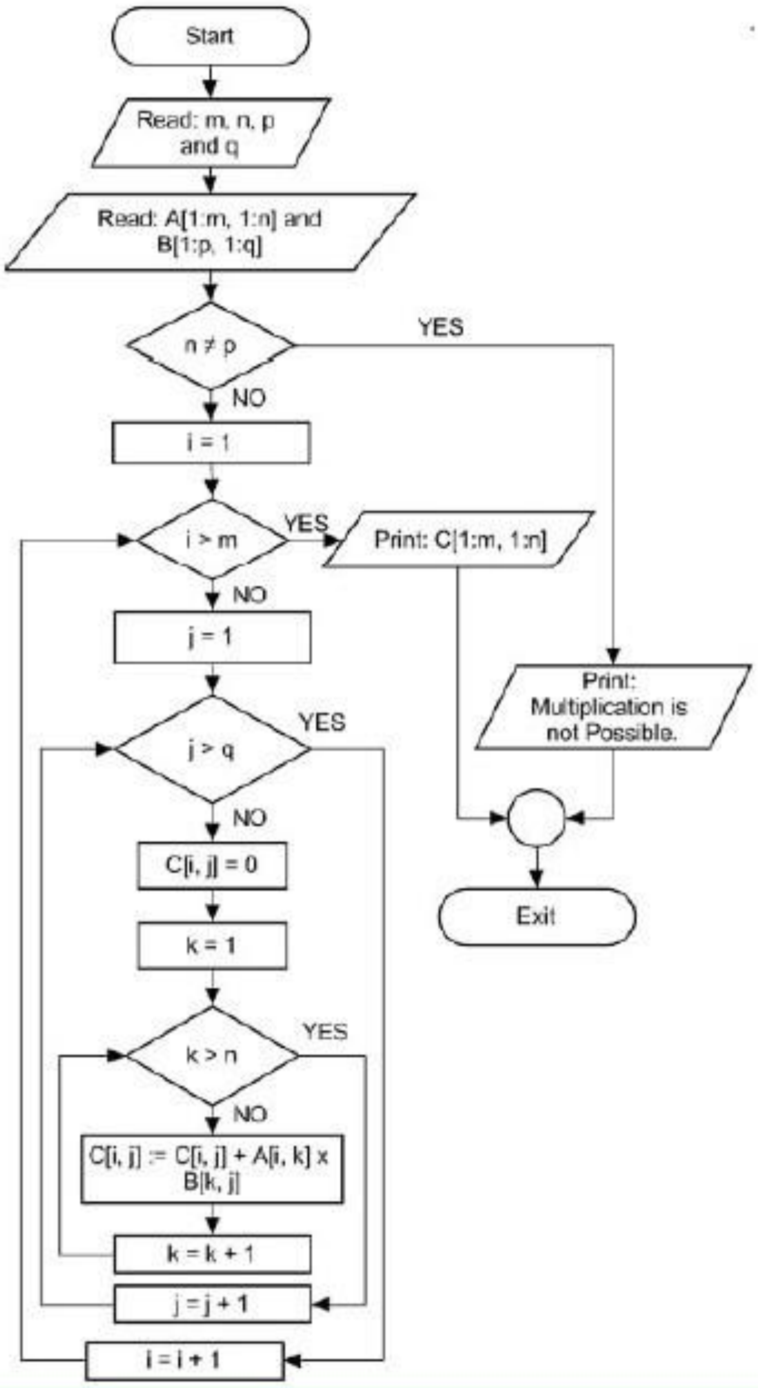
  Where $(AB)_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} + ... + A_{in}B_{nj}$

- The number of columns in the first matrix must be equal to the number of rows in the second matrix

- Matrix multiplication is not commutative **Matrix Multiplication Algorithm:**

1. Start
2. Declare variables and initialize necessary variables
3. Enter the elements of matrices row wise using loops
4. Check the number of rows and column of first and second matrices
5. If number of rows of first matrix is equal to the number of columns of second matrix, go to step 6. Otherwise, print 'Matrices are not conformable for multiplication' and go to step 3
6. Multiply the matrices using nested loops
7. Print the product in matrix form as console output
8. Stop

Flow Chart for Matrix Multiplication



Scilab Code for Matrix Multiplication

```
clc

m=input(”Enter number of rows in the first Matrix: ”);
n=input(”Enter number of columns in the first Matrix: ”);
p=input(”Enter number of rows in the second Matrix: ”);
```

```
q=input("Enter number of columns in the second Matrix: "); if
n==p

disp('Matrices are conformable for multiplication') else
disp('Matrices are not conformable for multiplication')
break;

end

disp('enter the first Matrix') for i=1:m for j=1:n

A(i,j)=input('\'); end end
disp('enter the second Matrix')
for i=1:p for j=1:q

B(i,j)=input('\');
end end
C=zeros(m,q);

for i=1:m for
j=1:q for k=1:n

C(i,j)=C(i,j)+A(i,k)*B(k,j);
end end
disp('The first matrix is')


disp(A)


disp('The Second matrix is')


disp(B)


disp('The product of the two matrices is')
disp(C)
```

Matrix Multiplication using functions

```
// Save file as multiplication.sce

    clc

function [ ] = multiplication(m, n, p, q, A, B)
C=zeros(m,n);
```

```
 if n==p
disp('Matrices are conformable for multiplication') else
disp('Matrices are not conformable for multiplication')
break;
 end
C=A*B
disp('The first matrix is')
disp ( A )
disp('The Second matrix is')
disp ( B )
disp('The multiplication of two matrices is')
disp ( C )
endfunction
```

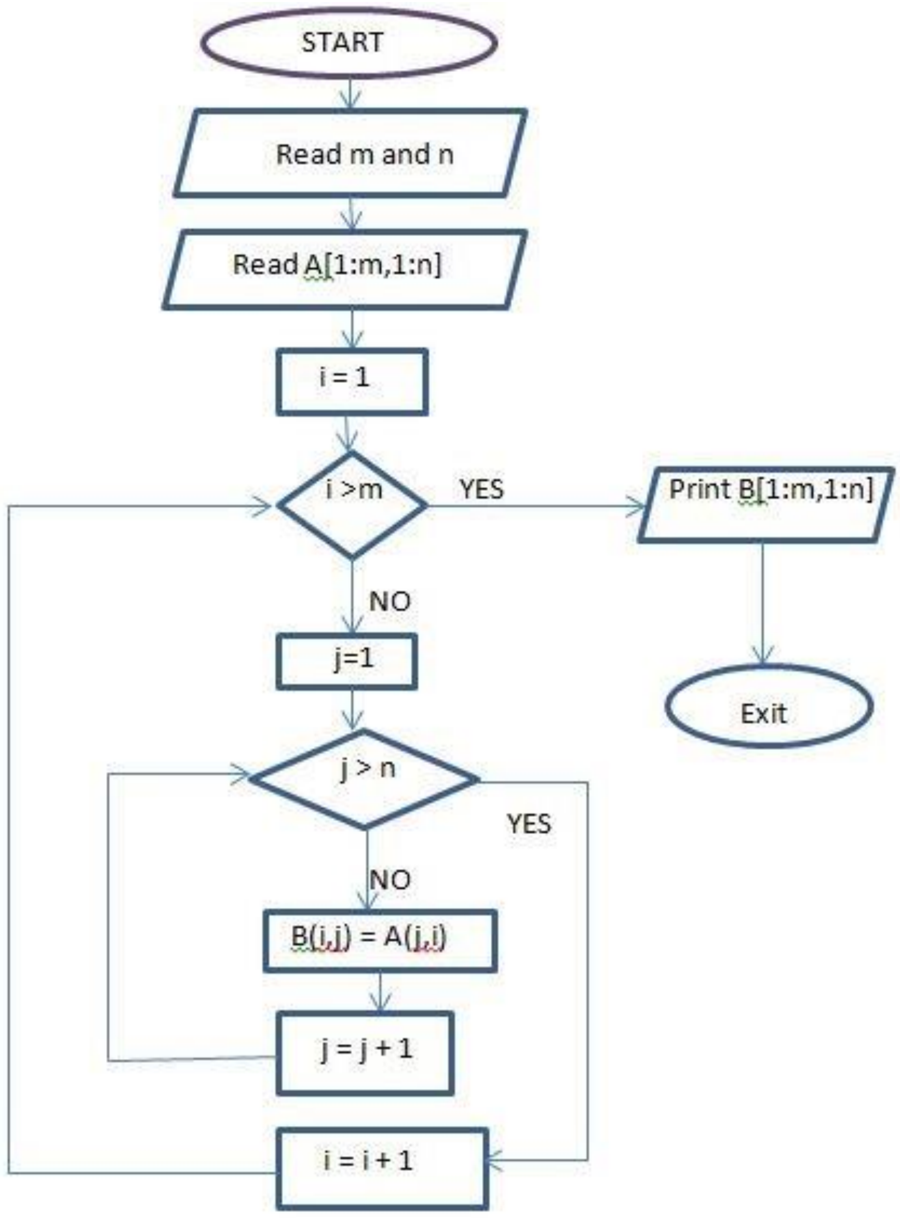
1.3 Matrix Transpose

- The transpose of an $m \times n$ matrix A is $n \times m$ matrix A^T
- Formed by interchanging rows into columns and vice versa
- $(AT\ )i,j = Aj,i$

Matrix Transpose Algorithm:

1. Start
2. Declare variables and initialize necessary variables
3. Enter the elements of matrix by row wise using loop
4. Interchange rows to columns using nested loops
5. Print the transposed matrix as console output
6. Stop

Flow Chart for Matrix Transpose



Scilab Code for Matrix Transpose

```
clc

m=input(”Enter number of rows in the Matrix: ”);
n=input(”Enter number of columns in the Matrix: ”);
disp(’Enter the Matrix’) for i=1:m for j=1:n

A(i,j)=input(’\’);
end end
B=zeros(n,m);
```

```
 for i=1:n for
j=1:m

B(i,j)=A(j,i) end end
disp('Entered matrix is')
disp(A)

 disp('Transposed matrix is')
disp(B)
```

Matrix Transpose using functions

```
 // Save file as transpose.sce function [
]=transpose(m, n, A)

B=zeros(m,n); B=A'

disp('The matrix is')
disp ( A)
 disp('Tansposed matrix is')
disp ( B )
 endfunction
```

Experiment No.2

Aim: Write a program to find factorial of a number using scilab.

Theory: The factorial function (symbol:)means to multiply a series of descending natural numbers. Examples:

- 4! =4x3x2x1=24
- 7! =7x6x5x4x3x2x1=5040
- 1! =1

Syntax: In scilab factorial can be calculated by using:

f=factorial (n)

f = factorial (n) returns the product of all positive integers less than or equal to n, where n is a nonnegative integer value. If n is an array, then f contains the factorial of each value of n. The data type and size of f is the same as that of n.

1) 10!
   f=factorial (10)

2) Factorial of an array element
   N= [0 1 2; 3 4 5]: f=factorial(n)

Code : In scilab factorial of a number can be calculated in

```
Clc
Clear
N=input('Enter the no.');
F=1
For i=1:n
F=f*I;
Disp(f)
```

Experiment No. 3

Aim: Write a program to find Fibonacci of a number using scilab.

Theory:

The sequence follows the rule that each number is equal to the sum of the preceding two numbers. The Fibonacci sequence begins with the following 14 integers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233 ...

Each number, starting with the third, adheres to the prescribed formula.

Code: In scilab fibonacci can be calculated by using:

```
Clc;
Clear;
a=0;
b=1;
n=input('Enter number of terms')
disp(a)
disp(b)
fori=1:n-2
c=a+b;
a=b;
b=c;
end
```

Experiment-4

Aim-To classify Odd & Even numbers upto a limit (Number).

Theory

Even Number-A number that is divisible by 2 and generates a remainder of 0 is called an even number**.**

Examples of even numbers are 2, 4, 6, 8, 10, etc. For example, assume you have ten chocolates. These chocolates may be divided into two groups, each having five chocolates. So, ten is an even number.

Odd Number-

An odd number is a number that is not divisible by 2. The remainder, in the case of an odd number is always “1”.

Syntex

```
clc;

disp(‘Name Roll No.’)

n=input(‘Enter the Number’)

disp(‘Even Number’);

for i=1:n

if modulo (i,2)==0

disp(i);

else

continue
```

```
end

end

disp('Odd numbers');

for i=1:n

if modulo(i,2)~=0

disp(i);

else

continue;

end

end
```

Experiment No. 5

Aim: Write a program to find the average of ten numbers using scilab.

Code:

```
function integer = UniformInt(a, b)
   integer =  min( floor( rand()*(b-a) ) + a , b);
endfunction

for i=1:5

   x(i) = UniformInt(1,10);

   if (x(i)<4) then
      cost(i) = 15*rand();

      elseif (4<=x(i) && x(i)<=8) then
      cost(i) = 27*rand();

      else
      cost(i) = 35*rand();

   end
```

Experiment-6

Aim- Fitting of linear regression line through given data set and testing of goodness of fit using mean error.

Fitting a Straight Line

Let $y = ax + b$ be the straight line to be fitted to the given set of data points

(x_1,y_1) , (x_2,y_2),···, (x_n,y_n), then normal equations are:

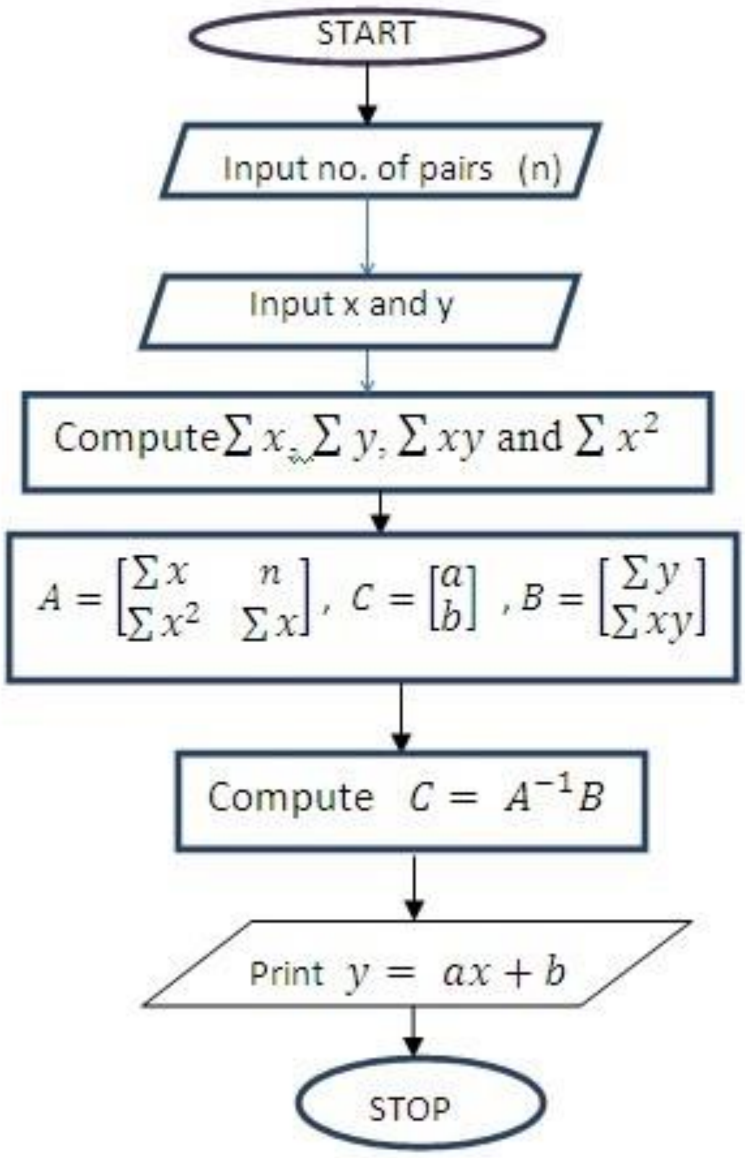
$P_y = aP_x + nb$ ··· (1)

$P_{xy} = aP_{x^2} + bP_x$ ··· (2)

Algorithm to fit a straight line to given set of data points

1. Start
2. Find number of pairs of data points (n) to be fitted
3. Input the x values and y values.
4. Find P_x , P_y, P_{x^2} and P_{xy}
5. Solve the system of equations given by (1) and (2) using matrix method, where
6. Required line of best fit is $y = ax + b$
7. Stop

Flow Chart for fitting a straight line to given set of data points //Scilab Code for fitting a straight line to given set of data points (x,y)


START
Input no. of pairs (n)
Input x and y
Compute $\sum x$, $\sum y$, $\sum xy$ and $\sum x^2$
$A = \begin{bmatrix} \sum x & n \\ \sum x^2 & \sum x \end{bmatrix}$, $C = \begin{bmatrix} a \\ b \end{bmatrix}$, $B = \begin{bmatrix} \sum y \\ \sum xy \end{bmatrix}$
Compute $C = A^{-1}B$
Print $y = ax + b$
STOP

Syntax

```
clc
n=input('Enter the number of terms:')
printf(' Enter the values of xi')
fori=1:n
x(i)=input('\');
end
printf(' Enter the values of yi')
fori=1:n
y(i)=input('\');
```

```
end

sumx=0;sumy=0;sumxy=0;sumx2=0;
fori=1:n

sumx=sumx+x(i);
sumx2=sumx2+x(i)*x(i);
sumy=sumy+y(i);
sumxy=sumxy+x(i)*y(i);

end
a=((sumx2*sumy-sumx*sumxy)*1.0/(n*sumx2-sumx*sumx)*1.0);
b=((n*sumxy-sumx*sumy)*1.0/(n*sumx2-sumx*sumx)*1.0);
printf('The line is Y=%3.3f +%3.3f X',a,b)
```

Experiment 7

Aim: Fitting of binomial distributions for given n and p using scilab.

Theory: In a binomial distribution, there is a summarization of the number of trials/observations when each occurrence has the same probability of achieving one particular value. That is it determines the probability of observing a particular number of successful outcomes in a specified number of trials.

The binomial distribution formula is for any random variable X, given by:

$P(x:n,p) = {}^nCx\ p^x\ (1-p)^{n-x}$ Or $P(x:n,p) = {}^nC_x\ p^x\ (q)^{n-x}$

where,

- n = the number of experiments
- x = 0, 1, 2, 3, 4, …
- p = Probability of success in a single experiment
- q = Probability of failure in a single experiment (= 1 – p)

The binomial distribution formula is also written in the form of n-Bernoulli trials, where ${}^nC_x = n!/x!(n-x)!$. Hence, $P(x:n,p) = n!/[x!(n-x)!].p^x.(q)^{n-x}$

Syntax: pr=binomial(p,n)

Code: n=10;p=0.3; clf(); plot2d3(0:n,binomial(p,n));

Experiment-8

Aim- Fitting of binomial distributions after computing mean and variance

Theory

A **binomial distribution** can be thought of as simply the probability of a SUCCESS or FAILURE outcome in an experiment or survey that is repeated multiple times. The binomial is a type of distribution that has **two possible outcomes** (the prefix "bi" means two, or twice). For example, a coin toss has only two possible outcomes: heads or tails and taking a test could have two possible outcomes: pass or fail.

- The first variable in the binomial formula, n, stands for the number of times the experiment runs.
- The second variable, p, represents the probability of one specific outcome.

For example, let's suppose you wanted to know the probability of getting a 1 on a die roll. if you were to roll a die 20 times, the probability of rolling a one on any throw is 1/6. Roll twenty times and you have a binomial distribution of (n=20, p=1/6). SUCCESS would be "roll a one" and FAILURE would be "roll anything else." If the outcome in question was the probability of the die landing on an even number, the binomial distribution would then become (n=20, p=1/2). That's because your probability of throwing an even number is one half.

Binomial distributions must also meet the following three criteria:

1. **The number of observations or trials is fixed.** In other words, you can only figure out the probability of something happening if you do it a certain number of times. This is common sense—if you toss a coin once, your probability of getting a tails is 50%. If you toss a coin a 20 times, your probability of getting a tails is very, very close to 100%.
2. **Each observation or trial is** independent. In other words, none of your trials have an effect on the probability of the next trial.
3. The **probability of success** (tails, heads, fail or pass) is **exactly the same** from one trial to another.

The binomial distribution formula is:

$$\mathbf{b(x; n, P) = {}_nC_x * P^x * (1 - P)^{n-x}}$$

Where:
b = binomial probability
x = total number of "successes" (pass or fail, heads or tails etc.)
P = probability of a success on an individual trial
n = number of trials

Note: The binomial distribution formula can also be written in a slightly different way, because ${}_nC_x = n! / x!(n - x)!$ (this binomial distribution formula uses factorials (What is a factorial?). “q” in this formula is just the probability of failure (subtract your probability of success from 1).

$$P(X) = \frac{n!}{(n-X)!\,X!} \cdot (p)^X \cdot (q)^{n-X}$$

Syntax

```
n=50;p=0.4;
mea=n*p; sigma=sqrt(n*p*(1-p));
x=( (0:n)-mea )/sigma;
clf()
plot2d(x, sigma*binomial(p,n));
deff('y=Gauss(x)','y=1/sqrt(2*%pi)*exp(-(x.^2)/2)')
plot2d(x, Gauss(x), style=2);
```

Experiment 9

Aim: Fitting of poission distributions for given lambda using scilab.

Theory:

Poisson Distribution:

The Poisson distribution is a discrete probability function that means the variable can only take specific values in a given list of numbers, probably infinite. A Poisson distribution measures how many times an event is likely to occur within “x” period of time. In other words, we can define it as the probability distribution that results from the Poisson experiment. A Poisson experiment is a statistical experiment that classifies the experiment into two categories, such as success or failure. Poisson distribution is a limiting process of the binomial distribution.

A Poisson random variable “x” defines the number of successes in the experiment. This distribution occurs when there are events that do not occur as the outcomes of a definite number of outcomes. Poisson distribution is used under certain conditions. They are:

- The number of trials “n” tends to infinity
- Probability of success “p” tends to zero
- np = 1 is finite

Poisson Distribution Formula

- The formula for the Poisson distribution function is given by:
- **$f(x) =(e^{-\lambda} \lambda^x)/x!$**
- Where,
- e is the base of the logarithm
- x is a Poisson random variable
- λ is an average rate of value

code:

```
 clear ;
  clc ;
close ;
  Frequency =[12 ,10 ,19 ,17 ,10 ,8 ,7 ,5 ,5 ,3 ,3 ,1]
  n = sum (Frequency )
 X =[0 ,1 ,2 ,3 ,4 ,5 ,6 ,7 ,8 ,9 ,10 ,11]
 F =[]

 P1 =[]
 P2=[]
mprintf (” −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−\n”)
 mprintf (” A r r i v a l s F r e q u e n c y \n”)
 mprintf (” −−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−−”)
  for i =1:12
  mprintf (”\n %d %d” ,i -1 , Frequency ( i ) )
F ($ +1) = Frequency ( i )
 product1 = F ( i ) * X (i )
```