## Experiment No. 1

**Objective:** Introduction to SCILAB and implementation of basic operations in Scilab.

This chapter is intended to get the user started using SCILAB through simple exercises in numerical calculations. The chapter starts by describing how to download and install SCILAB in a Windows environment. Installation of the software in other operating systems is very similar and is explained in detail in the SCILAB website.

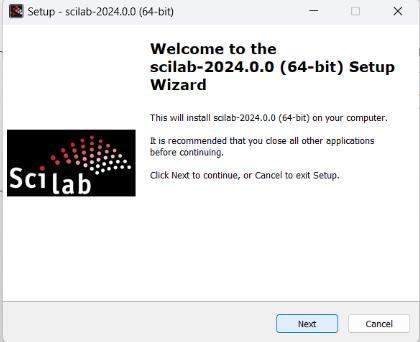
## What is Scilab?

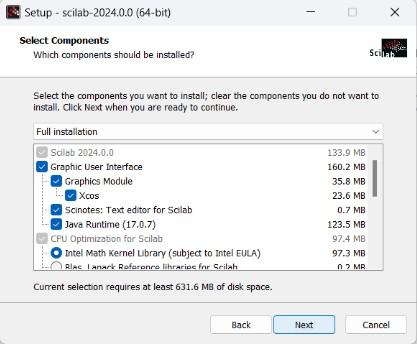
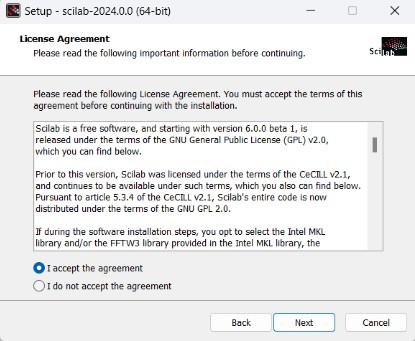
Scilab is a free and open-source software for engineers & scientists, with a long history (first release in 1994) and a growing community. Scilab is mainly developed by the Scilab team within ESI Group SCILAB is a numerical, programming, and graphics environment available for free from the French Government's "Institut Nationale de Recherche en Informatique et en Automatique - INRIA (National Institute for Informatics and Automation Research)." It is similar in operation to MATLAB and other existing numerical/graphic environments, and it can be run using a variety of operating systems including UNIX, Windows, Linux, etc. SCILAB is a self-contained package including a large number of intrinsic numeric, programming and graphics functions. Once unpacked and installed in your computer it will consume about 50 MB of your hard disk. Make sure you have at least that much memory in your hard disk before downloading and installing SCILAB.

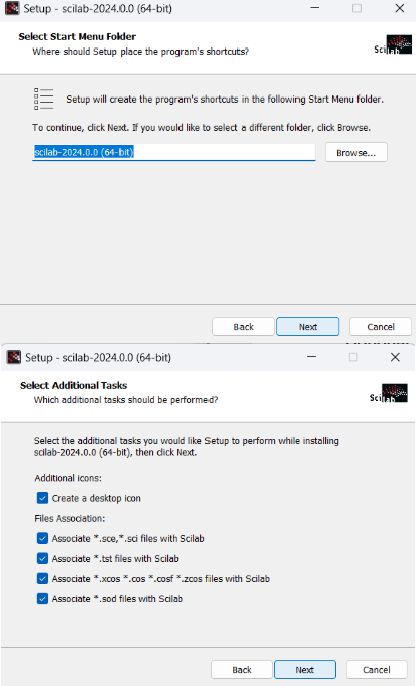
Website link for downloading latest version of scilab: <https://www.scilab.org/download/scilab-2024.0.0>. Scilab 2024.0.0 is released under the terms of the [GNU General Public License (GPL) v2.0](https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html).

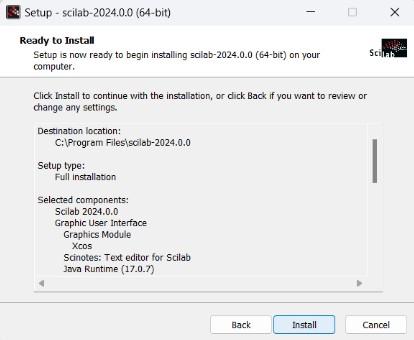
## How to install SCILAB?

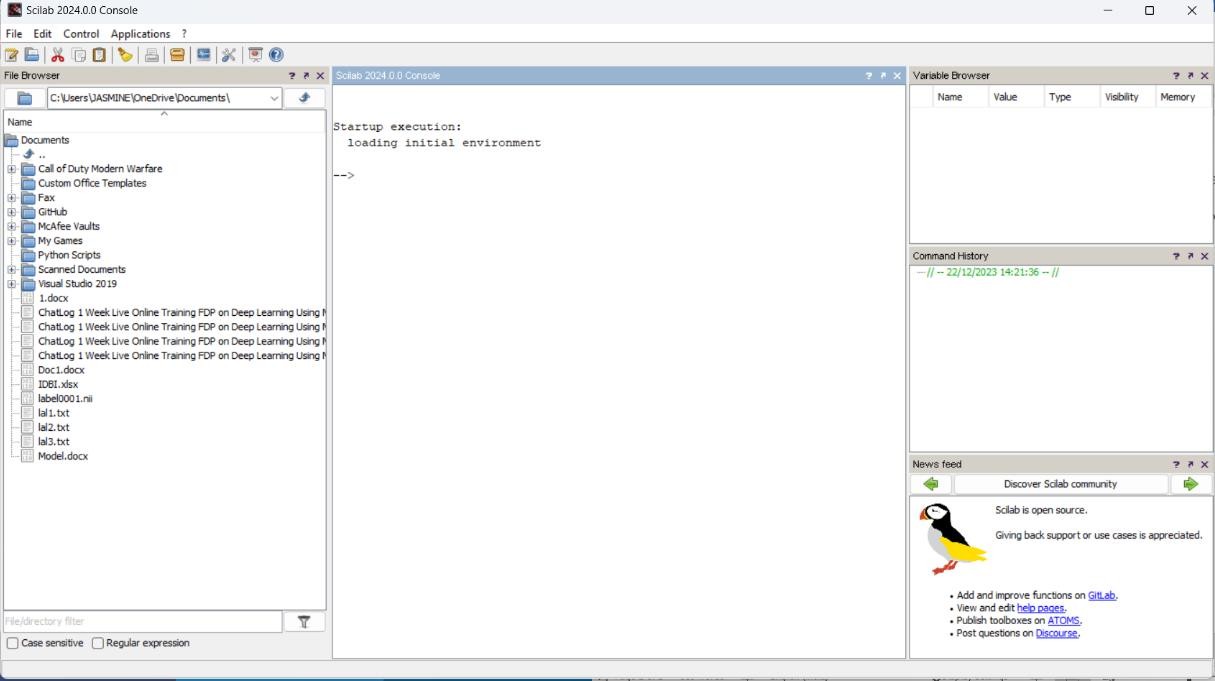
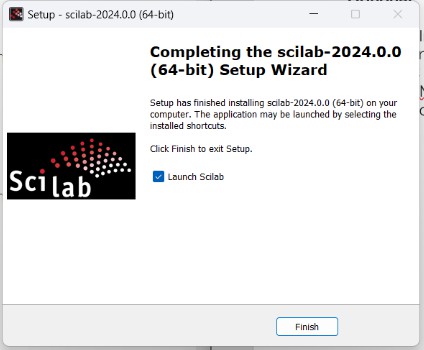












**Getting started with Scilab**

1. Launch Scilab: Locate the Scilab icon on your computer and double-click to open it. You should see a window with three panels: the Console, the History, and the Workspace.
2. Basic Calculations:
   * Enter simple arithmetic expressions in the Console, like 2 + 3, and press Enter. Scilab will show the result (5) below the expression.
   * Try other calculations, including \*, /, -, ^ (exponentiation), and sqrt (square root).
3. Variables and Data Types:
   * Assign a value to a variable using the = symbol. For example, type a = 10 and press Enter. Now, a stores the value 10.
   * Check the variable's value in the Workspace panel. You can also type a in the Console to see its value.
   * Scilab has different data types, such as integers, floats (decimal numbers), and strings. Experiment with assigning different types of values to variables.
4. Vectors and Matrices:
   * Create a vector (one-dimensional array) by enclosing elements in square brackets, separated by commas. For example, type v = [1, 2, 3, 4] to create a vector named v with four elements.
   * Access individual elements using their index (starting from 1). For example, type v(2) to see the second element (2).
   * Create a matrix (two-dimensional array) by nesting vectors within square brackets. For example, type m = [[1, 2], [3, 4]] to create a 2x2 matrix named m.
   * Access elements in a matrix using row and column indices. For example, type m(1, 2) to see the element at row 1, column 2 (4).
5. Plotting:
   * Scilab allows you to visualize data using various plot types.
   * To plot a simple line graph, use the plot function. For example, type plot([1, 2, 3], [4, 5, 6]) to plot a line with three points.
   * Experiment with different plot types like bar charts, scatter plots, and pie charts. You can find the syntax for each type in the Scilab documentation.
6. Help and Documentation:
   * Scilab provides extensive help documentation for its functions and commands.
   * To access help for a specific function, type help function\_name in the Console. For example, help plot will show you information about the plot function.
   * You can also access the online Scilab documentation for more detailed information: <https://gitlab.com/scilab>

Let us start by defining a vector.

This can be done in two ways: first one is using spaces

-->p = [1 2 3]

p = 1. 2. 3.

or using commas.

-->q = [2, 3, 4]

q = 2. 3. 4.

We can find the length of a vector as follows:

-->length(p) ans = 3.

We can work with vectors, just as we do in Maths classes. We can add them:

-->p + q

ans = 3. 5. 7. We can subtract

-->q - p

ans = 1. 1. 1. and so on.

## Program:

## 

## 

## 

## 

## Conclusion:

This experiment provided a basic introduction to Scilab's functionalities and capabilities. By continuing to explore and practice, you can unlock the potential of this powerful tool for solving engineering problems and analyzing data.

## Experiment No. 2

**Objective:** Exercises to implement the basic matrix operations in Scilab.

## Equipment:

* + Computer with Scilab installed

## Procedure:

1. Launch Scilab: Open the Scilab software.
2. Creating Matrices:
   * Use the zeros() function to create a 3x4 matrix of zeros. Assign it to a variable named A.
   * Use the ones() function to create a 2x3 matrix of ones. Assign it to a variable named B.
   * Use manual entry to create a 4x2 matrix with the following elements:

o [1 5; 2 6; 3 7; 4 8]

Assign it to a variable named C.

1. Accessing Elements:
   * Display the element at the second row, third column of matrix A.
   * Display the first row of matrix B.
   * Display the last column of matrix C.
2. Matrix Addition and Subtraction:
   * Perform element-wise addition of matrices A and C. Assign the result to a variable named D.
   * Perform element-wise subtraction of matrix B from matrix C. Assign the result to a variable named E.
3. Matrix Multiplication:
   * Perform matrix multiplication of A and C. Assign the result to a variable named F.
4. Matrix Transpose:
   * Find the transpose of matrix B. Assign the result to a variable named G.
5. Matrix Inverse:
   * Find the inverse of matrix C. Assign the result to a variable named H.

## Program:

## 

## 

## 

## 

## Conclusion:

Through these exercises, you've gained practical experience with matrix operations in Scilab, laying a solid foundation for further exploration and application in various engineering contexts.

## Experiment No. 3

**Objective:** Exercises to find the Eigenvalues and eigenvectors in Scilab.

## Equipment:

* + A computer with Scilab installed.

## Procedure:

1. Basic Matrix Operations:
   * Familiarize yourself with basic matrix creation and manipulation in Scilab. Create a simple matrix by enclosing elements in square brackets, separated by commas. For example, type A = [[1, 2], [3, 4]] to create a 2x2 matrix named A.
   * Access individual elements using row and column indices. For example, type A(1, 2) to see the element at row 1, column 2 (4).
   * Practice basic operations like matrix addition, subtraction, multiplication, and transposition.
2. Finding Eigenvalues:
   * Scilab provides the built-in function eig to find the eigenvalues of a matrix. For example, type eig(A) to get a vector containing the eigenvalues of matrix A.
   * Explore the obtained eigenvalues and understand their relationship to the matrix.
   * Experiment with different types of matrices, including symmetric, triangular, and diagonal, and observe how their eigenvalues are affected.
3. Finding Eigenvectors:
   * Use the eigs function to find the eigenvectors corresponding to the eigenvalues. For example, type eigs(A) to get a matrix containing the eigenvectors as columns.
   * Each column in the obtained matrix corresponds to an eigenvector associated with a corresponding eigenvalue.
   * Verify the relationship between eigenvectors and eigenvalues using the equation A\*v = lambda\*v, where v is an eigenvector and lambda is its corresponding eigenvalue.
4. Applications of Eigenvalues and Eigenvectors:
   * Research and discuss some real-world engineering applications of eigenvalues and eigenvectors, such as modal analysis of vibrations, principal component analysis (PCA) for data compression, and image processing.
   * Consider how these concepts can be applied in your specific engineering discipline.

## Conclusion:

This experiment provided a fundamental understanding of finding eigenvalues and eigenvectors in Scilab. By continuing to explore these concepts and their applications, you'll unlock powerful tools for solving complex engineering problems involving matrix analysis and data interpretation.

## Experiment No. 4(a)

**Objective:** Exercises to solve equations by Gauss elimination in Scilab.

## Equipment:

* + A computer with Scilab installed.

## Procedure:

**Gauss Elimination in Scilab**

1. Formulate a system of linear equations with a 3x3 coefficient matrix and a 3x1 constant matrix.
2. Write a Scilab script to implement Gauss elimination for solving the system.
3. Execute the script and display the intermediate steps, including the transformed augmented matrix and the final solution vector.

## Program and Output:

function [x]=gaussElimination(A, b)

n = length(b)

disp([A, b])

A1 = A

b1 = b

for i=1:n-1

m1 = A(i,i)

A(i,:) = A(i,:)/m1

b(i) = b(i)/m1

for j=i+1:n

m2 = A(j,i)

A(j,:) = A(j,:)-m2\*A(i,:)

b(j) = b(j)-m2\*b(i)

end

disp([A,b])

end

x = zeros(n,1)

for i=n:-1:1

s = 0

for j=i+1:n

s = s + A(i,j)\*x(j)

end

x(i,1) = (b(i) - s)/A(i,i)

end

disp(x)

disp([A1\*x])

end

*//% Example usage:*

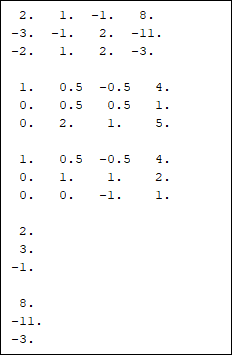
*//% Define the coefficient matrix A and constant vector b*

A = [2 1 -1; -3 -1 2; -2 1 2];

b = [8; -11; -3];

*//% Solve the system using Gauss elimination*

gaussElimination(A, b);



## Experiment No. 4(b)

## Objective: Exercises to solve equations by Gauss Jordan Method in Scilab.

## Equipment:

* + A computer with Scilab installed.

## Procedure:

## Gauss-jordan method in Scilab

1. Create a new system of linear equations with a 4x4 coefficient matrix and a 4x1 constant matrix.
2. Develop a Scilab script to apply the Gauss-Jordan method to solve the system.
3. Run the script and output the reduced row-echelon form and the solution vector.

## Program and Output:

## function [x] = gaussJordan(A, b)

## n = length(b)

## //% Augmented matrix [A|b]

## Ab = [A, b]

## //% Forward elimination

## for k = 1:n

## //% Pivot scaling

## pivot = Ab(k,k);

## Ab(k,:) = Ab(k,:) / pivot

## //% Elimination

## for i = 1:n

## if i ~= k

## factor = Ab(i,k)

## Ab(i,:) = Ab(i,:) - factor \* Ab(k,:)

## end

## end

## end

## //% Extract solution vector

## x = Ab(:, n+1);

## end

## //% Example usage for a 4x4 system

## A = [2 1 3 4; -1 3 2 1; 4 2 1 5; 3 -2 4 6];

## b = [10; 13; 12; 31];

## //% Solve the system using Gauss-Jordan method

## solution = gaussJordan(A, b);

## disp("Solution using Gauss-Jordan method:"); disp(solution);

## 

## Experiment No. 4(c)

## Objective: Exercises to solve equations by Gauss Siedel Method in Scilab.

## Equipment:

* + A computer with Scilab installed.

## Procedure:

## Gauss-Seidel Method in Scilab

1. Define a system of linear equations with a 3x3 coefficient matrix and a 3x1 constant matrix.
2. Implement a Scilab script for the Gauss-Seidel iterative method.
3. Execute the script, providing an initial guess for the solution vector, and observe the convergence of the method to the solution

**Program and Outputs:**

function [x, iter] = gaussSeidel(A, b, initial\_guess, tol, max\_iter)

n = length(b)

x = initial\_guess

iter = 0

while iter < max\_iter

x\_old = x

for i = 1:n

sum1 = 0

for j = 1:n

if i ~= j

sum1 = sum1 + A(i,j)\*x\_old(j)

end

end

x(i) = (b(i) - sum1) / A(i, i)

end

//% Check for convergence

if norm(x - x\_old, 'inf') < tol

break

end

iter = iter + 1

end

end

//% Example usage for a 3x3 system

A = [4 -1 0; -1 4 -1; 0 -1 4]

b = [15; 10; 10]

initial\_guess = [0; 0; 0]

tolerance = 1e-6

max\_iterations = 1000

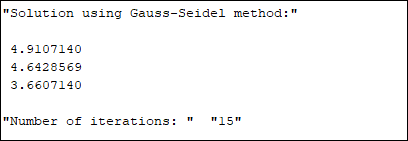
//% Solve the system using Gauss-Seidel method

[solution, iterations] = gaussSeidel(A, b, initial\_guess, tolerance, max\_iterations)

disp('Solution using Gauss-Seidel method:')

disp(solution)

disp(['Number of iterations: ', string(iterations)])



**Conclusion:**

Reflect on the exercises, discuss the strengths and limitations of each method, and consider the practical implications of using these techniques for solving linear systems. This set of exercises will provide hands-on experience in implementing and comparing different methods for solving linear equations using Scilab.

## Experiment No.-5

**Objective:** Exercises to find the reduced row echelon form of a matrix in Scilab.

## Equipment:

* + A computer with Scilab installed.

## Procedure:

Reduced Row Echelon Form of a matrix is used to find the rank of a matrix and further allows solving a system of linear equations. A matrix is in Row Echelon form if

* + All rows consisting of only zeroes are at the bottom.
  + The first nonzero element of a nonzero row is always strictly to the right of the first nonzero element of the row above it.

A matrix can have several row echelon forms. A matrix is in Reduced Row Echelon Form if

* + It is in row echelon form.
  + The first nonzero element in each nonzero row is a 1.
  + Each column containing a nonzero as 1 has zeros in all its other entries.

## Rank of matrix

The rank of the matrix is the number of non-zero rows in the row echelon form. To find the rank, we need to perform the following steps:

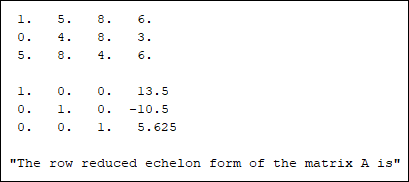
* Find the row-echelon form of the given matrix
* Count the number of non-zero rows.

## # Code

A=[1 5 8 6; 0 4 8 3;5 8 4 6];

disp(A)

disp(rref(A),"The row reduced echelon form of the matrix A is");



**Conclusion :** This experiment provided a fundamental understanding of finding reduced row echelon form and rank of matrix in Scilab. It can also be used as a way of finding a solution to the system of linear equations.

## Experiment No.-6

**Objective:** Exercises to plot the functions and solve first order and second order differential equation in Scilab.

## Equipment:

* A computer with Scilab installed.

**Procedure:** plot command is used to create plots in the plane. Color and appearance can be specified by putting indications of color and points style within quotes:

* Colors

"b" = blue (by default), "k" = black, "r" = red, "g" = green, "c" = cyan, "m" = magenta, "y" = yellow, "w" = white

Point styles Joined (by default), or ".", "+", "o", "x", "\*". Other options are available with: "s", "d", "v", "<", and ">".

To create a 2-D x/y plot,

* + create a vector of X values
  + create a vector of Y values (which must have the same number of elements)
  + type **plot (x, y)**

For example, try these commands: x = 0 : 0.1 : 30;

y = sin(x);

plot(x, y);

If you call **plot** a second time, it will sit on top of the first plot: y = sqrt(x);

plot(x, y);

If you want to erase the old plot and start from scratch, and add to it the new one on the same set of axes, use the **clf** command before making the second plot:

clf;

y = sin(x);

plot(x, y);

There's another way to do it, too: you can supply more than a single pair of (x, y) values to the **plot** function:

x = 1 : 0.1: 30;

y1 = sin(x); y2 = cos(x); y3 = sqrt(x);

plot(x, y1, x, y2, x, y3);

Subplot in sci lab clc

clear

x=-10:1:10

y1=x.\*x y2=-x.\*x

subplot(2,1,1) plot(x,y1) subplot(2,1,2) plot(x,y2) xlabel(“x-axis”)

ylabel(“y-axis”)

title(“2-D-plot”)

//axes properties a=gca() a.xlabel.font\_size=4 y.xlabel.font\_size=4 a.title.font\_size=4

**Source code:**

## //First Order Differential Equation

**//Solve first order differential equation**

clear;

clc;

clf;

function[yp]=f(t,y);

yp=t\*t+1;

endfunction

t0=0;

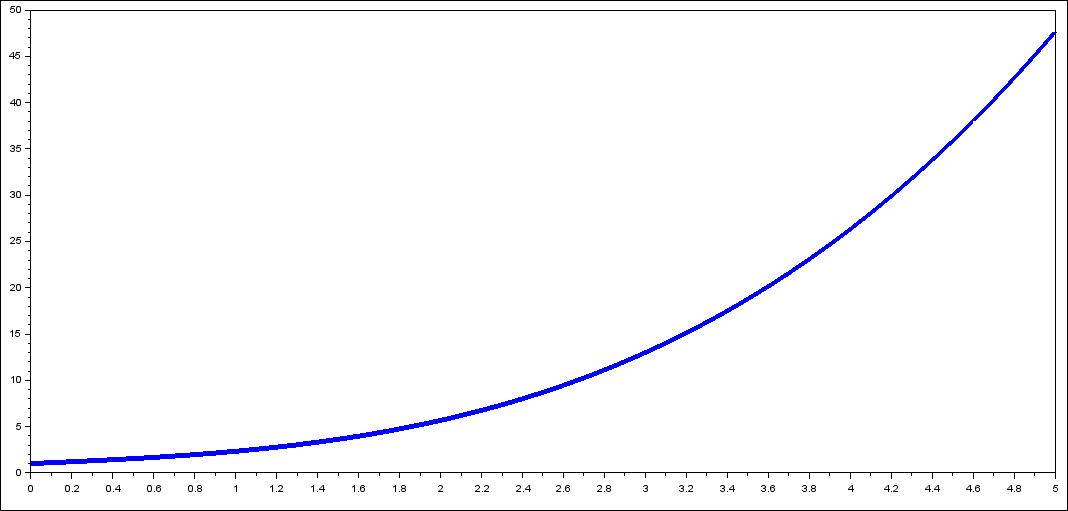
y0=1;

t=0:0.1:5;

y=ode(y0,t0,t,f);

plot(t,y,'Linewidth',5);

**Output:**



## Source Code:

## //Second Order Differential Equation

clear;

clc;

clf;

function[xp]=f(t,x)

xp(1)=x(2);

xp(2)=-3\*x(2)+10\*x(1);

endfunction

t0=1;

y0=1;

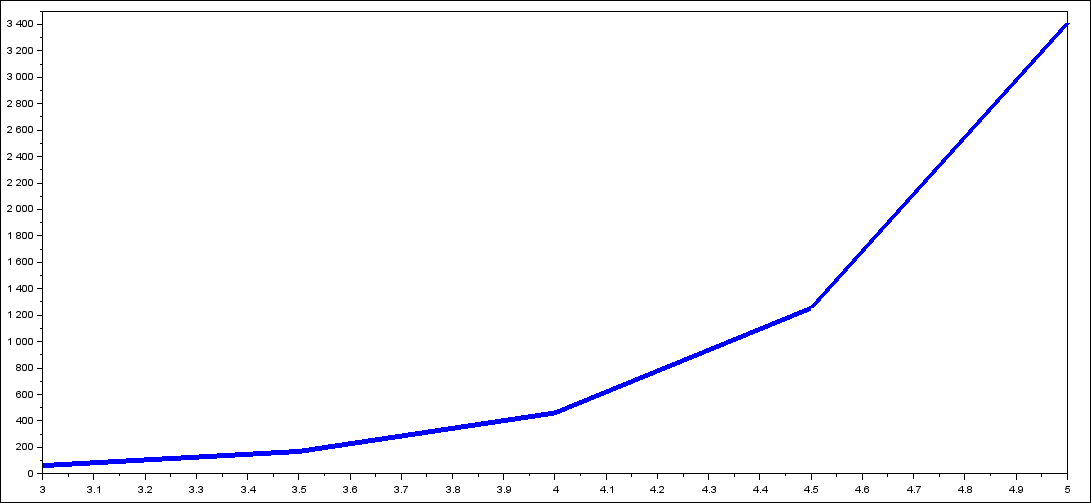
yp0=3;

t=3:0.5:5;

y=ode([y0;yp0],t0,t,f);

plot(t,y(1,:),'Linewidth',5);

**Output:**



**Conclusion:** This experiment provided a fundamental understanding of plot of function in Scilab. It also gives solution of first and second order differential equation.

## Experiment 7

**Objective:** Introduction to R Programming and implementation of basic operations in R.

This exercise is intended to get the user started using R through simple exercises in numerical calculations. The chapter starts by describing how to download and install R in a Windows environment. Installation of the software in other operating systems is very similar and is explained in detail in the R website.

## Introduction to R programming

R is a programming language and free software developed by Ross Ihaka and Robert Gentleman in 1993. R possesses an extensive catalog of statistical and graphical methods. It includes machine learning algorithms, linear regression, time series, statistical inference to name a few. Most of the R libraries are written in R, but for heavy computational tasks, C, C++ and Fortran codes are preferred. R is not only entrusted by academic, but many large companies also use R programming language, including Uber, Google, Airbnb, Facebook and so on.

Data analysis with R is done in a series of steps; programming, transforming, discovering, modelling and communicate the results.

**Program:** R is a clear and accessible programming tool

**Transform:** R is made up of a collection of libraries designed specifically for data science

**Discover:** Investigate the data, refine your hypothesis and analyse them

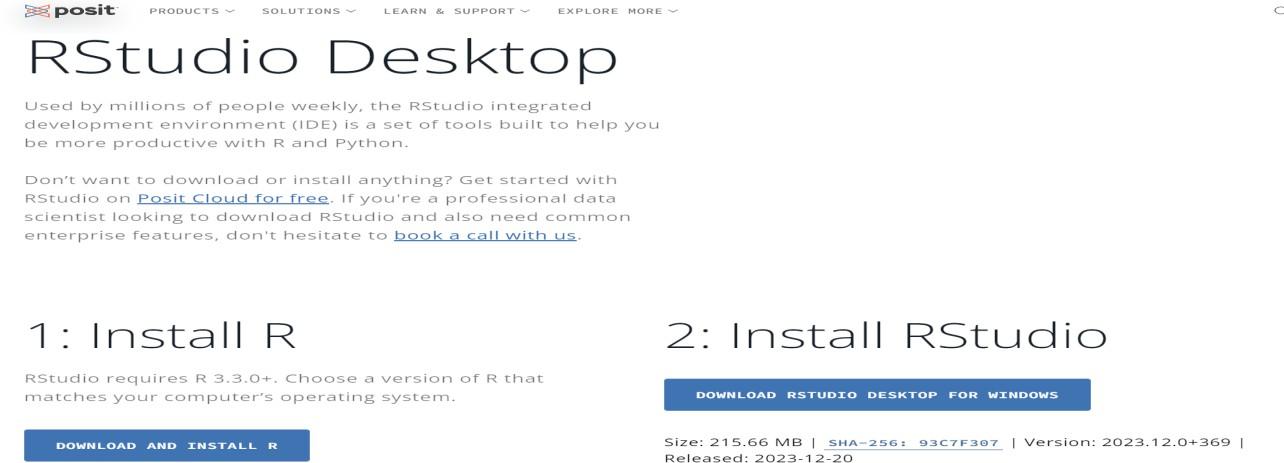
**Model:** R provides a wide array of tools to capture the right model for your data **Communicate:** Integrate codes, graphs, and outputs to a report with R Markdown or build Shiny apps to share with the world

What is R used for?

1. Statistical inference
2. Data analysis
3. Machine learning algorithm

## Installation of R-Studio on windows:

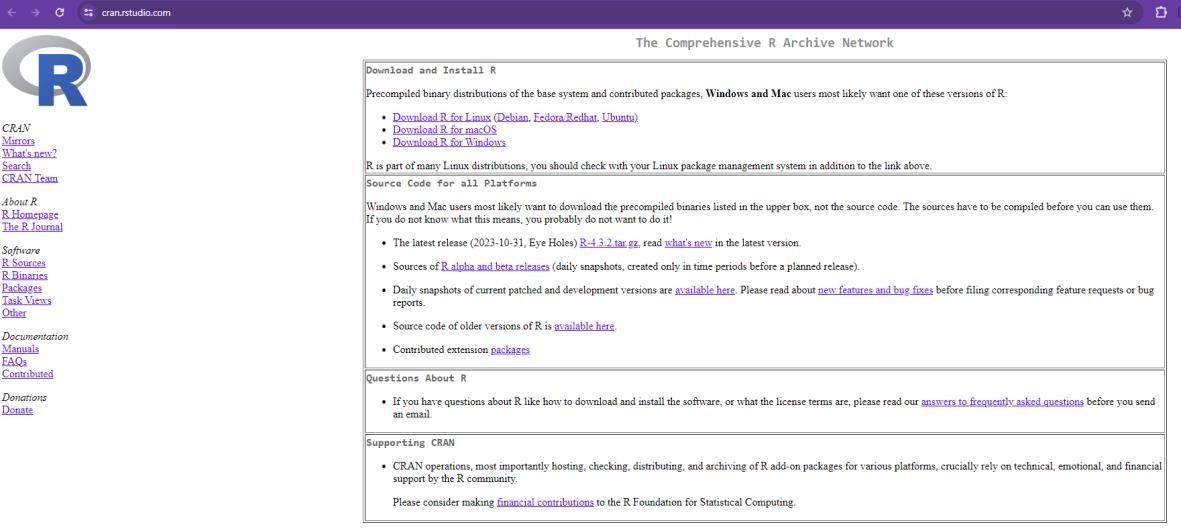
Step 1: [Download RStudio](https://posit.co/download/rstudio-desktop/) (<https://posit.co/download/rstudio-desktop/>) using the following steps



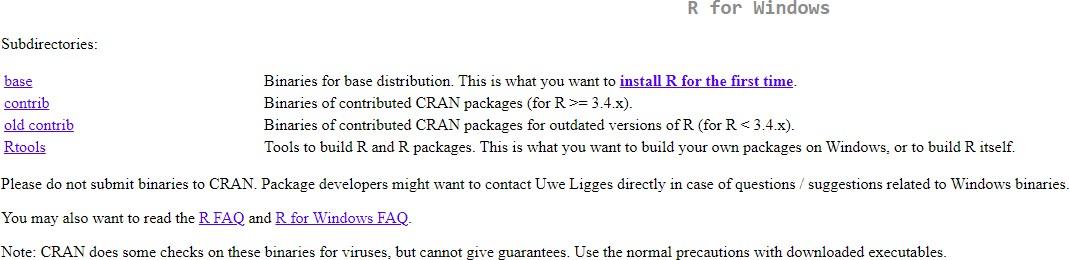
## 1: Install R

Step 1: Click on download and install

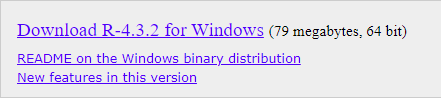
Step 2: Download R for windows / macOS / Linux



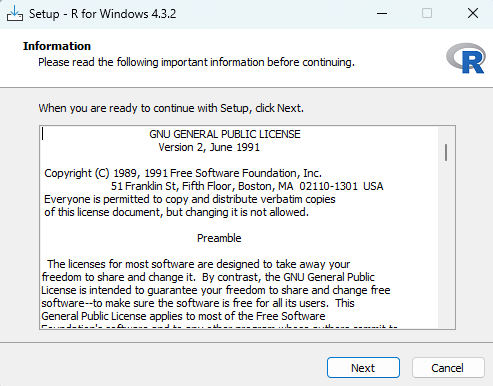
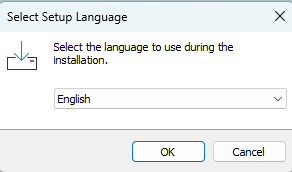
Step 3: Click install R (base) for the first time



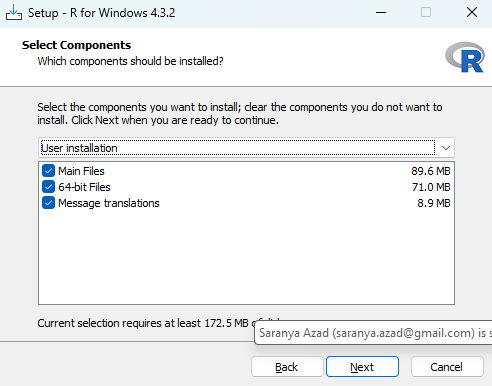
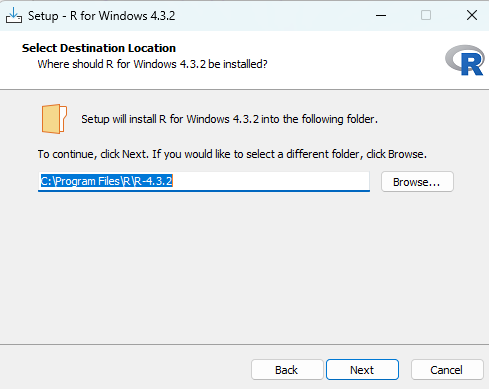
Step 4: Download R-4.3.2 and click on R-4.3.2-win.exe in the download folder



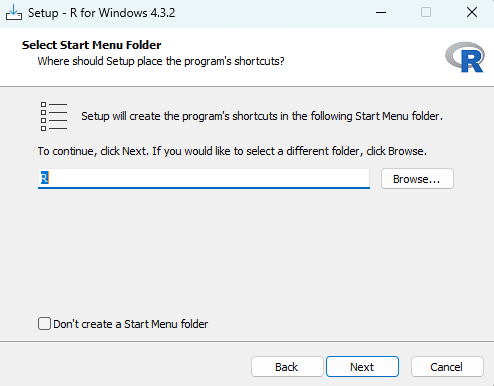
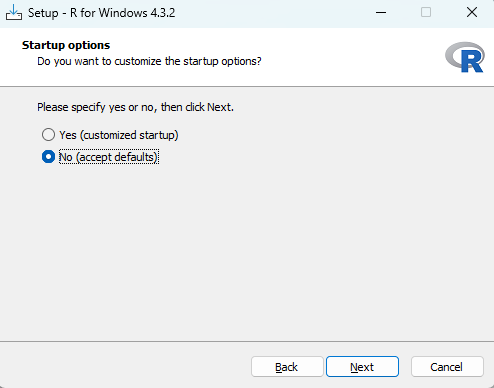
Step 5: Click Ok Step 6: Click Next



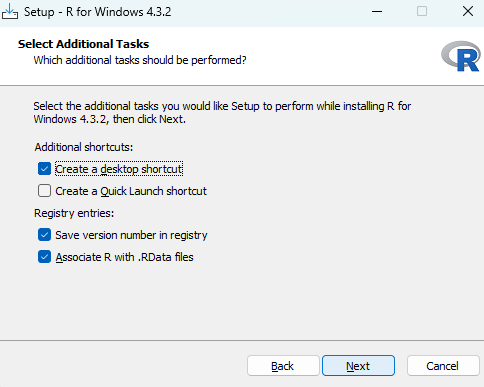
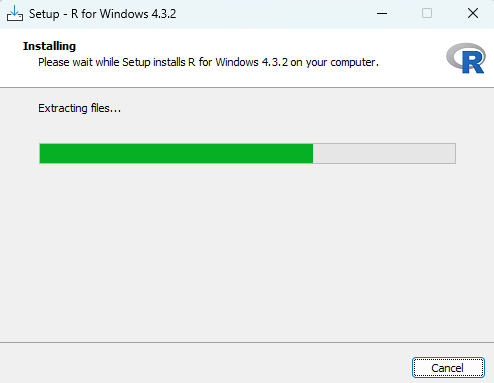
Step 6: Select Destination Location Step 7: Select Components



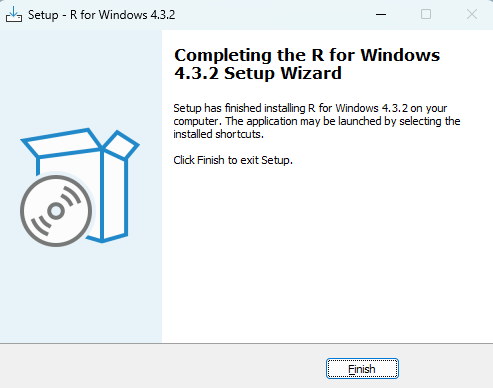
Step 8: Choose No (accept defaults) Step 9: Select Start Menu Folder



Step 10: Select Additional Tasks Step 11: Installation will begin

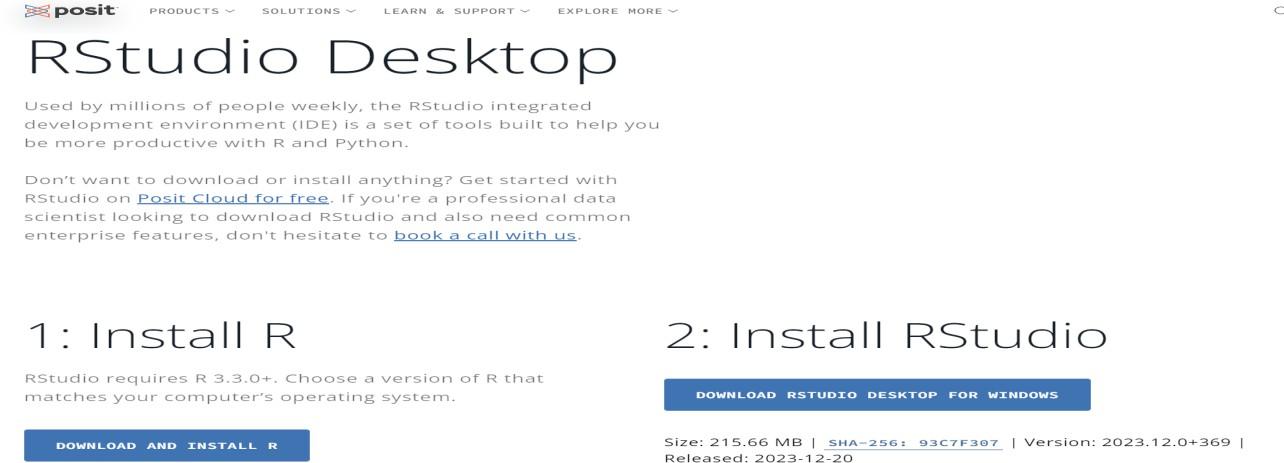
 

Step 12: Click Finish to complete the installation

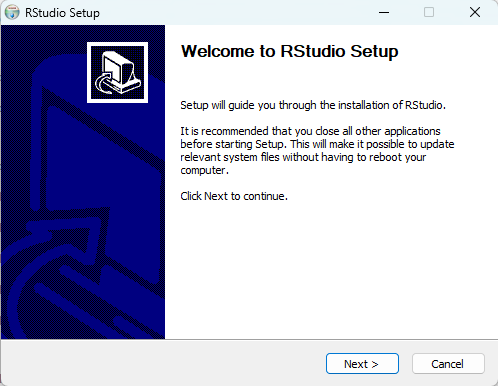


## 2: Install R Studio

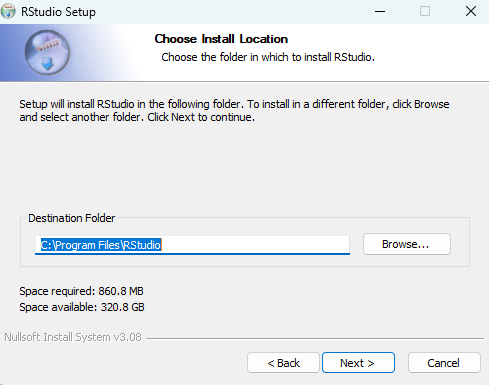
Step 1: Click on download RStudio Desktop for Windows



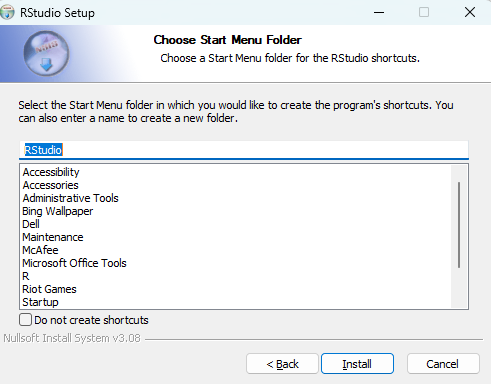
Step 2: Click RStudio-2023.12.0-369 exe file and click Next on the welcome window.



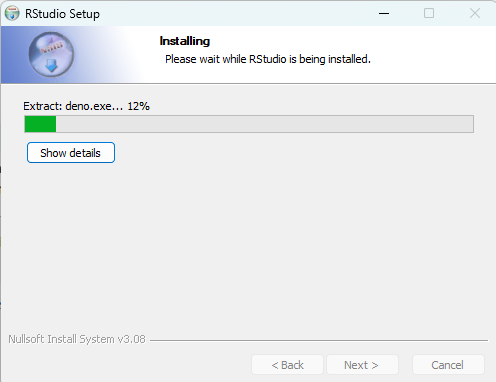
Step 3: Enter/browse the path to the installation folder and click Next to proceed.



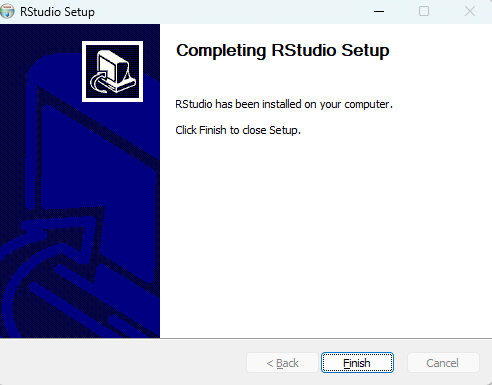
Step 4: Select the folder for the start menu shortcut or click on do not create shortcuts and then click Next.



Step 5: Wait for the installation process to complete.



Step 6: Finish the installation



## Getting started with Scilab Installing Packages:-

The most common place to get packages from is CRAN. To install packages from CRAN you use install.packages("package name"). For instance, if you want to install the ggplot2 package, which is a very popular visualization package, you would type the following in the console:- Syntax:-

# install package from CRAN install.packages("ggplot2")

## Loading Packages:-

Once the package is downloaded to your computer you can access the functions and resources provided by the package in two different ways:

# load the package to use in the current R session library (packagename)

## Getting Help on Packages:-

For more direct help on packages that are installed on your computer you can use the help and vignette functions. Here we can get help on the ggplot2 package with the following: help(package = "ggplot2") # provides details regarding contents of a package vignette(package

= "ggplot2") # list vignettes available for a specific package vignette("ggplot2-specs") # view specific vignette

vignette() # view all vignettes on your computer

## Assignment Operators:-

The first operator you’ll run into is the assignment operator. The assignment operator is used to assign a value. For instance we can assign the value 3 to the variable x using the <- assignment operator.

# assignment x <- 3 or x=3

## Evaluation

We can then evaluate the variable by simply typing x at the command line which will return the value of x.

# evaluation x

## [1] 3

## Basic Arithmetic

At its most basic function R can be used as a calculator. When applying basic arithmetic, the PEMDAS order of operations applies: parentheses first followed by exponentiation, multiplication and division, and final addition and subtraction.

8 + 9 / 5 ^ 2

## [1] 8.36

8 + 9 / (5 ^ 2)

## [1] 8.36

8 + (9 / 5) ^ 2

## [1] 11.24

(8 + 9) / 5 ^ 2

## [1] 0.68

By default R will display seven digits but this can be changed using options() as previously outlined.

1 / 7

## [1] 0.1428571

options(digits = 3)

1 / 7

## [1] 0.143

pi

## [1] 3.141592654

options(digits = 22) pi

## [1] 3.141592653589793115998

We can also perform integer divide (%/%) and modulo (%%) functions. The integer divide function will give the integer part of a fraction while the modulo will provide the remainder. 42 / 4 # regular division

## [1] 10.5

42 %/% 4 # integer division

## [1] 10

42 %% 4 # modulo (remainder)

## [1] 2

## R Objects:-

1. Vectors
2. Lists
3. Matrices
4. Arrays
5. Factors
6. Data Frames
7. **Vectors**

R Vectors are the same as the arrays in R language which are used to hold multiple data values of the same type. One major key point is that in R Programming Language the indexing of the vector will start from ‘1’ and not from ‘0’. We can create numeric vectors and character vectors as well.

## # R program to create Vectors

# we can use the c function

# to combine the values as a vector. # By default the type will be double X<- c(61, 4, 21, 67, 89, 2)

cat('using c function', X, '\n') # seq() function for creating

# a sequence of continuous values.

# length.out defines the length of vector. Y<- seq(1, 10, length.out = 5)

cat('using seq() function', Y, '\n') # use':' to create a vector

# of continuous values.

Z<- 2:7

cat('using colon', Z)

## Output:

using c function 61 4 21 67 89 2

using seq() function 1 3.25 5.5 7.75 10

using colon 2 3 4 5 6 7

1. **Lists**

A list in R is a generic object consisting of an ordered collection of objects. Lists are one- dimensional, heterogeneous data structures. The list can be a list of vectors, a list of matrices, a list of characters and a list of functions, and so on.

## Creating a List

To create a List in R you need to use the function called “list()”. In other words, a list is a generic vector containing other objects. To illustrate how a list looks, we take an example here. We want to build a list of employees with the details. So for this, we want attributes such as ID, employee name, and the number of employees.

# R program to create a List

# The first attributes is a numeric vector

# containing the employee IDs which is created # using the command here

empId = c(1, 2, 3, 4)

# The second attribute is the employee name # which is created using this line of code here # which is the character vector

empName = c("Debi", "Sandeep", "Subham", "Shiba")

# The third attribute is the number of employees # which is a single numeric variable. numberOfEmp = 4

# We can combine all these three different # data types into a list

# containing the details of employees

# which can be done using a list command empList = list(empId, empName, numberOfEmp) print(empList)

## Output:

[[1]]

[1] 1 2 3 4

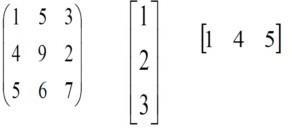
[[2]]

[1] "Debi" "Sandeep" "Subham" "Shiba" [[3]]

[1] 4

1. **Matrices**

Matrix is a rectangular arrangement of numbers in rows and columns. In a matrix, as we know rows are the ones that run horizontally and columns are the ones that run vertically. In R programming, matrices are two-dimensional, homogeneous data structures. These are some examples of matrices:



To create a matrix in R you need to use the function called matrix(). The arguments to this matrix() are the set of elements in the vector. You have to pass how many numbers of rows and how many numbers of columns you want to have in your matrix.

# R program to create a matrix A = matrix(

# Taking sequence of elements c(1, 2, 3, 4, 5, 6, 7, 8, 9),

# No of rows nrow = 3,

# No of columns ncol = 3,

# By default matrices are in column-wise order

# So this parameter decides how to arrange the matrix byrow = TRUE

)

# Naming rows

rownames(A) = c("a", "b", "c")

# Naming columns colnames(A) = c("c", "d", "e")

cat("The 3x3 matrix:\n") print(A)

## Output:

The 3x3 matrix:

|  |  |  |
| --- | --- | --- |
| c | d | e |
| a 1 | 2 | 3 |
| b 4 | 5 | 6 |
| c 7 | 8 | 9 |

1. **Arrays**

Arrays are essential data storage structures defined by a fixed number of dimensions. Arrays are used for the allocation of space at contiguous memory locations.

In R Programming Language Uni-dimensional arrays are called vectors with the length being their only dimension. Two-dimensional arrays are called matrices, consisting of fixed numbers of rows and columns. R Arrays consist of all elements of the same data type. Vectors are supplied as input to the function and then create an array based on the number of dimensions.

Creating an Array

An R array can be created with the use of array() the function. A list of elements is passed to the array() functions along with the dimensions as required.

## Syntax:

array(data, dim = (nrow, ncol, nmat), dimnames=names) where

nrow: Number of rows ncol : Number of columns

nmat: Number of matrices of dimensions nrow \* ncol dimnames : Default value = NULL.

## Uni-Dimensional Array

A vector is a uni-dimensional array, which is specified by a single dimension, length. A Vector can be created using ‘c()‘ function. A list of values is passed to the c() function to create a vector.

vec1 <- c(1, 2, 3, 4, 5, 6, 7, 8, 9)

print (vec1)

# cat is used to concatenate # strings and print it.

cat ("Length of vector : ", length(vec1))

## Output:

[1] 1 2 3 4 5 6 7 8 9

Length of vector : 9

## Multi-Dimensional Array

A two-dimensional matrix is an array specified by a fixed number of rows and columns, each containing the same data type. A matrix is created by using array() function to which the values and the dimensions are passed.

# arranges data from 2 to 13

# in two matrices of dimensions 2x3 arr = array(2:13, dim = c(2, 3, 2)) print(arr)

Output:

, , 1

[,1] [,2] [,3]

[1,] 2 4 6

[2,] 3 5 7

, , 2

[,1] [,2] [,3]

[1,] 8 10 12

[2,] 9 11 13

1. **Factors**

Factors in R Programming Language are data structures that are implemented to categorize the data or represent categorical data and store it on multiple levels.

They can be stored as integers with a corresponding label to every unique integer. The R factors may look similar to character vectors, they are integers and care must be taken while using them as strings. The R factor accepts only a restricted number of distinct values. For example, a data field such as gender may contain values only from female, male, or transgender.

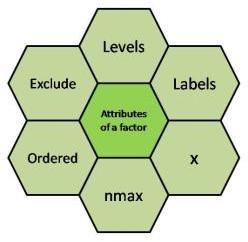
In the above example, all the possible cases are known beforehand and are predefined. These distinct values are known as levels. After a factor is created it only consists of levels that are by default sorted alphabetically.

## Attributes of Factors in R Language

x: It is the vector that needs to be converted into a factor.

Levels: It is a set of distinct values which are given to the input vector x. Labels: It is a character vector corresponding to the number of labels.

Exclude: This will mention all the values you want to exclude. Ordered: This logical attribute decides whether the levels are ordered. nmax: It will decide the upper limit for the maximum number of levels.



## Creating a Factor in R Programming Language

The command used to create or modify a factor in R language is – factor() with a vector as input. The two steps to creating an R factor :

## Creating a vector

Converting the vector created into a factor using function factor()

**Examples**: Let us create a factor gender with levels female, male and transgender.

## # Creating a vector

x <-c("female", "male", "male", "female") print(x)

## # Converting the vector x into a factor

# named gender gender <-factor(x) print(gender)

# Creating a factor with levels defined by programmer gender <- factor(c("female", "male", "male", "female"), levels = c("female", "transgender", "male"));

gender

## Output

[1] female male male female Levels: female transgender male

Further one can check the levels of a factor by using function levels().

## Checking for a Factor in R

The function is.factor() is used to check whether the variable is a factor and returns “TRUE” if it is a factor.

gender <- factor(c("female", "male", "male", "female")); print(is.factor(gender))

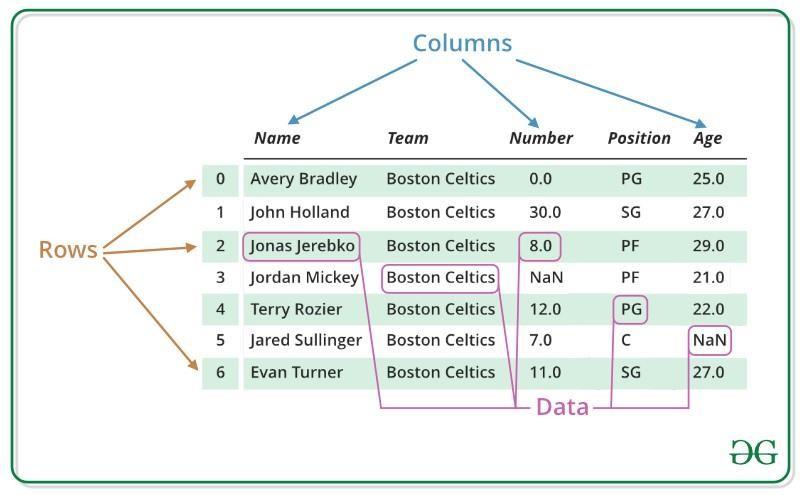
Output

[1] TRUE

1. **Data Frames**

R Programming Language is an open-source programming language that is widely used as a statistical software and data analysis tool. Data Frames in R Language are generic data objects of R that are used to store tabular data.

Data frames can also be interpreted as matrices where each column of a matrix can be of different data types. R DataFrame is made up of three principal components, the data, rows, and columns.



***R – Data Frames***

## R Data Frames Structure

As you can see in the image below, this is how a data frame is structured.

The data is presented in tabular form, which makes it easier to operate and understand.

## Create Dataframe in R Programming Language

To create an R data frame use data.frame() function and then pass each of the vectors you have created as arguments to the function.

# R program to create dataframe # creating a data frame

friend.data <- data.frame( friend\_id = c(1:5),

friend\_name = c("Sachin", "Sourav", "Dravid", "Sehwag", "Dhoni"),

stringsAsFactors = FALSE

)

# print the data frame print(friend.data)

## Output:

|  |  |  |
| --- | --- | --- |
|  | friend\_id | friend\_name |
| 1 | 1 | Sachin |
| 2 | 2 | Sourav |
| 3 | 3 | Dravid |
| 4 | 4 | Sehwag |
| 5 | 5 | Dhoni |

**Conclusion:**

This experiment provided a basic introduction to R programming and its objects. By continuing to explore and practice, you can unlock the potential of this powerful tool for solving engineering problems and analysing data

## Experiment No. 8

**Objective:** Exercises to draw a scatter diagram, residual plots, outliers leverage and influential data points in R

## Equipment:

* Computer with R installed

## Theory and Technique:

**Scatter Plot** - A scatter plot is a set of dotted points representing individual data pieces on the horizontal and vertical axis. In a graph in which the values of two variables are plotted along the X-axis and Y-axis, the pattern of the resulting points reveals a correlation between them.

## Scatter plot in R Programming Language using the plot() function.

Syntax: plot(x, y, main, xlab, ylab, xlim, ylim, axes), where

* ***x*** *sets the horizontal coordinates.*
* ***y*** *sets the vertical coordinates.*
* ***xlab*** *is the label for horizontal axis.*
* ***ylab*** *is the label for vertical axis.*
* ***main*** *is the title of the chart.*
* ***xlim*** *is used for plotting values of x.*
* ***ylim*** *is used for plotting values of y.*
* ***axes*** *indicates whether both axes should be drawn on the plot.*

**Exercise 1:** Plot the Scatter diagram from the following pairs of values.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Sales | 50 | 50 | 55 | 60 | 65 | 65 | 65 | 60 | 60 | 50 |
| Expenses | 11 | 13 | 14 | 16 | 16 | 15 | 15 | 14 | 13 | 13 |

## Steps

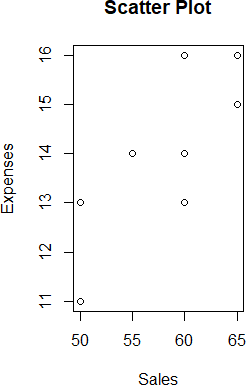
1. Consider a set of data.
2. Write an R program to plot the Scatter diagram from the data.
3. Execute the program

## #code

x = c(50,50,55,60,65,65,65,60,60,50)

y=c(11,13,14,16,16,15,15,14,13,13)

plot(x,y,main="Scatter Plot",xlab="Sales",ylab="Expenses")



**Exercise 2:** Plot the Scatter diagram using a dataset

## Steps:

1. Create a data set
2. Create a Data Frame
3. Write an R program to plot the Scatter diagram from the data.

## #code

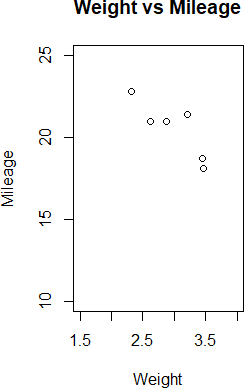
x <- c(2.620,2.875,2.320,3.215,3.440,3.460);

y<- c(21.0,21.0,22.8,21.4,18.7,18.1)

cars = data.frame(x,y)

cars = data.frame("wt" = x, "mileage" = y) input <- cars[, c('wt', 'mpg')]

plot(x = input$wt, y = input$mileage, xlab = "Weight", ylab = "Mileage", xlim = c(1.5, 4), ylim = c(10, 25),main = "Weight vs Mileage")



**Residual plots** are often used to assess whether or not the residuals in regression analysis are normally distributed and whether or not they exhibit heteroscedasticity.

Residual: A residual is the vertical difference between the actual value and the predicted value. That is, Residual Plot: A residual plot is a scatterplot that displays the residuals on the vertical axis and the independent variable on the horizontal axis.

A residual plot tells us about the quality of the linear regression model by showing the differences between the predicted and actual values.

## Exercise 3: Plot a Residual plot using the given example

A Computer manager needs to know how efficiency of her new computer program depends on the size of the incoming data. Efficiency will be measured by the number of processed requests per hour.

The response variable here is the number of processed requests Y and the explanatory variable is the data size X. Applying the program to datasets of different sizes, she gets the following results, Data size (gigabytes) = x, Processed request = y

## Steps

* 1. Consider a set of data.
  2. Calculate the least squares regression equation
  3. Plot Size of Data Vs Requests
  4. Calculate and plot residuals

## # Code

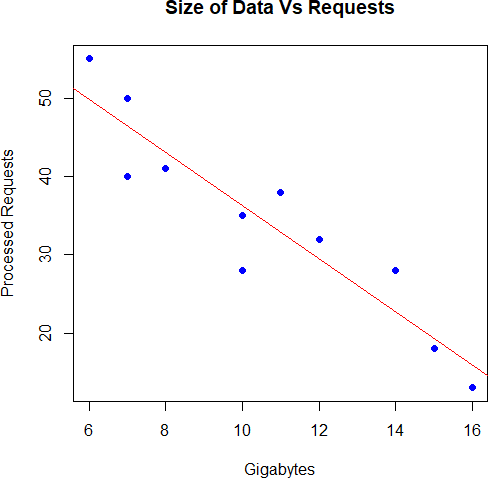
x = c(6,7,7,8,10,10,11,12,14,15,16)

y=c(55,40,50,41,35,28,38,32,28,18,13)

mod=lm(y~x) summary(mod)

plot(x,y,main="Size of Data Vs Requests", xlab="Gigabytes", ylab="Processed Requests",pch=16, col="blue")

abline(a=70.16, -3.39, col="red");



mod$residuals

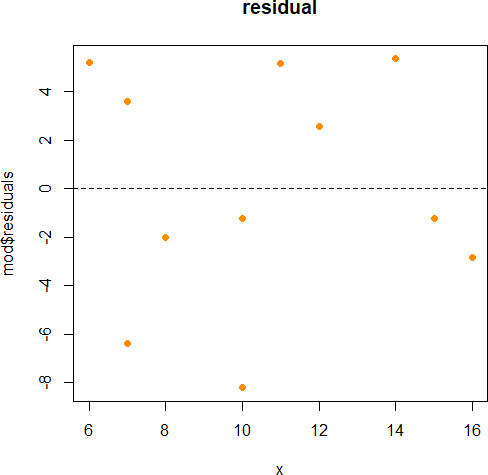
1 2 3 4 5 6 7 8

5.208723 -6.397196 3.602804 -2.003115 -1.214953 -8.214953 5.179128 2.573209

9 10 11

5.361371 -1.244548 -2.850467

plot(x,mod$residuals, pch=16, col="darkorange", main="residual") abline(h=0, lty=2)



Outlier Detection

**Exercise 4:** Outlier Detection

**Outliers**: Outliers are the points that are distinct and deviant from the bulk of the dataset. In general, the outliers have **high residual** values means that the difference is greater than the b/w observed and predicted value.

**Leverage Points**: A leverage point is defined as an observation that has a value of x that is far away from the mean of x.

**Influential Points**: An influential observation is defined as an observation that has a large influence on the fit of the model. One method to find influential points is to compare the fit of the model with and without each observation.

## Steps

1. Consider a Data Frame
2. Plot the data
3. Detecting outliers
4. Identify observations with high residuals

## # Code

data <- data.frame(x,y) plot(data$x, data$y)

# Example: Detecting outliers

# Identify observations with high residuals

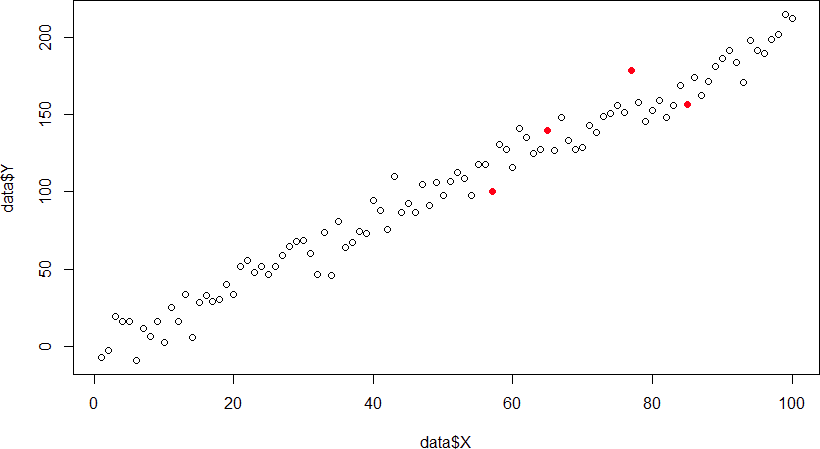
outliers <- which(abs(resid(mod)) > 2 \* sd(resid(mod))) X <- 1:100

Y <- 2 \* X + rnorm(100, mean = 0, sd = 10) model <- lm(Y ~ X, data = data)

data <- data.frame(X = 1:100, Y = 2 \* X + rnorm(100, mean = 0, sd = 10)) outliers <- which(abs(resid(model)) > 2 \* sd(resid(model)))

plot(data$X, data$Y)

points(data$X[outliers], data$Y[outliers], col = "red", pch = 19)



## Exercise 4: Influential Points

An influential point is a point that has a large impact on the regression. Surprisingly, these are not the same thing. A point can be an outlier without being influential. A point can be influential without being an outlier. A point can be both or neither.

# Example: Influence and Cook's Distance

# Calculate Cook's distance for each observation influential <- cooks.distance(model)

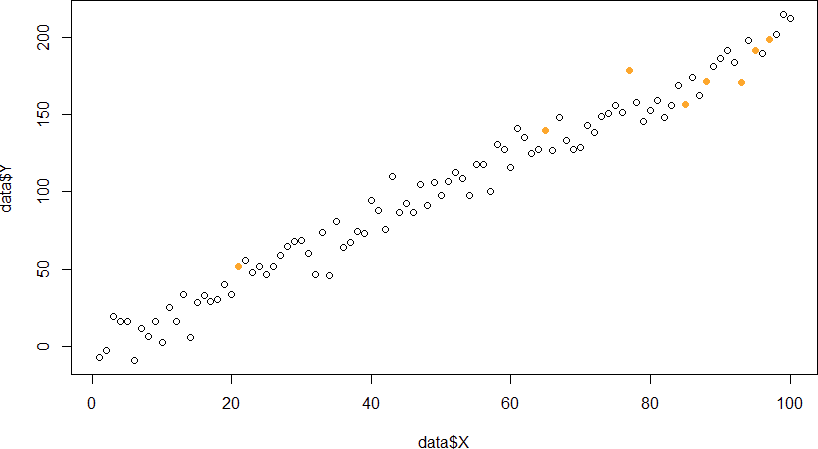
# Identify observations with Cook's distance exceeding a threshold #(e.g., 3/n, where n is the number of observations)

threshold <- 3 / length(data$X)

influential\_obs <- which(influential > threshold)

# Highlight influential observations in the scatterplot plot(data$X, data$Y)

points(data$X[influential\_obs], data$Y[influential\_obs], col = "orange", pch = 19)



## Conclusion:

This experiment provided a basic introduction to R programming and its objects. By continuing to explore and practice, you can unlock the potential of this powerful tool for solving engineering problems and analysing data

**Experiment No. 9 Objective:** Exercises to calculate correlation using R **Equipment:**

* Computer with R installed

## Theory and Technique:

**Pearson Correlation Testing in R Programming**

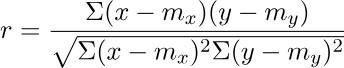
Correlation is a statistical measure that indicates how strongly two variables are related. It involves the relationship between multiple variables as well. For instance, if one is interested to know whether there is a relationship between the heights of fathers and sons, a correlation coefficient can be calculated to answer this question. Generally, it lies between -1 and +1. It is a scaled version of covariance and provides the direction and strength of a relationship. Correlation coefficient test in R

There are mainly two types of correlation:

1. **Parametric Correlation** – Pearson correlation(r): It measures a linear dependence between two variables (x and y) is known as a parametric correlation test because it depends on the distribution of the data.
2. **Non-Parametric Correlation** – Kendall(tau) and Spearman(rho): They are rank- based correlation coefficients, and are known as non-parametric correlation.

## Pearson Rank Correlation Coefficient Formula

Pearson Rank Correlation is a parametric correlation. The Pearson correlation coefficient is probably the most widely used measure for linear relationships between two normal distributed variables and thus often just called “correlation coefficient”. The formula for calculating the Pearson Rank Correlation is as follows:



* ***r:*** *pearson correlation coefficient*
* ***x and y:*** *two vectors of length n*
* ***mx and my:*** *corresponds to the means of x and y, respectively.*

## Note:

* r takes a value between -1 (negative correlation) and 1 (positive correlation).
* r = 0 means no correlation.
* Can not be applied to ordinal variables.
* The sample size should be moderate (20-30) for good estimation.
* Outliers can lead to misleading values means not robust with outliers.

## Implementation in R

R Programming Language provides two methods to calculate the pearson correlation coefficient. By using the functions cor() or cor.test() it can be calculated. It can be noted that cor() computes the correlation coefficient whereas cor.test() computes the test for association or correlation between paired samples. It returns both the correlation coefficient and the significance level(or p-value) of the correlation.

***Syntax:*** *cor(x, y, method = “pearson”) cor.test(x, y, method = “pearson”)* ***Parameters:***

* ***x, y:*** *numeric vectors with the same length*
* ***method:*** *correlation method*

**Exercise 1:** Correlation Coefficient Test in R using cor() method

## Steps:

1. Consider two numeric Vectors with same length
2. Calculate Correlation coefficient

## # Code

# R program to illustrate

# pearson Correlation Testing # Using cor()

# Taking two numeric Vectors with same length x = c(1, 2, 3, 4, 5, 6, 7)

y = c(1, 3, 6, 2, 7, 4, 5)

# Calculating Correlation coefficient # Using cor() method

result = cor(x, y, method = "pearson") # Print the result

cat("Pearson correlation coefficient is:", result)

## Output:

Pearson correlation coefficient is: 0.5357143

**Experiment No. 10 Objective:** Exercises to implement Time series Analysis using R **Equipment:**

* Computer with R installed

## Theory and Technique:

Time series is a series of data points in which each data point is associated with a timestamp. A simple example is the price of a stock in the stock market at different points of time on a given day. Another example is the amount of rainfall in a region at different months of the year. R language uses many functions to create, manipulate and plot the time series data. The data for the time series is stored in an R object called **time-series object**. It is also a R data object like a vector or data frame.

The time series object is created by using the **ts()** function.

**Syntax**

The basic syntax for **ts()** function in time series analysis is − timeseries.object.name <- ts(data, start, end, frequency) Following is the description of the parameters used −

* **data** is a vector or matrix containing the values used in the time series.
* **start** specifies the start time for the first observation in time series.
* **end** specifies the end time for the last observation in time series.
* **frequency** specifies the number of observations per unit time. Except the parameter "data" all other parameters are optional.

**Exercise:** Consider the annual rainfall details at a place starting from January 2012. We create an R time series object for a period of 12 months and plot it

## # Code

# Get the data points in form of a R vector.

rainfall <- c(799,1174.8,865.1,1334.6,635.4,918.5,685.5,998.6,784.2,985,882.8,1071)

# Convert it to a time series object.

rainfall.timeseries <- ts(rainfall,start = c(2012,1),frequency = 12) # Print the timeseries data.

print(rainfall.timeseries)

# Give the chart file a name. png(file = "rainfall.png")

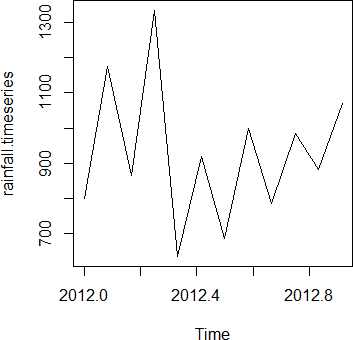
# Plot a graph of the time series. plot(rainfall.timeseries)

When we execute the above code, it produces the following result and chart − Jan Feb Mar Apr May Jun Jul Aug Sep

2012 799.0 1174.8 865.1 1334.6 635.4 918.5 685.5 998.6 784.2

Oct Nov Dec 2012 985.0 882.8 1071.0

The Time series chart −



*Time Series Data visualization chart*