

DEPARTMENT OF MATHEMATICS
(Under DBT Star College Scheme)

LABORATORY MANUAL



SAGEMATH

23CMAT4P



G.T.N. ARTS COLLEGE (AUTONOMOUS)

A Telugu Linguistic Minority Institution | Affiliated to Madurai Kamaraj University
Re-accredited (Cycle - II) by NAAC with 'A++' Grade
Recognized under DBT STAR COLLEGE SCHEME | Supported by DST - FIST Program
Listed in NIRF- 2024 Ranking
Old Karur Road, Dindigul – 624 005.

Sl. No.	Program	Page No.
	Introduction	3
	Syntax	9
1.	Compute first three Derivatives	10
2.	Plot 2D and 3D Graph	11
3.	Find Surface Area using Package	13
4.	Find roots using Newton's method	14
5.	Area between Curves using Integrals	16
6.	Check the group is Abelian or not	17
7.	Compute Volume of Solid of Revolution	18
8.	Compute solution of stem of Linear Equation	19
9.	Compute Divergence and Curl	20
10.	Find Local Extrema and Inflection Point	21

Introduction to SageMath

SageMath (or simply **Sage**) is an open-source mathematics software system designed for solving a wide range of mathematical problems. It combines many existing open-source math tools into one unified system and provides a powerful programming interface similar to **Python**.

SageMath is a free mathematical computing environment built on Python that helps perform symbolic and numerical computations, graphing, algebra, calculus, and more. It is widely used in mathematics education, research, and scientific computing.

What SageMath Can Do?

SageMath can be used for:

Basic Mathematics	Arithmetic, algebra, inequalities
Calculus	Limits, derivatives, integrals, differential equations
Algebra	Matrices, vectors, linear algebra, abstract algebra
Graph Theory	Graphs, networks, Laplacian matrix, shortest paths
Plotting & Visualization	2D and 3D graph plotting
Statistics & Data Analysis	Probability, random sampling
Number Theory	Prime numbers, modular arithmetic, cryptography

Why SageMath?

- Free and open-source
- Based on **Python**, so easy to learn
- Supports symbolic and numerical computation
- Used in research, education, and engineering

Key Features of SageMath 9.1

SageMath 9.1 supports:

- ✓ Python 3-based environment
- ✓ Symbolic computation using Maxima
- ✓ Numerical computation using NumPy, SciPy
- ✓ Linear algebra
- ✓ Graph theory and combinatorics
- ✓ Number theory and cryptography tools
- ✓ 2D and 3D plotting using Matplotlib and Tachyon
- ✓ Jupyter Notebook interface

Version: **SageMath 9.1**
Released on **May 01, 2020**

Installation Steps for Windows

1. Download the Installer

- Visit the official SageMath page:
 🔗 *sagemath.org* → *Downloads*
- Choose **SageMath 9.1 for Windows** (ZIP installer).

2. Extract the ZIP File

- Right-click the downloaded file → **Extract All**
- Extract to a folder (example: C:\SageMath)

3. Run SageMath

- Open the extracted folder and double-click:
- SageMath 9.1.lnk
- You can also run via **Jupyter Notebook** by selecting:
- SageMath 9.1 Notebook

4. Optional: Create Desktop Shortcut

- Right-click → **Send to** → **Desktop (shortcut)**

Getting started with SageMath

Installation of the software is easy like any other software. The software gets ready after its installation into a computer system.

SageMath Console Workspace

When you open **SageMath 9.1 (Console)**, you see a text-based interface that looks similar to Python.

Jupyter Notebook / Sage Notebook Workspace

Many users prefer SageMath through **Jupyter Notebook**, which offers a graphical, interactive workspace.

When you run:

```
sage -n jupyter
```

You open a browser interface where:

- You write commands in **cells**
- Output appears below the code
- You can save your work in `.ipynb` format

Example Notebook cell:

```
x = var('x')
f = x^2 - 4*x + 6
show(diff(f,x))
plot(f, (x, -5, 5))
```

Notebook Workspace Features

Feature	Description
Code Cells	Type and run Sage/Python commands
Markdown Cells	Add notes, headings, explanations
Plots	Show 2D and 3D graphs inline
Autosave	Saves notebooks automatically
Export Options	Export to PDF, HTML, or Python script

File Browser and Tools (Notebook Mode)

On the left side of the notebook interface, you see:

- 📁 Folder list
- ✚ Create new file

🗑 Delete/rename items

▶ Run commands

This helps organize projects, graphs, and programs.

Graphical Output Examples

The Sage workspace supports visual outputs like:

- Plots
- Graphs
- Matrices
- Tables

Example:

```
plot(sin(x), (x, -2*pi, 2*pi))
```

Will show a smooth sine wave inside the workspace.

Help and Documentation in Workspace

SageMath includes built-in help.

Assignment Operator

The assignment operator “=” is used for assigning a value (or a matrix) to variable. On the left-hand side of “=” is placed a variable name to which the value on right hand side is to be assigned.

Variable Names

Variable names may be as long as the user wants, but only the first 24 characters are taken into account in Sagemath. For consistency, we should consider only variable names which are not made of more than 24 characters. All ASCII letters from "a" to "z", from "A" to "Z" and from "0" to "9" are allowed with the additional letters "%", "_", "#", "!", "\$", "?".

Displaying Output in the Console:

print () function: `print()` displays variables or strings. It's useful for formatted output or messages. For example, `x = 10; print(x);` displays 10. `disp("The value of x is: " + string(x));` displays "The value of x is: 10". `disp("Hello, World!");` displays "Hello, World!".

Method to take input in the Console:

Assigning values: `x = 10; y = "hello";`

Using input(): `a=int(input('Enter the value : '))`

1. Basic Arithmetic Operations

- Addition: `a + b`
- Subtraction: `a - b`
- Multiplication: `a * b`
- Division: `a / b`

2. Mathematical Functions

- Square root: `sqrt(x)`
- Exponential: `exp(x)`
- Logarithm (natural): `log(x)`
- Logarithm (base 10): `log10(x)`
- Sine: `sin(x)`
- Cosine: `cos(x)`
- Tangent: `tan(x)`
- Absolute value: `abs(x)`

3. Defining Variables & Functions

Assigning a value: $x = 5$

Defining a function: `def fact(x):`

4. Loops and Conditional Statements

For Loop:

`for i in range (1,var+1):`

`print(i)`

If Statement:

`if x > 0 then`

`print("Positive number")`

`elif x==0`

`print("x is zero ")`

`elif`

`print("Non-positive number")`

Syntax

1. Differentiation

```
f.diff(x)
```

2. Loops and Conditional Statements

For Loop:

```
for i in range (1,var+1):
```

```
    print(i)
```

If Statement:

```
if x > 0 then
```

```
    print("Positive number")
```

```
elif x==0
```

```
    print("x is zero ")
```

```
elif
```

```
    print("Non-positive number")
```

3. 2D graph & 3D graph

```
print('*** Program to create ***')
```

```
x = var('x')
```

```
f(x) = x+1
```

```
g = plot(f(x), (x, -3, 3))
```

```
print('Here is the graph:')
```

```
g.show()
```

```
print('*** Program to create 3D graph ***')
```

```
f(x,y) = x**2 + y**2
```

```
g = plot3d(f,(x,-2,2), (y,-2,2), color = 'green', opacity = 0.5)
```

```
g.show()
```

Problem 1:

Write a SAGEMATH program to compute the first three derivatives of $f(x) = x\sin(x)$.

Aim: To compute the first, second and third derivatives of a given function $f(x) = x\sin(x)$ using SAGEMATH

Program:

```
print('*** Differentiation ***')

x = var('x')
f = x*sin(x)

f1 = f.diff(x)
print('First derivative' , f1)
f2 = f1.diff(x)
print('Second derivative' , f2)
f3 = f2.diff(x)
print('Third derivative' , f3)
```

Output:

```
*** Differentiation ***
First derivative x*cos(x) + sin(x)
Second derivative -x*sin(x) + 2*cos(x)
Third derivative -x*cos(x) - 3*sin(x)
```

Problem 2:

Write a SAGEMATH program to create and plot 2D graph and 3D graph.

Aim: To create and plot 2D graph using SAGEMATH.

Program:

```
print('*** Program to create 2D graph ***')
```

```
x = var('x')
```

```
f(x) = x+1
```

```
g = plot(f(x), (x, -3, 3))
```

```
print('Here is the graph:')
```

```
g.show()
```

```
print('*** Program to create 3D graph ***')
```

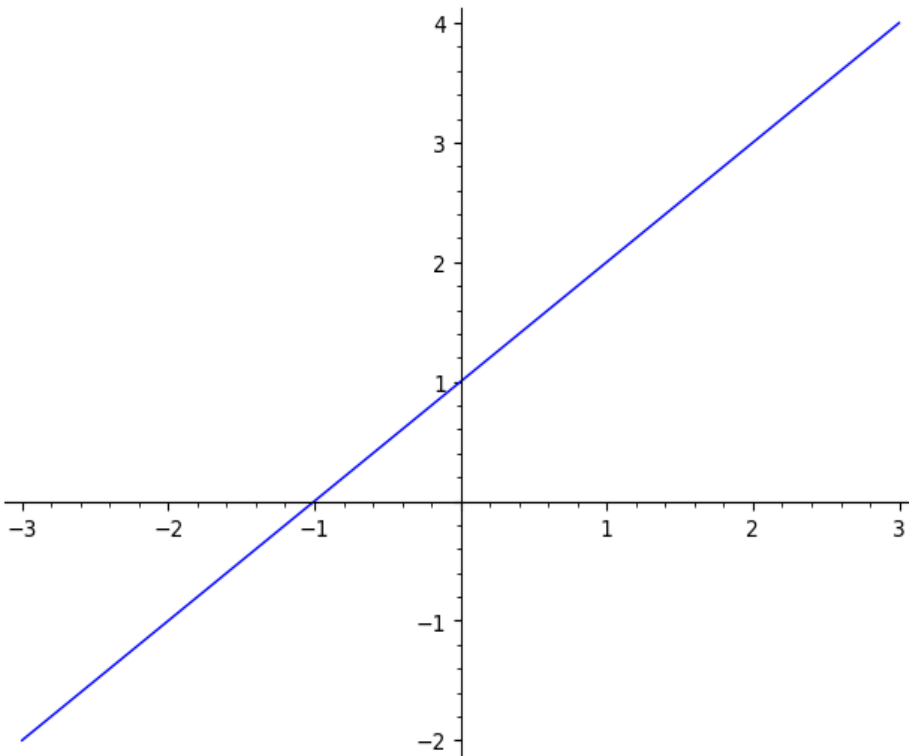
```
f(x,y) = x**2 + y**2
```

```
g = plot3d(f,(x,-2,2), (y,-2,2), color = 'green', opacity = 0.5)
```

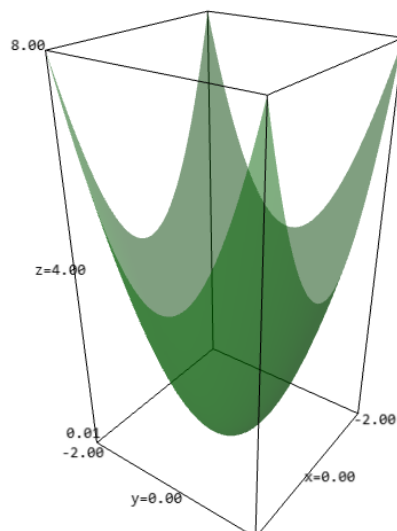
```
g.show()
```

Output:

*** Program to create 2D graph ***
Here is the graph:



*** Program to create 3D graph ***



①

Problem 3:

Write a SAGEMATH program to find the surface area of given surface using package.

Aim: To find the surface area of given surface using package in SAGEMATH.

Program:

```
#Define the variable
```

```
var('x y')
```

```
f = x^2 + y^2 # <-- Change this function as needed
```

```
S = sqrt(1 + diff(f,x)^2 + diff(f,y)^2)
```

```
area = integral(integral(S, (y, 0, 1)), (x, 0, 1)) # limits: x=0..1, y=0..1
```

```
print("Surface Area =", area)
```

Output:

```
Surface Area = -1/6*arctan(1/2) + 7/24*log(5) + 7/12*log(1/5*sqrt(5)*(3*sqrt(5)*sqrt(1/5) + 2)) + 1
```

Problem 4:

Write a SAGEMATH program to find the approximate roots using Newton's method.

Aim:

To find the approximate roots using Newton's method in SAGEMATH.

Program:

```
# Newton's Method

# Define the variable and function

x = var('x')

f(x) = input("Enter the function f(x): ")

# Derivative of the function

f_prime = diff(f(x), x)

# Get initial guess and number of iterations

x0 = float(input("Enter the initial guess: "))

n = int(input("Enter number of iterations: "))

print("\nNewtons Method Iterations:")

print("-----")

print("Iteration\tApproximate Root")

# Newton's Method Loop

for i in range(1, n+1):

    x1 = x0 - float(f(x0)/f_prime(x0))

    print(i, "\t\t", x1)

    x0 = x1

print("\nFinal Approximate Root:", x1)
```

Output:

Enter the function $f(x)$: $x^3 - 2x - 5$

Enter the initial guess: 2

Enter number of iterations: 5

Iteration	Approximate Root
1	2.091
2	2.094551
3	2.094551482
4	2.094551481543
5	2.094551481543

Final Approximate Root: 2.094551481543

Problem 5:

Write a SAGEMATH program to plot and find the area between curves using integrals.

Aim: To plot and find the area between curves using integrals in SAGEMATH.

Program:

```
f(x) = input('Enter the function f(x): ') g(x) =
input('Enter the function g(x): ')
h = plot(f(x), x, -10, 10, ymin = -10, ymax = 20, legend_label = 'f(x)', color = 'red', figsize = 5)
j = plot(g(x), x, -10, 10, ymin = -10, ymax = 20, legend_label = 'g(x)', color = 'yellow', figsize = 5)
q = h+j q.show()
a = solve(f(x) == g(x), x)
print(a)
b = eval(input('Enter the lower limit for (f(x)-g(x)): '))
c = eval(input('Enter the upper limit for (f(x)-g(x)): '))
d = eval(input('Enter the lower limit for (g(x)-f(x)): '))
e = eval(input('Enter the upper limit for (g(x)-f(x)): '))
f = integral(f(x)-g(x), x, b, c)+integral(g(x)-f(x), x, d, e)
print('The area between the curves is' , f)
```

Output:

Enter the function f(x): $x^2 + 4$

Enter the function g(x): $x + 6$

[x == -1, x == 4]

Enter the lower limit for (f(x)-g(x)): -1

Enter the upper limit for (f(x)-g(x)): 4

Enter the lower limit for (g(x)-f(x)): 4

Enter the upper limit for (g(x)-f(x)): -1

The area between the curves is $31/6$

Problem 6:

Write a SAGEMATH program to find the given group is abelian or not.

Aim: To find the given group is abelian or not using SAGEMATH.

Program:

```
#Define the graph
G = SymmetricGroup(2)
#Define a function to check if the group is abelian
def is_abelian(G):
    for g in G:
        for h in G:
            if g*h != h*g:
                return False
    return True
#Check if the group is abelian if
is_abelian(G):
    print("The group is abelian.")
else:
    print("The group is not abelian.")
```

Output:

The group is abelian.

Problem 7:

Write a SAGEMATH program to find the volume of the solid of revolution obtained by rotating the region bounded by the graph of $f(x) = \sqrt{u+1}$, the x-axis, and the vertical line $x=2$.

Aim: To find the volume of solid of revolution by revolving a given function around an axis and visualize using SAGEMATH.

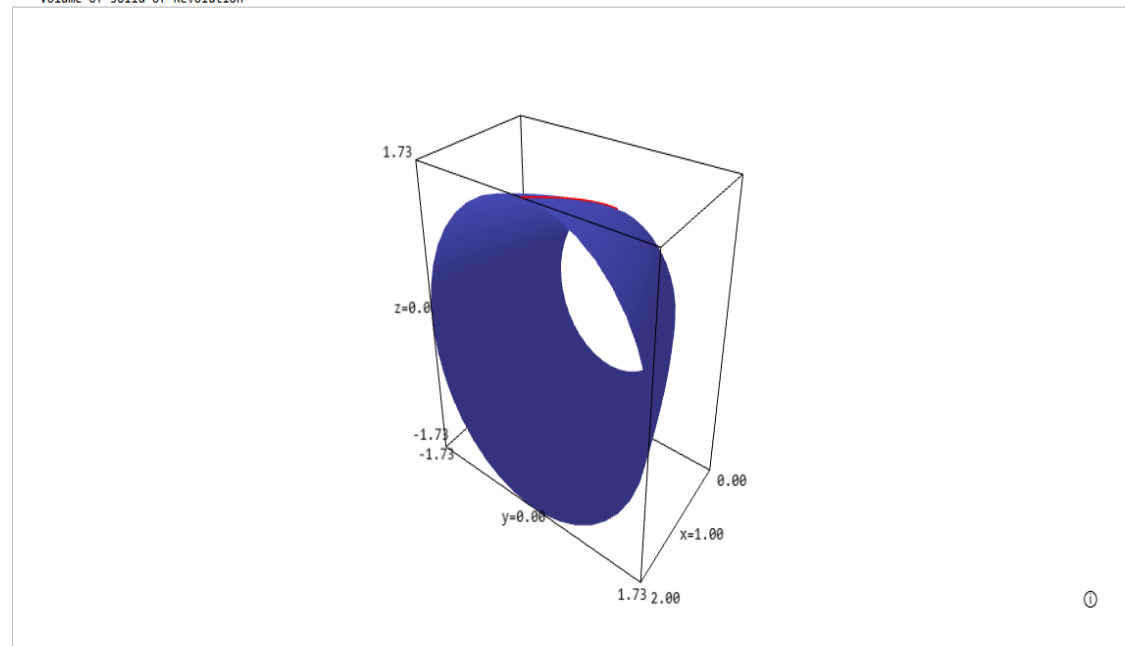
Program:

```
print('*** Volume of solid of Revolution ***') var('u')
f(u) = sqrt(u+1)
h = plot(f(u), u, 0, 2, figsize = 3, fill = True)
s = revolution_plot3d(f(u), (u, 0, 2), show_curve = True, opacity = 7, parallel_axis =
'x').show(aspect_ratio = (1, 1, 1))

v = pi*integral(f(u)^2, u, 0, 3)
print('Volume = ', v)
```

Output:

*** Volume of solid of Revolution ***



Volume = $15/2\pi$

Problem 8:

Write a SAGEMATH program to find the solution for a system of linear equations.

$$2x + 3y - z = 5$$

$$x - 2y + 4z = -2$$

$$3x + y + 2z = 7$$

Aim: To solve the system of linear equations using SAGEMATH.

Program:

```
print('*** System of linear equations ***')
#Define the variables
x, y, z = var('x y z')
#Define the equations
eq1 = 2*x+3*y-z == 5
eq2 = x-2*y+4*z == -2
eq3 = 3*x+y+2*z == 7
#Define the system of equations
eqs = [eq1, eq2, eq3]
#Solve the system of equations
solution = solve(eqs, x, y, z)
#Print the solution
print('The solution of System of Linear Equations')
print(solution)
```

Output:

```
*** System of linear equations ***
The solution of System of Linear Equations
[
[x == (44/7), y == (-27/7), z == -4]
]
```

Problem 9:

Write a SAGEMATH program to find the divergence and curl of vector valued function.

Aim:

To find the divergence and curl of vector valued function using SAGEMATH

Program:

```
print('*** Divergence and Curl ***')
#Define the variables
x, y, z = var('x y z')
#Define the vector valued function
F = vector([x^2*y, y^2*z, z^2*x])
#Calculate the divergence
divergence = F.div([x, y, z])
print('The Divergence of Given Vector Valued Function F is : ')
print('divF = ')
print(divergence)
print('The Curl of Given Vector Valued Function F is : ')
print('curlF= ')
#Calculate the curl
curl = F.curl([x, y, z])
print(curl)
```

Output:

```
*** Divergence and Curl ***
The Divergence of Given Vector Valued Function F is :
divF =
2*x*y + 2*x*z + 2*y*z
The Curl of Given Vector Valued Function F is :
curlF=
(-y^2, -z^2, -x^2)
```

Problem 10:

Write a SAGEMATH program to find all local extrema and inflection points of a function.

Aim:

To find all local extrema and inflection points of a function using SAGEMATH.

Program:

#Define the variable and function

```
var('x')
```

```
f = x^3-3*x^2+2
```

#First derivative

```
f1 = diff(f, x)
```

```
critical_points = solve(f1 == 0,x)
```

#Second derivative

```
f2 = diff(f1, x)
```

#Classify extrema using the second derivative test

```
extrema = []
```

```
for cp in critical_points: cp_val = cp.rhs()
```

```
second_derivative_val = f2.subs(x = cp_val)
```

```
if second_derivative_val > 0:
```

```
    extrema.append((cp_val, "Local Min"))
```

```
elif second_derivative_val < 0:
```

```
    extrema.append((cp_val, "Local Max"))
```

#Find inflection points

```
inflection_points = solve(f2 == 0, x)
```

#Display results

```
print("Critical Points (Possible Extrema):\n", critical_points)
```

```
print("Classified Extreme:",extrema)
```

```
print("Inflection Points:\n",inflection_points)
```

Output:

```
Critical Points (Possible Extrema):  
[  
x == 0,  
x == 2  
]  
Classified Extreme: [(2, 'Local Min')]  
Inflection Points:  
[  
x == 1  
]
```