Rao Bahadur Y. Mahabaleswarappa Engineering College

Department of Mathematics

LAB MANUAL

of

First Semester Engineering Mathematics

Academic Year: 2025-2026

Contents

1 Partial Derivatives of Functions of Several Variables. 1

2 Jacobian of Functions of Several Variables 4

3 Finding gradient, divergent, curl and their geometrical interpretation 7

4 Finding rank, reduced echelon form, solving system of linear equations using Gauss elimination method 13

5 Test the Consistency of a System of Linear Equations 18

6 Determine Eigenvalues and Eigenvectors 20

7 Linearly Independence and Dependence sets 22

8 Basis and dimension 23

9 Linear transformation-range space and null space 24

10 Verification of the rank nullity theorem 26

Lab 1: Partial Derivatives of Functions of Several Variables

Aim: To find partial derivatives of functions of several variables.

1.1 Write a program to prove that mixed partial derivatives,

$$u_{xy} = u_{yx} \quad \text{for} \quad u = e^x \left(x\cos(y) - y\sin(y)\right).$$

Code:

```
from sympy import *

# Define variables
x, y = symbols('x y')

# Define the function
u = exp(x) *( x*cos(y) - y*sin(y))

# Partial derivatives
dux = diff(u, x)
duy = diff(u, y)

# Mixed partials
duxy = diff(dux, y)
duyx = diff(duy, x)

# Check equality
if duxy == duyx:
    print("Mixed partial derivatives are equal")
else:
    print("Mixed partial derivatives are not equal")
```

Output:

```
Mixed partial derivatives are equal
```

1.2: Write a program to prove that if

$$u = \tan^{-1}\left(\frac{y}{x}\right)$$

then $u_{xx} + u_{yy} = 0$.

Code:

```
from sympy import *

# Define variables
x, y = symbols('x y')

# Define the function
u = atan(y/x)


# First derivatives
dux = diff(u, x)
duy = diff(u, y)

# Second derivatives
uxx = diff(dux, x)    # or diff(u, x, 2)
uyy = diff(duy, y)    # or diff(u, y, 2)

# Sum of second derivatives
w = uxx + uyy

# Simplify result
w1 = simplify(w)

print("Ans:", w1)
```

Output:

```
Ans: 0
```

Exercise 1:If $u = \log(\frac{x^2+y^2}{x+y})$, **write a program show that** $xu_x + yu_y = 1$.

Lab 2: Jacobian of Functions of Several Variables

Aim: To find Jacobian of function of two and three variables.

2.1: Write a program to compute the Jacobian determinant of the transformation

$$u = \frac{xy}{z}, \quad v = \frac{yz}{x}, \quad w = \frac{zx}{y}.$$

Code:

```
from sympy import *

# Define variables
x, y, z = symbols('x y z')

# Define functions
u = x*y/z
v = y*z/x
w = z*x/y

# Partial derivatives
dux = diff(u, x); duy = diff(u, y); duz = diff(u, z)
dvx = diff(v, x); dvy = diff(v, y); dvz = diff(v, z)
dwx = diff(w, x); dwy = diff(w, y); dwz = diff(w, z)

# Construct Jacobian matrix
J = Matrix([[dux, duy, duz],
            [dvx, dvy, dvz],
            [dwx, dwy, dwz]])

print("The Jacobian matrix is:\n")
display(J)

# Determinant of Jacobian
Jac = J.det()
print("\nJ =", Jac)
```

Output:

$$J = \begin{bmatrix} \frac{y}{z} & \frac{x}{z} & -\frac{xy}{z^2} \\ -\frac{yz}{x^2} & \frac{z}{x} & \frac{y}{x} \\ \frac{z}{y} & -\frac{zx}{y^2} & \frac{x}{y} \end{bmatrix}$$

$$J = 4$$

2.2 Write a program to compute the Jacobian matrix and its determinant for

$$u = x + 3y^2 - z^3, \quad v = 4x^2yz, \quad w = 2z^2 - xy$$

and evaluate the determinant at $(x, y, z) = (1, -1, 0)$**.**

Code:

```
from sympy import *

# Define variables
x, y, z = symbols('x y z')

# Define functions
u = x + 3*y**2 - z**3
v = 4*x**2*y*z
w = 2*z**2 - x*y

# Partial derivatives
dux, duy, duz = diff(u, x), diff(u, y), diff(u, z)
dvx, dvy, dvz = diff(v, x), diff(v, y), diff(v, z)
dwx, dwy, dwz = diff(w, x), diff(w, y), diff(w, z)

# Jacobian matrix
J = Matrix([
    [dux, duy, duz],
    [dvx, dvy, dvz],
    [dwx, dwy, dwz]
])

print("The Jacobian matrix is:\n")
display(J)

# Determinant of Jacobian
Jac = J.det()
print("\nJ =")
display(Jac)

# Substituting values (x=1, y=-1, z=0)
J1 = J.subs({x:1, y:-1, z:0})
print("\nJ at (1, -1, 0):\n")
display(J1)

Jac1 = J1.det()
print("\nDeterminant of J at (1, -1, 0):")
display(Jac1)
```

Output:

$$J = \begin{bmatrix} 1 & 6y & -3z^2 \\ 8xyz & 4x^2z & 4x^2y \\ -y & -x & 4z \end{bmatrix}$$

$$J = 32x^3yz + 24xyz^3 + 24x^2y^2z - 32x^2z^2$$

At $(x, y, z) = (1, -1, 0)$:

$$J = \begin{bmatrix} 1 & -6 & 0 \\ 0 & 0 & -4 \\ 1 & -1 & 0 \end{bmatrix}$$

$$\det(J) = 20$$

.

.

Exercise 1:Write a program to find $\dfrac{\partial(u, v, w)}{\partial(x, y, z)}$ **for**

$u = x + 3y^2 - z^3, \quad v = 4x^2yz, \quad w = 2z^2 - xy$ **at the point** $(-2, -1, 1)$.

Lab 3: Finding gradient, divergent, curl and their geometrical interpretation

Aim:

1. to find the gradient of a given scalar function.
2. to find find divergence and curl of a vector function.

3.1: Write a program to find the gradient of

$$\phi(x, y, z) = x^2y + 2xz - 4.$$

Code:

```
from sympy.vector import *
from sympy import simplify

# Setting the coordinate system
N=CoordSys3D('N')
x,y,z=N.x,N.y,N.z

F=x**2*y+2*x*z-4
display("Given scalar function F",F)

# To find the gradient of function
delop=Del()
grad=delop(F)
display("Gradient of F:")
display(grad)
display("Simplified Gradient")
display(simplify(grad))
```

Output:

Given scalar function F:

$$F = x_N^2 y_N + 2x_N z_N - 4$$

Gradient of F:

$$\left(\frac{\partial}{\partial x_N}(x_N^2 y_N + 2x_N z_N - 4)\right)\hat{i}_N + \left(\frac{\partial}{\partial y_N}(x_N^2 y_N + 2x_N z_N - 4)\right)\hat{j}_N + \left(\frac{\partial}{\partial z_N}(x_N^2 y_N + 2x_N z_N - 4)\right)\hat{k}_N$$

Simplified Gradient:

$$(2x_N y_N + 2z_N)\hat{i}_N + (x_N^2)\hat{j}_N + (2x_N)\hat{k}_N$$

3.2: Find the divergence of the vector field

$$\mathbf{A}(x, y, z) = x^2yz\,\mathbf{i} + y^2zx\,\mathbf{j} + z^2xy\,\mathbf{k}$$

Code:

```
from sympy.vector import *
from sympy import simplify

# Define coordinate system
N = CoordSys3D('N')
x, y, z = N.x, N.y, N.z
i, j, k = N.i, N.j, N.k

# Define vector field A
A = x**2*y*z*i + y**2*z*x*j + z**2*x*y*k

# Define Del operator
delop = Del()

# Divergence of A
divA = delop.dot(A)

display("Divergence of A:")
display(divA)

display("Simplified Divergence:")
display(simplify(divA))
```

Output:

Divergence of A:

$$\left(\frac{\partial}{\partial x_N}(x_N^2 y_N z_N)\right) + \left(\frac{\partial}{\partial y_N}(y_N^2 z_N x_N)\right) + \left(\frac{\partial}{\partial z_N}(z_N^2 x_N y_N)\right)$$

Simplified Divergence:

$$6x_N y_N z_N$$

3.3: Find the curl of the vector field

$$\mathbf{A}(x, y, z) = x^2yz\,\mathbf{i} + y^2zx\,\mathbf{j} + z^2xy\,\mathbf{k}$$

Code:

```
from sympy.vector import *
from sympy import symbols, simplify

# Setting the coordinate system
N = CoordSys3D('N')
x, y, z = N.x, N.y, N.z
i, j, k = N.i, N.j, N.k

# Define A as a vector field
A = x**2 * y * z * i + y**2 * z * x * j + z**2 * x * y * k

# To find curl of a vector point function
delop = Del()
curlA = delop.cross(A)

display("Curl of A:")
display(curlA)
display("Simplified Curl of A:")
display(simplify(curlA))
```

Output:

Curl of A:

$$\left(\frac{\partial}{\partial y_N}(z_N^2x_Ny_N) - \frac{\partial}{\partial z_N}(y_N^2z_Nx_N)\right)\mathbf{i} - \left(\frac{\partial}{\partial x_N}(z_N^2x_Ny_N) - \frac{\partial}{\partial z_N}(x_N^2y_Nz_N)\right)\mathbf{j} + \left(\frac{\partial}{\partial x_N}(y_N^2z_Nx_N) - \frac{\partial}{\partial y_N}(x\right.$$

Simplified Curl:

$$(-x_Ny_N^2 + x_Nz_N^2)\,\hat{i}_N + (x_N^2y_N - y_Nz_N^2)\,\hat{j}_N + (-x_N^2z_N + y_N^2z_N)\,\hat{k}_N$$

Exercise 1: If $u = x+y+z,\ v = x^2+y^2+z^2,\ w = yz+zx+xy,$ find ∇u, ∇v and ∇w.

Exercise 2: Prove that the vector $\mathbf{F} = (yz - x^2)\,\hat{i} + (4y - z^2x)\,\hat{j} + (2xz - 4z)\,\hat{k}$ is solenoidal.

Exercise 3: If $\mathbf{R} = x\hat{i} + y\hat{j} + z\hat{k}$, show that:

$$(1) \quad \nabla \cdot \mathbf{R} = 3, \qquad (2) \quad \nabla \times \mathbf{R} = 0.$$

Lab 4: Finding rank, reduced echelon form, solving system of linear equations using Gauss elimination method

Aim:

1. To write a Python program to determine the **rank of a given matrix** .
2. To write a Python program to compute the **Reduced Row Echelon Form (RREF)** of a given matrix .
3. To write a Python program to **solve a system of linear equations** using the **Gauss Elimination method**.

4.1: Write a Python program to determine the **rank of the given matrix**:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 1 & 1 & 1 \end{bmatrix}$$

Code:

```
import sympy as sp

# Input matrix
A = sp.Matrix([[1, 2, 3],
               [2, 4, 6],
               [1, 1, 1]])

# Compute rank
rank_A = A.rank()

print("Matrix A:")
sp.pprint(A)
print("\nRank of A:", rank_A)
```

Output:

```
Matrix A:
[1  2  3]
[2  4  6]
[1  1  1]

Rank of A: 2
```

4.2: Write a Python program to compute the **Reduced Row Echelon Form (RREF)** of the following matrix:

$$A = \begin{bmatrix} 1 & 2 & -1 & -4 \\ 2 & 3 & -1 & -11 \\ -2 & 0 & -3 & 22 \end{bmatrix}$$

Code:

```python
import sympy as sp

# Input matrix
A = sp.Matrix([[1, 2, -1, -4],
               [2, 3, -1, -11],
               [-2, 0, -3, 22]])

# Compute RREF
rref_A, pivots = A.rref()

print("Matrix A:")
sp.pprint(A)
print("\nRREF of A:")
sp.pprint(rref_A)
print("\nPivot Columns:", pivots)
```

Output:

```
Matrix A:
[ 1   2  -1  -4]
[ 2   3  -1 -11]
[-2   0  -3  22]

RREF of A:
[1  0  0  8]
[0  1  0  1]
[0  0  1  -2]

Pivot Columns: (0, 1, 2)
```

4.3: Solve System using Gauss Elimination

$$\begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 8 \\ -11 \\ -3 \end{bmatrix}$$

Code:

```python
import numpy as np

# Coefficient matrix
A = np.array([[2, 1, -1],
              [-3, -1, 2],
              [-2, 1, 2]], dtype=float)

# Constants vector
b = np.array([8, -11, -3], dtype=float)

n = len(b)

print("Original system:")
print("A =")
```

```
print(A)
print("b =", b)
print()

# Forward elimination
for i in range(n):
    print(f"Step {i+1}: Making diagonal element at position ({i},{i}) = 1")

    # Store the pivot value before modifying the row
    pivot = A[i, i]

    # Make the diagonal element 1
    A[i] = A[i] / pivot
    b[i] = b[i] / pivot

    print(f"After making pivot 1:")
    print("A =")
    print(A)
    print("b =", b)

    # Eliminate below
    for j in range(i+1, n):
        factor = A[j, i]
        A[j] = A[j] - factor * A[i]
        b[j] = b[j] - factor * b[i]

    print(f"After elimination below row {i}:")
    print("A =")
    print(A)
    print("b =", b)
    print()

# Back substitution
x = np.zeros(n)
for i in range(n-1, -1, -1):
    x[i] = b[i] - np.dot(A[i, i+1:], x[i+1:])
    print(f"x[{i}] = {x[i]}")

print("\nFinal solution:")
print("x =", x)

# Verification
print("\nVerification (A*x should equal b):")
print("A*x =", np.dot(A, x))
print("Original b =", b)
```

Output

```
Original system:
```

```
A =
[[ 2.  1. -1.]
 [-3. -1.  2.]
 [-2.  1.  2.]]
b = [  8. -11.  -3.]

Step 1: Making diagonal element at position (0,0) = 1
After making pivot 1:
A =
[[ 1.   0.5 -0.5]
 [-3.  -1.   2. ]
 [-2.   1.   2. ]]
b = [  4. -11.  -3.]
After elimination below row 0:
A =
[[ 1.   0.5 -0.5]
 [ 0.   0.5  0.5]
 [ 0.   2.   1. ]]
b = [4. 1. 5.]

Step 2: Making diagonal element at position (1,1) = 1
After making pivot 1:
A =
[[ 1.   0.5 -0.5]
 [ 0.   1.   1. ]
 [ 0.   2.   1. ]]
b = [4. 2. 5.]
After elimination below row 1:
A =
[[ 1.   0.5 -0.5]
 [ 0.   1.   1. ]
 [ 0.   0.  -1. ]]
b = [4. 2. 1.]

Step 3: Making diagonal element at position (2,2) = 1
After making pivot 1:
A =
[[ 1.   0.5 -0.5]
 [ 0.   1.   1. ]
 [-0.  -0.   1. ]]
b = [ 4.  2. -1.]
After elimination below row 2:
A =
[[ 1.   0.5 -0.5]
 [ 0.   1.   1. ]
 [-0.  -0.   1. ]]
b = [ 4.  2. -1.]
```

```
x[2] = -1.0
x[1] = 3.0
x[0] = 2.0

Final solution:
x = [ 2.  3. -1.]

Verification (A*x should equal b):
A*x = [ 4.  2. -1.]
Original b = [ 4.  2. -1.]
```

Lab 5: Test the Consistency of a System of Linear Equations

Aim: To write a Python program to test the consistency of a system of linear equations using the rank of the coefficient matrix and the augmented matrix.

5.1: Test the consistency the following system of linear equations:

$$x + 2y + 3z = 6$$
$$2x + 4y + 6z = 12$$
$$x + y + z = 3$$

Code:

```
1
2  import numpy as np
3
4  # Coefficient matrix A
5  A = np.array([
6      [1, 2, 3],
7      [2, 4, 6],
8      [1, 1, 1]
9  ], dtype=float)
10
11 # Constant matrix B
12 B = np.array([
13     [6],
14     [12],
15     [3]
16 ], dtype=float)
17
18 # Number of variables
19 n = A.shape[1]
20
21 # Augmented matrix
22 Aug = np.hstack((A, B))
23
24 # Print matrices
25 print("Coefficient Matrix A:")
26 print(A)
27
28 print("\nAugmented Matrix [A | B]:")
29 print(Aug)
30
31 # Compute ranks
32 rank_A = np.linalg.matrix_rank(A)
33 rank_Aug = np.linalg.matrix_rank(Aug)
34
35 print("\nRank of A =", rank_A)
36 print("Rank of [A | B] =", rank_Aug)
```

```
37
38 # Consistency test using if-else
39 if rank_A == rank_Aug:
40     print("\nSystem is CONSISTENT")
41
42     if rank_A == n:
43         print("Unique solution")
44     else:
45         print(" Infinitely many solutions")
46 else:
47     print("\nSystem is INCONSISTENT (No solution)")
```

Output:

```
Coefficient Matrix A:
[[1. 2. 3.]
 [2. 4. 6.]
 [1. 1. 1.]]

Augmented Matrix [A | B]:
[[ 1.  2.  3.  6.]
 [ 2.  4.  6. 12.]
 [ 1.  1.  1.  3.]]

Rank of A = 2
Rank of [A | B] = 2

System is CONSISTENT
 Infinitely many solutions
```

Lab 6: Determine Eigenvalues and Eigenvectors

Aim: To write a Python program to determine the eigenvalues and eigenvectors of a given square matrix using `numpy.linalg.eig()`.

6.1: Determine the eigenvalues and eigenvectors of the matrix

$$A = \begin{bmatrix} 2 & 1 & -1 \\ -3 & -1 & 2 \\ -2 & 1 & 2 \end{bmatrix}.$$

Code:

```
import numpy as np

A = np.array([[ 2.,  1., -1.],
              [-3., -1.,  2.],
              [-2.,  1.,  2.]])

# Compute eigenvalues and right eigenvectors
w, v = np.linalg.eig(A)

print("Eigenvalues:\n", w)
print("\nEigenvectors (columns):\n", v)
```

Output

```
Eigenvalues:
 [ 3.21431974  0.46081068 -0.67513042]

Eigenvectors (columns):
 [[-0.22113631  0.58085679  0.4669257 ]
 [ 0.54213894 -0.0844347  -0.65511165]
 [ 0.81066862  0.80961414  0.59397557]]
```

.

Lab 7: Linearly Independence and Dependence sets

Aim: To write a Python program to check whether a given set of vectors is linearly independent or linearly dependent using the rank of the matrix.

7.1: Check whether the following set of vectors is linearly independent or dependent:

$$\Big\{ (1,4,7),\ (2,5,8),\ (3,6,9) \Big\}$$

Code:

```
import numpy as np

# Define matrix with vectors as columns
A = np.array([[1, 4, 7],
              [2, 5, 8],
              [3, 6, 9]])

# Find rank
rank = np.linalg.matrix_rank(A)

# Check linear independence
if rank == A.shape[1]:
    print("The set of vectors is Linearly Independent")
else:
    print("The set of vectors is Linearly Dependent")
```

Output

```
The set of vectors is Linearly Dependent
```

Lab 8: Basis and dimension

Aim: To write a Python program to determine the basis and dimension of a vector space spanned by a given set of vectors.

8.1:Find the basis and dimension of the vector space spanned by:

$$\Big\{ (1,2,3),\ (2,4,6),\ (1,1,1) \Big\}$$

Code:

```
from sympy import Matrix

# Define matrix with vectors as columns
A = Matrix([[1, 2, 3],
            [2, 4, 6],
            [1, 1, 1]])

# Column space gives basis
basis = A.columnspace()

# Dimension is the rank
dimension = A.rank()

print("Basis of the vector space:")
for vec in basis:
    print(vec)

print("Dimension of the vector space:", dimension)
```

Output

```
Basis of the vector space:
Matrix([[1], [2], [1]])
Matrix([[2], [4], [6]])

Dimension of the vector space: 2
```

Lab 9: Linear transformation-range space and null space

Aim: To find the range space, null space, rank and nullity of a matrix representing a linear transformation.

9.1: Determine the range space, null space, rank and nullity of the following matrix:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 1 & 1 & 1 \end{bmatrix}$$

Code:

```
from sympy import Matrix

# Define matrix for linear transformation
A = Matrix([[1, 2, 3],
            [2, 4, 6],
            [1, 1, 1]])

print("Matrix A:")
print(A)

# Range space (column space / image)
range_space = A.columnspace()
print("\nRange Space (Column Space):")
for v in range_space:
    print(v)

# Null space (kernel)
null_space = A.nullspace()
print("\nNull Space:")
for v in null_space:
    print(v)

# Dimensions
print("\nDimension of Range Space (Rank):", A.rank())
print("Dimension of Null Space (Nullity):", len(null_space))
```

Output:

```
Matrix A:
Matrix([[1, 2, 3], [2, 4, 6], [1, 1, 1]])

Range Space (Column Space):
Matrix([[1], [2], [1]])
Matrix([[2], [4], [1]])

Null Space:
Matrix([[-2], [1], [0]])
```

```
Matrix([[-3], [0], [1]])

Dimension of Range Space (Rank): 2
Dimension of Null Space (Nullity): 1
```

Lab 10: Verification of the rank nullity theorem

Aim: To verify the Rank–Nullity Theorem for a given matrix by computing the rank and nullity and checking that

$$\text{rank}(A) + \text{nullity}(A) = \text{number of columns of } A.$$

10.1: Verify the Rank–Nullity Theorem for the matrix

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 1 & 1 & 1 \end{bmatrix}.$$

Code:

```
from sympy import Matrix

# Define matrix A
A = Matrix([[1, 2, 3],
            [2, 4, 6],
            [1, 1, 1]])

# Compute rank and null space
rankA = A.rank()
null_space = A.nullspace()
nullityA = len(null_space)
num_cols = A.shape[1]

# Print results
print("Matrix A:")
print(A)

print("\nRank(A):", rankA)
print("Null space basis vectors:")
for v in null_space:
    print(v)
print("Nullity(A):", nullityA)

# Verify Rank-Nullity Theorem using if-else
print("\nNumber of columns:", num_cols)
if rankA + nullityA == num_cols:
    print("Rank-Nullity Theorem holds: True")
else:
    print("Rank-Nullity Theorem holds: False")
```

Output

```
Matrix A:
```

```
Matrix([[1, 2, 3], [2, 4, 6], [1, 1, 1]])

Rank(A): 2
Null space basis vectors:
Matrix([[-2], [1], [0]])
Nullity(A): 1

Number of columns: 3
Rank-Nullity Theorem holds: True
```