# 📘 Module-4: ODE Solving and Numerical Integration using SciPy

## 🔹 Introduction

Scientific computing often involves solving differential equations and performing numerical integration where analytical solutions are difficult or impossible. Python's **SciPy library**, especially its `integrate` subpackage, provides powerful tools to:

- Solve **ordinary differential equations (ODEs)** with different solvers like **RK45** and **LSODA**
- Perform **definite integration** using **Gaussian quadrature** methods such as `quad` and `quadrature`
- Apply **numerical methods** such as the **trapezoidal rule**, **Simpson's rule**, and **Romberg integration**

This module introduces learners to practical applications of these techniques using real-world function examples.

---

# 💻 Python Programmes

## 🔹 1. Solve ODE using `solve_ivp` with RK45

```
from scipy.integrate import solve_ivp
def dydt(t, y): return -2 * y + 1
sol = solve_ivp(dydt, [0, 5], [0], method='RK45')
print("t values:", sol.t)
print("y values:", sol.y[0])
```

---

## 🔹 2. Solve ODE using `solve_ivp` with LSODA

```
from scipy.integrate import solve_ivp
def dydt(t, y): return -2 * y + 1
sol = solve_ivp(dydt, [0, 5], [0], method='LSODA')
print("t values:", sol.t)
print("y values:", sol.y[0])
```

---

## 🔹 3. Plot ODE Solution using Matplotlib

```
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
def dydt(t, y): return -2 * y + 1
sol = solve_ivp(dydt, [0, 5], [0], method='RK45')
plt.plot(sol.t, sol.y[0])
plt.xlabel('t')
plt.ylabel('y')
```

```
plt.title('ODE Solution using RK45')
plt.grid()
plt.show()
```

---

### 🔹 4. System of ODEs (Coupled Equations)

```
from scipy.integrate import solve_ivp
def system(t, y):
    dydt = [y[1], -0.5*y[1] - 4*y[0]]
    return dydt
sol = solve_ivp(system, [0, 10], [2, 0], method='RK45')
print("Solution:\n", sol.y)
```

---

### 🔹 5. Gaussian Quadrature using `quad`

```
from scipy.integrate import quad
import numpy as np
result, _ = quad(np.sin, 0, np.pi)
print("Integral of sin(x) from 0 to pi:", result)
```

---

### 🔹 6. Gaussian Quadrature using `quadrature`

```
from scipy.integrate import quadrature
import numpy as np
result, _ = quadrature(np.exp, 0, 1)
print("Integral of exp(x) from 0 to 1:", result)
```

---

### 🔹 7. Trapezoidal Rule using `trapezoid`

```
from scipy.integrate import trapezoid
import numpy as np
x = np.linspace(0, np.pi, 100)
y = np.sin(x)
area = trapezoid(y, x)
print("Trapezoidal area under sin(x):", area)
```

---

### 🔹 8. Simpson's 1/3 Rule using `simpson`

```
from scipy.integrate import simpson
import numpy as np
x = np.linspace(0, np.pi, 100)
y = np.sin(x)
area = simpson(y, x)
print("Simpson's Rule area under sin(x):", area)
```

---

### 🔹 9. Romberg Integration using `romb`

```
from scipy.integrate import romb
import numpy as np
x = np.linspace(0, 1, 129)  # 2^n + 1 samples required
y = np.exp(-x**2)
area = romb(y, dx=x[1]-x[0])
print("Romberg integration result:", area)
```

---

### 🔹 10. Integrate a custom function

```
from scipy.integrate import quad
def f(x): return x**2 + 2*x + 1
result, _ = quad(f, 0, 2)
print("Integral of x^2 + 2x + 1 from 0 to 2:", result)
```

---

### 🔹 11. Integrate with infinite limits

```
from scipy.integrate import quad
import numpy as np
result, _ = quad(lambda x: np.exp(-x**2), -np.inf, np.inf)
print("Gaussian integral:", result)
```

---

### 🔹 12. Improper Integral using `quad`

```
from scipy.integrate import quad
import numpy as np
result, _ = quad(lambda x: 1/np.sqrt(x), 0, 1)
print("Integral of 1/sqrt(x) from 0 to 1:", result)
```

---

### 🔹 13. Multiple Evaluations using Loop

```
from scipy.integrate import quad
for i in range(1, 6):
    res, _ = quad(lambda x: x**i, 0, 1)
    print(f"Integral of x^{i} from 0 to 1:", res)
```

---

### 🔹 14. ODE: Exponential Growth Model

```
from scipy.integrate import solve_ivp
def dydt(t, y): return 0.5 * y
sol = solve_ivp(dydt, [0, 10], [1], method='RK45')
print("Exponential growth values:\n", sol.y[0])
```

---

### 🔹 15. Compare Trapezoidal vs Simpson

```
import numpy as np
from scipy.integrate import trapezoid, simpson
x = np.linspace(0, np.pi, 50)
```

```
y = np.sin(x)
trap = trapezoid(y, x)
simp = simpson(y, x)
print("Trapezoidal:", trap)
print("Simpson:", simp)
```