Welcome to Python Programming!

Teaching Assistant of Introduction to Numerical Analysis Class

Class Date: 2024-2025

Instructor: Dr. Mina Zarei

Class Link: Telegram group.

Exercise class time: **Monday at 9:30 P002 Class** (determined by student vote)

Session 1 (Install Python in VS Code)

Follow the instructions in the GitHub link.

link: www.github.com/AliSeif96/Introduction-to-Numerical-Analysis/).

---

session 2 (Basic training)

1. Variables and Data Types
2. Taking Output and Input from the User
3. Conditionals (if, elif, else)
4. Loops
5. Functions

---

session 3 (Basic training 2)

6. Lists
7. Arrays
8. Matrices
9. Using `NumPy`

---

session 4 (Root finding)

10.1. Bisection Method (روش تصنیف)

10.2. False Position Method (روش نابجایی)

10.3. Newton-Raphson Method (روش نیوتن رافسون)

10.4. Secant Method (روش وتری)

10.5. Fixed-Point Iteration (روش تکرار ساده)

10.6. Brent's Method

---

session 5 (Solving and displaying the Kuramoto model)

11. Matplotlib
12. Kuramoto model

---

session 6 (File Handling and Runge-Kutta 4th)

13. File Handling
14. Runge-Kutta 4th

In [6]:
```python
f = open("demofile.txt")
```

`"r" - Read - Default value. Opens a file for reading, error if the file does not exist`

`"a" - Append - Opens a file for appending, creates the file if it does not exist`

`"w" - Write - Opens a file for writing, creates the file if it does not exist`

`"x" - Create - Creates the specified file, returns an error if the file exists`

In [11]:
```python
f = open("demofile.txt", "r")
print(f.read())
```

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!
```

In [12]:
```python
f = open("D:/TA/session 6 (Runge-Kutta 4th)/Hello.txt", "r")
print(f.read())
```

```
If the file is located in a different location,
you will have to specify the file path
```

In [23]:
```python
# Read Only Parts of the File
f = open("demofile.txt", "r")
print(f.read(7))
print(f.read(8))
print(f.read(12))
```

```
Hello!
Welcome
to demofile.
```

In [25]:
```python
# Read Lines
f = open("demofile.txt", "r")
print(f.readline())
```

```
Hello! Welcome to demofile.txt
```

Test:

`f = open("demofile.txt", "r")`

`print(f.readline())`

`print(f.readline())`

In [38]:
```python
# By looping through the lines of the file, you can read the whole file, line by line:
f = open("demofile.txt", "r")
for x in f:
    print(x)



f = open("demofile.txt", "r")
i=0
for x in f:
    print("number of loop: ",i)
    print(x)
    i+=1
```

```
Hello! Welcome to demofile.txt

This file is for testing purposes.

Good Luck!
number of loop:  0
Hello! Welcome to demofile.txt

number of loop:  1
This file is for testing purposes.

number of loop:  2
Good Luck!
```

In [36]:
```python
f = open("demofile.txt", "r")
for x in f.readline():
    print(x)
```

```
H
e
l
l
o
!

W
e
l
c
o
m
e

t
o

d
e
m
o
f
i
l
e
.
t
x
t
```

In [39]:
```python
# Close Files
f = open("demofile.txt", "r")
print(f.readline())
f.close()
```

```
Hello! Welcome to demofile.txt
```

In [40]:
```python
# Write to an Existing File

f = open("demofile2.txt", "a")
f.write("Now the file has more content!")
f.close()

#open and read the file after the appending:
f = open("demofile2.txt", "r")
print(f.read())
```

```
Hello! Welcome to demofile.txt
This file is for testing purposes.
Good Luck!Now the file has more content!
```

In [42]:
```python
#  overwrite the content

f = open("demofile3.txt", "w")
f.write("Woops! I have deleted the content!")
f.close()

#open and read the file after the overwriting:
f = open("demofile3.txt", "r")
print(f.read())
```

```
Woops! I have deleted the content!
```

In [49]:
```python
# Create a New File
f = open("myfile1.txt", "x")
```

In [47]:
```python
# Create a new file if it does not exist:
f = open("myfile2.txt", "w")
```

حل عددی و تحلیلی یک معادله دیفرانسیل ساده

معادله دیفرانسیل

معادله دیفرانسیل موردنظر به‌صورت زیر تعریف می‌شود:

$$\frac{dy}{dt} = -2y$$

با مقدار اولیه:

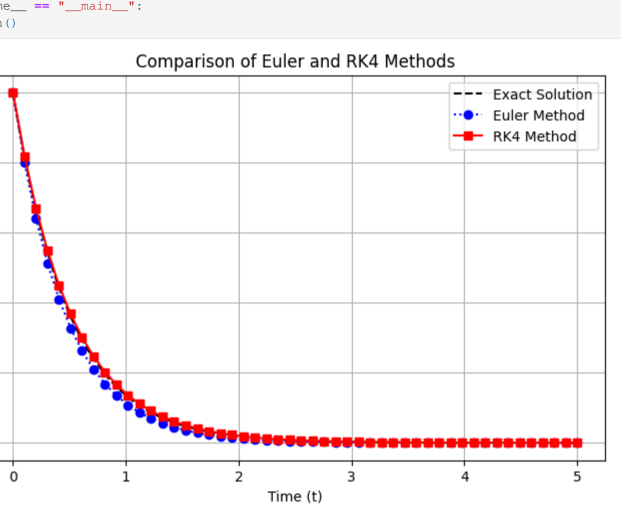
$$y(0) = 1$$

حل تحلیلی:

حل دقیق این معادله به‌صورت زیر است:

$$y(t) = e^{-2t}$$

In [ ]:
```python
import numpy as np
import matplotlib.pyplot as plt

# معادله دیفرانسیل
def dydt(y):
    """
    Defines the differential equation dy/dt = -2y.

    Parameters:
        y (float): The current value of y.

    Returns:
        float: The rate of change dy/dt.
    """
    return -2 * y

# روش اویلر
def euler_method(y0, dt, T):
    """
    Solves the differential equation dy/dt = -2y using the Euler method.

    Parameters:
        y0 (float): Initial value of y at t=0.
        dt (float): Time step for the numerical solution.
        T (float): Total simulation time.

    Returns:
        t_vals (numpy array): Array of time points.
        y_vals (numpy array): Array of y values computed using the Euler method.
    """
    steps = int(T / dt)
    y_vals = np.zeros(steps)
    t_vals = np.linspace(0, T, steps)
    y_vals[0] = y0

    for i in range(1, steps):
        y_vals[i] = y_vals[i - 1] + dydt(y_vals[i - 1]) * dt                              # فرمول اویلر

    return t_vals, y_vals

# روش رانگ-کوتا مرتبه چهارم
def rk4_method(y0, dt, T):
    """
    Solves the differential equation dy/dt = -2y using the 4th-order Runge-Kutta method (RK4).

    Parameters:
        y0 (float): Initial value of y at t=0.
        dt (float): Time step for the numerical solution.
        T (float): Total simulation time.

    Returns:
        t_vals (numpy array): Array of time points.
        y_vals (numpy array): Array of y values computed using the RK4 method.
    """
    steps = int(T / dt)
    y_vals = np.zeros(steps)
    t_vals = np.linspace(0, T, steps)
    y_vals[0] = y0

    for i in range(1, steps):
        k1 = dydt(y_vals[i - 1])
        k2 = dydt(y_vals[i - 1] + 0.5 * dt * k1)
        k3 = dydt(y_vals[i - 1] + 0.5 * dt * k2)
        k4 = dydt(y_vals[i - 1] + dt * k3)

        y_vals[i] = y_vals[i - 1] + (dt / 6) * (k1 + 2*k2 + 2*k3 + k4)                  # فرمول RK4

    return t_vals, y_vals



def main():
    # مقدار اولیه و تنظیمات
    y0 = 1.0
    dt = 0.1
    T = 5

    # حل معادله به دو روش
    t_euler, y_euler = euler_method(y0, dt, T)                                          # روش اویلر
    t_rk4, y_rk4 = rk4_method(y0, dt, T)                                                # روش رانگ-کوتا مرتبه چهارم

    # حل تحلیلی
    t_exact = np.linspace(0, T, 100)
    y_exact = np.exp(-2 * t_exact)

    # رسم نمودار
    plt.figure(figsize=(8, 5))
    plt.plot(t_exact, y_exact, label="Exact Solution", linestyle="dashed", color="black")            # حل تحلیلی
    plt.plot(t_euler, y_euler, label="Euler Method", marker="o", linestyle="dotted", color="blue")  # روش اویلر
    plt.plot(t_rk4, y_rk4, label="RK4 Method", marker="s", linestyle="solid", color="red")           # روش رانگ-کوتا مرتبه چهارم
    plt.xlabel("Time (t)")
    plt.ylabel("y(t)")
    plt.title("Comparison of Euler and RK4 Methods")
    plt.legend()
    plt.grid()
    plt.show()

if __name__ == "__main__":
    main()                                                                              # اجرای تابع اصلی
```



In [54]:
```python
y_euler
```

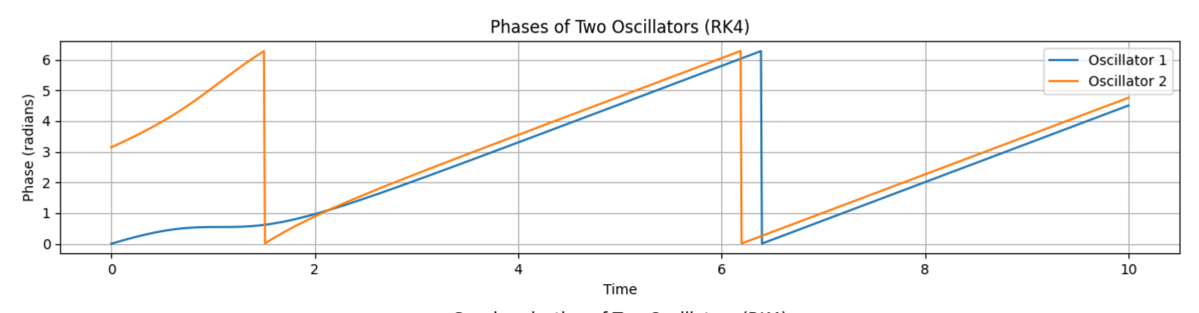
Out[54]:
```
array([1.        , 0.8       , 0.64      , 0.512     , 0.4096    ,
       0.32768   , 0.262144  , 0.2097152 , 0.16777216, 0.13421773,
       0.10737418, 0.08589935, 0.06871948, 0.05497558, 0.04398047,
       0.03518437, 0.0281475 , 0.022518  , 0.0180144 , 0.01441152])
```

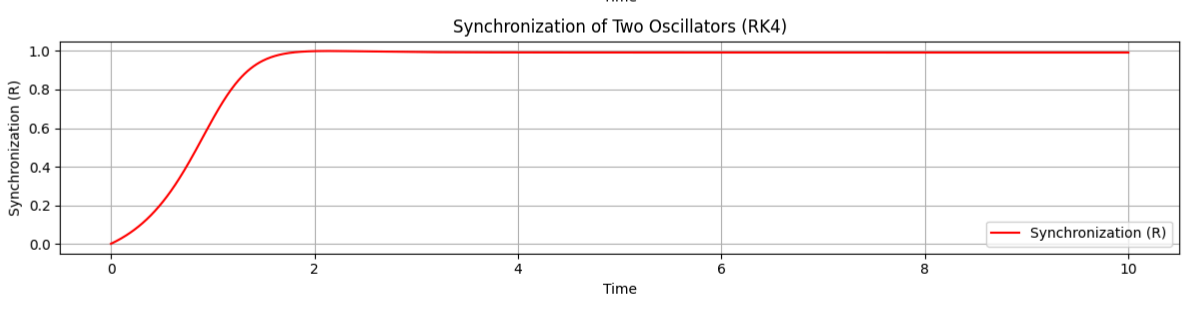
In [55]:
```python
y_rk4
```

Out[55]:
```
array([1.        , 0.81873333, 0.67032427, 0.54881682, 0.44933463,
       0.36788524, 0.30119991, 0.2466024 , 0.20190161, 0.16530358,
       0.13533955, 0.110807  , 0.09072138, 0.07427662, 0.06081275,
       0.04978942, 0.04076426, 0.03337506, 0.02732527, 0.02237211])
```

شبیه‌سازی مدل کوراموتو برای دو نوسانگر با روش رانگ-کوتا مرتبه چهارم

In [ ]:
```python
import numpy as np  # کتابخانه عددی برای محاسبات ریاضی
import matplotlib.pyplot as plt  # کتابخانه برای رسم نمودار

def kuramoto_rk4(theta1, theta2, omega1, omega2, K, dt, T):
    """
    شبیه‌سازی مدل کوراموتو برای دو نوسانگر با روش رانگ-کوتا مرتبه چهارم.
    """
    steps = int(T / dt)  # تعداد گام‌های زمانی
    time = np.linspace(0, T, steps)  # ایجاد آرایه زمان

    # ایجاد آرایه‌ها برای ذخیره مقادیر فازها
    theta1_vals = np.zeros(steps)  # مقدارهای فاز نوسانگر 1
    theta2_vals = np.zeros(steps)  # مقدارهای فاز نوسانگر 2
    sync_vals = np.zeros(steps)  # مقدارهای همزمانی

    # تنظیم شرایط اولیه
    theta1_vals[0] = theta1  # مقدار اولیه فاز نوسانگر 1
    theta2_vals[0] = theta2  # مقدار اولیه فاز نوسانگر 2

    # حلقه برای اجرای روش رانگ-کوتا مرتبه چهارم
    for i in range(1, steps):
        def dtheta1_dt(theta1, theta2):
            return omega1 + K * np.sin(theta2 - theta1)  # معادله تغییر فاز نوسانگر 1

        def dtheta2_dt(theta1, theta2):
            return omega2 + K * np.sin(theta1 - theta2)  # معادله تغییر فاز نوسانگر 2

        # محاسبه مقادیر واسط در روش رانگ-کوتا
        k1_1 = dtheta1_dt(theta1_vals[i-1], theta2_vals[i-1])  # مقدار k1 برای نوسانگر 1
        k1_2 = dtheta2_dt(theta1_vals[i-1], theta2_vals[i-1])  # مقدار k1 برای نوسانگر 2

        k2_1 = dtheta1_dt(theta1_vals[i-1] + 0.5 * dt * k1_1, theta2_vals[i-1] + 0.5 * dt * k1_2)  # مقدار k2 برای نوسانگر 1
        k2_2 = dtheta2_dt(theta1_vals[i-1] + 0.5 * dt * k1_1, theta2_vals[i-1] + 0.5 * dt * k1_2)  # مقدار k2 برای نوسانگر 2

        k3_1 = dtheta1_dt(theta1_vals[i-1] + 0.5 * dt * k2_1, theta2_vals[i-1] + 0.5 * dt * k2_2)  # مقدار k3 برای نوسانگر 1
        k3_2 = dtheta2_dt(theta1_vals[i-1] + 0.5 * dt * k2_1, theta2_vals[i-1] + 0.5 * dt * k2_2)  # مقدار k3 برای نوسانگر 2

        k4_1 = dtheta1_dt(theta1_vals[i-1] + dt * k3_1, theta2_vals[i-1] + dt * k3_2)  # مقدار k4 برای نوسانگر 1
        k4_2 = dtheta2_dt(theta1_vals[i-1] + dt * k3_1, theta2_vals[i-1] + dt * k3_2)  # مقدار k4 برای نوسانگر 2

        # به‌روزرسانی مقدار فاز نوسانگرها با استفاده از مقادیر k
        theta1_vals[i] = theta1_vals[i-1] + (dt / 6) * (k1_1 + 2*k2_1 + 2*k3_1 + k4_1)
        theta2_vals[i] = theta2_vals[i-1] + (dt / 6) * (k1_2 + 2*k2_2 + 2*k3_2 + k4_2)

        # نگه‌داشتن مقدار فازها در بازه [0, 2*pi]
        theta1_vals[i] = theta1_vals[i] % (2 * np.pi)
        theta2_vals[i] = theta2_vals[i] % (2 * np.pi)

        # محاسبه همزمانی نوسانگرها
        cos_sum = np.cos(theta1_vals[i]) + np.cos(theta2_vals[i])
        sin_sum = np.sin(theta1_vals[i]) + np.sin(theta2_vals[i])
        sync_vals[i] = 0.5 * np.sqrt(cos_sum**2 + sin_sum**2)

    return time, theta1_vals, theta2_vals, sync_vals  # بازگشت مقادیر زمان، فازها و همزمانی

def main():
    # تعریف پارامترهای اولیه
    theta1 = 0.0  # مقدار اولیه فاز نوسانگر 1
    theta2 = np.pi  # مقدار اولیه فاز نوسانگر 2
    omega1 = 1.0  # فرکانس طبیعی نوسانگر 1
    omega2 = 1.5  # فرکانس طبیعی نوسانگر 2
    K = 1.0  # شدت کوپلینگ
    dt = 0.01  # گام زمانی شبیه‌سازی
    T = 10  # زمان کل شبیه‌سازی

    # اجرای شبیه‌سازی مدل کوراموتو با روش رانگ-کوتا 4
    time, theta1_vals, theta2_vals, sync_vals = kuramoto_rk4(theta1, theta2, omega1, omega2, K, dt, T)

    # رسم نمودار فازها
    plt.figure(figsize=(12, 6))
    plt.subplot(2, 1, 1)
    plt.plot(time, theta1_vals, label='Oscillator 1')
    plt.plot(time, theta2_vals, label='Oscillator 2')
    plt.xlabel('Time')
    plt.ylabel('Phase (radians)')
    plt.title('Phases of Two Oscillators (RK4)')
    plt.legend()
    plt.grid()

    # رسم نمودار همزمانی
    plt.subplot(2, 1, 2)
    plt.plot(time, sync_vals, label='Synchronization (R)', color='red')
    plt.xlabel('Time')
    plt.ylabel('Synchronization (R)')
    plt.title('Synchronization of Two Oscillators (RK4)')
    plt.legend()
    plt.grid()

    plt.tight_layout()
    plt.show()

if __name__ == "__main__":
    main()  # اجرای تابع اصلی
```


Phases of Two Oscillators (RK4)
Oscillator 1
Oscillator 2
Phase (radians)
Time
0
2
4
6
8
10
0
1
2
3
4
5
6


Synchronization of Two Oscillators (RK4)
Synchronization (R)
Synchronization (R)
Time
0
2
4
6
8
10
0.0
0.2
0.4
0.6
0.8
1.0

In [ ]: