Lab Manual for Numerical Analysis

Lab No. 11

NUMERICAL SOLUTION OF ODES: RUNGE-KUTTA METHODS

Software Engineering Department
Bahria University (Karachi Campus)

BAHRIA UNIVERSITY KARACHI CAMPUS
Department of Software Engineering

NUMERICAL ANALYSIS

LAB EXPERIMENT # 11

Numerical Solution of ODEs: Runge-Kutta methods

OBJECTIVE:

The aim of this lab is to introduce **Runge-Kutta methods** for numerically solving **ordinary differential equations (ODEs),** focusing on their accuracy enhancements and exploring their applications in comparison to other numerical methods.

Introduction

In the previous lab, we extensively covered the **Euler method**, a fundamental numerical technique used for approximating solutions to ordinary differential equations (ODEs). This method involves stepwise iteration through the ODE based on its derivative at a given point, advancing the solution incrementally.

While the Euler method provides a straightforward approach, it's essential to acknowledge its limitations, especially regarding accuracy and stability, particularly when dealing with stiff equations or complex dynamics. Building upon this foundation, our focus in this lab will shift to exploring and analyzing the **Runge-Kutta (RK) methods**, specifically delving into RK methods of **2^{nd}, 3^{rd}** and **4^{th}** order

Implementation of Runga-Kutta methods:

The **Runge-Kutta (R-K)** technique is an efficient and commonly used approach for solving initial-value problems of differential equations. It's used to generate high-order accurate numerical methods without the necessity for high-order derivatives of functions.

It offers various orders of accuracy, with the first order equivalent to Euler's method. In this section, we'll focus on implementing the higher-order **RK2, RK3**, and **RK4** methods in Python. These higher orders provide increased accuracy compared to Euler's method, enabling more precise solutions to ordinary differential equations (ODEs) through iterative computations within each interval.

a. 2^{nd} Order Runge – Kutta Method

Runge-Kutta 2^{nd} order method is a numerical technique used to approximate solutions to ordinary differential equations (ODEs). It operates by computing the slope at the beginning and midpoint of a small interval, then using a weighted average of these slopes to estimate the

next value of the function. Only first-order ordinary differential equations can be solved by using this method.

The following series of formulae are involved in calculating 2nd order R-K method while considering an ordinary differential equation $\frac{dy}{dx}$ = **f(x,y)** with the initial condition **$y(x_0) = y_0$.**

$$y_1 = y_0 + \frac{1}{2}(k_1 + k_2)$$

Here:

$$k_1 = h \cdot f(x_0, y_0)$$

$$k_2 = h \cdot f(x_0 + h, y_0 + k_1)$$

Implementation in Python

- The following code provides a function **runge_kutta_2nd_order()** to solve the ordinary differential equation **x + y - 2** at **y (2)** with initial condition **y (0) = 1** and step size **h = 0.2**.

```
# Function representing the differential equation (x + y - 2)
def function(x, y):
    return (x + y - 2)

# Runge-Kutta 2nd order method implementation
def runge_kutta_2nd_order(x0, y0, x, h):
    # Number of steps
    n = int((x - x0) / h)
    y = y0

    # Iterating through each step using RK 2nd order
    for i in range(1, n+1):
        # Calculating slopes k1 and k2
        k1 = h * function(x0, y)
        k2 = h * function(x0 + h, y + k1)

        # Computing the next value of y using the RK method
        y = y + 0.5 * (k1 + k2)
        x0 += h  # Updating x0 for the next iteration
    return y

# Given conditions
x0 = 0
y0 = 1
x = 2
h = 0.2

# Solving the ODE using Runge-Kutta 2nd order method
print("y(x) = ", runge_kutta_2nd_order(x0, y0, x, h))
```

b. 3rd Order Runge – Kutta Method

The third-order Runge-Kutta method **(RK3)** is another numerical technique used to approximate solutions to ordinary differential equations (ODEs). It provides a higher level of accuracy compared to lower-order methods by incorporating additional intermediate steps in the approximation process.

Again considering an ordinary differential equation $\frac{dy}{dx}$ **= f(x,y)** with the initial condition **$y(x_0) = y_0$,** this level of R-K method comprises the following:

$$y_1 = y_0 + \frac{1}{6}(k_1 + 4k_2 + k_3)$$

Here,

$$k_1 = h \cdot f(x_0, y_0)$$

$$k_2 = h \cdot f\left(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}\right)$$

$$k_3 = h \cdot f(x_0 + h, y_0 + 2k_2 - k_1)$$

Implementation in Python

- The following code provides a function **runge_kutta_3rd_order()** to solve the ordinary differential equation **x + y - 2** at **y (2)** with initial condition **y (0) = 1** and step size **h = 0.2**.

```
# Function representing the differential equation (x + y - 2)
def function(x, y):
    return (x + y - 2)

# Runge-Kutta 3rd order method implementation
def runge_kutta_3rd_order(x0, y0, x, h):
    # Number of steps
    n = int((x - x0) / h)
    y = y0

    # Iterating through each step using RK 3rd order
    for i in range(1, n+1):
        # Calculating slopes k1 and k2
        k1 = h * function(x0, y)
        k2 = h * function(x0 + (h/2), y + (k1/2))
        k3 = h * function(x0 + h, y + k2)

        # Computing the next value of y using the RK method
        y = y + (1/6) * (k1 + 4*k2 + k3)
        x0 += h  # Updating x0 for the next iteration
    return y
```

```
# Given conditions
x0 = 0
y0 = 1
x = 2
h = 0.2

# Solving the ODE using Runge-Kutta 3rd order method
print("y(x) = ", runge_kutta_3rd_order(x0, y0, x, h))
```

c. 4th Order Runge – Kutta Method

The fourth-order Runge-Kutta method (RK4) is a widely used numerical technique for approximating solutions to ordinary differential equations (ODEs). It provides higher accuracy by considering four intermediate steps within each interval.

Again considering an ordinary differential equation $\frac{dy}{dx}$ = f(x,y) with the initial condition $y(x_0) = y_0$, equations for calculating 4^{th} order are stated below:

$$y_1 = y_0 + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Here,

$$k_1 = h \cdot f(x_0, y_0)$$

$$k_2 = h \cdot f\left(x_0 + \frac{h}{2}, y_0 + \frac{k_1}{2}\right)$$

$$k_3 = h \cdot f\left(x_0 + \frac{h}{2}, y_0 + \frac{k_2}{2}\right)$$

$$k_4 = h \cdot f(x_0 + h, y_0 + k_3)$$

Where:

- h is the interval size
- k_1 is the slope at the beginning using y
- k_2 is the midpoint slop using y and k_1
- k_3 is again the midpoint slope using y and k_2
- k_4 is the slope at the end of the interval using y and k_3

Implementation in Python

- The following code provides a function **runge_kutta_4th_order()** to solve the ordinary differential equation **x + y - 2** at **y (2)** with initial condition **y (0) = 1** and step size **h = 0.2**.

```
# Function representing the differential equation (x + y - 2)
def function(x, y):
    return (x + y - 2)

# Runge-Kutta 4th order method implementation
def runge_kutta_4th_order(x0, y0, x, h):
    # Number of steps
    n = int((x - x0) / h)
    y = y0

    # Iterating through each step using RK 4th order
    for i in range(1, n+1):
        # Calculating slopes k1 and k2
        k1 = h * function(x0, y)
        k2 = h * function(x0 + (h/2), y + (k1/2))
        k3 = h * function(x0 + (h/2), y + (k2/2))
        k4 = h * function(x0 + h, y + k3)

        # Computing the next value of y using the RK method
        y = y + (1/6) * (k1 + 2*k2 + 2*k3 + k4)
        x0 += h  # Updating x0 for the next iteration
    return y

# Given conditions
x0 = 0
y0 = 1
x = 2
h = 0.2

# Solving the ODE using Runge-Kutta 4th order method
print("y(x) = ", runge_kutta_4th_order(x0, y0, x, h))
```

Lab Tasks

1. Write a Python program that implements range – kutta methods of the orders mentioned against each part to approximate the 1st order derivative of the following:

   a. approximate **y(0.2)** for $y' = \frac{x-y}{2}$, having $x_0 = 0$, $y_0 = 1$, and $h = 0.1$. (2nd order)

   b. approximate **y(0.3)** for $y' = 1 - y^3$, having $x_0 = 0$, $y_0 = 0$, and $h = 0.1$. (3rd order)

   c. approximate **y(0.1)** for $y' = -2y + x^3e^{-2x}$, having $x_0 = 0$, $y_0 = 1$, & $h = 0.1$. (4th order)

a.	0.9146296875
b.	0.19960155661136297
c.	0.8187538028303738