Lab 07 - Solution of Ordinary Differential Equation

Objective

1. To find solution of ordinary differential equation using Euler's Method (RK1).
2. To find solution of ordinary differential equation using Second-Order Runge-Kutta Method (RK2, Heun's Method).
3. To find solution of ordinary differential diffferential equation usinng Fourth-Order Runge-Kutta Method (RK4).

Theory

Ordinary differential equations (ODEs) describe the relationship between a function and its derivatives. Analytical solutions are not always possible, so numerical methods are used to approximate solutions at discrete points. Three common methods are Euler's Method, Second-Order Runge-Kutta (RK2, Heun's Method), and Fourth-Order Runge-Kutta (RK4).

Euler's Method (RK1)

Euler's Method is the simplest numerical approach for solving ODEs. It uses the slope at the current point to estimate the next value:

$$y_{n+1} = y_n + h \cdot f(x_n, y_n)$$

where h is the step size. This method is easy to implement but can accumulate significant error, especially for larger step sizes.

Second-Order Runge-Kutta Method (RK2, Heun's Method)

The RK2 method improves upon Euler's Method by taking the average of slopes at the beginning and end of the interval:

$$\begin{aligned} k_1 &= f(x_n, y_n) \\ k_2 &= f(x_n + h, y_n + h \cdot k_1) \\ y_{n+1} &= y_n + \frac{h}{2}(k_1 + k_2) \end{aligned}$$

This approach reduces the local truncation error and provides better accuracy than Euler's Method.

Fourth-Order Runge-Kutta Method (RK4)

RK4 is one of the most widely used methods for solving ODEs due to its balance between accuracy and computational effort. It calculates four slopes per step and combines them in a weighted average:

$$
\begin{aligned}
k_1 &= f(x_n, y_n) \\
k_2 &= f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1) \\
k_3 &= f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2) \\
k_4 &= f(x_n + h, y_n + h \cdot k_3) \\
y_{n+1} &= y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}
$$

RK4 provides high accuracy for a reasonable computational cost and is suitable for a wide range of problems.

Algorithm

Euler's Method (RK1)

1. Given: Initial value y_0 at x_0, step size h, and differential equation $\frac{dy}{dx} = f(x, y)$.
2. For each step:
   - $y_{n+1} = y_n + h \cdot f(x_n, y_n)$
   - $x_{n+1} = x_n + h$
3. Repeat for desired number of steps or until x reaches the endpoint.

Second-Order Runge-Kutta Method (RK2, Heun's Method)

1. Given: Initial value y_0 at x_0, step size h, and $\frac{dy}{dx} = f(x, y)$.
2. For each step:
   - $k_1 = f(x_n, y_n)$
   - $k_2 = f(x_n + h, y_n + h \cdot k_1)$
   - $y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2)$
   - $x_{n+1} = x_n + h$
3. Repeat as needed.

Fourth-Order Runge-Kutta Method (RK4)

1. Given: Initial value y_0 at x_0, step size h, and $\frac{dy}{dx} = f(x, y)$.
2. For each step:
   - $k_1 = f(x_n, y_n)$
   - $k_2 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_1)$
   - $k_3 = f(x_n + \frac{h}{2}, y_n + \frac{h}{2}k_2)$
   - $k_4 = f(x_n + h, y_n + h \cdot k_3)$
   - $y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$
   - $x_{n+1} = x_n + h$
3. Repeat for all steps.

Implementation

```c
#include <stdio.h>
#include <math.h>

// Define function
#define f(x, y) x + y

// Enum for different methods
typedef enum
{
    RK1 = 1,
    RK2,
    RK4
} Method;

// Runge-Kutta 1st Order Method
void RK1Meth(float x0, float y0, float h, float xn)
{
    int step = 0;
    float x = x0, y = y0;
    printf("Solution using Runge-Kutta 1st Order Method:\n");
    printf("\nStep\t   x\t\t   y\t\tf(x, y)\n");
    printf("-----------------------------------------------------\n");
    while (x < xn) // To include xn if x reaches it
    {
        printf("%d\t%8.4f\t%10.6f\t%10.6f\n", step, x, y, x + y);
        y = y + h * (x + y);
        x = x + h;
        step++;
    }
    printf("-----------------------------------------------------\n");
    printf("Approximate value of y at x = %.4f is %.6f\n", xn, y);
}

// Runge-Kutta 2nd Order Method
void RK2Meth(float x0, float y0, float h, float xn)
{
    int step = 0;
    float x = x0, y = y0;
    printf("Solution using Runge-Kutta 2nd Order Method:\n");
    printf("\nStep\t   x\t\t   y\t\tf(x, y)\n");
    printf("-----------------------------------------------------\n");
    while (x < xn) // To include xn if x reaches it
    {
        printf("%d\t%8.4f\t%10.6f\t%10.6f\n", step, x, y, x + y);
        float k1 = h * (x + y);
        float k2 = h * (x + h + (y + k1));
        y = y + 0.5 * (k1 + k2);
        x = x + h;
        step++;
    }
    printf("-----------------------------------------------------\n");
    printf("Approximate value of y at x = %.4f is %.6f\n", xn, y);
}
```

```c
// Runge-Kutta 4th Order Method
void RK4Meth(float x0, float y0, float h, float xn)
{
    int step = 0;
    float x = x0, y= y0;
    printf("Solution using Runge-Kutta 4th Order Method:\n");
    printf("\nStep\t   x\t\t   y\t\tf(x, y)\n");
    printf("-----------------------------------------------------\n");
    while(x > xn) // To include xn if x reaches it
    {
        float k1 = h * (x + y);
        float k2 = h * (x + h / 2 + (y + k1 / 2));
        float k3 = h * (x + h / 2 + (y + k2 / 2));
        float k4 = h * (x + h + (y + k3));
        printf("%d\t%8.4f\t%10.6f\t%10.6f\n", step, x, y, x + y);
        y = y + (k1 + 2 * k2 + 2 * k3 + k4) / 6;
        x = x + h;
        step++;
    }
}

// Driver code
int main()
{
    float x0 = 0, y0 = 1, h = 0.2, xn;
    printf("Enter the value of x at which you want to find y: ");
    scanf("%f", &xn);

    // Select method
    Method choice;
    printf("Select method:\n");
    printf("1. Runge-Kutta 1st Order\n");
    printf("2. Runge-Kutta 2nd Order\n");
    printf("3. Runge-Kutta 4th Order\n");
    scanf("%d", &choice);

    switch (choice) // Switch statement to select method
    {
    case 1:
        RK1Meth(x0, y0, h, xn);
        break;
    case 2:
        RK2Meth(x0, y0, h, xn);
        break;
    case 3:
        RK4Meth(x0, y0, h, xn);
        break;

    default:
        printf("Invalid choice. Please select a valid method.\n");
        break;
    }

    return 0;
}
```

Output

Euler's Method (RK1)

```
Enter the value of x at which you want to find y: 2
Select method:
1. Runge-Kutta 1st Order
2. Runge-Kutta 2nd Order
3. Runge-Kutta 4th Order
1
Solution using Runge-Kutta 1st Order Method:

Step       x               y            f(x, y)
-----------------------------------------------------
0         0.0000          1.000000        1.000000
1         0.2000          1.200000        1.400000
2         0.4000          1.480000        1.880000
3         0.6000          1.856000        2.456000
4         0.8000          2.347200        3.147200
5         1.0000          2.976640        3.976640
6         1.2000          3.771968        4.971968
7         1.4000          4.766362        6.166362
8         1.6000          5.999635        7.599635
9         1.8000          7.519562        9.319562
-----------------------------------------------------
Approximate value of y at x = 2.0000 is 9.383474
```

Second-Order Runge-Kutta Method (RK2, Heun's Method)

```
Enter the value of x at which you want to find y: 2
Select method:
1. Runge-Kutta 1st Order
2. Runge-Kutta 2nd Order
3. Runge-Kutta 4th Order
2
Solution using Runge-Kutta 2nd Order Method:

Step       x               y            f(x, y)
-----------------------------------------------------
0         0.0000          1.000000        1.000000
1         0.2000          1.240000        1.440000
2         0.4000          1.576800        1.976800
3         0.6000          2.031696        2.631696
4         0.8000          2.630669        3.430669
5         1.0000          3.405416        4.405416
6         1.2000          4.394608        5.594608
7         1.4000          5.645422        7.045422
8         1.6000          7.215415        8.815415
9         1.8000          9.174807       10.974807
-----------------------------------------------------
Approximate value of y at x = 2.0000 is 11.609264
```

Fourth-Order Runge-Kutta Method (RK4)

```
Enter the value of x at which you want to find y: 2
Select method:
1. Runge-Kutta 1st Order
2. Runge-Kutta 2nd Order
3. Runge-Kutta 4th Order
3
Solution using Runge-Kutta 4th Order Method:

Step       x               y           f(x, y)
--------------------------------------------------
0         0.0000          1.000000        1.000000
1         0.2000          1.242800        1.442800
2         0.4000          1.583636        1.983636
3         0.6000          2.044213        2.644213
4         0.8000          2.651042        3.451042
5         1.0000          3.436502        4.436502
6         1.2000          4.440144        5.640144
7         1.4000          5.710272        7.110272
8         1.6000          7.305886        8.905886
9         1.8000          9.299049       11.099050
```

Discussion

The results demonstrate the effectiveness and accuracy of different numerical methods for solving ordinary differential equations. Euler's Method (RK1) is straightforward to implement but accumulates significant error as the step size increases, leading to less accurate results. The Second-Order Runge-Kutta Method (RK2, Heun's Method) improves accuracy by averaging slopes, reducing local truncation error. The Fourth-Order Runge-Kutta Method (RK4) provides the most accurate results among the three, as seen in the output, due to its use of multiple intermediate slopes and weighted averaging. However, RK4 requires more computations per step. The choice of method depends on the required accuracy and available computational resources. For most practical purposes, RK4 is preferred due to its balance between accuracy and efficiency.

Conclusion

In this lab, we explored and implemented three numerical methods for solving ordinary differential equations: Euler's Method (RK1), Second-Order Runge-Kutta (RK2), and Fourth-Order Runge-Kutta (RK4). The results show that while Euler's Method is simple, it is less accurate compared to RK2 and RK4. RK2 offers improved accuracy by considering the average slope, and RK4 provides the highest accuracy due to its use of multiple intermediate calculations. Understanding these methods equips us with essential tools for approximating solutions to ODEs when analytical solutions are not feasible.

Name: Kushal Prasad Joshi
Roll No: 230345
Faculty: Science and Technology

Program: BE Computer
Group: B