Experiment-5
Ordinary Differential Equations

(Duration: 120 mins)

Author: Ertunga Burak Koçal
ertungakocal@iyte.edu.tr

Purpose: In this experiment, you are going to approximate solution of an ordinary differential equation using different methods.

Introduction

The Runge-Kutta methods are family of implicit and explicit iterative methods for obtaining numerical approximations to solutions of ordinary differential equations.

Let an initial value problem be specified as below:

$$\frac{dy}{dx} = g(x,y), \qquad y(x_0) = y_0 \tag{1}$$

where y is an unknown function of x and $\frac{dy}{dx}$ is the rate at which y changes. The value of $y(x_{last})$ can be approximated using $g(x,y)$, x_0, y_0 and chosen step size (h).

The Euler Method

The Euler method is the simplest Runge-Kutta method with one stage.
-Pick a step-size h > 0 and calculate y_{n+1} for $n = 0, 1, 2, 3...$ using

$$y_{n+1} = y_n + h * g(x_n, y_n) \tag{2}$$

First step:

$$y_1 = y_0 + h * g(x_0, y_0) \tag{3}$$

The Midpoint Method

The midpoint method is a second-order Runge-Kutta method with two stages.
-Pick a step-size h > 0 and calculate y_{n+1} for $n = 0, 1, 2, 3...$ using

$$y_{n+1} = y_n + h * g(x_n + \frac{h}{2}, y_n + \frac{h * g(x_n, y_n)}{2}) \tag{4}$$

The Runge-Kutta Method

The Runge-Kutta method is also known as "classic Runge-Kutta method".
-Pick a step-size h > 0 and calculate y_{n+1} for $n = 0, 1, 2, 3...$ using

$$k_1 = g(x_n, y_n) \tag{5}$$

$$k_2 = g(x_n + \frac{h}{2}, y_n + \frac{hk_1}{2}) \tag{6}$$

$$k_3 = g(x_n + \frac{h}{2}, y_n + \frac{hk_2}{2}) \tag{7}$$

$$k_4 = g(x_n + h, y_n + hk_3) \tag{8}$$

$$y_{n+1} = y_n + \frac{h * (k_1 + 2k_2 + 2k_3 + k_4)}{6} \tag{9}$$

Problem Statement

Solve the following initial value problem over the interval [0,2] using $stepSize(h) = 0.1$ and $y(0) = 1$.

$$\frac{dy}{dx} = yx^3 - 0.1y \tag{10}$$

Analytical solution is:

$$y(x) = e^{(x^4/4)-0.1x} \tag{11}$$

Lab Procedure

We highly recommend you to follow lab procedure without skipping any step and read each step thoroughly before you start.

1- Ask x_0, y_0, x_{last} and $stepSize$ values from user in main function. (10 pts) These values will be used in both Euler and midpoint methods.

2- Write two separate functions to calculate $y(x)$ and $\frac{dy}{dx} = g(x, y)$. Call these functions in main and print calculated values for $x = 0.2$ and $y = 0.8$ values. (20 pts)

y(0.2) = 0.9805 g(0.2,0.8) = -0.0736

```
double fY(double x)
double fYdx(double x, double y)
```

Note: You should be able to extract the y values (yEuler,yMidpoint,yRK4) from the corresponding functions to print them in the main.

3- Implement Euler method and check its operation. Do not use any loop to call Euler function more than one time in main; you should call it only once. The number of steps should be determined using x_0, x_{last} and $stepSize$ and you should alocate proper amount of memory to store obtained results considering the number of steps. Check Fig:1 (15 pts)

```
void euler(double (*fYdx1)(double, double), double *yEuler, double xFirst, double yFirst,
    double xLast, double stepSize)
```

4- Implement midpoint method and check its operation. Do not use any loop to call midpoint function more than one time in main; you should call it only once. The number of steps should be determined using x_0, x_{last} and $stepSize$ and you should alocate proper amount of memory to store obtained results considering the number of steps. Check Fig:2 (30 pts)

```
void midpoint(double (*fYdx2)(double, double), double *yMidpoint, double xFirst, double
    yFirst, double xLast, double stepSize)
```

5- Implement Runge-Kutta method and check its operation. Do not use any loop to call Runge-Kutta function more than one time in main; you should call it only once. The number of steps should be determined using x_0, x_{last} and $stepSize$ and you should alocate proper amount of memory to store obtained results considering the number of steps. Check Fig:3 (15 pts)

```
void RK4(double (*fYdx3)(double, double), double *yRK4, double xFirst, double yFirst, double
    xLast, double stepSize)
```

6- Print calculated values obtained with all methods and function you wrote for $y(x)$ in the second step for each iteration with values $x_0 = 0$, $y_0 = 1$, $x_{last} = 2$ and $stepSize = 0.1$. Comment on results. Check Fig:4 (10 pts)

```
Please enter x0 and y0
0 1
Please enter xLast and stepSize
2 0.5

x0 = 0.000000, y0 = 1.000000, xLast = 2.000000, stepSize = 0.500000

                      Euler
Step 0-y(0.000000)--- 1.000000
Step 1-y(0.500000)--- 0.950000
Step 2-y(1.000000)--- 0.961875
Step 3-y(1.500000)--- 1.394719
Step 4-y(2.000000)--- 3.678571

Process returned 0 (0x0)   execution time : 1.875 s
Press any key to continue.
```

Figure 1: $y(x)$ values obtained with stepSize = 0.5 -Euler Method

```
Please enter x0 and y0
0 1
Please enter xLast and stepSize
2 0.5

x0 = 0.000000, y0 = 1.000000, xLast = 2.000000, stepSize = 0.500000

                      MidPoint
Step 0-y(0.000000)--- 1.000000
Step 1-y(0.500000)--- 0.958867
Step 2-y(1.000000)--- 1.114149
Step 3-y(1.500000)--- 2.378752
Step 4-y(2.000000)--- 13.755717

Process returned 0 (0x0)   execution time : 3.421 s
Press any key to continue.
```

Figure 2: $y(x)$ values obtained with stepSize = 0.5 -Midpoint Method

```
Please enter x0 and y0
0 1
Please enter xLast and stepSize
2 0.5

x0 = 0.000000, y0 = 1.000000, xLast = 2.000000, stepSize = 0.500000

                      RK4
Step 0-y(0.000000)--- 1.000000
Step 1-y(0.500000)--- 0.966180
Step 2-y(1.000000)--- 1.161446
Step 3-y(1.500000)--- 3.027639
Step 4-y(2.000000)--- 37.450162

Process returned 0 (0x0)   execution time : 2.140 s
Press any key to continue.
```

Figure 3: $y(x)$ values obtained with stepSize = 0.5 -Runge-Kutta Method

```
Please enter x0 and y0
0 1
Please enter xLast and stepSize
2 0.1
                      Euler      Midpoint   KR4        Exact
Step 0-y(0.000000)--- 1.000000---1.000000---1.000000---1.000000
Step 1-y(0.100000)--- 0.990000---0.990062---0.990075---0.990075
Step 2-y(0.200000)--- 0.980199---0.980543---0.980591---0.980591
Step 3-y(0.300000)--- 0.971181---0.972308---0.972413---0.972413
Step 4-y(0.400000)--- 0.964092---0.966774---0.966958---0.966958
Step 5-y(0.500000)--- 0.960621---0.965918---0.966209---0.966209
Step 6-y(0.600000)--- 0.963022---0.972337---0.972777---0.972777
Step 7-y(0.700000)--- 0.974193---0.989415---0.990075---0.990075
Step 8-y(0.800000)--- 0.997866---1.021648---1.022653---1.022653
Step 9-y(0.900000)--- 1.038978---1.075256---1.076834---1.076834
Step 10-y(1.000000)--- 1.104330---1.159254---1.161834---1.161834
Step 11-y(1.100000)--- 1.203720---1.287377---1.291785---1.291785
Step 12-y(1.200000)--- 1.351898---1.481556---1.489438---1.489440
Step 13-y(1.300000)--- 1.571987---1.778456---1.793236---1.793242
Step 14-y(1.400000)--- 1.901632---2.242252---2.271387---2.271408
Step 15-y(1.500000)--- 2.404424---2.990813---3.051395---3.051475
Step 16-y(1.600000)--- 3.191873---4.252124---4.385620---4.385923
Step 17-y(1.700000)--- 4.467345---6.492853---6.806338---6.807500
Step 18-y(1.800000)--- 6.617478---10.729468---11.519025---11.523633
Step 19-y(1.900000)--- 10.410617---19.331936---21.480330---21.499399
Step 20-y(2.000000)--- 17.447153---38.251980---44.617585---44.701184
```

Figure 4: $y(x)$ values obtained with stepSize = 0.1