Experiment # 4

Ordinary Differential Equations

Purpose of the experiment:

In this experiment we will deal with first order differential equations (ordinary differential equations) with 4 different numerical methods.

For a given Ordinary Differential Equation with its initial values, numerical methods can be used to reach a desired point. For example, let's look at the example below.

dy/dx = y – x^2 +1 ←→ *dy/dx = f (x, y)* .

y (0) = 0.5 ←→ *y(x0) = y0*

Step Size = 0.4 ←→ Step Size = h

y (2) =? ←→ y($x_{desired}$) =?

In this example we will do a process of iterations to reach y (2) by starting from y (0). As our step size is 0.4 and y (2) can be reached in 5 steps when the initial point is y (0).

y (0) → y (0.4) → y (0.8) → y (1.2) → y (1.6) → y (2)

(Hint: After each iteration, your initial point should be updated as the new point you find.)

The numerical methods which we use in this experiment:

- *Euler's Method*
- *Midpoint method (a type of 2^{nd} order Runge-Kutta Method)*
- *3^{rd} order Runge-Kutta Method (its most common used type)*
- *4^{th} order Runge-Kutta Method (its most common used type)*

Euler's method:

y (xi + h) = y(xi) + y'(xi) * h

Midpoint method:

y (xi + h) = y(xi) + k2 * h

k1 = f (xi, yi)

k2 = f (xi + h/2, yi + ½ * k1 * h)

3^{rd} order Runge-Kutta Method:

y (xi + h) = y(xi) + (1/6 * k1 + 4/6 * k2 + 1/6 * k3) * h

k1 = f (xi, yi)

k2 = f (xi + ½* h, yi + ½*k1*h)

k3 = f (xi + h, yi – k1*h + 2*k2*h)

4^{th} order Runge-Kutta Method:

y (xi + h) = y(xi) + (1/6 * k1 + 2/6 * k2 + 2/6 * k3 + 1/6 * k4) * h

k1 = f (xi, yi)

k2 = f (xi + ½* h, yi + ½* k1 * h)

k3 = f (xi + ½* h, yi + ½*k2*h)

k4 = f (xi + h, yi + k3 * h)

Laboratory Procedure

Write a C code for the initial value first order differential equation problem to find desired value (y_desired) at desired point (x_last):

dy/dx = 10*x + y – 8

y (0) = 1

y(x) = -10*x – 2 + 3* e^x → (Exact solution)

Step #1 (10 points): *Ask to user to enter x0, y0, x_last, Step size in the main function.*

Step #2 (10 points): *Write 2 functions: first one calculates the dy/dx at given "x" and given "y" and second one calculates the exact y(x) at given "x".*

```
double dydx (double x, double y)

double yx (double x)
```

Step #3 (10 points): *Create 4 different arrays for 4 different numeric method which will be send to the functions that we will create in next steps.*

(Hint: using pointer will help. Thanks to it, values of arrays will be changed automatically in the main function, also use malloc () to set the size of array.)

Step #4 (50 points): *Create 4 functions for Euler's Method, Midpoint method, 3^{rd} order Runge-Kutta Method, and 4^{th} order Runge-Kutta Method.*

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

void midpoint(double (*dydx)(double,double), double *yMidPoint, double xFirst, double yFirst, double xLast, double stepSize)

void RK3(double (*dydx)(double,double), double *yRK3, double xFirst, double yFirst, double xLast, double stepSize)

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

!!! Important → ***All iterations should be done in these functions. (Do not use loops in main function)***

Step #5 (10 points): *Use the functions you created at step#4 in the main function and obtain the values for each iteration in your arrays you created at step#3.*

Step #6 (10 points): *In your main function write all iteration results by using a single loop.*

(Hint: You don't need to print all of them in figure below if you could not write all methods. Just print your work as you could do in lab section.)

```
enter x0  y0 xLast and stepSize
enter x0 (initial x)   3
enter y0 (initial y)   28.25
enter xLast (desired point)    4
enter h (stepSize)     0.1
                         Euler        Midpoint      RungeKutta3  RungeKutta4    Exact
Step 0- y(3.000000)---- 28.250000---- 28.250000---- 28.250000---- 28.250000---- 28.256611
Step 1- y(3.100000)---- 33.275000---- 33.576250---- 33.586292---- 33.586543---- 33.593854
Step 2- y(3.200000)---- 38.902500---- 39.566756---- 39.588950---- 39.589505---- 39.597591
Step 3- y(3.300000)---- 45.192750---- 46.291266---- 46.328055---- 46.328974---- 46.337917
Step 4- y(3.400000)---- 52.212025---- 53.826849---- 53.881055---- 53.882410---- 53.892300
Step 5- y(3.500000)---- 60.033228---- 62.258668---- 62.333546---- 62.335418---- 62.346356
Step 6- y(3.600000)---- 68.736550---- 71.680828---- 71.780124---- 71.782607---- 71.794703
Step 7- y(3.700000)---- 78.410205---- 82.197315---- 82.325334---- 82.328536---- 82.341913
Step 8- y(3.800000)---- 89.151226---- 93.923033---- 94.084714---- 94.088759---- 94.103553
Step 9- y(3.900000)---- 101.066348---- 106.984951---- 107.185957---- 107.190985---- 107.207347
Step 10- y(4.000000)---- 114.272983---- 121.523371---- 121.770180---- 121.776355---- 121.794450

Process returned 0 (0x0)   execution time : 17.803 s
Press any key to continue.
```