COMPUTATIONAL PRACTICUM assignment

Differential Equations

Artur Akhmetshin
Group: BS17-05

y’ = sin(x) + y

Exact solution of IVP(Initial Value Problem)

there x0 = 0 and y0 = 1

Given Differential Equation:

$$y' = \sin(x) + y$$

Solution of DE:

$$y = C_1 e^x - \frac{(\sin(x) + \cos(x))}{2}$$

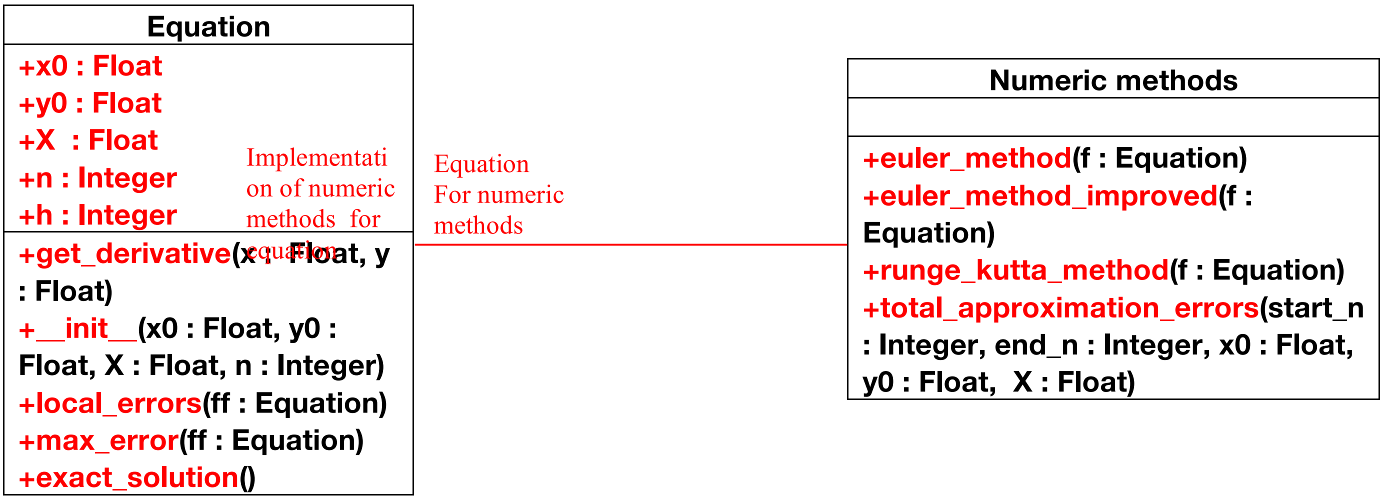
Solution of IVP:

$$y = \frac{3}{2} e^x - \frac{(\sin(x) + \cos(x))}{2}$$

There isn’t any point of discontinuity in solution of given differential equation.

System contains 3 classes: Equation, Numeric methods and supporting class Plotting

UML diagram for Equation and Numeric methods



Explanation of Equation class attributes and methods

Equation class is used to operate with given y' = sin(x) + y differential equation, change initial values, grid size to solve IVP inside class.

Attributes:

+x0 : Float Starting point of x-axis of IVP problem segment
+y0 : Float Value of function in x0 point y(x0) = y0
+X : Float Ending point of x-axis of IVP problem segment
+n : Integer Grid size
+h : Integer Value of one grid step(depends on x0, X0 and n)

Methods:

+__init__(x0 : Float, y0 : Float, X : Float, n : Integer)

```
def __init__(self, x0, y0, X, n): #initialization
    self.x0 = x0
    self.y0 = y0
    self.X = X
    self.n = n

    h = (X - x0)/n
    self.h = h

    self.x = numpy.arange(x0, X + 0.000001, h)
```

Method initialize initial values, scope and grid size for solving IVP problem, these values are assigned to Equation instance

+get_derivative(x : Float, y : Float)

```
def get_derivative(self, x, y):  #local derivative
    return math.sin(x) + y
```

Method returns local derivative of function in the point (x, y) using following function y' = sin(x) + y

+local_errors(ff : Equation)

```
def local_errors(self, ff):  #graph of local errors

    y = ff[1]

    y_error = [0] * len(self.x)
    ex = self.exact_solution()

    for i in range(len(self.x)):
        y_error[i] = ex[1][i] - y[i]

    return [self.x, y_error]
```

Method calculates function of local errors of function ff with respect to exact solution of given differential equation

+max_error(ff : Equation)

```
def max_error(self, ff):  #maximum error of function ff

    l = self.local_errors(ff)
    mx = 0.0
    a = l[1]

    for i in range(len(a)):
        mx = max(mx, abs(a[i]))

    return mx
```

Method calculate value of the maximum error of function ff with respect to exact solution of given differential equation

+exact_solution()

```
def exact_solution(self):

    y = [0] * len(self.x)

    for i in range(len(self.x)):
        y[i] = 3.0/2.0 * (math.e**self.x[i]) - (math.sin(self.x[i]) + math.cos(self.x[i]))/2.0

    return [self.x, y]
```

Method calculate values of exact solution for given differential equation with determined scope and grid size in __init__

Explanation of Numeric_methods class attributes and methods

Numeric_methods class is used to operate solve IVP problem using Euler's, Euler' Improved and Runge-Kutta methods for given Equation e that is accepted by the class.

Numeric_methods has no any attribute.

+euler_method(f : Equation)

```
def euler_method(self, f):  #function got by Euler's method

    y = [0] * len(f.x)
    y[0] = f.y0

    for i in range(1, len(f.x)):
        y[i] = y[i - 1] + f.h * f.get_derivative(f.x[i - 1], y[i - 1]) #augmentation

    return [f.x, y]
```

Method calculate values of function of given differential equation ff on scope [x; X] with grid step n. It uses Euler's method for calculations.

+euler_method_improved(f : Equation)

```
def euler_method_improved(self, f):   #function got by improved Euler's method

    y = [0] * len(f.x)
    y[0] = f.y0

    for i in range(1, len(f.x)):
        d = f.h * f.get_derivative(f.x[i - 1] + f.h/2, y[i - 1] + f.h/2 * f.get_derivative(f.x[i - 1], y[i - 1]))
        y[i] = y[i - 1] + d #augmentation

    return [f.x, y]
```

Method calculate values of function of given differential equation ff on scope [x; X] with grid step n. It uses Improved Euler's method for calculations.

+runge_kutta_method(f : Equation)

```
def runge_kutta_method(self, f):   #function got by improved Runge-Kutta method

    y = [0] * len(f.x)
    y[0] = f.y0

    for i in range(1, len(f.x)):

        d1 = f.get_derivative(f.x[i - 1], y[i - 1])
        d2 = f.get_derivative(f.x[i - 1] + f.h/2, y[i - 1] + f.h * d1/2)
        d3 = f.get_derivative(f.x[i - 1] + f.h/2, y[i - 1] + f.h * d2/2)
        d4 = f.get_derivative(f.x[i - 1] + f.h,   y[i - 1] + f.h * d3)

        d = f.h * (d1 + 2 * d2 + 2 * d3 + d4)/6
        y[i] = y[i - 1] + d #augmentation

    return [f.x, y]
```

Method calculate values of function of given differential equation ff on scope [x; X] with grid step n. It uses Runge-Kutta method for calculations(calculating local derivatives, getting local result and augment it.

+total_approximation_errors(start_n : Integer, end_n : Integer, x0 : Float, y0 : Float, X : Float)

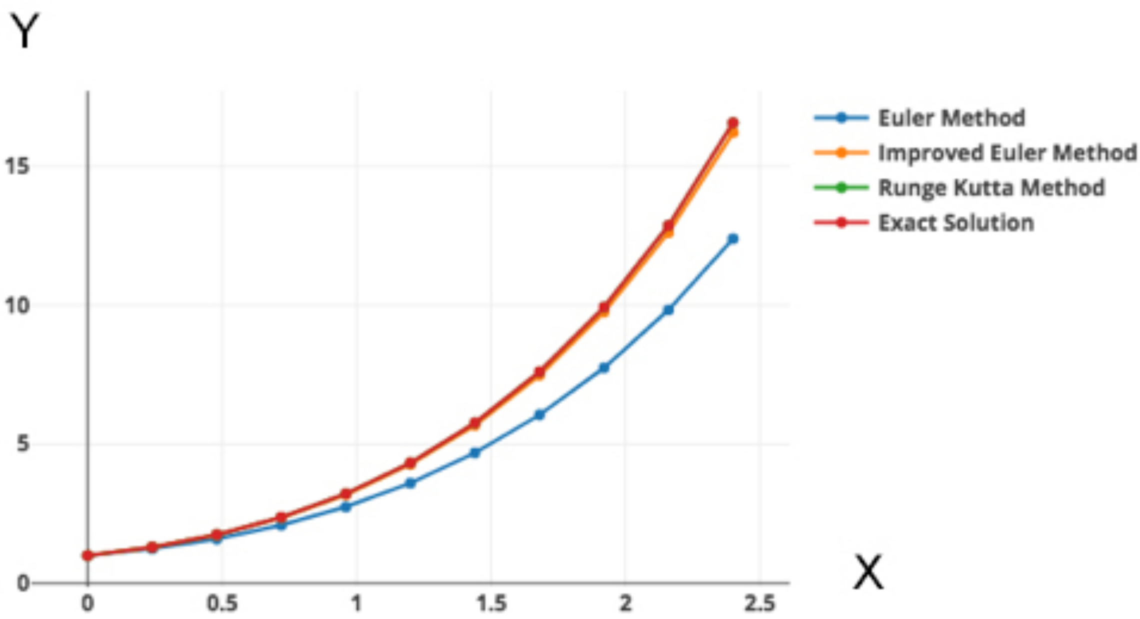
```
def total_approximation_errors(self, start_n, end_n, x0, y0, X):

    euler_total_y = [0] * (end_n - start_n + 1)
    euler_improved_total_y = [0] * (end_n - start_n + 1)
    runge_kutta_total_y = [0] * (end_n - start_n + 1)
    total_x = [0] * (end_n - start_n + 1)

    for i in range(start_n, end_n + 1):

        j = i - start_n
        total_x[j] = i

        e = Equation(x0, y0, X, i)

        euler_result = (self.euler_method(e))
        euler_improved_result = (self.euler_method_improved(e))
        runge_kutta_result = (self.runge_kutta_method(e))

        euler_total_y[j] = e.max_error(euler_result)
        euler_improved_total_y[j] = e.max_error(euler_improved_result)
        runge_kutta_total_y[j] = e.max_error(runge_kutta_result)

    euler_total = [total_x, euler_total_y]
    euler_improved_total = [total_x, euler_improved_total_y]
    runge_kutta_total = [total_x, runge_kutta_total_y]

    return [euler_total, euler_improved_total, runge_kutta_total]
```

Method analyze the total approximation error depending on the number of grid cells, method accepts strat_n – starting point of grid step and end_n – finishing point of grid step and return functions of total errors of Euler's, Improved Euler's and Runge-Kutta methods for given differential equation y' = sin(x) + y. It was implemented by using methods inside Numeric_methods class for all integer grid steps values on [start_n; end_n] and calculating maximum error of taken functions.

Graph of solutions of differential equation y’ = sin(x) + y using Euler’s method, Improved Euler’s method, Runge-Kutta method and graph of exact solution.

Initial values: x0 = 0, y0 = 1, X = 12/5

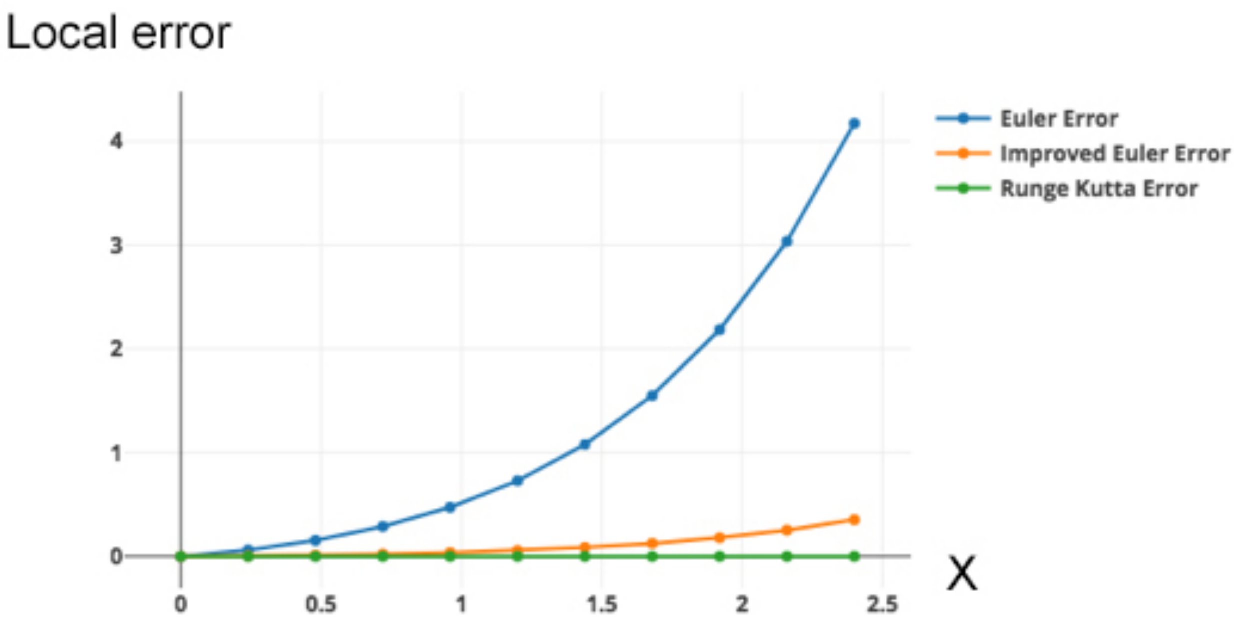
Grid step N = 10



Graphs of local errors, difference between exact solution of differential equation $y' = \sin(x) + y$ and solutions calculated using Euler's method, Improved Euler's method, Runge-Kutta method.

Initial values: $x0 = 0$, $y0 = 1$, $X = 12/5$

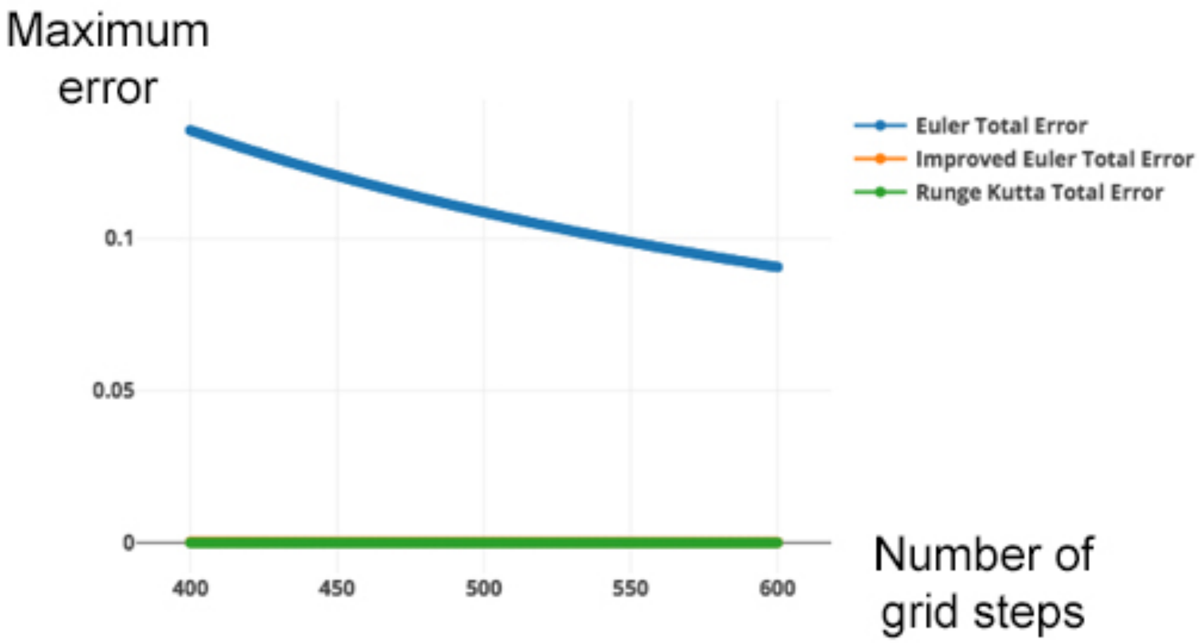
Number of grid steps $N = 10$



Graph of total approximation errors of solutions of differential equation y'= sin(x) + y calculated using Euler's method, Improved Euler's method, Runge-Kutta method, depends on number of grid cells (x-axis).

Initial values: x0 = 0, y0 = 1, X = 12/5

Number of grid steps: [400; 600]



Summary

In this work we studied different method solution of differential equations: Euler’s Method, Improved Euler’s Method, Runge-Kutta Method.
We understood that Runge-Kutta Method is the most accurate and the nearest to exact solution. Improved Euler’s Method is more accurate that Euler’s Method.
We observed approximation according to number of grid steps. For all following methods accuracy as growths as number of grid steps growths.