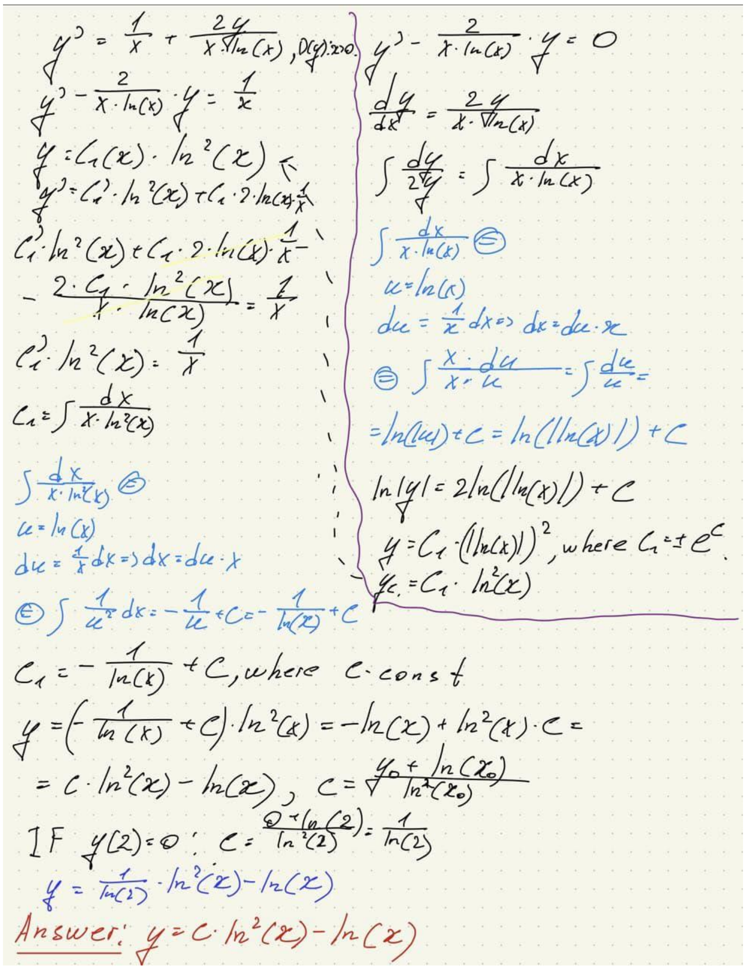
Report

Report created by Kuznetsov Kirill B20-04.

Link to github: https://github.com/Kirill-Kuznetsov-git/DiffEq-Assignment

Goal: Goal ofg this application is analyze numerical methods and compare them with exact solution.

I have difference equation and initial x0 and y0. This is solution of my dufference equation and eqaution of constant, which depend on the x0 and y0:



$$y' = \frac{1}{x} + \frac{2y}{x \cdot \ln(x)},\ D(y): x>0$$
$$y' - \frac{2}{x \cdot \ln(x)} \cdot y = \frac{1}{x}$$
$$y = C_1(x) \cdot \ln^2(x)$$
$$y' = C_1' \cdot \ln^2(x) + C_1 \cdot 2 \cdot \ln(x) \cdot \frac{1}{x}$$
$$C_1' \cdot \ln^2(x) + C_1 \cdot 2 \cdot \ln(x) \cdot \frac{1}{x} - \frac{2 \cdot C_1 \cdot \ln^2(x)}{x \cdot \ln(x)} = \frac{1}{x}$$
$$C_1' \cdot \ln^2(x) = \frac{1}{x}$$
$$C_1 = \int \frac{dx}{x \cdot \ln^2(x)}$$
$$\int \frac{dx}{x \cdot \ln^2(x)} \circleddash$$
$$u = \ln(x)$$
$$du = \frac{1}{x} dx \Rightarrow dx = du \cdot x$$
$$\circleddash \int \frac{1}{u^2} du = -\frac{1}{u} + C = -\frac{1}{\ln(x)} + C$$

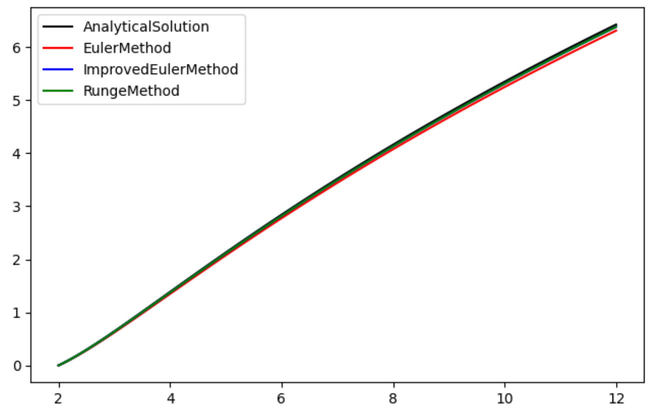
$$y' - \frac{2}{x \cdot \ln(x)} \cdot y = 0$$
$$\frac{dy}{dx} = \frac{2y}{x \cdot \ln(x)}$$
$$\int \frac{dy}{2y} = \int \frac{dx}{x \cdot \ln(x)}$$
$$\int \frac{dx}{x \cdot \ln(x)} \circleddash$$
$$u = \ln(x)$$
$$du = \frac{1}{x} dx \Rightarrow dx = du \cdot x$$
$$\circleddash \int \frac{x \cdot du}{x \cdot u} = \int \frac{du}{u} =$$
$$= \ln(|u|) + C = \ln(|\ln(x)|) + C$$
$$\ln|y| = 2\ln(|\ln(x)|) + C$$
$$y = C_1 \cdot (|\ln(x)|)^2, \text{ where } C_1 = \pm e^C$$
$$y_c = C_1 \cdot \ln^2(x)$$

$$C_1 = -\frac{1}{\ln(x)} + C, \text{ where } C\text{-const}$$
$$y = \left(-\frac{1}{\ln(x)} + C\right) \cdot \ln^2(x) = -\ln(x) + \ln^2(x) \cdot C =$$
$$= C \cdot \ln^2(x) - \ln(x),\quad C = \frac{y_0 + \ln(x_0)}{\ln^2(x_0)}$$
$$\text{IF } y(2) = 0:\ C = \frac{0 + \ln(2)}{\ln^2(2)} = \frac{1}{\ln(2)}$$
$$y = \frac{1}{\ln(2)} \cdot \ln^2(x) - \ln(x)$$
Answer: $y = C \cdot \ln^2(x) - \ln(x)$

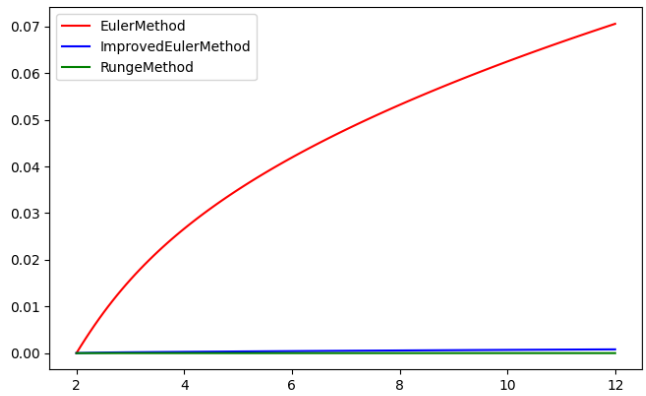
Then I, using PyQt5 and myplotlib, create application, which visualize graphs of each numerical method and analytic solution:

If N = 120 and n0=10, then:

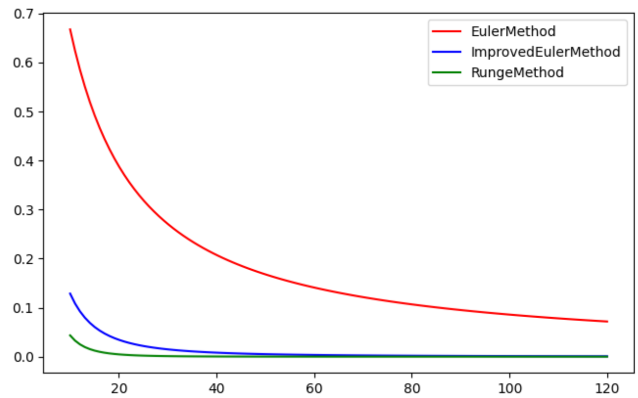
Solutions:



LTE:

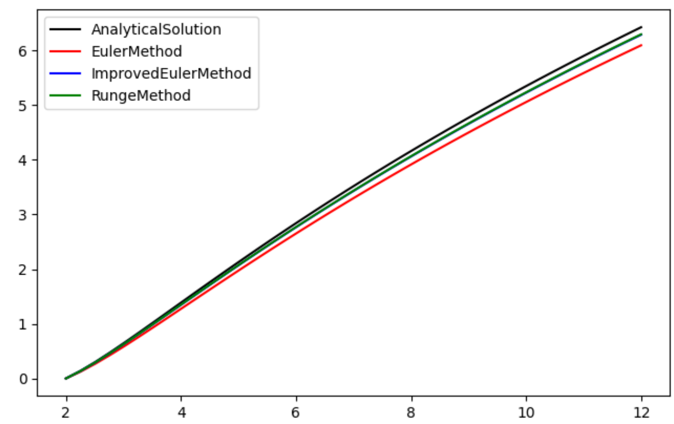


GTE:

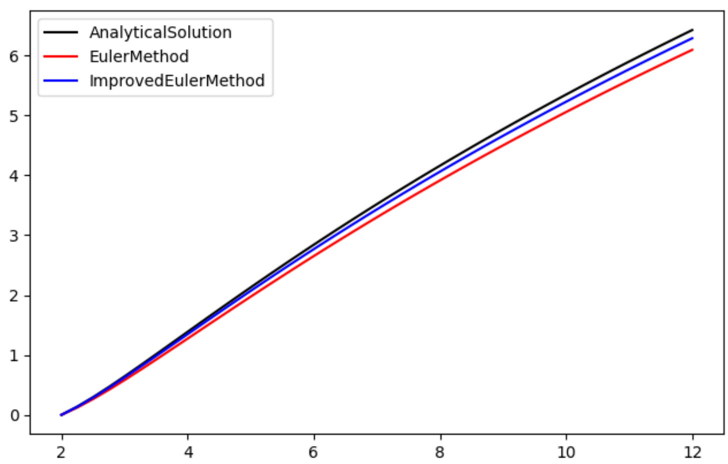


If I change attributes and set N = 40 and n0 = 20, then it’s clearly seen that graphs stay less accurate:

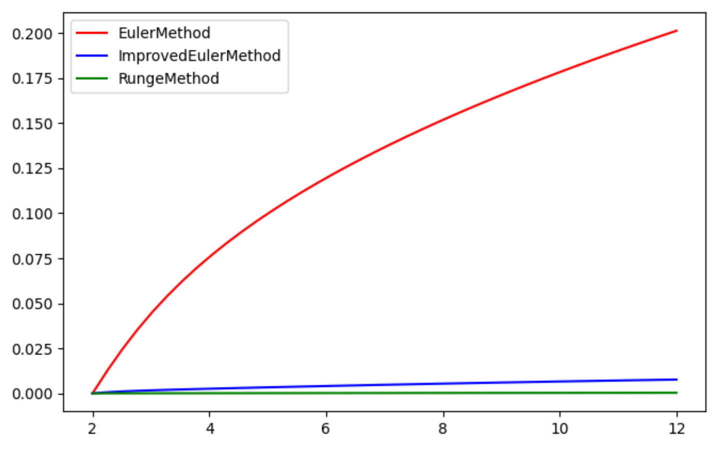
Solutions:



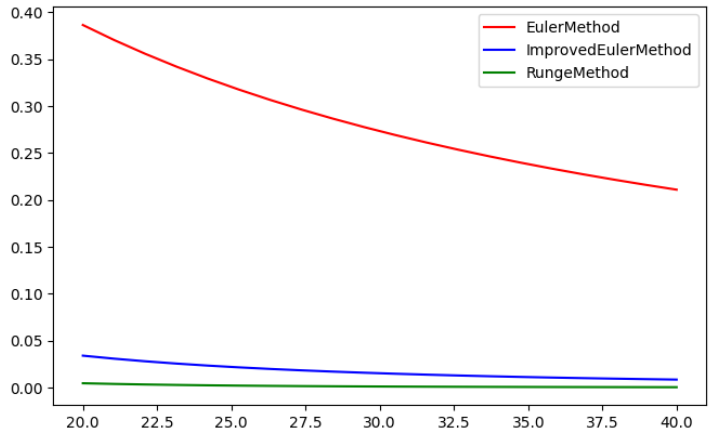
Solutions without method Runge-Kutta:



LTE:

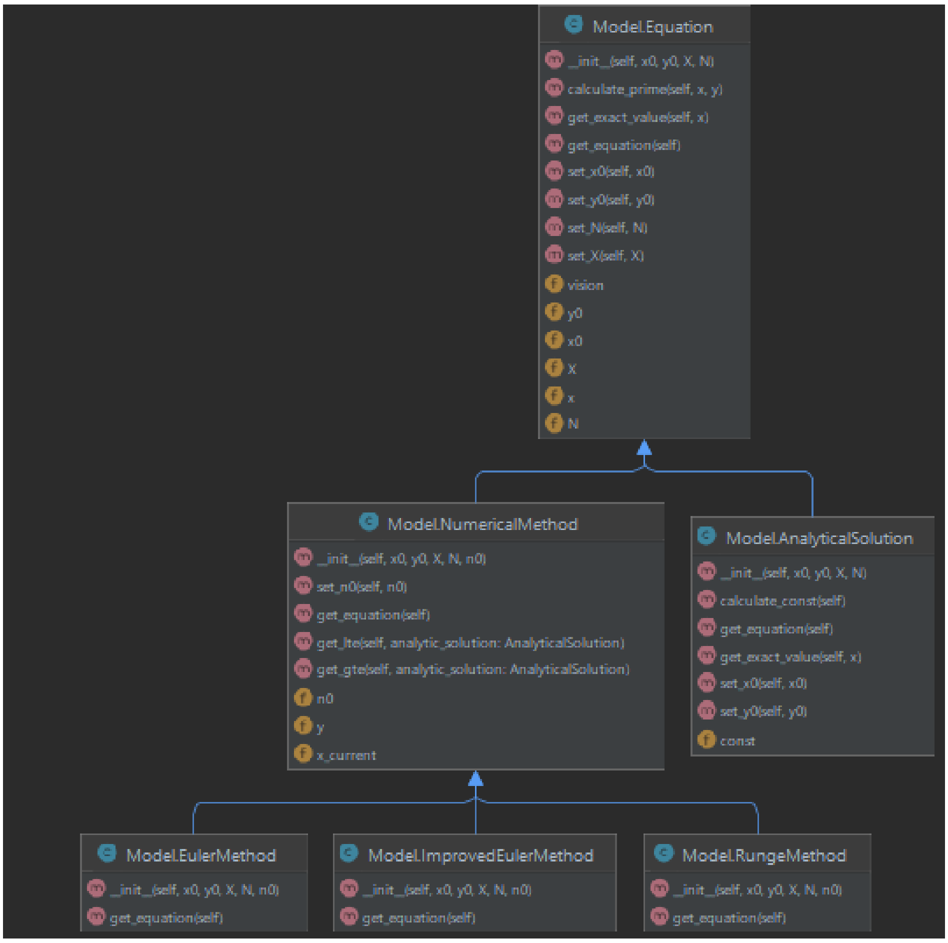


GTE:



Conclusion: It is clearly seen from the presented graph that the Euler method is the most inaccurate, the improved Euler method is more accurate and the Runge-Kutta method is the most accurate. If N is growing, that difference beetwen all numerical methods become unvisible, but in general: Runge-Kutta method is the best method. It show all graphs: Solutions, LTE and GTE.

UML diagram:



CODE:

I my model, I have abstract Equation model.

It model has some set value method.

Function “calculate prime” - return a value of f(x, y).

Function “get_equation()” return a equation in a form of np.linespace and python array.

```python
class Equation:
    def __init__(self, x0, y0, X, N):
        self.x0 = x0
        self.X = X
        self.y0 = y0
        self.N = N
        self.x = np.linspace(self.x0, self.X, self.N)
        self.vision = True

    def calculate_prime(self, x, y):
        return 1 / x + 2 * y / (x * np.log(x))

    def get_exact_value(self, x):
        pass

    def get_equation(self):
        pass

    def set_x0(self, x0):
        self.x0 = x0
        self.x = np.linspace(self.x0, self.X, self.N)
```

From Equation extended two other classes:

1) Numerical Method.(Abstract class wich represent all numerical methods.)

Fucntions “get_lte()” and “get_gte()” return arrays of lte and gte respectivly.

```
def get_lte(self, analytic_solution: AnalyticalSolution) -> list:
    errors = []
    h = (self.X - self.x0) / self.N
    self.y = self.get_equation()[1]
    self.x_current = self.x0
    for i in range(self.N):
        errors.append(abs(self.y[i] - analytic_solution.get_exact_value(self.x_current)))

        self.x_current += h
    return [analytic_solution.x, errors]

def get_gte(self, analytic_solution: AnalyticalSolution):
    if self.n0 >= self.N:
        return [[0], [0]]
    errors = []
    n_old = self.N
    for i in range(int(self.N - self.n0)):
        self.set_N(self.n0 + i - 1)
        errors.append(max(self.get_lte(analytic_solution)[1]))
    self.set_N(n_old)
    return [np.linspace(self.n0, self.N, int(self.N - self.n0)), errors]
```

2) Analytic Method.

```
class AnalyticalSolution(Equation):
    def __init__(self, x0, y0, X, N):
        super().__init__(x0, y0, X, N)
        self.const = (self.y0 + np.log(self.x0)) / (np.log(self.x0) ** 2)

    def calculate_const(self):
        self.const = (self.y0 + np.log(self.x0)) / (np.log(self.x0) ** 2)
        return self.const

    def get_equation(self):
        super().get_equation()
        y = self.const * (np.log(self.x) ** 2) - np.log(self.x)
        return [self.x, y]

    def get_exact_value(self, x):

        return self.const * (np.log(x) ** 2) - np.log(x)

    def set_x0(self, x0):
        super().set_x0(x0)
        self.calculate_const()
```

From a Numerical method extended three other classes:

Each of this classes overlwrite her own get_eqation() fucntion.

1) Euler Method

```python
class EulerMethod(NumericalMethod):
    def __init__(self, x0, y0, X, N, n0):
        super().__init__(x0, y0, X, N, n0)

    def get_equation(self):
        res = super().get_equation()
        print(self.y)
        if res: return res
        h = (self.X - self.x0) / self.N
        for i in range(self.N - 1):
            self.y.append(self.y[-1] + h * self.calculate_prime(self.x_current, self.y[-1]))
            self.x_current += (self.X - self.x0) / self.N

        return [self.x, self.y]
```

2) Improved Euler Method0

```python
class ImprovedEulerMethod(NumericalMethod):
    def __init__(self, x0, y0, X, N, n0):
        super().__init__(x0, y0, X, N, n0)

    def get_equation(self):
        res = super().get_equation()
        if res: return res
        h = (self.X - self.x0) / self.N
        for i in range(self.N - 1):
            self.y.append(self.y[-1] + h * self.calculate_prime(self.x_current + h / 2,
                                                                 self.y[-1] + h / 2 * self.calculate_prime(
                                                                     self.x_current, self.y[-1])))
            self.x_current += (self.X - self.x0) / self.N

        return [self.x, self.y]
```

3) Runge-Katta Method

```python
class RungeMethod(NumericalMethod):
    def __init__(self, x0, y0, X, N, n0):
        super().__init__(x0, y0, X, N, n0)

    def get_equation(self):
        res = super().get_equation()
        if res: return res
        h = (self.X - self.x0) / self.N
        for i in range(self.N - 1):
            k1 = self.calculate_prime(self.x_current, self.y[-1])
            k2 = self.calculate_prime(self.x_current + h / 2, self.y[-1] + h / 2 * k1)
            k3 = self.calculate_prime(self.x_current + h / 2, self.y[-1] + h / 2 * k2)
            k4 = self.calculate_prime(self.x_current + h, self.y[-1] + h * k3)
            self.y.append(self.y[-1] + h / 6 * (k1 + 2 * k2 + 2 * k3 + k4))
            self.x_current += (self.X - self.x0) / self.N

        return [self.x, self.y]
```

To visualize graphs, I use a PyQt5 and MatPlotLib.

MainWindow – represent application in general.

FirstWindow, SecondWindow, ThirdWindow – represent each window with gpaphs. With Solutions, LTE and GTE respectivly.

From PyQt5 I use Canvas CheckBoxes and SpinBoxes.