Report

- **Student**: Matvey Abramov
- **Variant**: 3
- **Programming language**: Python 3
- **GUI library**: TKinter + matplotlib
- **Additional libraries**: Sympy (calculating exact solution); numba, numpy (fast calculations in python)

3	$sec(x) - y\,tg(x)$	1	0	7

Solution of IVP

$$\begin{cases} y' = Sec(x) - y\,tg(x) \\ y(0) = 1 \\ x \in (1, 8) \end{cases}$$

$$y' + y\,tg(x) = sec(x)$$

1. $y' + y\,tg(x) = 0$ – complementary eq.

$\frac{dy}{dx} = -y\,tg(x)$ $\quad \int \frac{dy}{y} = -\int tg(x)\,dx$ $\quad \ln_e|y| = \ln_e|Cos(x)| + C$

$y_c = Cos(x) \cdot e^C$ $\quad e^C = C_1$ $\quad y_c = C_1 \cos(x)$. $\quad y_p = Cos(x)$

2. Substitution: $y = u\,y_p$, where y_p – particular sol. of y_c

$y = u\,Cos(x)$ $\quad y' = u'Cos(x) - u\,Sin(x)$

$u'Cos(x) - u\,Sin(x) = sec(x) - u\,Cos(x)\,tg(x)$

$u'Cos(x) = sec(x) \cancel{- u\sin x + u\sin x}$ $\quad u' = \frac{sec(x)}{Cos(x)} = \frac{1}{Cos^2 x}$

$u = \int \frac{1}{cos^2 x}\,dx = \tan x + C_2$

$y = (\tan x + C_2)\,Cos\,x$ $\quad y = Sin\,x + C_2\,Cos\,x$

3. Solve IVP.

$y(0) = 1 = Sin\,0 + C_2\,Cos\,0$ $\quad 1 = 0 + C_2 \cdot 1$ $\quad C_2 = 1$

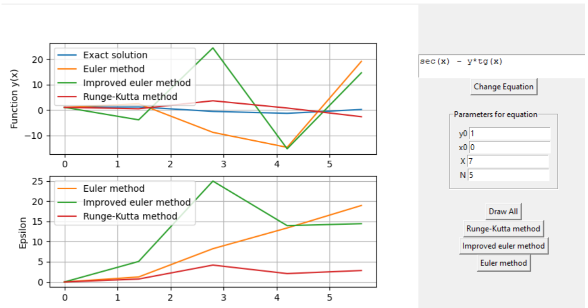
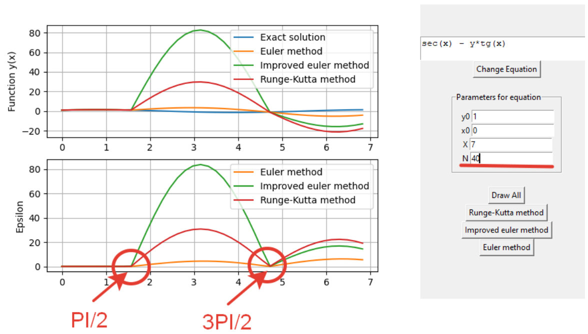
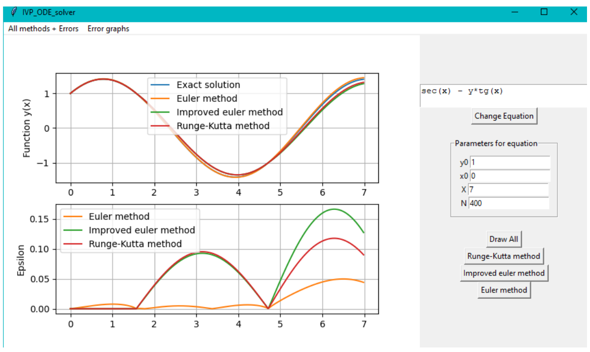
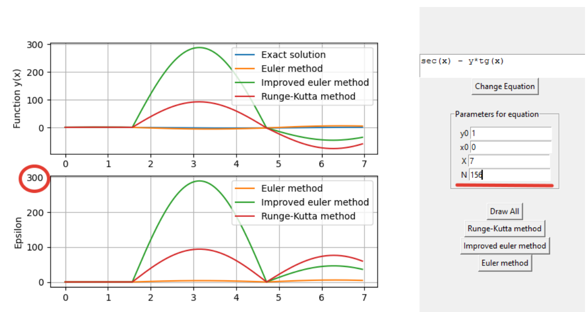
$y = \sin x + Cos\,x$ $\quad$ Answer: $Sin\,x + Cos\,x = y$

Points of discontinuity

```
cos(x) = 0  --->  x = PI*n - PI/2; n is Integer Number
```

Analysis of methods

All graphs and local error graph

Image	N range	Text
	`N <= 7`	Step size, which is greater than or equal to 1, so numerical graphs change very rapidly and approximation is not correct.
	`7 < N < 50`	With small N numerical graphs look correct before they hit point of discontinuity at `PI/2` . After that, the error get very high and the graphs look completely wrong. Until graphs reach `3PI/2` , the error gets smaller, and at `3PI/2` the error become large again.
	`N > 50`	With large N we still see raise of errors after points of discontinuity. However, errors became very small and graphs look almost identical.
	`N = 156`	With some N, errors become obscenely large after hitting point of discontinuity.

Graph of total errors

On large range(For example n from 10 to 200) there are a lot of spontaneous picks. The largest errors are in improved euler method, then runge-kutta method with smaller picks, and with almost no pick goes euler method.

Source code

I used **MVC** (*Model-View-Controller*) design pattern to organize my program.

- **Model class** - aggregates different methods (Exact solution and Butcher Schema methods) and manages them.
- **View class** - interacts directly with *matplotlib* and *tkinter*, draws everything to screen.
- **Controller class** - aggregates **Model** and **View** and manages them. On actions, gets new points from **Model** instance and updates graphics by calling **View** instance.

Methods

- **Exact solution method** - implemented using library *sympy*. Parts of code for calculating:

```
# Solve ODE
def calc_solved_func(self):
    x, y, c = self.symbols.x, self.symbols.y, self.symbols.const
    if self.eq != parse_expr("1/cos(x) - y*tg(x)", local_dict={"x": x, "y": y, "tg": tan_sympy}):
        dydx = Derivative(y, x)
        solved = ode.dsolve(Eq(dydx, self.eq), y)
        return solved
    return parse_expr("-y(x) + C1*cos(x) + sin(x)")
```

```
# Solve IVP ODE
def solve_ivp(self, x0, y0, _):
    x, y, c = self.symbols.x, self.symbols.y, self.symbols.const
    f = self.solved_func
    if c in f.free_symbols:
        f_for_const = f.subs([(x, x0), (self.symbols.y_without_params(x0), y0)])
        constant = {c: solveset(f_for_const, c, domain=S.Reals).args[0]}
        solved_ivp, = solve(f.subs(constant), y)
    else:
        solved_ivp = solve(f, [y])
    return lambdify([x], solved_ivp)
```

- **Butcher Schema Method** - General method for all methods, using butcher schema (euler method, improved euler method, runge-kutta method)

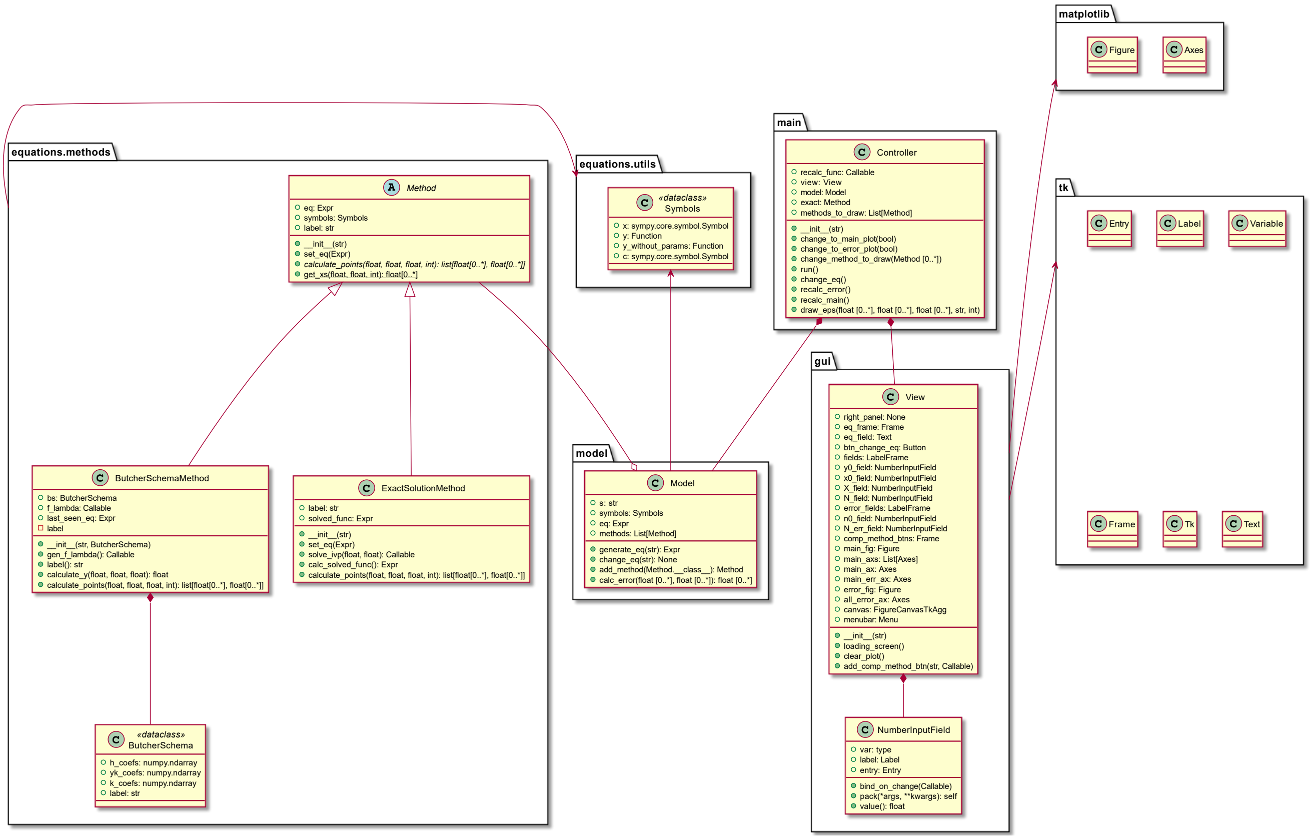
```
# Generate next y
def calc(h_coefs, yk_coefs, k_coefs, x, y, h, f):
    shape = k_coefs.shape[0]
    k = np.zeros((shape,), dtype=np.float64)
    for i in range(shape):
        ck = np.sum(np.multiply(k[:i], yk_coefs[i][:i]))
        k[i] = h * f(x + h_coefs[i] * h, y + ck)
    return y + np.sum(np.multiply(k, k_coefs))
```

```python
# Initialize butcher methods
def load_butcher_schemas() -> Iterable[ButcherSchema]:
    euler_method = ButcherSchema(
        np.array([0], dtype=np.float64),
        np.array([[0]], dtype=np.float64),
        np.array([1], dtype=np.float64),
        "Euler method"
    )
    improved_euler_method = ButcherSchema(
        np.array([0, 1], dtype=np.float64),
        np.array([[0, 0], [1, 0]], dtype=np.float64),
        np.array([0.5, 0.5], dtype=np.float64),
        "Improved euler method"
    )
    runge_kutta_method = ButcherSchema(
        np.array([0, 0.5, 0.5, 1], dtype=np.float64),
        np.array([[0, 0, 0, 0],
                  [0.5, 0, 0, 0],
                  [0, 0.5, 0, 0],
                  [0, 0, 1, 0]], dtype=np.float64),
        np.array([1 / 6, 2 / 6, 2 / 6, 1 / 6], dtype=np.float64),
        "Runge-Kutta method"
    )
    return euler_method, improved_euler_method, runge_kutta_method
```

Error calculating

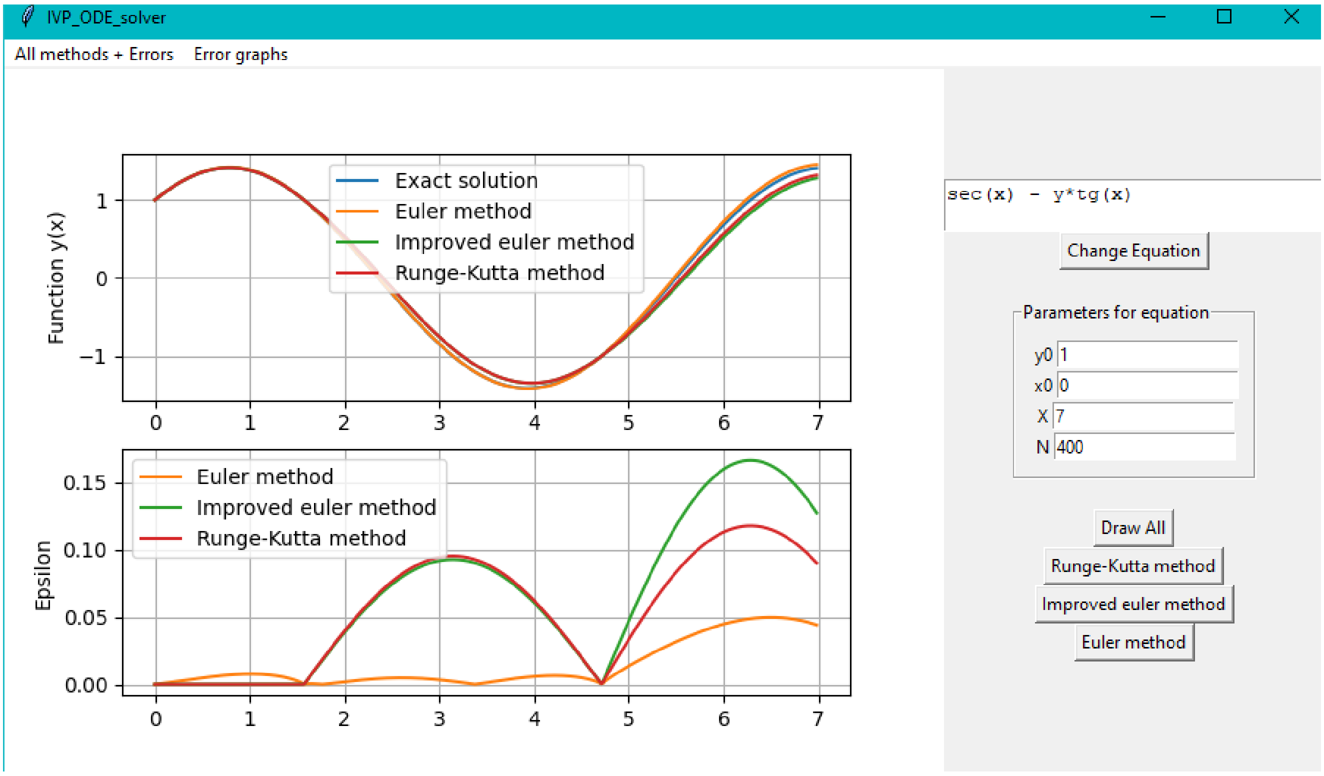
```python
def calc_error(self, right_y, y):
    return [abs(y_exact - y) for y_exact, y in zip(right_y, y)]
```

UML class diagram

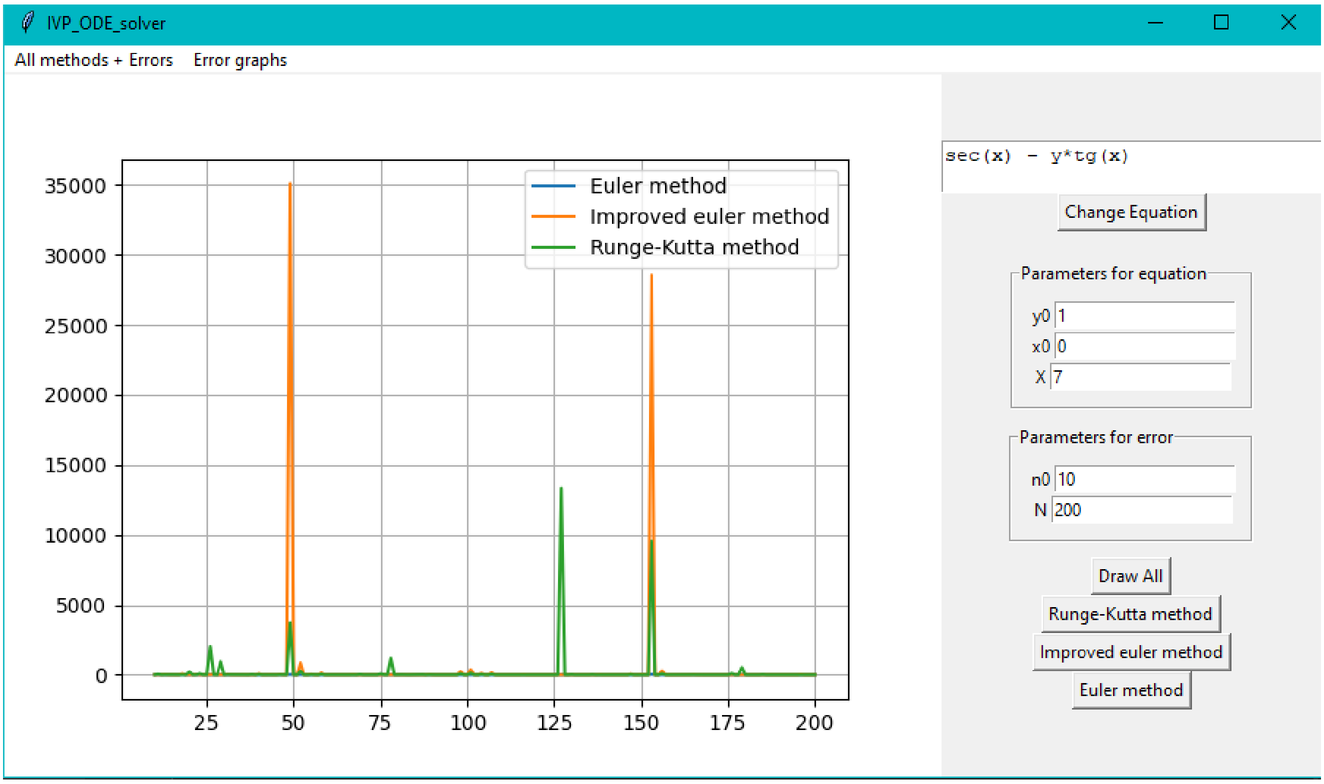


Screenshots of the program

Main window



Errors window


IVP_ODE_solver
All methods + Errors
Error graphs
Euler method
Improved euler method
Runge-Kutta method
35000
30000
25000
20000
15000
10000
5000
0
25
50
75
100
125
150
175
200
sec(x) - y*tg(x)
Change Equation
Parameters for equation
y0 1
x0 0
X 7
Parameters for error
n0 10
N 200
Draw All
Runge-Kutta method
Improved euler method
Euler method