Variant 10.

$x \in (x_0\ X)$, $y(x_0) = y_0$;

		y_0	x_0	X
10	$-y^2/3 - 2/(3x^2)$	2	1	5

This is a **Riccati eq.** Consider: $y_1 = 1/x$

Substitution: $y = y_1 + z(x) = 1/x + z$; So, $y' = -1/x^2 + z'$

$z = y - \frac{1}{x}$

$$\frac{-1/x^2 - 2z/x - z^2}{3} - \frac{2}{3x^2} = -\frac{1}{x^2} + z';$$

$$\frac{-1 - 2zx - x^2z^2 - 2 + 3}{3x^2} = z';$$

$$-2zx - x^2z^2 = 3z'x^2;$$

$$z^2 + 2z/x + 3z' = 0; \quad /: z^2$$

Now, this is a **Bernoulli equation** [n=2]

$$1 + \frac{2}{zx} + \frac{3z'}{z^2} = 0; \Rightarrow \frac{1}{3} + \frac{2}{3x}\cdot\frac{1}{z} = -\frac{z'}{z^2};$$

Substitution: $t = 1/z$; $t' = -z'/z^2$

$$1/3 + 2t/3x = t'; \Rightarrow t' - 2t/3x = 1/3;$$

Solve it as a **Linear FO DE**:

1) $t' - 2t/3x = 0; \Rightarrow 3x\,dt = 2t\,dx;$

$$\frac{dx}{3x} = 2\frac{dt}{t}; \Rightarrow \frac{1}{3}\int\frac{dx}{x} = 2\int\frac{dt}{t} \Rightarrow t = \sqrt[3]{x^2}\cdot C;$$

2) $C \to B(x)$; $\Rightarrow t = \sqrt[3]{x^2}\cdot B$; $t' = \frac{2B}{3\sqrt[3]{x}} + B'\cdot\sqrt[3]{x^2}$

$$\frac{2B}{3\sqrt[3]{x^2}} + B'\cdot\sqrt[3]{x^2} - \frac{2\cdot\sqrt[3]{x^2}B}{3x} = \frac{1}{3};$$

$$\frac{B}{x^{2/3}} + 3B'\cdot\sqrt[3]{x^2} - \frac{B}{x^{2/3}} = 1;$$

$$3B'\cdot\sqrt[3]{x^2} = 1; \Rightarrow B' = \frac{1}{3\sqrt[3]{x^2}};$$

$$B = \frac{1}{3}\int x^{-2/3} \Rightarrow B = x^{1/3} + C$$

Thus, $t = \sqrt[3]{x^2}\cdot B = x + C\cdot\sqrt[3]{x^2}$;

Substitute back t:

$$\frac{1}{z} = x + C\cdot\sqrt[3]{x^2}; \Rightarrow z = \frac{1}{(x + C_1\cdot\sqrt[3]{x^2})}$$

Substitute back z:

$$y - \frac{1}{x} = \frac{1}{(x + C_1\cdot\sqrt[3]{x^2})} \Rightarrow \boxed{y = \frac{1}{x} + \frac{1}{(x + C_1\cdot\sqrt[3]{x^2})}}$$

Discontinuity, if $x = 0$ or $x = -C_1^3$

Solution of the IVP:

$$y = \frac{1}{x} + \frac{1}{(x + C_1\cdot\sqrt[3]{x^2})}$$

$$2 = 1 + \frac{1}{1 + C_1};$$

$$1 = \frac{1}{1 + C_1};$$

$$1 + C_1 = 1;$$

$$\boxed{C_1 = 0}$$

General solution:

Suppose $y(a) = b$

$$b = \frac{1}{a} + \frac{1}{a + C_1\cdot a^{2/3}};$$

$$b(a + C_1\cdot a^{2/3}) = 1 + \frac{a + C_1\cdot a^{2/3}}{a};$$

$$b(a + C_1\cdot a^{2/3}) = 1 + 1 + \frac{C_1\cdot}{\sqrt[3]{a}};$$

$$C_1 = b(a^{4/3} + C_1 a) - 2a^{1/3}$$

$$C_1 = ba^{4/3} + C_1 ab - 2a^{1/3}$$

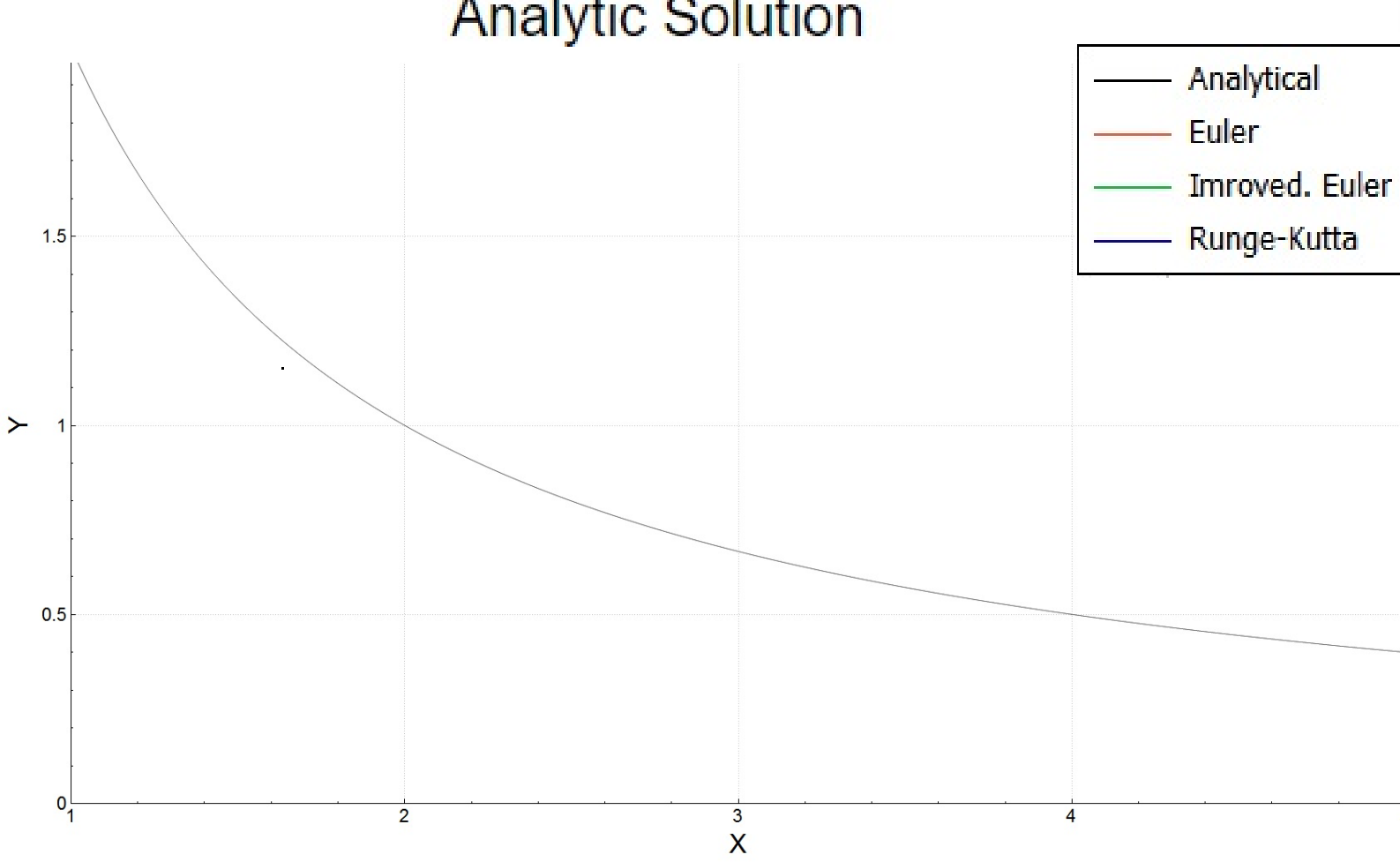
$$C_1(1 - ab) = ba^{4/3} - 2a^{1/3}$$

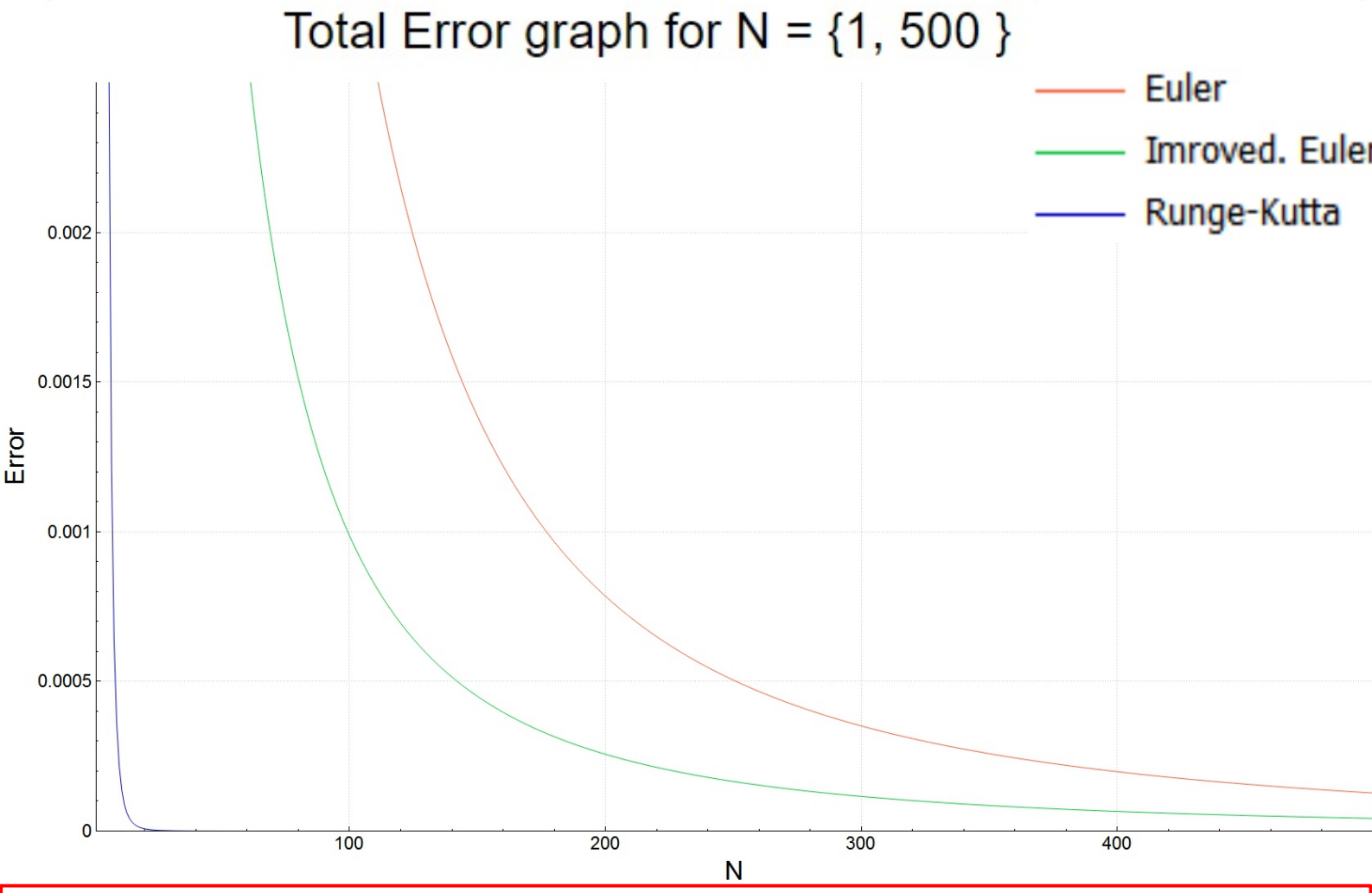
$$C_1 = \frac{b\cdot a^{4/3} - 2a^{1/3}}{1 - a\cdot b};$$

Answer:

$$y = \frac{2}{x}, \quad y(5) = 0,4$$

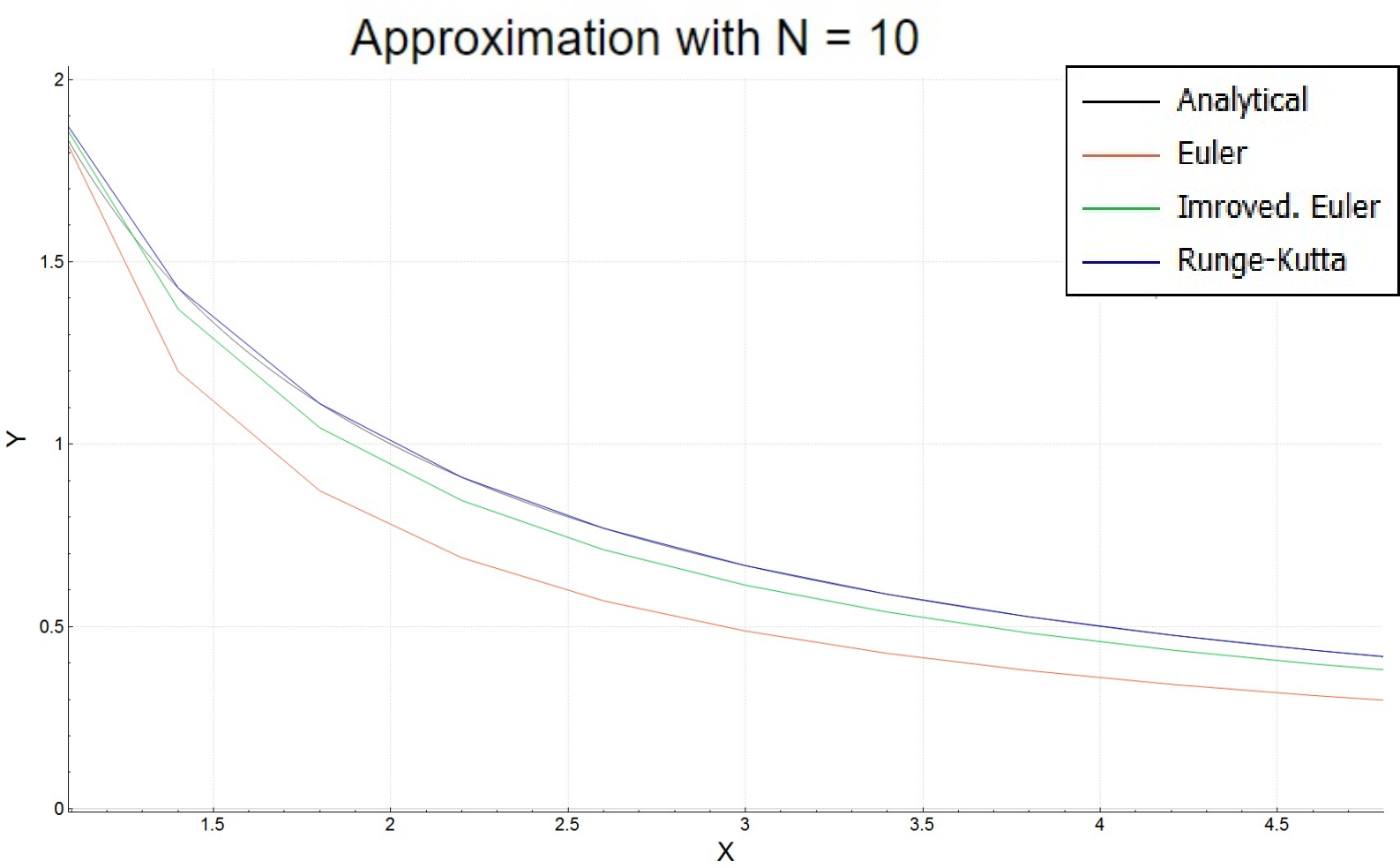
$$\boxed{y = \frac{1}{x} + \frac{1}{(x + C_1\cdot\sqrt[3]{x^2})}}$$

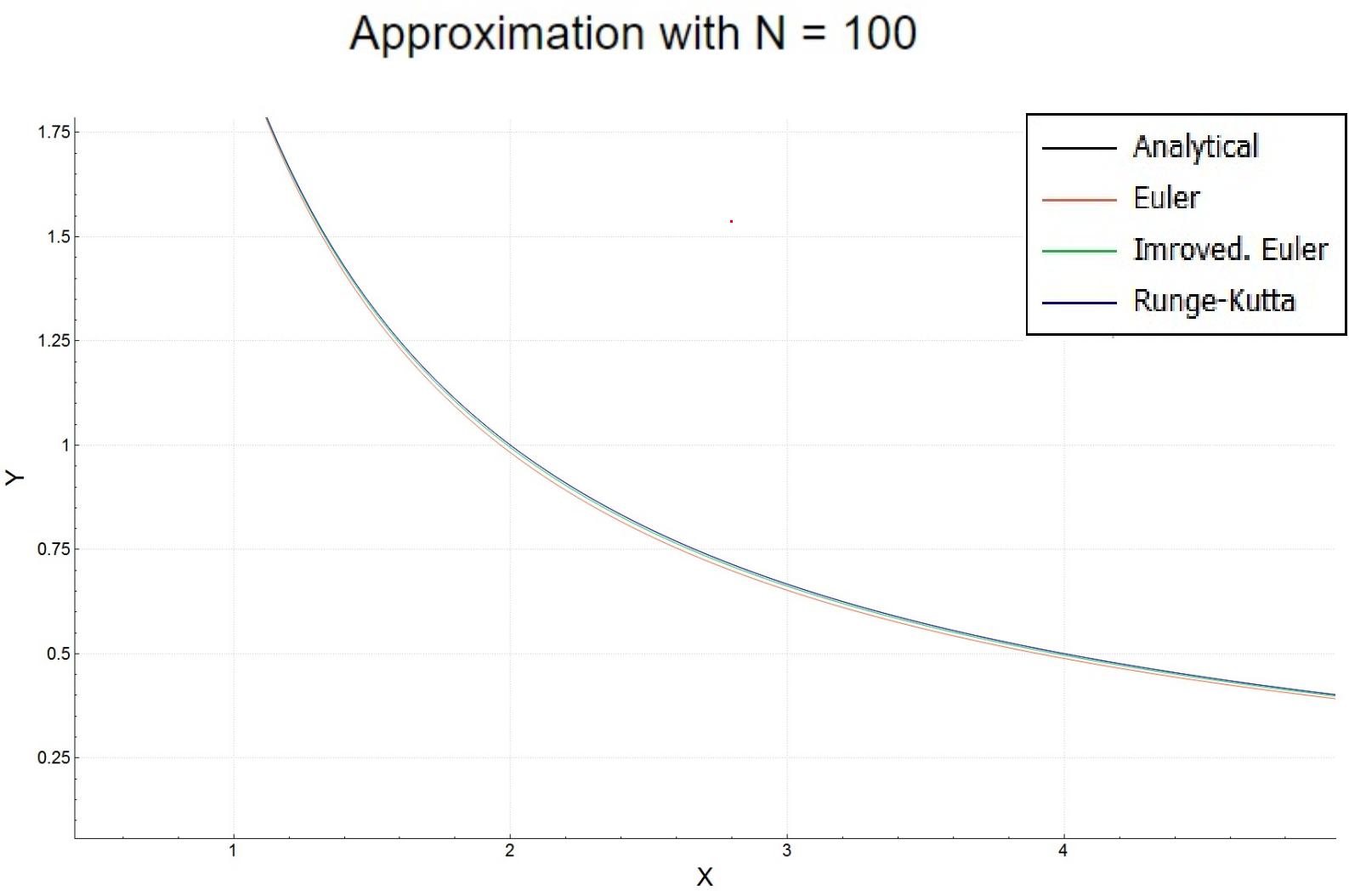




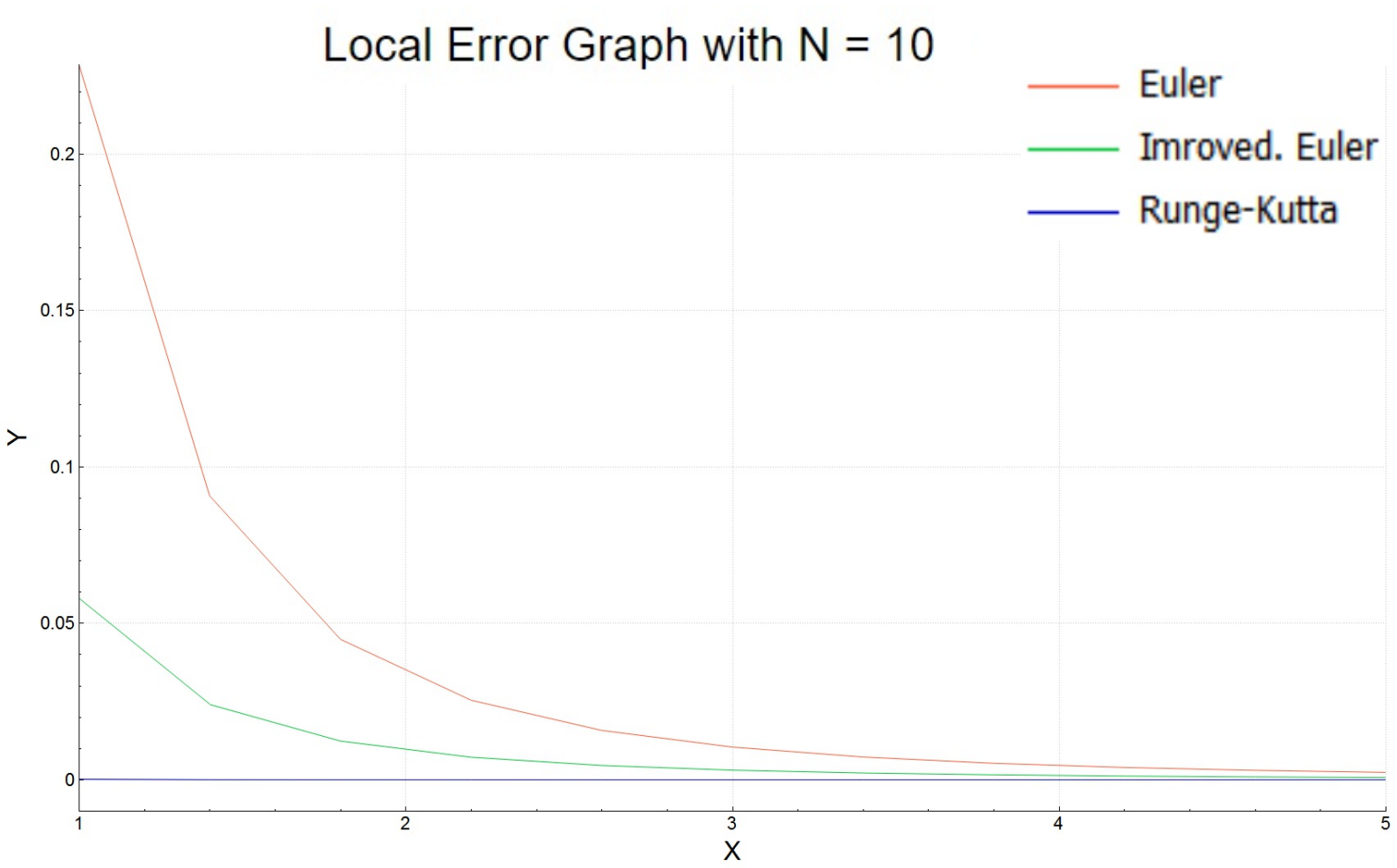
The error of the Runge-Kutta method decreases faster than the error of the Improved Euler method, which in turn decreases faster than the error of the regular Euler method.

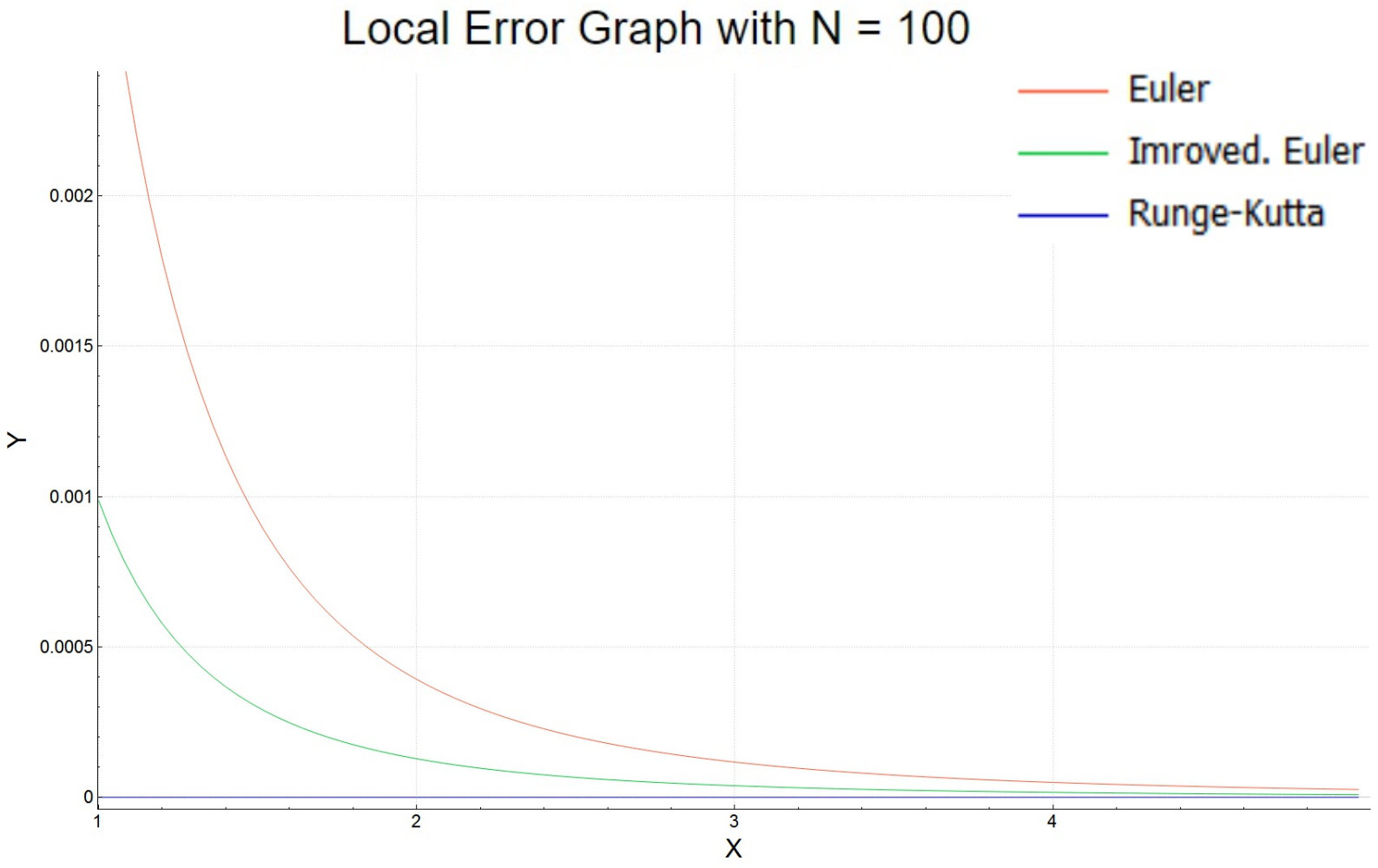
It simply shows the maximum error each method produces at some value of N





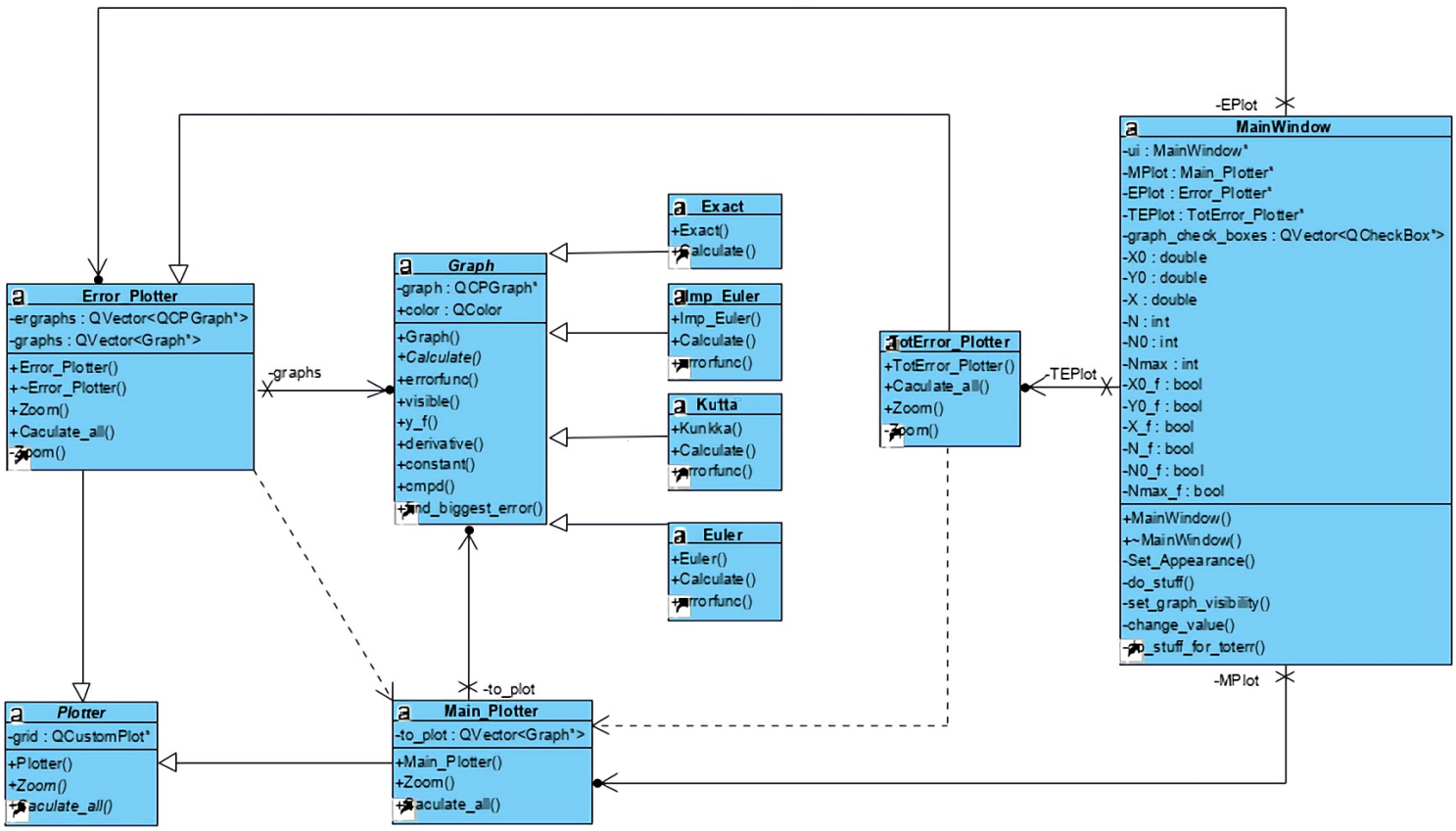
The more steps there are, the better the approximation is





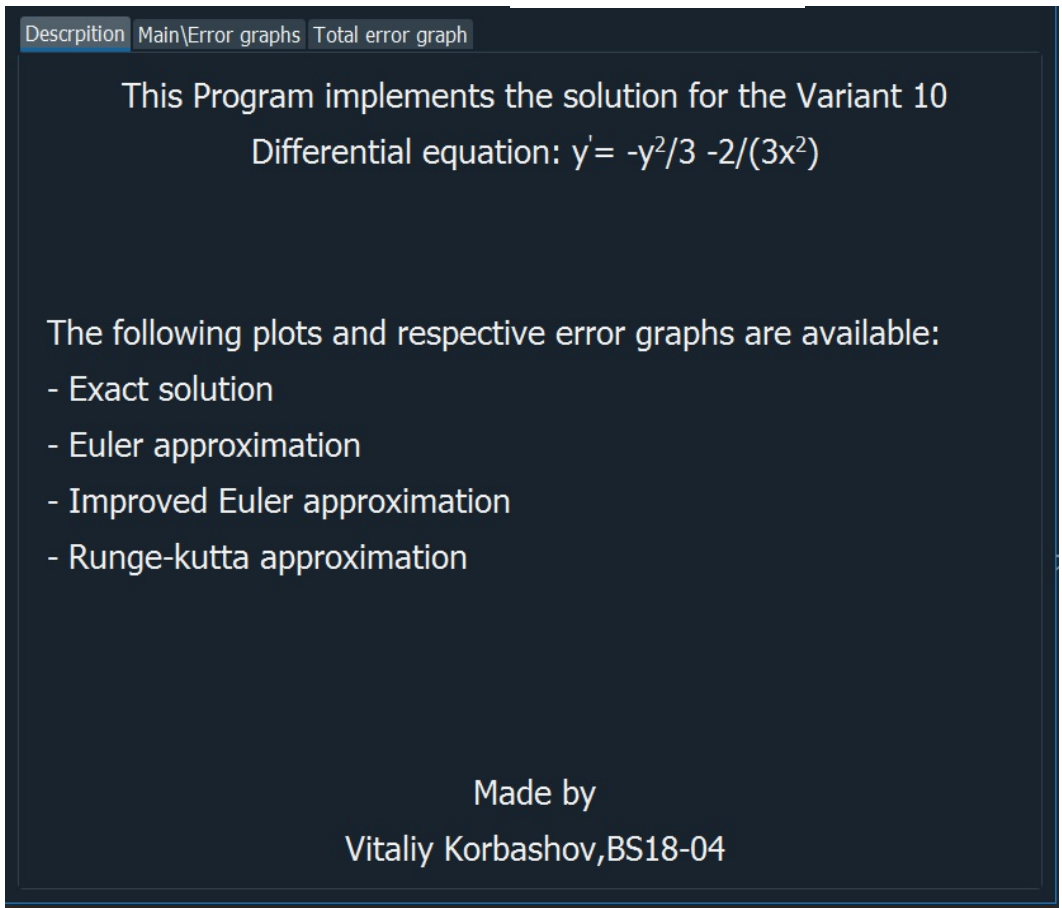
As you can see, all error graphs are $O(h^2)$

UML Diagram



GUI Description

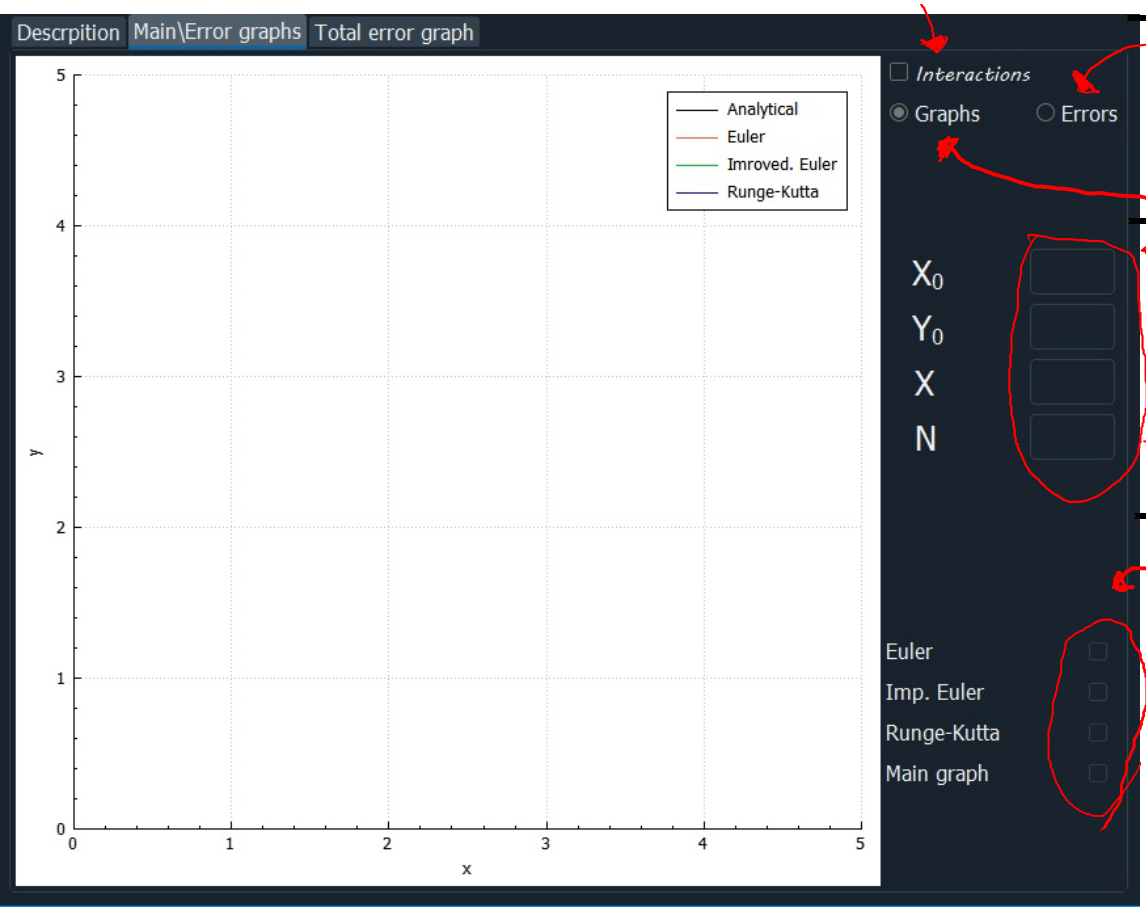
Tab I



In the first tab you
Can see the program
Description and the
Differential equation
It is supposed to
Solve and approximate

Tab II

Zoom/scroll can be enabled via mouse scroll/drag if needed



One can switch Between two Types of Graph

Graphs get Automatically Plotted on input

Toggle the Graphs you Want to see

If the user enters Invalid data, some of the field labels light up red to indicate the mistake; Thus, only the valid input is passed down to the functions.

Valid ranges:

$X = \{ -999.00 , 999.00 \}$

$Y_0 = \{ -999.00 , 999.00 \}$

$X_0 = \{ -999.00 , 999.00 \}$

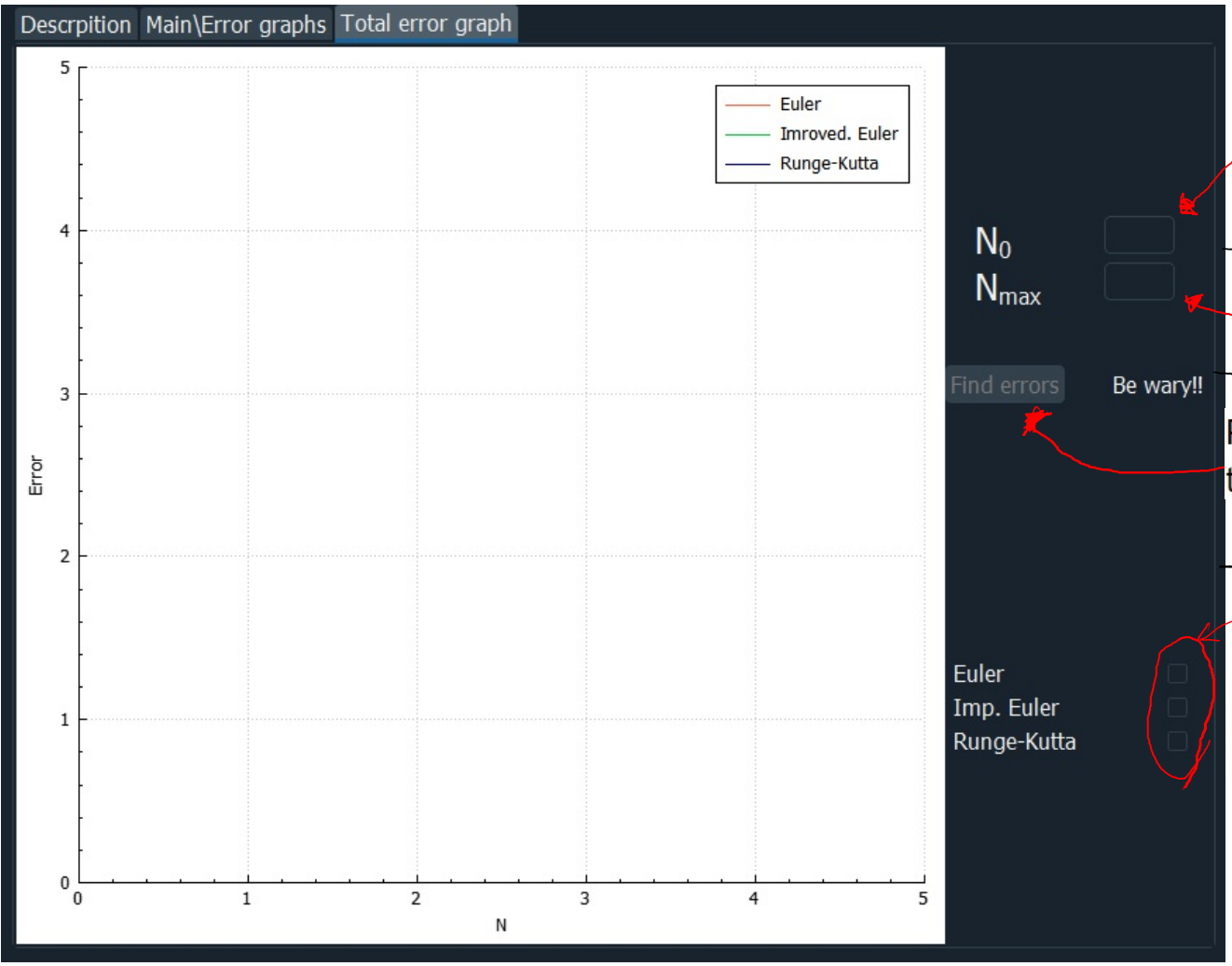
$N = \{ 1, 99999\}$

$X > X_0$

The exact, “Main graph”, is built on the bigger range than its approximation to allow for the ability to better investigate it; (All asymptotes are handled correctly)

Also, some constraints were put on the possible zoom ranges to avoid graph being too small;

Tab III



Here, one can view how the error of each approximation graph changes with the number of steps;
The “Find” button was added to remove lag, caused by auto calculations on input.
The field labels light up if the following rules are violated:

$$N = \{1, 9999\}$$

$$N_0 = \{1, 9999\}$$

$$N > N_0$$

To see all the code click the link github.com/BlackNuziGuy/Differ_eq/tree/master/Differ_eq

The program was written in C++ with the use of Qt creator.

It obeys SOLID programming principles:

1. **Single responsibility principle** (Different method implementations are only to store the ways for computation of the graph and its errors, while plotters store the graphs themselves and allow for plotting and etc.)
2. **Open/Closed principle** (The code can be easily expanded by just adding newly implemented Plotters and Graphs in the respective arrays and connecting them with the new UI elements.)
3. **Liskov substitution principle** (Both Graph objects and Plotter objects can be replaced by their subtypes)
4. **Interface segregation principle** (All the methods the client can call only depend on the constructor, which is called automatically)
5. **Dependency inversion principle** (Abstractions do not depend on details)

The base class Graph:

```
class Graph
{
public:

    QCPGraph *graph;
    QColor color;

    Graph(QCustomPlot *gr){ ...}
    virtual void Calculate(double, double, double , int) = 0;
    virtual double errorfunc(double , double , double )
    virtual void visible(bool x){ graph->setVisible(x);}

    //Necesasry calculation functions
    double y_f( double x, double c){return (1/x + 1/(x + c*pow(pow(x,2),1/3.0)) ); }
    double derivative(double x, double y){return ( -y*y/3 - 2.0/(3*x*x) );}
    double constant(double x0, double y0){ ...}
    double find_biggest_error(double x0, double y0, double X, int N) {...}
};
```

It has the implementation of some functions, such as derivative(), constant(), y_f() and find_biggest_error(), which are needed for computations; [all the same for inherited classes] The class also has the interface of necessary functions which all derived classes have to implement. [Calculate(), errorFunc()]

```
class Plotter{     ⚠ 'Plotter' has virtual functions but non-virtual desti

public:
    QCustomPlot* grid;
    QVector<Graph*> graphs;

    Plotter(QCustomPlot *element){ ...}


    virtual void Zoom(double, double, double) = 0;
    virtual void Caculate_all(double x0, double y0, double X, int N) = 0;
};
```

The class Plotter allows its derived classes to plot different types of graphs with the use of an array of the base class Graph via polymorphism and dynamic binding.

Implementations:

```
Euler(QCustomPlot *gr) :Graph(gr){
    color = QColor(255,87,51);
    graph->setPen(color);
    graph->setName("Euler");
}
double Euler::errorfunc(double xi, double xi_1, double c){
    return ( (y_f(xi_1,c) - y_f(xi,c)) - (xi_1 - xi)*derivative(xi, y_f(xi,c) ) );
}

void Euler::Calculate(double x0, double y0, double X, int N){

    QVector<double> x(N+1), y(N+1);
    double h = (X-x0)/N;
    double xe = x0,ye = y0;
    int i = 0;

    y[i] = y0;
    x[i] = x0;

    for (i = 1; i<=N; i++) {
        ye  += h * derivative(xe, ye);
        xe  += h;

        y[i] = ye;
        x[i] = xe;
    }

    graph->setData(x, y);
}
```

To see all the code click the link github.com/BlackNuziGuy/Differ_eq/tree/master/Differ_eq

```
Exact(QCustomPlot *gr):Graph(gr){
    color = QColor(0,0,0);
    graph->setPen(color);
    graph->setName("Analytical");
}
void Exact::Calculate(double x0, double y0, double X, int N){

   int i1 = 0,i2 = 0; //To account for Gaps ( there are 2 asymptotes)
   double c = constant(x0,y0); //Const and value to start stepping from

   //Plot a bit more
   double val =  std::max(abs(x0),abs(X)); val *= 1.5;
   double x_v = -val; X = val;
   N = 499999;//For the best precision
   double h = (X-x_v)/N;
   QVector<double> x(N+1), y(N+1);//N steps + 1 final point

   for(int i = 0; i<=N; i++){

       x[i] = x_v;
       y[i] = y_f(x[i],c);

       //Find assymtote
       if( abs(x[i] - 0) < abs(x[i1] - 0) )
           i1 = i;
       if( abs(x[i] + c*c*c) < abs(x[i2] + c*c*c) )
           i2 = i;

       x_v+=h;
   }


   //DESTROY the assymptote
   if ( (-val)<0 && 0<val)
       y[i1] = std::numeric_limits<double>::quiet_NaN();
   if ( (-val)<(-c*c*c) && (-c*c*c)<val  )
       y[i2] = std::numeric_limits<double>::quiet_NaN();

   graph->setData(x,y);
```

To see all the code click the link github.com/BlackNuziGuy/Differ_eq/tree/master/Differ_eq

```
Imp_Euler(QCustomPlot *gr):Graph(gr){
    color = QColor(0,200,51);
    graph->setPen(color);
    graph->setName("Imroved. Euler");
}
double Imp_Euler::errorfunc(double xi, double xi_1, double c){

    double k1,k2, h = xi_1 - xi, y = y_f(xi,c) ,y1, ya;

    k1= h * derivative(xi, y);    //calculate slope or dy/dx at x0,y0
    y1= y + k1;                    //calculate new y, which is y0+h*dy/dx
    k2= h * derivative(xi, y1); //calculate slope or dy/dx at x0,new y
    ya=(k1+k2)/2.0;        //calculate the average of the slopes at y0 and new y

    return ( (y_f(xi_1,c) - y) - ya);
}
void Imp_Euler::Calculate(double x0, double y0, double X, int N){

    QVector<double> x(N+1), y(N+1);
    double h = (X-x0)/N;
    double ya, k1,k2, y1,y2;

    for (int i = 0; i<=N; i++) {
        {
            k1= h * derivative(x0, y0);   //calculate slope or dy/dx at x0,y0
            y1=y0+k1;                    //calculate new y, which is y0+h*dy/dx
            k2= h * derivative(x0, y1); //calculate slope or dy/dx at x0,new y
            ya=(k1+k2)/2.0;            //calculate the average of the slopes at y0 and new y
            y2=y0+ya;                 //calculate new y, which is y0+h*average(dy/dx)

            y[i] = y0;
            x[i] = x0;

            x0=x0+h;                //calculate new x.
            y0=y2;                 //pass this new y as y0 in the next iteration.
        }
    }
    graph->setData(x, y);
}
```

To see all the code click the link github.com/BlackNuziGuy/Differ_eq/tree/master/Differ_eq

```
Kunkka(QCustomPlot *gr):Graph(gr){
    color = QColor(0,0,200);
    graph->setPen(color);
    graph->setName("Runge-Kutta");
}
double Kunkka::errorfunc(double xi, double xi_1, double c){
    double h = xi_1 - xi, y = y_f(xi,c),y1, y2, y3, y4;

    y1 = h * derivative(xi , y);
    y2 = h * derivative(xi + h/2.0, y + y1/2.0);
    y3 = h * derivative(xi + h/2.0, y + y2/2.0);
    y4 = h * derivative(xi + h    , y + y3    );

    y4 = ( y1 + 2*y2+ 2*y3 + y4)/6.0;

    return ( (y_f(xi_1,c) - y) - y4 );
}


void Kunkka::Calculate(double x0, double y0, double X, int N){

    QVector<double> x(N+1), y(N+1); // N steps + N0
    const double h = (X-x0)/N;
    double y1, y2, y3, y4;

    x[0] = x0; y[0] = y0;

    for(int i=1; i<=N; i++){

        y1 = h * derivative(x[i-1] , y[i-1]);
        y2 = h * derivative(x[i-1] + h/2.0, y[i-1] + y1/2.0);
        y3 = h * derivative(x[i-1] + h/2.0, y[i-1] + y2/2.0);
        y4 = h * derivative(x[i-1] + h    , y[i-1] + y3    );

        x[i]=x[i-1] + h;
        y[i] = y[i-1] + ( y1 + 2*y2+ 2*y3 + y4)/6.0;
    }

    graph->setData(x, y);
}
```

To see all the code click the link github.com/BlackNuziGuy/Differ_eq/tree/master/Differ_eq